

$$\boldsymbol{E} = -\nabla \phi$$

有限元基础与程序设计

Finite Element Method
Fundamentals and Programming

张军军 夏广庆 何晓明 鹿畅 编著

$$\oint_S \boldsymbol{F} \cdot n \mathrm{d}S = \iiint_V \nabla \cdot \boldsymbol{F} \mathrm{d}V$$

内容简介

本书介绍了有限元方法基本理论、不同问题模型有限元方法求解策略、不同边界条件在有限元方法中的施加方法及有限元方法误差计算，全书分为 8 章，主要包括：有限元方法基础准备；1D 有限元方法、不同边界条件处理、误差计算及算例；2D 椭圆方程有限元方法、边界条件处理、误差计算及算例；2D 线弹性方程有限元方法、边界条件处理、误差计算及算例；2D 稳态流体力学有限元方法（主要为 Stokes 方程与 Navier-Stokes 方程）、边界条件处理、误差计算及算例；2D 抛物与双曲型方程有限元方法、边界条件处理、误差计算及算例；2D 非稳态流体力学有限元方法（主要为 Stokes 方程与 Navier-Stokes 方程）、边界条件处理、误差计算及算例；3D 有限元方法简介。

本书可作为高等学校理工科相关专业本科生和研究生教材，也可供从事有限元方法、流体力学及科学与工程计算的科技人员参考。

前 言

本书是为高等院校固体、流体力学及有限元数值计算相关专业编写的教学用书。编者希望读者在学完本书后能掌握微分方程基本知识、有限元方法基本理论与程序设计，为今后有限元数值计算、软件开发打下必要基础。

全书分为 8 章，即有限元方法基础准备；1D 有限元方法、不同边界条件处理、误差计算及算例；2D 椭圆方程有限元方法、边界条件处理、误差计算及算例；2D 线弹性方程有限元方法、边界条件处理、误差计算及算例；2D 稳态流体力学有限元方法（主要为 Stokes 方程与 Navier-Stokes 方程）、边界条件处理、误差计算及算例；2D 抛物与双曲型方程有限元方法、边界条件处理、误差计算及算例；2D 非稳态流体力学有限元方法（主要为 Stokes 方程与 Navier-Stokes 方程）、边界条件处理、误差计算及算例；3D 有限元方法简介。每章内容都包含基础理论、算例模型、程序实现三个部分。

全书由大连理工大学工业装备结构分析优化与 CAE 软件全国重点实验室微纳卫星与先进推进技术研究团队成员基于何晓明老师开源课件编写。

由于编者水平有限，难免有疏漏之处，对本书中的错误和不足之处，欢迎读者批评指正。本书后续也会持续改进、更新，不断迭代。

编者
2024 年 9 月

目 录

第一章 有限元方法基础准备	1
1.1 引言	1
1.2 偏微分方程简介	1
1.3 散度定理	2
1.3.1 散度	2
1.3.2 散度定理	3
1.4 有限元中基本插值方法	4
1.5 有限元方法中基函数与有限元空间	5
1.5.1 1D 基函数与有限元空间	6
1.5.2 2D 基函数与有限元空间	14
1.6 高斯积分	26
1.6.1 1D 高斯积分	27
1.6.2 2D 高斯积分	31
1.7 有限元方法的一般流程	34
第二章 1D 有限元方法	35
2.1 引言	35
2.2 问题模型	35
2.2.1 弱格式	35
2.2.2 方程离散	37
2.2.3 线性系统构建	38
2.3 不同边界条件处理	40
2.3.1 第 1 类边界条件	40
2.3.2 第 2 类边界条件	41
2.3.3 第 3 类边界条件	42

2.4	误差计算	44
2.4.1	最大节点误差	44
2.4.2	无穷范数误差 L^∞	44
2.4.3	L^2 范数误差	45
2.4.4	H^1 范数误差	45
2.5	应用实例及程序实现	47
2.5.1	Example 1	47
2.5.2	Example 2	61
2.5.3	Example 3	64
	习题	67
第三章 2D 椭圆方程有限元方法		69
3.1	引言	69
3.2	问题模型	69
3.2.1	弱格式	69
3.2.2	方程离散	70
3.2.3	线性系统构建	71
3.3	不同边界条件处理	73
3.3.1	第 1 类边界条件	73
3.3.2	第 2 类边界条件	74
3.3.3	第 3 类边界条件	75
3.3.4	混合边界条件	76
3.4	各项异性问题	76
3.5	通用 2 阶问题	77
3.6	误差计算	78
3.6.1	最大节点误差	78
3.6.2	无穷范数误差 L^∞	78
3.6.3	L^2 范数误差	79
3.6.4	H^1 范数误差	80
3.7	应用实例及程序实现	82
3.7.1	Example 1	82
3.7.2	Example 2	95
3.7.3	Example 3	99
	习题	101

第四章 2D 线弹性方程有限元方法	103
4.1 引言	103
4.2 线弹性理论一般原理与基本方程	103
4.2.1 线弹性问题基本概念	103
4.2.2 线弹性问题一般形式	105
4.3 问题模型	109
4.3.1 弱格式	110
4.3.2 方程离散	112
4.3.3 线性系统构建	112
4.4 不同边界条件处理	115
4.4.1 第 1 类边界条件	115
4.4.2 第 2 类边界条件	115
4.4.3 第 3 类边界条件	116
4.4.4 混合边界条件	117
4.5 误差计算	118
4.5.1 最大节点误差	118
4.5.2 无穷范数误差 L^∞	118
4.5.3 L^2 范数误差	119
4.5.4 H^1 范数误差	119
4.6 应用实例及程序实现	119
4.6.1 Example 1	119
4.6.2 Example 2	127
习题	130
第五章 2D 稳态流体力学有限元方法	131
5.1 引言	131
5.2 稳态 Stokes 方程	131
5.2.1 问题模型	131
5.2.2 弱格式	132
5.2.3 方程离散	134
5.2.4 线性系统构建	135
5.2.5 不同边界条件处理	138
5.2.6 误差计算	142
5.3 稳态 Navier-Stokes 方程	143

5.3.1	问题模型	143
5.3.2	弱格式	144
5.3.3	方程离散	146
5.3.4	非线性项处理	148
5.3.5	构建线性系统	149
5.3.6	不同边界条件处理	154
5.3.7	误差计算	159
5.4	应用实例及程序实现	160
5.4.1	Example 1	160
5.4.2	Example 2	164
	习题	168
第六章	2D 抛物及双曲方程有限元方法	169
6.1	引言	169
6.2	2D2 阶抛物方程	169
6.2.1	问题模型	169
6.2.2	弱格式	169
6.2.3	方程离散	170
6.2.4	非稳态项处理	171
6.2.5	线性系统构建	174
6.2.6	不同边界条件处理	177
6.2.7	各项异性抛物型方程	177
6.2.8	误差计算	178
6.3	2D2 阶双曲方程	178
6.3.1	问题模型	178
6.3.2	弱格式	178
6.3.3	方程离散	179
6.3.4	构建线性系统	180
6.3.5	不同边界条件处理	180
6.3.6	各项异性双曲方程	181
6.3.7	误差计算	181
6.4	应用实例及程序实现	181
6.4.1	Example 1	181
	习题	187

第七章 2D 非稳态流体与固体问题有限元方法	189
7.1 引言	189
7.2 非稳态 Stokes 方程	189
7.2.1 问题模型	189
7.2.2 弱格式	190
7.2.3 方程离散	192
7.2.4 线性系统构建	193
7.2.5 不同边界条件处理	198
7.2.6 误差计算	198
7.3 非稳态 Navier-Stokes 方程	199
7.3.1 问题模型	199
7.3.2 弱格式	200
7.3.3 方程离散	201
7.3.4 非稳态项处理	203
7.3.5 非线性项处理	204
7.3.6 构建线性系统	205
7.3.7 不同边界条件处理	211
7.3.8 误差计算	212
7.4 非稳态线弹性方程	212
7.4.1 弱格式	213
7.4.2 方程离散	215
7.4.3 线性系统构建	215
7.4.4 不同边界条件处理	218
7.4.5 误差计算	219
7.5 应用实例及程序实现	219
7.5.1 Example 1	219
7.5.2 Example 2	222
习题	223
第八章 3D 有限元方法简介	225
8.1 引言	225
8.2 问题模型	225
8.2.1 弱格式	225
8.2.2 方程离散	226

8.2.3	线性系统构建	227
8.3	3D 基函数与有限元空间	229
8.3.1	六面体单元	230
8.3.2	四面体单元	231
8.4	应用实例	232
8.4.1	Example 1	232
	参考文献	237
	附录 A 2D 常用高斯积分点及权重	241
	附录 B 本书所述有限元方法参考程序 (Python)	243
B.1	第二章参考程序	243
B.1.1	1D_FEM.py	243
B.1.2	Mesh.py	245
B.1.3	linearSystem.py	247
B.1.4	functions.py	254
B.1.5	postProcess.py	255
B.2	第三章参考程序	256
B.2.1	2D_FEM.py	256
B.2.2	Mesh.py	258
B.2.3	linearSystem.py	263
B.2.4	functions.py	278
B.2.5	postProcess.py	279
B.3	第四章参考程序	280
B.3.1	2D_FEM.py	281
B.3.2	Mesh.py	284
B.3.3	linearSystem.py	290
B.3.4	functions.py	304
B.3.5	postProcess.py	305
B.4	第五章参考程序	307
B.4.1	2D_FEM.py	307
B.4.2	Mesh.py	310
B.4.3	linearSystem.py	317

B.4.4	functions.py	333
B.4.5	postProcess.py	334
B.5	第六章参考程序	336
B.5.1	2D_FEM.py	336
B.5.2	Mesh.py	337
B.5.3	linearSystem.py	342
B.5.4	functions.py	359
B.5.5	postProcess.py	359

1

有限元方法基础准备

1.1 引言

有限元方法（Finite Element Method, FEM）是一种数值求解复杂问题模型（通常为偏微分方程）的仿真方法，在各个学科领域均有非常广泛的应用。该方法主要通过将连续空间问题进行分割离散，将大型问题求解转换为小问题求解，最后再将小问题的解组合得到整体空间问题的解。从而达到将无限问题化简为有限问题的近似替代，从而能够采用有限的人力或者计算机资源将复杂问题进行求解，随着计算机技术的发展该方法在空间离散足够细致的情况下得到无限接近问题的真实解，进而解决大部分工程问题。

本章主要对有限元方法中重要基础理论进行介绍，主要有偏微分方程基本类型；有限元空间离散的基础：散度定理（Divergence Theorem），类似分部积分法；有限元空间构造的基础：基本插值函数，也称为形函数（Basis/Shape Function）；有限元空间的积分方法；有限元方法求解问题的一般流程。

1.2 偏微分方程简介

偏微分方程（Partial Differential Equation, PDE）是用于描述某一函数随时间、空间变化的一类方程，通常用于描述物理、工程、生物等领域中的问题。对于 2D 情况，偏微分方程的一般形式如下（暂不考虑时间项）：

$$a \frac{\partial^2 \phi}{\partial x^2} + b \frac{\partial^2 \phi}{\partial x \partial y} + c \frac{\partial^2 \phi}{\partial y^2} + d \frac{\partial \phi}{\partial x} + e \frac{\partial \phi}{\partial y} + f \phi + g = 0 \quad (1.1)$$

对于具有如上形式的方程，根据特征方程的根，如判别式 $b^2 - 4ac$ ，可以将方程分为以下几类：

- ▶ 椭圆型方程 (Elliptic PDE)： $b^2 - 4ac < 0$ 。该方程用以描述某种现象朝所有方向发展，强度逐渐衰减，结果是平滑的，例如，稳定的热传导方程和拉普拉斯方程，亚音速下的流体运动。
- ▶ 双曲型方程 (Hyperbolic PDE)： $b^2 - 4ac > 0$ 。该方程用以描述某种现象朝特定方向发展，强度倾向于保持不变，结果通常不是平滑的，具有非连续性，比如超音速问题。
- ▶ 抛物型方程 (Parabolic PDE)： $b^2 - 4ac = 0$ 。该方程为一种受限的双曲型，是强度会具有耗散性的双曲型，例如波动方程 (Wave Equation)。

1.3 散度定理

1.3.1 散度

在介绍散度定理之前，首先需要了解微分算符 ∇ ，其在 3D 空间中有如下形式^[1-2]：

$$\nabla = \frac{\partial}{\partial x} \vec{i} + \frac{\partial}{\partial y} \vec{j} + \frac{\partial}{\partial z} \vec{k} \quad (1.2)$$

明确散度这一算子作用于矢量，其结果为一标量，可以表示如下式：

$$\operatorname{div}(\vec{v}) = \nabla \cdot \vec{v} \quad (1.3)$$

其在 3D 空间中具体表达式如下：

$$\nabla \cdot \vec{v} = \left(\frac{\partial}{\partial x} \vec{i} + \frac{\partial}{\partial y} \vec{j} + \frac{\partial}{\partial z} \vec{k} \right) \cdot (u \vec{i} + v \vec{j} + w \vec{k}) = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \quad (1.4)$$

性质 → 散度的物理意义

描述一个矢量场是否发散，如果发散，发散强度是多少。发散也分为正发散或者负发散，正发散就是普通的发散，而负发散指的是聚集。所以如果散度的值为负，说明这个矢量场在向某一点聚集（这个点可以看做是一个洞/汇），反之，如果如果散度的值为正，则说明这个矢量场在某一点向外发散（这个点可以看做是一个源）。如果一个矢量场的散度为零，则说明这个矢量场既不聚集也不发散，即矢量场中的每个矢量都是相互平行的。所以，散度不为零的场叫做有散场或者有源场，散度为零的场叫做无散场或者无源场。



1.3.2 散度定理

散度定理又称为高斯散度定理、高斯公式^[3]，是指在矢量分析中，一个把矢量场通过曲面的流动（即通量）与曲面内部的矢量场的表现联系起来的定理。散度定理适用于1D、2D、3D，在处理实际问题时根据需要选择合适的形式即可。

在物理和工程中，散度定理通常运用在3D空间中。在1D，它等价于微积分基本定理；在2D，它等价于格林公式。其具体定义与方程描述如下：

定理 → 散度定理

Let \mathbf{F} be a vector field whose components have continuous first partial derivatives, and let S be a piecewise smooth oriented closed surface. The flux of \mathbf{F} across S in the direction of the surface's outward unit normal field \mathbf{n} equals the triple integral of the divergence $\nabla \cdot \mathbf{F}$ over the region V enclosed by the surface:

$$\iint_S \mathbf{F} \cdot \mathbf{n} dS = \iiint_V \nabla \cdot \mathbf{F} dV \quad (1.5)$$

在3D空间中，设矢量场 $\vec{F} = v\nabla u = v\frac{\partial u}{\partial x}\vec{i} + v\frac{\partial u}{\partial y}\vec{j} + v\frac{\partial u}{\partial z}\vec{k}$ ，可以得到：

$$\begin{aligned} \nabla \cdot \vec{F} &= \left(\frac{\partial}{\partial x} \vec{i} + \frac{\partial}{\partial y} \vec{j} + \frac{\partial}{\partial z} \vec{k} \right) \cdot \left(v \frac{\partial u}{\partial x} \vec{i} + v \frac{\partial u}{\partial y} \vec{j} + v \frac{\partial u}{\partial z} \vec{k} \right) \\ &= \frac{\partial}{\partial x} \left(v \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(v \frac{\partial u}{\partial y} \right) + \frac{\partial}{\partial z} \left(v \frac{\partial u}{\partial z} \right) \\ &= \frac{\partial v}{\partial x} \frac{\partial u}{\partial x} + v \frac{\partial^2 u}{\partial x^2} + \frac{\partial v}{\partial y} \frac{\partial u}{\partial y} + v \frac{\partial^2 u}{\partial y^2} + \frac{\partial v}{\partial z} \frac{\partial u}{\partial z} + v \frac{\partial^2 u}{\partial z^2} \\ &= \nabla u \cdot \nabla v + v \Delta u \end{aligned} \quad (1.6)$$

其中， $\Delta u = \nabla \cdot \nabla u = u_{xx} + u_{yy} + u_{zz} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}$ ，由此结合散度定理可以得到：

$$\begin{aligned} \iiint_{\Omega} \nabla \cdot \vec{F} dV &= \iiint_{\Omega} (\nabla u \cdot \nabla v + v \Delta u) dV \\ &= \iint_{\partial\Omega} \vec{F} \cdot \vec{n} ds = \iint_{\partial\Omega} v \nabla u \cdot \vec{n} ds = \iint_{\partial\Omega} v \frac{\partial u}{\partial n} ds \end{aligned} \quad (1.7)$$

其中， $\vec{n} = (n_x, n_y, n_z)$ ，并且 $n_x^2 + n_y^2 + n_z^2 = 1$ 为单位法向量，由此可以得到： $\frac{\partial u}{\partial n} = \nabla u \cdot \vec{n} = n_x \frac{\partial u}{\partial x} + n_y \frac{\partial u}{\partial y} + n_z \frac{\partial u}{\partial z}$ 为方程 u 的法向导数。该结果为多元分部积分基础。

1.4 有限元中基本插值方法

有限元方法就是利用基本插值 (Piecewise Interpolation) 方法得到空间各个节点处的值，常用的插值有 Lagrange、Newton、Hermite 插值^[4-6]。在有限元方法中常用的是 Lagrange 插值方法，接下来对该插值方法的相关概念及应用进行介绍。

定义 → Lagrange 插值公式

设 x_0, x_1, \dots, x_n 是 $[a, b]$ 上的 $n+1$ 个互异点，令

$$\begin{aligned} l_k(x) &= \frac{(x - x_0) \cdots (x - x_{k-1}) (x - x_{k+1}) \cdots (x - x_n)}{(x_k - x_0) \cdots (x_k - x_{k-1}) (x_k - x_{k+1}) \cdots (x_k - x_n)} \\ &= \frac{\omega_{n+1}(x)}{(x - x_k) \omega'_{n+1}(x_k)}, \quad k = 0, 1, \dots, n. \end{aligned} \quad (1.8)$$

其中 $\omega_{n+1}(x) = (x - x_0) \cdots (x - x_{k-1}) (x - x_{k+1}) \cdots (x - x_n)$ ，显然有

$$l_k(x_i) = \begin{cases} 1, & i = k \\ 0, & i \neq k, \end{cases} \quad i, k = 0, 1, \dots, n. \quad (1.9)$$

$l_k(x) (k = 0, 1, \dots, n)$ 称为 **n 次 Lagrange 插值基函数**。容易验证：

$$p_n(x) = \sum_{k=0}^n y_k l_k(x) \quad (1.10)$$

是 P_n 中满足 $p_n(x_i) = y_i (i = 0, 1, \dots, n)$ 的唯一的多项式， $p_n(x)$ 称为 **n 次 Lagrange 插值多项式**。

可以看到对于节点 k 而言，其对应的插值基函数的函数值满足当节点自变量 x_i 满足 $x_i = x_k$ 时其值为 1，否则为 0。有限元方法就是利用基函数这一特性排除了复杂线性系统中其他节点对待求未知节点函数值的影响，从而得到所求节点的函数值。事实上有限元方法中在计算空间某一节点函数值就是综合考虑包含该节点在内其他所有节点函数值在该处的影响，将其影响进行线性拟合得到最终结果。下面以一实例说明 Lagrange 插值公式的应用：

例 1.1 已知函数 $f(x)$ 的如下函数值：

x_i	1	2	3
$y_i = f(x_i)$	-1	-1	1

求 $f(x)$ 的二次 Lagrange 插值多项式 $p_2(x)$, 并利用 $p_2(x)$ 计算 $f(1.5)$ 的近似值。

解:

$$l_0(x) = \frac{(x-2)(x-3)}{(1-2)(1-3)} = \frac{1}{2}(x-1)(x-3)$$

$$l_1(x) = \frac{(x-1)(x-3)}{(2-1)(2-3)} = -(x-1)(x-3)$$

$$l_2(x) = \frac{(x-1)(x-2)}{(3-1)(3-2)} = \frac{1}{2}(x-1)(x-2)$$

$$\begin{aligned} p_2(x) &= \sum_{k=0}^2 y_k l_k(x) = -\frac{1}{2}(x-1)(x-3) + (x-1)(x-3) + \frac{1}{2}(x-1)(x-2) \\ &= x^2 - 3x + 1 \end{aligned}$$

所以有:

$$f(1.5) \approx p_2(1.5) = -1.25$$

在有限元方法中 $p_n(x)$ 即为有限空间内待求解函数在空间的集合, 通常写作 $u(x)$, 意为未知 (unkonwn) 函数; y_k 为有限空间内节点处 x_i 处的函数值, 通常写作 $u_j(x_i)$, $l_k(x_i)$ 为节点 x_i 处的 Lagrange 插值基函数, 通常写作 $\phi(x_i)$, 所以, 在有限元方法中, 连续空间内方程的解通常用离散后空间各节点解的线性组合近似表示为:

$$u(x) = \sum_{j=1}^{NN} u_j \phi_j(x) \quad (1.11)$$

其中, $u(x)$ 为计算域空间的解, x_i 为离散后计算域内编号为 i 节点坐标, NN 为离散空间网格节点总数, $u_j(x_i)$ 为离散后函数在节点 x_i 处的值, ϕ_j 为基函数 (形函数), 该函数具有如式 (1.9) 所述的性质, 很显然该表达式与 $n+1$ 点的 n 次 Lagrange 插值多项式一致。

$u(x)$ 为期望得到的连续空间内的解, 但在实际计算时通过求解不同节点 x_i 对应的 $u_j(x_i)$, 并用所有 $u_j(x_i)$ 的线性组合近似替代 $u(x)$ 。为此需要构造合适的基函数用来求解节点处的函数值, 并且注意到基函数个数与所求未知节点数量一致, 且精度也将影响所得空间节点值的精度。

1.5 有限元方法中基函数与有限元空间

如前所述, 基函数个数与未知节点函数值数量相同, 未知函数节点值的组合为待求函数在空间的解, 类似地, 各节点基函数的张集 (span) 形成的空间即为有限元空间, 该

空间将连续空间进行离散，使得无限元的问题转换为有限元问题，从而得到近似解。

在有限元方法中为适应不同问题的求解，以 Lagrange 插值函数为基础形成了一系列基函数，这些基函数在有限元方法中起到了至关重要的作用。有限元方法中基函数是一组线性无关的函数，它们的线性组合可以表示有限元空间中任意函数。在有限元方法中，基函数的选取是有限元方法的关键，不同基函数选取会导致不同的有限元方法，从而影响问题求解效果。在有限元方法中常用基函数^[4,6-8]有线性基函数、二次基函数、三次基函数等，这些基函数在不同维度下也有不同的表达形式，下面将对各个维度下常用基函数及其构造方法进行介绍。

1.5.1 1D 基函数与有限元空间

(1) 1D 线性元

对于 1D 线性元而言，各节点基函数为线性函数，根据前述基函数性质容易得到空间各节点基函数具有图 1-1 中所示曲线分布，容易得到对应节点基函数在对应节点处函数值为 1，在其他节点处函数值为 0。

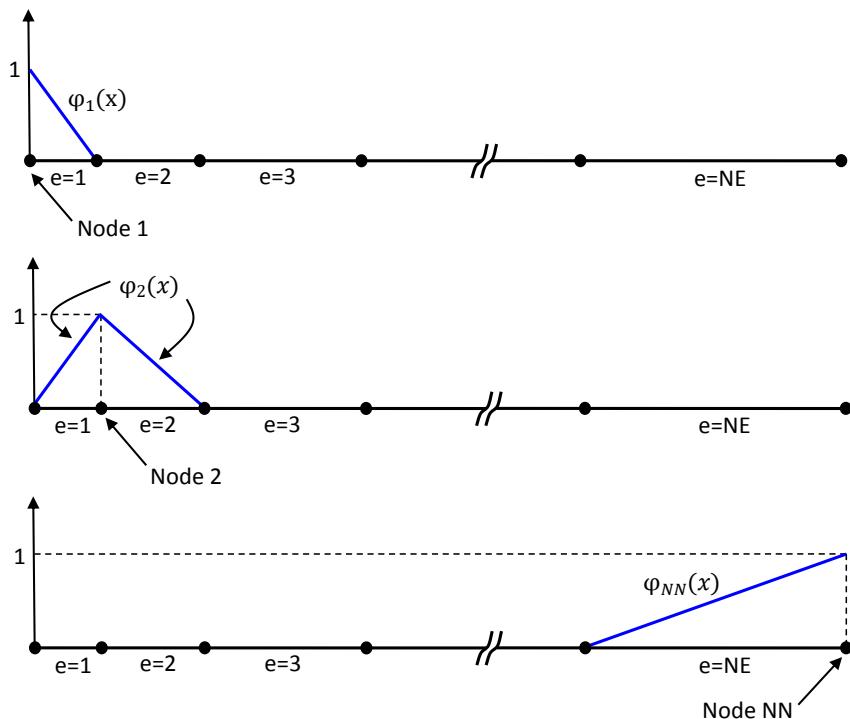


图 1-1 1D FEM 线性元基函数

由上述各个节点基函数的张集可以得到整个计算域空间，该空间称为有限元空间，

易得该空间表达式为：

$$\text{FE Space} \left\{ \begin{array}{l} \phi_1(x) = \begin{cases} \frac{x_2 - x}{h}, & \text{if } x_1 \leq x \leq x_2 \\ 0, & \text{otherwise} \end{cases} \\ \phi_j(x) = \begin{cases} \frac{x - x_{j-1}}{h}, & \text{if } x_{j-1} \leq x \leq x_j \\ \frac{x_{j+1} - x}{h}, & \text{if } x_j \leq x \leq x_{j+1} \\ 0, & \text{otherwise} \end{cases} \\ \phi_{NN}(x) = \begin{cases} \frac{x - x_{NE}}{h}, & \text{if } x_{NN-1} \leq x \leq x_{NN} \\ 0, & \text{otherwise} \end{cases} \end{array} \right. \quad (1.12)$$

在有限元方法中网格划分形状种类繁多，针对每一类型单独构造基函数是不现实的，为了更快、更方便得到仿真中单元节点的基函数，更好的方法是通过参考单元进行仿射变换（Affine Map）得到局部单元的节点基函数，随后由局部单元节点基函数的张集得到有限元空间的基函数，该基函数的求解流程可以描述为：reference \rightarrow local \rightarrow global。

构造线性元基函数时为了与前述内容保持一致，选择参考单元区间为 $[0,1]$ ，随后构造该区间上基函数作为参考元节点基函数 $N_i(\xi)$, $i = 0, 1$ ，之后通过仿射变换可以得到局部元节点基函数，这一过程如图 1-2 所示，最后将所有局部元节点基函数进行整合即可得到整个有限元空间的节点基函数。

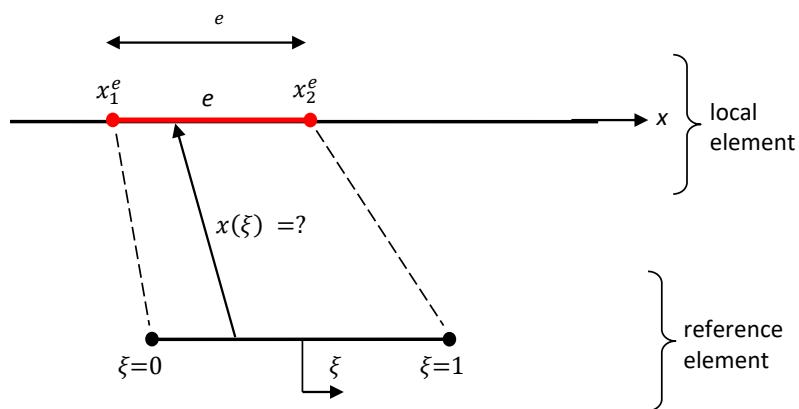


图 1-2 参考元到局部元 ($\text{reference} \rightarrow \text{local}$)

- 参考单元节点基函数构造 (Reference)

参考单元坐标系为： ξ ，其基函数表示为： $N_i(\xi)$ 。对于线性参考单元节点基函数而言其构造方法有多种，这里主要介绍常见的 3 种方法。

a) 参考单元节点基函数构造方法 1

根据单元类型与基函数基本定义，假设单元 2 个节点上基函数为：

$$\begin{cases} N_0(\xi) = a_0\xi + b_0 \\ N_1(\xi) = a_1\xi + b_1 \end{cases}, \quad \xi \in [0, 1] \quad (1.13)$$

将单元上两点坐标 ($\xi_0 = 0, \xi_1 = 1$) 代入上式中可以得到参考单元节点基函数：

$$\begin{cases} N_0(\xi_0) = a_0 \times 0 + b_0 = 1 \\ N_0(\xi_1) = a_0 \times 1 + b_0 = 0 \\ N_1(\xi_0) = a_1 \times 0 + b_1 = 0 \\ N_1(\xi_1) = a_1 \times 1 + b_1 = 1 \end{cases} \rightarrow \begin{cases} a_0 = -1 \\ b_0 = 1 \\ a_1 = 1 \\ b_1 = 0 \end{cases} \rightarrow \begin{cases} N_0(\xi) = 1 - \xi \\ N_1(\xi) = \xi \end{cases} \quad (1.14)$$

b) 参考单元节点基函数构造方法 2

根据基函数（形函数）的几何意义：对于 1D 而言，节点形函数表示给定位置点与单元另一节点的距离占单元长度的大小，其含义可看做该节点值在给定位置处的权重大小。所以基函数可以表示为：

$$N_i(\xi) = \frac{L_i}{L_e} \quad (1.15)$$

在 1D 情况下线段的长度可以由如下公式给出：

$$L_e = \begin{vmatrix} 1 & \xi_0 \\ 1 & \xi_1 \end{vmatrix} = \xi_1 - \xi_0 \quad (1.16)$$

同理给定节点 ξ 与节点 $\xi_0 = 0, \xi_1 = 1$ 的距离可以类比表示如上式，进而得到基函数如下：

$$\begin{cases} L_0 = \begin{vmatrix} 1 & \xi \\ 1 & \xi_1 \end{vmatrix} = \xi_1 - \xi \\ L_1 = \begin{vmatrix} 1 & \xi_0 \\ 1 & \xi \end{vmatrix} = \xi - \xi_0 \end{cases} \rightarrow \begin{cases} N_0(\xi) = \frac{L_0}{L_e} = \frac{\xi_1 - \xi}{\xi_1 - \xi_0} = 1 - \xi \\ N_1(\xi) = \frac{L_1}{L_e} = \frac{\xi - \xi_0}{\xi_1 - \xi_0} = \xi \end{cases} \quad (1.17)$$

c) 参考单元节点基函数构造方法 3

单元上任一节点的函数值根据线性插值可以轻松得到，其值也为单元上所有节点基函数与其对应函数值乘积的和，即：

$$u(\xi) = \begin{bmatrix} 1 & \xi \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 1 & \xi \end{bmatrix} \begin{bmatrix} 1 & \xi_0 \\ 1 & \xi_1 \end{bmatrix}^{-1} \begin{bmatrix} u_0 \\ u_1 \end{bmatrix} = N\delta = \begin{bmatrix} N_0 & N_1 \end{bmatrix} \delta \quad (1.18)$$

将给定节点 $\xi_0 = 0, \xi_1 = 1$ 代入可得到基函数如下：

$$N = \begin{bmatrix} 1 & \xi \end{bmatrix} \begin{bmatrix} 1 & \xi_0 \\ 1 & \xi_1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & \xi \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 - \xi & \xi \end{bmatrix} \quad (1.19)$$

可以看到当选定参考单元后，上述节点基函数构造方法可以得到相同节点基函数表示。

- 局部单元节点基函数构造 (Local)

局部单元基函数所在坐标系与仿真空间坐标系保持一致为： x ，基函数表示为： $\phi_i(x)$ ，在得到参考单元基函数后就可以方便的得到对应局部单元各节点基函数。

- a) 局部单元节点基函数构造方法 1

为了利用已经得到的参考单元 $[0, 1]$ 上的节点基函数来确定任一单元 $[x_n, x_{n+1}]$ 上的节点基函数，需要将变量 x 由局部单元 $[x_n, x_{n+1}]$ 转换至参考区间 $[0, 1]$ ，之后代入所得参考单元节点基函数中即可得到局部元的节点基函数表达式，如图 1-2 所示，这一过程称为等参映射。

- ✓ 等参映射方法 1

由 $x \in [x_n, x_{n+1}]$ 经过变换： $0 \leq x - x_n \leq x_{n+1} - x_n \rightarrow 0 \leq \frac{x - x_n}{x_{n+1} - x_n} \leq 1$ ，令 $\xi = \frac{x - x_n}{x_{n+1} - x_n} \in [0, 1]$ ，由此将 x 由 $[x_n, x_{n+1}]$ 等参变换至区间 $[0, 1]$ ，此时 x 满足参考元基函数。

- ✓ 等参映射方法 2

将参考单元映射到局部单元，就是经过一系列“线性变换（旋转、缩放、变形、升降维度）+ 平移”后将参考单元与局部单元各个顶点一一对应，对 1D 单元而言，由参考元映射到局部元的变换矩阵形式如下：

$$[x] = [b][\xi] + [a] \quad (1.20)$$

其中， $[x]$ 为局部元节点坐标， $[b]$ 为旋转、缩放、变形、升降维度矩阵， $[\xi]$ 为参考元节点坐标， $[a]$ 为平移矩阵。

将参考元节点坐标： $\xi_0 = 0, \xi_1 = 1$ ；局部元坐标： x_n, x_{n+1} 代入上式可以得到：

$$\begin{cases} x_n = b\xi_0 + a \\ x_{n+1} = b\xi_1 + a \end{cases} \rightarrow \begin{cases} x_n = b \times 0 + a \\ x_{n+1} = b \times 1 + a \end{cases} \rightarrow \begin{cases} a = x_n \\ b = x_{n+1} - x_n \end{cases} \quad (1.21)$$

进而用参考单元坐标可以表示局部坐标如下，同时经过变换有：

$$x = (x_{n+1} - x_n)\xi + x_n \rightarrow \xi = \frac{x - x_n}{x_{n+1} - x_n} \in [0, 1] \quad (1.22)$$

经过等参变换后 x 满足参考元基函数，将其代入即可得到局部元基函数表达式：

$$\begin{cases} \phi_{x_n}(x) = N_0(\xi) = \frac{x_{n+1} - x}{x_{n+1} - x_n} \\ \phi_{x_{n+1}}(x) = N_1(\xi) = \frac{x - x_n}{x_{n+1} - x_n} \end{cases} \quad (1.23)$$

b) 局部单元节点基函数构造方法 2

由基函数（形函数）几何意义可知其与坐标系选择无关。所以局部也可以表示为：

$$\phi_i(x) = \frac{L_i}{L_e} \quad (1.24)$$

在 1D 情况下线段的长度可以由如下公式给出：

$$L_e = \begin{vmatrix} 1 & x_n \\ 1 & x_{n+1} \end{vmatrix} = x_{n+1} - x_n \quad (1.25)$$

同理给定节点 x 与节点 x_n 、 x_{n+1} 的距离可以类比表示如上式，进而得到基函数如下：

$$\begin{cases} L_{x_n} = \begin{vmatrix} 1 & x \\ 1 & x_{n+1} \end{vmatrix} = x_{n+1} - x \\ L_{x_{n+1}} = \begin{vmatrix} 1 & x_n \\ 1 & x \end{vmatrix} = x - x_n \end{cases} \rightarrow \begin{cases} \phi_{x_n}(x) = \frac{L_{x_n}}{L_e} = \frac{x_{n+1} - x}{x_{n+1} - x_n} \\ \phi_{x_{n+1}}(x) = \frac{L_{x_{n+1}}}{L_e} = \frac{x - x_n}{x_{n+1} - x_n} \end{cases} \quad (1.26)$$

(2) 1D 2 次元

类比 1D FEM 线性元基函数构造方法，为了得到高阶单元在参考元与对应局部元内增加一个中点，根据 Lagrange 插值基函数的基本定义可得到该基函数具有最高为 2 次插值多项式。

• 2 次元参考单元节点基函数构造 (Reference)

参考单元坐标系为： ξ ，其基函数表示为： $N_i(\xi)$ 。

a) 参考单元节点基函数构造方法 1

根据单元类型与基函数基本定义，将给定参考单元节点坐标代入上式可以得到：

$$\begin{cases} N_0(\xi) = a_0\xi^2 + b_0\xi + c_0 \\ N_1(\xi) = a_1\xi^2 + b_1\xi + c_1, \quad \xi \in [0, 1] \\ N_2(\xi) = a_2\xi^2 + b_2\xi + c_2 \end{cases} \quad (1.27)$$

将单元上 3 点坐标 ($\xi_0 = 0$, $\xi_1 = 1$, $\xi_2 = \frac{1}{2}$) 代入上式中 (这里需要注意单元中各节点顺序), 可以得到参考单元基函数如下:

$$\left\{ \begin{array}{l} N_0(\xi_0) = c_0 = 1 \\ N_0(\xi_1) = a_0 + b_0 + c_0 = 0 \\ N_0(\xi_2) = \frac{1}{4}a_0 + \frac{1}{2}b_0 + c_0 = 0 \\ N_1(\xi_0) = c_1 = 0 \\ N_1(\xi_1) = a_1 + b_1 + c_1 = 1 \\ N_1(\xi_2) = \frac{1}{4}a_1 + \frac{1}{2}b_1 + c_1 = 0 \\ N_2(\xi_0) = c_2 = 0 \\ N_2(\xi_1) = a_2 + b_2 + c_2 = 0 \\ N_2(\xi_2) = \frac{1}{4}a_2 + \frac{1}{2}b_2 + c_2 = 1 \end{array} \right. \rightarrow \left\{ \begin{array}{l} a_0 = 2, b_0 = -3, c_0 = 1 \\ a_1 = 2, b_1 = -1, c_1 = 0 \\ a_2 = -4, b_0 = 4, c_2 = 0 \end{array} \right. \quad (1.28)$$

所以 1D FEM 2 次元参考基函数为:

$$\left\{ \begin{array}{l} N_0(\xi) = 2\xi^2 - 3\xi + 1 \\ N_1(\xi) = 2\xi^2 - \xi \\ N_2(\xi) = -4\xi^2 + 4\xi \end{array} \right. , \quad \xi \in [0, 1] \quad (1.29)$$

b) 参考单元节点基函数构造方法 2

单元上任一节点函数值与各节点函数值关系为:

$$u(\xi) = a_1 + a_2\xi + a_3\xi^2 = \begin{bmatrix} 1 & \xi & \xi^2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad (1.30)$$

将单元节点坐标及对应函数值代入可得:

$$\begin{bmatrix} u_0 \\ u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 1 & \xi_0 & \xi_0^2 \\ 1 & \xi_1 & \xi_1^2 \\ 1 & \xi_2 & \xi_2^2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \rightarrow \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & \xi_0 & \xi_0^2 \\ 1 & \xi_1 & \xi_1^2 \\ 1 & \xi_2 & \xi_2^2 \end{bmatrix}^{-1} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \end{bmatrix} \quad (1.31)$$

进而得到基函数如下：

$$\begin{aligned}
 N &= \begin{bmatrix} 1 & \xi & \xi^2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & \xi & \xi^2 \end{bmatrix} \begin{bmatrix} 1 & \xi_0 & \xi_0^2 \\ 1 & \xi_1 & \xi_1^2 \\ 1 & \xi_2 & \xi_2^2 \end{bmatrix}^{-1} \\
 &= \begin{bmatrix} 1 & \xi & \xi^2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & \frac{1}{2} & \frac{1}{4} \end{bmatrix}^{-1} = \begin{bmatrix} 1 & \xi & \xi^2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -3 & -1 & 4 \\ 2 & 2 & -4 \end{bmatrix} \\
 &= \begin{bmatrix} 2\xi^2 - 3\xi + 1 & 2\xi^2 - \xi & -4\xi^2 + 4\xi \end{bmatrix} = \begin{bmatrix} N_0 & N_1 & N_2 \end{bmatrix}
 \end{aligned} \tag{1.32}$$

可以看到当选定参考单元后，两种构造基函数的方法都可以得到相同的基函数表示，各节点基函数在空间为一抛物线，如图 1-3 所示。

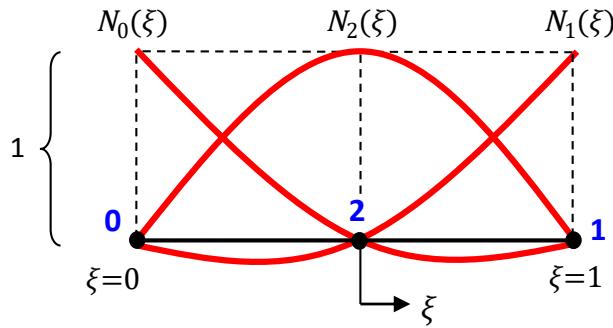


图 1-3 1D2 次元参考基函数

• 2 次元局部单元节点基函数构造 (Local)

局部单元基函数所在坐标系与仿真空间坐标系保持一致为： x ，基函数表示为： $\phi_i(x)$ ，在得到参考单元基函数后就可以方便的得到对应局部单元的基函数。

a) 局部单元节点基函数构造方法 1

与线性元局部单元基函数构造方法 1 相同，将局部元变量范围进行等参变换转换至参考区间 $\xi = \frac{x - x_n}{x_{n+1} - x_n} \in [0, 1]$ ，随后变换后的变量代入既得参考元基函数中即可得到局部元基函数表达式：

$$\left\{ \begin{array}{l} \phi_{x_n}(x) = N_0(\xi) = 2 \left(\frac{x - x_n}{x_{n+1} - x_n} \right)^2 - 3 \frac{x - x_n}{x_{n+1} - x_n} + 1 \\ \phi_{x_{n+1}}(x) = N_1(\xi) = 2 \left(\frac{x - x_n}{x_{n+1} - x_n} \right)^2 - \frac{x - x_n}{x_{n+1} - x_n} \\ \phi_{\frac{x_n+x_{n+1}}{2}}(x) = N_2(\xi) = -4 \left(\frac{x - x_n}{x_{n+1} - x_n} \right)^2 + 4 \frac{x - x_n}{x_{n+1} - x_n} \end{array} \right. \tag{1.33}$$

为了方便后续程序实现，将其展开整理可以得到：

$$\begin{cases} \phi_{x_n}(x) = \frac{2x^2 - (x_n + 3x_{n+1})x + x_{n+1}^2 + x_n x_{n+1}}{h^2} \\ \phi_{x_{n+1}}(x) = \frac{2x^2 - (3x_n + x_{n+1})x + x_n^2 + x_n x_{n+1}}{h^2} \\ \phi_{\frac{x_n+x_{n+1}}{2}}(x) = \frac{-4x^2 + 4(x_n + x_{n+1})x - 4x_n x_{n+1}}{h^2} \end{cases} \quad (1.34)$$

局部元基函数的 1 次导数和 2 次导数分别为：

$$\begin{cases} \phi'_{x_n}(x) = \frac{4x - (x_n + 3x_{n+1})}{h^2} \\ \phi'_{x_{n+1}}(x) = \frac{4x - (3x_n + x_{n+1})}{h^2} \\ \phi'_{\frac{x_n+x_{n+1}}{2}}(x) = \frac{-8x + 4(x_n + x_{n+1})}{h^2} \end{cases}, \quad \begin{cases} \phi^{(2)}_{x_n}(x) = \frac{4}{h^2} \\ \phi^{(2)}_{x_{n+1}}(x) = \frac{4}{h^2} \\ \phi^{(2)}_{\frac{x_n+x_{n+1}}{2}}(x) = \frac{-8}{h^2} \end{cases} \quad (1.35)$$

b) 局部单元节点基函数构造方法 2

由节点函数值的构成原理可知其与坐标系选择无关，类比参考单元基函数构造方法 2 可以构造得到对应局部单元基函数，设节点函数值为：

$$u(x) = a_1 + a_2x + a_3x^2 = \begin{bmatrix} 1 & x & x^2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad (1.36)$$

将单元节点坐标及对应函数值代入可得：

$$\begin{bmatrix} u_0 \\ u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 1 & x_n & x_n^2 \\ 1 & x_{n+1} & x_{n+1}^2 \\ 1 & \frac{x_n+x_{n+1}}{2} & \left(\frac{x_n+x_{n+1}}{2}\right)^2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \rightarrow \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & x_n & x_n^2 \\ 1 & x_{n+1} & x_{n+1}^2 \\ 1 & \frac{x_n+x_{n+1}}{2} & \left(\frac{x_n+x_{n+1}}{2}\right)^2 \end{bmatrix}^{-1} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \end{bmatrix} \quad (1.37)$$

进而得到基函数如下：

$$N = \begin{bmatrix} 1 & x & x^2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & x & x^2 \end{bmatrix} \begin{bmatrix} 1 & x_n & x_n^2 \\ 1 & x_{n+1} & x_{n+1}^2 \\ 1 & \frac{x_n+x_{n+1}}{2} & \left(\frac{x_n+x_{n+1}}{2}\right)^2 \end{bmatrix}^{-1} \quad (1.38)$$

将所得局部基函数在整个计算域进行张集操作即可得到有限元空间，其分布如图 1-3 所示。

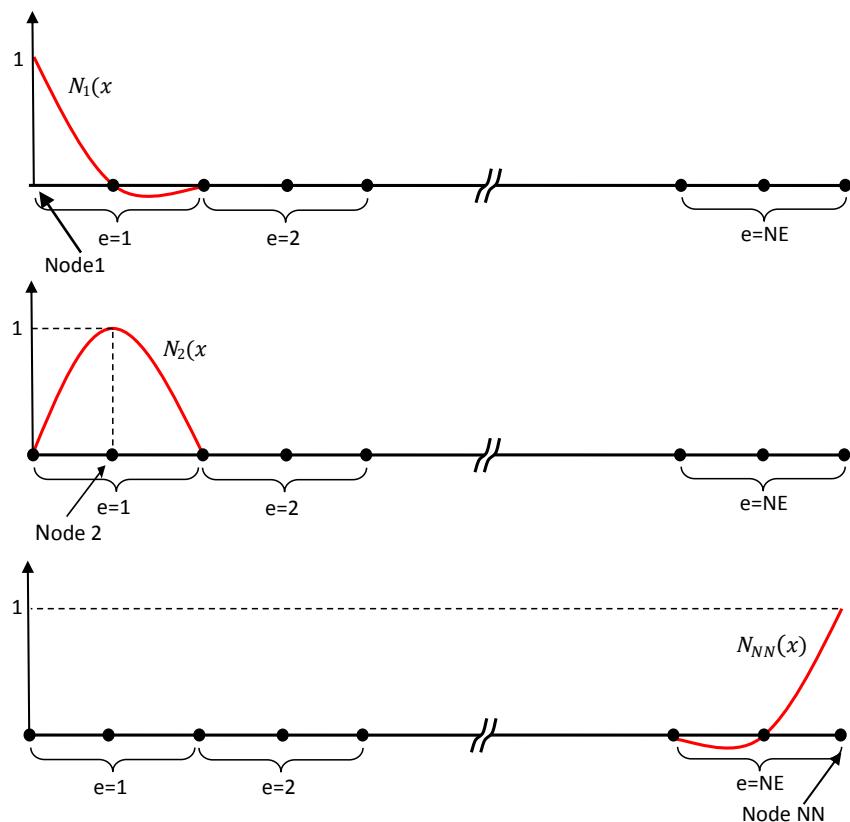


图 1-4 1D2 次元有限元空间

1.5.2 2D 基函数与有限元空间

对于 2D 空间其空间离散后形状较 1D 空间离散更多，对应网格单元的基函数也更加丰富，在 2D FEM 方法中常用的网格单元为三角形单元（Triangular）、四边形单元（Quadrilateral），或者混合单元（Multi Type）。本节对 2D 空间常见参考单元类型及其对应的基函数进行介绍。

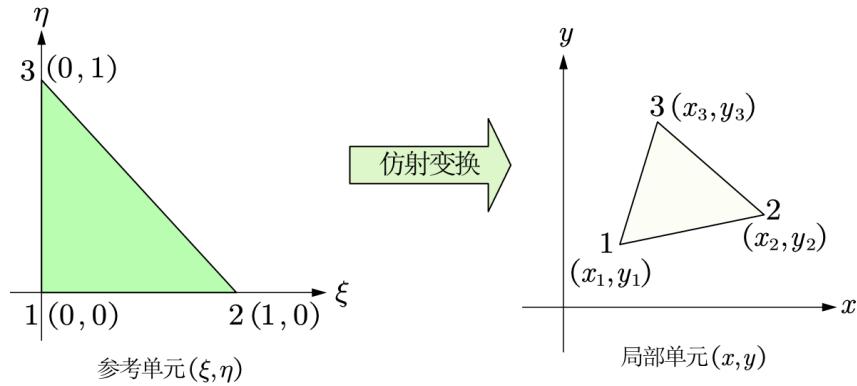
(1) 2D 三节点单元

参考单元的节点编号起点为左下角开始沿逆时针变化（Counterclockwise, CCW），参考单元所在坐标系为： (ξ, η) ，2D FEM 仿真所在全局坐标系为： (x, y) ，参考元与局部元的变换如图 1-5 所示。

2D FEM 单元节点基函数构造方法与 1D FEM 中类似，下面主要介绍 2 种常见的基函数构造方法。

- 2D 三节点参考单元节点基函数构造（Reference）

- a) 参考单元节点基函数构造方法 1

图 1-5 二维三节点单元参考到局部变换 (*reference* → *local*)

参考单元确定后，根据选择单元类型（这里为三角形线性元）可以直接假设三角形线性单元各节点基函数表达式为： $N_i = a_i\xi + b_i\eta + c_i$ ，随后结合基函数特性，将节点坐标代入可以得到 9 个线性方程组，从而得到系数 a_i , b_i , c_i ，进而确定各节点基函数表达式，这里不再详细演示该计算过程。

b) 参考单元节点基函数构造方法 2

在 1D 基函数构造方法中实际上已经介绍了基函数构造的更方便的一种方法：通过构造函数根据采用的参考单元形状，结合帕斯卡三角形（如图 1-6 所示），结果直观表现为待定系数法）。

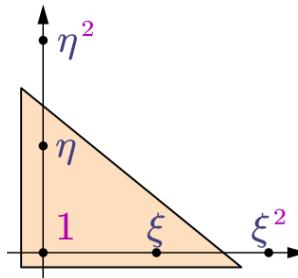


图 1-6 二维三节点单元形函数组成

参考单元内任一点函数值可以表示为该单元上给定节点坐标及对应函数值的线性拟合：

$$u(\xi, \eta) = \begin{bmatrix} 1 & \xi & \eta \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & \xi & \eta \end{bmatrix} \begin{bmatrix} 1 & \xi_1 & \eta_1 \\ 1 & \xi_2 & \eta_2 \\ 1 & \xi_3 & \eta_3 \end{bmatrix}^{-1} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = N\delta \quad (1.39)$$

将单元上 3 点坐标：(0, 0), (1, 0), (0, 1) 代入上式中可以得到参考单元基函数如

下：

$$N = \begin{bmatrix} 1 & \xi & \eta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & \xi & \eta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 - \xi - \eta & \xi & \eta \end{bmatrix} \quad (1.40)$$

c) 参考单元节点基函数构造方法 3

根据基函数（形函数）的几何意义：对于 2D 而言，节点形函数表示给定位置点与单元其他节点组成的三角形面积占单元面积的大小，其含义同样可以看作对应节点 (ξ_i, η_i) 在给定空间位置节点处 (ξ, η) 的权重大小：

$$N_i(\xi, \eta) = \frac{S_i}{S_e} \quad (1.41)$$

在 2D 情况下参考三角形的面积可以由如下公式给出：

$$S_e = \frac{1}{2} \begin{vmatrix} 1 & \xi_1 & \eta_1 \\ 1 & \xi_2 & \eta_2 \\ 1 & \xi_3 & \eta_3 \end{vmatrix} = \frac{1}{2} \begin{vmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{vmatrix} = \frac{1}{2} \quad (1.42)$$

如图 1-7 所示为三角形单元形函数及其几何意义示意，如上所述三角形单元形函数可以表示为各个三角形的面积比。

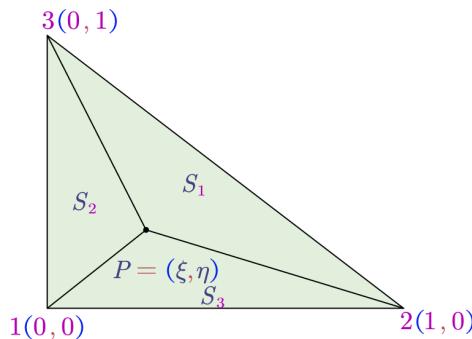


图 1-7 三角形单元形函数的几何意义

同理可以计算得到 S_1, S_2, S_3 ：

$$S_1 = \frac{1}{2} \begin{vmatrix} 1 & \xi_1 & \eta_1 \\ 1 & \xi_2 & \eta_2 \\ 1 & \xi_3 & \eta_3 \end{vmatrix} = \frac{1}{2} \begin{vmatrix} 1 & \xi & \eta \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{vmatrix} = \frac{1}{2} (1 - \xi - \eta)$$

$$\begin{aligned}
 S_2 &= \frac{1}{2} \begin{vmatrix} 1 & \xi_1 & \eta_1 \\ 1 & \xi & \eta \\ 1 & \xi_3 & \eta_3 \end{vmatrix} = \frac{1}{2} \begin{vmatrix} 1 & 0 & 0 \\ 1 & \xi & \eta \\ 1 & 0 & 1 \end{vmatrix} = \frac{1}{2} \xi \\
 S_3 &= \frac{1}{2} \begin{vmatrix} 1 & \xi_1 & \eta_1 \\ 1 & \xi_2 & \eta_2 \\ 1 & \xi & \eta \end{vmatrix} = \frac{1}{2} \begin{vmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & \xi & \eta \end{vmatrix} = \frac{1}{2} \eta
 \end{aligned} \tag{1.43}$$

根据基函数几何意义可以得到基函数为：

$$\begin{cases} N_1(\xi, \eta) = S_1/S_e = 1 - \xi - \eta \\ N_2(\xi, \eta) = S_2/S_e = \xi \\ N_3(\xi, \eta) = S_3/S_e = \eta \end{cases} \tag{1.44}$$

由此可以得到三角形单元的参考基函数，如图 1-8 所示。

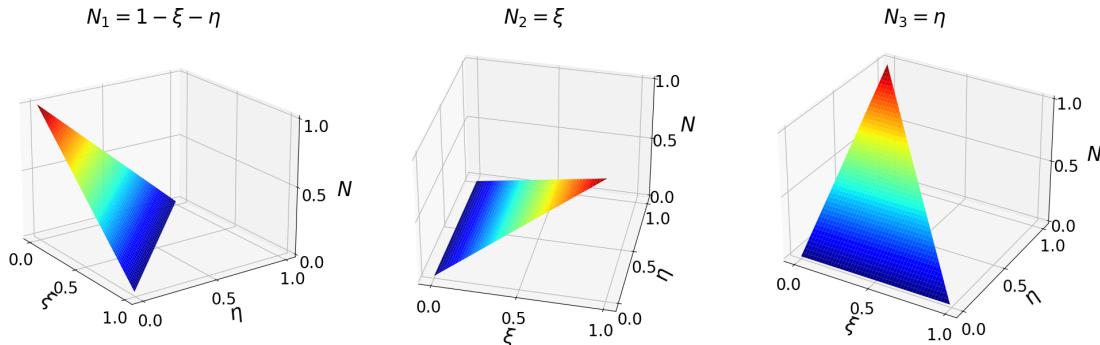


图 1-8 2D 三节点单元参考基函数

- 2D 三节点局部单元节点基函数构造 (Local)

将参考单元映射到局部单元，就是经过一系列“线性变换（旋转、缩放、变形、升降维度）+ 平移”后将参考单元与局部单元的各个顶点一一对应，对 2D 单元而言由参考元映射到局部元的变换矩阵形式如下：

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b_x & c_x \\ b_y & c_y \end{bmatrix} \begin{bmatrix} \xi \\ \eta \end{bmatrix} + \begin{bmatrix} a_x \\ a_y \end{bmatrix} \tag{1.45}$$

其中， $\begin{bmatrix} x \\ y \end{bmatrix}$ 为局部元节点坐标， $\begin{bmatrix} b_x & c_x \\ b_y & c_y \end{bmatrix}$ 为旋转、缩放、变形、升降维度矩阵，该矩阵也被定义为雅可比 (Jacobian) 矩阵， $\begin{bmatrix} \xi \\ \eta \end{bmatrix}$ 为参考元节点坐标， $\begin{bmatrix} a_x \\ a_y \end{bmatrix}$ 为平移矩阵。

对三角形单元而言，将参考元节点坐标：(0, 0), (1, 0), (0, 1)，局部元坐标：(x_1, y_1), (x_2, y_2), (x_3, y_3) 代入上式可以得到：

$$\begin{cases} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} b_x & c_x \\ b_y & c_y \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} a_x \\ a_y \end{bmatrix} \\ \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} b_x & c_x \\ b_y & c_y \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} a_x \\ a_y \end{bmatrix} \\ \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} = \begin{bmatrix} b_x & c_x \\ b_y & c_y \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} a_x \\ a_y \end{bmatrix} \end{cases} \quad (1.46)$$

所以可以得到系数为：

$$\begin{cases} a_x = x_1, \quad a_y = y_1 \\ b_x = x_2 - x_1, \quad b_y = y_2 - y_1 \\ c_x = x_3 - x_1, \quad c_y = y_3 - y_1 \end{cases} \quad (1.47)$$

进而用参考单元坐标可以表示局部坐标如下：

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \begin{bmatrix} \xi \\ \eta \end{bmatrix} + \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \quad (1.48)$$

从而可以得到：

$$\begin{cases} \xi = \frac{(x - x_1)(y_3 - y_1) - (x_3 - x_1)(y - y_1)}{|J|} \\ \eta = \frac{-(x - x_1)(y_2 - y_1) + (x_2 - x_1)(y - y_1)}{|J|} \end{cases} \quad (1.49)$$

其中， $|J| = (x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)$, $J = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix}$ 。

将等参变换后的坐标代入参考单元节点基函数中即可得到对应的局部元基函数，从上述表达式可以看到根据等参变换将局部元转换至参考元下随后得到的局部元基函数形式已经较为复杂，事实上在进行有限元仿真时通常将局部元转换至参考元下进行相关计算，而在此过程中 Jacobian 矩阵、其逆矩阵及行列式在计算中都发挥着重要

作用。进一步注意到雅可比矩阵与参考单元基函数及局部单元坐标满足：

$$\begin{aligned} J = \frac{\partial(x, y)}{\partial(\xi, \eta)} &= \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{NN} \frac{\partial N_i(\xi, \eta)}{\partial \xi} x_i & \sum_{i=1}^{NN} \frac{\partial N_i(\xi, \eta)}{\partial \eta} x_i \\ \sum_{i=1}^{NN} \frac{\partial N_i(\xi, \eta)}{\partial \xi} y_i & \sum_{i=1}^{NN} \frac{\partial N_i(\xi, \eta)}{\partial \eta} y_i \end{bmatrix} \\ &= \begin{bmatrix} x_1 & x_2 & \cdots & x_{NN} \\ y_1 & y_2 & \cdots & y_{NN} \end{bmatrix} \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & \frac{\partial N_1}{\partial \eta} \\ \frac{\partial N_2}{\partial \xi} & \frac{\partial N_2}{\partial \eta} \\ \vdots & \vdots \\ \frac{\partial N_{NN}}{\partial \xi} & \frac{\partial N_{NN}}{\partial \eta} \end{bmatrix} \end{aligned} \quad (1.50)$$

其中， NN 表示单元节点数量，对三角形单元 $NN = 3$ 。将对应参考单元节点基函数及局部单元坐标代入上式可以得到雅可比矩阵为：

$$J = \frac{\partial(x, y)}{\partial(\xi, \eta)} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} -1 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \quad (1.51)$$

性质 → 参考单元坐标转换至局部单元坐标

进一步可以用基函数与局部单元坐标的线性组合得到局部单元坐标系的参考元表达，由此该表达满足参考单元基函数性质：

$$\begin{cases} x(\xi, \eta) = \sum_{i=1}^{NN} N_i(\xi, \eta) x_i \\ y(\xi, \eta) = \sum_{i=1}^{NN} N_i(\xi, \eta) y_i \end{cases} \quad (1.52)$$

上式为有限元单元坐标变换通用方法，根据该变换将参考三角形单元坐标：(0,0), (1,0), (0,1) 代入并变换后可以得到：

$$\begin{cases} x(\xi, \eta) = (1 - \xi - \eta)x_1 + \xi x_2 + \eta x_3 = (x_2 - x_1)\xi + (x_3 - x_1)\eta + x_1 \\ y(\xi, \eta) = (1 - \xi - \eta)y_1 + \xi y_2 + \eta y_3 = (y_2 - y_1)\xi + (y_3 - y_1)\eta + y_1 \end{cases} \quad (1.53)$$

将雅可比矩阵引入上述变换有：

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & \frac{\partial N_1}{\partial \eta} \\ \frac{\partial N_2}{\partial \xi} & \frac{\partial N_2}{\partial \eta} \\ \frac{\partial N_3}{\partial \xi} & \frac{\partial N_3}{\partial \eta} \end{bmatrix} \begin{bmatrix} \xi \\ \eta \end{bmatrix} + \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \quad (1.54)$$

可以看到所得结果和采用矩阵变换之后结果一致，通用方法的表示计算更简洁。

(2) 2D 四节点单元

2D FEM 采用四边形参考单元，其节点编号也为左下角开始沿逆时针变化（Counterclockwise, CCW），参考单元所在坐标系为： (ξ, η) ，2D FEM 仿真所在全局坐标系为： (x, y) ，参考元与局部元的变换如图 1-9 所示。

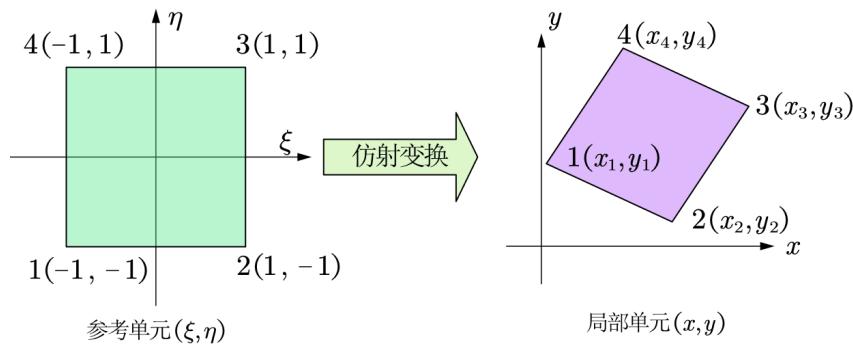


图 1-9 二维四节点单元参考到局部变换 (*reference \rightarrow local*)

- 2D 四节点参考单元节点基函数构造 (Reference)

- a) 参考单元节点基函数构造方法 1

类比三角形基函数的构造方法，同样可以得到四边形的基函数构造方法。参考单元确定后，根据选择单元类型（这里为四边形线性元）可以直接假设四边形线性单元各节点基函数表达式为： $N_i = a_i\xi + b_i\eta + c_i\xi\eta + d_i$ ，随后结合基函数特性，将节点坐标代入可以得到 16 个线性方程组，从而得到系数 a_i , b_i , c_i , d_i ，进而确定各节点基函数表达式。

- b) 参考单元节点基函数构造方法 2

为了避免求解复杂线性方程组可以利用帕斯卡三角形计算四边形基函数：

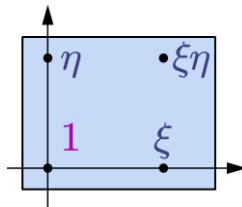


图 1-10 二维四节点单元形函数组成

$$u(\xi, \eta) = \begin{bmatrix} 1 & \xi & \eta & \xi\eta \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} 1 & \xi & \eta & \xi\eta \end{bmatrix} \begin{bmatrix} 1 & \xi_1 & \eta_1 & \xi_1\eta_1 \\ 1 & \xi_2 & \eta_2 & \xi_2\eta_2 \\ 1 & \xi_3 & \eta_3 & \xi_3\eta_3 \\ 1 & \xi_4 & \eta_4 & \xi_4\eta_4 \end{bmatrix}^{-1} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = N\delta \quad (1.55)$$

将单元上 4 点坐标: (-1, -1), (1, -1), (1, 1), (-1, 1) 代入上式中可以得到参考单元基函数如下:

$$N = \begin{bmatrix} 1 & \xi & \eta & \xi\eta \end{bmatrix} \begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & \xi & \eta & \xi\eta \end{bmatrix} \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} \frac{1}{4}(1-\xi)(1-\eta) & \frac{1}{4}(1+\xi)(1-\eta) & \frac{1}{4}(1+\xi)(1+\eta) & \frac{1}{4}(1-\xi)(1+\eta) \end{bmatrix} \quad (1.56)$$

c) 参考单元节点基函数构造方法 3

根据基函数(形函数)的几何意义:对于 2D 四节点单元而言,节点形函数表示给定位置点与单元其他节点组成的四边形面积占单元面积的大小,其含义同样可以看作对应节点 (ξ_i, η_i) 在给定空间位置节点处 (ξ, η) 的权重大小:

$$N_i(\xi, \eta) = \frac{S_i}{S_e} \quad (1.57)$$

参考单元为标准正方形,易得其面积为 $S_e = 4$, 单元内任意位置将该单元分割为 4 个不同大小的长方形,由此可以分别得到对应节点的基函数:

$$\begin{cases} N_1(\xi, \eta) = S_1/S_e = \frac{1}{4}(1-\xi)(1-\eta) \\ N_2(\xi, \eta) = S_2/S_e = \frac{1}{4}(1+\xi)(1-\eta) \\ N_3(\xi, \eta) = S_3/S_e = \frac{1}{4}(1+\xi)(1+\eta) \\ N_4(\xi, \eta) = S_4/S_e = \frac{1}{4}(1-\xi)(1+\eta) \end{cases} \quad (1.58)$$

由此可以得到 2D 四节点参考单元基函数,如图 1-11 所示。

- **2D 四节点局部单元节点基函数构造 (Local)**

通过三角形单元局部基函数的构造方法,可以知道局部单元上任一空间位置的坐标可以由参考单元基函数及局部单元坐标进行插值得到:

$$\begin{cases} x(\xi, \eta) = \sum_{i=1}^{NN} N_i(\xi, \eta)x_i \\ y(\xi, \eta) = \sum_{i=1}^{NN} N_i(\xi, \eta)y_i \end{cases} \quad (1.59)$$

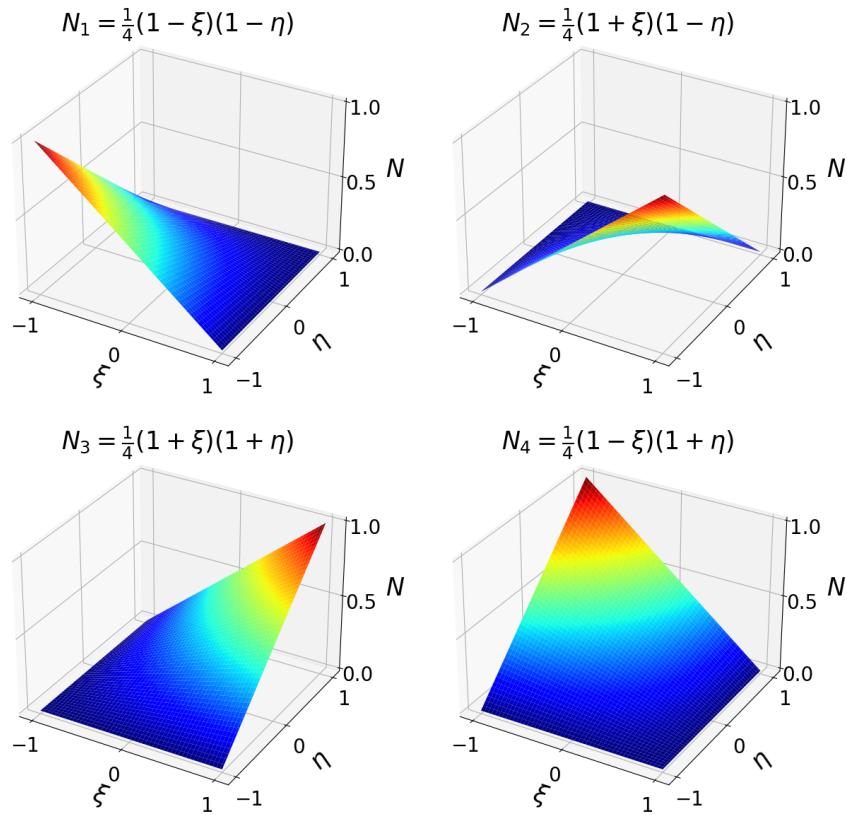


图 1-11 2D 四节点单元参考基函数

其中, NN 表示单元节点的数量, 对四边形单元有: $NN = 4$ 。

将参考元节点坐标: $(-1, -1)$, $(1, -1)$, $(1, 1)$, $(-1, 1)$ 代入上式可以得到全局坐标的参考元坐标系表示, 经过变换可以得到对应的局部元基函数。但是通常不进行复杂的变换, 经过观察可以看到上述变换能够将任意一单元的坐标转换至参考元坐标系表示, 且表示较为方便, 所以有限元的计算通常将其转换至参考单元下进行计算。

(3) 2D 六节点单元

对于高阶相对较为复杂的基函数比较方便的是使用更为通用的基函数构造方法, 即“帕斯卡三角形”方法。构造高阶单元需要在单元内添加更多合适数量并且能够表现该单元特性的点, 对于三角形单元而言采用如图 1-12 所示的“帕斯卡三角形”使其内部包含对应精度要求的点的数量即可, 这里构造二阶三角形单元包含 6 个点。

- 2D 六节点参考单元节点基函数构造 (Reference)

- a) 参考单元节点基函数构造方法 1

这里为六节点三角形单元, 可以直接假设该单元各节点基函数表达式为: $N_i =$

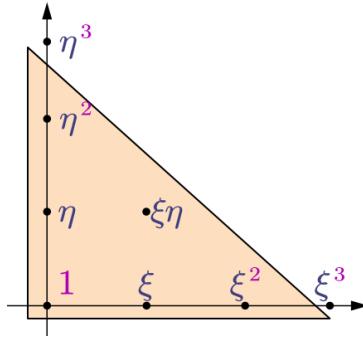


图 1-12 二维六节点单元形函数组成

$a_i\xi^2 + b_i\xi\eta + c_i\eta^2 + d_i\xi + e_i\eta + f_i$, 随后结合基函数特性, 将节点坐标代入可以得到 36 个线性方程组, 从而得到系数 $a_i, b_i, c_i, d_i, e_i, f_i$, 进而确定各节点基函数表达式, 这里不再详细演示该计算过程。

b) 参考单元节点基函数构造方法 2

参考单元内任一点函数值可以表示为该单元上给定节点坐标及对应函数值的线性拟合:

$$u(\xi, \eta) = \begin{bmatrix} 1 & \xi & \eta & \xi\eta & \xi^2 & \eta^2 \end{bmatrix} \begin{bmatrix} 1 & \xi_1 & \eta_1 & \xi_1\eta_1 & \xi_1^2 & \eta_1^2 \\ 1 & \xi_2 & \eta_2 & \xi_2\eta_2 & \xi_2^2 & \eta_2^2 \\ 1 & \xi_3 & \eta_3 & \xi_3\eta_3 & \xi_3^2 & \eta_3^2 \\ 1 & \xi_4 & \eta_4 & \xi_4\eta_4 & \xi_4^2 & \eta_4^2 \\ 1 & \xi_5 & \eta_5 & \xi_5\eta_5 & \xi_5^2 & \eta_5^2 \\ 1 & \xi_6 & \eta_6 & \xi_6\eta_6 & \xi_6^2 & \eta_6^2 \end{bmatrix}^{-1} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = N\delta \quad (1.60)$$

将单元上 6 个点坐标: $(0,0), (1,0), (0,1), (0.5,0), (0.5,0.5), (0,0.5)$ 按顺序代入上式中可以得到参考单元基函数如下:

$$N = \begin{bmatrix} 1 & \xi & \eta & \xi\eta & \xi^2 & \eta^2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0.5 & 0 & 0 & 0.25 & 0 \\ 1 & 0.5 & 0.5 & 0.25 & 0.25 & 0.25 \\ 1 & 0 & 0.5 & 0 & 0 & 0.25 \end{bmatrix}^{-1} \quad (1.61)$$

所以可以得到对应各节点基函数如下：

$$\begin{cases} N_1(\xi, \eta) = 2\xi^2 + 2\eta^2 + 4\xi\eta - 3\xi - 3\eta + 1 \\ N_2(\xi, \eta) = 2\xi^2 - \xi \\ N_3(\xi, \eta) = 2\eta^2 - \eta \\ N_4(\xi, \eta) = -4\xi^2 - 4\xi\eta + 4\xi \\ N_5(\xi, \eta) = 4\xi\eta \\ N_6(\xi, \eta) = -4\eta^2 - 4\xi\eta + 4\eta \end{cases} \quad (1.62)$$

由此可以得到 2D 六节点单元的参考基函数，如图 1-12 所示。

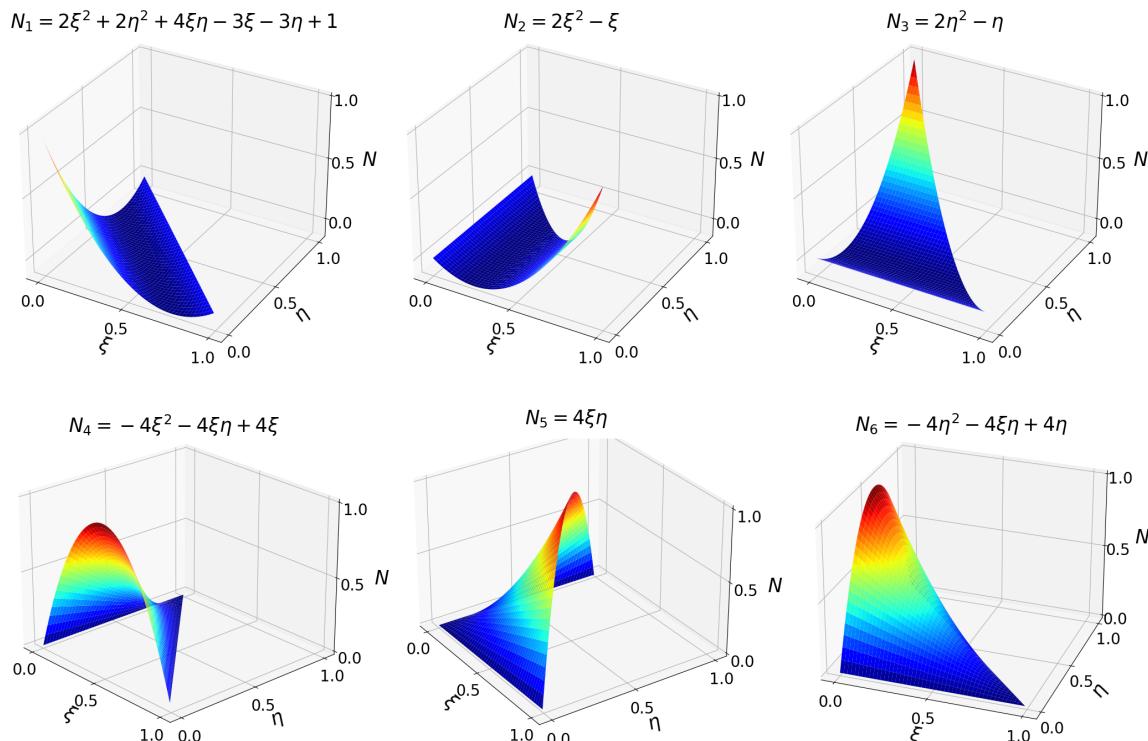


图 1-13 二维六节点参考单元基函数

• 2D 六节点局部单元节点基函数构造 (Local)

对于高阶单元其局部基函数构造首先由坐标转换式 (1.52) 得到局部单元在参考坐标系下的表示，随后将所得坐标代入上述参考元基函数即可得到高阶局部单元各节点基函数，这里不再详细演示。

(4) 2D 九节点单元

类比三角形基函数的构造方法，同样可以得到四边形的基函数构造方法，如图 1-14 所示为二维 9 节点高阶四边形单元对应的“帕斯卡三角形”。

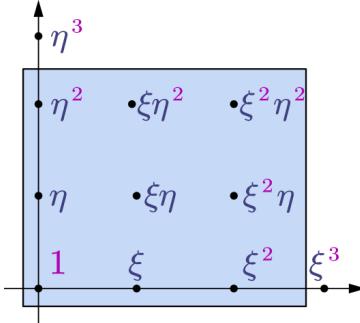


图 1-14 二维九节点单元形函数组成

- 2D 九节点参考单元节点基函数构造 (Reference)

- a) 参考单元节点基函数构造方法 1

这里为九节点四边形单元，假设该单元各节点基函数表达式为： $N_i = a_i\xi^2\eta^2 + b_i\xi^2\eta + c_i\xi\eta^2 + d_i\xi^2 + e_i\xi\eta + f_i\eta^2 + g_i\xi + h_i\eta + i_i$ ，随后结合基函数特性，将节点坐标代入可以得到 36 个线性方程组，从而得到系数 $a_i, b_i, c_i, d_i, e_i, f_i, g_i, h_i, i_i$ ，进而确定各节点基函数表达式，这里不再详细演示该计算过程。

- b) 参考单元节点基函数构造方法 2

类比二维六节点单元形函数，将该单元 9 个点坐标：(-1,-1), (1,-1), (1,1), (-1,1), (0,-1), (1,0), (0,1), (-1,0), (0,0) 按序代入上式中可以得到参考单元基函数如下：

$$\left\{ \begin{array}{l} N_1(\xi, \eta) = 0.25\xi^2\eta^2 - 0.25\xi\eta^2 - 0.25\xi^2\eta + 0.25\xi\eta \\ N_2(\xi, \eta) = 0.25\xi^2\eta^2 + 0.25\xi\eta^2 - 0.25\xi^2\eta - 0.25\xi\eta \\ N_3(\xi, \eta) = 0.25\xi^2\eta^2 + 0.25\xi\eta^2 + 0.25\xi^2\eta + 0.25\xi\eta \\ N_4(\xi, \eta) = 0.25\xi^2\eta^2 - 0.25\xi\eta^2 + 0.25\xi^2\eta - 0.25\xi\eta \\ N_5(\xi, \eta) = -0.5\xi^2\eta^2 + 0.5\xi^2\eta + 0.5\eta^2 - 0.5\eta \\ N_6(\xi, \eta) = -0.5\xi^2\eta^2 - 0.5\xi\eta^2 + 0.5\xi^2 + 0.5\xi \\ N_7(\xi, \eta) = -0.5\xi^2\eta^2 - 0.5\xi^2\eta + 0.5\eta^2 + 0.5\eta \\ N_8(\xi, \eta) = -0.5\xi^2\eta^2 + 0.5\xi\eta^2 + 0.5\xi^2 - 0.5\xi \\ N_9(\xi, \eta) = \xi^2\eta^2 - \eta^2 - \xi^2 + 1 \end{array} \right. \quad (1.63)$$

由此可以得到 2D 九节点单元参考基函数，如图 1-15 所示。

- 2D 九节点局部单元节点基函数构造 (Local)

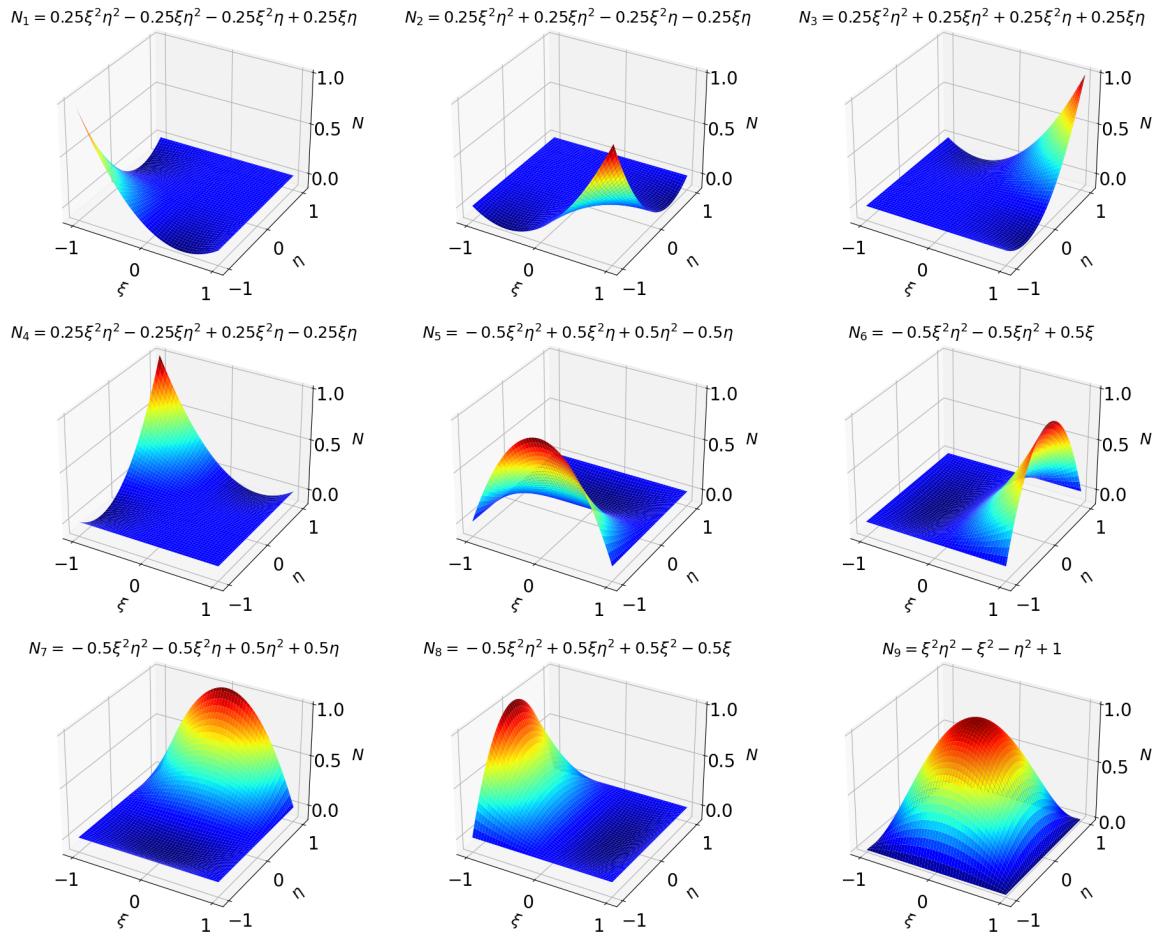


图 1-15 二维九节点参考单元基函数

对于高阶单元其局部基函数构造首先由坐标转换式 (1.52) 得到局部单元在参考坐标系下的表示，随后将所得坐标代入上述参考元基函数即可得到高阶局部单元各节点基函数，这里不再详细演示。

1.6 高斯积分

有限元方法中为了构造求解的线性系统，需要对各单元节点在线性系统中刚度矩阵元素进行计算，而这些元素通常是通过基函数或包含基函数的函数在指定单元区间上的积分得到。积分是一种连续无限空间运算，这对计算机而言是不可实现的，为了得到具有一定精度的结果通常采用数值求积的方法计算积分。常用的数值求积方法^[5]有：矩形、梯形、Simpson、Cotes、复化梯形、复化 Simpson、复化 Cotes 求积公式，但是对于有限积分节点而言这些积分公式所得精度难以满足实际需求。为此，有限元方法中多采

用具有最高代数精度的高斯 (Gauss) 型求积公式。

定义 → Gauss 型求积公式

形如

$$I_n(f) = \sum_{k=0}^n A_k f(x_k) \quad (1.64)$$

的插值型求积公式 (此处并未要求取等距节点) 的代数精度至少为 n 。事实上当 $f(x) \in P_n$ 时, 其 n 次 Lagrange 插值多项式 $p_n(x) = f(x)$, 所以 $I_n(f) = I(f)$ 。然而式 (1.64) 的代数精度必小于 $2n+2$ 。事实上, 选取一个 $2n+2$ 次多项式 $p_{2n+2}(x) = \prod_{j=0}^n (x - x_j)^2$, 则 $\int_a^b \rho(x) p_{2n+2}(x) dx > 0$, 而 $\sum_{k=0}^n A_k p_{2n+2}(x_k) = 0$, 数值积分不等于积分真值, 所以式 (1.64) 的代数精度小于 $2n+2$ 。

如果形如 (1.64) 的求积公式具有代数精度 $2n+1$, 则称其为 Gauss 型求积公式, 并称其中的求积节点 $x_k (k = 0, 1, \dots, n)$ 为 Gauss 点。



1.6.1 1D 高斯积分

(1) 1D 高斯积分点及权重系数

常见的 Gauss 型求积公式有: 1) Gauss-Chebyshev (Mehler) 公式; 2) Gauss-Legendre (Gauss) 公式; 3) Gauss-Hermite 公式; 4) Gauss-Laguerre 公式。在有限元中常见为 Gauss 公式, 在使用 Gauss 公式时, 通常需要知道指定的积分区间、积分点及其对应权重系数, Gauss 公式定义如下:

定义 → Gauss-Legendre 公式 (Gauss 公式)

对形如

$$\int_{-1}^1 f(x) dx$$

的积分, 可以选择 Legendre 多项式 $L_{n+1}(x)$ 的零点作为 Gauss 点。此时

$$\omega_{n+1}(x) = \frac{2^{n+1} [(n+1)!]^2}{(2n+2)!} L_{n+1}(x)$$

求积系数

$$A_k = \frac{2}{(1-x_k^2) [L'_{n+1}(x_k)]^2}, \quad k = 0, 1, \dots, n. \quad (1.65)$$

求积余项

$$\tilde{E}_n(f) = \frac{2^{2n+3} [(n+1)!]^4}{(2n+3) [(2n+2)!]^3} f^{(2n+2)}(\xi), \quad \xi \in [-1, 1]. \quad (1.66)$$



如前所述，使用 Gauss 型求积公式计算给定积分时，需要知道对应的 Gauss 积分点及对应的权重系数，可以选 Legendre 多项式的零点作为 Gauss 点， n 次 Legendre 多项式满足：

例 1.2 设 $\{L_n(x)\}_{n=0}^k$ 是 $[-1, 1]$ 上以 $\rho(x) = 1$ 为权函数的正交多项式族，称 $L_n(x)$ ($n = 0, 1, \dots, n$) 为 n 次 Legendre 多项式。其一般表达式为：

$$L_n(x) = \frac{1}{2^n \cdot n!} \cdot \frac{d^n}{dx^n} [(x^2 - 1)^n], \quad n = 0, 1, 2, \dots.$$

常见的 n 次 Legendre 多项式如下表所示

n	$p_n(x)$
0	1
1	x
2	$\frac{1}{2}(3x^2 - 1)$
3	$\frac{1}{2}(5x^3 - 3x)$
4	$\frac{1}{8}(35x^4 - 30x^2 + 3)$
5	$\frac{1}{8}(63x^5 - 70x^3 + 15x)$
6	$\frac{1}{16}(231x^6 - 315x^4 + 105x^2 - 5)$
7	$\frac{1}{16}(429x^7 - 693x^5 + 315x^3 - 35x)$
8	$\frac{1}{128}(6435x^8 - 12012x^6 + 6930x^4 - 1260x^2 + 35)$
9	$\frac{1}{128}(12155x^9 - 25740x^7 + 18018x^5 - 4620x^3 + 315x)$
10	$\frac{1}{256}(46189x^{10} - 109395x^8 + 90090x^6 - 30030x^4 + 3465x^2 - 63)$

通过计算对应多项式的零点即可得到对应的 Gauss 点，将所得 Gauss 点代入式 (1.65) 即可得到对应权重系数。

(2) 1D 高斯积分点及权重构构造示例

本小节对 1D 情况下 Gauss 积分中 Gauss 积分点及其系数的求解进行举例说明，主要对 1D 空间下 FEM 方法中用到的线性元、二次元对应的两点和三点 Gauss 积分点及对应系数的求解方法进行举例说明。

例 1.3 【2 点 Gauss 积分】 构造在区间 $[-1, 1]$ 上关于权函数 $\rho(x) = 1$ 的两点 Gauss-Legendre 型求积式： $\int_{-1}^1 f(x) dx \approx A_0 f(x_0) + A_1 f(x_1)$ 。其中 x_i 为 Gauss 积分点， A_i 为对应的权重系数。

【解】 首先构造 2 次正交多项式：

$$\mu_0 = \int_{-1}^1 1 dx = 2, \quad \mu_1 = \int_{-1}^1 x dx = 0, \quad \mu_2 = \int_{-1}^1 x^2 dx = \frac{2}{3}, \quad \mu_3 = \int_{-1}^1 x^3 dx = 0 \quad (1.67)$$

$$\phi_2(x) = \begin{vmatrix} \mu_0 & \mu_1 & 1 \\ \mu_1 & \mu_2 & x \\ \mu_2 & \mu_3 & x^2 \end{vmatrix} = \begin{vmatrix} 2 & 0 & 1 \\ 0 & \frac{2}{3} & x \\ \frac{2}{3} & 0 & x^2 \end{vmatrix} = \frac{4}{9} (3x^2 - 1) \quad (1.68)$$

$$\text{令 } \phi_2(x) = \frac{4}{9} (3x^2 - 1) = 0 \rightarrow \begin{cases} x_0 = -\frac{1}{\sqrt{3}} \\ x_1 = \frac{1}{\sqrt{3}} \end{cases}$$

可以得到：

$$\begin{cases} A_0 = \int_1^{-1} l_0(x) dx = \int_1^{-1} \frac{x - \frac{1}{\sqrt{3}}}{-\frac{1}{\sqrt{3}} - \frac{1}{\sqrt{3}}} dx = 1 \\ A_1 = \int_1^{-1} l_1(x) dx = \int_1^{-1} \frac{x + \frac{1}{\sqrt{3}}}{\frac{1}{\sqrt{3}} + \frac{1}{\sqrt{3}}} dx = 1 \end{cases} \quad (1.69)$$

例 1.4 【3 点 Gauss 积分】构造在区间 $[-1, 1]$ 上关于权函数 $\rho(x) = 1$ 的具有 5 次代数精度的 Gauss-Legendre 型求积式： $\int_{-1}^1 f(x) dx \approx A_0 f(x_0) + A_1 f(x_1) + A_2 f(x_2)$ 。其中 x_i 为 Gauss 积分点， A_i 为对应的权重系数。

【解】该求积式的 Gauss 积分点及其权重系数计算方法同 2 点 Gauss 求积式积分点与权重计算方法相同。首先构造 3 次正交多项式：

$$\phi_3(x) = \left(\frac{8}{15} - \frac{8}{27} \right) x^3 + \frac{2}{5} \left(\frac{4}{9} - \frac{4}{5} \right) \quad (1.70)$$

$$\text{令 } \phi_3(x) = \frac{32}{45} \left(\frac{1}{3}x^3 - \frac{1}{5}x \right) = 0 \rightarrow \begin{cases} x_0 = -\sqrt{\frac{3}{5}} \\ x_1 = 0 \\ x_2 = \sqrt{\frac{3}{5}} \end{cases}$$

可以得到：

$$\begin{cases} A_0 = A_2 = \frac{5}{9} \\ A_1 = \frac{8}{9} \end{cases} \quad (1.71)$$

(3) 1D 常用高斯积分表

在实际程序设计时无需构造对应 Gauss 积分点及其对应系数，往往通过查表的方式得到对应 Gauss 积分点及其相应权重系数，在程序中直接对其进行应用。

表 1-1 1D 常用 Gauss 积分表

Number of Points	Accuracy (2n-1)	Location ξ_i	Weight w
1	1	0	2
2	3	$\pm\sqrt{1/3}(\pm 0.57735)$	1
3	5	$\pm\sqrt{3/5}, (\pm 0.774596)$ 0	$\pm 5/9, (0.555556)$ $8/9, (0.888889)$
4	7	$\pm\sqrt{\frac{3}{7} - \frac{2}{7}\sqrt{\frac{6}{5}}}, (\pm 0.339981)$ $\pm\sqrt{\frac{3}{7} + \frac{2}{7}\sqrt{\frac{6}{5}}}, (\pm 0.861136)$	$\frac{18+\sqrt{30}}{36}, (0.652145)$ $\frac{18+\sqrt{30}}{36}, (0.347855)$

(4) 1D 高斯积分示例

从前述 Gauss 积分的定义, Gauss 点及对应权重系数的计算注意到, Gauss 积分点及其对应权重系数在不同积分区间有不同的数值, 那么在实际 FEM 方法中由于不同问题的需要可能需要采用不同的 Gauss 点及权重系数, 这将为计算带来许多不必要的麻烦。所以更为一般的方法是将给定的积分区间通过变换将其用已知区间的积分进行替代, 由此需要将任意区间的 Gauss 积分转换为用已知区间上 Gauss 点及权重系数进行计算。

性质 → 任意区间 Gauss 积分

对于任意区间 $[a, b]$ 上权函数 $\rho(x) = 1$ 的 Gauss 型求积公式, 只需要作变量替换:

$$x = \frac{a+b}{2} + \frac{b-a}{2} \cdot t$$

则有 $x \in [a, b] \leftrightarrow t \in [-1, 1]$, 这样给定区间的积分可以变为:

$$\begin{aligned} \int_a^b f(x) dx &= \frac{b-a}{2} \int_{-1}^1 f\left(\frac{b+a}{2} + \frac{b-a}{2} \cdot t\right) dt \\ &\approx \frac{b-a}{2} \sum_{k=0}^n A_k f\left(\frac{b+a}{2} + \frac{b-a}{2} \cdot t_k\right) \end{aligned}$$

例 1.5 【Gauss 积分实例】 用 Gauss-Legendre 求积公式 ($n=1,2$) 计算积分: $I = \int_0^1 x^2 e^x dx$ 。

【解】 由于区间为 $[0, 1]$, 所以先做变量替换 $x = \frac{1+t}{2}$, 得:

$$I = \int_0^1 x^2 e^x dx = \frac{1}{8} \int_{-1}^1 (t+1)^2 e^{(1+t)/2} dt$$

令 $f(t) = (t+1)^2 e^{(1+t)/2}$, 对于 $n=1$, 由 2 点 Gauss-Legendre 求积公式可得:

$$I \approx \frac{1}{8} \left[f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right) \right] = 0.71194774$$

对于 $n=2$, 由 3 点 Gauss-Legendre 求积公式可得:

$$I \approx \frac{1}{8} \left[\frac{5}{9} f\left(-\frac{\sqrt{15}}{5}\right) + \frac{8}{9} f(0) + \frac{5}{9} f\left(\frac{\sqrt{15}}{5}\right) \right] = 0.718251799$$

此定积分的精确值为 $I = e - 2 = 0.718281828$, 得 $n=1$ 时误差为: 0.0063340054, $n=2$ 时的误差为: 0.000030049。

1.6.2 2D 高斯积分

对于 2D 平面单元的高斯积分点及相关权重系数可查表得到, 这里给出常用的 2D 平面单元的高斯积分点及权重系数表。

(1) 2D 三节点单元高斯积分点及权重系数

采用前述三节点参考单元, 其对应的高斯积分点与响应权重系数如图 1-16 所示:

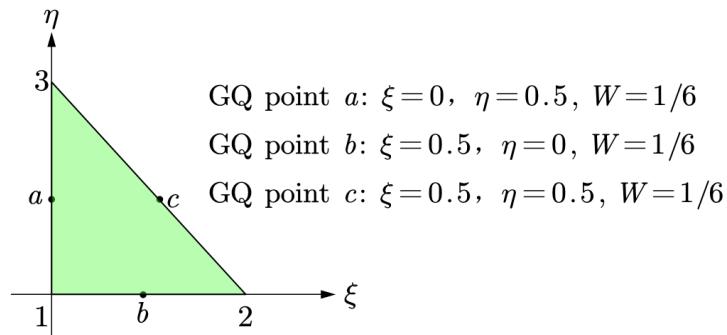


图 1-16 二维三节点参考单元 Gauss 点及权重系数

(2) 2D 四节点单元高斯积分点及权重系数

采用四边形参考单元进行高斯积分对应的高斯积分点及权重系数如图 1-17 所示:

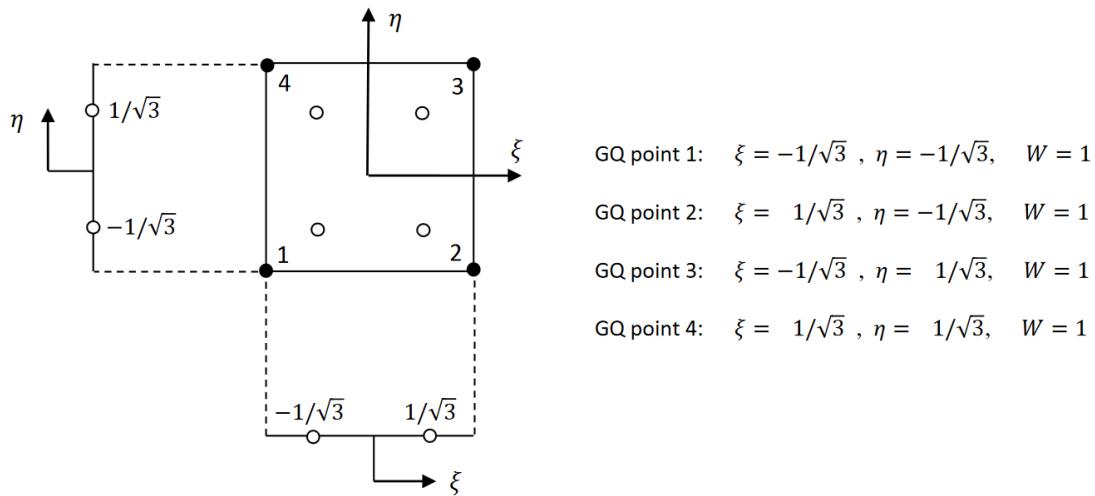


图 1-17 二维四节点参考形单元 Gauss 点及权重系数

(3) 2D 高斯积分示例

2D 高斯积分基本方法与 1D 情况类似，区别仅在于积分函数变为关于 (x, y) 的函数，以此可以猜测对于 3D 的高斯积分方法也具有相同的形式。

$$\int_{\Omega} f(x, y) dx dy = \sum_{i=1}^N w_i f(x_i, y_i) = \begin{bmatrix} w_1 & w_2 & \cdots & w_N \end{bmatrix} \begin{bmatrix} f(x_1, y_1) \\ f(x_2, y_2) \\ \vdots \\ f(x_N, y_N) \end{bmatrix} \quad (1.72)$$

例 1.6 【计算】 $\int_{x=0}^1 \int_{y=0}^{1-x} f(x, y) dx dy = ?$ ，为了说明原理这里取 $f(x, y) = 1$ ，从几何意义可知所得结果为三角形的面积： $S = \frac{1}{2}$ 。

容易得到该积分的数学积分结果为：

$$\int_{x=0}^1 \int_{y=0}^{1-x} 1 dx dy = \int_{\xi=0}^1 y|_0^{1-x} dx = \frac{1}{2} \quad (1.73)$$

Gauss 积分，容易得到该积分域为标准三节点单元，由此可以直接使用对应积分点及权重系数进行计算：

$$\int_{\Omega} f(x, y) dx dy = \sum_{i=1}^3 w_i f(x_i, y_i) = \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{1}{2} \quad (1.74)$$

例 1.7 【计算】 $\int_{x=-1}^1 \int_{y=-1}^1 f(x, y) dx dy = ?$, 同三角形一样取 $f(x, y) = 1$, 从几何意义可知所得结果为四边形的面积: $S = 4$ 。

数学积分解析结果:

$$\int_{x=-1}^1 \int_{y=-1}^1 1 dx dy = \int_{x=-1}^1 y \Big|_{-1}^1 dx = 4 \quad (1.75)$$

Gauss 积分, 与前述问题相似, 本例中积分域为标准四节点单元, 由此可以直接使用对应积分点及权重系数进行计算:

$$\int_{\Omega} f(x, y) dx dy = \sum_{i=1}^4 w_i f(x_i, y_i) = 1 + 1 + 1 + 1 = 4 \quad (1.76)$$

例 1.8 【计算】 $\int_{x=0}^1 \int_{y=0}^1 f(x, y) dx dy = ?$, 取 $f(x, y) = 1$, 从几何意义可知所得结果为四边形的面积: $S = 1$ 。

可以看出给定积分区间不再是标准参考单元积分区间, 所以无法直接使用对应参考单元积分点及权重系数进行计算, 这在 2D FEM 中是更为普遍的情况。采用 Gauss 积分法计算该积分时需要将一般的单元积分变换为参考单元上的积分, 随后采用对应参考单元的积分点及权重系数进行计算。

$$\int_{\Omega(x,y)} f(x, y) dx dy = \int_{\Omega(\xi,\eta)} f(\phi(\xi, \eta)) \left| \frac{\partial(x, y)}{\partial(\xi, \eta)} \right| d\xi d\eta \quad (1.77)$$

其中, $\left| \frac{\partial(x, y)}{\partial(\xi, \eta)} \right|$ 为前述 Jacobian 矩阵的行列式, 该矩阵表示从局部单元变换到参考单元的变换矩阵, 在 2D 情况下其行列式的值表示局部元与参考元的面积比。

将四边形各节点基函数及对应的节点坐标代入式 (1.50) 中得到对应的 Jacobian 矩阵:

$$\begin{aligned} J &= \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \xi} & \frac{\partial N_3}{\partial \xi} & \frac{\partial N_4}{\partial \xi} \\ \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \frac{\partial N_3}{\partial \eta} & \frac{\partial N_4}{\partial \eta} \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix} \\ &= \frac{1}{4} \begin{bmatrix} -(1-\eta) & (1-\eta) & (1+\eta) & -(1+\eta) \\ -(1-\xi) & -(1+\xi) & (1+\xi) & (1-\xi) \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \end{aligned} \quad (1.78)$$

将 Jacobian 矩阵代入积分式即可得到：

$$\begin{aligned} \int_{\Omega(x,y)} f(x,y) dx dy &= \int_{\Omega(\xi,\eta)} f(\phi(\xi,\eta)) |J| d\xi d\eta \\ &= \int_{-1}^1 \int_{-1}^1 0.25 d\xi d\eta = 0.25 \times 4 = 1 \end{aligned} \quad (1.79)$$

1.7 有限元方法的一般流程

有限元方法的基本实现流程如图 1-18 所示，其中网格划分与线性系统求解、后处理部分通常由第三方软件完成，本书主要对不同问题模型、边界条件、刚度矩阵组装与载荷向量组装等有限元关键内容进行介绍，其余部分仅以结果形式展示作为参考。

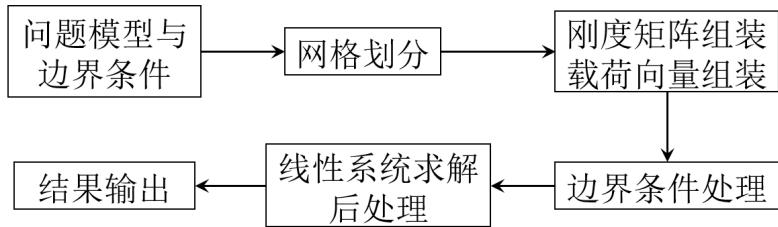


图 1-18 有限元方法基本流程

2

1D 有限元方法

2.1 引言

本章对 1D 空间下 FEM 方法问题模型进行介绍，以该问题模型为例对 FEM 离散格式、线性系统构造、边界条件处理方法等过程进行介绍，针对所提模型实际 FEM 计算的详细过程、方法进行了举例，并将该流程的程序实现及所得结果与解析解进行比较，验证了所述方法的正确性。除此之外对 1D 空间下不同的边界条件类型、程序正确性校验的误差计算方法及更高阶单元类型的构造及使用方法进行了详细说明与正确性校验。

2.2 问题模型

以下述 1D 2 阶椭圆方程^[6-7,9-10] (1D Second Order Elliptic Equation) 的求解为例说明有限元方法具体实现过程，该方程为一偏微分方程 (Partial Differential Equation, PDE)，在有限元方法中待求问题通常称为试探函数 (Trial Function)。

$$\begin{cases} -\frac{d}{dx} \left(c(x) \frac{du(x)}{dx} \right) = f(x), & a < x < b \\ u(a) = g_a, \quad u(b) = g_b \end{cases} \quad (2.1)$$

2.2.1 弱格式

方程 (2.1) 所描述问题为一连续空间 $[a, b]$ 上的问题，采用有限元方法将无限问题转换为有限问题并用计算机进行求解需要构造对应问题的线性系统，即 $\mathbf{K}\mathbf{u} = \mathbf{F}$ 。为此

需要对方程进行一系列有效变换以便构造该问题的线性系统从而使问题得以求解，该变换过程为弱格式推导过程，针对问题 (2.1) 其详细弱格式推导如下。

(1) 方程两边同乘测试函数 (Test Function) $v(x)$

$$-\frac{d}{dx} \left(c(x) \frac{du(x)}{dx} \right) v(x) = f(x) v(x), \quad a < x < b \quad (2.2)$$

测试函数 $v(x)$ 属于一个函数空间 V ，并且要求测试函数在给定区间连续： $v(x) \in C[a, b]$ ， $\frac{dv(x)}{dx}$ 分段连续，并且在给定区间有界： $v(a) = v(b) = 0$ 。

【分段连续】： 测试函数在给定区间内可被分为有限个子区间，在每一个子区间上函数都是连续的。

(2) 两边同时积分

$$\int_a^b -\frac{d}{dx} \left(c(x) \frac{du(x)}{dx} \right) v(x) dx = \int_a^b f(x) v(x) dx \quad (2.3)$$

(3) 积分变换

应用散度定理（1D 情况下为简单分部积分）对域积分进行降阶处理：

$$\begin{aligned} \int_a^b -\frac{d}{dx} \left(c(x) \frac{du(x)}{dx} \right) v(x) dx &= \int_a^b f(x) v(x) dx \\ \longrightarrow -\int_a^b (cu')' v dx &= -\int_a^b v d(cu') = - \left(cu'v \Big|_a^b - \int_a^b cu' dv \right) \\ &= - \left(c(b) u'(b) v(b) - c(a) u'(a) v(a) - \int_a^b cu' v' dx \right) \end{aligned} \quad (2.4)$$

对上式进行变换整理后可以得到：

$$-c(b) u'(b) v(b) + c(a) u'(a) v(a) + \int_a^b cu' v' dx = \int_a^b f(x) v(x) dx \quad (2.5)$$

由上可知，在边界处的函数值通过边界条件给出： $u(a) = g_a, u(b) = g_b$ 。因为测试函数可以随意进行选取，这里令测试函数 $v(x)$ 在边界处为 0，即 $v(a) = v(b) = 0$ ，可以得到所求方程的弱格式（这里 u, v 都属于索伯列夫空间 (Sobolev spaces)）：

$$\int_a^b cu' v' dx = \int_a^b f(x) v(x) dx \quad (2.6)$$

令 $a(u, v) = \int_a^b cu'v' dx$, $(f, v) = \int_a^b fv dx$ 则有:

$$a(u, v) = (f, v) \longleftrightarrow \int_a^b cu'v' dx = \int_a^b f(x)v(x) dx \quad (2.7)$$

定义 → 强解与弱解

满足式 (2.6) 的解 $u(x)$ 称为弱解, 满足式 (2.1) 的解称为强解。强解一定是满足式 (2.6) 的弱解, 反之不一定成立。特别地, 如果 $u(x)$ 为 2 阶导数连续, $u(x) \in C^2(\Omega)$, 则 $u(x)$ 既是强解也是弱解。

所以可以找到一个分片测试函数 $V_h \in V$, 使得在该空间测试函数满足: $\forall v \in V_h$, 并且所求函数也在该空间中: $u_h \in V_h$, 方程 (2.6) 成立。此时得到的空间 V_h 为有限元空间 (Finite Element Space)。

所以, 在得到方程弱格式后需要寻找合适的有限元空间得到 $u(x)$ 。而当前介绍有限元方法就是利用 Lagrange 插值方法, 利用计算域内网格点组成的插值基函数及其导数, 结合 Gauss 积分求解 PDE 问题。



方程 (2.6) 对问题模型中所有计算域都成立, 在进行有限元方法求解时需要将给定计算域空间离散, 该离散空间为有限元空间, 可由第一章中所述对应方法构造得到, 在每一个空间上, 方程 (2.6) 都成立:

$$a(u_h, v_h) = (f, v_h) \longleftrightarrow \int_a^b cu'_h v'_h dx = \int_a^b f(x)v_h(x) dx \quad (2.8)$$

2.2.2 方程离散

连续空间内方程的解可由离散空间各节点解的线性组合近似表示为:

$$u(x) = \sum_{j=1}^{NN} u_j(x_i) \phi_j(x_i) \quad (2.9)$$

其中, $u(x)$ 为计算域空间的解, x_i 为离散后计算域内编号为 i 节点坐标, NN 为离散空间网格节点总数, $u_j(x_i)$ 为离散后函数在节点 x_i 处的值, ϕ_j 为基函数 (形函数)。

利用式 (2.9) 对方程 (2.6) 等号左右两端各项离散, 将测试函数 $v(x)$ 用所得有限元空间基函数替换后可以得到:

$$\begin{aligned} & \int_a^b c \left(\sum_{j=1}^{NN} u_j \phi_j \right)' \phi_i' dx = \int_a^b f \phi_i dx, \quad i = 1, \dots, NN \\ & \rightarrow \sum_{j=1}^{NN} u_j \left[\int_a^b c \phi_j' \phi_i' dx \right] = \int_a^b f \phi_i dx, \quad i = 1, \dots, NN \end{aligned} \quad (2.10)$$

所得式 (2.10) 组成的线性系统即是待求解线性系统，为方便计算域程序设计，定义刚度矩阵和载荷向量分别如下：

$$\mathbf{K} = [K_{ij}]_{i,j=1}^{NN} = \left[\int_a^b c \phi_j' \phi_i' dx \right]_{i,j=1}^{NN} \quad (2.11)$$

$$\mathbf{F} = [b_i]_{i=1}^{NN} = \left[\int_a^b f \phi_i dx \right]_{i=1}^{NN} \quad (2.12)$$

定义未知量向量为：

$$\mathbf{u} = [u_i]_{j=1}^{NN} \quad (2.13)$$

由此可以得到线性系统： $\mathbf{Ku} = \mathbf{F}$ ，其中刚度矩阵 \mathbf{K} 对称、正定、非奇异、对角占优，所以线性系统解唯一。

2.2.3 线性系统构建

在完成方程离散后即可分别计算刚度矩阵、载荷向量，之后将其组装得到待求解的线性系统，下面分别对刚度矩阵、及载荷向量的组装方法进行介绍。

(1) 组装刚度矩阵 \mathbf{K}

由式 (2.11) 定义的刚度矩阵中各个元素可以表示为网格单元各个节点的组合：

$$K_{ij} = \int_a^b c \phi_j' \phi_i' dx = \sum_{n=1}^{NE} \int_{x_n}^{x_{n+1}} c \phi_j' \phi_i' dx, \quad i, j = 1, \dots, NN \quad (2.14)$$

其中， NE 为离散空间单元数，其与空间节点数量满足 $NN = NE + 1$ 。

根据基函数的性质，容易知道对刚度矩阵系数 K_{ij} 产生影响的仅有相邻的节点，即当 $|i - j| > 1$ 时， $K_{ij} = 0$ ，为此仅需要关注刚度矩阵中非 0 的元素求解。在有限元方法中，待求节点处函数值的大小仅受到其相邻节点的影响，其他距离较远的节点影响为 0。

由前述分析可以知道：刚度矩阵非零元素仅存在于某一单元的相邻节点上，因此，在计算刚度矩阵元素时仅需要遍历所有单元，然后求得对应单元节点处的 K_{ij} ，然后对其进行整理即可得到整个系统的刚度矩阵。

对于每个单元而言，组成该单元的节点数量固定（对于 1D 线性元而言每个单元节点数为 2），所以可以首先得到每个单元的局部刚度矩阵，然后根据该节点在全局的编号确定其值在整体刚度阵中的位置，实际有限元方法程序设计的一般做法也是先计算每个单元的局部刚度矩阵，然后将其整理到全局刚度矩阵中。刚度矩阵构造过程如算法 1 所示。

Algorithm 1 1D FEM 刚度矩阵组装

Require: $\mathbf{K} = \text{sparse}(N_b^{test}, N_b^{trial})$

- 1: **for** $n = 1, \dots, NE$ **do**
- 2: **for** $\alpha = 1, \dots, N_{lb}^{trial}$ **do**
- 3: **for** $\beta = 1, \dots, N_{lb}^{test}$ **do**
- 4: Compute $r = \int_{E_n} c \varphi_{n\alpha}^r \varphi_{n\beta}^s dx$
- 5: $\mathbf{K}(T_b^{test}(\beta, n), T_b^{trial}(\alpha, n)) += r$
- 6: **end for**
- 7: **end for**
- 8: **end for**

其中, N_b 为有限元离散空间节点数量, N_{lb} 为局部单元基函数 (局部单元节点) 数量, NE 为有限元离散空间单元数, E_n 为局部单元 n 的积分区域, T_b 为离散空间各单元的全局编号信息, 根据该矩阵可以找到对应局部单元刚度矩阵在全局刚度矩阵中的位置。

(2) 构造载荷向量 \mathbf{F}

由式 (2.12) 定义的载荷向量中各个元素可以表示为网格单元各个节点的组合:

$$b_i = \int_a^b f \phi_i dx = \sum_{n=1}^{NE} \int_{x_n}^{x_{n+1}} f \phi_i dx, \quad i, j = 1, \dots, NN \quad (2.15)$$

根据基函数性质, 系数 b_i 在当 $|i - j| > 1$ 时, $b_i = 0$, 所以仅需要关注刚度矩阵中非 0 元素求解。求解过程与刚度矩阵系数计算方法类似, 载荷向量阵组装方法如算法 2 所示。

Algorithm 2 1D FEM 载荷向量组装

Require: $\mathbf{F} = \text{sparse}(N_b, 1)$

- 1: **for** $n = 1, \dots, NE$ **do**
- 2: **for** $\beta = 1, \dots, N_{lb}$ **do**
- 3: Compute $r = \int_{E_n} f \varphi_{n\beta}^s dx$
- 4: $\mathbf{F}(T_b^{test}(\beta, n), 1) += r$
- 5: **end for**
- 6: **end for**

2.3 不同边界条件处理

2.3.1 第 1 类边界条件

第 1 类边界条件^[4,7,11] (Essential Boundary Condition, EBC), 也称为: Dirichlet BC。该边界条件直接给出了待求函数在边界处的值, 无需进行求解, 其表示如下:

$$T_{\Gamma} = T_0 \quad (2.16)$$

针对这类型边界条件需要将已经获得的刚度矩阵、载荷向量进行相应调整。对刚度矩阵而言, 若给定节点函数值通过边界条件给定, 则将其对应刚度矩阵系数“置 1”即可, 对应载荷向量则为边界条件给定值。在对刚度矩阵进行“置 1”操作时有 2 种方式:

1. 为方便编程操作, 可以将其所在行除了对应边界节点处系数“置 1”外, 其余系数“置 0”, 此时对应载荷向量仅需要改变对应边界节点;
2. 为了使得边界处理后矩阵仍然保持对称、正定、非奇异, 也可将边界点所在位置处“置 1”, 将其所在的行、列的其他系数均“置 0”, 此时除了该节点外所有其他节点需要对刚度矩阵修正引起的变化进行修正。

上述两种边界处理方法各有优劣: 方法 1 在程序实现时较为方便, 但是在边界处理后可能会导致矩阵不再对称, 导致线性系统求解速度有所下降; 方法 2 在边界处理后矩阵对称、正定、非奇异, 线性系统求解效率高, 但是该方法边界处理相对较为繁琐。方法 1 对应边界处理如算法 3 所示。

Algorithm 3 1D FEM Dirichlet 边界处理

```

1: for  $k = 1, \dots, N_{BN}$  do
2:   if  $BC[0, k] ==$  Dirichlet then
3:      $i = BC[1, k]$ 
4:      $\mathbf{K}[i, :] = 0$ 
5:      $\mathbf{K}[i, i] = 1$ 
6:      $\mathbf{F}[i] = g(Pb[i])$ 
7:   end if
8: end for
```

其中, N_{BN} 为边界节点数, BC 为边界条件信息矩阵, 第一行为边界条件类型, 第二行为边界节点全局编号, $g(x)$ 为边界条件函数, Pb 为有限元空间节点坐标矩阵。

第一类边界条件为最简单的边界条件类型, 可以此为基础进行其他边界条件处理。

2.3.2 第2类边界条件

第2类边界条件为自然边界条件^[4,7,11] (Natural Boundary Condition, NBC) 也称为: Neumann BC。具有重要的物理意义, 如在热传导问题中通过边界的热通量, 其具有如下形式:

$$k \frac{dT}{dx} n_x = q_0 \quad (2.17)$$

(1) 左边: Neumann, 右边: Dirichlet

以问题(2.1)为例, 假设其左边界为 Neumann 边界类型, 右边界为 Dirichlet 边界:

$$\begin{cases} -\frac{d}{dx} \left(c(x) \frac{du(x)}{dx} \right) = f(x), & a < x < b \\ u'(a) = r_a, \quad u(b) = g_b \end{cases} \quad (2.18)$$

由前述推导的弱格式可得:

$$-c(b)u'(b)v(b) + c(a)u'(a)v(a) + \int_a^b cu'v'dx = \int_a^b f(x)v(x)dx \quad (2.19)$$

由问题可知, 其左边界为 Neumann 边界, 右边界已经给出: $u(b) = g_b$, 因此, 可以选择合适的测试函数 $v(x)$ 满足 $v(b) = 0$, 由此可以得到仅包含左界的弱格式:

$$\begin{aligned} & r_a c(a) v(a) + \int_a^b cu'v'dx = \int_a^b f(x)v(x)dx \\ \rightarrow & \int_a^b cu'v'dx = \int_a^b f(x)v(x)dx - r_a c(a) v(a) \end{aligned} \quad (2.20)$$

所以, 在进行边界处理时仅需要将左边界 ($x = a$) 处载荷向量增加 $-r_a c(a)$ 即可。

(2) 左边: Dirichlet, 右边: Neumann

以问题(2.1)为例, 假设其左边界为 Dirichlet 边界类型, 右边界为 Neumann 边界:

$$\begin{cases} -\frac{d}{dx} \left(c(x) \frac{du(x)}{dx} \right) = f(x), & a < x < b \\ u(a) = g_a, \quad u'(b) = r_b \end{cases} \quad (2.21)$$

由前述推导的弱格式可得:

$$-c(b)u'(b)v(b) + c(a)u'(a)v(a) + \int_a^b cu'v'dx = \int_a^b f(x)v(x)dx \quad (2.22)$$

由问题可知，其左边界已经给出： $u(a) = g_a$ ，右边界为 Neumann 边界，因此可以选择合适的测试函数 $v(x)$ 满足 $v(b) = 0$ ，由此可以得到仅包含右界的弱格式：

$$\begin{aligned} -r_b c(b) v(b) + \int_a^b c u' v' dx &= \int_a^b f(x) v(x) dx \\ \rightarrow \int_a^b c u' v' dx &= \int_a^b f(x) v(x) dx + r_b c(b) v(b) \end{aligned} \quad (2.23)$$

所以，在进行边界处理时仅需要将右边界 ($x = b$) 处载荷向量增加 $r_b c(b)$ 即可。

(3) 左右：Neumann，解不唯一

以问题 (2.1) 为例，假设其左、右边界均为 Neumann 边界：

$$\begin{cases} -\frac{d}{dx} \left(c(x) \frac{du(x)}{dx} \right) = f(x), & a < x < b \\ u'(a) = r_a, \quad u'(b) = r_b \end{cases} \quad (2.24)$$

由前述推导的弱格式可得：

$$-c(b) u'(b) v(b) + c(a) u'(a) v(a) + \int_a^b c u' v' dx = \int_a^b f(x) v(x) dx \quad (2.25)$$

由上式可以看出：当左、右边界均为 Neumann 边界时，可能找不到合适的测试函数 $v(x)$ 得到唯一的边界值：

$$-r_b c(b) v(b) + r_a c(a) v(a) + \int_a^b c u' v' dx = \int_a^b f(x) v(x) dx \quad (2.26)$$

2.3.3 第 3 类边界条件

第 3 类边界条件^[4,7,11]，也称为：Robin BC，通常为第 1、2 类边界条件组合，其具有如下形式：

$$k \frac{dT}{dx} n_x + \alpha T = \beta \quad (2.27)$$

(1) 左边：Robin，右边：Dirichlet

以问题 (2.1) 为例，假设其左边界为 Robin 边界类型，右边界为 Dirichlet 边界：

$$\begin{cases} -\frac{d}{dx} \left(c(x) \frac{du(x)}{dx} \right) = f(x), & a < x < b \\ u'(a) + q_a u(a) = p_a, \quad u(b) = g_b \end{cases} \quad (2.28)$$

由前述推导的弱格式可得：

$$-c(b)u'(b)v(b) + c(a)u'(a)v(a) + \int_a^b cu'v'dx = \int_a^b f(x)v(x)dx \quad (2.29)$$

将左边界代入可以得到：

$$-c(b)u'(b)v(b) + c(a)[p_a - q_a u(a)]v(a) + \int_a^b cu'v'dx = \int_a^b f v dx \quad (2.30)$$

由问题可知，其左边界为 Robin 边界，右边界已经给出： $u(b) = g_b$ ，因此可以选择合适的测试函数 $v(x)$ 满足 $\begin{cases} v(a) = 1 \\ v(b) = 0 \end{cases}$ ，则有：

$$\begin{aligned} & c(a)[p_a - q_a u(a)]v(a) + \int_a^b cu'v'dx = \int_a^b f v dx \\ \longrightarrow & -q_a c(a)u(a)v(a) + \int_a^b cu'v'dx = \int_a^b f v dx - p_a c(a)v(a) \end{aligned} \quad (2.31)$$

因此只需要将 $-q_a c(a)$ 加入边界 a 所在的刚度阵，将 $-p_a c(a)$ 加入边界 a 所在向量。

(2) 左边：Neumann，右边：Robin

以问题 (2.1) 为例，假设其左边界为 Neumann 边界类型，右边界为 Robin 边界：

$$\begin{cases} -\frac{d}{dx} \left(c(x) \frac{du(x)}{dx} \right) = f(x), & a < x < b \\ u'(a) = r_a, \quad u'(b) + q_b u(b) = p_b \end{cases} \quad (2.32)$$

由前述推导的弱格式可得：

$$-c(b)u'(b)v(b) + c(a)u'(a)v(a) + \int_a^b cu'v'dx = \int_a^b f(x)v(x)dx \quad (2.33)$$

将边界条件代入可以得到：

$$\begin{aligned} & -[p_b - q_b u(b)]c(b)v(b) + r_a c(a)v(a) + \int_a^b cu'v'dx = \int_a^b f v dx \\ \longrightarrow & q_b c(b)u(b)v(b) + \int_a^b cu'v'dx = \int_a^b f v dx - r_a c(a)v(a) + p_b c(b)v(b) \end{aligned} \quad (2.34)$$

因此只需要将 $q_b c(b)$ 加入边界 b 所在的刚度阵，将 $-r_a c(a)$ 与 $p_b c(b)$ 分别加入边界 a、b 所在向量。

2.4 误差计算

2.4.1 最大节点误差

最大节点误差即各个节点理论值与数值解差值的最大值：

$$\maxNodeError = \max [res(i) - exact(i)] \quad (2.35)$$

2.4.2 无穷范数误差 L^∞

(1) 无穷范与无穷范误差定义

定义 L^∞ 空间^[9]为：

$$L^\infty(I) = \left\{ v : I \rightarrow R : \sup_{x \in I} |v(x)| < \infty \right\} \quad (2.36)$$

其中， $I \in (a, b)$ 。

无穷范 L^∞ 定义为：

$$\|u\|_\infty = \sup_{x \in I} |u(x)|, \quad u \in L^\infty(I) \quad (2.37)$$

类比无穷范数，无穷范误差定义为：

$$\|u - u_h\|_\infty = \sup_{x \in I} |u(x) - u_h(x)| \quad (2.38)$$

(2) 无穷范误差计算方法

将 $u_h = \sum_{j=1}^{NN} u_j \phi_j$ ，代入无穷范数，利用网格单元节点编号矩阵与对应局部基函数即可得到：

$$\begin{aligned} \|u - u_h\|_\infty &= \sup_{x \in I} |u(x) - u_h(x)| = \max_{1 \leq n \leq N} \max_{Ex_n \leq x \leq x_{n+1}} |u(x) - u_h(x)| \\ &= \max_{1 \leq n \leq N} \max_{Ex_n \leq x \leq x_{n+1}} \left| u(x) - \sum_{j=1}^{NN} u_j \phi_j \right| \\ &= \max_{1 \leq n \leq N} \max_{Ex_n \leq x \leq x_{n+1}} \left| u(x) - \sum_{k=1}^{NN_{local}} u_{T_b(k,n)} \phi_{nk}(x) \right| \\ &= \max_{1 \leq n \leq N} \max_{Ex_n \leq x \leq x_{n+1}} |u(x) - w_n(x)| \end{aligned} \quad (2.39)$$

其中， $w_n(x) = \sum_{k=1}^{NN_{local}} u_{T_b(k,n)} \phi_{nk}(x)$ 。

无穷范数误差计算方法如算法 4 所示：首先得到所有单元上最大误差，然后取最大值即可。

Algorithm 4 1D FEM 无穷范数误差计算

Require: $error = 0$

- 1: **for** $n = 1, \dots, N$ **do**
- 2: Compute $r \approx \max_{x_n \leq x \leq x_{n+1}} |u(x) - w_n(x)|$
- 3: **end for**
- 4: $error = \max_{1 \leq n \leq N} r_n$

2.4.3 L^2 范数误差**(1) L^2 范与 L^2 范误差定义**

L^2 范^[9]定义为：

$$\|u\|_0 = \sqrt{\int_I u^2 dx}, \quad u \in L^\infty(I) \quad (2.40)$$

类比 L^2 范， L^2 范误差定义为：

$$\|u - u_h\|_0 = \sqrt{\int_I (u - u_h)^2 dx} \quad (2.41)$$

(2) L^2 范误差计算方法

将 $u_h = \sum_{j=1}^{NN} u_j \phi_j$ ，代入无穷范数，利用网格单元节点编号矩阵与对应局部基函数即可得到：

$$\begin{aligned} \|u - u_h\|_0 &= \sqrt{\int_I (u - u_h)^2 dx} = \sqrt{\sum_{n=1}^{NE} \int_{x_n}^{x_{n+1}} (u - u_h)^2 dx} \\ &= \sqrt{\sum_{n=1}^{NE} \int_{x_n}^{x_{n+1}} \left(u - \sum_{j=1}^{NN} u_j \phi_j \right)^2 dx} \\ &= \sqrt{\sum_{n=1}^{NE} \int_{x_n}^{x_{n+1}} \left(u - \sum_{k=1}^{NN_{local}} u_{T_b(k,n)} \psi_{nk}(x) \right)^2 dx} \end{aligned} \quad (2.42)$$

2.4.4 H^1 范数误差

(1) H^1 范与 H^1 范误差定义

H^1 范^[9]定义为：

$$\|u\|_1 = \sqrt{\int_I u'^2 dx}, \quad u \in H^1(I) \quad (2.43)$$

类比 H^1 范, H^1 范误差定义为：

$$\|u - u_h\|_1 = \sqrt{\int_I (u' - u'_h)^2 dx} \quad (2.44)$$

(2) H^1 范误差计算方法

将 $u_h = \sum_{j=1}^{NN} u_j \phi_j$, 代入无穷范数, 利用网格单元节点编号矩阵与对应局部基函数即可得到：

$$\begin{aligned} \|u - u_h\|_1 &= \sqrt{\int_I (u' - u'_h)^2 dx} = \sqrt{\sum_{n=1}^{NE} \int_{x_n}^{x_{n+1}} (u' - u'_h)^2 dx} \\ &= \sqrt{\sum_{n=1}^{NE} \int_{x_n}^{x_{n+1}} \left(u' - \sum_{j=1}^{NN} u_j \phi'_j \right)^2 dx} \\ &= \sqrt{\sum_{n=1}^{NE} \int_{x_n}^{x_{n+1}} \left(u' - \sum_{k=1}^{NN_{local}} u_{T_b(k,n)} \psi'_{nk} \right)^2 dx} \end{aligned} \quad (2.45)$$

性质 → L^2 范与 H^1 范的特点

观察发现 L^2 范和 H^1 范具有相同特点：即其值分别为待求函数解析解与数值解的 0 阶导数与 1 阶导数的差。因此可以定义 2 者误差范数的统一形式如下：

$$\sqrt{\sum_{n=1}^{NE} \int_{x_n}^{x_{n+1}} \left(u^{(s)} - \sum_{k=1}^{NN_{local}} u_{T_b(k,n)} \psi_{nk}^{(s)} \right)^2 dx} \quad (2.46)$$

为方便表示可以进一步简化如下：

$$\sqrt{\sum_{n=1}^{NE} \int_{x_n}^{x_{n+1}} (u^{(s)} - w_{n,s})^2 dx}, \quad w_{n,s} = \sum_{k=1}^{NN_{local}} u_{T_b(k,n)} \psi_{nk}^{(s)} \quad (2.47)$$

可以看到当 $s = 0$ 时, 所得结果为 L^2 范数误差；当 $s = 1$ 时为 H^1 范数误差。

L^2 或者 H^1 范数误差计算方法如算法 5 所示。

Algorithm 5 1D FEM L^2 或 H^1 范数误差计算

Require: $error = 0, s$

- 1: **for** $n = 1, \dots, N$ **do**
- 2: Compute $error+ = \int_{x_n}^{x_{n+1}} \left(u^{(s)} - \sum_{k=1}^{NN_{local}} u_{T_b(k,n)} \psi_{nk}^{(s)} \right)^2 dx$
- 3: **end for**
- 4: $error = \sqrt{error}$

2.5 应用实例及程序实现

2.5.1 Example 1

例 2.1 【问题描述】使用 1D 有限元方法求解下述问题：

$$\begin{cases} -\frac{d}{dx} \left(e^x \frac{du(x)}{dx} \right) = -e^x [\cos x - 2 \sin x - x \cos x - x \sin x], & (0 \leq x \leq 1) \\ u(0) = 0, u(1) = \cos(1) \end{cases}$$

【说明】上述问题的解析解为： $u = x \cos x$ ，该解可以用来对 FEM 方法求解的结果误差进行估计。

(1) 线性元求解

1、网格划分

针对本问题，为了方便描述 FEM 实现过程中各个步骤及其实现，这里划分简单的网格进行示意，具体网格相关的各个参数的含义如下：

- 计算域起点： $xStart = 0$ ；计算域终点： $xEnd = 1$ ；
- 网格单元数： $NE = 4$ ；网格节点数： $NN = NE + 1 = 5$ ；
- 网格单元大小： $h = \frac{xEnd - xStart}{NE} = \frac{1}{4} = 0.25$
- 几何网格节点坐标： $P = \begin{bmatrix} 0 & 0.25 & 0.5 & 0.75 & 1 \end{bmatrix}$ ，该矩阵列数为单元节点数量；
- 几何模型网格单元节点编号矩阵： $T = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \end{bmatrix}$ ，该矩阵列数为单元数；
- 有限元空间节点坐标矩阵： $Pb = P$ ；
- 有限元空间单元节点编号矩阵： $Tb = T$ ；
- 边界信息矩阵： $BC = \begin{bmatrix} -1 & -1 \\ 1 & 5 \end{bmatrix}$ ，第 1 行为边界类型，这里 -1 表示 Dirichlet 边界；第 2 行为边界节点编号。

根据给定各参数划分网格示意如图 2-1 所示：

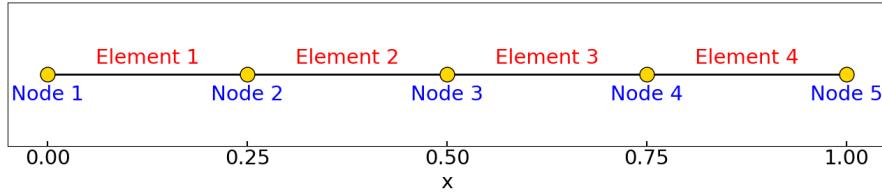


图 2-1 Dirichlet Example 线性元网格划分

2、刚度矩阵组装

生成有限元网格后需要根据对应网格计算相应的刚度矩阵，该矩阵的规模与有限元网格节点数量相关，为： $NN \times NN = 5 \times 5 = 25$ 。

$$\mathbf{K} = \begin{bmatrix} K_{11} & K_{12} & K_{13} & K_{14} & K_{15} \\ K_{21} & K_{22} & K_{23} & K_{24} & K_{25} \\ K_{31} & K_{32} & K_{33} & K_{34} & K_{35} \\ K_{41} & K_{42} & K_{43} & K_{44} & K_{45} \\ K_{51} & K_{52} & K_{53} & K_{54} & K_{55} \end{bmatrix} = \begin{bmatrix} K_{11}^1 & K_{12}^1 & 0 & 0 & 0 \\ K_{21}^1 & K_{22}^1 + K_{11}^2 & K_{12}^2 & 0 & 0 \\ 0 & K_{21}^2 & K_{22}^2 + K_{11}^3 & K_{12}^3 & 0 \\ 0 & 0 & K_{21}^3 & K_{22}^3 + K_{11}^4 & K_{12}^4 \\ 0 & 0 & 0 & K_{21}^4 & K_{22}^4 \end{bmatrix} \quad (2.48)$$

其中， K_{ij}^n 表示第 n 个单元，节点基函数 i, j 对应元素。

由前文可知，该刚度阵为稀疏矩阵，矩阵中各非 0 元素仅由某一单元或其相邻单元上的节点产生，如单元节点 i, j ，如果二者距离超过一个单元： $|i - j| > 1$ 则两者节点可以认为相互不产生影响，故其刚度矩阵系数为 0，该特性也可由基函数特点得到。所以，在进行刚度矩阵求解时仅需要遍历单元对单元刚度矩阵求解后合并各单元刚度阵即可得到总体刚度阵。

- 第 1 个单元刚度阵计算

单元为线性单元，单元节点个数为 2，故局部刚度阵为 2×2 ：

$$K_{2 \times 2}^1 = \begin{bmatrix} K_{11}^1 & K_{12}^1 \\ K_{21}^1 & K_{22}^1 \end{bmatrix} \quad (2.49)$$

组成该单元的节点通过矩阵 T 第 1 列得到，之后所得节点通过矩阵 P 得到节点坐标，将所得节点坐标、对应单元各节点基函数代入式 (2.14) 中，即可得到该单元刚度矩阵各元素。

根据存储的几何信息模型信息矩阵 P 、 T 得到局部单元积分上、下区间：

$$low = P(T(1, 1)) = 0, \quad up = P(T(2, 1)) = 0.25 \quad (2.50)$$

将对应积分区间、对应节点基函数代入式(2.14)中，即可得到局部刚度阵各个元素：

$$\begin{aligned} K_{11}^1 &= \int_{low}^{up} c\phi_j' \phi_i' dx = \int_0^{0.25} e^x \left(\frac{0.25-x}{h}\right)' \left(\frac{0.25-x}{h}\right)' dx = 4.5444 \\ K_{12}^1 = K_{21}^1 &= \int_{low}^{up} c\phi_j' \phi_i' dx = \int_0^{0.25} e^x \left(\frac{0.25-x}{h}\right)' \left(\frac{x}{h}\right)' dx = -4.5444 \\ K_{22}^1 &= \int_{low}^{up} c\phi_j' \phi_i' dx = \int_0^{0.25} e^x \left(\frac{x}{h}\right)' \left(\frac{x}{h}\right)' dx = 4.5444 \end{aligned} \quad (2.51)$$

由此得到单元 1 的局部刚度矩阵：

$$K_{2 \times 2}^1 = \begin{bmatrix} 4.5444 & -4.5444 \\ -4.5444 & 4.5444 \end{bmatrix} \quad (2.52)$$

将得到的 $K_{2 \times 2}^1$ 添加至刚度阵 \mathbf{K} 中：

$$\mathbf{K} + K_{2 \times 2}^1 = \begin{bmatrix} K_{11}^1 & K_{12}^1 & 0 & 0 & 0 \\ K_{21}^1 & K_{22}^1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 4.5444 & -4.5444 & 0 & 0 & 0 \\ -4.5444 & 4.5444 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (2.53)$$

• 第 2 个单元刚度矩阵计算

单元为线性单元，单元节点个数为 2，故局部刚度阵为 2×2 ：

$$K_{2 \times 2}^2 = \begin{bmatrix} K_{11}^2 & K_{12}^2 \\ K_{21}^2 & K_{22}^2 \end{bmatrix} \quad (2.54)$$

积分上、下限

$$low = P(T(1, 2)) = 0.25, \quad up = P(T(2, 2)) = 0.5 \quad (2.55)$$

单元刚度阵各元素计算：

$$\begin{aligned} K_{11}^2 &= \int_{low}^{up} c\phi_j' \phi_i' dx = \int_{0.25}^{0.5} e^x \left(\frac{0.25-x}{h}\right)' \left(\frac{0.25-x}{h}\right)' dx = 5.8351 \\ K_{12}^2 = K_{21}^2 &= \int_{low}^{up} c\phi_j' \phi_i' dx = \int_{0.25}^{0.5} e^x \left(\frac{0.25-x}{h}\right)' \left(\frac{x}{h}\right)' dx = -5.8351 \\ K_{22}^2 &= \int_{low}^{up} c\phi_j' \phi_i' dx = \int_{0.25}^{0.5} e^x \left(\frac{x}{h}\right)' \left(\frac{x}{h}\right)' dx = 16 \int_{0.25}^{0.5} e^x dx = 5.8351 \end{aligned} \quad (2.56)$$

由此得到单元 2 的局部刚度矩阵:

$$K_{2 \times 2}^2 = \begin{bmatrix} 5.8351 & -5.8351 \\ -5.8351 & 5.8351 \end{bmatrix} \quad (2.57)$$

将得到的 $K_{2 \times 2}^2$ 添加至刚度阵 \mathbf{K} 中:

$$\mathbf{K} = \begin{bmatrix} K_{11}^1 & K_{12}^1 & 0 & 0 & 0 \\ K_{21}^1 & K_{22}^1 + K_{11}^2 & K_{12}^2 & 0 & 0 \\ 0 & K_{21}^2 & K_{22}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 4.5444 & -4.5444 & 0 & 0 & 0 \\ -4.5444 & 10.3795 & -5.8351 & 0 & 0 \\ 0 & -5.8351 & 5.8351 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (2.58)$$

- 同理可以获得其他单元刚度阵，最终组装为总体刚度阵 \mathbf{K}

$$\mathbf{K} = \begin{bmatrix} 4.5444 & -4.5444 & 0 & 0 & 0 \\ -4.5444 & 10.3795 & -5.8351 & 0 & 0 \\ 0 & -5.8351 & 13.3276 & -7.4925 & 0 \\ 0 & 0 & -7.4925 & 17.1130 & -9.6205 \\ 0 & 0 & 0 & -9.6205 & 9.6205 \end{bmatrix} \quad (2.59)$$

3、载荷向量组装

载荷向量与矩阵相对应，其规模为有限元网格节点数量，对于本例而言为: $NN = 5$ 。

$$\mathbf{F} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \end{bmatrix} = \begin{bmatrix} F_1^1 \\ F_2^1 + F_1^2 \\ F_2^2 + F_1^3 \\ F_2^3 + F_1^4 \\ F_2^4 \end{bmatrix} \quad (2.60)$$

- 第 1 个单元载荷向量计算

$$F_{2 \times 1}^1 = \begin{bmatrix} F_1^1 \\ F_2^1 \end{bmatrix} \quad (2.61)$$

积分上、下限

$$low = P(T(1, 1)) = 0, \quad up = P(T(2, 1)) = 0.25 \quad (2.62)$$

局部载荷向量各个元素计算:

$$\begin{aligned} F_1^1 &= \int_{low}^{up} f \phi_1 dx = \int_0^{0.25} -e^x [\cos x - 2 \sin x - x \cos x - x \sin x] \frac{0.25 - x}{h} dx \\ &= -0.0986 \end{aligned}$$

$$F_2^1 = \int_{low}^{up} f \phi_2 dx = \int_0^{0.25} -e^x [\cos x - 2 \sin x - x \cos x - x \sin x] \frac{x}{h} dx = -0.0661 \quad (2.63)$$

由此得到单元 1 的局部向量:

$$\mathbf{F}_{2 \times 1}^1 = \begin{bmatrix} F_1^1 \\ F_2^1 \end{bmatrix} = \begin{bmatrix} -0.0986 \\ -0.0661 \end{bmatrix} \quad (2.64)$$

将得到的 $F_{2 \times 1}^1$ 添加至向量 \mathbf{F} 中:

$$\mathbf{F}+ = \mathbf{F}_{2 \times 1}^1 = \begin{bmatrix} F_1^1 \\ F_2^1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.0986 \\ -0.0661 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (2.65)$$

• 第 2 个单元载荷向量计算

积分上、下限:

$$low = P(T(1, 2)) = 0.25, \quad up = P(T(2, 2)) = 0.5 \quad (2.66)$$

局部载荷向量各个元素计算:

$$\begin{aligned} F_1^2 &= \int_{low}^{up} f \phi_0 dx = \int_{0.25}^{0.5} -e^x [\cos x - 2 \sin x - x \cos x - x \sin x] \frac{0.5 - x}{h} dx \\ &= 0.0259 \end{aligned}$$

$$\begin{aligned} F_2^2 &= \int_{low}^{up} f \phi_1 dx = \int_{0.25}^{0.5} -e^x [\cos x - 2 \sin x - x \cos x - x \sin x] \frac{x - 0.25}{h} dx \\ &= 0.0871 \end{aligned} \quad (2.67)$$

由此得到单元 2 的局部向量:

$$\mathbf{F}_{2 \times 1}^2 = \begin{bmatrix} F_1^2 \\ F_2^2 \end{bmatrix} = \begin{bmatrix} 0.0259 \\ 0.0871 \end{bmatrix} \quad (2.68)$$

将得到的 $F_{2 \times 1}^2$ 添加至向量 \mathbf{F} 中:

$$\mathbf{F}+ = \mathbf{F}_{2 \times 1}^2 = \begin{bmatrix} F_1^1 \\ F_2^1 + \mathbf{F}_1^2 \\ \mathbf{F}_2^2 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.0986 \\ -0.0402 \\ 0.0871 \\ 0 \\ 0 \end{bmatrix} \quad (2.69)$$

- 同理可以获得其他单元载荷向量，最终组装为总体载荷向量 \mathbf{F}

$$\mathbf{F} = \begin{bmatrix} -0.0986 & -0.0402 & 0.3311 & 0.9153 & 0.7111 \end{bmatrix}^T \quad (2.70)$$

3、边界条件处理及线性系统求解

在构造完成刚度矩阵与载荷向量之后，需要对已经获得的总体刚度矩阵与总体载荷向量进行处理，以得到最终需要计算的线性系统。

根据不同类型的边界条件在处理所得总体刚度矩阵与总体载荷向量时采取的策略也有所区别，本算例中边界条件均为 Dirichlet 边界，为最简单的边界条件类型。根据该边界条件类型的定义，可以知道在给定边界处所求控制方程中待求函数值已知，线性系统的存在是为了求解未知函数值，既然函数值在边界处已知，则无需再通过线性系统进行求解，所以可以采取的边界条件处理方法有：

- 1) 将边界对应方程从线性系统中去掉，使线性系统本身由未知函数组成，由于各个节点满足的线性方程之间都有相互关联，所以在剩下的未知函数方程的右端项中也需要去掉由于边界条件方程去掉产生的影响。
- 2) 由于边界处函数值已知，那么在线性系统中无需对其进行求解，这意味着该边界处对应的方程可以直接简化为 $u_i = \text{constant}$ ，因此可以将边界对应的刚度矩阵系数直接变为 1，由于该处函数值不受其他节点函数值影响，所以将其他节点处刚度矩阵系数变为 0，该处函数值已知，所以对应载荷向量为给定的边界值，该方法为“置 1”法。

本例采用第 2 种边界处理方法，具体程序中实现过程为：遍历所有边界节点并将边界节点对应的刚度矩阵系数“置 1”，同行其余系数为 0，同时对应节点载荷向量为给定边界值大小。经过边界处理后得到对应刚度矩阵与载荷向量，得到待求线性系统，采用合适的矩阵求解算法即可得到所求问题的解：

$$\mathbf{K}\mathbf{u} = \mathbf{F} \rightarrow \mathbf{u} = \mathbf{FK}^{-1}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -4.5444 & 10.3795 & -5.8351 & 0 & 0 \\ 0 & -5.8351 & 13.3276 & -7.4925 & 0 \\ 0 & 0 & -7.4925 & 17.1130 & -9.6205 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} 0 \\ -0.0402 \\ 0.3311 \\ 0.9153 \\ 0.5403 \end{bmatrix} \rightarrow \mathbf{u} = \begin{bmatrix} 0.0000 \\ 0.2441 \\ 0.4411 \\ 0.5504 \\ 0.5403 \end{bmatrix} \quad (2.71)$$

4、后处理

如图 2-2 为本算例仿真结果与解析解的对比及相关误差信息。可以看到采用有限元方法计算的节点值都在解析解所在曲线上，说明 FEM 方法所得结果是正确的，由于单

元数量较少，所得结果构成的曲线与解析解尚有一定差距，可以通过增加单元数量减少仿真与解析解的误差。

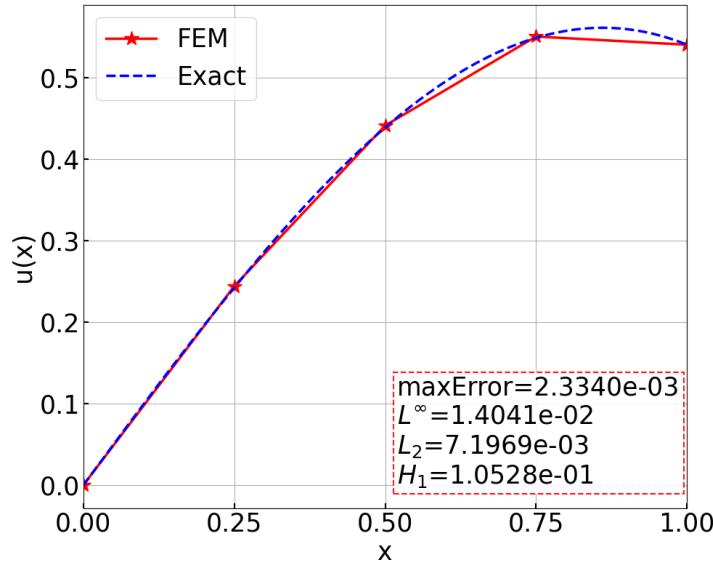


图 2–2 Dirichlet Example 线性元 FEM 仿真结果、解析解与误差

表 2–1 给出了不同单元大小时线性元 FEM 的各类计算误差，可以看到随着单元划分数量增加，各类误差均明显减小，说明所得数值解越来越接近解析解。

表 2–1 Example 1 线性元不同空间步长下各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	5.8317×10^{-4}	3.6803×10^{-3}	1.7951×10^{-3}	5.2731×10^{-2}
1/16	1.4645×10^{-4}	9.4048×10^{-4}	4.4854×10^{-4}	2.6376×10^{-2}
1/32	3.6675×10^{-5}	2.3760×10^{-4}	1.1212×10^{-4}	1.3189×10^{-2}
1/64	9.1700×10^{-6}	5.9704×10^{-5}	2.8029×10^{-5}	6.5949×10^{-3}

(2) 2 次元求解

1、网格划分

如前所述矩阵 P 、 T 为计算域几何模型相关信息，在实际采用 FEM 方法进行仿真计算时使用的为 FEM 网格节点、单元信息，这些信息存储在矩阵 Pb 、 Tb 中，而采用高阶单元时产生的 Pb 、 Tb 矩阵将与计算域基本信息矩阵 P 、 T 不同。针对本问题，网格相关的各个参数的含义如下：

- ▶ 计算域起点: $xStart = 0$; 计算域终点: $xEnd = 1$;
- ▶ 网格单元数: $NE = 2$; 网格节点数: $NN = 2 \times NE + 1 = 5$;
- ▶ 网格单元大小: $h = \frac{xEnd - xStart}{NE} = \frac{1-0}{2} = 0.5$
- ▶ 几何网格节点坐标: $P = \begin{bmatrix} 0 & 0.5 & 1 \end{bmatrix}$, 该矩阵列数为几何模型节点数量;
- ▶ 几何模型网格单元节点编号矩阵: $T = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$, 该矩阵列数为单元数;
- ▶ 有限元空间节点坐标矩阵: $Pb = \begin{bmatrix} 0 & 0.25 & 0.5 & 0.75 & 1 \end{bmatrix}$, 列数为有限元节点数量;
- ▶ 有限元空间单元节点编号矩阵: $Tb = \begin{bmatrix} 1 & 3 \\ 3 & 5 \\ 2 & 4 \end{bmatrix}$, 列数为有限元单元数量, 第 1 行为第 n 单元左节点编号, 第 2 行为第 n 个单元右节点编号, 第 3 行为第 n 个单元内部节点编号;
- ▶ 边界信息矩阵: $BC = \begin{bmatrix} -1 & -1 \\ 1 & 5 \end{bmatrix}$, 第 1 行为边界类型, 这里 -1 表示 Dirichlet 边界; 第 2 行为边界节点编号。

根据给定各参数划分网格示意如图 2-3 所示:

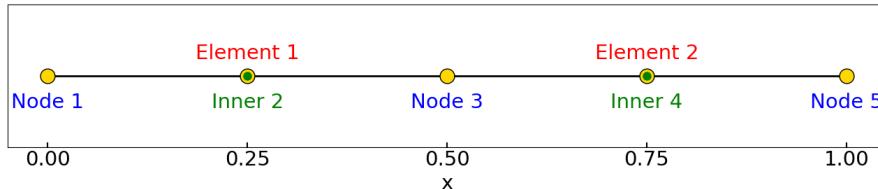


图 2-3 Dirichlet Example 2 次元网格划分

2、刚度矩阵组装

生成有限元网格后需要根据对应网格计算相应的刚度矩阵, 该矩阵的规模与有限元网格节点数量相关, 为: $NN \times NN = 5 \times 5 = 25$ 。

$$\mathbf{K} = \begin{bmatrix} K_{11} & K_{12} & K_{13} & K_{14} & K_{15} \\ K_{21} & K_{22} & K_{23} & K_{24} & K_{25} \\ K_{31} & K_{32} & K_{33} & K_{34} & K_{35} \\ K_{41} & K_{42} & K_{43} & K_{44} & K_{45} \\ K_{51} & K_{52} & K_{53} & K_{54} & K_{55} \end{bmatrix} = \begin{bmatrix} K_{11}^1 & K_{12}^1 & K_{13}^1 & 0 & 0 \\ K_{21}^1 & K_{22}^1 & K_{23}^1 & 0 & 0 \\ K_{31}^1 & K_{32}^1 & K_{33}^1 + K_{11}^2 & K_{12}^2 & K_{13}^2 \\ 0 & 0 & K_{21}^2 & K_{22}^2 & K_{23}^2 \\ 0 & 0 & K_{31}^2 & K_{32}^2 & K_{33}^2 \end{bmatrix} \quad (2.72)$$

其中, K_{ij}^n 表示第 n 个单元, 节点基函数 i, j 对应元素。

• 第1个单元刚度阵计算

单元为2次元，节点个数为3，故局部刚度阵为 3×3 :

$$K_{3 \times 3}^1 = \begin{bmatrix} K_{11}^1 & K_{12}^1 & K_{13}^1 \\ K_{21}^1 & K_{22}^1 & K_{23}^1 \\ K_{31}^1 & K_{32}^1 & K_{33}^1 \end{bmatrix} \quad (2.73)$$

根据存储的几何信息模型信息矩阵 P 、 T 得到局部单元积分上、下区间:

$$low = P(T(1, 1)) = 0, \quad up = P(T(2, 1)) = 0.5 \quad (2.74)$$

局部刚度阵各个元素计算:

$$K_{11}^1 = \int_{low}^{up} c\phi_j' \phi_i' dx = 16 \int_0^{0.5} e^x [4x - (x_n + 3x_{n+1})]^2 dx = 5.2220 \quad (2.75)$$

$$\begin{aligned} K_{12}^1 = K_{21}^1 &= \int_{low}^{up} c\phi_j' \phi_i' dx \\ &= 16 \int_0^{0.5} e^x (4x - (x_n + 3x_{n+1}))(-8x + 4(x_n + x_{n+1})) dx = -6.1156 \end{aligned} \quad (2.76)$$

$$\begin{aligned} K_{13}^1 = K_{31}^1 &= \int_{low}^{up} c\phi_j' \phi_i' dx \\ &= 16 \int_0^{0.5} e^x (4x - (x_n + 3x_{n+1}))(4x - (3x_n + x_{n+1})) dx = 0.8936 \end{aligned} \quad (2.77)$$

$$\begin{aligned} K_{22}^1 &= \int_{low}^{up} c\phi_j' \phi_i' dx = \left(\frac{1}{h^2}\right)^2 \int_0^{0.5} e^x [(-4x^2 + 4(x_n + x_{n+1})x - 4x_n x_{n+1})']^2 dx \\ &= 16 \int_0^{0.5} e^x (-8x + 4(x_n + x_{n+1}))^2 dx = 13.9540 \end{aligned} \quad (2.78)$$

$$\begin{aligned} K_{23}^1 = K_{32}^1 &= \int_{low}^{up} c\phi_j' \phi_i' dx \\ &= 16 \int_0^{0.5} e^x (4x - (3x_n + x_{n+1}))(-8x + 4(x_n + x_{n+1})) dx = -7.8384 \end{aligned} \quad (2.79)$$

$$\begin{aligned} K_{33}^1 &= \int_{low}^{up} c\phi_j' \phi_i' dx = \left(\frac{1}{h^2}\right)^2 \int_0^{0.25} e^x [(2x^2 - (3x_n + x_{n+1})x + x_n^2 + x_n x_{n+1})']^2 dx \\ &= 16 \int_0^{0.25} e^x [(4x - (3x_n + x_{n+1}))]^2 dx = 6.9448 \end{aligned} \quad (2.80)$$

由此得到单元 1 的局部刚度矩阵:

$$K_{3 \times 3}^1 = \begin{bmatrix} K_{11}^1 & K_{12}^1 & K_{13}^1 \\ K_{21}^1 & K_{22}^1 & K_{23}^1 \\ K_{31}^1 & K_{32}^1 & K_{33}^1 \end{bmatrix} = \begin{bmatrix} 5.2220 & -6.1156 & 0.8936 \\ -6.1156 & 13.9540 & -7.8384 \\ 0.8936 & -7.8384 & 6.9448 \end{bmatrix} \quad (2.81)$$

将得到的 $K_{3 \times 3}^1$ 添加至刚度阵 \mathbf{K} 中:

$$\mathbf{K} = \begin{bmatrix} K_{11}^1 & K_{12}^1 & K_{13}^1 & 0 & 0 \\ K_{21}^1 & K_{22}^1 & K_{23}^1 & 0 & 0 \\ K_{31}^1 & K_{32}^1 & K_{33}^1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 5.2220 & -6.1156 & 0.8936 & 0 & 0 \\ -6.1156 & 13.9540 & -7.8384 & 0 & 0 \\ 0.8936 & -7.8384 & 6.9448 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (2.82)$$

• 第 2 个单元刚度矩阵计算

$$K_{3 \times 3}^2 = \begin{bmatrix} K_{11}^2 & K_{12}^2 & K_{13}^2 \\ K_{21}^2 & K_{22}^2 & K_{23}^2 \\ K_{31}^2 & K_{32}^2 & K_{33}^2 \end{bmatrix} \quad (2.83)$$

积分上、下限

$$low = P(T(1, 2)) = 0.5, \quad up = P(T(2, 2)) = 1 \quad (2.84)$$

单元刚度阵各元素计算:

$$\begin{aligned} K_{11}^2 &= \int_{low}^{up} c \phi_j' \phi_i' dx = \left(\frac{1}{h^2}\right)^2 \int_{0.25}^{0.5} e^x [(2x^2 - (x_n + 3x_{n+1})x + x_{n+1}^2 + x_n x_{n+1})']^2 dx \\ &= 16 \int_{0.5}^1 e^x [4x - (x_n + 3x_{n+1})]^2 dx = 8.6096 \end{aligned} \quad (2.85)$$

$$\begin{aligned} K_{12}^2 = K_{21}^2 &= \int_{low}^{up} c \phi_j' \phi_i' dx \\ &= 16 \int_{0.5}^1 e^x (4x - (x_n + 3x_{n+1}))(-8x + 4(x_n + x_{n+1})) dx = -10.0830 \end{aligned} \quad (2.86)$$

$$\begin{aligned} K_{13}^2 = K_{31}^2 &= \int_{low}^{up} c \phi_j' \phi_i' dx \\ &= 16 \int_{0.5}^1 e^x (4x - (x_n + 3x_{n+1}))(4x - (3x_n + x_{n+1})) dx = 1.4733 \end{aligned} \quad (2.87)$$

$$\begin{aligned} K_{22}^2 &= \int_{low}^{up} c\phi_j' \phi_i' dx = \left(\frac{1}{h^2}\right)^2 \int_{0.25}^{0.5} e^x [(-4x^2 + 4(x_n + x_{n+1})x - 4x_n x_{n+1})']^2 dx \\ &= 16 \int_{0.5}^1 e^x (-8x + 4(x_n + x_{n+1}))^2 dx = 23.0063 \end{aligned} \quad (2.88)$$

$$\begin{aligned} K_{23}^2 = K_{32}^2 &= \int_{low}^{up} c\phi_j' \phi_i' dx \\ &= 16 \int_{0.5}^1 e^x (4x - (3x_n + x_{n+1}))(-8x + 4(x_n + x_{n+1})) dx = -12.9233 \end{aligned} \quad (2.89)$$

$$\begin{aligned} K_{33}^2 &= \int_{low}^{up} c\phi_j' \phi_i' dx = \left(\frac{1}{h^2}\right)^2 \int_{0.25}^{0.5} e^x [(2x^2 - (3x_n + x_{n+1})x + x_n^2 + x_n x_{n+1})']^2 dx \\ &= 16 \int_{0.5}^1 e^x [(4x - (3x_n + x_{n+1}))]^2 dx = 11.4500 \end{aligned} \quad (2.90)$$

由此得到单元 2 的局部刚度矩阵:

$$K_{3 \times 3}^2 = \begin{bmatrix} K_{11}^2 & K_{12}^2 & K_{13}^2 \\ K_{21}^2 & K_{22}^2 & K_{23}^2 \\ K_{31}^2 & K_{32}^2 & K_{33}^2 \end{bmatrix} = \begin{bmatrix} 8.6096 & -10.0830 & 1.4733 \\ -10.0830 & 23.0063 & -12.9233 \\ 1.4733 & -12.9233 & 11.4500 \end{bmatrix} \quad (2.91)$$

将得到的 $K_{3 \times 3}^2$ 添加至刚度阵 \mathbf{K} 中:

$$\mathbf{K} = \begin{bmatrix} 5.2220 & -6.1156 & 0.8936 & 0 & 0 \\ -6.1156 & 13.9540 & -7.8384 & 0 & 0 \\ 0.8936 & -7.8384 & 15.5544 & -10.0830 & 1.4733 \\ 0 & 0 & -10.0830 & -23.0063 & -12.9233 \\ 0 & 0 & 1.4733 & -12.9233 & 11.4500 \end{bmatrix} \quad (2.92)$$

由于 2 次元网格仅有 2 个, 所以以上即为总体刚度阵 \mathbf{K} 。

3、载荷向量组装

载荷向量与矩阵相对应, 其规模为有限元网格节点数量, 对于本例而言为: $NN = 5$ 。

$$\mathbf{F} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \end{bmatrix} = \begin{bmatrix} F_1^1 \\ F_2^1 \\ F_3^1 + \textcolor{red}{F}_1^2 \\ \textcolor{red}{F}_2^2 \\ \textcolor{red}{F}_3^2 \end{bmatrix} \quad (2.93)$$

- 第 1 个单元载荷向量计算

$$F_{3 \times 1}^1 = \begin{bmatrix} F_1^1 \\ F_2^1 \\ F_3^1 \end{bmatrix} \quad (2.94)$$

积分上、下限

$$low = P(T(1, 1)) = 0, \quad up = P(T(2, 1)) = 0.5 \quad (2.95)$$

局部载荷向量各个元素计算：

$$\begin{aligned} F_1^1 &= \int_{low}^{up} f \phi_0 dx = \frac{1}{h^2} \int_0^{0.5} -e^x [\cos x - 2 \sin x - x \cos x - x \sin x] \\ &\quad (2x^2 - (x_n + 3x_{n+1})x + x_{n+1}^2 + x_n x_{n+1}) dx = -0.0938 \end{aligned} \quad (2.96)$$

$$\begin{aligned} F_2^1 &= \int_{low}^{up} f \phi_0 dx = \frac{1}{h^2} \int_0^{0.5} -e^x [\cos x - 2 \sin x - x \cos x - x \sin x] \\ &\quad (-4x^2 + 4(x_n + x_{n+1})x - 4x_n x_{n+1}) dx = -0.0498 \end{aligned} \quad (2.97)$$

$$\begin{aligned} F_3^1 &= \int_{low}^{up} f \phi_0 dx = \frac{1}{h^2} \int_0^{0.5} -e^x [\cos x - 2 \sin x - x \cos x - x \sin x] \\ &\quad (2x^2 - (3x_n + x_{n+1})x + x_n^2 + x_n x_{n+1}) dx = 0.0919 \end{aligned} \quad (2.98)$$

由此得到单元 1 的局部向量：

$$F_{3 \times 1}^1 = \begin{bmatrix} F_1^1 \\ F_2^1 \\ F_3^1 \end{bmatrix} = \begin{bmatrix} -0.0938 \\ -0.0498 \\ 0.0919 \end{bmatrix} \quad (2.99)$$

将得到的 $F_{3 \times 1}^1$ 添加至向量 \mathbf{F} 中：

$$\mathbf{F} = F_{3 \times 1}^1 = \begin{bmatrix} F_1^1 \\ F_2^1 \\ F_3^1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.0938 \\ -0.0498 \\ 0.0919 \\ 0 \\ 0 \end{bmatrix} \quad (2.100)$$

- 第 2 个单元载荷向量计算

积分上、下限：

$$low = P(T(1, 2)) = 0.5, \quad up = P(T(2, 2)) = 1 \quad (2.101)$$

局部载荷向量各元素计算：

$$\begin{aligned} F_1^2 &= \int_{low}^{up} f \phi_0 dx = \frac{1}{h^2} \int_{0.5}^1 -e^x [\cos x - 2 \sin x - x \cos x - x \sin x] \\ &\quad (2x^2 - (x_n + 3x_{n+1})x + x_{n+1}^2 + x_n x_{n+1}) dx = 0.0888 \end{aligned} \quad (2.102)$$

$$\begin{aligned} F_2^2 &= \int_{low}^{up} f \phi_0 dx = \frac{1}{h^2} \int_{0.5}^1 -e^x [\cos x - 2 \sin x - x \cos x - x \sin x] \\ &\quad (-4x^2 + 4(x_n + x_{n+1})x - 4x_n x_{n+1}) dx = 1.2257 \end{aligned} \quad (2.103)$$

$$\begin{aligned} F_3^2 &= \int_{low}^{up} f \phi_0 dx = \frac{1}{h^2} \int_{0.5}^1 -e^x [\cos x - 2 \sin x - x \cos x - x \sin x] \\ &\quad (2x^2 - (3x_n + x_{n+1})x + x_n^2 + x_n x_{n+1}) dx = 0.5559 \end{aligned} \quad (2.104)$$

由此得到单元 2 的局部向量：

$$F_{3 \times 1}^2 = \begin{bmatrix} F_1^2 \\ F_2^2 \\ F_3^2 \end{bmatrix} = \begin{bmatrix} 0.0888 \\ 1.2257 \\ 0.5559 \end{bmatrix} \quad (2.105)$$

将得到的 $F_{3 \times 1}^2$ 添加至向量 \mathbf{F} 中：

$$\mathbf{F}^+ = \begin{bmatrix} F_1^1 \\ F_2^1 \\ F_3^1 + \textcolor{red}{F}_1^2 \\ \textcolor{red}{F}_2^2 \\ \textcolor{red}{F}_3^2 \end{bmatrix} = \begin{bmatrix} -0.0938 \\ -0.0498 \\ 0.1807 \\ 1.2257 \\ 0.5559 \end{bmatrix} \quad (2.106)$$

所得载荷向量即为当前 2 次元网格划分时最终载荷向量 \mathbf{F} 。

3、边界条件处理及线性系统求解

采用前述“置 1”法处理边界条件，经过边界处理后得到对应刚度矩阵与载荷向量，得到待求线性系统，采用合适的矩阵求解算法即可得到所求问题的解：

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -6.1156 & 13.9540 & -7.8384 & 0 & 0 \\ 0.8936 & -7.8384 & 15.5544 & -10.0830 & 1.4733 \\ 0 & 0 & -10.0830 & -23.0063 & -12.9233 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} 0 \\ -0.0498 \\ 0.1807 \\ 1.2257 \\ \textcolor{red}{0.5403} \end{bmatrix} \rightarrow \mathbf{u} = \begin{bmatrix} 0.0000 \\ 0.2429 \\ 0.4388 \\ 0.5491 \\ 0.5403 \end{bmatrix} \quad (2.107)$$

4、后处理

图 2-4 为本算例中采用 2 次元 FEM 仿真结果与解析解对比及相关误差，可以看到在相同节点数量下，采用高阶单元计算量与线性元相当，但是采用高阶单元仿真误差减小了一个数量级，所得结果更加接近理论解，因此，在有限元方法中除了细化网格之外，采用高阶单元也是提高计算精度的重要手段，且多数情况下高阶单元的使用较细化网格更加高效。

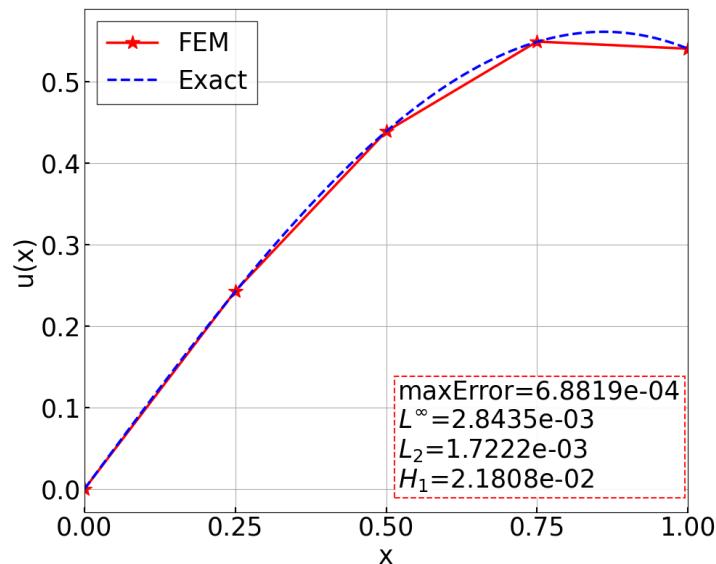


图 2-4 Dirichlet Example 2 次元 FEM 仿真结果、解析解与误差

为了方便程序校验，表 2-2 给出了采用不同 2 次元单元大小下各类误差。

表 2-2 Example 2 二次元不同空间步长下各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	2.9918×10^{-6}	3.9240×10^{-5}	2.6144×10^{-5}	1.3534×10^{-3}
1/16	1.8901×10^{-7}	4.7518×10^{-6}	3.2631×10^{-6}	3.3823×10^{-4}
1/32	1.1869×10^{-8}	5.8390×10^{-7}	4.0774×10^{-7}	8.4550×10^{-5}
1/64	7.4356×10^{-10}	7.2343×10^{-8}	5.0962×10^{-8}	2.1137×10^{-5}

2.5.2 Example 2

例 2.2 【问题描述】分别使用 1D 线性元和二次元求解下述问题：

$$\begin{cases} -\frac{d}{dx} \left(e^x \frac{du(x)}{dx} \right) = -e^x [\cos x - 2 \sin x - x \cos x - x \sin x], \quad (0 \leq x \leq 1) \\ u(0) = 0, \quad u'(1) = \cos(1) - \sin(1) \end{cases}$$

【说明】上述问题的解析解为： $u = x \cos x$ ，该解可以用来对 FEM 方法求解的结果误差进行估计。

本算例模型中在边界处理前所得刚度矩阵、载荷向量与前述问题在边界处理前完全一致。区别仅在于边界条件，在本算例模型中左边界为 Dirichlet 边界，右边界为 Neumann 边界。根据不同边界类型的定义与特征可以知道：左边界值已经给定，仅需对右边界进行处理，右边界处理详细理论与方法参考 2.3.2 节中第 2 类边界条件介绍。

(1) 线性元求解

采用 2.5.1 节中图 2-1 所示线性元网格划分，如前所述采用线性单元，在边界处理前所得刚度矩阵与载荷向量分别为式 (2.59) 和式 (2.70)，在此基础上，针对本问题进行边界条件处理。

左侧为 Dirichlet 边界，因此该边界节点对应刚度矩阵采用“置 1”法处理，载荷向量为给定边界值 0；右侧为 Neumann 边界条件，刚度矩阵不再变化，对载荷向量而言，需要根据右侧边界条件计算 $u'(1) \times e$ 项，并将所得结果添加至载荷向量 \mathbf{F} 中：

$$u'(1) \times e = e \times (\cos(1) - \sin(1)) = -0.818661 \quad (2.108)$$

边界处理后得到待求线性系统，采用合适方法进行求解可得到本问题有限元解：

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -4.5444 & 10.3795 & -5.8351 & 0 & 0 \\ 0 & -5.8351 & 13.3276 & -7.4925 & 0 \\ 0 & 0 & -7.4925 & 17.1130 & -9.6205 \\ 0 & 0 & 0 & -9.6205 & 9.6205 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} 0 \\ -0.0402 \\ 0.3311 \\ 0.9153 \\ -0.1076 \end{bmatrix} \rightarrow \mathbf{u} = \begin{bmatrix} 0.0000 \\ 0.2417 \\ 0.4369 \\ 0.5447 \\ 0.5335 \end{bmatrix} \quad (2.109)$$

仿真后得到的结果、解析解及相关误差信息如图 2-5 所示，表 2-3 给出了采用不同大小线性元时 FEM 仿真结果的各类误差结果参考。

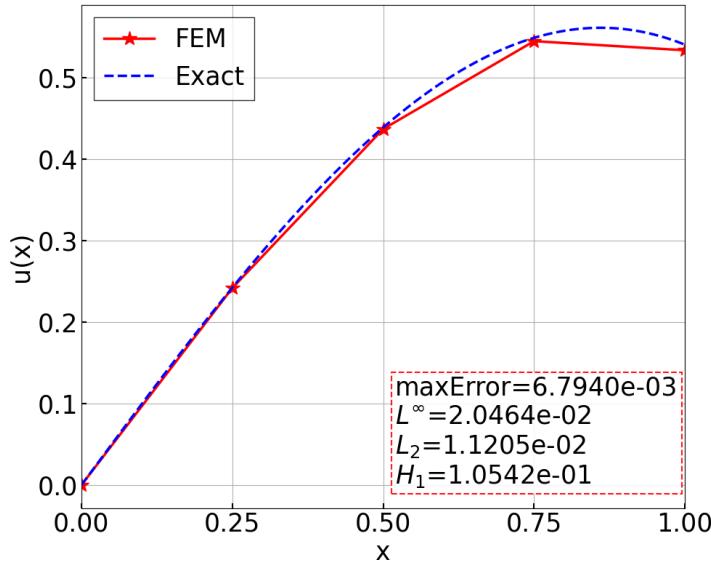


图 2-5 Neumann Example 线性元 FEM 仿真结果、解析解与误差

表 2-3 Neumann Example 线性元不同空间步长下各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	1.6953×10^{-3}	5.3323×10^{-3}	2.8009×10^{-3}	5.2748×10^{-2}
1/16	4.2362×10^{-4}	1.3589×10^{-3}	7.0020×10^{-4}	2.6378×10^{-2}
1/32	1.0589×10^{-4}	3.4285×10^{-4}	1.7505×10^{-4}	1.3190×10^{-2}
1/64	2.6473×10^{-5}	8.6097×10^{-5}	4.3762×10^{-5}	6.5949×10^{-3}

(2) 2 次元求解

采用 2.5.1 节中图 2-3 所示二次元网格划分，如前所述采用 2 阶单元，在边界处理前所得刚度矩阵与载荷向量分别为式 (2.92) 和式 (2.106)，在此基础上，针对本问题进行边界条件处理。

与采用线性元时类似，左侧为 Dirichlet 边界，因此该边界节点对应刚度矩阵采用“置 1”法处理，载荷向量为给定边界值 0；右侧为 Neumann 边界条件，刚度矩阵不再变化，对载荷向量而言，需要根据右侧边界条件计算 $u'(1) \times e$ 项，其结果与式 (2.108) 相同，并将所得结果添加至载荷向量 \mathbf{F} 中：

边界处理后得到待求线性系统，采用合适方法进行求解可得到本问题有限元解：

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -6.1156 & 13.9540 & -7.8384 & 0 & 0 \\ 0.8936 & -7.8384 & 15.5544 & -10.0830 & 1.4733 \\ 0 & 0 & -10.0830 & -23.0063 & -12.9233 \\ 0 & 0 & 1.4733 & -12.9233 & 11.4500 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -0.2628 \end{bmatrix} \rightarrow \mathbf{u} = \begin{bmatrix} 0.0000 \\ 0.2430 \\ 0.4389 \\ 0.5493 \\ 0.5405 \end{bmatrix} \quad (2.110)$$

仿真后得到的结果、解析解及相关误差信息如图 2-6 所示：

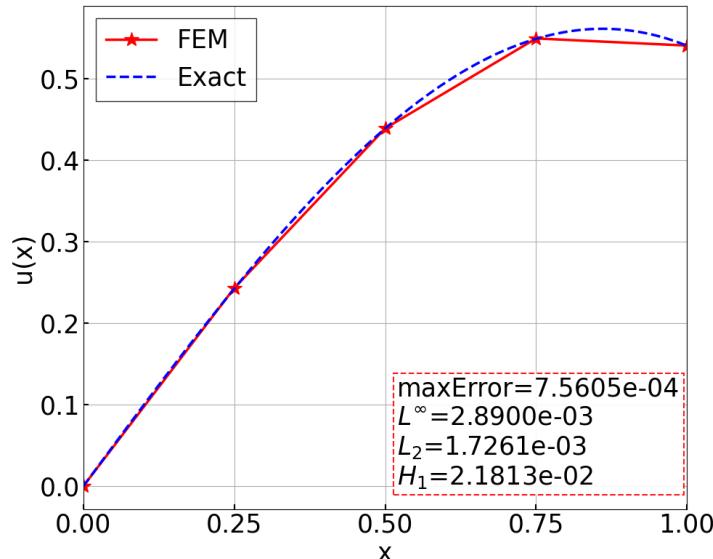


图 2-6 Neumann Example 二次元 FEM 仿真结果、解析解与误差

表 2-4 给出了采用不同大小二次元时 FEM 仿真结果的各类误差结果参考。相同网格情况下采用 2 次元具有更高计算精度，随着网格单元大小的减小，计算误差缩小越来越快。

表 2-4 Neumann Example 二次元不同空间步长下各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	3.0641×10^{-6}	3.9288×10^{-5}	2.6148×10^{-5}	1.3534×10^{-3}
1/16	1.9154×10^{-7}	4.7533×10^{-6}	3.2632×10^{-6}	3.3823×10^{-4}
1/32	1.1976×10^{-8}	5.8395×10^{-7}	4.0774×10^{-7}	8.4550×10^{-5}
1/64	7.4865×10^{-10}	7.2344×10^{-8}	5.0962×10^{-8}	2.1137×10^{-5}

2.5.3 Example 3

例 2.3 【问题描述】分别使用 1D 线性元和二次元求解下述问题：

$$\begin{cases} -\frac{d}{dx} \left(e^x \frac{du(x)}{dx} \right) = -e^x [\cos x - 2 \sin x - x \cos x - x \sin x], \quad (0 \leq x \leq 1) \\ u'(0) + u(0) = 1, \quad u(1) = \cos(1) \end{cases}$$

【说明】上述问题的解析解为： $u = x \cos x$ ，该解可以用来对 FEM 方法求解的结果误差进行估计。

本算例模型中在边界处理前所得刚度矩阵、载荷向量与前述问题在边界处理前完全一致。区别仅在于边界条件，在本算例模型中左边为 Robin 边界，右边为 Dirichlet 边界。根据不同边界类型的定义与特征可以知道：右边界值已经给定，仅需对左边界进行处理，左边界处理详细理论与方法参考 2.3.3 节中第 3 类边界条件介绍，本算例中 $q_a = 1$ ， $c(x) = e^x$ ， $p_a = 1$ 。

(1) 线性元求解

采用 2.5.1 节中图 2-1 所示线性元网格划分，如前所述采用线性单元，在边界处理前所得刚度矩阵与载荷向量分别为式 (2.59) 和式 (2.70)，在此基础上，针对本问题进行边界条件处理。

对右边界采用“置 1”法处理，载荷向量为边界值 $\cos(1)$ ；左侧为 Robin 边界，对应刚度矩阵中需要增加 $-q_0 c(0)$ 项，对应载荷向量需要增加 $-p_0 c(0)$ 项：

$$-q_0 c(0) = -1 \times e^0 = -1, \quad -p_0 c(0) = -1 \times e^0 = -1 \quad (2.111)$$

边界处理后得到待求线性系统，采用合适方法进行求解可得到本问题有限元解：

$$\begin{bmatrix} 3.5444 & -4.5444 & 0 & 0 & 0 \\ -4.5444 & 10.3795 & -5.8351 & 0 & 0 \\ 0 & -5.8351 & 13.3276 & -7.4925 & 0 \\ 0 & 0 & -7.4925 & 17.1130 & -9.6205 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} -1.0986 \\ -0.0402 \\ 0.3311 \\ 0.9153 \\ -0.5403 \end{bmatrix} \rightarrow \mathbf{u} = \begin{bmatrix} 0.0183 \\ 0.2560 \\ 0.4480 \\ 0.5534 \\ 0.5403 \end{bmatrix} \quad (2.112)$$

仿真后得到的结果、解析解及相关误差信息如图 2-7 所示。可以看到当左边界为 Robin 边界时在网格数量较少时该边界处所得函数值与解析解相差较大，当网格数量逐渐增加时，该边界处的函数值才逐渐与解析解吻合。到此通过不同类型边界条件的有限元方法数值模拟可以看出，第一类边界条件能够给出问题准确解，该边界条件在各类边界条件中最为重要，其他两类边界条件对求解结果影响较大。

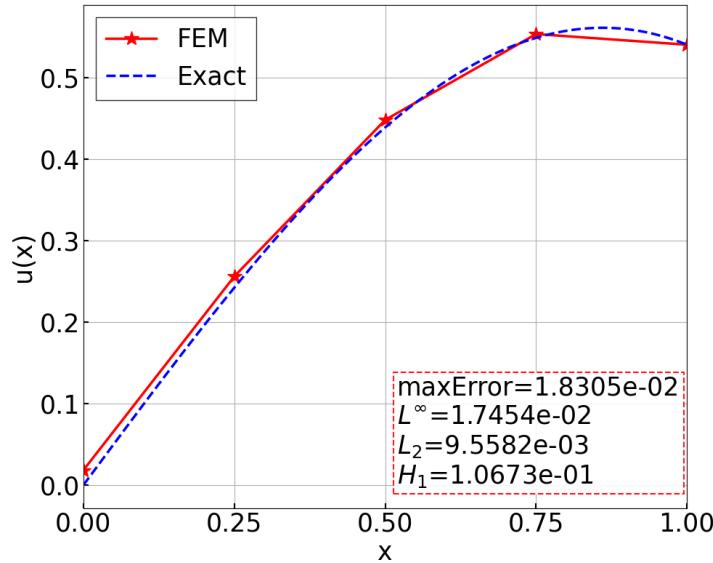


图 2-7 Robin Example 线性元 FEM 仿真结果、解析解与误差

表 2-5 给出了采用不同大小线性元时 FEM 仿真结果的各类误差结果参考。

表 2-5 Robin Example 线性元不同空间步长下各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	4.5980×10^{-3}	4.4910×10^{-3}	2.4222×10^{-3}	5.2914×10^{-2}
1/16	1.1509×10^{-3}	1.1375×10^{-3}	6.0759×10^{-4}	2.6399×10^{-2}
1/32	2.8781×10^{-4}	2.8613×10^{-4}	1.5202×10^{-4}	1.3192×10^{-2}
1/64	7.1957×10^{-5}	7.1748×10^{-5}	3.8014×10^{-5}	6.5953×10^{-3}

(2) 2 次元求解

采用 2.5.1 节中图 2-3 所示二次元网格划分，如前所述采用 2 阶单元，在边界处理前所得刚度矩阵与载荷向量分别为式 (2.92) 和式 (2.106)，在此基础上，针对本问题进行边界条件处理。与采用线性元时类似，左侧为 Robin 边界，对应刚度矩阵中需要增加 $-q_0 c(0)$ 项，对应载荷向量需要增加 $-p_0 c(0)$ 项，其结果与式 (2.111) 相同；右侧为 Dirichlet 边界，对应刚度矩阵采用“置 1”法处理，载荷向量为边界值 $\cos(1)$ 。

边界处理后得到待求线性系统，采用合适方法进行求解可得到本问题有限元解：

$$\begin{bmatrix} 4.2220 & -6.1156 & 0.8936 & 0 & 0 \\ -6.1156 & 13.9540 & -7.8384 & 0 & 0 \\ 0.8936 & -7.8384 & 15.5544 & -10.0830 & 1.4733 \\ 0 & 0 & -10.0830 & -23.0063 & -12.9233 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} -1.0938 \\ -0.0498 \\ 0.1807 \\ 1.2257 \\ 0.5403 \end{bmatrix} \rightarrow \mathbf{u} = \begin{bmatrix} -0.0005 \\ 0.2426 \\ 0.4386 \\ 0.5490 \\ 0.5403 \end{bmatrix} \quad (2.113)$$

仿真后得到的结果、解析解及相关误差信息如图 2-8 所示。可以明显看到当采用 2 次元之后所得结果与解析解拟合得更好，尤其在 Robin 边界处，可以看出采用 2 次元所得函数值与线性元相比更加接近解析解。

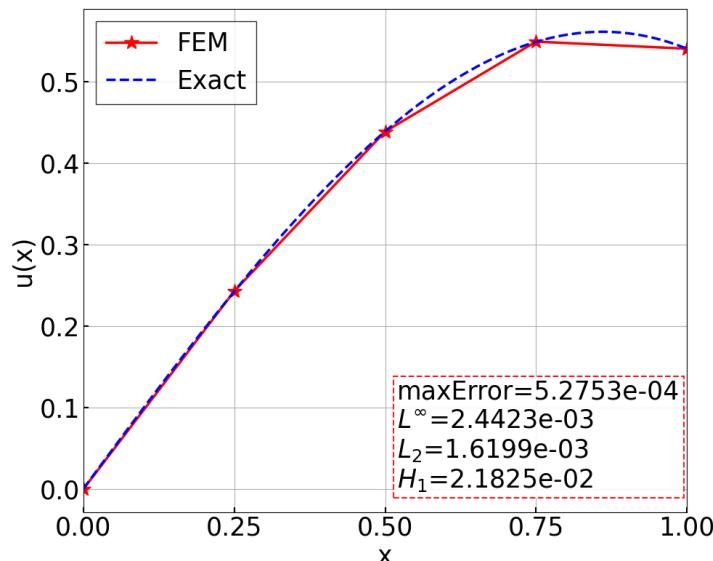


图 2-8 Robin Example 二次元元 FEM 仿真结果、解析解与误差

表 2-6 给出了采用不同大小二次元时 FEM 仿真结果的各类误差结果参考。

表 2-6 Robin Example 二次元不同空间步长下各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	2.0500×10^{-6}	3.7321×10^{-5}	2.6042×10^{-5}	1.3534×10^{-3}
1/16	1.2809×10^{-7}	4.6278×10^{-6}	3.2599×10^{-6}	3.3823×10^{-4}
1/32	8.0048×10^{-9}	5.7602×10^{-7}	4.0764×10^{-7}	8.4550×10^{-5}
1/64	4.9985×10^{-10}	7.1847×10^{-8}	5.0959×10^{-8}	2.1137×10^{-5}

习题

1. 请写出如下泊松方程的有限元求解过程，单元采用线性元，高斯积分点数为 4，网格划分为 4 个单元，计算各类误差（最大节点误差、 L^∞ 误差、 L^2 误差、 H^1 误差）。

$$\begin{cases} -\Delta u = -6x, & (-2 \leq x \leq 2) \\ u(-2) = -8, \quad u(2) = 8 \end{cases}$$

该问题的解析解为： $u(x) = x^3$ 。

2. 请写出如下泊松方程的有限元求解过程，单元采用 2 次元，高斯积分点数为 4，网格划分为 2 个单元，计算各类误差（最大节点误差、 L^∞ 误差、 L^2 误差、 H^1 误差）。

$$\begin{cases} \Delta u = 2, & (-1 \leq x \leq 1) \\ u(-1) = 0, \quad \frac{du}{dx} \Big|_{x=1} = 4 \end{cases}$$

该问题的解析解为： $u(x) = (x + 1)^2$ 。

3. 请写出如下泊松方程的有限元求解过程，单元采用 2 次元，高斯积分点数为 4，网格划分为 2 个单元，计算各类误差（最大节点误差、 L^∞ 误差、 L^2 误差、 H^1 误差）。

$$\begin{cases} \Delta u = 2, & (-1 \leq x \leq 1) \\ u(-1) = 0, \quad \frac{du}{dx} \Big|_{x=1} = u \end{cases}$$

该问题的解析解为： $u(x) = (x + 1)^2$ 。

3

2D 椭圆方程有限元方法

3.1 引言

本章主要对 2D2 阶椭圆方程^[6-7,9-10] (2D second order elliptic equation) 的有限元方法进行介绍，主要包括：2D2 阶椭圆方程弱格式推导、构建线性系统、不同边界条件处理、误差计算等内容。

3.2 问题模型

以下述 2D2 阶椭圆方程为例对 2D FEM 方法求解方程详细过程进行介绍：

$$\begin{cases} -\nabla \cdot (c \nabla u) = f, & (x, y) \in \Omega \\ u = g, & (x, y) \in \partial\Omega \end{cases} \quad (3.1)$$

其中， Ω 为一个 2D 空间域， $f(x, y)$ 和 $c(x, y)$ 为空间 Ω 上已知函数， $g(x, y)$ 为边界 $\partial\Omega$ 上已知函数， $u(x, y)$ 为待求函数——试探函数 (Trial Function)。

3.2.1 弱格式

(1) 方程两边同时乘以一个测试函数 $v(x, y)$

$$-\nabla \cdot (c \nabla u) v = fv \quad (3.2)$$

(2) 对方程两边进行积分

$$-\int_{\Omega} \nabla \cdot (c \nabla u) v d\Omega = \int_{\Omega} f v d\Omega \quad (3.3)$$

(3) 积分变换

$$\int_{\Omega} \nabla \cdot (c \nabla u) v dx dy = \int_{\partial\Omega} (c \nabla u \cdot \vec{n}) v ds - \int_{\Omega} c \nabla u \cdot \nabla v dx dy \quad (3.4)$$

$$\int_{\Omega} c \nabla u \cdot \nabla v dx dy - \int_{\partial\Omega} (c \nabla u \cdot \vec{n}) v ds = \int_{\Omega} f v dx dy \quad (3.5)$$

在边界 $\partial\Omega$ 处试探函数值 $u(x, y) = g(x, y)$ 为已知, 由于测试函数 $v(x, y)$ 为任意函数, 故此可以令其在边界处为 0, 即 $v(x, y) = 0, (x, y) \in \partial\Omega$ 。由此可以将上式进一步简化得到:

$$\int_{\Omega} c \nabla u \cdot \nabla v dx dy = \int_{\Omega} f v dx dy \quad (3.6)$$

方程 (3.6) 即为 2D2 阶椭圆方程的弱格式。

3.2.2 方程离散

令 $a(u, v) = \int_{\Omega} c \nabla u \cdot \nabla v dx dy$, $(f, v) = \int_{\Omega} f v dx dy$, 求解试探函数, 就是在空间 Ω 中寻找 u , 使其对任意 $v \in H_0^1(\Omega)$ 满足:

$$a(u, v) = (f, v) \quad (3.7)$$

上述弱格式就是在空间划分后的子空间内 ($U_h \in H^1(\Omega)$), 得到各个子空间内试探函数的解 u_h , 随后将各个子空间得解合并得到整个空间的解。

$$\begin{aligned} a(u_h, v_h) &= (f, v_h) \\ \leftrightarrow \int_{\Omega} c \nabla u_h \cdot \nabla v_h dx dy &= \int_{\Omega} f v_h dx dy \end{aligned} \quad (3.8)$$

将离散后的各节点函数值按照基底展开:

$$u_h(x_k, y_k) = \sum_{j=1}^{N_b} u_j \phi_j(\xi_k, \eta_k) = u_k \quad (3.9)$$

其中, N_b 为有限元基函数节点数。

选取合适测试函数：较为简单的操作就是将基函数作为测试函数，在各个离散空间上选择基函数作为测试函数： $v_h = \phi_i, (i = 1, \dots, N_b)$ ，可以得到有限元方程格式为：

$$\begin{aligned} & \int_{\Omega} c \nabla \left(\sum_{j=1}^{N_b} u_j \phi_j \right) \cdot \nabla \phi_i dx dy = \int_{\Omega} f \phi_i dx dy \\ & \rightarrow \sum_{j=1}^{N_b} u_j \left[\int_{\Omega} c \nabla \phi_j \cdot \nabla \phi_i dx dy \right] = \int_{\Omega} f \phi_i dx dy, (i = 1, \dots, N_b) \end{aligned} \quad (3.10)$$

类比 1D 情况可以知道，刚度矩阵为：

$$\mathbf{K} = [K_{ij}]_{i,j=1}^{N_b} = \left[\int_{\Omega} c \nabla \phi_j \cdot \nabla \phi_i dx dy \right]_{i,j=1}^{N_b} \quad (3.11)$$

载荷向量：

$$\mathbf{F} = [b_i]_{i=1}^{N_b} = \left[\int_{\Omega} f \phi_i dx dy \right]_{i=1}^{N_b} \quad (3.12)$$

3.2.3 线性系统构建

(1) 组装刚度矩阵

为了方便对刚度矩阵中各个元素进行计算，通常先计算各个单元的局部刚度阵，可以看到当前刚度阵各个元素的计算是在全局坐标系下进行，如前所述为了避免复杂的坐标转换，通常将刚度阵元素的计算由全局坐标转换至参考单元坐标系下进行。其转换关系如下：

$$\begin{aligned} K_{ij}^e &= \int_{\Omega} c \nabla \phi_j \cdot \nabla \phi_i dV \\ &= \int_{\Omega} c(\xi, \eta) \nabla_{(x,y)} \psi_j \cdot \nabla_{(x,y)} \psi_i \left| \frac{\partial(x, y)}{\partial(\xi, \eta)} \right| d\xi d\eta \\ &= \int_{\Omega} c(\xi, \eta) \left(\frac{\partial \psi_j}{\partial x} \frac{\partial \psi_i}{\partial x} + \frac{\partial \psi_j}{\partial y} \frac{\partial \psi_i}{\partial y} \right) |J| d\xi d\eta \end{aligned} \quad (3.13)$$

可以看到矩阵系统中各元素主要由 x, y 两部分组成，故可以将其分为两部分分别计算，详细计算方法可以通过算法 6 分别得到，详细过程如下：

- 令算法 6 中 $r = 1, s = 0, p = 1, q = 0$ ，得到 x 分量；
- 令算法 6 中 $r = 0, s = 1, p = 0, q = 1$ ，得到 y 分量；

将得到的分量相加即可得到矩阵系统中各元素的值， $K_{ij}^e = K_{ij,x}^e + K_{ij,y}^e$ 。

Algorithm 6 2D2 阶椭圆方程刚度矩阵组装

Require: $\mathbf{K} = \text{sparse}(N_b^{test}, N_b^{trial})$

```

1: for  $n = 1, \dots, N$  do
2:   for  $\alpha = 1, \dots, N_{lb}^{trial}$  do
3:     for  $\beta = 1, \dots, N_{lb}^{test}$  do
4:       Compute  $r = \int_{E_n} c \frac{\partial^{r+s} \varphi_{n\alpha}}{\partial x^r \partial y^s} \frac{\partial^{p+q} \varphi_{n\beta}}{\partial x^p \partial y^q} dx dy$ 
5:        $\mathbf{K}(T_b^{test}(\beta, n), T_b^{trial}(\alpha, n)) += r$ 
6:     end for
7:   end for
8: end for

```

(2) 载荷向量组装

载荷向量中各元素的计算方式与刚度阵中各元素的计算方法相似。

$$\int_{\Omega} f \phi_i dV = \int_{\Omega} f(\xi, \eta) \psi_i(\xi, \eta) \left| \frac{\partial(x, y)}{\partial(\xi, \eta)} \right| d\xi d\eta = \int_{\Omega} f(\xi, \eta) \psi_i(\xi, \eta) |J| d\xi d\eta \quad (3.14)$$

其中, J 为雅可比矩阵, 在 2D FEM 中为单元与参考单元的面积比, 其具有如下形式:

$$\begin{aligned}
J &= \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{NN} \frac{\partial N_i(\xi, \eta)}{\partial \xi} x_i & \sum_{i=1}^{NN} \frac{\partial N_i(\xi, \eta)}{\partial \xi} y_i \\ \sum_{i=1}^{NN} \frac{\partial N_i(\xi, \eta)}{\partial \eta} x_i & \sum_{i=1}^{NN} \frac{\partial N_i(\xi, \eta)}{\partial \eta} y_i \end{bmatrix} \\
&= \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \xi} & \dots & \frac{\partial N_{NN}}{\partial \xi} \\ \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \dots & \frac{\partial N_{NN}}{\partial \eta} \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_{NN} & y_{NN} \end{bmatrix} \quad (3.15)
\end{aligned}$$

各项节点基函数对参考单元的导数为:

$$\begin{cases} \frac{\partial N_i}{\partial \xi} = \frac{\partial N_i}{\partial x} \frac{\partial x}{\partial \xi} + \frac{\partial N_i}{\partial y} \frac{\partial y}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} = \frac{\partial N_i}{\partial x} \frac{\partial x}{\partial \eta} + \frac{\partial N_i}{\partial y} \frac{\partial y}{\partial \eta} \end{cases} \rightarrow \begin{bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} \\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{bmatrix} = J^{-1} \begin{bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{bmatrix} \quad (3.16)$$

对于 2×2 的 Jacobian 矩阵有: $J = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \longrightarrow J^{-1} = \frac{1}{|J|} \begin{bmatrix} D & -B \\ -C & A \end{bmatrix}$, 其中 $|J| = AD - BC$ 。

载荷向量的组装方法如算法 7 所示, 令 $p = q = 0$ 即可得到 \mathbf{F} 。

Algorithm 7 2D 椭圆方程载荷向量组装

Require: $\mathbf{F} = \text{sparse}(N_b, 1)$

```

1: for  $n = 1, \dots, N$  do
2:   for  $\beta = 1, \dots, N_{lb}$  do
3:     Compute  $r = \int_{E_n} f \frac{\partial^{p+q} \varphi_{n\beta}}{\partial x^p \partial y^q} dx dy$ 
4:      $\mathbf{F}(T_b^{test}(\beta, n), 1) += r$ 
5:   end for
6: end for

```

(3) 待求函数值

$$\mathbf{u} = [u_j]_{j=1}^{N_b} \quad (3.17)$$

到此, 已经得到了 2D FEM 方法需要求解的线性系统, 随后根据待求问题给定的边界条件, 参考 1D FEM 方法中介绍的不同边界条件下刚度矩阵及载荷向量的处理方式, 得到边界处理后的线性系统并对其进行求解即可得到最终结果, 最后根据需要进行相应后处理即可。

3.3 不同边界条件处理

3.3.1 第 1 类边界条件

在 2D FEM 情况下, 该边界条件的处理方法与 1D FEM 处理方式相同主要有:

- 1) 乘大数法;
- 2) “置 1” 法, 在该方法中有两种处理方式:
 - 仅行“置 1”;
 - 行列都“置 1”, 此时对应的载荷向量也需要处理由于列“置 1”带来的影响。

本章中第 1 类边界处理方法与 1D 情况同类型边界处理方式相同, 详细算法流程如算法 8 所示:

Algorithm 8 2D 椭圆方程 Dirichlet 边界处理

```

1: for  $k = 1, \dots, N_{BN}$  do
2:   if  $BC[0, k] ==$  Dirichlet then
3:      $i = BC[1, k]$ 
4:      $A[i, :] = 0$ 
5:      $A[i, i] = 1$ 
6:      $b[i] = g(Pb[:, i])$ 
7:   end if
8: end for

```

3.3.2 第 2 类边界条件

以问题 (3.1) 为例, 假设边界为 Neumann 边界类型, 则上述问题具有如下形式:

$$\begin{cases} -\nabla \cdot (c \nabla u) = f, & (x, y) \in \Omega \\ \nabla u \cdot \vec{n} = p, & (x, y) \in \partial\Omega \end{cases} \quad (3.18)$$

由前述推导可得:

$$\int_{\Omega} c \nabla u \cdot \nabla v dx dy - \int_{\partial\Omega} (c \nabla u \cdot \vec{n}) v ds = \int_{\Omega} f v dx dy \quad (3.19)$$

将给定问题的边界条件代入上式可以得到:

$$\int_{\Omega} c \nabla u \cdot \nabla v dx dy = \int_{\Omega} f v dx dy + \int_{\partial\Omega} \textcolor{red}{c p v ds} \quad (3.20)$$

可以看到由式 (3.20) 得到的方程的解并不唯一! 很显然载荷向量一侧的大小, 由于测试函数 v 的选择不同将产生不同的结果, 所以该情况下所得方程的解不唯一。

所以通常对于上述问题边界条件为 Dirichlet + Neumann 的混合边界条件, 由此问题 (3.18) 就需要变换为:

$$\begin{cases} -\nabla \cdot (c \nabla u) = f, & (x, y) \in \Omega \\ \nabla u \cdot \vec{n} = p, & \Gamma_N \subset \partial\Omega \\ u = g, & (x, y) \in \partial\Omega / \Gamma_N \end{cases} \quad (3.21)$$

所以, 式 (3.20) 中边界部分由两部分组成:

$$\int_{\partial\Omega} (c \nabla u \cdot \vec{n}) v ds = \int_{\Gamma_N} (c \nabla u \cdot \vec{n}) v ds + \int_{\partial\Omega / \Gamma_N} (c \nabla u \cdot \vec{n}) v ds \quad (3.22)$$

问题(3.21)中有Dirichlet边界，在该边界处函数值已知，无需继续计算，可以选择对应测试函数使其在该边界处为0，这样式(3.22)可以变换为：

$$\int_{\partial\Omega} (c\nabla u \cdot \vec{n}) v ds = \int_{\Gamma_N} c p v ds \quad (3.23)$$

将式(3.23)代入式(3.19)中整理变换后可以得到：

$$\int_{\Omega} c \nabla u \cdot \nabla v dx dy = \int_{\Omega} f v dx dy + \int_{\Gamma_N} c p v ds \quad (3.24)$$

所以仅需要在对应的Neumann边界对应的载荷向量处增加 $\int_{\Gamma_N} c p v ds$ 即可。

3.3.3 第3类边界条件

以下述问题为例，边界中包含Robin边界条件类型：

$$\begin{cases} -\nabla \cdot (c \nabla u) = f, (x, y) \in \Omega \\ \nabla u \cdot \vec{n} + ru = q, \Gamma_R \subseteq \partial\Omega \\ u = g, \partial\Omega / \Gamma_R \end{cases} \quad (3.25)$$

由前述推导可得：

$$\int_{\Omega} c \nabla u \cdot \nabla v dx dy - \int_{\partial\Omega} (c \nabla u \cdot \vec{n}) v ds = \int_{\Omega} f v dx dy \quad (3.26)$$

由所得弱格式(3.26)对应边界条件可以知道，该边界由两部分组成：Dirichlet + Robin，在Dirichlet边界条件 $\partial\Omega/\Gamma_R$ 下，可以取测试函数 $v(x) = 0$ 。所以边界处的积分可以写作：

$$\begin{aligned} \int_{\partial\Omega} (c \nabla u \cdot \vec{n}) v ds &= \int_{\Gamma_R} (c \nabla u \cdot \vec{n}) v ds + \int_{\partial\Omega/\Gamma_R} (c \nabla u \cdot \vec{n}) v ds \\ &= \int_{\Gamma_R} c (q - ru) v ds \\ &= \int_{\Gamma_R} cq v ds - \int_{\Gamma_R} cruv ds \end{aligned} \quad (3.27)$$

将式(3.27)代入(3.26)中可以得到：

$$\int_{\Omega} c \nabla u \cdot \nabla v dx dy - \left(\int_{\Gamma_R} cq v ds - \int_{\Gamma_R} cruv ds \right) = \int_{\Omega} f v dx dy \quad (3.28)$$

对所得式进行整理变换可以得到：

$$\int_{\Omega} c \nabla u \cdot \nabla v dx dy + \int_{\Gamma_R} cruv ds = \int_{\Omega} f v dx dy + \int_{\Gamma_R} cq v ds \quad (3.29)$$

所以需要在对应的Robin边界对应的刚度矩阵处增加 $\int_{\Gamma_R} cruv ds$ ，载荷向量处增加 $\int_{\Gamma_R} cq v ds$ 即可。

3.3.4 混合边界条件

以下述问题为例，边界为 Dirichlet + Neumann + Robin 类型：

$$\begin{cases} -\nabla \cdot (c \nabla u) = f, (x, y) \in \Omega \\ \nabla u \cdot \vec{n} = p, \text{ on } \Gamma_N \subset \partial\Omega \\ \nabla u \cdot \vec{n} + ru = q, \Gamma_R \subseteq \partial\Omega \\ u = g, \partial\Omega / (\Gamma_N \cup \Gamma_R) \end{cases} \quad (3.30)$$

由前述推导可得：

$$\int_{\Omega} c \nabla u \cdot \nabla v dx dy - \int_{\partial\Omega} (c \nabla u \cdot \vec{n}) v ds = \int_{\Omega} f v dx dy \quad (3.31)$$

在边界 $\partial\Omega / (\Gamma_N \cup \Gamma_R)$ 处，函数值由 $u = g$ 给定，所以，在该边界处可以取测试函数 $v(x) = 0$ ，由此可以得到：

$$\int_{\Omega} c \nabla u \cdot \nabla v dx dy + \int_{\Gamma_R} \cancel{cruv ds} = \int_{\Omega} f v dx dy + \int_{\Gamma_N} \cancel{cpv ds} + \int_{\Gamma_R} \cancel{cqv ds} \quad (3.32)$$

在求解该系统时，依次按照 Robin 型、Neumann 型、Dirichlet 型边界条件进行处理，始终保证 Dirichlet 边界发挥作用，具体而言为：在 Robin 型对应边界刚度矩阵增加 $\int_{\Gamma_R} \cancel{cruv ds}$ ，载荷向量增加 $\int_{\Gamma_R} \cancel{cqv ds}$ ；在 Neumann 型对应边界载荷向量增加 $\int_{\Gamma_N} \cancel{cpv ds}$ ；对 Dirichlet 类型边界对应刚度矩阵采用“置 1”法，载荷向量为给定边界条件值。

3.4 各项异性问题

当扩散系数 c 为一张量时，此时问题 (3.1) 表示各向异性椭圆问题：

$$\begin{cases} -\nabla \cdot (c \nabla u) = f, (x, y) \in \Omega \\ c \nabla u \cdot \vec{n} = p, \text{ on } \Gamma_N \subset \partial\Omega \\ c \nabla u \cdot \vec{n} + ru = q, \Gamma_R \subseteq \partial\Omega \\ u = g, \partial\Omega / (\Gamma_N \cup \Gamma_R) \end{cases} \quad (3.33)$$

其中， $c = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$ ，由前述推导可得：

$$\int_{\Omega} c \nabla u \cdot \nabla v dx dy - \int_{\partial\Omega} (c \nabla u \cdot \vec{n}) v ds = \int_{\Omega} f v dx dy \quad (3.34)$$

在边界 $\partial\Omega/(\Gamma_N \cup \Gamma_R)$ 处，函数值由 $u = g$ 给定，所以，在该边界处可以取测试函数 $v(x) = 0$ ，由此可以得到：

$$\int_{\Omega} c \nabla u \cdot \nabla v dx dy + \int_{\Gamma_R} r u v ds = \int_{\Omega} f v dx dy + \int_{\Gamma_N} p v ds + \int_{\Gamma_R} q v ds \quad (3.35)$$

其中，

$$c \nabla u \cdot \nabla v = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} c_{11}u_x + c_{12}u_y \\ c_{21}u_x + c_{22}u_y \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \end{bmatrix} \quad (3.36)$$

$$= c_{11}u_xv_x + c_{12}u_yv_x + c_{21}u_xv_y + c_{22}u_yv_y$$

- 令算法 6 中 $c = c_{11}$, $r = p = 1$, $s = q = 0$, 得到矩阵 \mathbf{K}_1 ;
- 令算法 6 中 $c = c_{12}$, $s = p = 1$, $r = q = 0$, 得到矩阵 \mathbf{K}_2 ;
- 令算法 6 中 $c = c_{21}$, $r = q = 1$, $s = p = 0$, 得到矩阵 \mathbf{K}_3 ;
- 令算法 6 中 $c = c_{22}$, $r = p = 0$, $s = q = 1$, 得到矩阵 \mathbf{K}_4 ;

由此得到系统总刚度矩阵： $\mathbf{K} = \mathbf{K}_1 + \mathbf{K}_2 + \mathbf{K}_3 + \mathbf{K}_4$ ，之后按照前述方法进行边界条件处理、求解即可。

在求解该系统时，依次按照 Robin 型、Neumann 型、Dirichlet 型边界条件进行处理，始终保证 Dirichlet 边界发挥作用，具体而言为：在 Robin 型对应边界刚度矩阵增加 $\int_{\Gamma_R} r u v ds$ ，载荷向量增加 $\int_{\Gamma_R} q v ds$ ；在 Neumann 型对应边界载荷向量增加 $\int_{\Gamma_N} p v ds$ ；对 Dirichlet 类型边界对应刚度矩阵采用“置 1”法，载荷向量为给定边界条件值。

3.5 通用 2 阶问题

考虑如下一般 2 阶问题：

$$\begin{cases} -\nabla \cdot (c \nabla u) + au = f, (x, y) \in \Omega \\ c \nabla u \cdot \vec{n} = p, \text{ on } \Gamma_N \subset \partial\Omega \\ c \nabla u \cdot \vec{n} + ru = q, \Gamma_R \subseteq \partial\Omega \\ u = g, \partial\Omega / (\Gamma_N \cup \Gamma_R) \end{cases} \quad (3.37)$$

其中， $c = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$ ，由前述推导可得：

$$\int_{\Omega} c \nabla u \cdot \nabla v dx dy - \int_{\partial\Omega} (c \nabla u \cdot \vec{n}) v ds + \int_{\Omega} a u v dx dy = \int_{\Omega} f v dx dy \quad (3.38)$$

在边界 $\partial\Omega/(\Gamma_N \cup \Gamma_R)$ 处，函数值由 $u = g$ 给定，所以，在该边界处可以取测试函数 $v(x) = 0$ ，由此可以得到：

$$\int_{\Omega} c \nabla u \cdot \nabla v dx dy + \int_{\Omega} a u v dx dy + \int_{\Gamma_R} r u v ds = \int_{\Omega} f v dx dy + \int_{\Gamma_N} p v ds + \int_{\Gamma_R} q v ds \quad (3.39)$$

其中,

$$c \nabla u \cdot \nabla v = c_{11}u_xv_x + c_{12}u_yv_x + c_{21}u_xv_y + c_{22}u_yv_y \quad (3.40)$$

- ▶ 令算法 6 中 $c = a, r = p = s = q = 0$, 得到矩阵 \mathbf{K}_0 ;
- ▶ 令算法 6 中 $c = c_{11}, r = p = 1, s = q = 0$, 得到矩阵 \mathbf{K}_1 ;
- ▶ 令算法 6 中 $c = c_{12}, s = p = 1, r = q = 0$, 得到矩阵 \mathbf{K}_2 ;
- ▶ 令算法 6 中 $c = c_{21}, r = q = 1, s = p = 0$, 得到矩阵 \mathbf{K}_3 ;
- ▶ 令算法 6 中 $c = c_{22}, r = p = 0, s = q = 1$, 得到矩阵 \mathbf{K}_4 ;

由此得到系统总刚度矩阵: $\mathbf{K} = \mathbf{K}_0 + \mathbf{K}_1 + \mathbf{K}_2 + \mathbf{K}_3 + \mathbf{K}_4$, 之后按照前述方法进行边界条件处理、求解即可。

在求解该系统时, 依次按照 Robin 型、Neumann 型、Dirichlet 型边界条件进行处理, 始终保证 Dirichlet 边界发挥作用, 具体而言为: 在 Robin 型对应边界刚度矩阵增加 $\int_{\Gamma_R} ruvsds$, 载荷向量增加 $\int_{\Gamma_R} qvds$; 在 Neumann 型对应边界载荷向量增加 $\int_{\Gamma_N} pvds$; 对 Dirichlet 类型边界对应刚度矩阵采用“置 1”法, 载荷向量为给定边界条件值。

3.6 误差计算

3.6.1 最大节点误差

最大节点误差即各个节点理论值与数值解差值的最大值:

$$\maxNodeError = [\text{res}(i) - \text{exact}(i)]_{\max} \quad (3.41)$$

3.6.2 无穷范数误差 L^∞

(1) 无穷范与无穷范误差定义

定义 L^∞ 空间为:

$$L^\infty(\Omega) = \left\{ v : \Omega \rightarrow R : \sup_{(x,y) \in \Omega} |u(x,y)| < \infty \right\} \quad (3.42)$$

无穷范 L^∞ 定义为:

$$\|u\|_\infty = \sup_{(x,y) \in \Omega} |u(x,y)|, u \in L^\infty(\Omega) \quad (3.43)$$

类比无穷范数, 无穷范误差定义为:

$$\|u - u_h\|_\infty = \sup_{(x,y) \in \Omega} |u(x,y) - u_h(x,y)| \quad (3.44)$$

(2) 无穷范误差计算方法

将 $u_h = \sum_{j=1}^{NN} u_j \phi_j$, 代入无穷范数, 利用网格单元节点编号矩阵与对应局部基函数即可得到:

$$\begin{aligned}
\|u - u_h\|_\infty &= \sup_{(x,y) \in \Omega} |u(x,y) - u_h(x,y)| \\
&= \max_{1 \leq n \leq NE} \max_{(x,y) \in E_n} |u(x,y) - u_h(x,y)| \\
&= \max_{1 \leq n \leq NE} \max_{(x,y) \in E_n} \left| u(x,y) - \sum_{j=1}^{NN} u_j \phi_j \right| \\
&= \max_{1 \leq n \leq NE} \max_{(x,y) \in E_n} \left| u(x,y) - \sum_{k=1}^{NN_{local}} u_{T_b(k,n)} \phi_{nk}(x,y) \right|
\end{aligned} \tag{3.45}$$

2D 椭圆方程无穷范数误差计算方法如算法 9 所示。

Algorithm 9 2D 椭圆方程无穷范数误差计算

Require: $error = 0$

```

1: for  $n = 1, \dots, N$  do
2:   Compute  $r_n \approx \max_{(x,y) \in E_n} |u(x,y) - w_n(x,y)|$ 
3:   if  $r_n > error$  then
4:      $error = r_n$ 
5:   end if
6: end for

```

3.6.3 L^2 范数误差

(1) L^2 范与 L^2 范误差定义

L^2 范定义为:

$$\|u\|_0 = \sqrt{\int_{\Omega} u^2 dx dy}, u \in L^\infty(\Omega) \tag{3.46}$$

类比 L^2 范, L^2 范误差定义为:

$$\|u - u_h\|_0 = \sqrt{\int_{\Omega} (u - u_h)^2 dx dy} \tag{3.47}$$

(2) L^2 范误差计算方法

将 $u_h = \sum_{j=1}^{NN} u_j \phi_j$, 代入无穷范数, 利用网格单元节点编号矩阵与对应局部基函数即可得到:

$$\begin{aligned}\|u - u_h\|_0 &= \sqrt{\int_{\Omega} (u - u_h)^2 dx dy} = \sqrt{\sum_{n=1}^{NE} \int_{E_n} (u - u_h)^2 dx dy} \\ &= \sqrt{\sum_{n=1}^{NE} \int_{E_n} \left(u - \sum_{j=1}^{NN} u_j \phi_j \right)^2 dx dy} \\ &= \sqrt{\sum_{n=1}^{NE} \int_{E_n} \left(u - \sum_{k=1}^{NN_{local}} u_{T_b(n,k)} \psi_{nk} \right)^2 dx dy}\end{aligned}\quad (3.48)$$

3.6.4 H^1 范数误差

(1) H^1 范与 H^1 范误差定义

H^1 范定义为:

$$\|u\|_1 = \sqrt{\int_{\Omega} \left(\frac{\partial u}{\partial x} \right)^2 dx dy + \int_{\Omega} \left(\frac{\partial u}{\partial y} \right)^2 dx dy}, u \in H^1(\Omega) \quad (3.49)$$

类比 H^1 范, H^1 范误差定义为:

$$\|u - u_h\|_1 = \sqrt{\int_{\Omega} \left(\frac{\partial (u - u_h)}{\partial x} \right)^2 dx dy + \int_{\Omega} \left(\frac{\partial (u - u_h)}{\partial y} \right)^2 dx dy} \quad (3.50)$$

(2) H^1 范误差计算方法

将 $u_h = \sum_{j=1}^{NN} u_j \phi_j$, 代入无穷范数, 利用网格单元节点编号矩阵与对应局部基函数即可得到:

$$\begin{aligned}\|u - u_h\|_{1,x} &= \sqrt{\int_{\Omega} \left(\frac{\partial (u - u_h)}{\partial x} \right)^2 dx dy} \\ &= \sqrt{\sum_{n=1}^{NE} \int_{E_n} \left(\frac{\partial (u - u_h)}{\partial x} \right)^2 dx dy} \\ &= \sqrt{\sum_{n=1}^{NE} \int_{E_n} \left(\frac{\partial u}{\partial x} - \sum_{j=1}^{NN} u_j \frac{\partial \phi_j}{\partial x} \right)^2 dx dy}\end{aligned}$$

$$= \sqrt{\sum_{n=1}^{NE} \int_{E_n} \left(\frac{\partial u}{\partial x} - \sum_{k=1}^{NN_{local}} u_{Tb(n,k)} \frac{\partial \psi_{nk}}{\partial x} \right)^2 dx dy} \quad (3.51)$$

同理能够得到 y 方向的误差：

$$\begin{aligned} \|u - u_h\|_{1,y} &= \sqrt{\int_{\Omega} \left(\frac{\partial (u - u_h)}{\partial y} \right)^2} \\ &= \sqrt{\sum_{n=1}^{NE} \int_{E_n} \left(\frac{\partial (u - u_h)}{\partial y} \right)^2 dx dy} \\ &= \sqrt{\sum_{n=1}^{NE} \int_{E_n} \left(\frac{\partial u}{\partial y} - \sum_{j=1}^{NN} u_j \frac{\partial \phi_j}{\partial y} \right)^2 dx dy} \\ &= \sqrt{\sum_{n=1}^{NE} \int_{E_n} \left(\frac{\partial u}{\partial y} - \sum_{k=1}^{NN_{local}} u_{Tb(n,k)} \frac{\partial \psi_{nk}}{\partial y} \right)^2 dx dy} \end{aligned} \quad (3.52)$$

将两部分合并即可得到完整的 H^1 范数误差：

$$\begin{aligned} H_1^2 &= \|u - u_h\|_1^2 \\ &= \|u - u_h\|_{1,x}^2 + \|u - u_h\|_{1,y}^2 \\ &= \sum_{n=1}^{NE} \int_{E_n} \left(\frac{\partial u}{\partial x} - \sum_{k=1}^{NN_{local}} u_{Tb(n,k)} \frac{\partial \psi_{nk}}{\partial x} \right)^2 dx dy \\ &\quad + \sum_{n=1}^{NE} \int_{E_n} \left(\frac{\partial u}{\partial y} - \sum_{k=1}^{NN_{local}} u_{Tb(n,k)} \frac{\partial \psi_{nk}}{\partial y} \right)^2 dx dy \end{aligned} \quad (3.53)$$

性质 → L^2 范与 H^1 范的特点

观察发现 L^2 范和 H^1 范具有相同特点：即其值分别为待求函数解析解与数值解的 0 阶导数与 1 阶导数的差，只不过在 2D 空间内函数导数由 x、y 两部分组成。因此可以定义 2 者误差范数的统一形式如下：

$$\sqrt{\sum_{n=1}^{NE} \int_{E_n} \left(\frac{\partial^{\alpha+\beta} u}{\partial x^{\alpha} \partial y^{\beta}} - \sum_{k=1}^{NN_{local}} u_{Tb(n,k)} \frac{\partial^{\alpha+\beta} \psi_{nk}}{\partial x^{\alpha} \partial y^{\beta}} \right)^2 dx dy} \quad (3.54)$$

可以看到当 $\alpha = 0, \beta = 0$ 时，所得结果为 L^2 范数误差；当 $\alpha = 1, \beta = 0$ 时为 $H_{1,x}$ 范数误差；当 $\alpha = 0, \beta = 1$ 时为 $H_{1,y}$ 范数误差。

2D 椭圆方程 L^2 或 H^1 范数误差计算方法如算法 10 所示。

Algorithm 10 2D 椭圆方程 L^2 或 H^1 范数误差计算

Require: $error = 0, \alpha, \beta$

- 1: **for** $n = 1, \dots, N$ **do**
- 2: Compute $error+ = \int_{E_n} \left(\frac{\partial^{\alpha+\beta} u}{\partial x^\alpha \partial y^\beta} - \sum_{k=1}^{NN_{local}} u_{Tb(n,k)} \frac{\partial^{\alpha+\beta} \psi_{nk}}{\partial x^\alpha \partial y^\beta} \right)^2 dx dy$
- 3: **end for**
- 4: $error = \sqrt{error}$

3.7 应用实例及程序实现

3.7.1 Example 1

例 3.1 【问题描述】以下述 2D 泊松方程的有限元求解对前述有限元方法进行验证，模型计算域为： $\Omega = [-1, -1] \times [1, 1]$ 。

$$\begin{cases} -\nabla \cdot (\nabla u) = -y(1-y)(1-x-\frac{x^2}{2})e^{x+y} - x(1-\frac{x}{2})(-3y-y^2)e^{x+y}, & (x, y) \in \Omega \\ u = -1.5y(1-y)e^{-1+y}, & x = -1 \\ u = 0.5y(1-y)e^{1+y}, & x = 1 \\ u = -2x(1-\frac{x}{2})e^{x-1}, & y = -1 \\ u = 0, & y = 1 \end{cases}$$

【说明】上述问题的解析解为： $u = xy(1 - \frac{x}{2})(1 - y)e^{x+y}$ ，可用于与有限元得到结果进行比对。

(1) 2D3 节点单元求解

1、网格划分

在 2D FEM 方法中网格划分之后对网格单元进行编号^[12-13]，常见的编号方向有纵向（ y 方向优先编号，图 3-1a）、横向（ x 方向优先编号，图 3-1b）之分，为方便计算演示，这里仅划分了 2 个单元，在此情况下，单元编号结果相同，编号方向对所求结果不会产生影响，这里以纵向编号为例。

根据划分的网格，产生对应的网格信息矩阵：

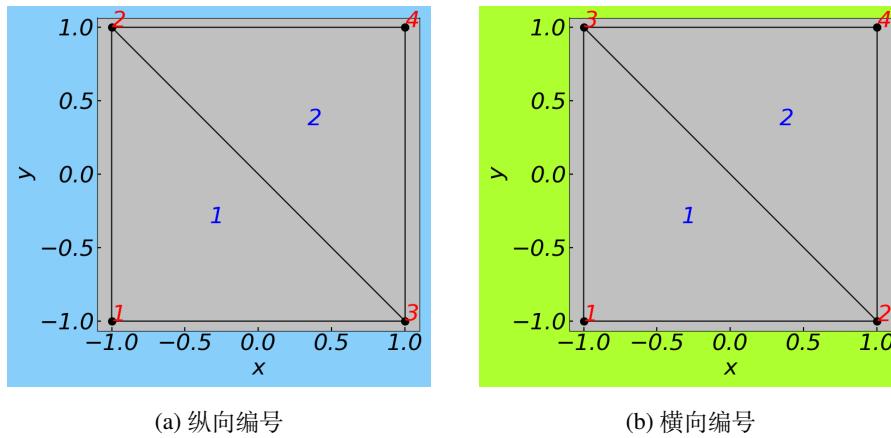


图 3-1 2D 三节点单元网格划分示意

- 几何网格节点坐标: $P = \begin{bmatrix} -1 & -1 \\ -1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix}$, 矩阵第 i 行表示第 i 个节点, 第 1 列为该节点 x 坐标, 第 2 列为该节点 y 坐标, 如第 2 个单元的 x 坐标可以表示为 $P[2][1] = -1$ 。
- 单元与节点关系矩阵: $T = \begin{bmatrix} 1 & 3 & 2 \\ 2 & 3 & 4 \end{bmatrix}$, 矩阵第 i 行表示第 i 个单元, 该行列表的大小为组成该单元节点的数量, 第 j 列表示该单元的第 j 个节点, 如第 2 个单元的第 3 个节点可以表示为 $T[2][3] = 4$, 这里局部单元编号与前述参考单元编号顺序一致, 按照逆时针 (counterclockwise, CCW) 方向编号。
- 有限元空间节点坐标矩阵: $Pb = P$ 。
- 有限元空间节点编号矩阵: $Tb = T$ 。
- 边界条件信息矩阵: $nodeBC = \begin{bmatrix} -1 & -1 & -1 & -1 \\ 1 & 3 & 4 & 2 \end{bmatrix}$, 该矩阵第 1 行为边界条件类型, 本算例中边界均为 Dirichlet 边界, 为与前述章节中保持一致, 这里仍然用 -1 表示该边界类型, 第 2 行为该边界类型对应的边界节点编号, 这里边界按照 CCW 方向进行排序。

2、刚度矩阵组装

根据当前划分网格节点数量可知矩阵的规模为: $NN \times NN = 4 \times 4 = 16$:

$$\mathbf{K} = \begin{bmatrix} K_{11} & K_{12} & K_{13} & K_{14} \\ K_{21} & K_{22} & K_{23} & K_{24} \\ K_{31} & K_{32} & K_{33} & K_{34} \\ K_{41} & K_{42} & K_{43} & K_{44} \end{bmatrix} = \begin{bmatrix} K_{11}^1 & K_{12}^1 & K_{13}^1 & 0 \\ K_{21}^1 & K_{22}^1 + \mathbf{K}_{11}^2 & K_{23}^1 + \mathbf{K}_{12}^2 & \mathbf{K}_{13}^2 \\ K_{31}^1 & K_{32}^1 + \mathbf{K}_{21}^2 & K_{33}^1 + \mathbf{K}_{22}^2 & \mathbf{K}_{23}^2 \\ 0 & \mathbf{K}_{31}^2 & \mathbf{K}_{32}^2 & \mathbf{K}_{33}^2 \end{bmatrix} \quad (3.55)$$

- 第 1 个单元刚度阵计算

第 1 个单元为三角形，组成该单元的节点数为 3，各节点的全局编号信息在矩阵 T 的第 1 行，全局编号为：(1,3,2)，组成该单元的局部刚度阵各元素详细计算过程如下：

$$K_{3 \times 3}^1 = \begin{bmatrix} K_{11}^1 & K_{12}^1 & K_{13}^1 \\ K_{21}^1 & K_{22}^1 & K_{23}^1 \\ K_{31}^1 & K_{32}^1 & K_{33}^1 \end{bmatrix} \quad (3.56)$$

✓ Jacobian 矩阵

$$\mathbf{J}^1 = \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \xi} & \frac{\partial N_3}{\partial \xi} \\ \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \frac{\partial N_3}{\partial \eta} \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \\ 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} -1 & -1 \\ 1 & -1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \quad (3.57)$$

✓ Jacobian 矩阵的行列式：

$$|\mathbf{J}^1| = 4 \quad (3.58)$$

✓ Jacobian 矩阵的逆

$$[\mathbf{J}^1]^{-1} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \quad (3.59)$$

✓ 组成该单元的各个元素

由前述章节可知参考单元面积可表示为 $\int_{\xi=0}^1 \int_{\eta=0}^{1-\xi} 1 d\xi d\eta = 0.5$

$$\begin{aligned} K_{11}^1 &= \int_{\Omega^e} \left(\frac{\partial \psi_1}{\partial x} \frac{\partial \psi_1}{\partial x} + \frac{\partial \psi_1}{\partial y} \frac{\partial \psi_1}{\partial y} \right) |\mathbf{J}^1| d\xi d\eta \\ &= \int_{\Omega^e} \left\{ \left(J_{11}^{-1} \frac{\partial N_1}{\partial \xi} + J_{12}^{-1} \frac{\partial N_1}{\partial \eta} \right) \left(J_{11}^{-1} \frac{\partial N_1}{\partial \xi} + J_{12}^{-1} \frac{\partial N_1}{\partial \eta} \right) \right. \\ &\quad \left. + \left(J_{21}^{-1} \frac{\partial N_1}{\partial \xi} + J_{22}^{-1} \frac{\partial N_1}{\partial \eta} \right) \left(J_{21}^{-1} \frac{\partial N_1}{\partial \xi} + J_{22}^{-1} \frac{\partial N_1}{\partial \eta} \right) \right\} |\mathbf{J}^1| d\xi d\eta \\ &= \int_{\xi=0}^1 \int_{\eta=0}^{1-\xi} 2 d\xi d\eta = 1 \end{aligned} \quad (3.60)$$

$$\begin{aligned} K_{12}^1 = K_{21}^1 &= \int_{\Omega^e} \left(\frac{\partial \psi_2}{\partial x} \frac{\partial \psi_1}{\partial x} + \frac{\partial \psi_2}{\partial y} \frac{\partial \psi_1}{\partial y} \right) |\mathbf{J}^1| d\xi d\eta \\ &= \int_{\Omega^e} \left\{ \left(J_{11}^{-1} \frac{\partial N_2}{\partial \xi} + J_{12}^{-1} \frac{\partial N_2}{\partial \eta} \right) \left(J_{11}^{-1} \frac{\partial N_1}{\partial \xi} + J_{12}^{-1} \frac{\partial N_1}{\partial \eta} \right) \right. \\ &\quad \left. + \left(J_{21}^{-1} \frac{\partial N_2}{\partial \xi} + J_{22}^{-1} \frac{\partial N_2}{\partial \eta} \right) \left(J_{21}^{-1} \frac{\partial N_1}{\partial \xi} + J_{22}^{-1} \frac{\partial N_1}{\partial \eta} \right) \right\} |\mathbf{J}^1| d\xi d\eta \\ &= \int_{\xi=0}^1 \int_{\eta=0}^{1-\xi} -1 d\xi d\eta = -0.5 \end{aligned} \quad (3.61)$$

$$\begin{aligned}
K_{13}^1 &= K_{31}^1 = \int_{\Omega^e} \left(\frac{\partial \psi_3}{\partial x} \frac{\partial \psi_1}{\partial x} + \frac{\partial \psi_3}{\partial y} \frac{\partial \psi_1}{\partial y} \right) |\mathbf{J}^1| d\xi d\eta \\
&= \int_{\Omega^e} \left\{ \left(J_{11}^{-1} \frac{\partial N_3}{\partial \xi} + J_{12}^{-1} \frac{\partial N_3}{\partial \eta} \right) \left(J_{11}^{-1} \frac{\partial N_1}{\partial \xi} + J_{12}^{-1} \frac{\partial N_1}{\partial \eta} \right) \right. \\
&\quad \left. + \left(J_{21}^{-1} \frac{\partial N_3}{\partial \xi} + J_{22}^{-1} \frac{\partial N_3}{\partial \eta} \right) \left(J_{21}^{-1} \frac{\partial N_1}{\partial \xi} + J_{22}^{-1} \frac{\partial N_1}{\partial \eta} \right) \right\} |\mathbf{J}^1| d\xi d\eta \\
&= \int_{\xi=0}^1 \int_{\eta=0}^{1-\xi} -1 d\xi d\eta = -0.5
\end{aligned} \tag{3.62}$$

$$\begin{aligned}
K_{22}^1 &= \int_{\Omega^e} \left(\frac{\partial \psi_2}{\partial x} \frac{\partial \psi_2}{\partial x} + \frac{\partial \psi_2}{\partial y} \frac{\partial \psi_2}{\partial y} \right) |\mathbf{J}^1| d\xi d\eta \\
&= \int_{\Omega^e} \left\{ \left(J_{11}^{-1} \frac{\partial N_2}{\partial \xi} + J_{12}^{-1} \frac{\partial N_2}{\partial \eta} \right) \left(J_{11}^{-1} \frac{\partial N_2}{\partial \xi} + J_{12}^{-1} \frac{\partial N_2}{\partial \eta} \right) \right. \\
&\quad \left. + \left(J_{21}^{-1} \frac{\partial N_2}{\partial \xi} + J_{22}^{-1} \frac{\partial N_2}{\partial \eta} \right) \left(J_{21}^{-1} \frac{\partial N_2}{\partial \xi} + J_{22}^{-1} \frac{\partial N_2}{\partial \eta} \right) \right\} |\mathbf{J}^1| d\xi d\eta \\
&= \int_{\xi=0}^1 \int_{\eta=0}^{1-\xi} 1 d\xi d\eta = 0.5
\end{aligned} \tag{3.63}$$

$$\begin{aligned}
K_{23}^1 &= K_{32}^1 = \int_{\Omega^e} \left(\frac{\partial \psi_3}{\partial x} \frac{\partial \psi_2}{\partial x} + \frac{\partial \psi_3}{\partial y} \frac{\partial \psi_2}{\partial y} \right) |\mathbf{J}^1| d\xi d\eta \\
&= \int_{\Omega^e} \left\{ \left(J_{11}^{-1} \frac{\partial N_3}{\partial \xi} + J_{12}^{-1} \frac{\partial N_3}{\partial \eta} \right) \left(J_{11}^{-1} \frac{\partial N_2}{\partial \xi} + J_{12}^{-1} \frac{\partial N_2}{\partial \eta} \right) \right. \\
&\quad \left. + \left(J_{21}^{-1} \frac{\partial N_3}{\partial \xi} + J_{22}^{-1} \frac{\partial N_3}{\partial \eta} \right) \left(J_{21}^{-1} \frac{\partial N_2}{\partial \xi} + J_{22}^{-1} \frac{\partial N_2}{\partial \eta} \right) \right\} |\mathbf{J}^1| d\xi d\eta \\
&= \int_{\xi=0}^1 \int_{\eta=0}^{1-\xi} 0 d\xi d\eta = 0
\end{aligned} \tag{3.64}$$

$$\begin{aligned}
K_{33}^1 &= \int_{\Omega^e} \left(\frac{\partial \psi_3}{\partial x} \frac{\partial \psi_3}{\partial x} + \frac{\partial \psi_3}{\partial y} \frac{\partial \psi_3}{\partial y} \right) |\mathbf{J}^1| d\xi d\eta \\
&= \int_{\Omega^e} \left\{ \left(J_{11}^{-1} \frac{\partial N_3}{\partial \xi} + J_{12}^{-1} \frac{\partial N_3}{\partial \eta} \right) \left(J_{11}^{-1} \frac{\partial N_3}{\partial \xi} + J_{12}^{-1} \frac{\partial N_3}{\partial \eta} \right) \right. \\
&\quad \left. + \left(J_{21}^{-1} \frac{\partial N_3}{\partial \xi} + J_{22}^{-1} \frac{\partial N_3}{\partial \eta} \right) \left(J_{21}^{-1} \frac{\partial N_3}{\partial \xi} + J_{22}^{-1} \frac{\partial N_3}{\partial \eta} \right) \right\} |\mathbf{J}^1| d\xi d\eta \\
&= \int_{\xi=0}^1 \int_{\eta=0}^{1-\xi} 1 d\xi d\eta = 0.5
\end{aligned} \tag{3.65}$$

由此可以得到单元 1 的局部刚度矩阵：

$$K_{3 \times 3}^1 = \begin{bmatrix} 1 & -0.5 & -0.5 \\ -0.5 & 0.5 & 0 \\ -0.5 & 0 & 0.5 \end{bmatrix} \tag{3.66}$$

将所得局部刚度矩阵添加至对应的整体刚度矩阵 \mathbf{K} 中，添加时局部刚度矩阵到全局刚度矩阵各个元素的位置由节点全局编号决定，如对局部刚度矩阵元素 K_{12}^1 而言，该元素值计算由全局编号节点 (1,3) 得到，所以其在全局刚度阵中的位置为 (1,3)。

$$\mathbf{K} = \begin{bmatrix} K_{11}^1 & K_{12}^1 & K_{13}^1 & 0 \\ K_{21}^1 & K_{22}^1 & K_{23}^1 & 0 \\ K_{31}^1 & K_{32}^1 & K_{33}^1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & -0.5 & -0.5 & 0 \\ -0.5 & 1 & 0 & 0 \\ -0.5 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (3.67)$$

• 第 2 个单元刚度阵计算

第 2 个单元为三角形，组成该单元的节点数为 3，各节点的全局编号信息在矩阵 \mathbf{T} 的第 2 行，全局编号为：(2,3,4)，组成该单元的局部刚度阵各元素详细计算过程如下：

$$K_{3 \times 3}^2 = \begin{bmatrix} K_{11}^2 & K_{12}^2 & K_{13}^2 \\ K_{21}^2 & K_{22}^2 & K_{23}^2 \\ K_{31}^2 & K_{32}^2 & K_{33}^2 \end{bmatrix} \quad (3.68)$$

✓ Jacobian 矩阵

$$\mathbf{J}^2 = \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \xi} & \frac{\partial N_3}{\partial \xi} \\ \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \frac{\partial N_3}{\partial \eta} \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 2 & -2 \\ 0 & 2 \end{bmatrix} \quad (3.69)$$

✓ Jacobian 矩阵的行列式：

$$|\mathbf{J}^2| = 4 \quad (3.70)$$

✓ Jacobian 矩阵的逆

$$[\mathbf{J}^2]^{-1} = \begin{bmatrix} 0 & 0.5 \\ -0.5 & 0.5 \end{bmatrix} \quad (3.71)$$

✓ 组成该单元的各个元素

$$\begin{aligned} K_{11}^2 &= \int_{\Omega^e} \left(\frac{\partial \psi_1}{\partial x} \frac{\partial \psi_1}{\partial x} + \frac{\partial \psi_1}{\partial y} \frac{\partial \psi_1}{\partial y} \right) |\mathbf{J}^2| d\xi d\eta \\ &= \int_{\Omega^e} \left\{ \left(J_{11}^{-1} \frac{\partial N_1}{\partial \xi} + J_{12}^{-1} \frac{\partial N_1}{\partial \eta} \right) \left(J_{11}^{-1} \frac{\partial N_1}{\partial \xi} + J_{12}^{-1} \frac{\partial N_1}{\partial \eta} \right) \right. \\ &\quad \left. + \left(J_{21}^{-1} \frac{\partial N_1}{\partial \xi} + J_{22}^{-1} \frac{\partial N_1}{\partial \eta} \right) \left(J_{21}^{-1} \frac{\partial N_1}{\partial \xi} + J_{22}^{-1} \frac{\partial N_1}{\partial \eta} \right) \right\} |\mathbf{J}^2| d\xi d\eta \\ &= \int_{\xi=0}^1 \int_{\eta=0}^{1-\xi} 1 d\xi d\eta = 0.5 \end{aligned} \quad (3.72)$$

$$\begin{aligned}
K_{12}^2 = K_{21}^2 &= \int_{\Omega^e} \left(\frac{\partial \psi_2}{\partial x} \frac{\partial \psi_1}{\partial x} + \frac{\partial \psi_2}{\partial y} \frac{\partial \psi_1}{\partial y} \right) |\mathbf{J}^2| d\xi d\eta \\
&= \int_{\Omega^e} \left\{ \left(J_{11}^{-1} \frac{\partial N_2}{\partial \xi} + J_{12}^{-1} \frac{\partial N_2}{\partial \eta} \right) \left(J_{11}^{-1} \frac{\partial N_1}{\partial \xi} + J_{12}^{-1} \frac{\partial N_1}{\partial \eta} \right) \right. \\
&\quad \left. + \left(J_{21}^{-1} \frac{\partial N_2}{\partial \xi} + J_{22}^{-1} \frac{\partial N_2}{\partial \eta} \right) \left(J_{21}^{-1} \frac{\partial N_1}{\partial \xi} + J_{22}^{-1} \frac{\partial N_1}{\partial \eta} \right) \right\} |\mathbf{J}^2| d\xi d\eta \\
&= \int_{\xi=0}^1 \int_{\eta=0}^{1-\xi} 0 d\xi d\eta = 0
\end{aligned} \tag{3.73}$$

$$\begin{aligned}
K_{13}^2 = K_{31}^2 &= \int_{\Omega^e} \left(\frac{\partial \psi_3}{\partial x} \frac{\partial \psi_1}{\partial x} + \frac{\partial \psi_3}{\partial y} \frac{\partial \psi_1}{\partial y} \right) |\mathbf{J}^2| d\xi d\eta \\
&= \int_{\Omega^e} \left\{ \left(J_{11}^{-1} \frac{\partial N_3}{\partial \xi} + J_{12}^{-1} \frac{\partial N_3}{\partial \eta} \right) \left(J_{11}^{-1} \frac{\partial N_1}{\partial \xi} + J_{12}^{-1} \frac{\partial N_1}{\partial \eta} \right) \right. \\
&\quad \left. + \left(J_{21}^{-1} \frac{\partial N_3}{\partial \xi} + J_{22}^{-1} \frac{\partial N_3}{\partial \eta} \right) \left(J_{21}^{-1} \frac{\partial N_1}{\partial \xi} + J_{22}^{-1} \frac{\partial N_1}{\partial \eta} \right) \right\} |\mathbf{J}^2| d\xi d\eta \\
&= \int_{\xi=0}^1 \int_{\eta=0}^{1-\xi} -1 d\xi d\eta = -0.5
\end{aligned} \tag{3.74}$$

$$\begin{aligned}
K_{22}^1 &= \int_{\Omega^e} \left(\frac{\partial \psi_2}{\partial x} \frac{\partial \psi_2}{\partial x} + \frac{\partial \psi_2}{\partial y} \frac{\partial \psi_2}{\partial y} \right) |\mathbf{J}^2| d\xi d\eta \\
&= \int_{\Omega^e} \left\{ \left(J_{11}^{-1} \frac{\partial N_2}{\partial \xi} + J_{12}^{-1} \frac{\partial N_2}{\partial \eta} \right) \left(J_{11}^{-1} \frac{\partial N_2}{\partial \xi} + J_{12}^{-1} \frac{\partial N_2}{\partial \eta} \right) \right. \\
&\quad \left. + \left(J_{21}^{-1} \frac{\partial N_2}{\partial \xi} + J_{22}^{-1} \frac{\partial N_2}{\partial \eta} \right) \left(J_{21}^{-1} \frac{\partial N_2}{\partial \xi} + J_{22}^{-1} \frac{\partial N_2}{\partial \eta} \right) \right\} |\mathbf{J}^2| d\xi d\eta \\
&= \int_{\xi=0}^1 \int_{\eta=0}^{1-\xi} 1 d\xi d\eta = 0.5
\end{aligned} \tag{3.75}$$

$$\begin{aligned}
K_{23}^2 = K_{32}^2 &= \int_{\Omega^e} \left(\frac{\partial \psi_3}{\partial x} \frac{\partial \psi_2}{\partial x} + \frac{\partial \psi_3}{\partial y} \frac{\partial \psi_2}{\partial y} \right) |\mathbf{J}^2| d\xi d\eta \\
&= \int_{\Omega^e} \left\{ \left(J_{11}^{-1} \frac{\partial N_3}{\partial \xi} + J_{12}^{-1} \frac{\partial N_3}{\partial \eta} \right) \left(J_{11}^{-1} \frac{\partial N_2}{\partial \xi} + J_{12}^{-1} \frac{\partial N_2}{\partial \eta} \right) \right. \\
&\quad \left. + \left(J_{21}^{-1} \frac{\partial N_3}{\partial \xi} + J_{22}^{-1} \frac{\partial N_3}{\partial \eta} \right) \left(J_{21}^{-1} \frac{\partial N_2}{\partial \xi} + J_{22}^{-1} \frac{\partial N_2}{\partial \eta} \right) \right\} |\mathbf{J}^2| d\xi d\eta \\
&= \int_{\xi=0}^1 \int_{\eta=0}^{1-\xi} -1 d\xi d\eta = -0.5
\end{aligned} \tag{3.76}$$

$$\begin{aligned}
K_{33}^1 &= \int_{\Omega^e} \left(\frac{\partial \psi_3}{\partial x} \frac{\partial \psi_3}{\partial x} + \frac{\partial \psi_3}{\partial y} \frac{\partial \psi_3}{\partial y} \right) |\mathbf{J}^2| d\xi d\eta \\
&= \int_{\Omega^e} \left\{ \left(J_{11}^{-1} \frac{\partial N_3}{\partial \xi} + J_{12}^{-1} \frac{\partial N_3}{\partial \eta} \right) \left(J_{11}^{-1} \frac{\partial N_3}{\partial \xi} + J_{12}^{-1} \frac{\partial N_3}{\partial \eta} \right) \right. \\
&\quad \left. + \left(J_{21}^{-1} \frac{\partial N_3}{\partial \xi} + J_{22}^{-1} \frac{\partial N_3}{\partial \eta} \right) \left(J_{21}^{-1} \frac{\partial N_3}{\partial \xi} + J_{22}^{-1} \frac{\partial N_3}{\partial \eta} \right) \right\} |\mathbf{J}^2| d\xi d\eta \\
&= \int_{\xi=0}^1 \int_{\eta=0}^{1-\xi} 2 d\xi d\eta = 1
\end{aligned} \tag{3.77}$$

由此可以得到单元 2 的局部刚度矩阵:

$$K_{3 \times 3}^2 = \begin{bmatrix} 0.5 & 0 & -0.5 \\ 0 & 0.5 & -0.5 \\ -0.5 & -0.5 & 1 \end{bmatrix} \tag{3.78}$$

将所得局部刚度矩阵添加至对应的整体刚度矩阵 \mathbf{K} 中, 即可得到本算例整体刚度矩阵:

$$\mathbf{K} = \begin{bmatrix} 1 & -0.5 & -0.5 & 0 \\ -0.5 & 1 & 0 & -0.5 \\ -0.5 & 0 & 1 & -0.5 \\ 0 & -0.5 & -0.5 & 1 \end{bmatrix} \tag{3.79}$$

3、载荷向量组装

载荷向量与矩阵相对应, 其规模为有限元网格节点数量, 对于本例而言为: $NN = 4$ 。

$$\mathbf{F} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} = \begin{bmatrix} F_1^1 \\ F_2^1 + \textcolor{red}{F}_1^2 \\ F_3^1 + \textcolor{red}{F}_2^2 \\ \textcolor{red}{F}_3^2 \end{bmatrix} \tag{3.80}$$

- 第 1 个单元载荷向量计算

$$F_{3 \times 1}^1 = \begin{bmatrix} F_1^1 \\ F_2^1 \\ F_3^1 \end{bmatrix} \tag{3.81}$$

由前所示可以知道载荷向量的计算也需要转换到参考元上进行, 对于第 1 个单元而言求解所用的 Jacobian 矩阵及其相关信息在求解局部刚度阵时已经得到, 所以可以直接对载荷向量各个元素进行求解。第 1 个单元载荷向量各元素结果如下:

$$F_1^1 = \int_{\Omega^e} f(\xi, \eta) \psi_1(\xi, \eta) |\mathbf{J}^1| d\xi d\eta = 0.2802 \tag{3.82}$$

$$F_2^1 = \int_{\Omega^e} f(\xi, \eta) \psi_2(\xi, \eta) |\mathbf{J}^1| d\xi d\eta = 0.0600 \quad (3.83)$$

$$F_3^1 = \int_{\Omega^e} f(\xi, \eta) \psi_3(\xi, \eta) |\mathbf{J}^1| d\xi d\eta = -0.3273 \quad (3.84)$$

由此得到单元 1 的局部向量:

$$F_{3 \times 1}^1 = \begin{bmatrix} F_1^1 \\ F_2^1 \\ F_3^1 \end{bmatrix} = \begin{bmatrix} 0.2802 \\ 0.0600 \\ -0.3273 \end{bmatrix} \quad (3.85)$$

将得到的 $F_{3 \times 1}^1$ 添加至向量 \mathbf{F} 中:

$$\mathbf{F} = F_{3 \times 1}^1 = \begin{bmatrix} F_1^1 \\ F_3^1 \\ F_2^1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.2802 \\ -0.3273 \\ 0.0600 \\ 0 \end{bmatrix} \quad (3.86)$$

• 第 2 个单元载荷向量计算

$$F_1^2 = \int_{\Omega^e} f(\xi, \eta) \psi_1(\xi, \eta) |\mathbf{J}^2| d\xi d\eta = -0.3805 \quad (3.87)$$

$$F_2^2 = \int_{\Omega^e} f(\xi, \eta) \psi_2(\xi, \eta) |\mathbf{J}^2| d\xi d\eta = -0.0152 \quad (3.88)$$

$$F_3^2 = \int_{\Omega^e} f(\xi, \eta) \psi_3(\xi, \eta) |\mathbf{J}^2| d\xi d\eta = 1.4655 \quad (3.89)$$

由此得到单元 2 的局部向量:

$$F_{3 \times 1}^2 = \begin{bmatrix} F_1^2 \\ F_2^2 \\ F_3^2 \end{bmatrix} = \begin{bmatrix} -0.3805 \\ -0.0152 \\ 1.4655 \end{bmatrix} \quad (3.90)$$

将得到的 $F_{3 \times 1}^2$ 添加至向量 \mathbf{F} 中:

$$\mathbf{F} = \begin{bmatrix} 0.2802 \\ -0.7078 \\ 0.0448 \\ 1.4655 \end{bmatrix} \quad (3.91)$$

所得载荷向量即为当前网格划分的载荷向量 \mathbf{F} 。

4、边界条件处理及线性系统求解

采用前述章节中“置 1”法对本算例中边界进行处理，经过边界处理后可以得到线性系统及各节点函数值：

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} 0.4060 \\ 0 \\ -1 \\ 0 \end{bmatrix} \rightarrow \mathbf{u} = \begin{bmatrix} 0.4060 \\ 0 \\ -1 \\ 0 \end{bmatrix} \quad (3.92)$$

5、后处理

求解边界处理后的线性系统，所得结果与解析解如图 3-2 所示，在当前网格划分情况下所得有限元解与解析解具有较大差距，细化网格后将有效减小这种差距。

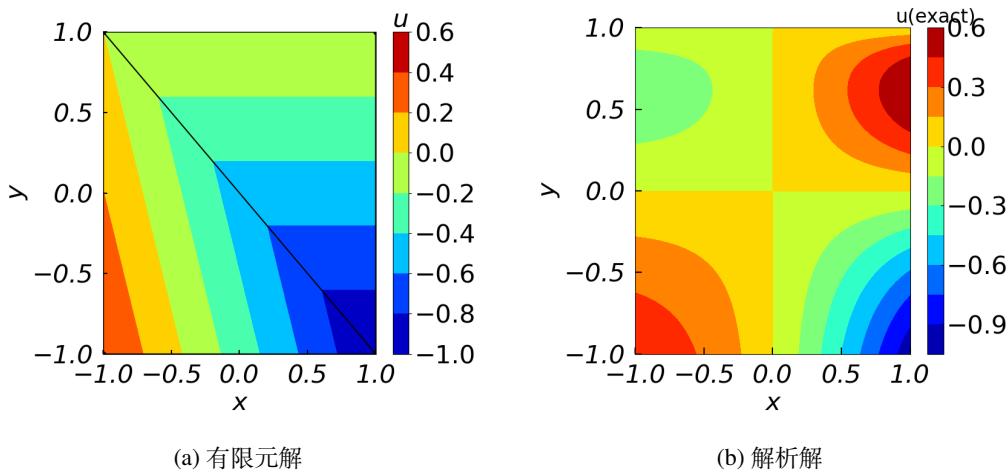


图 3-2 Example 1 三节点单元有限元解与解析解

表 3-1 给出了不同空间步长下，2D FEM 采用线性三角形网格单元时各类误差范数的结果参考。

表 3-1 Example 1 三节点单元不同空间步长下各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	1.7300×10^{-3}	2.3620×10^{-2}	6.8300×10^{-3}	1.8774×10^{-1}
1/16	4.3521×10^{-4}	6.3422×10^{-3}	1.7189×10^{-3}	9.4167×10^{-2}
1/32	1.0902×10^{-4}	1.6430×10^{-3}	4.3049×10^{-4}	4.7121×10^{-2}
1/64	2.7270×10^{-5}	4.1810×10^{-4}	1.0767×10^{-4}	2.3565×10^{-2}

(2) 2D6 节点单元求解

1、网格划分

构造高阶三角形单元简单易行的方法就是在各个单元边上加一个中点，同样有纵向和横向编号之分（仍以纵向为例，与前述线性元保持一致），2D 六节点单元网格划分示意如图 3-3 所示。

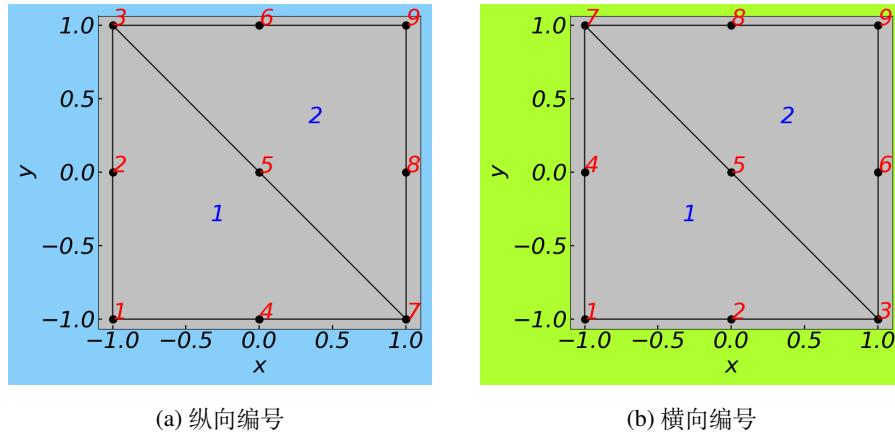


图 3-3 2D 六节点单元网格划分示意

根据划分的网格，产生对应的网格信息矩阵：

- 几何网格节点坐标： $P = \begin{bmatrix} -1 & -1 \\ -1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix}$ ，矩阵第 i 行表示第 i 个节点，第 1 列为该节点 x 坐标，第 2 列为该节点 y 坐标，如第 2 个单元的 x 坐标可以表示为 $P[2][1] = -1$ 。

- 单元与节点关系矩阵： $T = \begin{bmatrix} 1 & 3 & 2 \\ 2 & 3 & 4 \end{bmatrix}$ ，矩阵第 i 行表示第 i 个单元，该行列表的大小为组成该单元节点的数量，第 j 列表示该单元的第 j 个节点，如第 2 个单元的第 3 个节点可以表示为 $T[2][3] = 4$ ，这里局部单元编号与前述参考单元编号顺序一致，按照逆时针（counterclockwise, CCW）方向编号。

- 有限元空间节点坐标矩阵： $Pb = \begin{bmatrix} -1 & -1 & -1 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}^T$ ，行数为有限元节点数，每行第 1 列为有限元节点 x 坐标，第 2 列为 y 坐标，高阶单元对应有限元节点较低阶单元大量增加。

- 有限元空间节点编号矩阵： $Tb = \begin{bmatrix} 1 & 7 & 3 & 4 & 5 & 2 \\ 3 & 7 & 9 & 5 & 8 & 6 \end{bmatrix}$ 。

- 边界条件信息矩阵: $nodeBC = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ 1 & 4 & 7 & 8 & 9 & 6 & 3 & 2 \end{bmatrix}$, 该矩阵第 1 行为边界条件类型, 本算例中边界均为 Dirichlet 边界, 为与前述章节中保持一致, 这里仍然用 -1 表示该边界类型, 第 2 行为该边界类型对应的边界节点编号, 这里边界按照 CCW 方向进行排序。

2、刚度矩阵组装

根据当前划分网格节点数量可知矩阵的规模为: $NN \times NN = 9 \times 9 = 81$, 这里直接给出在图 3-3a 所示网格划分时所得系统刚度矩阵 \mathbf{K} 。

$$\mathbf{K} = \begin{bmatrix} 1 & -0.6667 & 0.1667 & -0.6667 & 0 & 0 & 0.1667 & 0 & 0 \\ -0.6667 & 2.6667 & -0.6667 & 0 & -1.3333 & 0 & 0 & 0 & 0 \\ 0.1667 & -0.6667 & 1 & 0 & 0 & -0.6667 & 0 & 0 & 0.1667 \\ -0.6667 & 0 & 0 & 2.6667 & -1.3333 & 0 & -0.6667 & 0 & 0 \\ 0 & -1.3333 & 0 & -1.3333 & 5.3333 & -1.3333 & 0 & -1.3333 & 0 \\ 0 & 0 & -0.6667 & 0 & -1.3333 & 2.6667 & 0 & 0 & -0.6667 \\ 0.1667 & 0 & 0 & -0.6667 & 0 & 0 & 1 & -0.6667 & 0.1667 \\ 0 & 0 & 0 & 0 & -1.3333 & 0 & -0.6667 & 2.6667 & -0.6667 \\ 0 & 0 & 0.1667 & 0 & 0 & -0.6667 & 0.1667 & -0.6667 & 1 \end{bmatrix} \quad (3.93)$$

3、载荷向量组装

载荷向量与矩阵相对应, 其规模为有限元网格节点数量, 对于本例而言为: $NN = 9$, 图 3-3a 网格划分情况下系统载荷向量如下:

$$\mathbf{F} = \begin{bmatrix} 0.1173 & 0.0301 & -0.8264 & 0.2957 & -0.2624 & 0.4696 & -0.2850 & 0.6263 & 0.9175 \end{bmatrix}^T \quad (3.94)$$

4、边界条件处理及线性系统求解

这里边界条件仍然为 Dirichlet 边界类型, 处理方法与 2D 三节点单元相同, 仍然采用“置 1”法, 经过边界处理后得到对应刚度矩阵与载荷向量。

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1.3333 & 0 & -1.3333 & 5.3333 & -1.3333 & 0 & -1.3333 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \end{bmatrix} = \begin{bmatrix} 0.4060 \\ 0 \\ 0 \\ 0 \\ -0.2624 \\ 0 \\ -1 \\ 0 \\ 0 \end{bmatrix} \quad (3.95)$$

根据网格划分情况，在采用 2D 六节点单元时，仅有节点 5 为内部节点，其余均为边界点，本算例模型中边界节点采用“置 1”法处理，因此所得线性系统中对应刚度矩阵除节点 5 外均为主对角线均为 1，其余元素均为 0，且载荷向量除节点 5 外，均为边界条件给定值。可以看到所得线性系统符合该规律，容易得到各节点函数值为：

$$\mathbf{u} = \begin{bmatrix} 0.4060 & 0 & 0 & 0 & -0.0492 & 0 & -1 & 0 & 0 \end{bmatrix}^T \quad (3.96)$$

5、后处理

所得结果与解析解如图 6-4 所示，表 3-2 给出了不同空间步长下 2D 六节点单元各类误差范数参考。

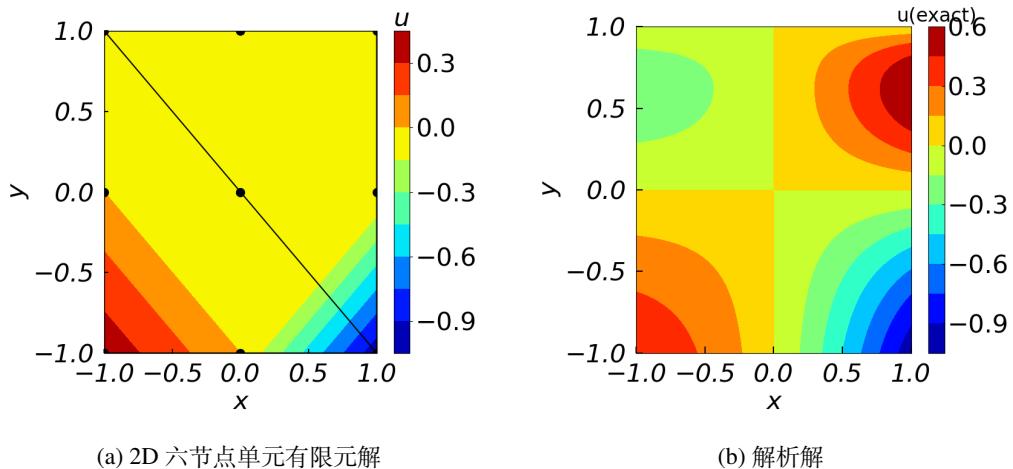


图 3-4 Example 1 六节点单元有限元解与解析解

表 3-2 Example 1 六节点单元不同空间步长下各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	1.7836×10^{-5}	3.3678×10^{-4}	1.1705×10^{-4}	8.9192×10^{-3}
1/16	1.3528×10^{-6}	4.4273×10^{-5}	1.4637×10^{-5}	2.2414×10^{-3}
1/32	9.3471×10^{-8}	5.6752×10^{-6}	1.8289×10^{-6}	5.6131×10^{-4}
1/64	6.1462×10^{-9}	7.1839×10^{-7}	2.2853×10^{-7}	1.4042×10^{-4}

(3) 2D4 节点单元

在有限元方法中采用 2D4 节点单元进行数值计算过程与前述采用 2D3 节点、2D6 节点单元方法仿真过程完全一致，区别仅在于网格划分产生的各节点单元坐标、组成单

元的各节点不同，进而导致 Jacobian 矩阵、各单元刚度矩阵、各单元载荷向量的计算方法也有所不同。由此产生不同的线性系统，进而所得有限元仿真结果有所不同。

与三角形单元编号方向相似，四边形单元编号也有横向与纵向之分，如图 3-5 所示为采用四边形单元组成的网格结构及其编号方式示意。线性四边形单元与线性三角形单元中网格节点的坐标与编号方式是完全一致的，区别在于组成单元的节点数不同，根据图 3-5a 所示网格划分示意可以得到对应信息矩阵如下，其中个信息矩阵含义同前。

- 几何网格节点坐标： $P = \begin{bmatrix} -1 & -1 & -1 & 0 & 0 & 0 & 1 & 1 & 1 \\ -1 & 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 \end{bmatrix}^T$
- 单元与节点关系矩阵： $T = \begin{bmatrix} 1 & 4 & 5 & 2 \\ 2 & 5 & 6 & 3 \\ 4 & 7 & 8 & 5 \\ 5 & 8 & 9 & 6 \end{bmatrix}$
- 有限元空间节点坐标矩阵： $Pb = P$
- 有限元空间节点编号矩阵： $Tb = T$
- 边界条件信息矩阵： $nodeBC = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ 1 & 4 & 7 & 8 & 9 & 6 & 3 & 2 \end{bmatrix}$

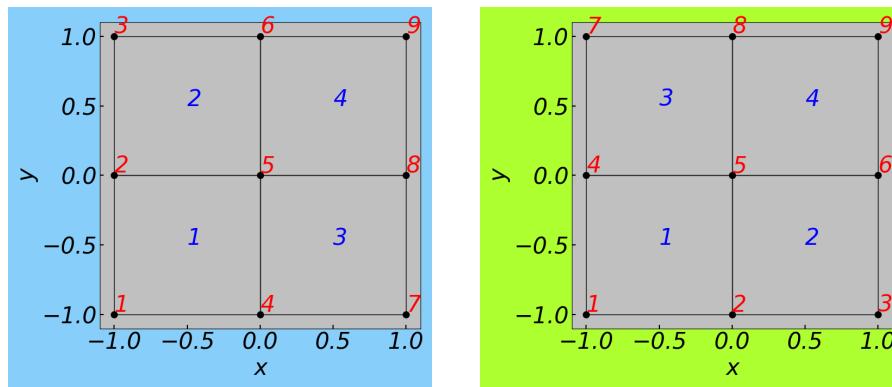


图 3-5 2D4 节点单元网格划分示意

之后即可根据前述方法对刚度矩阵、载荷向量进行组装、边界处理、线性系统求解得到有限元解，并对相关误差进行分析，对所得结果进行后处理等。需要注意的是，在进行各单元刚度矩阵元素计算时对应 Jacobian 矩阵及相关基函数为当前单元类型对应的信息，这里不再赘述。

(4) 2D9 节点单元单元

对于四边形单元而言，其高阶单元构造方式有多种，常见的有 2D8 节点单元和 2D9 节点单元。2D8 节点单元构造方法为：在前述 2D4 节点单元各边增加一个中点；对于 2D9 节点单元而言，其构造方法为：在 2D8 节点单元基础上，在单元中心再增加一个点，该单元类型对应标准“帕斯卡”三角形，其编号同样有纵向和横向编号之分，如图 3-6 所示为采用九节点四边形单元组成的网格结构及其编号方式示意。

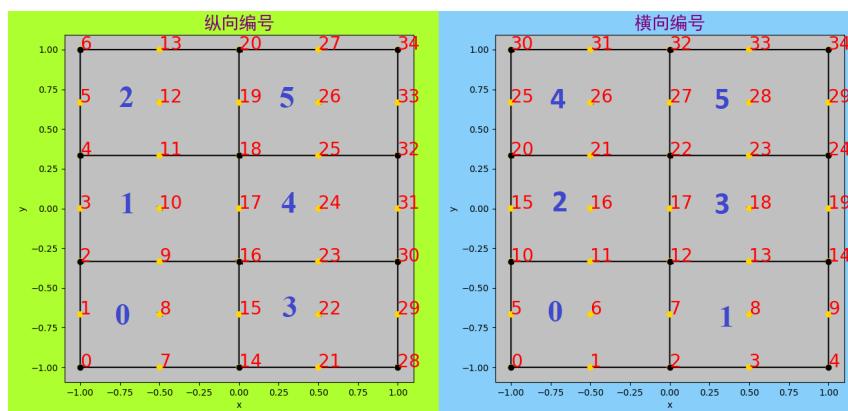


图 3-6 2D9 节点单元网格划分及编号示例

完成网格划分后按照前述有限元方法步骤生成对应网格、边界信息矩阵，计算单元刚度阵、载荷向量，组装线性系统、边界处理并进行计算后处理，详细执行步骤请参考前述 2D3 节点、2D6 节点单元有限元仿真过程。

3.7.2 Example 2

例 3.2 【问题描述】采用有限元方法求解包含 Dirichlet 和 Neumann 边界的 2D 泊松方程，计算域为： $\Omega = [-1, -1] \times [1, 1]$ 。

$$\begin{cases} -\nabla \cdot (\nabla u) = -2e^{x+y}, (x, y) \in \Omega \\ u = e^{-1+y}, x = -1 \\ u = e^{1+y}, x = 1 \\ \nabla u \cdot \vec{n} = -e^{x-1}, y = -1 \\ u = e^{x+1}, y = 1 \end{cases}$$

【说明】上述问题的解析解为： $u = e^{x+y}$ ，可用于与有限元得到结果进行比对。

(1) 2D 三节点单元求解

采用 3.7.1 节中图 3-1a 所示网格划分，其基本信息矩阵如前所述。本算例中控制方程左侧与 3.7.1 节中相同，源项 $f(x, y) = -2e^{x+y}$ 与边界条件不同，故在边界条件处理前所得刚度矩阵与式 (3.79) 相同，将源项代入前述对应载荷向量元素求解公式中即可得到本算例在使用 2D3 节点单元时对应的载荷向量，这里直接给出载荷向量对应结果作为参考。

$$\mathbf{F} = \begin{bmatrix} -0.5413 & -3.2537 & -3.2548 & -3.9987 \end{bmatrix}^T \quad (3.97)$$

本算例模型中存在 Neumann 边界类型，根据 3.3.2 节知道该边界条件不改变系统刚度矩阵，仅改变载荷向量。需要在对应载荷向量处增加 Neumann 边界带来的影响，详细推导过程可参考 3.3.2 节。注意到，Neumann 边界条件的影响为边界的线积分，其物理意义为物理量 u 的通量，与 1D 情况和 2D Dirichlet 边界不同，在进行该通量计算时，需要明确该边界起、始位置，由前述网格划分矩阵信息不再能够满足需要，为此引入边界信息矩阵 $edgeBC$ ，其组成及各元素含义如下：

$$edgeBC = \begin{bmatrix} type \\ ownerCellID \\ point1 \\ point2 \end{bmatrix} \quad (3.98)$$

其中， $type$ 表示边界的 $edge$ 的类型：-1 表示 Dirichlet 边界；-2 表示 Neumann 边界；-3 表示 Robin 边界； $ownerCellID$ 表示 $edge$ 所属单元； $point1$, $point2$ 表示组成该 $edge$ 的端点 1 和端点 2 的全局编号，这样通过单元编号及节点全局编号就可以知道节点的坐标值，进一步得到线积分的大小，也可以确定所得线积分应该作用在载荷向量的具体位置。就本算例而言该边界信息矩阵为：

$$edgeBC = \begin{bmatrix} -2 & -1 & -1 & -1 \\ 1 & 2 & 2 & 1 \\ 1 & 3 & 4 & 2 \\ 3 & 4 & 2 & 1 \end{bmatrix} \quad (3.99)$$

本问题模型中涉及到两种不同的边界条件：Dirichlet 边界和 Neumann 边界，所以在进行处理时需要不同的处理策略。根据边界类型的重要性及对方程求解的作用：Dirichlet 边界为强约束边界 (Essential)；Neumann 边界又称为 Natural Boundary Condition，为自然边界。因此在边界处理时需要保证 Dirichlet 边界始终发挥作用，Neumann 边界条件优先级在其之后。因此本算例中边界处理方式为：1)、首先对 Neumann 边界条件进行处

理，遍历 edgeBC 信息矩阵，如果给定的边界 type 为 -2，则根据其 point1、point2 得到组成该 edge 的线积分上下限，然后进行线积分，积分方法采用 1D FEM 中相同的积分处理方式。2) 对 Dirichlet 边界进行处理，处理方法与 Example 1 中边界处理方法相同。经过边界处理后得到对应刚度矩阵、载荷向量，得到待求线性系统，采用合适矩阵求解算法后即可得到问题的有限元解。

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} 0.1353 \\ 1 \\ 1 \\ 7.3891 \end{bmatrix} \rightarrow \mathbf{u} = \begin{bmatrix} 0.1353 \\ 1 \\ 1 \\ 7.3891 \end{bmatrix} \quad (3.100)$$

由于在 x 、 y 方向网格划分数量有限，导致 Neumann 自然边界被 Dirichlet 强约束边界替代，所以得到线性系统各节点均为 Dirichlet 边界节点，所得载荷向量亦为对应节点函数值。图 3-7a 和图 3-7b 分别为 2D3 节点单元所得有限元解与对应解析解。

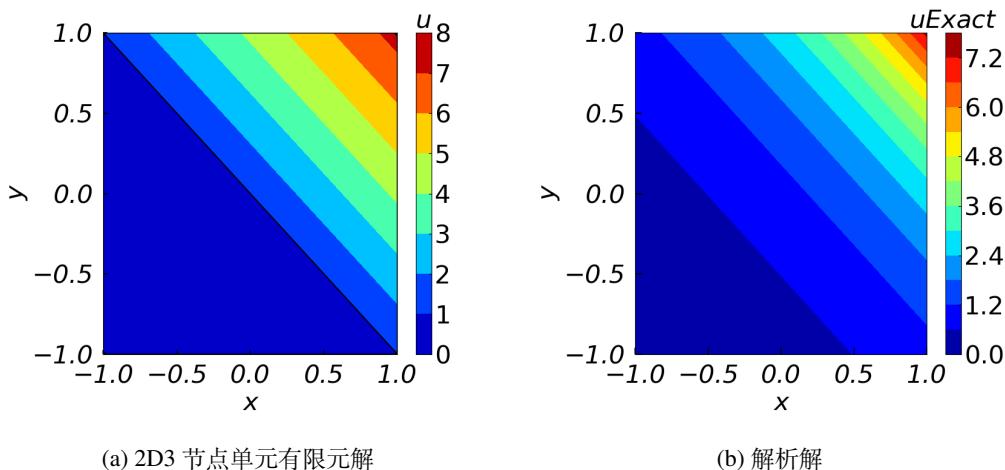


图 3-7 Example 2 三节点单元有限元解与解析解

表 3-3 给出了不同空间步长下，2D3 节点单元各类误差范数结果参考。

表 3-3 Example 2 三节点单元不同空间步长下各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	4.2350×10^{-4}	1.3358×10^{-2}	5.1224×10^{-3}	1.8523×10^{-1}
1/16	1.0633×10^{-4}	3.4487×10^{-3}	1.2793×10^{-3}	9.2559×10^{-2}
1/32	2.6609×10^{-5}	8.7622×10^{-4}	3.1973×10^{-4}	4.6273×10^{-2}
1/64	6.6555×10^{-6}	2.2084×10^{-4}	7.9928×10^{-5}	2.3136×10^{-2}

(2) 2D6 节点单元求解

采用 3.7.1 节图 3-3a 中所示网格划分, 网格基本信息与前述章节一致, Neumann 边界为几何模型的一部分, 故其边界矩阵信息 $edgeBC$ 与 2D3 节点单元一致。本问题中 2D6 节点单元刚度矩阵与式 (3.93) 一致, 根据本问题源项计算对应载荷向量。

$$\mathbf{F} = \begin{bmatrix} 0.1051 & -0.6472 & 0.2063 & -0.6456 & -3.0503 & -3.2224 & 0.2084 & -3.2306 & -0.7722 \end{bmatrix}^T \quad (3.101)$$

与本问题中采用 2D3 节点单元边界处理方法完全一致, 边界处理后得到线性系统:

$$\left[\begin{array}{ccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.6667 & 0 & 0 & 2.6667 & -1.3333 & 0 & -0.6667 & 0 & 0 \\ 0 & -1.3333 & 0 & -1.3333 & 5.3333 & -1.3333 & 0 & -1.3333 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right] \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \end{bmatrix} = \begin{bmatrix} 0.1353 \\ 0.3679 \\ 1 \\ -1.1869 \\ -3.0503 \\ 2.7182 \\ 1 \\ 2.7183 \\ 7.3891 \end{bmatrix} \quad (3.102)$$

可以看到对本问题中网格划分而言, 4、5 号节点为 Neumann 边界, 且这两处节点不会被 Dirichlet 边界覆盖, 因此对应两节点处矩阵信息未变, 载荷向量对应增加了边界处通量值, 其余节点均为给定边界值。采用合适矩阵求解算法后可以得到各节点函数值:

$$\mathbf{u} = \begin{bmatrix} 0.1353 & 0.3679 & 1 & 0.3181 & 0.9587 & 2.7183 & 1 & 2.7183 & 7.3891 \end{bmatrix}^T \quad (3.103)$$

图 3-8a 和图 3-8b 分别为 2D6 节点单元所得有限元解与对应解析解, 可以看到该数值结果较采用 2D3 节点单元所得结果更加接近解析解, 数值分布更加平滑。

表 3-4 给出了不同空间步长下, 采用 2D6 节点单元时各类误差范数的结果参考:

表 3-4 Example 2 二维六节点单元不同空间步长下各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	1.4630×10^{-5}	1.0956×10^{-4}	3.9285×10^{-5}	2.9874×10^{-3}
1/16	1.8384×10^{-6}	1.4074×10^{-5}	4.9015×10^{-6}	7.4668×10^{-4}
1/32	2.3048×10^{-7}	1.7835×10^{-6}	6.1244×10^{-7}	1.8667×10^{-4}
1/64	2.8855×10^{-8}	2.2447×10^{-7}	7.6549×10^{-8}	4.6667×10^{-5}

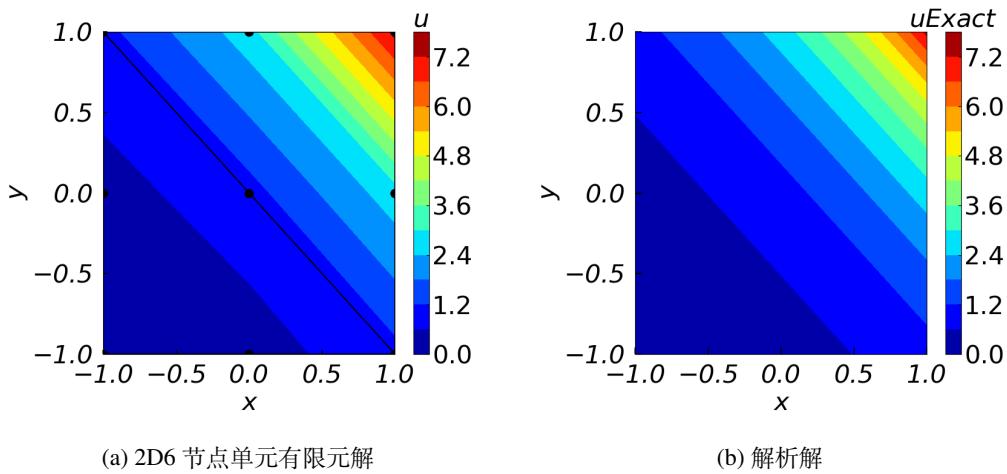


图 3-8 Example 2 二维六节点单元有限元解与解析解

3.7.3 Example 3

例 3.3 【问题描述】采用有限元方法求解包含 Dirichlet 和 Robin 边界的 2D 泊松方程。将模型区域为： $\Omega = [-1, -1] \times [1, 1]$ 。

$$\begin{cases} -\nabla \cdot (\nabla u) = -2e^{x+y}, & (x, y) \in \Omega \\ u = e^{-1+y}, & x = -1 \\ u = e^{1+y}, & x = 1 \\ \nabla u \cdot \vec{n} + u = 0, & y = -1 \\ u = e^{x+1}, & y = 1 \end{cases}$$

【说明】上述问题的解析解为： $u = e^{x+y}$ ，可用于与有限元得到结果进行比对。

(1) 2D 三节点单元求解

采用 3.7.1 节图 3-1a 所示网格划分。边界条件信息矩阵与 Example 2 中一致，仅将 Neumann 对应边界变为 Robin 边界。本问题模型控制方程与 Example 2 相同，所以，在边界处理前本问题中刚度矩阵与式 (3.79)，载荷向量与式 (3.97) 相同。

由于存在 Robin 边界条件，刚度矩阵和载荷向量需要同时处理，处理方法可参考 3.3.3 节。根据本问题中边界条件的重要性，首先对 Robin 边界进行处理，之后处理 Dirichlet 边界，始终保证重要边界发挥作用，经过边界处理后得到对应线性系统，并采

取合适方法求解得到各节点函数值。

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} 0.1353 \\ 1 \\ 1 \\ 7.3891 \end{bmatrix} \rightarrow \mathbf{u} = \begin{bmatrix} 0.1353 \\ 1 \\ 1 \\ 7.3891 \end{bmatrix} \quad (3.104)$$

由于网格划分较少, Robin 边界在处理时被 Dirichlet 边界覆盖。因此所得有限元解与 Example 2 中 2D3 节点单元有限元解相同, 结果分布如图 3-7a 所示, 表 3-5 给出了不同空间步长下三节点单元各类误差范数结果参考。

表 3-5 Example 3 三节点单元不同空间步长下各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	5.4686×10^{-4}	1.3358×10^{-2}	5.1094×10^{-3}	1.8523×10^{-1}
1/16	1.3691×10^{-4}	3.4487×10^{-3}	1.2760×10^{-3}	9.2559×10^{-2}
1/32	3.4236×10^{-5}	8.7622×10^{-4}	3.1893×10^{-4}	4.6273×10^{-2}
1/64	8.5614×10^{-6}	2.2084×10^{-4}	7.9727×10^{-5}	2.3136×10^{-2}

(2) 2D 六节点单元求解

采用 3.7.1 节图 3-3a 中所示网格划分, 信息矩阵如前述对应章节所示, Neumann 对应边界变为 Robin 边界。边界处理前系统刚度矩阵与式 (3.93) 一致, 载荷向量与式 (3.101) 一致。与本问题中采用 2D3 节点单元边界处理方法完全一致, 边界处理后得到线性系统:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.5333 & 0 & 0 & 3.7333 & -1.3333 & 0 & -0.5333 & 0 & 0 \\ 0 & -1.3333 & 0 & -1.3333 & 5.3333 & -1.3333 & 0 & -1.3333 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \end{bmatrix} = \begin{bmatrix} 0.1353 \\ 0.3679 \\ 1 \\ -0.6456 \\ -3.0503 \\ 2.7182 \\ 1 \\ 2.7183 \\ 7.3891 \end{bmatrix} \quad (3.105)$$

对本问题中网格划分而言, 4、5 号节点为 Robin 边界, 且这两处节点不会被 Dirichlet 边界覆盖, 因此对应两节点处矩阵信息根据 Robin 边界进行修改, Robin 边界中参数

$q = 0$, 因此载荷向量对应处未发生变化, 其余节点均为给定边界值。

$$\mathbf{u} = \begin{bmatrix} 0.1353 & 0.3679 & 1 & 0.3330 & 0.9624 & 2.7183 & 1 & 2.7183 & 7.3891 \end{bmatrix}^T \quad (3.106)$$

图 3–9a 和图 3–9b 分别为 2D6 节点单元所得有限元解与对应解析解, 可以看到该数值结果较采用 2D3 节点单元所得结果更加接近解析解, 数值分布更加平滑。

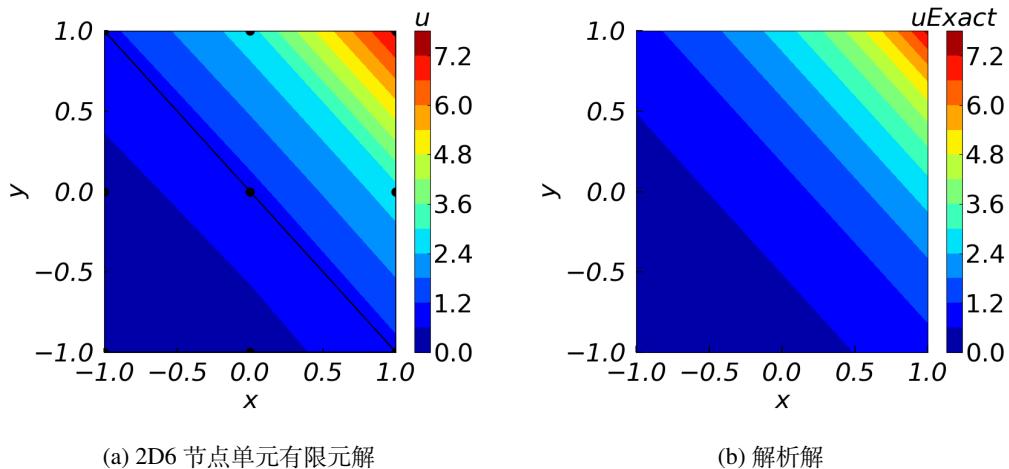


图 3–9 Example 3 二维六节点单元有限元解与解析解

表 3–6 给出了不同空间步长下六节点单元各类误差范数结果参考。

表 3–6 Example 3 六节点单元不同空间步长下各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	1.4076×10^{-5}	1.0956×10^{-4}	3.9278×10^{-5}	2.9874×10^{-3}
1/16	1.8025×10^{-6}	1.4074×10^{-5}	4.9012×10^{-6}	7.4668×10^{-4}
1/32	2.2819×10^{-7}	1.7835×10^{-6}	6.1243×10^{-7}	1.8667×10^{-4}
1/64	2.8710×10^{-8}	2.2447×10^{-7}	7.6549×10^{-8}	4.6667×10^{-5}

习题

- 采用有限元方法计算如下 PDE 问题, 分别采用线性三角形单元与四边形单元, 对比两种单元的计算结果, 并分析其不同单元下有限元解与解析解的各类误差 (最大

节点误差、 L^∞ 误差、 L^2 误差、 H^1 误差)。问题描述如下：

$$\begin{cases} -\nabla^2 u = -6, \text{ in } \Omega = [0, 1]^2 \\ u = 1 + 2y^2, x = 0 \\ u = 2 + 2y^2, x = 1 \\ -\frac{\partial u}{\partial n} = 0, y = 0 \\ -\frac{\partial u}{\partial n} = -4, y = 1 \end{cases}$$

该问题的解析解为 $u = 1 + x^2 + 2y^2$ 。

2. 采用有限元方法计算如下 PDE 问题，分别采用线性三角形单元与四边形单元，对比两种单元的计算结果，并分析其不同单元下有限元解与解析解的各类误差（最大节点误差、 L^∞ 误差、 L^2 误差、 H^1 误差）。问题描述如下：

$$\begin{cases} -\nabla^2 u = -6, \text{ in } \Omega = [0, 1]^2 \\ u = 1 + 2y^2, x = 0 \\ u = 2 + 2y^2, x = 1 \\ -\frac{\partial u}{\partial n} = -4, y = 1 \\ -\frac{\partial u}{\partial n} = 1000(u - 1 - x^2), y = 0 \end{cases}$$

该问题的解析解为 $u = 1 + x^2 + 2y^2$ 。

3. 采用有限元方法计算如下 PDE 问题，分别采用线性三角形单元与四边形单元，对比两种单元的计算结果，并分析其不同单元下有限元解与解析解的各类误差（最大节点误差、 L^∞ 误差、 L^2 误差、 H^1 误差）。问题描述如下：

$$\begin{cases} -\nabla^2 u = 2\pi^2 \cos(\pi x) \cos(\pi y), \text{ in } \Omega = [-1, 1]^2 \\ u(-1, y) = -\cos(\pi y) - 1 \\ u(1, y) = -\cos(\pi y) - 1 \\ u(x, -1) = -\cos(\pi x) - 1 \\ u(x, 1) = -\cos(\pi x) - 1 \end{cases}$$

该问题的解析解为 $u = \cos(\pi x) \cos(\pi y) - 1$ 。

4

2D 线弹性方程有限元方法

4.1 引言

本章主要对 2D 线弹性方程^[14-22] 的有限元方法进行介绍，首先对 2D 线弹性问题及其一般方程进行简要介绍，随后对 2D 线弹性方程的有限元方法进行介绍，包括：2D 线弹性方程的弱格式推导、线性系统的构建、不同边界条件下线性系统的处理等。

4.2 线弹性理论一般原理与基本方程

线弹性理论是弹性力学中的一个重要分支，其主要研究材料在小应变条件下的力学性质，描述的物理场为矢量场，如位移、形变、应力等，该理论主要基于如下假设：

1. 材料发生变形很小；
2. 材料变形满足线性关系（应变与位移、应力与应变）；
3. 材料动力学响应可以忽略；
4. 材料变形过程中未产生裂痕或重叠。

线弹性问题较为复杂，其涉及到材料的特性、材料变形、载荷等，在使用 FEM 对该问题进行求解之前，首先对方程中涉及到的相关概念、线弹性问题的一般形式进行介绍。

4.2.1 线弹性问题基本概念

在对线弹性问题中涉及到的基本方程进行介绍前首先对线弹性问题中涉及到的一些概念名词进行简要介绍，主要包括：应力状态、应力矩阵、柯西应力等，这些概念对

于理解线弹性问题具有一定帮助。

► 应力状态

通过物体内一点可以作无数个方向不同的切平面，各切平面上的应力矢量一般各不相同，通常将物体内一点各切平面上的应力情况称为该点的应力状态。

► 应力矩阵

应力矩阵是描述应力状态的一种方式，在笛卡尔坐标系下，分别沿平行坐标平面的 3 个坐标方向进行应力分解后，可以得到 9 个应力分量，它们整体构成了一个应力矩阵，其具有如下形式：

$$\sigma_{ij} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} \quad (4.1)$$

► 柯西应力公式

柯西证明了只需要知道 3 个相互垂直的面上的应力矢量，即 9 个分量构成的应力矩阵，就可确定任意一个方向的应力矢量，进而得到该方向对应平面上的正应力和剪应力。柯西应力公式如下：

$$\mathbf{T} = \mathbf{n} \cdot \bar{\boldsymbol{\sigma}} = \begin{bmatrix} n_1 & n_2 & n_3 \end{bmatrix} \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} \quad (4.2)$$

► 主应力和应力主轴（主方向）

当坐标转动时，受力物体内任一确定点的 9 个应力分量也会随之变化。在坐标系不断转动过程中，必然可以找到一个坐标系，使得该点所在坐标系中只有正应力分量，剪应力分量为 0。这样的微分面称为主微分面，简称主平面，其法方向称为应力主方向，其上的应力称为主应力。此时该坐标系称为应力主轴（主方向），应力主轴是对于物体中点而言的，不同点的应力主轴一般不同。

► 主应力、主方向和应力张量不变量

设主方向对 x、y、z 轴的方向余弦为 l, m, n ，对应主应力是 σ 。主应力在 3 个坐标轴上的投影为 $\sigma l, \sigma m, \sigma n$ ，由柯西应力公式可得：

$$\begin{cases} \sigma_x l + \tau_{xy} m + \tau_{xz} n = \sigma l \\ \tau_{yx} l + \sigma_y m + \tau_{yz} n = \sigma m \\ \tau_{zx} l + \tau_{zy} m + \sigma_z n = \sigma n \end{cases} \rightarrow \begin{cases} (\sigma_x - \sigma) l + \tau_{xy} m + \tau_{xz} n = 0 \\ \tau_{yx} l + (\sigma_y - \sigma) m + \tau_{yz} n = 0 \\ \tau_{zx} l + \tau_{zy} m + (\sigma_z - \sigma) n = 0 \end{cases} \quad (4.3)$$

$$\begin{vmatrix} \sigma_x - \sigma & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y - \sigma & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z - \sigma \end{vmatrix} = 0 \longrightarrow \sigma^3 - I_1\sigma^2 + I_2\sigma - I_3 = 0 \quad (4.4)$$

► 应变分析

外力（或温度变化）作用下，物体内部各部分之间要产生相对运动，这种运动形态称为变形。如果各点（或部分点）间的相对距离发生变化，则物体发生了变形。这种变形一方面表现在微线段长度的变化，称为线应变；另一方面表现在微线段间夹角的变化，称为切应变。

4.2.2 线弹性问题一般形式

线弹性方程通常由平衡（运动）微分方程、几何方程、物理方程（本构方程）、边界条件方程组成，下面将对方程各组成部分进行介绍。

(1) 平衡（运动）微分方程

这里直接给出变形物体在空间的平衡（运动）微分方程，其具体推导过程可以参考相关书籍：

$$\left\{ \begin{array}{l} \frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + \mathbf{f}_x = 0 \\ \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \frac{\partial \tau_{zy}}{\partial z} + \mathbf{f}_y = 0 \\ \frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + \mathbf{f}_z = 0 \end{array} \right. \quad (4.5)$$

将上式整理变换得到：

$$\underbrace{\left[\frac{\partial}{\partial x} \quad \frac{\partial}{\partial y} \quad \frac{\partial}{\partial z} \right]}_{\nabla} \underbrace{\begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix}}_{\sigma} + \mathbf{f} = 0 \quad (4.6)$$

所以可以得到变形体的平衡（运动）微分方程的一般形式为：

$$-\nabla \cdot \sigma = \mathbf{f} \quad (4.7)$$

(2) 几何方程

几何方程主要描述了运动位移与应变之间的关系，当位移完全确定时，应变即完全确定，反之不成立。其一般形式为：

$$\begin{cases} \varepsilon_x = \frac{\partial u}{\partial x}, & \varepsilon_{xy} = \varepsilon_{yx} = \frac{1}{2}\gamma_{xy} = \frac{1}{2}\left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right) \\ \varepsilon_y = \frac{\partial v}{\partial y}, & \varepsilon_{yz} = \varepsilon_{zy} = \frac{1}{2}\gamma_{yz} = \frac{1}{2}\left(\frac{\partial w}{\partial y} + \frac{\partial v}{\partial z}\right) \\ \varepsilon_z = \frac{\partial w}{\partial z}, & \varepsilon_{zx} = \varepsilon_{xz} = \frac{1}{2}\gamma_{zx} = \frac{1}{2}\left(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}\right) \end{cases} \quad (4.8)$$

将上述方程整理变换得到：

$$\varepsilon_{ij} = \begin{bmatrix} \varepsilon_{11} & \varepsilon_{12} & \varepsilon_{13} \\ \varepsilon_{21} & \varepsilon_{22} & \varepsilon_{23} \\ \varepsilon_{31} & \varepsilon_{32} & \varepsilon_{33} \end{bmatrix} = \begin{bmatrix} \varepsilon_x & \varepsilon_{xy} & \varepsilon_{xz} \\ \varepsilon_{yx} & \varepsilon_y & \varepsilon_{yz} \\ \varepsilon_{zx} & \varepsilon_{zy} & \varepsilon_z \end{bmatrix}, \quad \varepsilon_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right), \quad \varepsilon = (\nabla u + \nabla^T u)/2 \quad (4.9)$$

【说明】对于某一初始连续的物体，按照某一应变状态变形后必须保持其整体性与连续性，即线弹性问题基本假设：物体既不开裂也不重叠，此时给定的应变状态是协调的，否则就是不协调的，该问题为非线弹性问题。从物理意义上说，如果位移函数是连续的变形，自然就可以协调，因此采用位移方法求解问题时，应变状态是自然满足的，而采用应力方法求解问题时，需要同时满足变形协调方程（连续性方程），变形协调方程的一般形式如下：

$$\begin{cases} \frac{\partial^2 \varepsilon_x}{\partial y^2} + \frac{\partial^2 \varepsilon_y}{\partial x^2} = \frac{\partial^2 \gamma_{xy}}{\partial x \partial y} \\ \frac{\partial^2 \varepsilon_y}{\partial z^2} + \frac{\partial^2 \varepsilon_z}{\partial y^2} = \frac{\partial^2 \gamma_{yz}}{\partial y \partial z} \\ \frac{\partial^2 \varepsilon_z}{\partial x^2} + \frac{\partial^2 \varepsilon_x}{\partial z^2} = \frac{\partial^2 \gamma_{zx}}{\partial z \partial x} \\ 2 \frac{\partial^2 \varepsilon_x}{\partial y \partial z} = \frac{\partial}{\partial x} \left(\frac{\partial \gamma_{zx}}{\partial y} + \frac{\partial \gamma_{xy}}{\partial z} + \frac{\partial \gamma_{yz}}{\partial x} \right) \\ 2 \frac{\partial^2 \varepsilon_y}{\partial z \partial x} = \frac{\partial}{\partial y} \left(\frac{\partial \gamma_{xy}}{\partial z} + \frac{\partial \gamma_{yz}}{\partial x} + \frac{\partial \gamma_{zx}}{\partial y} \right) \\ 2 \frac{\partial^2 \varepsilon_z}{\partial x \partial y} = \frac{\partial}{\partial z} \left(\frac{\partial \gamma_{yz}}{\partial x} + \frac{\partial \gamma_{zx}}{\partial y} + \frac{\partial \gamma_{xy}}{\partial z} \right) \end{cases} \quad (4.10)$$

(3) 物理方程（本构方程）

物理方程（本构方程）主要描述了应力与应变之间的关系，通过该方程应力与应变可以相互转换。线弹性问题中通常采用胡克定律描述应力与应变之间的关系：假设物体

是连续、均匀的，并且是弹性的、没有初应力，且形变是微小的，可以取线性关系：

$$\left\{ \begin{array}{l} \sigma_x = c_{11}\varepsilon_x + c_{12}\varepsilon_y + c_{13}\varepsilon_z + c_{14}\gamma_{xy} + c_{15}\gamma_{yz} + c_{16}\gamma_{zx} \\ \sigma_y = c_{21}\varepsilon_x + c_{22}\varepsilon_y + c_{23}\varepsilon_z + c_{24}\gamma_{xy} + c_{25}\gamma_{yz} + c_{26}\gamma_{zx} \\ \vdots \\ \tau_{zx} = c_{61}\varepsilon_x + c_{62}\varepsilon_y + c_{63}\varepsilon_z + c_{64}\gamma_{xy} + c_{65}\gamma_{yz} + c_{66}\gamma_{zx} \end{array} \right. \quad (4.11)$$

上式中各项独立存在， $\sigma_{ij} = c_{ijkl}\varepsilon_{kl}$, $i, j, k, l = 1, 2, 3$, c_{mn} 称为弹性系数（常数），构成 6×6 的矩阵，共 36 个，可以由能量守恒定律证明 $c_{mn} = c_{nm}$ ，所以对于一般的各向异性介质，弹性系数矩阵具有 21 个独立分量。

对于各向同性弹性材料，只有两个弹性常数，由此上式可以表示为：

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{zx} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{12} & 0 & 0 & 0 \\ c_{12} & c_{11} & c_{12} & 0 & 0 & 0 \\ c_{12} & c_{12} & c_{11} & 0 & 0 & 0 \\ 0 & 0 & 0 & c_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & c_{44} & 0 \\ 0 & 0 & 0 & 0 & 0 & c_{44} \end{bmatrix} \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{bmatrix} \quad (4.12)$$

其中， $2c_{44} = c_{11} - c_{12}$ ，设 $\lambda = c_{12}$, $\mu = c_{44}$ ，这两个独立的常数称为拉梅常数（Lamé parameter），由此 $c_{11} = \lambda + 2\mu$ ，可以进一步得到：

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{zx} \end{bmatrix} = \begin{bmatrix} \lambda + 2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda + 2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda + 2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix} \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{bmatrix} \quad (4.13)$$

所以有：

$$\sigma_{ij} = \lambda\theta\delta_{ij} + 2\mu\varepsilon_{ij} \text{ 或 } \sigma_{ij} = \lambda(\nabla \cdot \mathbf{u})\delta_{ij} + 2\mu\varepsilon_{ij} \quad (4.14)$$

其中， $\theta = \nabla \cdot \mathbf{u} = \left[\frac{\partial}{\partial x} \quad \frac{\partial}{\partial y} \quad \frac{\partial}{\partial z} \right] \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \varepsilon_x + \varepsilon_y + \varepsilon_z$ 为体应变， $\delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$ 。

拉梅常数与弹性模量 E 、泊松比 ν 之间的关系为：

$$\left\{ \begin{array}{l} E = \frac{\mu(3\lambda + 2\mu)}{\lambda + \mu} \\ \nu = \frac{\lambda}{2(\lambda + \mu)} \end{array} \right. \longleftrightarrow \left\{ \begin{array}{l} \lambda = \frac{E\nu}{(1 + \nu)(1 - 2\nu)} \\ \mu = \frac{E}{2(1 + \nu)} \end{array} \right. \quad (4.15)$$

用弹性模量 E 和泊松比 ν 来描述胡克定律的应力-应变关系: $\sigma = \mathbf{D}\varepsilon$, 其中 \mathbf{D} 为弹性矩阵, 对于 2D 平面应变与平面应力问题^[23] 该矩阵形式有所区别:

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{zx} \end{bmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{bmatrix} \quad (4.16)$$

该矩阵也可以简写为如下表达式:

$$\begin{cases} \sigma_{ij} = \frac{E}{1+\nu} \varepsilon_{ij} + \frac{E\nu\theta}{(1+\nu)(1-2\nu)} \delta_{ij} \\ \varepsilon_{ij} = \frac{1}{E} [(1+\nu)\sigma_{ij} - \nu\sigma_{kk}\delta_{ij}] \end{cases} \quad (4.17)$$

其中, $\sigma_{kk} = \sigma_x + \sigma_y + \sigma_z$ 。

此外, 工程上常用体积模量 K 和剪切模量 G 来描述材料的性质, 其与拉梅常数之间的关系为:

$$\begin{cases} K = \frac{E}{3(1-2\nu)} = \lambda + \frac{2}{3}\mu \\ G = \frac{E}{2(1+\nu)} = \mu \end{cases} \longleftrightarrow \begin{cases} \lambda = K - \frac{2}{3}G \\ \mu = G \end{cases} \quad (4.18)$$

用体积模量 K 和剪切模量 G 来描述胡克定律的应力-应变关系, 其一般形式为:

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{zx} \end{bmatrix} = \begin{bmatrix} K + \frac{4}{3}G & K - \frac{2}{3}G & K - \frac{2}{3}G & 0 & 0 & 0 \\ K - \frac{2}{3}G & K + \frac{4}{3}G & K - \frac{2}{3}G & 0 & 0 & 0 \\ \nu & \nu & K + \frac{4}{3}G & 0 & 0 & 0 \\ 0 & 0 & 0 & G & 0 & 0 \\ 0 & 0 & 0 & 0 & G & 0 \\ 0 & 0 & 0 & 0 & 0 & G \end{bmatrix} \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{bmatrix} \quad (4.19)$$

(4) 边界条件

按照边界条件不同, 弹性力学问题可以分为位移边界问题、应力边界问题和混合边界问题。

1. 位移边界问题：在所有边界上给定位移：

$$\begin{cases} u = \bar{u} \\ v = \bar{v} \\ w = \bar{w} \end{cases} \quad (4.20)$$

2. 应力边界问题：在所有边界上给定外力（面力），应力应满足应力边界条件：

$$\begin{cases} \bar{f}_x = \sigma_x l + \tau_{yx} m + \tau_{zx} n \\ \bar{f}_y = \tau_{xy} l + \sigma_y m + \tau_{zy} n \\ \bar{f}_z = \tau_{xz} l + \tau_{yz} m + \sigma_z n \end{cases} \quad (4.21)$$

3. 混合边界问题：既有应力边界，又有位移边界。

(5) 弹性力学问题的一般形式

根据上述分析，弹性力学问题为平衡微分方程、几何方程、物理方程、边界条件的组合：

$$\left. \begin{array}{l} \text{微分体的平衡条件} \rightarrow \text{平衡微分方程} \\ \text{微分线段上形变与位移关系} \rightarrow \text{几何方程} \\ \text{应力与形变的物理关系} \rightarrow \text{物理方程} \\ \text{给定的面力边界微分体平衡} \rightarrow \text{应力边界条件} \\ \text{给定的约束边界上约束条件} \rightarrow \text{位移边界条件} \end{array} \right\} \begin{array}{l} \text{基本方程} \\ \text{边界条件} \end{array} \right\} \text{线弹性问题}$$

可以看到该问题涉及的未知量有位移、应力、应变，方程组较为复杂，在求解该问题时，通常采用位移法和应方法，其中位移法更为常用，因此这里对位移法求解的基本思路进行描述。当采用位移法求解线弹性问题时，主要步骤为：

1. 将几何方程代入物理方程，得到应力与位移的关系；
2. 将上述方程代入平衡方程，得到位移形式的平衡微分方程，该方程也称为拉梅方程 (Lamé-Navier Equation)；
3. 求解得到位移场；
4. 计算得到应变场和应力场。

4.3 问题模型

根据前述线弹性方程的一般组成形式，构造 2D 情况下待求的线弹性方程：

$$\begin{cases} -\nabla \cdot \sigma(\mathbf{u}) = \mathbf{f}, & (x, y) \in \Omega \\ \mathbf{u} = \mathbf{g}, & (x, y) \in \partial\Omega \end{cases} \quad (4.22)$$

其中, $\mathbf{u}(x, y) = (u_x, u_y)^T$, $\mathbf{f}(x, y) = (f_x, f_y)^T$, $\mathbf{g}(x, y) = (g_x, g_y)^T$, σ 为应力张量, 定义如下:

$$\sigma(\mathbf{u}) = \begin{bmatrix} \sigma_{11}(\mathbf{u}) & \sigma_{12}(\mathbf{u}) \\ \sigma_{21}(\mathbf{u}) & \sigma_{22}(\mathbf{u}) \end{bmatrix}, \quad \sigma_{ij}(\mathbf{u}) = \lambda(\nabla \cdot \mathbf{u})\delta_{ij} + 2\mu\varepsilon_{ij}(\mathbf{u}) \quad (4.23)$$

其中, λ 和 μ 为拉梅常数, $\delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$, ε_{ij} 为应变张量, 定义如下:

$$\varepsilon = \begin{bmatrix} \varepsilon_{11}(\mathbf{u}) & \varepsilon_{12}(\mathbf{u}) \\ \varepsilon_{21}(\mathbf{u}) & \varepsilon_{22}(\mathbf{u}) \end{bmatrix}, \quad \varepsilon_{ij}(\mathbf{u}) = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad (4.24)$$

由此可以得到应力张量的具体形式为:

$$\sigma = \begin{bmatrix} \lambda \frac{\partial u_x}{\partial x} + 2\mu \frac{\partial u_x}{\partial x} + \lambda \frac{\partial u_y}{\partial y} & \mu \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \\ \mu \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) & \lambda \frac{\partial u_x}{\partial x} + \lambda \frac{\partial u_y}{\partial y} + 2\mu \frac{\partial u_y}{\partial y} \end{bmatrix} \quad (4.25)$$

4.3.1 弱格式

(1) 在方程两端同乘以任意的测试函数 (Test Function) $\mathbf{v}(x, y) = (v_x, v_y)^T$ 做内积 (Inner Product), 并在整个求解域 Ω 上积分

$$\begin{aligned} -\nabla \cdot \sigma(\mathbf{u}) &= \mathbf{f}, \quad (x, y) \in \Omega \\ \rightarrow -(\nabla \cdot \sigma(\mathbf{u})) \cdot \mathbf{v} &= \mathbf{f} \cdot \mathbf{v}, \quad (x, y) \in \Omega \\ \rightarrow - \int_{\Omega} (\nabla \cdot \sigma(\mathbf{u})) \cdot \mathbf{v} dx dy &= \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \end{aligned} \quad (4.26)$$

(2) 根据多元分部积分法 (散度定理) 对方程等号左边项进行变换

$$\int_{\Omega} (\nabla \cdot \sigma(\mathbf{u})) \cdot \mathbf{v} dx dy = \int_{\partial\Omega} (\sigma(\mathbf{u}) \mathbf{n}) \cdot \mathbf{v} ds - \int_{\Omega} \sigma(\mathbf{u}) : \nabla \mathbf{v} dx dy \quad (4.27)$$

其中, $\mathbf{n} = (n_1, n_2)^T$ 为计算域边界 $\partial\Omega$ 的单位外法向量 (Unit Outer Normal Vector), 由此可以得到:

$$\int_{\Omega} \sigma(\mathbf{u}) : \nabla \mathbf{v} dx dy - \int_{\partial\Omega} (\sigma(\mathbf{u}) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \quad (4.28)$$

对于两个张量, 其内积的定义为:

$$A : B = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} : \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = a_{12}b_{11} + a_{12}b_{12} + a_{21}b_{21} + a_{22}b_{22} \quad (4.29)$$

且有：

$$\nabla \mathbf{v} = \begin{bmatrix} \frac{\partial v_x}{\partial x} & \frac{\partial v_x}{\partial y} \\ \frac{\partial v_y}{\partial x} & \frac{\partial v_y}{\partial y} \end{bmatrix} \quad (4.30)$$

待求函数在计算域边界 $\partial\Omega$ 处的值由 $\mathbf{u} = \mathbf{g}$ 给定，因此可以选择合适的测试函数 $\mathbf{v}(x, y)$ 使其在边界 $\partial\Omega$ 处满足 $\mathbf{v}(x, y) = 0$ ，此时方程 (4.28) 等号左边第二项为零，可以得到：

$$\int_{\Omega} \sigma(\mathbf{u}) : \nabla \mathbf{v} dx dy = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \quad (4.31)$$

方程 (4.31) 即为待求问题的弱格式 (Weak Formulation)。

$$\begin{aligned} \sigma(\mathbf{u}) : \nabla \mathbf{v} &= \begin{bmatrix} \sigma_{11}(\mathbf{u}) & \sigma_{12}(\mathbf{u}) \\ \sigma_{21}(\mathbf{u}) & \sigma_{22}(\mathbf{u}) \end{bmatrix} : \begin{bmatrix} \frac{\partial v_x}{\partial x} & \frac{\partial v_x}{\partial y} \\ \frac{\partial v_y}{\partial x} & \frac{\partial v_y}{\partial y} \end{bmatrix} \\ &= \sigma_{11}(\mathbf{u}) \frac{\partial v_x}{\partial x} + \sigma_{12}(\mathbf{u}) \frac{\partial v_x}{\partial y} + \sigma_{21}(\mathbf{u}) \frac{\partial v_y}{\partial x} + \sigma_{22}(\mathbf{u}) \frac{\partial v_y}{\partial y} \\ &= \left(\lambda \frac{\partial u_x}{\partial x} + 2\mu \frac{\partial u_x}{\partial x} + \lambda \frac{\partial u_y}{\partial y} \right) \frac{\partial v_x}{\partial x} \\ &\quad + \left(\mu \frac{\partial u_x}{\partial y} + \mu \frac{\partial u_y}{\partial x} \right) \frac{\partial v_x}{\partial y} + \left(\mu \frac{\partial u_x}{\partial y} + \mu \frac{\partial u_y}{\partial x} \right) \frac{\partial v_y}{\partial x} \\ &\quad + \left(\lambda \frac{\partial u_x}{\partial x} + \lambda \frac{\partial u_y}{\partial y} + 2\mu \frac{\partial u_y}{\partial y} \right) \frac{\partial v_y}{\partial y} \end{aligned} \quad (4.32)$$

所以可以得到方程等号左边为：

$$\begin{aligned} &\int_{\Omega} \sigma(\mathbf{u}) : \nabla \mathbf{v} dx dy \\ &= \int_{\Omega} \left(\lambda \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + 2\mu \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + \lambda \frac{\partial u_y}{\partial y} \frac{\partial v_x}{\partial x} \right. \\ &\quad + \mu \frac{\partial u_x}{\partial y} \frac{\partial v_x}{\partial y} + \mu \frac{\partial u_y}{\partial x} \frac{\partial v_x}{\partial y} + \mu \frac{\partial u_x}{\partial y} \frac{\partial v_y}{\partial x} + \mu \frac{\partial u_y}{\partial x} \frac{\partial v_y}{\partial x} \\ &\quad \left. + \lambda \frac{\partial u_x}{\partial x} \frac{\partial v_y}{\partial y} + \lambda \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} + 2\mu \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} \right) dx dy \end{aligned} \quad (4.33)$$

同理可以得到方程等号右边为：

$$\int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy = \int_{\Omega} (f_x v_x + f_y v_y) dx dy \quad (4.34)$$

所以可以得到方程 (4.31) 的标量形式：

$$\begin{aligned} & \int_{\Omega} \left(\lambda \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + 2\mu \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + \lambda \frac{\partial u_y}{\partial y} \frac{\partial v_x}{\partial x} \right. \\ & \quad + \mu \frac{\partial u_x}{\partial y} \frac{\partial v_x}{\partial y} + \mu \frac{\partial u_y}{\partial x} \frac{\partial v_x}{\partial y} + \mu \frac{\partial u_x}{\partial y} \frac{\partial v_y}{\partial x} + \mu \frac{\partial u_y}{\partial x} \frac{\partial v_y}{\partial x} \\ & \quad \left. + \lambda \frac{\partial u_x}{\partial x} \frac{\partial v_y}{\partial y} + \lambda \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} + 2\mu \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} \right) dx dy \\ & = \int_{\Omega} (f_x v_x + f_y v_y) dx dy \end{aligned} \quad (4.35)$$

其中， $\mathbf{u} \in H^1(\Omega) \times H^1(\Omega)$ 对任意 $\mathbf{v} \in H_0^1(\Omega) \times H_0^1(\Omega)$ 成立。

4.3.2 方程离散

令 $a(\mathbf{u}, \mathbf{v}) = \int_{\Omega} \sigma(\mathbf{u}) : \nabla \mathbf{v} dx dy$, $(\mathbf{f}, \mathbf{v}) = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy$, 待求问题为找到 $\mathbf{u} \in H^1(\Omega) \times H^1(\Omega)$ 对任意 $\mathbf{v} \in H_0^1(\Omega) \times H_0^1(\Omega)$ 都成立：

$$a(\mathbf{u}, \mathbf{v}) = (\mathbf{f}, \mathbf{v}) \quad (4.36)$$

所以在有限元空间上，对任意 $\mathbf{v}_h \in U_h \times U_h$ 有：

$$\begin{aligned} a(\mathbf{u}_h, \mathbf{v}_h) &= (\mathbf{f}, \mathbf{v}_h) \\ \longleftrightarrow \int_{\Omega} \sigma(\mathbf{u}_h) : \nabla \mathbf{v}_h dx dy &= \int_{\Omega} \mathbf{f} \cdot \mathbf{v}_h dx dy \end{aligned} \quad (4.37)$$

也容易得到标量形式在有限元空间上的表达式，这里不再赘述。

4.3.3 线性系统构建

(1) 刚度矩阵组装

将待求函数中未知变量用有限元基函数展开，得到：

$$u_{xh} = \sum_{i=1}^{N_b} u_{xi} \phi_i, \quad u_{yh} = \sum_{j=1}^{N_b} u_{yj} \phi_j \quad (4.38)$$

选取合适的基函数，得到关于 u_{xi} 和 u_{yj} 的线性系统，然后求解该线性系统即可得到有限元解： $\mathbf{u}_h = (u_{xh}, u_{yh})^T$ 。

在选择基函数时，分别对所得弱格式的两个分量进行处理：分别将 $\mathbf{v}_h = (\phi_i, 0)^T$ ($i = 1, \dots, N_b$) 和 $\mathbf{v}_h = (0, \phi_i)^T$ ($i = 1, \dots, N_b$) 代入弱格式，即对第一组测试函数取 $v_{xh} = \phi_i$ ($i = 1, \dots, N_b$) 和 $v_{yh} = 0$ ，对第二组测试函数取 $v_{xh} = 0$ 和 $v_{yh} = \phi_i$ ($i = 1, \dots, N_b$)，两组测试函数对应弱格式如下：

► 令 $\mathbf{v}_h = (\phi_i, 0)^T$, 即 $v_{xh} = \phi_i$ 和 $v_{yh} = 0$ ($i = 1, \dots, N_b$), 得到:

$$\begin{aligned} & \int_{\Omega} \lambda \left(\sum_{j=1}^{N_b} u_{xj} \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial x} dx dy + 2 \int_{\Omega} \mu \left(\sum_{j=1}^{N_b} u_{xj} \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial x} dx dy \\ & + \int_{\Omega} \lambda \left(\sum_{j=1}^{N_b} u_{yj} \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \mu \left(\sum_{j=1}^{N_b} u_{yj} \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial y} dx dy \\ & + \int_{\Omega} \mu \left(\sum_{j=1}^{N_b} u_{yj} \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial y} dx dy = \int_{\Omega} f_x \phi_i dx dy \end{aligned} \quad (4.39)$$

► 令 $\mathbf{v}_h = (0, \phi_i)^T$, 即 $v_{xh} = 0$ 和 $v_{yh} = \phi_i$ ($i = 1, \dots, N_b$), 得到:

$$\begin{aligned} & \int_{\Omega} \mu \left(\sum_{j=1}^{N_b} u_{xj} \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \mu \left(\sum_{j=1}^{N_b} u_{yj} \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial x} dx dy \\ & + \int_{\Omega} \lambda \left(\sum_{j=1}^{N_b} u_{xj} \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial y} dx dy + \int_{\Omega} \lambda \left(\sum_{j=1}^{N_b} u_{yj} \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial y} dx dy \\ & + 2 \int_{\Omega} \mu \left(\sum_{j=1}^{N_b} u_{yj} \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial y} dx dy = \int_{\Omega} f_y \phi_i dx dy \end{aligned} \quad (4.40)$$

将上述两式进行整理后可以得到:

$$\begin{aligned} & \sum_{j=1}^{N_b} u_{xj} \left(\int_{\Omega} \lambda \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy + 2 \int_{\Omega} \mu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \mu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy \right) \\ & + \sum_{j=1}^{N_b} u_{yj} \left(\int_{\Omega} \lambda \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \mu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial y} dx dy \right) \\ & = \int_{\Omega} f_x \phi_i dx dy \end{aligned} \quad (4.41)$$

$$\begin{aligned} & \sum_{j=1}^{N_b} u_{xj} \left(\int_{\Omega} \lambda \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial y} dx dy + \int_{\Omega} \mu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial x} dx dy \right) \\ & + \sum_{j=1}^{N_b} u_{yj} \left(\int_{\Omega} \lambda \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy + 2 \int_{\Omega} \mu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy + \int_{\Omega} \mu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy \right) \\ & = \int_{\Omega} f_y \phi_i dx dy \end{aligned} \quad (4.42)$$

为了方便描述, 定义如下矩阵组:

$$\begin{aligned}\mathbf{K}_1 &= \left[\int_{\Omega} \lambda \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b}, & \mathbf{K}_2 &= \left[\int_{\Omega} \mu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b} \\ \mathbf{K}_3 &= \left[\int_{\Omega} \mu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b}, & \mathbf{K}_4 &= \left[\int_{\Omega} \lambda \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b} \\ \mathbf{K}_5 &= \left[\int_{\Omega} \mu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b}, & \mathbf{K}_6 &= \left[\int_{\Omega} \lambda \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b} \\ \mathbf{K}_7 &= \left[\int_{\Omega} \mu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b}, & \mathbf{K}_8 &= \left[\int_{\Omega} \lambda \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b}\end{aligned}\tag{4.43}$$

- ▶ 令算法 6 中 $c = \lambda, r = 1, s = 0, p = 1, q = 0$, 得到矩阵 \mathbf{K}_1 ;
- ▶ 令算法 6 中 $c = \mu, r = 1, s = 0, p = 1, q = 0$, 得到矩阵 \mathbf{K}_2 ;
- ▶ 令算法 6 中 $c = \mu, r = 0, s = 1, p = 0, q = 1$, 得到矩阵 \mathbf{K}_3 ;
- ▶ 令算法 6 中 $c = \lambda, r = 0, s = 1, p = 1, q = 0$, 得到矩阵 \mathbf{K}_4 ;
- ▶ 令算法 6 中 $c = \mu, r = 1, s = 0, p = 0, q = 1$, 得到矩阵 \mathbf{K}_5 ;
- ▶ 令算法 6 中 $c = \lambda, r = 1, s = 0, p = 0, q = 1$, 得到矩阵 \mathbf{K}_6 ;
- ▶ 令算法 6 中 $c = \mu, r = 0, s = 1, p = 1, q = 0$, 得到矩阵 \mathbf{K}_7 ;
- ▶ 令算法 6 中 $c = \lambda, r = 0, s = 1, p = 0, q = 1$, 得到矩阵 \mathbf{K}_8 ;

根据线性系统对上述矩阵组进行整理可以得到:

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_1 + 2\mathbf{K}_2 + \mathbf{K}_3 & \mathbf{K}_4 + \mathbf{K}_5 \\ \mathbf{K}_6 + \mathbf{K}_7 & \mathbf{K}_8 + 2\mathbf{K}_3 + \mathbf{K}_2 \end{bmatrix}\tag{4.44}$$

(2) 载荷向量组装

定义载荷向量如下:

$$\mathbf{F} = \begin{bmatrix} \mathbf{F}_x \\ \mathbf{F}_y \end{bmatrix} \longrightarrow \mathbf{F}_x = \left[\int_{\Omega} f_x \phi_i dx dy \right]_{i=1}^{N_b}, \quad \mathbf{F}_y = \left[\int_{\Omega} f_y \phi_i dx dy \right]_{i=1}^{N_b}\tag{4.45}$$

- ▶ 令算法 7 中 $f = f_x, p = q = 0$, 得到载荷向量阵 \mathbf{F}_x ;
- ▶ 令算法 7 中 $f = f_y, p = q = 0$, 得到载荷向量阵 \mathbf{F}_y ;

(3) 未知向量

定义待求向量如下:

$$\mathbf{u} = \begin{bmatrix} \mathbf{u}_x \\ \mathbf{u}_y \end{bmatrix} \longrightarrow \mathbf{u}_x = [u_{xi}]_{i=1}^{N_b}, \quad u_y = [\mathbf{u}_{yi}]_{i=1}^{N_b}\tag{4.46}$$

(4) 线性系统组成

根据得到的刚度矩阵、载荷向量和未知向量，可以得到线性系统：

$$\mathbf{K}\mathbf{u} = \mathbf{F} \quad (4.47)$$

得到线性系统后根据所给问题模型的边界条件进行处理，得到最终需要求解的线性系统，然后求解该线性系统即可得到有限元解，在得到位移后可以计算得到相应应力、应变等物理量。

4.4 不同边界条件处理

4.4.1 第1类边界条件

线弹性问题中待求量矢量，所以在进行边界处理时需要同时对矢量分量边界进行处理，这里处理方式仍然采用置1法，详细算法流程如算法 11 所示：

Algorithm 11 : 2D 线弹性方程 Dirichlet 边界处理

```

1: for  $k = 1, \dots, N_{BN}$  do
2:   if  $BC[0, k] ==$  Dirichlet then
3:      $i = BC[1, k]$ 
4:      $\mathbf{K}[i, :] = 0$ 
5:      $\mathbf{K}[i, i] = 1$ 
6:      $\mathbf{F}[i] = g_x(Pb[:, i])$ 
7:      $\mathbf{K}[N_b + i, :] = 0$ 
8:      $\mathbf{K}[N_b + i, N_b + i] = 1$ 
9:      $\mathbf{F}[N_b + i] = g_y(Pb[:, i])$ 
10:   end if
11: end for

```

4.4.2 第2类边界条件

在线弹性问题中第2类边界条件为应力边界，此时控制方程为：

$$\begin{cases} -\nabla \cdot \sigma(\mathbf{u}) = \mathbf{f}, & (x, y) \in \Omega \\ \sigma(\mathbf{u})\mathbf{n} = \mathbf{p}, & (x, y) \in \partial\Omega \end{cases} \quad (4.48)$$

其中, $\mathbf{n} = (n_1, n_2)^T$ 为边界 $\partial\Omega$ 的法向量, $\sigma(\mathbf{u})$ 为应力张量, $\mathbf{f}(x, y) = (f_x, f_y)^T$ 为外力, $\mathbf{p}(x, y) = (p_x, p_y)^T$ 为应力。

根据散度定理及给定边界条件, 控制方程等价为:

$$\int_{\Omega} \sigma(\mathbf{u}) : \nabla \mathbf{v} dx dy - \int_{\partial\Omega} \mathbf{p} \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \quad (4.49)$$

可以看到由式(4.49)得到的解并不唯一! 很显然载荷向量一侧的大小, 由于测试函数 \mathbf{v} 的选择不同将产生不同的结果, 所以该情况下所得方程的解不唯一。从实际工程来看, 物体施加一定大小的力, 如果其空间位置不固定, 则该物体在空间位置将无法固定。

所以通常对于上述问题边界条件为 Dirichlet + Neumann 的混合边界条件, 由此控制方程变换为:

$$\begin{cases} -\nabla \cdot \sigma(\mathbf{u}) = \mathbf{f}, & (x, y) \in \Omega \\ \sigma(\mathbf{u}) \mathbf{n} = \mathbf{p}, & \Gamma_S \subset \partial\Omega \\ \mathbf{u} = \mathbf{g}, & \Gamma_D = \partial\Omega / \Gamma_S \end{cases} \quad (4.50)$$

由于边界 $\Gamma_D = \partial\Omega / \Gamma_S$ 上的解由式 $\mathbf{u} = \mathbf{g}$ 确定, 所以, 可以选择合适的测试函数使其在边界上满足 $\mathbf{v} = 0$ 。由边界条件可以得到:

$$\int_{\partial\Omega} (\sigma(\mathbf{u}) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Gamma_S} (\sigma(\mathbf{u}) \mathbf{n}) \cdot \mathbf{v} ds + \int_{\partial\Omega / \Gamma_S} (\sigma(\mathbf{u}) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Gamma_S} \mathbf{f} \cdot \mathbf{v} ds \quad (4.51)$$

因此, 方程的弱格式就是找到 $\mathbf{u} \in H^1(\Omega) \times H^1(\Omega)$ 使其对任何 $\mathbf{v} \in H_{0D}^1(\Omega) \times H_{0D}^1(\Omega)$ 都有:

$$\int_{\Omega} \sigma(\mathbf{u}) : \nabla \mathbf{v} dx dy = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy + \int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds \quad (4.52)$$

其中, $\int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds = \int_{\Gamma_S} p_x v_x ds + \int_{\Gamma_S} p_y v_y ds$, $H_{0D}^1(\Omega) = \{\mathbf{v} \in H^1(\Omega) : \mathbf{v} = 0 \text{ on } \Gamma_D\}$ 。

所以, 式(4.52)即为包含第二类边界条件时待求方程的弱格式, 仅需要在 Neumann 边界对应的载荷向量处增加 $\int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds$ 即可。

4.4.3 第3类边界条件

$$\begin{cases} -\nabla \cdot \sigma(\mathbf{u}) = \mathbf{f}, & (x, y) \in \Omega \\ \sigma(\mathbf{u}) \mathbf{n} + r \mathbf{u} = \mathbf{q}, & \Gamma_R \subseteq \partial\Omega \\ \mathbf{u} = \mathbf{g}, & \Gamma_D = \partial\Omega / \Gamma_R \end{cases} \quad (4.53)$$

其中, $\mathbf{n} = (n_1, n_2)^T$ 为边界 Γ_R 的法向量。

由于边界 $\Gamma_D = \partial\Omega/\Gamma_R$ 上的解由式 $\mathbf{u} = \mathbf{g}$ 确定，所以，可以选择合适的测试函数使其在边界上满足 $\mathbf{v} = 0$ 。由边界条件可以得到：

$$\begin{aligned}\int_{\partial\Omega} (\sigma(\mathbf{u}) \mathbf{n}) \cdot \mathbf{v} ds &= \int_{\Gamma_R} (\sigma(\mathbf{u}) \mathbf{n}) \cdot \mathbf{v} ds + \int_{\partial\Omega/\Gamma_R} (\sigma(\mathbf{u}) \mathbf{n}) \cdot \mathbf{v} ds \\ &= \int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds - \int_{\Gamma_R} r\mathbf{u} \cdot \mathbf{v} ds\end{aligned}\quad (4.54)$$

因此，方程的弱格式就是找到 $\mathbf{u} \in H^1(\Omega) \times H^1(\Omega)$ 使其对任何 $\mathbf{v} \in H_{0D}^1(\Omega) \times H_{0D}^1(\Omega)$ 都有：

$$\int_{\Omega} \sigma(\mathbf{u}) : \nabla \mathbf{v} dx dy + \int_{\Gamma_R} r\mathbf{u} \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy + \int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds \quad (4.55)$$

其中， $\int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds = \int_{\Gamma_R} q_x v_x ds + \int_{\Gamma_R} q_y v_y ds$, $\int_{\Gamma_R} r\mathbf{u} \cdot \mathbf{v} ds = \int_{\Gamma_R} r u_x v_x ds + \int_{\Gamma_R} r u_y v_y ds$, $H_{0D}^1(\Omega) = \{\mathbf{v} \in H^1(\Omega) : \mathbf{v} = 0 \text{ on } \Gamma_D\}$ 。

所以，式(4.55)即为包含第三类边界条件时待求方程的弱格式，需要在 Robin 边界对应刚度矩阵处增加 $\int_{\Gamma_R} r\mathbf{u} \cdot \mathbf{v} ds$ ，在载荷向量处增加 $\int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds$ 。

4.4.4 混合边界条件

$$\left\{ \begin{array}{l} -\nabla \cdot \sigma(\mathbf{u}) = \mathbf{f}, \quad (x, y) \in \Omega \\ \sigma(\mathbf{u}) \mathbf{n} = \mathbf{p}, \quad \Gamma_S \subset \partial\Omega \\ \sigma(\mathbf{u}) \mathbf{n} + r\mathbf{u} = \mathbf{q}, \quad \Gamma_R \subseteq \partial\Omega \\ \mathbf{u} = \mathbf{g}, \quad \Gamma_D = \partial\Omega / (\Gamma_S \cup \Gamma_R) \end{array} \right. \quad (4.56)$$

其中， $\mathbf{n} = (n_1, n_2)^T$ 为边界 $\Gamma_S \cup \Gamma_R$ 的法向量。

由前述推导可知：

$$\int_{\Omega} \sigma(\mathbf{u}) : \nabla \mathbf{v} dx dy - \int_{\partial\Omega} (\sigma(\mathbf{u}) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \quad (4.57)$$

由于边界 $\Gamma_D = \partial\Omega / (\Gamma_S \cup \Gamma_R)$ 上的解由式 $\mathbf{u} = \mathbf{g}$ 确定，所以，可以选择合适的测试函数使其在该边界上满足 $\mathbf{v} = 0$ ，可以得到：

$$\begin{aligned}\int_{\partial\Omega} (\sigma(\mathbf{u}) \mathbf{n}) \cdot \mathbf{v} ds &= \int_{\Gamma_S} (\sigma(\mathbf{u}) \mathbf{n}) \cdot \mathbf{v} ds + \int_{\Gamma_R} (\sigma(\mathbf{u}) \mathbf{n}) \cdot \mathbf{v} ds \\ &+ \int_{\partial\Omega / (\Gamma_S \cup \Gamma_R)} (\sigma(\mathbf{u}) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds + \int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds - \int_{\Gamma_R} r\mathbf{u} \cdot \mathbf{v} ds\end{aligned}\quad (4.58)$$

因此，方程的弱格式就是找到 $\mathbf{u} \in H^1(\Omega) \times H^1(\Omega)$ 使其对任何 $\mathbf{v} \in H_{0D}^1(\Omega) \times H_{0D}^1(\Omega)$ 都有：

$$\int_{\Omega} \sigma(\mathbf{u}) : \nabla \mathbf{v} dx dy + \int_{\Gamma_R} r\mathbf{u} \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy + \int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds + \int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds \quad (4.59)$$

在求解该系统时，依次按照 Robin 型、Neumann 型、Dirichlet 型边界条件进行处理，始终保证 Dirichlet 边界发挥作用，具体而言为：在 Robin 型对应边界刚度矩阵增加 $\int_{\Gamma_R} \mathbf{r} \mathbf{u} \cdot \mathbf{v} ds$ ，载荷向量增加 $\int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds$ ；在 Neumann 型对应边界载荷向量增加 $\int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds$ ；对 Dirichlet 类型边界对应刚度矩阵采用“置 1”法，载荷向量为给定边界条件值。

4.5 误差计算

线弹性求解过程及程序的正确性同样通过与解析解（如果存在或者较为方便得到的话）的误差进行衡量，这里通过：1、最大节点值误差；2、无穷范数误差 L^∞ ；3、 L^2 范数误差；4、 H^1 范数误差进行衡量。与前述问题模型误差不同的是，线弹性方程为矢量方程，所求结果包含两个方向的结果，因此对应的误差计算方法也需要进行相应的调整，各误差计算方法介绍如下。

4.5.1 最大节点误差

分别计算 x 、 y 方向上最大节点误差，随后取二者中最大值作为计算域中最大节点值误差。

$$\maxNodeError = \max(\maxNodeErrorX, \maxNodeErrorY) \quad (4.60)$$

4.5.2 无穷范数误差 L^∞

无穷范数误差同样分别计算 x 、 y 方向上无穷范数误差，随后取二者中最大值作为计算域中无穷范数误差。

$$\begin{aligned} \|u_x - u_{xh}\|_\infty &= \sup_{\Omega} |u_x - u_{xh}| \\ \|u_y - u_{yh}\|_\infty &= \sup_{\Omega} |u_y - u_{yh}| \\ \rightarrow \|u - u_h\|_\infty &= \max (\|u_x - u_{xh}\|_\infty, \|u_y - u_{yh}\|_\infty) \end{aligned} \quad (4.61)$$

4.5.3 L^2 范数误差

同理可以得到 L^2 范数误差：

$$\begin{aligned}\|u_x - u_{xh}\|_0 &= \sqrt{\int_{\Omega} (u_x - u_{xh})^2 dx dy} \\ \|u_y - u_{yh}\|_0 &= \sqrt{\int_{\Omega} (u_y - u_{yh})^2 dx dy} \\ \rightarrow \|u - u_h\|_0 &= \sqrt{\|u_x - u_{xh}\|_0^2 + \|u_y - u_{yh}\|_0^2}\end{aligned}\quad (4.62)$$

4.5.4 H^1 范数误差

$$\begin{aligned}\|u_x - u_{xh}\|_1 &= \sqrt{\int_{\Omega} \left(\frac{\partial (u_x - u_{xh})}{\partial x}\right)^2 dx dy + \int_{\Omega} \left(\frac{\partial (u_x - u_{xh})}{\partial y}\right)^2 dx dy} \\ \|u_y - u_{yh}\|_1 &= \sqrt{\int_{\Omega} \left(\frac{\partial (u_y - u_{yh})}{\partial x}\right)^2 dx dy + \int_{\Omega} \left(\frac{\partial (u_y - u_{yh})}{\partial y}\right)^2 dx dy} \\ \rightarrow \|u - u_h\|_1 &= \sqrt{\|u_x - u_{xh}\|_1^2 + \|u_y - u_{yh}\|_1^2}\end{aligned}\quad (4.63)$$

4.6 应用实例及程序实现

4.6.1 Example 1

例 4.1 【问题描述】以下述 2D 线弹性方程的有限元求解对前述有限元方法进行验证，对应验证的模型满足方程如下，模型计算域为： $\Omega = [0, 1] \times [0, 1]$ ，取 $\lambda = 1, \mu = 2$ 。

$$\begin{cases} -\nabla \cdot \sigma(\mathbf{u}) = \mathbf{f}, (x, y) \in \Omega \\ u_x = 0, u_y = 0, (x, y) \in \partial\Omega \end{cases}$$

其中：

$$\begin{aligned}f_x &= -(\lambda + 2\mu)(-\pi^2 \sin(\pi x) \sin(\pi y)) \\ &\quad - (\lambda + \mu)((2x - 1)(2y - 1)) - \mu(-\pi^2 \sin(\pi x) \sin(\pi y)) \\ f_y &= -(\lambda + 2\mu)(2x(x - 1)) \\ &\quad - (\lambda + \mu)(\pi^2 \cos(\pi x) \cos(\pi y)) - \mu(2y(y - 1))\end{aligned}$$

说明】上述问题的解析解为: $u_x = \sin(\pi x) \sin(\pi y)$, $u_y = x(x - 1)y(y - 1)$, 可用于与有限元得到结果进行比对。

(1) 2D 三节点单元求解

1、网格划分

采用第 [三章 3.7.1](#) 节中图 [3-1a](#) 所示网格划分, 基本信息矩阵与对应算例一致。为方便刚度矩阵与对应载荷向量的计算将问题模型写为标量形式, 并将给定参数代入可以得到:

$$\begin{aligned} & \int_{\Omega} \left(\frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + 4 \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + \frac{\partial u_y}{\partial y} \frac{\partial v_x}{\partial x} \right. \\ & + 2 \frac{\partial u_x}{\partial y} \frac{\partial v_x}{\partial y} + 2 \frac{\partial u_y}{\partial x} \frac{\partial v_x}{\partial y} + 2 \frac{\partial u_x}{\partial y} \frac{\partial v_y}{\partial x} + 2 \frac{\partial u_y}{\partial x} \frac{\partial v_y}{\partial x} \\ & \left. + \frac{\partial u_x}{\partial x} \frac{\partial v_y}{\partial y} + \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} + 4 \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} \right) dx dy \\ & = \int_{\Omega} (f_x v_x + f_y v_y) dx dy \end{aligned} \quad (4.64)$$

2、刚度矩阵组装

根据前述章节中刚度矩阵计算方法得到刚度矩阵 $\mathbf{K}_1 \sim \mathbf{K}_8$, 随后由式 [\(4.44\)](#) 可以得到整个系统刚度矩阵, 这里不再对矩阵各元素计算过程进行详细演示, 具体计算过程可参考前述章节对应内容, 经过计算可以得到各刚度矩阵结果分别如下:

$$\mathbf{K}_1 = \begin{bmatrix} 0.5 & 0 & -0.5 & 0 \\ 0 & 0.5 & 0 & -0.5 \\ -0.5 & 0 & 0.5 & 0 \\ 0 & -0.5 & 0 & 0.5 \end{bmatrix}, \quad \mathbf{K}_2 = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \quad (4.65)$$

$$\mathbf{K}_3 = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}, \quad \mathbf{K}_4 = \begin{bmatrix} 0.5 & -0.5 & 0 & 0 \\ 0 & 0 & 0.5 & -0.5 \\ -0.5 & 0.5 & 0 & 0 \\ 0 & 0 & -0.5 & 0.5 \end{bmatrix} \quad (4.66)$$

$$\mathbf{K}_5 = \begin{bmatrix} 1 & 0 & -1 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & -1 & 0 & 1 \end{bmatrix}, \quad \mathbf{K}_6 = \begin{bmatrix} 0.5 & 0 & -0.5 & 0 \\ -0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & -0.5 \\ 0 & -0.5 & 0 & 0.5 \end{bmatrix} \quad (4.67)$$

$$\mathbf{K}_7 = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}, \quad \mathbf{K}_8 = \begin{bmatrix} 0.5 & -0.5 & 0 & 0 \\ -0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & -0.5 \\ 0 & 0 & -0.5 & 0.5 \end{bmatrix} \quad (4.68)$$

将所得 8 个刚度矩阵，按照式 (4.44) 进行合并得到系统刚度矩阵：

$$\mathbf{K} = \begin{bmatrix} 3.5 & -1 & -2.5 & 0 & 1.5 & -0.5 & -1 & 0 \\ -1 & 3.5 & 0 & -2.5 & -1 & 0 & 1.5 & -0.5 \\ -2.5 & 0 & 3.5 & -1 & -0.5 & 1.5 & 0 & -1 \\ 0 & -2.5 & -1 & 3.5 & 0 & -1 & -0.5 & 1.5 \\ 1.5 & -1 & -0.5 & 0 & 3.5 & -2.5 & -1 & 0 \\ -0.5 & 0 & 1.5 & -1 & -2.5 & 3.5 & 0 & -1 \\ -1 & 1.5 & 0 & -0.5 & -1 & 0 & 3.5 & -2.5 \\ 0 & -0.5 & -1 & 1.5 & 0 & -1 & -2.5 & 3.5 \end{bmatrix} \quad (4.69)$$

3、载荷向量组装

由前所述可知载荷向量为：

$$\mathbf{F} = \begin{bmatrix} \mathbf{F}_x \\ \mathbf{F}_y \end{bmatrix} \longrightarrow \mathbf{F}_x = \left[\int_{\Omega} f_x \phi_i dx dy \right]_{i=1}^{N_b}, \quad \mathbf{F}_y = \left[\int_{\Omega} f_y \phi_i dx dy \right]_{i=1}^{N_b} \quad (4.70)$$

由此分别得到 x 和 y 的载荷向量分别如下：

$$\mathbf{F}_x = \begin{bmatrix} 3.3274 \\ 10.3348 \\ 11.0325 \\ 3.3274 \end{bmatrix}, \quad \mathbf{F}_y = \begin{bmatrix} -1.1137 \\ 2.4246 \\ 2.1356 \\ -1.1137 \end{bmatrix} \quad (4.71)$$

将 x 、 y 方向载荷向量合并得到整体载荷向量：

$$\mathbf{F} = \begin{bmatrix} 3.3274 & 10.3348 & 11.0325 & 3.3274 & -1.1137 & 2.4246 & 2.1356 & -1.1137 \end{bmatrix}^T \quad (4.72)$$

4、边界条件处理及线性系统求解：本问题中边界类型为 Dirichlet，采用前述章节中“置 1”法进行处理。容易知道，采用当前网格，各节点均为 Dirichlet 边界，问题模型中该边界处各节点对应函数值均为 0。

为了有效对比本章所述有限元方法在线弹性方程中的正确性，将网格细化为 15×15 ，之后将有限元仿真结果与解析解进行对比：图 4-1a 和图 4-1b 分别为有限元方法和解析解的 x 方向节点位移；图 4-2a 和图 4-2b 分别为有限元方法和解析解的 y 方向节点位移，可以看到在一定数量的网格划分情况下，所得有限元解与解析解较为接近，说明所述方法对可以有效求解线弹性问题。表 4-1 给出了不同空间步长下，2D FEM 采用线性三角形网格单元时各类误差范数的结果参考。

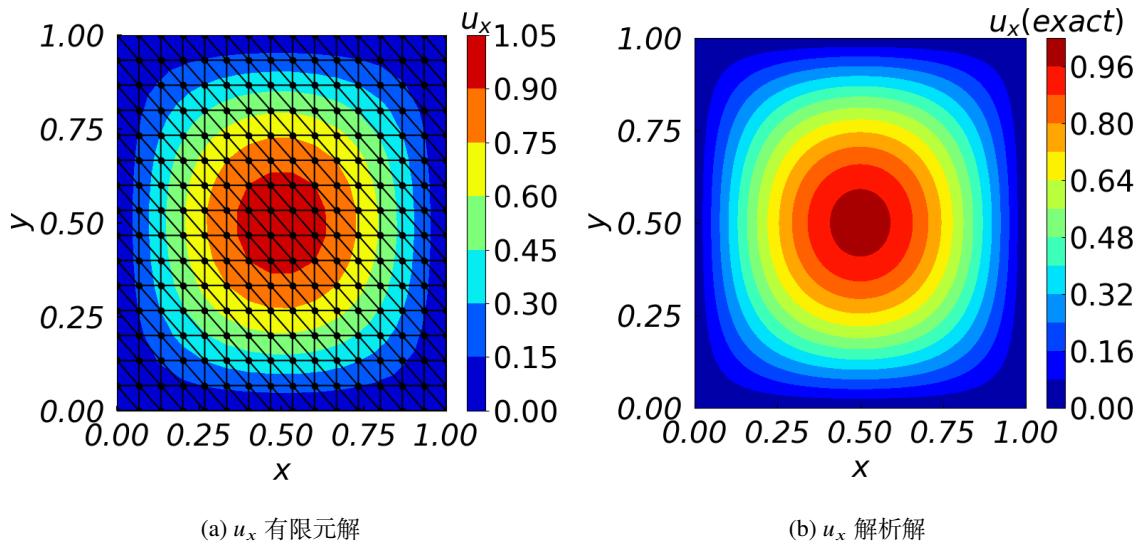
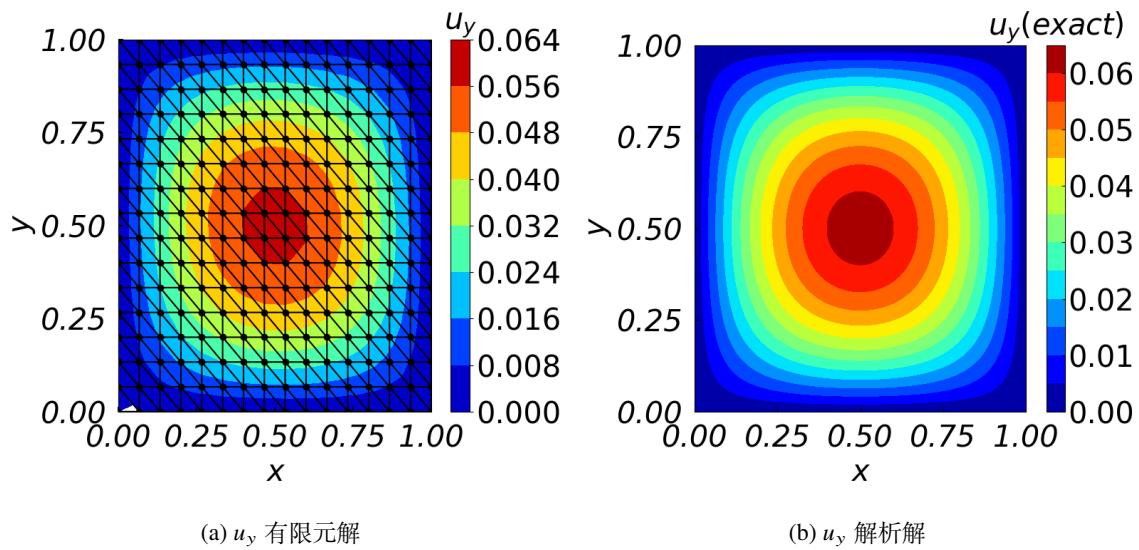
图 4-1 Example 1 三节点单元 u_x 图 4-2 Example 1 三节点单元 u_y

表 4-1 线弹性问题模型三节点单元不同空间步长下各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	1.4542×10^{-2}	5.1175×10^{-2}	2.2934×10^{-2}	4.3382×10^{-1}
1/16	3.7830×10^{-3}	1.3250×10^{-2}	5.9217×10^{-3}	2.1821×10^{-1}
1/32	9.5640×10^{-4}	3.3437×10^{-3}	1.4938×10^{-3}	1.0926×10^{-1}
1/64	2.3980×10^{-5}	8.3793×10^{-4}	3.7431×10^{-4}	5.4649×10^{-2}

(2) 2D 六节点单元求解

1、网格划分

同样为了方便计算与方法对比，采用第 [三 章 3.7.1](#) 节图 3-3a 所示网格划分，网格基本信息矩阵与前述章节一致，本问题模型中控制方程的标量形式如前述 2D3 节点方法中所示，接下来按照 2D6 节点有限元方法依次得到式 (4.43) 中各刚度矩阵，将其合并得到系统刚度矩阵。

2、刚度矩阵组装

这里直接给出对应刚度矩阵作为参考：

$$\mathbf{K}_1 = \begin{bmatrix} 0.5 & 0 & 0 & -0.7 & 0 & 0 & 0.2 & 0 & 0 \\ 0 & 1.3 & 0 & 0 & -1.3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & -0.7 & 0 & 0 & 0.2 \\ -0.7 & 0 & 0 & 1.3 & 0 & 0 & -0.7 & 0 & 0 \\ 0 & -1.3 & 0 & 0 & 2.7 & 0 & 0 & -1.3 & 0 \\ 0 & 0 & -0.7 & 0 & 0 & 1.3 & 0 & 0 & -0.7 \\ 0.2 & 0 & 0 & -0.7 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1.3 & 0 & 0 & 1.3 & 0 \\ 0 & 0 & 0.2 & 0 & 0 & -0.7 & 0 & 0 & 0.5 \end{bmatrix} \quad (4.73)$$

$$\mathbf{K}_2 = \begin{bmatrix} 1 & 0 & 0 & -1.3 & 0 & 0 & 0.3 & 0 & 0 \\ 0 & 2.7 & 0 & 0 & -2.7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1.3 & 0 & 0 & 0.3 \\ -1.3 & 0 & 0 & 2.7 & 0 & 0 & -1.3 & 0 & 0 \\ 0 & -2.7 & 0 & 0 & 5.3 & 0 & 0 & -2.7 & 0 \\ 0 & 0 & -1.3 & 0 & 0 & 2.7 & 0 & 0 & -1.3 \\ 0.3 & 0 & 0 & -1.3 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2.7 & 0 & 0 & 2.7 & 0 \\ 0 & 0 & 0.3 & 0 & 0 & -1.3 & 0 & 0 & 1 \end{bmatrix} \quad (4.74)$$

$$\mathbf{K}_3 = \begin{bmatrix} 0 & -1.3 & 0.3 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1.3 & 2.7 & -1.3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.3 & -1.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2.7 & -2.7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2.7 & 5.3 & -2.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2.7 & 2.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.3 & 0.3 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1.3 & 2.7 & -1.3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.3 & 0 \end{bmatrix} \quad (4.75)$$

$$\mathbf{K}_4 = \begin{bmatrix} 0.5 & -0.7 & 0.2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.7 & -0.7 & 0.7 & -0.7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.7 & -0.7 & -0.2 & 0 & 0.2 \\ -0.7 & 0.7 & 0 & 0.7 & -0.7 & 0 & 0 & 0 & 0 \\ 0 & -0.7 & 0.7 & -0.7 & 1.3 & -0.7 & 0.7 & -0.7 & 0 \\ 0 & 0 & 0 & 0 & -0.7 & 0.7 & 0 & 0.7 & -0.7 \\ 0.2 & 0 & -0.2 & -0.7 & 0.7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.7 & 0.7 & -0.7 & 0.7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.2 & -0.7 & 0.5 \end{bmatrix} \quad (4.76)$$

$$\mathbf{K}_5 = \begin{bmatrix} 1 & 0 & 0 & -1.3 & 0 & 0 & 0.3 & 0 & 0 \\ -1.3 & 1.3 & 0 & 1.3 & -1.3 & 0 & 0 & 0 & 0 \\ 0.3 & -1.3 & 0 & 0 & 1.3 & 0 & -0.3 & 0 & 0 \\ 0 & 1.3 & 0 & 1.3 & -1.3 & 0 & -1.3 & 0 & 0 \\ 0 & -1.3 & 1.3 & -1.3 & 2.7 & -1.3 & 1.3 & -1.3 & 0 \\ 0 & 0 & -1.3 & 0 & -1.3 & 1.3 & 0 & 1.3 & 0 \\ 0 & 0 & -0.3 & 0 & 1.3 & 0 & 0 & -1.3 & 0.3 \\ 0 & 0 & 0 & 0 & -1.3 & 1.3 & 0 & 1.3 & -1.3 \\ 0 & 0 & 0.3 & 0 & 0 & -1.3 & 0 & 0 & 1 \end{bmatrix} \quad (4.77)$$

$$\mathbf{K}_6 = \begin{bmatrix} 0.5 & 0 & 0 & -0.7 & 0 & 0 & 0.2 & 0 & 0 \\ -0.7 & 0.7 & 0 & 0.7 & -0.7 & 0 & 0 & 0 & 0 \\ 0.2 & -0.7 & 0 & 0 & 0.7 & 0 & -0.2 & 0 & 0 \\ 0 & 0.7 & 0 & 0.7 & -0.7 & 0 & -0.7 & 0 & 0 \\ 0 & -0.7 & 0.7 & -0.7 & 1.3 & -0.7 & 0.7 & -0.7 & 0 \\ 0 & 0 & -0.7 & 0 & -0.7 & 0.7 & 0 & 0.7 & 0 \\ 0 & 0 & -0.2 & 0 & 0.7 & 0 & 0 & -0.7 & 0.2 \\ 0 & 0 & 0 & 0 & -0.7 & 0.7 & 0 & 0.7 & -0.7 \\ 0 & 0 & 0.2 & 0 & 0 & -0.7 & 0 & 0 & 0.5 \end{bmatrix} \quad (4.78)$$

$$\mathbf{K}_7 = \begin{bmatrix} 1 & -1.3 & 0.3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.3 & -1.3 & 1.3 & -1.3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.3 & -1.3 & -0.3 & 0 & 0.3 \\ -1.3 & 1.3 & 0 & 1.3 & -1.3 & 0 & 0 & 0 & 0 \\ 0 & -1.3 & 1.3 & -1.3 & 2.7 & -1.3 & 1.3 & -1.3 & 0 \\ 0 & 0 & 0 & 0 & -1.3 & 1.3 & 0 & 1.3 & -1.3 \\ 0.3 & 0 & -0.3 & -1.3 & 1.3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1.3 & 1.3 & -1.3 & 1.3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & -1.3 & 1 \end{bmatrix} \quad (4.79)$$

$$\mathbf{K}_8 = \begin{bmatrix} 0.5 & -0.7 & 0.2 & 0 & 0 & 0 & 0 & 0 \\ -0.7 & 1.3 & -0.7 & 0 & 0 & 0 & 0 & 0 \\ 0.2 & -0.7 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.3 & -1.3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1.3 & 2.7 & -1.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1.3 & 1.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & -0.7 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.7 & 1.3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.2 & -0.7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix} \quad (4.80)$$

将上述矩阵根据式 (4.44) 合并即可得到系统刚度矩阵。

3、载荷向量组装

类比前述 2D3 节点载荷向量，得到 2D6 节点单元时对应载荷向量如下：

$$\begin{aligned} \mathbf{F}_x &= \begin{bmatrix} -0.9028 & 4.1820 & -1.0376 & 4.2785 & 14.3806 & 4.1820 & -0.4363 & 4.2785 & -0.9028 \end{bmatrix}^T \\ \mathbf{F}_y &= \begin{bmatrix} -0.9907 & -0.1585 & 1.5736 & -0.0874 & 1.9523 & -0.0918 & 1.2801 & -0.1541 & -0.9907 \end{bmatrix}^T \end{aligned} \quad (4.81)$$

4、边界条件处理、线性系统求解

采用如前所述 2D3 节点中矩阵处理方法，根据网格划分情况，容易知道仅有节点 5 为内部节点，即仅该节点对应矩阵系数不为 1，其余节点对应矩阵系数均为 1。对载荷向量而言，仅该节点对应值不为 0。经边界处理后可以得到各节点位移分别为：

$$\begin{aligned} \mathbf{u} &= \begin{bmatrix} \mathbf{u}_x \\ \mathbf{u}_y \end{bmatrix} \\ \mathbf{u}_x &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0.7840 & 0 & 0 & 0 & 0 \end{bmatrix}^T \\ \mathbf{u}_y &= \begin{bmatrix} 0 & 0 & 0 & 0 & -0.0634 & 0 & 0 & 0 & 0 \end{bmatrix}^T \end{aligned} \quad (4.82)$$

5、后处理

将所得有限元解与解析解进行对比，各节点 x 方向位移分别如图 4-3a 和图 4-3b 所示， y 方向各节点位移分别如图 4-4a 和图 4-4b 所示。可以看到在当前网格划分情况下，所得有限元解与解析解相差较大。网格细化后所得有限元解与解析解误差将有效减小，图 4-5a 和图 4-5b 分别展示了 2D6 节点网格单元为 10×10 时的 u_x 和 u_y 位移，可以看到所得有限元解与解析解误差已经大幅减小。在得到各节点位移后可由式 (4.8) 得到各节点应变，由式 (4.13) 得到各节点应力，至此线弹性方程基本问题求解完成。

表 4-2 为不同空间步长二维六节点单元各类误差范数参考。

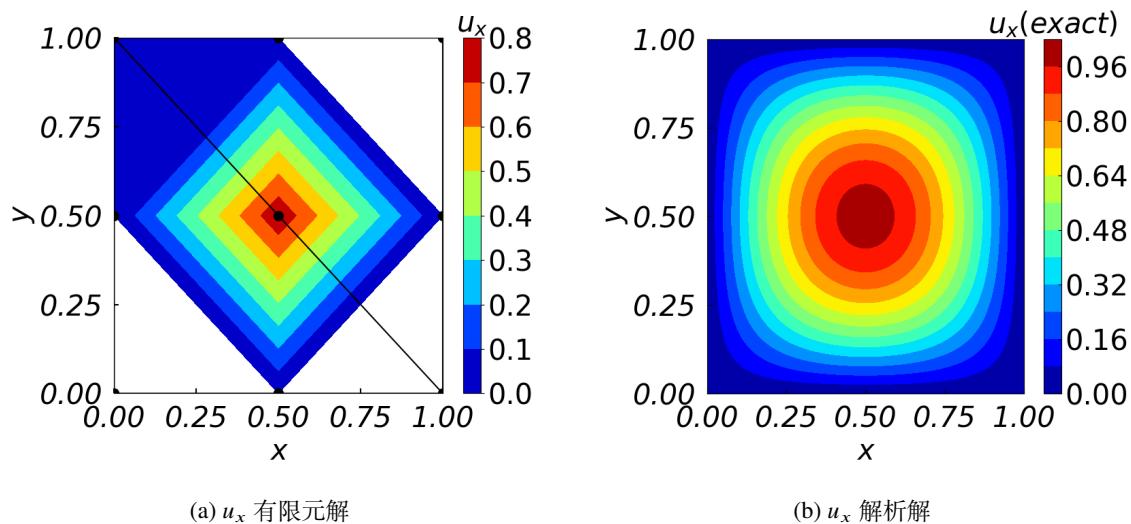
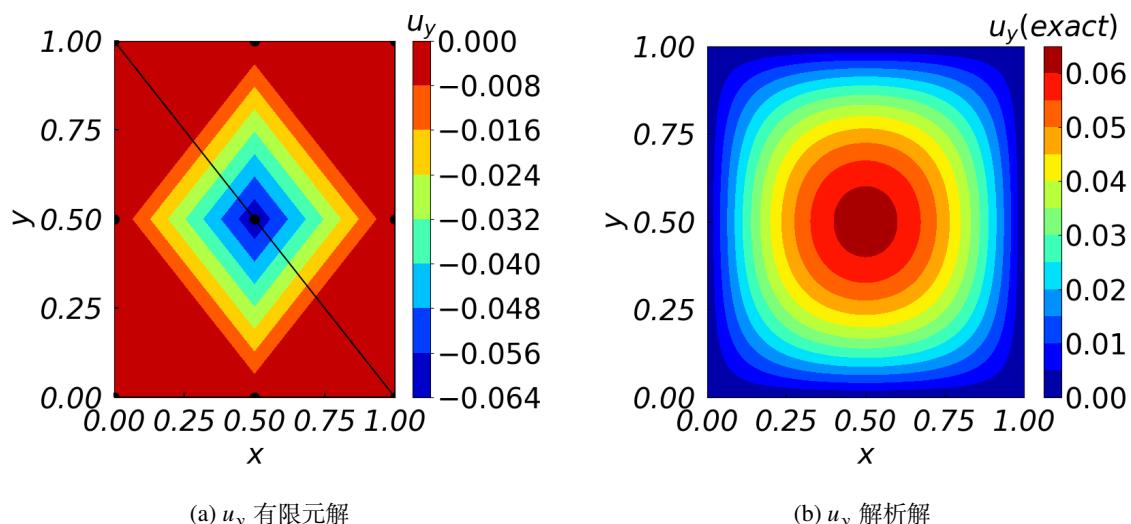
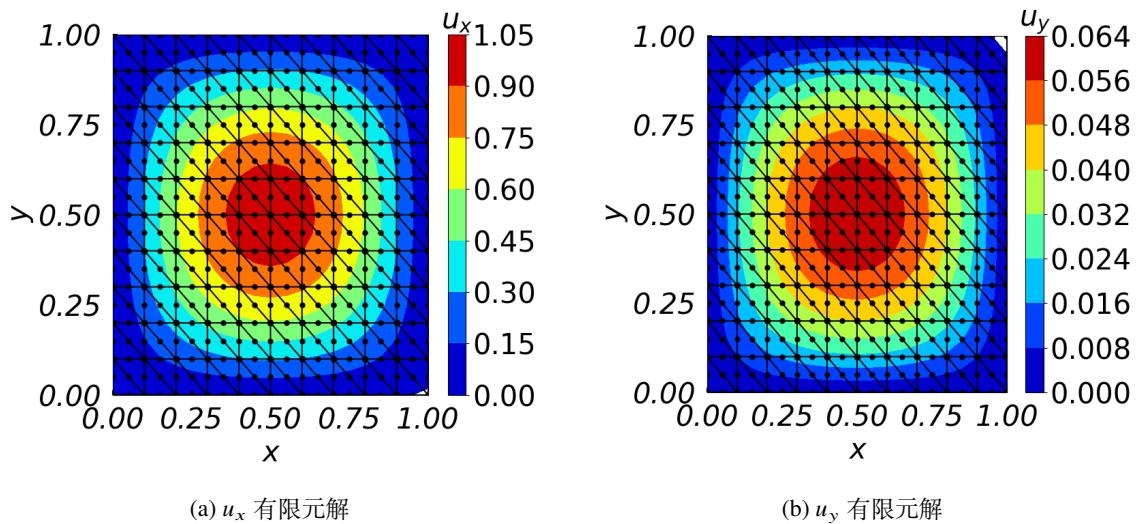
图 4-3 Example 1 六节点单元 u_x 图 4-4 Example 1 六节点单元 u_y

表 4-2 Example 1 2D 六节点单元不同空间步长下各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	2.6628×10^{-4}	1.4862×10^{-3}	5.0157×10^{-4}	3.3555×10^{-2}
1/16	1.6764×10^{-5}	1.8944×10^{-4}	6.2157×10^{-5}	8.4431×10^{-3}
1/32	1.0488×10^{-6}	2.3799×10^{-5}	7.7475×10^{-6}	2.1142×10^{-3}
1/64	6.5549×10^{-8}	2.9797×10^{-6}	9.6770×10^{-7}	5.2876×10^{-4}

图 4-5 10×10 六节点单元各节点位移

4.6.2 Example 2

例 4.2 【问题描述】如图 4-6 所示为一带孔薄板，平板左侧固定，右侧施加大小为 $\mathbf{F} = 100\text{N/m}$ 均布力，通过有限元方法计算薄板位移、应力、应变大小。杨氏模量 $E = 2 \times 10^{10}\text{Pa}$ ，泊松比 $\nu = 0.3$ 。

$$\begin{cases} -\nabla \cdot \sigma(\mathbf{u}) = 0, & (x, y) \in \Omega \\ \mathbf{F} = 100\text{N/m}, & (x = 10, y \in [0, 5]) \\ \mathbf{u} = (0, 0), & (x = 0, y \in [0, 5]) \end{cases}$$

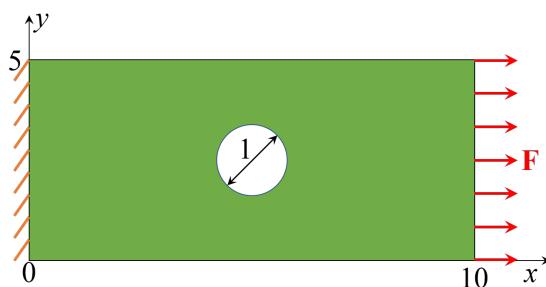


图 4-6 带孔板仿真模型

1、网格划分

本算例模型中网格划分如图 4-7 所示，这里采用非结构网格，其划分方式可参考相

关网格划分算法及相关软件，网格单元为四边形单元。

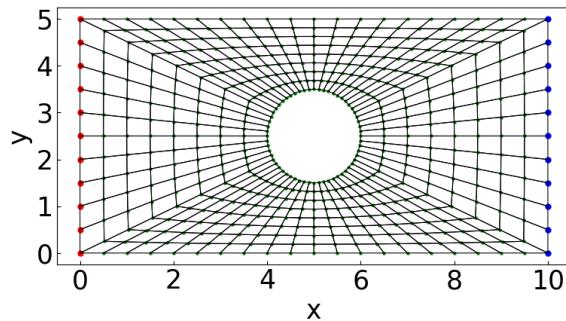


图 4-7 带孔板模型网格划分示意

2、刚度矩阵组装

本问题为平面应力问题，根据给定杨氏模量与泊松比计算得到弹性矩阵 \mathbf{D} :

$$\mathbf{D} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \quad (4.83)$$

对每一个单元取其坐标矩阵:

$$\mathbf{x}^e = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix} \quad (4.84)$$

根据单元对应基函数，计算对应雅可比矩阵、对应行列式及其逆:

$$\mathbf{J}^e = \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \xi} & \frac{\partial N_3}{\partial \xi} & \frac{\partial N_4}{\partial \xi} \\ \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \frac{\partial N_3}{\partial \eta} & \frac{\partial N_4}{\partial \eta} \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix} \quad (4.85)$$

$$\det(\mathbf{J}^e) = \mathbf{J}_{11}^e \mathbf{J}_{22}^e - \mathbf{J}_{12}^e \mathbf{J}_{21}^e \quad (4.86)$$

$$\mathbf{J}^{e-1} = \frac{1}{\det(\mathbf{J}^e)} \begin{bmatrix} \mathbf{J}_{22}^e & -\mathbf{J}_{12}^e \\ -\mathbf{J}_{21}^e & \mathbf{J}_{11}^e \end{bmatrix} \quad (4.87)$$

计算应变-位移矩阵：

$$\mathbf{B}^e = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 & \frac{\partial N_4}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial y} & 0 & \frac{\partial N_4}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial y} & \frac{\partial N_4}{\partial x} \end{bmatrix} \quad (4.88)$$

由此可以得到局部单元刚度阵：

$$\begin{aligned} \mathbf{K}^e &= \int_{\Omega^e} \mathbf{B}^{eT} \mathbf{D}^e \mathbf{B}^e d\Omega = \int_{-1}^1 \int_{-1}^1 \mathbf{B}^{eT} \mathbf{D}^e \mathbf{B}^e \det(\mathbf{J}^e) d\xi d\eta \\ &= \sum_{i=1}^2 \sum_{j=1}^2 W_i W_j |\mathbf{J}^e(\xi_i, \eta_j)| \mathbf{B}^{eT}(\xi_i, \eta_j) \mathbf{D}^e \mathbf{B}^e(\xi_i, \eta_j) \end{aligned} \quad (4.89)$$

依次对网格中每一个单元进行上述操作，最后将各单元刚度矩阵进行合并，组装得到整个计算域刚度矩阵。

3、载荷向量组装

单元上载荷向量的计算可由下式完成：

$$\mathbf{f}^e = \int_{\Omega^e} \mathbf{N}^{eT} \mathbf{f}^e d\Omega = \int_{-1}^1 \int_{-1}^1 \mathbf{N}^{eT} \mathbf{f}^e \det(\mathbf{J}^e) d\xi d\eta \quad (4.90)$$

需要注意的是这里 \mathbf{f} 为单元上所有外力，如果是边界单元该力主要由单元体积力与边界力组成。

4、边界条件处理及线性系统求解

完成矩阵与载荷向量组装后根据问题模型中给定边界条件依次对刚度矩阵及载荷向量进行处理，本问题模型中右边为第 2 类边界条件，相关处理可看到本章第 2 类边界条件处理方法。经过边界处理后即可求解线性系统得到有限元解。求解得到薄板内各节点位移后，由 $\boldsymbol{\varepsilon} = \mathbf{B}\mathbf{u}$ 得到对应各节点应变，再由 $\boldsymbol{\sigma} = \mathbf{D}\boldsymbol{\varepsilon}$ 得到各节点应力。

5、仿真结果

为了清晰看到受力情况下薄板各节点的位移变化情况，将所得仿真结果各节点位移扩大一定倍数。图 4-8a 和 4-8b 分别为平板 x 和 y 方向的位移。

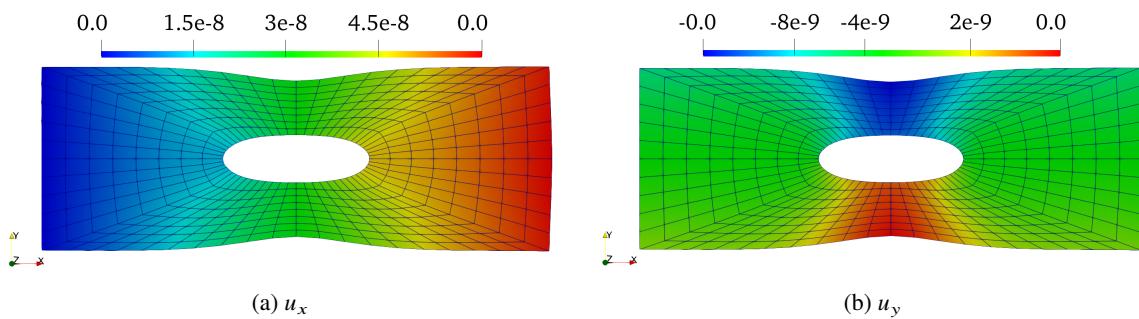


图 4-8 平板位移

图 4-9a, 4-9b 和 4-9c 分别为平板 x 、 y 方向应力和剪切应力。

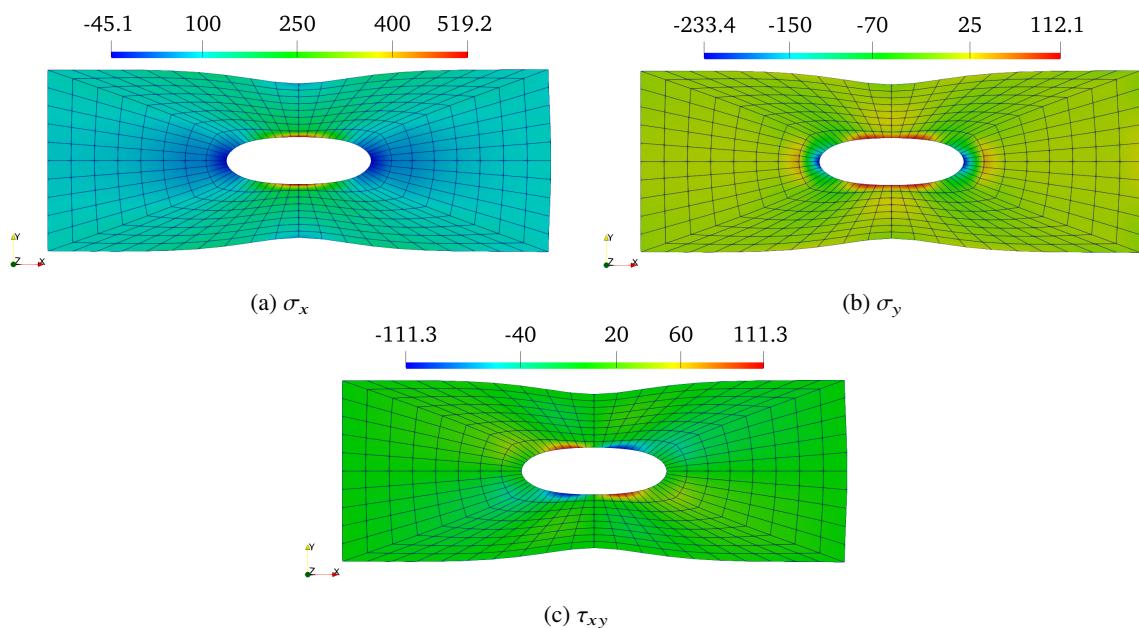


图 4-9 平板应力

习题

- 一个大小为 $1\text{m} \times 1\text{m}$ 的平板，平板左侧固定，右端施加大小为 $\mathbf{F} = 10\text{N}$ 的力，采用线性三角形单元，计算各节点位移、应变及应力。材料杨氏模量 $E = 100$ ，泊松比 $\nu = 0.3$ ，网格划分为 1×1 。

5

2D 稳态流体力学有限元方法

5.1 引言

流体力学基本控制方程为质量守恒方程、动量守恒方程和能量守恒方程^[24-29]，这些方程描述了流体的运动规律。流体力学的数学模型主要由 Euler 模型和 Navier-Stokes 模型，其中 Euler 模型忽略了流体的粘性，适用于高速流动，而 Navier-Stokes 模型考虑了流体的粘性，适用于低速流动的非牛顿流体。在本章中，主要介绍了 2D 稳态不可压缩流体（大多数液体和气体）的有限元方法，主要包括 Stokes 方程和 Navier-Stokes 方程的有限元方法。

5.2 稳态 Stokes 方程

5.2.1 问题模型

稳态情况下或粘度极高情况下，流体的运动可以由 Stokes 方程描述，其形式如下：

$$\begin{cases} -\nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, & \text{in } \Omega \\ \nabla \cdot \mathbf{u} = 0, & \text{in } \Omega \\ \mathbf{u} = \mathbf{g}, & \text{on } \partial\Omega \end{cases} \quad (5.1)$$

其中， $\mathbf{u}(x, y) = (u_x, u_y)^T$ 为速度场， p 为压力场， $\mathbf{f}(x, y) = (f_x, f_y)^T$ 为体积力， $\mathbf{g}(x, y) = (g_x, g_y)^T$ 为边界条件， $\mathbb{T}(\mathbf{u}, p)$ 为应力张量，定义如下：

$$\mathbb{T}(\mathbf{u}, p) = 2\nu\mathbb{D}(\mathbf{u}) - p\mathbb{I} \quad (5.2)$$

其中, ν 为粘度, $\mathbb{D}(\mathbf{u})$ 为应变张量, 定义如下:

$$\mathbb{D}(\mathbf{u}) = \frac{1}{2} (\nabla \mathbf{u} + \nabla \mathbf{u}^T) \quad (5.3)$$

将速度各分量及 ∇ 算符代入可以得到, 应变张量具体形式为:

$$\mathbb{D}(\mathbf{u}) = \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \\ \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) & \frac{\partial u_y}{\partial y} \end{bmatrix} \quad (5.4)$$

将应变张量代入应力张量中可以得到, 应力张量具体形式为:

$$\mathbb{T}(\mathbf{u}, p) = \begin{bmatrix} 2\nu \frac{\partial u_x}{\partial x} - p & \nu \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \\ \nu \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) & 2\nu \frac{\partial u_y}{\partial y} - p \end{bmatrix} \quad (5.5)$$

在该条件下, 由于方程中 p 仅出现在梯度项中, 如果 (\mathbf{u}, p) 是一个解, 那么 $(\mathbf{u}, p+c)$ 也是一个解, 其中 c 是一个常数。因此, 为得到确定的解, 需要为 p 附加条件, 通常有如下选择:

- (1) 在计算域 Ω 某点处固定压力 p 。
- (2) 在计算域部分边界处施加压力或者 Robin 边界条件。
- (3) 施加 $\int_{\Omega} p dx dy = 0$ 的边界条件。

5.2.2 弱格式

- (1) 方程两端同乘以任意的测试函数 (Test Function) $\mathbf{v}(x, y) = (v_x, v_y)^T$ 做内积 (Inner Product), 并在整个求解域 Ω 上积分**

$$\begin{aligned} -\nabla \cdot \mathbb{T}(\mathbf{u}, p) &= \mathbf{f}, \quad \text{in } \Omega \\ -(\nabla \cdot \mathbb{T}(\mathbf{u}, p)) \cdot \mathbf{v} &= \mathbf{f} \cdot \mathbf{v}, \quad \text{in } \Omega \\ -\int_{\Omega} (\nabla \cdot \mathbb{T}(\mathbf{u}, p)) \cdot \mathbf{v} dx dy &= \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \end{aligned} \quad (5.6)$$

- (2) 在无散度项乘以函数 $q(x, y)$, 并在计算域上积分**

$$\nabla \cdot \mathbf{u} = 0, \quad \text{in } \Omega$$

$$\begin{aligned} (\nabla \cdot \mathbf{u}) q &= 0, \quad \text{in } \Omega \\ \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy &= 0 \end{aligned} \quad (5.7)$$

在上述方程中， $\mathbf{u}(x, y)$ 、 $p(x, y)$ 为试探函数 (Trial Function)， $\mathbf{v}(x, y)$ 、 $q(x, y)$ 为测试函数 (Test Function)。

(3) 根据多元分部积分法 (散度定理) 对方程等号左边项进行变换

$$\int_{\Omega} (\nabla \cdot \mathbb{T}) \cdot \mathbf{v} dx dy = \int_{\partial\Omega} (\mathbb{T}\mathbf{n}) \cdot \mathbf{v} ds - \int_{\Omega} \mathbb{T} : \nabla \mathbf{v} dx dy \quad (5.8)$$

其中， $\mathbf{n} = (n_1, n_2)^T$ 为计算域边界 $\partial\Omega$ 的单位外法向量 (Unit Outer Normal Vector)，由此可以得到：

$$\int_{\Omega} \mathbb{T}(\mathbf{u}, p) : \nabla \mathbf{v} dx dy - \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \quad (5.9)$$

对于两个张量，其内积的定义为：

$$A : B = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} : \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = a_{12}b_{11} + a_{12}b_{12} + a_{21}b_{21} + a_{22}b_{22} \quad (5.10)$$

使用上述张量内积的定义，可以得到：

$$\mathbb{T}(\mathbf{u}, p) : \nabla \mathbf{v} = (2\nu \mathbb{D}(\mathbf{u}) - p \mathbb{I}) : \nabla \mathbf{v} = 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) - p (\nabla \cdot \mathbf{v}) \quad (5.11)$$

由此可以进一步得到：

$$\begin{aligned} \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy - \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds &= \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \\ - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy &= 0 \end{aligned} \quad (5.12)$$

【说明】 这里在式 (5.7) 中乘以 -1 用以保证矩阵对称性。

待求函数在计算域边界 $\partial\Omega$ 处的值由 $\mathbf{u} = \mathbf{g}$ 给定，因此可以选择合适的测试函数 $\mathbf{v}(x, y)$ 使其在边界 $\partial\Omega$ 处满足 $\mathbf{v}(x, y) = 0$ ，此时方程 (5.9) 等号左边第二项为零，可以得到：

$$\begin{aligned} \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy &= \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \\ - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy &= 0 \end{aligned} \quad (5.13)$$

方程 (5.13) 即为待求问题的弱格式 (Weak Formulation)。该方程的求解就是找到 $\mathbf{u} \in H^1(\Omega) \times H^1(\Omega)$ 和 $p \in L^2(\Omega)$ 对任意 $\mathbf{v} \in H_0^1(\Omega) \times H_0^1(\Omega)$ 和 $q \in L^2(\Omega)$ 都成立。

5.2.3 方程离散

令 $a(\mathbf{u}, \mathbf{v}) = \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy$, $b(\mathbf{u}, q) = - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy$, $(\mathbf{f}, \mathbf{v}) = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy$, 待求问题为找到 $\mathbf{u} \in H^1(\Omega) \times H^1(\Omega)$ 和 $p \in L^2(\Omega)$ 对任意 $\mathbf{v} \in H_0^1(\Omega) \times H_0^1(\Omega)$ 和 $q \in L^2(\Omega)$ 都成立。

$$\begin{aligned} a(\mathbf{u}, \mathbf{v}) + b(\mathbf{v}, p) &= (\mathbf{f}, \mathbf{v}) \\ b(\mathbf{u}, q) &= 0 \end{aligned} \quad (5.14)$$

根据张量双点积的定义可以得到：

$$\begin{aligned} \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) &= \left[\begin{array}{cc} \frac{\partial u_x}{\partial x} & \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \\ \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) & \frac{\partial u_y}{\partial y} \end{array} \right] : \left[\begin{array}{cc} \frac{\partial v_x}{\partial x} & \frac{1}{2} \left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x} \right) \\ \frac{1}{2} \left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x} \right) & \frac{\partial v_y}{\partial y} \end{array} \right] \\ &= \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x} \right) + \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} \end{aligned} \quad (5.15)$$

进一步对上式进行简化，可以得到：

$$\begin{aligned} \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) &= \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + \frac{1}{2} \left(\frac{\partial u_x}{\partial y} \frac{\partial v_x}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_y}{\partial x} + \frac{\partial u_y}{\partial x} \frac{\partial v_x}{\partial y} + \frac{\partial u_y}{\partial x} \frac{\partial v_y}{\partial x} \right) + \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} \end{aligned} \quad (5.16)$$

所以，

$$\begin{aligned} &\int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy \\ &= \int_{\Omega} \nu \left(2 \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + 2 \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_x}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_y}{\partial x} + \frac{\partial u_y}{\partial x} \frac{\partial v_x}{\partial y} + \frac{\partial u_y}{\partial x} \frac{\partial v_y}{\partial x} \right) dx dy \end{aligned} \quad (5.17)$$

同理可以得到：

$$\int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy = \int_{\Omega} p \left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right) dx dy \quad (5.18)$$

$$\int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy = \int_{\Omega} (f_x v_x + f_y v_y) dx dy \quad (5.19)$$

$$\int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = \int_{\Omega} \left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} \right) q dx dy \quad (5.20)$$

所以 Stokes 方程弱格式的标量形式为找到 $u_x \in H^1(\Omega)$ 、 $u_y \in H^1(\Omega)$ 和 $p \in L^2(\Omega)$ ，使得对任意 $v_x \in H_0^1(\Omega)$ 、 $v_y \in H_0^1(\Omega)$ 和 $q \in L^2(\Omega)$ 都成立：

$$\begin{aligned} & \int_{\Omega} v \left(2 \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + 2 \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_x}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_y}{\partial x} + \frac{\partial u_y}{\partial x} \frac{\partial v_x}{\partial y} + \frac{\partial u_y}{\partial x} \frac{\partial v_y}{\partial x} \right) dx dy \\ & - \int_{\Omega} p \left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right) dx dy = \int_{\Omega} (f_x v_x + f_y v_y) dx dy \\ & - \int_{\Omega} \left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} \right) q dx dy = 0 \end{aligned} \quad (5.21)$$

所以在有限元空间上，对任意 $\mathbf{v}_h \in U_h \times U_h$ 和 $q_h \in W_h$ 有：

$$\begin{aligned} a(\mathbf{u}_h, \mathbf{v}_h) + b(\mathbf{v}_h, p_h) &= (\mathbf{f}, \mathbf{v}_h) \\ b(\mathbf{u}_h, q_h) &= 0 \end{aligned} \quad (5.22)$$

在采用有限元方法对速度场与压力场进行求解时，基函数不能任意选取，必须满足一些条件，因为动量方程与连续性方程的耦合会产生稳定性问题，即使程序运行无误也将产生无效结果。该问题与“inf-sup”或“Ladyzhenskaya-Babuska-Brezzi, LBB”有关。

$$\inf_{0 \neq q_h \in W_h} \sup_{0 \neq \mathbf{u}_h \in U_h \times U_h} \frac{b(\mathbf{u}_h, q_h)}{\|\nabla \mathbf{u}_h\|_0 \|q_h\|_0} > \beta \quad (5.23)$$

其中 $\beta > 0$ 为一无关单元大小的常数。

为避免出现稳定性问题，在求解时通常采用“Taylor-Hood”或“Hilbert-Schmidt”基函数对分别处理速度与压力，在“Taylor-Hood”基函数对中，速度基函数阶数通常较压力基函数阶数更高，因此，这里选择速度对应有限单元为 2 次元，其对应基函数空间为 $V_h = \{v \in [C^0(\Omega)]^d : v|_k \in [P^2(K)]^d\}$ ，压力对应有限元基函数为线性元，其基函数空间为 $Q_h = \{q \in [C^0(\Omega)]^1 : q|_k \in P^1(K)^1\}$ 。

5.2.4 线性系统构建

(1) 刚度矩阵

将待求函数中未知变量用有限元基函数展开，得到：

$$u_{xh} = \sum_{j=1}^{N_b} u_{xj} \phi_j, \quad u_{yh} = \sum_{j=1}^{N_b} u_{yj} \phi_j, \quad p_h = \sum_{j=1}^{N_bp} p_j \psi_j \quad (5.24)$$

选取合适的基函数，得到关于 u_{xj} 、 u_{yj} 和 p_j 的线性系统，然后求解该线性系统即可得到有限元解： $\mathbf{u}_h = (u_{xh}, u_{yh})^T$ 和 p_h 。

对于第一个方程，将 $\mathbf{v}_h = (\phi_i, 0)^T (i = 1, \dots, N_b)$ 和 $\mathbf{v}_h = (0, \phi_i)^T (i = 1, \dots, N_b)^T$ 代入弱格式，即对第一组测试函数取 $v_{xh} = \phi_i (i = 1, \dots, N_b)$ 和 $v_{yh} = 0$ ，对第二组测试函数取 $v_{xh} = 0$ 和 $v_{yh} = \phi_i (i = 1, \dots, N_b)$ 。

对于第二个方程，取 $q_h = \psi_i, 0 (i = 1, \dots, N_{bp})$ 。

► 令 $\mathbf{v}_h = (\phi_i, 0)^T$ ，即 $v_{xh} = \phi_i$ 和 $v_{yh} = 0 (i = 1, \dots, N_b)$ ，得到：

$$\begin{aligned} & 2 \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{xj} \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{xj} \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial y} dx dy \\ & + \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{yj} \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial y} dx dy - \int_{\Omega} \left(\sum_{j=1}^{N_{bp}} p_j \psi_j \right) \frac{\partial \phi_i}{\partial x} dx dy \\ & = \int_{\Omega} f_x \phi_i dx dy \end{aligned} \quad (5.25)$$

► 令 $\mathbf{v}_h = (0, \phi_i)^T$ ，即 $v_{xh} = 0$ 和 $v_{yh} = \phi_i (i = 1, \dots, N_b)$ ，得到：

$$\begin{aligned} & 2 \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{yj} \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial y} dx dy + \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{yj} \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial x} dx dy \\ & + \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{xj} \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial x} dx dy - \int_{\Omega} \left(\sum_{j=1}^{N_{bp}} p_j \psi_j \right) \frac{\partial \phi_i}{\partial y} dx dy \\ & = \int_{\Omega} f_y \phi_i dx dy \end{aligned} \quad (5.26)$$

► 令 $q_h = \psi_i (i = 1, \dots, N_{bp})$ ，得到：

$$-\int_{\Omega} \left(\sum_{j=1}^{N_b} u_{xj} \frac{\partial \phi_j}{\partial x} \right) \psi_i dx dy - \int_{\Omega} \left(\sum_{j=1}^{N_b} u_{yj} \frac{\partial \phi_j}{\partial y} \right) \psi_i dx dy = 0 \quad (5.27)$$

将上述三式进行整理后可以得到：

$$\begin{aligned} & \sum_{j=1}^{N_b} u_{xj} \left(2 \int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy \right) \\ & + \sum_{j=1}^{N_b} u_{yj} \left(\int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial y} dx dy \right) + \sum_{j=1}^{N_{bp}} p_j \left(- \int_{\Omega} \psi_j \frac{\partial \phi_i}{\partial x} dx dy \right) = \int_{\Omega} f_x \phi_i dx dy \end{aligned} \quad (5.28)$$

$$\begin{aligned} & \sum_{j=1}^{N_b} u_{xj} \left(\int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial x} dx dy \right) + \sum_{j=1}^{N_b} u_{yj} \left(2 \int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy + \int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy \right) \\ & + \sum_{j=1}^{N_{bp}} p_j \left(- \int_{\Omega} \psi_j \frac{\partial \phi_i}{\partial y} dx dy \right) = \int_{\Omega} f_y \phi_i dx dy \end{aligned} \quad (5.29)$$

$$\sum_{j=1}^{N_b} u_{xj} \left(- \int_{\Omega} \frac{\partial \phi_j}{\partial x} \psi_i dx dy \right) + \sum_{j=1}^{N_b} u_{yj} \left(- \int_{\Omega} \frac{\partial \phi_j}{\partial y} \psi_i dx dy \right) + \sum_{j=1}^{N_{bp}} p_j * 0 = 0 \quad (5.30)$$

为了方便描述，定义如下矩阵组：

$$\begin{aligned} \mathbf{K}_1 &= \left[\int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b}, & \mathbf{K}_2 &= \left[\int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b} \\ \mathbf{K}_3 &= \left[\int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b}, & \mathbf{K}_4 &= \left[\int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b} \\ \mathbf{K}_5 &= \left[\int_{\Omega} -\psi_j \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b, N_{bp}}, & \mathbf{K}_6 &= \left[\int_{\Omega} -\psi_j \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b, N_{bp}} \\ \mathbf{K}_7 &= \left[\int_{\Omega} -\frac{\partial \phi_j}{\partial x} \psi_i dx dy \right]_{i,j=1}^{N_{bp}, N_b}, & \mathbf{K}_8 &= \left[\int_{\Omega} -\frac{\partial \phi_j}{\partial y} \psi_i dx dy \right]_{i,j=1}^{N_{bp}, N_b} \end{aligned} \quad (5.31)$$

定义 0 矩阵为 $\mathbb{O}_1 = [0]_{i,j=1}^{N_{bp}, N_{bp}}$ ，该矩阵大小为 $N_{bp} \times N_{bp}$ ，由此可以得到线性系统矩阵 A ：

$$\mathbf{K} = \begin{bmatrix} 2\mathbf{K}_1 + \mathbf{K}_2 & \mathbf{K}_3 & \mathbf{K}_5 \\ \mathbf{K}_4 & 2\mathbf{K}_2 + \mathbf{K}_1 & \mathbf{K}_6 \\ \mathbf{K}_7 & \mathbf{K}_8 & \mathbb{O}_1 \end{bmatrix} \quad (5.32)$$

其中， $\mathbf{K}_4 = \mathbf{K}_3^T$ ， $\mathbf{K}_7 = \mathbf{K}_5^T$ ， $\mathbf{K}_8 = \mathbf{K}_6^T$ ，所以该矩阵为一对称矩阵。

矩阵组中各个矩阵可通过算法 6 分别得到，详细过程如下：

- 令算法 6 中 $c = \nu$, $r = 1$, $s = 0$, $p = 1$, $q = 0$, 得到矩阵 \mathbf{K}_1 ;
- 令算法 6 中 $c = \nu$, $r = 0$, $s = 1$, $p = 0$, $q = 1$, 得到矩阵 \mathbf{K}_2 ;
- 令算法 6 中 $c = \nu$, $r = 1$, $s = 0$, $p = 0$, $q = 1$, 得到矩阵 \mathbf{K}_3 ;
- 令算法 6 中 $c = -1$, $r = 0$, $s = 0$, $p = 1$, $q = 0$, 得到矩阵 \mathbf{K}_5 ;
- 令算法 6 中 $c = -1$, $r = 0$, $s = 0$, $p = 0$, $q = 1$, 得到矩阵 \mathbf{K}_6 ;

根据线性系统对上述矩阵组进行整理可以得到：

$$\mathbf{K} = \begin{bmatrix} 2\mathbf{K}_1 + \mathbf{K}_2 & \mathbf{K}_3 & \mathbf{K}_5 \\ \mathbf{K}_3^T & 2\mathbf{K}_2 + \mathbf{K}_1 & \mathbf{K}_6 \\ \mathbf{K}_5^T & \mathbf{K}_6^T & \mathbb{O}_1 \end{bmatrix} \quad (5.33)$$

(2) 载荷向量

定义载荷向量如下：

$$\mathbf{F} = \begin{bmatrix} \mathbf{F}_x \\ \mathbf{F}_y \\ \mathbf{0} \end{bmatrix} \longrightarrow \mathbf{F}_x = \left[\int_{\Omega} f_x \phi_i dx dy \right]_{i=1}^{N_b}, \quad \mathbf{F}_y = \left[\int_{\Omega} f_y \phi_i dx dy \right]_{i=1}^{N_b} \quad (5.34)$$

这里 $\mathbf{0}$ 矩阵大小为 $N_{bp} \times 1$, 由此可以得到载荷向量阵 b , 载荷向量可以采用算法 7 得到, 详细过程如下:

- ▶ 令算法 7 中 $f = f_x$, $p = q = 0$, 得到载荷向量阵 \mathbf{F}_x ;
- ▶ 令算法 7 中 $f = f_y$, $p = q = 0$, 得到载荷向量阵 \mathbf{F}_y ;

(3) 未知向量

定义待求向量如下:

$$\mathbf{u} = \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \end{bmatrix} \longrightarrow \mathbf{u}_1 = [u_{1j}]_{i=1}^{N_b}, \quad \mathbf{u}_2 = [u_{2j}]_{i=1}^{N_b}, \quad \mathbf{u}_3 = [p_j]_{i=1}^{N_{bp}} \quad (5.35)$$

(4) 线性系统组成

根据得到的刚度矩阵、载荷向量和未知向量, 可以得到线性系统:

$$\mathbf{Ku} = \mathbf{F} \quad (5.36)$$

得到线性系统后根据所给问题模型的边界条件进行处理, 得到最终需要求解的线性系统, 然后求解该线性系统即可得到有限元解。

5.2.5 不同边界条件处理

(1) 第 1 类边界条件

在 Stokes 方程中, 第一类边界条件为速度边界: $\mathbf{u} = \mathbf{g}$, 也称为无滑移边界 (No-slip), 常用固体法向。由于 Stokes 问题中速度为一矢量, 所以在进行边界处理时需要同时对矢量分量边界进行处理, 这里处理方式仍然采用置 1 法, 详细算法流程如算法 11 所示。需要注意的是, 与线弹性问题中第一类边界条件处理方式有所区别, 如前所述, 压力在方程中仅有梯度项, 因此需要附加压力条件以获得唯一的解, 为操作方便, 这里在计算域中给定某点参考压力即可。

(2) 第 2 类边界条件

在 Stokes 方程中，第 2 类边界条件为压力边界，这类边界常作为计算域出口边界条件，且其出口外计算域较大，使得该处压力很小，此时控制方程为：

$$\begin{cases} -\nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, & \text{in } \Omega \\ \nabla \cdot \mathbf{u} = 0, & \text{in } \Omega \\ \mathbb{T}(\mathbf{u}, p) \mathbf{n} = \mathbf{p}, & \text{on } \partial\Omega \end{cases} \quad (5.37)$$

其中， $\mathbf{u}(x, y) = (u_x, u_y)^T$ 为速度场， p 为压力场， $\mathbf{f}(x, y) = (f_x, f_y)^T$ 为体积力， $\mathbf{n}(x, y) = (n_x, n_y)^T$ 为边界 $\partial\Omega$ 的法向量， $\mathbb{T}(\mathbf{u}, p)$ 为应力张量， $\mathbf{p}(x, y) = (p_x, p_y)^T$ 。

由前述推导可知：

$$\begin{aligned} \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy - \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds &= \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \\ - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy &= 0 \end{aligned} \quad (5.38)$$

将 Neumann 边界条件代入方程中，得到：

$$\begin{aligned} \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy - \int_{\partial\Omega} \mathbf{p} \cdot \mathbf{v} ds &= \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \\ - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy &= 0 \end{aligned} \quad (5.39)$$

可以看到所得方程的解不唯一！如果 $\mathbf{u}(x, y) = (u_x, u_y)^T$ 为该方程的解，则 $\mathbf{u} + \mathbf{c}$ 也为方程的解，其中 \mathbf{c} 为常矢量。

所以通常对于上述问题常为 Dirichlet + Neumann 的混合边界条件，由此控制方程变换为：

$$\begin{cases} -\nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, & \text{in } \Omega \\ \nabla \cdot \mathbf{u} = 0, & \text{in } \Omega \\ \mathbb{T}(\mathbf{u}, p) \mathbf{n} = \mathbf{p}, & \text{on } \Gamma_S \subset \partial\Omega \\ \mathbf{u} = \mathbf{g}, & \text{on } \Gamma_D = \partial\Omega / \Gamma_S \end{cases} \quad (5.40)$$

由于边界 $\Gamma_D = \partial\Omega / \Gamma_S$ 上的解由式 $\mathbf{u} = \mathbf{g}$ 确定，所以，可以选择合适的测试函数使其在边界上满足 $\mathbf{v} = 0$ 。由边界条件可以得到：

$$\int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Gamma_S} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds + \int_{\partial\Omega / \Gamma_S} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds \quad (5.41)$$

因此，方程的弱格式就是找到 $\mathbf{u} \in H^1(\Omega) \times H^1(\Omega)$ 和 $p \in L^2(\Omega)$ 使其对任何 $\mathbf{v} \in H_{0D}^1(\Omega) \times H_{0D}^1(\Omega)$ 和 $q \in L^2(\Omega)$ 都有：

$$\int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy + \int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds \\ - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \quad (5.42)$$

其中， $\int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds = \int_{\Gamma_S} p_x v_x ds + \int_{\Gamma_S} p_y v_y ds$ ， $H_{0D}^1(\Omega) = \{v \in H^1(\Omega) : v = 0 \text{ on } \Gamma_D\}$ 。

所以，上式即为包含第二类边界条件时待求方程的弱格式，仅需要在 Neumann 边界对应的载荷向量处增加 $\int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds$ 即可。

(3) 第3类边界条件

第3类边界条件与第2类边界条件应用场景类似，常作为流体出口边界，区别在于第3类边界更适合作为长管道计算域出口边界。

$$\begin{cases} -\nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, & \text{in } \Omega \\ \nabla \cdot \mathbf{u} = 0, & \text{in } \Omega \\ \mathbb{T}(\mathbf{u}, p) \mathbf{n} + r\mathbf{u} = \mathbf{q}, & \text{on } \Gamma_R \subseteq \partial\Omega \\ \mathbf{u} = \mathbf{g}, & \text{on } \Gamma_D = \partial\Omega / \Gamma_R \end{cases} \quad (5.43)$$

由前述推导可知：

$$\int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy - \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \\ - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \quad (5.44)$$

由于边界 $\Gamma_D = \partial\Omega / \Gamma_R$ 上的解由式 $\mathbf{u} = \mathbf{g}$ 确定，所以，可以选择合适的测试函数使其在边界上满足 $\mathbf{v} = 0$ 。由边界条件可以得到：

$$\int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Gamma_S} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds + \int_{\partial\Omega / \Gamma_R} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds \\ = \int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds - \int_{\Gamma_R} r\mathbf{u} \cdot \mathbf{v} ds \quad (5.45)$$

因此，方程的弱格式就是找到 $\mathbf{u} \in H^1(\Omega) \times H^1(\Omega)$ 和 $p \in L^2(\Omega)$ 使其对任何 $\mathbf{v} \in H_{0D}^1(\Omega) \times H_{0D}^1(\Omega)$ 和 $q \in L^2(\Omega)$ 都有：

$$\int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy + \int_{\Gamma_R} r\mathbf{u} \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy + \int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds$$

$$-\int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \quad (5.46)$$

其中, $\int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds = \int_{\Gamma_R} q_x v_x ds + \int_{\Gamma_R} q_y v_y ds$, $\int_{\Gamma_R} r \mathbf{u} \cdot \mathbf{v} ds = \int_{\Gamma_R} r u_x v_x ds + \int_{\Gamma_R} r u_y v_y ds$, $H_{0D}^1(\Omega) = \{v \in H^1(\Omega) : v = 0 \text{ on } \Gamma_D\}$ 。

所以, 上式即为包含第 3 类边界条件时待求方程的弱格式, 需要在 Robin 边界对应刚度矩阵处增加 $\int_{\Gamma_R} r \mathbf{u} \cdot \mathbf{v} ds$, 在载荷向量处增加 $\int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds$ 。

(4) 混合边界条件

考虑如下包含混合边界的稳态 Stokes 方程:

$$\begin{cases} -\nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, & \text{in } \Omega \\ \nabla \cdot \mathbf{u} = 0, & \text{in } \Omega \\ \mathbb{T}(\mathbf{u}, p) \mathbf{n} = \mathbf{p}, & \text{on } \Gamma_S \subset \partial\Omega \\ \mathbb{T}(\mathbf{u}, p) \mathbf{n} + r \mathbf{u} = \mathbf{q}, & \text{on } \Gamma_R \subseteq \partial\Omega \\ \mathbf{u} = \mathbf{g}, & \text{on } \Gamma_D = \partial\Omega / (\Gamma_S \cup \Gamma_R) \end{cases} \quad (5.47)$$

由前述推导可知:

$$\begin{aligned} \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy - \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds &= \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \\ - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy &= 0 \end{aligned} \quad (5.48)$$

由于边界 $\Gamma_D = \partial\Omega / (\Gamma_S \cup \Gamma_R)$ 上的解由式 $\mathbf{u} = \mathbf{g}$ 确定, 所以, 可以选择合适的测试函数使其在该边界上满足 $\mathbf{v} = 0$ 。由边界条件可以得到:

$$\begin{aligned} \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds &= \int_{\Gamma_S} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds + \int_{\Gamma_R} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds \\ + \int_{\partial\Omega / (\Gamma_S \cup \Gamma_R)} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds &= \int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds + \int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds - \int_{\Gamma_R} r \mathbf{u} \cdot \mathbf{v} ds \end{aligned} \quad (5.49)$$

因此, 方程的弱格式就是找到 $\mathbf{u} \in H^1(\Omega) \times H^1(\Omega)$ 和 $p \in L^2(\Omega)$ 使其对任何 $\mathbf{v} \in H_{0D}^1(\Omega) \times H_{0D}^1(\Omega)$ 和 $q \in L^2(\Omega)$ 都有:

$$\begin{aligned} \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy + \int_{\Gamma_R} r \mathbf{u} \cdot \mathbf{v} ds \\ = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy + \int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds + \int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds \\ - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \end{aligned} \quad (5.50)$$

其中, $\int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds = \int_{\Gamma_R} q_x v_x ds + \int_{\Gamma_R} q_y v_y ds$, $\int_{\Gamma_R} r \mathbf{u} \cdot \mathbf{v} ds = \int_{\Gamma_R} r u_x v_x ds + \int_{\Gamma_R} r u_y v_y ds$, $H_{0D}^1(\Omega) = \{v \in H^1(\Omega) : v = 0 \text{ on } \Gamma_D\}$ 。

在求解该系统时, 依次按照 Robin 型、Neumann 型、Dirichlet 型边界条件进行处理, 始终保证 Dirichlet 边界发挥作用, 具体而言为: 在 Robin 型对应边界刚度矩阵增加 $\int_{\Gamma_R} r \mathbf{u} \cdot \mathbf{v} ds$, 载荷向量增加 $\int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds$; 在 Neumann 型对应边界载荷向量增加 $\int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds$; 对 Dirichlet 类型边界对应刚度矩阵采用“置 1”法, 载荷向量为给定边界条件值。

5.2.6 误差计算

Stokes 有限元解的正确性通过与解析解 (如果存在或者较为方便得到的话) 的误差进行衡量, 这里通过: 1、最大节点值误差; 2、无穷范数误差 L^∞ ; 3、 L^2 范数误差; 4、 H^1 范数误差进行恒量。与前述问题模型误差不同的是, Stokes 方程需要对速度场、压力场误差分别进行计算, 且由于采用“Taylor-Hood”元, 速度场和压力场的误差对应计算方法及精度有所区别。

(1) 最大节点误差

分别计算速度 x 、 y 方向上最大节点误差, 随后取二者中最大值作为计算域中速度的最大节点值误差, 同时计算压力的最大节点值误差。

$$\begin{aligned} maxNodeErrorU &= \max(maxNodeErrorU_x, maxNodeErrorU_y) \\ maxNodeErrorP &= \max(resP(i) - exactP(i)) \end{aligned} \quad (5.51)$$

(2) 无穷范数误差 L^∞

$$\begin{aligned} \|u_x - u_{xh}\|_\infty &= \sup_{\Omega} |u_x - u_{xh}| \\ \|u_y - u_{yh}\|_\infty &= \sup_{\Omega} |u_y - u_{yh}| \\ \rightarrow \|u - u_h\|_\infty &= \max(\|u_x - u_{xh}\|_\infty, \|u_y - u_{yh}\|_\infty) \\ \rightarrow \|p - p_h\|_\infty &= \sup_{\Omega} |p - p_h| \end{aligned} \quad (5.52)$$

(3) L^2 范数误差

$$\|u_x - u_{xh}\|_0 = \sqrt{\int_{\Omega} (u_x - u_{xh})^2 dx dy}$$

$$\begin{aligned}
\|u_y - u_{yh}\|_0 &= \sqrt{\int_{\Omega} (u_y - u_{yh})^2 dx dy} \\
\rightarrow \|\mathbf{u} - \mathbf{u}_h\|_0 &= \sqrt{\|u_x - u_{xh}\|_0^2 + \|u_y - u_{yh}\|_0^2} \\
\rightarrow \|p - p_h\|_0 &= \sqrt{\int_{\Omega} (p - p_h)^2 dx dy}
\end{aligned} \tag{5.53}$$

(4) H^1 范数误差

$$\begin{aligned}
\|u_x - u_{xh}\|_1 &= \sqrt{\int_{\Omega} \left(\frac{\partial (u_x - u_{xh})}{\partial x} \right)^2 + \left(\frac{\partial (u_x - u_{xh})}{\partial y} \right)^2 dx dy} \\
\|u_y - u_{yh}\|_1 &= \sqrt{\int_{\Omega} \left(\frac{\partial (u_y - u_{yh})}{\partial x} \right)^2 + \left(\frac{\partial (u_y - u_{yh})}{\partial y} \right)^2 dx dy} \\
\rightarrow \|\mathbf{u} - \mathbf{u}_h\|_1 &= \sqrt{\|u_x - u_{xh}\|_1^2 + \|u_y - u_{yh}\|_1^2} \\
\rightarrow \|p - p_h\|_1 &= \sqrt{\int_{\Omega} \left(\frac{\partial (p - p_h)}{\partial x} \right)^2 + \left(\frac{\partial (p - p_h)}{\partial y} \right)^2 dx dy}
\end{aligned} \tag{5.54}$$

5.3 稳态 Navier-Stokes 方程

5.3.1 问题模型

稳态或粘度较高情况下，流体的运动可由 Navier-Stokes 方程（NS 方程）描述，其形式如下：

$$\begin{cases} (\mathbf{u} \cdot \nabla) \mathbf{u} - \nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, & \text{in } \Omega \\ \nabla \cdot \mathbf{u} = 0, & \text{in } \Omega \\ \mathbf{u} = \mathbf{g}, & \text{on } \partial\Omega \end{cases} \tag{5.55}$$

其中， $\mathbf{u}(x, y) = (u_x, u_y)^T$ 为速度场， p 为压力场， $\mathbf{f}(x, y) = (f_x, f_y)^T$ 为体积力， $\mathbf{g}(x, y) = (g_x, g_y)^T$ 为边界条件，与稳态 Stokes 方程相比，NS 方程考虑了非线性项 $(\mathbf{u} \cdot \nabla) \mathbf{u}$ ，该项为对流项，其定义如下：

$$(\mathbf{u} \cdot \nabla) \mathbf{u} = \begin{bmatrix} u_x \frac{\partial u_x}{\partial x} + u_y \frac{\partial u_x}{\partial y} \\ u_x \frac{\partial u_y}{\partial x} + u_y \frac{\partial u_y}{\partial y} \end{bmatrix} \tag{5.56}$$

$\mathbb{T}(\mathbf{u}, p)$ 为应力张量, 定义如下:

$$\mathbb{T}(\mathbf{u}, p) = 2\nu\mathbb{D}(\mathbf{u}) - p\mathbb{I} \quad (5.57)$$

其中, ν 为粘度, $\mathbb{D}(\mathbf{u})$ 为应变张量, 定义如下:

$$\mathbb{D}(\mathbf{u}) = \frac{1}{2} (\nabla \mathbf{u} + \nabla \mathbf{u}^T) \quad (5.58)$$

将速度各分量及 ∇ 算符代入可以得到, 应变张量具体形式为:

$$\mathbb{D}(\mathbf{u}) = \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \\ \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) & \frac{\partial u_y}{\partial y} \end{bmatrix} \quad (5.59)$$

将应变张量代入应力张量中可以得到, 应力张量具体形式为:

$$\mathbb{T}(\mathbf{u}, p) = \begin{bmatrix} 2\nu \frac{\partial u_x}{\partial x} - p & \nu \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \\ \nu \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) & 2\nu \frac{\partial u_y}{\partial y} - p \end{bmatrix} \quad (5.60)$$

在该条件下, 由于方程中 p 仅出现在梯度项中, 如果 (\mathbf{u}, p) 是一个解, 那么 $(\mathbf{u}, p+c)$ 也是一个解, 其中 c 是一个常数。因此, 为得到确定的解, 需要为 p 附加条件, 通常有如下选择:

- (1) 在计算域 Ω 某点处固定压力 p 。
- (2) 在计算域部分边界处施加压力或者 Robin 边界条件。
- (3) 施加 $\int_{\Omega} p dx dy = 0$ 的边界条件。

5.3.2 弱格式

(1) 方程两端同乘以任意测试函数 (Test Function) $\mathbf{v}(x, y) = (v_x, v_y)^T$ 做内积 (Inner Product), 并在整个求解域 Ω 上积分

$$\begin{aligned} & (\mathbf{u} \cdot \nabla) \mathbf{u} - \nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, \quad \text{in } \Omega \\ & \rightarrow (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} - (\nabla \cdot \mathbb{T}(\mathbf{u}, p)) \cdot \mathbf{v} = \mathbf{f} \cdot \mathbf{v}, \quad \text{in } \Omega \\ & \rightarrow \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy - \int_{\Omega} (\nabla \cdot \mathbb{T}(\mathbf{u}, p)) \cdot \mathbf{v} dx dy = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \end{aligned} \quad (5.61)$$

(2) 在无散度项乘以函数 $q(x, y)$

$$\begin{aligned} \nabla \cdot \mathbf{u} &= 0, \quad \text{in } \Omega \\ \rightarrow (\nabla \cdot \mathbf{u}) q &= 0, \quad \text{in } \Omega \\ \rightarrow \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy &= 0 \end{aligned} \quad (5.62)$$

在上述方程中， $\mathbf{u}(x, y)$ 、 $p(x, y)$ 为试探函数 (Trial Function)， $\mathbf{v}(x, y)$ 、 $q(x, y)$ 为测试函数 (Test Function)。

(3) 根据多元分部积分法 (散度定理) 对方程等号左边项进行变换

$$\int_{\Omega} (\nabla \cdot \mathbb{T}) \cdot \mathbf{v} dx dy = \int_{\partial\Omega} (\mathbb{T} \mathbf{n}) \cdot \mathbf{v} ds - \int_{\Omega} \mathbb{T} : \nabla \mathbf{v} dx dy \quad (5.63)$$

其中， $\mathbf{n} = (n_1, n_2)^T$ 为计算域边界 $\partial\Omega$ 的单位外法向量 (Unit Outer Normal Vector)，由此可以得到：

$$\int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy + \int_{\Omega} \mathbb{T}(\mathbf{u}, p) : \nabla \mathbf{v} dx dy - \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \quad (5.64)$$

对于两个张量，其内积的定义为：

$$A : B = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} : \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = a_{12}b_{11} + a_{12}b_{12} + a_{21}b_{21} + a_{22}b_{22} \quad (5.65)$$

使用上述张量内积的定义，可以得到：

$$\mathbb{T}(\mathbf{u}, p) : \nabla \mathbf{v} = (2\nu \mathbb{D}(\mathbf{u}) - p \mathbb{I}) : \nabla \mathbf{v} = 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) - p (\nabla \cdot \mathbf{v}) \quad (5.66)$$

由此可以进一步得到：

$$\begin{aligned} \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy \\ - \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \\ - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \end{aligned} \quad (5.67)$$

【说明】：这里在式 (5.62) 中乘以 -1 用以保证矩阵对称性。

待求函数在计算域边界 $\partial\Omega$ 处的值由 $\mathbf{u} = \mathbf{g}$ 给定，因此可以选择合适的测试函数 $\mathbf{v}(x, y)$ 使其在边界 $\partial\Omega$ 处满足 $\mathbf{v}(x, y) = 0$ ，可以得到：

$$\begin{aligned} \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy &= \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \\ - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy &= 0 \end{aligned} \quad (5.68)$$

方程 (5.68) 即为待求问题的弱格式 (Weak Formulation)。该方程的求解就是找到 $\mathbf{u} \in H^1(\Omega) \times H^1(\Omega)$ 和 $p \in L^2(\Omega)$ 对任意 $\mathbf{v} \in H_0^1(\Omega) \times H_0^1(\Omega)$ 和 $q \in L^2(\Omega)$ 都成立。

5.3.3 方程离散

令 $c(\mathbf{w}, \mathbf{u}, \mathbf{v}) = \int_{\Omega} (\mathbf{w} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy$, $a(\mathbf{u}, \mathbf{v}) = \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy$, $b(\mathbf{u}, q) = - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy$, $(\mathbf{f}, \mathbf{v}) = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy$, 待求问题为：找到 $\mathbf{u} \in H^1(\Omega) \times H^1(\Omega)$ 和 $p \in L^2(\Omega)$ 对任意 $\mathbf{v} \in H_0^1(\Omega) \times H_0^1(\Omega)$ 和 $q \in L^2(\Omega)$ 都成立。

$$\begin{aligned} c(\mathbf{u}, \mathbf{u}, \mathbf{v}) + a(\mathbf{u}, \mathbf{v}) + b(\mathbf{v}, p) &= (\mathbf{f}, \mathbf{v}) \\ b(\mathbf{u}, q) &= 0 \end{aligned} \quad (5.69)$$

根据张量双点积的定义可以得到：

$$\begin{aligned} \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) &= \left[\begin{array}{cc} \frac{\partial u_x}{\partial x} & \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \\ \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) & \frac{\partial u_y}{\partial y} \end{array} \right] : \left[\begin{array}{cc} \frac{\partial v_x}{\partial x} & \frac{1}{2} \left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x} \right) \\ \frac{1}{2} \left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x} \right) & \frac{\partial v_y}{\partial y} \end{array} \right] \\ &= \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x} \right) + \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} \end{aligned} \quad (5.70)$$

进一步对上式进行简化，可以得到：

$$\begin{aligned} \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) &= \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + \frac{1}{2} \left(\frac{\partial u_x}{\partial y} \frac{\partial v_x}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_y}{\partial x} + \frac{\partial u_y}{\partial x} \frac{\partial v_x}{\partial y} + \frac{\partial u_y}{\partial x} \frac{\partial v_y}{\partial x} \right) + \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} \end{aligned} \quad (5.71)$$

所以，

$$\begin{aligned} &\int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy \\ &= \int_{\Omega} \nu \left(2 \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + 2 \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_x}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_y}{\partial x} + \frac{\partial u_y}{\partial x} \frac{\partial v_x}{\partial y} + \frac{\partial u_y}{\partial x} \frac{\partial v_y}{\partial x} \right) dx dy \end{aligned} \quad (5.72)$$

同理可以得到：

$$\int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy = \int_{\Omega} \left(u_x \frac{\partial u_x}{\partial x} v_x + u_y \frac{\partial u_x}{\partial y} v_x + u_x \frac{\partial u_y}{\partial x} v_y + u_y \frac{\partial u_y}{\partial y} v_y \right) dx dy \quad (5.73)$$

$$\int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy = \int_{\Omega} p \left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right) dx dy \quad (5.74)$$

$$\int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy = \int_{\Omega} (f_x v_x + f_y v_y) dx dy \quad (5.75)$$

$$\int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = \int_{\Omega} \left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} \right) q dx dy \quad (5.76)$$

所以 Navier-Stokes 方程弱格式的标量形式为：找到 $u_x \in H^1(\Omega)$ 、 $u_y \in H^1(\Omega)$ 和 $p \in L^2(\Omega)$ ，使得对任意 $v_x \in H_0^1(\Omega)$ 、 $v_y \in H_0^1(\Omega)$ 和 $q \in L^2(\Omega)$ 都成立：

$$\begin{aligned} & \int_{\Omega} \left(u_x \frac{\partial u_x}{\partial x} v_x + u_y \frac{\partial u_x}{\partial y} v_x + u_x \frac{\partial u_y}{\partial x} v_y + u_y \frac{\partial u_y}{\partial y} v_y \right) dx dy + \\ & \int_{\Omega} \nu \left(2 \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + 2 \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_x}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_y}{\partial x} + \frac{\partial u_y}{\partial x} \frac{\partial v_x}{\partial y} + \frac{\partial u_y}{\partial x} \frac{\partial v_y}{\partial x} \right) dx dy \\ & - \int_{\Omega} p \left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right) dx dy = \int_{\Omega} (f_x v_x + f_y v_y) dx dy \\ & - \int_{\Omega} \left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} \right) q dx dy = 0 \end{aligned} \quad (5.77)$$

所以在有限元空间上，对任意 $\mathbf{v}_h \in U_h \times U_h$ 和 $q_h \in W_h$ 有：

$$\begin{aligned} & c(\mathbf{u}_h, \mathbf{u}_h, \mathbf{v}_h) + a(\mathbf{u}_h, \mathbf{v}_h) + b(\mathbf{v}_h, p_h) = (\mathbf{f}, \mathbf{v}_h) \\ & b(\mathbf{u}_h, q_h) = 0 \end{aligned} \quad (5.78)$$

在采用有限元方法对速度场与压力场进行求解时，基函数不能任意选取，必须满足一些条件，因为动量方程与连续性方程的耦合会产生稳定性问题，即使程序运行无误也将产生无效结果。该问题与“inf-sup”或“Ladyzhenskaya-Babuska-Brezzi, LBB”有关。

$$\inf_{0 \neq q_h \in W_h} \sup_{0 \neq \mathbf{u}_h \in U_h \times U_h} \frac{b(\mathbf{u}_h, q_h)}{\|\nabla \mathbf{u}_h\|_0 \|q_h\|_0} > \beta \quad (5.79)$$

其中 $\beta > 0$ 为一无关单元大小的常数。

为避免出现稳定性问题，在求解时通常采用“Taylor-Hood”或“Hilbert-Schmidt”基函数对分别处理速度与压力，在“Taylor-Hood”基函数对中，速度基函数阶数通常较压力基函数阶数更高，因此，这里选择速度对应有限单元为 2 次元，压力对应有限元基函数为线性元。

5.3.4 非线性项处理

与稳态 Stokes 问题不同, NS 方程中有对流项, 该项为非线性, 需要采取合适的方法对该项进行线性化处理, 一个直接可靠但收敛运行较慢为 Uzawa 迭代。在本章中, 采用 Newton 迭代法, 在该方法中, 一部分为上一步迭代解, 另一部分为当前迭代解, 加快了迭代速度。在采用 Newton 迭代处理前述方程时, 首先需要确定待求物理场的初始状态, 这里假设初始速度场为 \mathbf{u}^0 , 初始压力场为 p^0 , 对上述方程进行变换可以得到:

$$\begin{aligned} c(\mathbf{u}^n, \mathbf{u}^{n-1}, \mathbf{v}) + c(\mathbf{u}^{n-1}, \mathbf{u}^n, \mathbf{v}) + a(\mathbf{u}^n, \mathbf{v}) + b(\mathbf{v}, p^n) &= (\mathbf{f}, \mathbf{v}) + c(\mathbf{u}^{n-1}, \mathbf{u}^{n-1}, \mathbf{v}) \\ b(\mathbf{u}^n, q) &= 0 \end{aligned} \quad (5.80)$$

其中, $n = 1, 2, \dots, N$ 为迭代次数, 该式中 $\mathbf{u}^n \in H^1(\Omega) \times H^1(\Omega)$, $p^n \in L^2(\Omega)$, 对任意 $\mathbf{v} \in H_0^1(\Omega) \times H_0^1(\Omega)$ 和 $q \in L^2(\Omega)$ 都成立。

上式矢量形式可表示为:

$$\begin{aligned} &\int_{\Omega} (\mathbf{u}^n \cdot \nabla) \mathbf{u}^{n-1} \cdot \mathbf{v} dx dy + \int_{\Omega} (\mathbf{u}^{n-1} \cdot \nabla) \mathbf{u}^n \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}^n) : \mathbb{D}(\mathbf{v}) dx dy \\ &- \int_{\Omega} p^n (\nabla \cdot \mathbf{v}) dx dy = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy + \int_{\Omega} (\mathbf{u}^{n-1} \cdot \nabla) \mathbf{u}^{n-1} \cdot \mathbf{v} dx dy \\ &\quad - \int_{\Omega} (\nabla \cdot \mathbf{u}^n) q dx dy = 0 \end{aligned} \quad (5.81)$$

为方便后续计算及程序实现进一步得到方程的标量形式如下:

$$\begin{aligned} &\int_{\Omega} \left(u_x^n \frac{\partial u_x^{n-1}}{\partial x} v_x + u_y^n \frac{\partial u_x^{n-1}}{\partial y} v_x + u_x^n \frac{\partial u_y^{n-1}}{\partial x} v_y + u_y^n \frac{\partial u_y^{n-1}}{\partial y} v_y \right) dx dy \\ &+ \int_{\Omega} \left(u_x^{n-1} \frac{\partial u_x^n}{\partial x} v_x + u_y^{n-1} \frac{\partial u_x^n}{\partial y} v_x + u_x^{n-1} \frac{\partial u_y^n}{\partial x} v_y + u_y^{n-1} \frac{\partial u_y^n}{\partial y} v_y \right) dx dy \\ &\int_{\Omega} \nu \left(2 \frac{\partial u_x^n}{\partial x} \frac{\partial v_x}{\partial x} + 2 \frac{\partial u_y^n}{\partial y} \frac{\partial v_y}{\partial y} + \frac{\partial u_x^n}{\partial y} \frac{\partial v_x}{\partial y} + \frac{\partial u_y^n}{\partial x} \frac{\partial v_y}{\partial x} + \frac{\partial u_x^n}{\partial x} \frac{\partial v_x}{\partial y} + \frac{\partial u_y^n}{\partial x} \frac{\partial v_y}{\partial x} \right) dx dy \\ &- \int_{\Omega} p^n \left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right) dx dy = \int_{\Omega} (f_x v_x + f_y v_y) dx dy \\ &+ \int_{\Omega} \left(u_x^{n-1} \frac{\partial u_x^{n-1}}{\partial x} v_x + u_y^{n-1} \frac{\partial u_x^{n-1}}{\partial y} v_x + u_x^{n-1} \frac{\partial u_y^{n-1}}{\partial x} v_y + u_y^{n-1} \frac{\partial u_y^{n-1}}{\partial y} v_y \right) dx dy \\ &\quad - \int_{\Omega} q \left(\frac{\partial u_x^n}{\partial x} + \frac{\partial u_y^n}{\partial y} \right) dx dy = 0 \end{aligned} \quad (5.82)$$

在有限元空间上, 当 $n = 1, 2, \dots, N$ 时, 对任意 $\mathbf{v}_h \in U_h \times U_h$ 和 $q_h \in W_h$ 有:

$$\begin{aligned} c(\mathbf{u}_h^n, \mathbf{u}_h^{n-1}, \mathbf{v}_h) + c(\mathbf{u}_h^{n-1}, \mathbf{u}_h^n, \mathbf{v}_h) + a(\mathbf{u}_h^n, \mathbf{v}_h) + b(\mathbf{v}_h, p_h^n) &= (\mathbf{f}, \mathbf{v}_h) + c(\mathbf{u}_h^{n-1}, \mathbf{u}_h^{n-1}, \mathbf{v}_h) \\ b(\mathbf{u}_h^n, q_h) &= 0 \end{aligned} \quad (5.83)$$

5.3.5 构建线性系统

(1) 刚度矩阵

将待求函数中未知变量用有限元基函数展开, 得到:

$$u_{xh}^n = \sum_{j=1}^{N_b} u_{xj}^n \phi_j, \quad u_{yh}^n = \sum_{j=1}^{N_b} u_{yj}^n \phi_j, \quad p_h^n = \sum_{j=1}^{N_{bp}} p_j^n \psi_j \quad (5.84)$$

选取合适的基函数, 得到关于 u_{xj}^n 、 u_{yj}^n 和 p_j^n 的线性系统, 然后求解该线性系统即可得到有限元解: $\mathbf{u}_h^n = (u_{xh}^n, u_{yh}^n)^T$ 和 p_h^n 。

在进行 Newton 迭代时, 对于第一个方程, 将 $\mathbf{v}_h = (\phi_i, 0)^T (i = 1, \dots, N_b)$ 和 $\mathbf{v}_h = (0, \phi_i)^T (i = 1, \dots, N_b)$ 代入弱格式, 即对第一组测试函数取 $v_{xh} = \phi_i (i = 1, \dots, N_b)$ 和 $v_{yh} = 0$, 对第二组测试函数取 $v_{xh} = 0$ 和 $v_{yh} = \phi_i (i = 1, \dots, N_b)$ 。

对于第二个方程, 取 $q_h = \psi_i, 0 (i = 1, \dots, N_{bp})$ 。

► 令 $\mathbf{v}_h = (\phi_i, 0)^T$, 即 $v_{xh} = \phi_i$ 和 $v_{yh} = 0 (i = 1, \dots, N_b)$, 得到:

$$\begin{aligned} & \int_{\Omega} \frac{\partial u_{xh}^{n-1}}{\partial x} \left(\sum_{j=1}^{N_b} u_{xj}^n \phi_j \right) \phi_i dx dy + \int_{\Omega} \frac{\partial u_{xh}^{n-1}}{\partial y} \left(\sum_{j=1}^{N_b} u_{yj}^n \phi_j \right) \phi_i dx dy \\ & + \int_{\Omega} u_{xh}^{n-1} \left(\sum_{j=1}^{N_b} u_{xj}^n \frac{\partial \phi_j}{\partial x} \right) \phi_i dx dy + \int_{\Omega} u_{yh}^{n-1} \left(\sum_{j=1}^{N_b} u_{yj}^n \frac{\partial \phi_j}{\partial y} \right) \phi_i dx dy \\ & 2 \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{xj}^n \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{xj}^n \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial y} dx dy \\ & + \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{yj}^n \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial y} dx dy - \int_{\Omega} \left(\sum_{j=1}^{N_{bp}} p_j^n \psi_j \right) \frac{\partial \phi_i}{\partial x} dx dy \\ & = \int_{\Omega} f_x \phi_i dx dy + \int_{\Omega} u_{xh}^{n-1} \frac{\partial u_{xh}^{n-1}}{\partial x} \phi_i dx dy + \int_{\Omega} u_{yh}^{n-1} \frac{\partial u_{xh}^{n-1}}{\partial y} \phi_i dx dy \end{aligned} \quad (5.85)$$

► 令 $\mathbf{v}_h = (0, \phi_i)^T$, 即 $v_{xh} = 0$ 和 $v_{yh} = \phi_i (i = 1, \dots, N_b)$, 得到:

$$\begin{aligned} & \int_{\Omega} \frac{\partial u_{yh}^{n-1}}{\partial x} \left(\sum_{j=1}^{N_b} u_{xj}^n \phi_j \right) \phi_i dx dy + \int_{\Omega} \frac{\partial u_{yh}^{n-1}}{\partial y} \left(\sum_{j=1}^{N_b} u_{yj}^n \phi_j \right) \phi_i dx dy \\ & + \int_{\Omega} u_{xh}^{n-1} \left(\sum_{j=1}^{N_b} u_{yj}^n \frac{\partial \phi_j}{\partial x} \right) \phi_i dx dy + \int_{\Omega} u_{yh}^{n-1} \left(\sum_{j=1}^{N_b} u_{yj}^n \frac{\partial \phi_j}{\partial y} \right) \phi_i dx dy \\ & 2 \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{yj}^n \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial y} dx dy + \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{xj}^n \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial x} dx dy \end{aligned}$$

$$\begin{aligned}
& + \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{yj}^n \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial x} dx dy - \int_{\Omega} \left(\sum_{j=1}^{N_{bp}} p_j^n \psi_j \right) \frac{\partial \phi_i}{\partial y} dx dy \\
& = \int_{\Omega} f_y \phi_i dx dy + \int_{\Omega} u_{xh}^{n-1} \frac{\partial u_{yh}^{n-1}}{\partial x} \phi_i dx dy + \int_{\Omega} u_{yh}^{n-1} \frac{\partial u_{xh}^{n-1}}{\partial y} \phi_i dx dy
\end{aligned} \quad (5.86)$$

► 令 $q_h = \psi_i$ ($i = 1, \dots, N_{bp}$), 得到:

$$- \int_{\Omega} \left(\sum_{j=1}^{N_b} u_{xj}^n \frac{\partial \phi_j}{\partial x} \right) \psi_i dx dy - \int_{\Omega} \left(\sum_{j=1}^{N_b} u_{yj}^n \frac{\partial \phi_j}{\partial y} \right) \psi_i dx dy = 0 \quad (5.87)$$

将上述三式进行整理后可以得到:

$$\begin{aligned}
& \sum_{j=1}^{N_b} u_{xj}^n \left(2 \int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy + \int_{\Omega} \frac{\partial u_{xh}^{n-1}}{\partial x} \phi_j \phi_i dx dy \right. \\
& \quad \left. + \int_{\Omega} u_{xh}^{n-1} \frac{\partial \phi_j}{\partial x} \phi_i dx dy + \int_{\Omega} u_{yh}^{n-1} \frac{\partial \phi_j}{\partial y} \phi_i dx dy \right) + \sum_{j=1}^{N_b} u_{yj}^n \left(\int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial y} dx dy \right. \\
& \quad \left. + \int_{\Omega} \frac{\partial u_{xh}^{n-1}}{\partial y} \phi_j \phi_i dx dy \right) + \sum_{j=1}^{N_{bp}} p_j^n \left(- \int_{\Omega} \psi_j \frac{\partial \phi_i}{\partial x} dx dy \right) \\
& = \int_{\Omega} f_x \phi_i dx dy + \int_{\Omega} u_{xh}^{n-1} \frac{\partial u_{xh}^{n-1}}{\partial x} \phi_i dx dy + \int_{\Omega} u_{yh}^{n-1} \frac{\partial u_{xh}^{n-1}}{\partial y} \phi_i dx dy \\
& \sum_{j=1}^{N_b} u_{xj}^n \left(\int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \frac{\partial u_{yh}^{n-1}}{\partial x} \phi_j \phi_i dx dy \right) + \sum_{j=1}^{N_b} u_{yj}^n \left(2 \int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy + \right. \\
& \quad \left. \int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \frac{\partial u_{yh}^{n-1}}{\partial y} \phi_j \phi_i dx dy + \int_{\Omega} u_{xh}^{n-1} \frac{\partial \phi_j}{\partial x} \phi_i dx dy + \int_{\Omega} u_{yh}^{n-1} \frac{\partial \phi_j}{\partial y} \phi_i dx dy \right) \\
& + \sum_{j=1}^{N_{bp}} p_j^n \left(- \int_{\Omega} \psi_j \frac{\partial \phi_i}{\partial y} dx dy \right) = \int_{\Omega} f_y \phi_i dx dy + \int_{\Omega} u_{xh}^{n-1} \frac{\partial u_{yh}^{n-1}}{\partial x} \phi_i dx dy + \int_{\Omega} u_{yh}^{n-1} \frac{\partial u_{yh}^{n-1}}{\partial y} \phi_i dx dy
\end{aligned} \quad (5.88)$$

$$\sum_{j=1}^{N_b} u_{xj}^n \left(- \int_{\Omega} \frac{\partial \phi_j}{\partial x} \psi_i dx dy \right) + \sum_{j=1}^{N_b} u_{yj}^n \left(- \int_{\Omega} \frac{\partial \phi_j}{\partial y} \psi_i dx dy \right) + \sum_{j=1}^{N_{bp}} p_j^n * 0 = 0 \quad (5.90)$$

为了方便描述, 定义如下矩阵组:

$$\begin{aligned}
\mathbf{K}_1 &= \left[\int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b}, \quad \mathbf{K}_2 = \left[\int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b} \\
\mathbf{K}_3 &= \left[\int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b}, \quad \mathbf{K}_4 = \left[\int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b}
\end{aligned}$$

$$\begin{aligned}\mathbf{K}_5 &= \left[\int_{\Omega} -\psi_j \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b, N_{bp}}, \quad \mathbf{K}_6 = \left[\int_{\Omega} -\psi_j \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b, N_{bp}} \\ \mathbf{K}_7 &= \left[\int_{\Omega} -\frac{\partial \phi_j}{\partial x} \psi_i dx dy \right]_{i,j=1}^{N_{bp}, N_b}, \quad \mathbf{K}_8 = \left[\int_{\Omega} -\frac{\partial \phi_j}{\partial y} \psi_i dx dy \right]_{i,j=1}^{N_{bp}, N_b}\end{aligned}\quad (5.91)$$

定义 0 矩阵为 $\mathbb{O}_1 = [0]_{i,j=1}^{N_{bp}, N_{bp}}$, 该矩阵大小为 $N_{bp} \times N_{bp}$, 由此可以得到线性系统矩阵 \mathbf{K} :

$$\mathbf{K} = \begin{bmatrix} 2\mathbf{K}_1 + \mathbf{K}_2 & \mathbf{K}_3 & \mathbf{K}_5 \\ \mathbf{K}_4 & 2\mathbf{K}_2 + \mathbf{K}_1 & \mathbf{K}_6 \\ \mathbf{K}_7 & \mathbf{K}_8 & \mathbb{O}_1 \end{bmatrix} \quad (5.92)$$

其中, $\mathbf{K}_4 = \mathbf{K}_3^T$, $\mathbf{K}_7 = \mathbf{K}_5^T$, $\mathbf{K}_8 = \mathbf{K}_6^T$, 所以该矩阵为一对称矩阵。

矩阵组中各个矩阵可通过算法 6 分别得到, 详细过程如下:

- 令算法 6 中 $c = v$, $r = 1$, $s = 0$, $p = 1$, $q = 0$, 得到矩阵 \mathbf{K}_1 ;
- 令算法 6 中 $c = v$, $r = 0$, $s = 1$, $p = 0$, $q = 1$, 得到矩阵 \mathbf{K}_2 ;
- 令算法 6 中 $c = v$, $r = 1$, $s = 0$, $p = 0$, $q = 1$, 得到矩阵 \mathbf{K}_3 ;
- 令算法 6 中 $c = -1$, $r = 0$, $s = 0$, $p = 1$, $q = 0$, 得到矩阵 \mathbf{K}_5 ;
- 令算法 6 中 $c = -1$, $r = 0$, $s = 0$, $p = 0$, $q = 1$, 得到矩阵 \mathbf{K}_6 ;

根据线性系统对上述矩阵组进行整理可以得到:

$$\mathbf{K} = \begin{bmatrix} 2\mathbf{K}_1 + \mathbf{K}_2 & \mathbf{K}_3 & \mathbf{K}_5 \\ \mathbf{K}_3^T & 2\mathbf{K}_2 + \mathbf{K}_1 & \mathbf{K}_6 \\ \mathbf{K}_5^T & \mathbf{K}_6^T & \mathbb{O}_1 \end{bmatrix} \quad (5.93)$$

将非线性项矩阵定义为:

$$\begin{aligned}\mathbf{KN}_1 &= \left[\int_{\Omega} \frac{\partial u_{xh}^{n-1}}{\partial x} \phi_j \phi_i dx dy \right]_{i,j=1}^{N_b}, \quad \mathbf{KN}_2 = \left[\int_{\Omega} u_{xh}^{n-1} \frac{\partial \phi_j}{\partial x} \phi_i dx dy \right]_{i,j=1}^{N_b} \\ \mathbf{KN}_3 &= \left[\int_{\Omega} u_{yh}^{n-1} \frac{\partial \phi_j}{\partial y} \phi_i dx dy \right]_{i,j=1}^{N_b}, \quad \mathbf{KN}_4 = \left[\int_{\Omega} \frac{\partial u_{xh}^{n-1}}{\partial y} \phi_j \phi_i dx dy \right]_{i,j=1}^{N_b} \\ \mathbf{KN}_5 &= \left[\int_{\Omega} \frac{\partial u_{yh}^{n-1}}{\partial x} \phi_j \phi_i dx dy \right]_{i,j=1}^{N_b}, \quad \mathbf{KN}_6 = \left[\int_{\Omega} \frac{\partial u_{yh}^{n-1}}{\partial y} \phi_j \phi_i dx dy \right]_{i,j=1}^{N_b}\end{aligned}\quad (5.94)$$

定义 0 矩阵 $\mathbb{O}_2 = [0]_{i,j=1}^{N_b, N_{bp}}$, 则可以得到矩阵 \mathbf{KN} :

$$\mathbf{KN} = \begin{bmatrix} \mathbf{KN}_1 + \mathbf{KN}_2 + \mathbf{KN}_3 & \mathbf{KN}_4 & \mathbb{O}_2 \\ \mathbf{KN}_5 & \mathbf{KN}_6 + \mathbf{KN}_2 + \mathbf{KN}_3 & \mathbb{O}_2 \\ \mathbb{O}_2^T & \mathbb{O}_2^T & \mathbb{O}_1 \end{bmatrix} \quad (5.95)$$

令算法 6 中 $c = \frac{\partial^{d+e} c_h}{\partial x^d \partial y^e}$, 则可以得到计算非线性部分矩阵算法 12, 非线性项矩阵组中各个矩阵可通过该算法分别得到, 详细过程如下:

- 令算法 12 中 $c_h = u_{xh}^{n-1}$, $d = 1$, $e = r = s = p = q = 0$, 得到矩阵 $\mathbf{K}\mathbf{N}_1$;
- 令算法 12 中 $c_h = u_{xh}^{n-1}$, $r = 1$, $d = e = s = p = q = 0$, 得到矩阵 $\mathbf{K}\mathbf{N}_2$;
- 令算法 12 中 $c_h = u_{yh}^{n-1}$, $s = 1$, $d = e = r = p = q = 0$, 得到矩阵 $\mathbf{K}\mathbf{N}_3$;
- 令算法 12 中 $c_h = u_{xh}^{n-1}$, $e = 1$, $d = r = s = p = q = 0$, 得到矩阵 $\mathbf{K}\mathbf{N}_4$;
- 令算法 12 中 $c_h = u_{yh}^{n-1}$, $d = 1$, $e = r = s = p = q = 0$, 得到矩阵 $\mathbf{K}\mathbf{N}_5$;
- 令算法 12 中 $c_h = u_{yh}^{n-1}$, $e = 1$, $d = r = s = p = q = 0$, 得到矩阵 $\mathbf{K}\mathbf{N}_6$;

由此可以得到对应非线性矩阵系统 $\mathbf{K}\mathbf{N}$, 结合已经得到的刚度矩阵 \mathbf{K} , 进而可以得到每次 Newton 迭代中对应的刚度矩阵:

$$\mathbf{K}^n = \mathbf{K} + \mathbf{K}\mathbf{N} \quad (5.96)$$

Algorithm 12 : 2D NS 方程非线性矩阵组装

Require: $\mathbf{K} = \text{sparse}(N_b^{test}, N_b^{trial})$

- 1: **for** $n = 1, \dots, N$ **do**
 - 2: **for** $\alpha = 1, \dots, N_{lb}^{trial}$ **do**
 - 3: **for** $\beta = 1, \dots, N_{lb}^{test}$ **do**
 - 4: Compute $r = \int_{E_n} \frac{\partial^{d+e} c_h}{\partial x^d \partial y^e} \frac{\partial^{r+s} \varphi_{n\alpha}}{\partial x^r \partial y^s} \frac{\partial^{p+q} \varphi_{n\beta}}{\partial x^p \partial y^q} dx dy$
 - 5: $\mathbf{K}(T_b^{test}(\beta, n), T_b^{trial}(\alpha, n)) += r$
 - 6: **end for**
 - 7: **end for**
 - 8: **end for**
-

(2) 载荷向量

定义载荷向量如下:

$$\mathbf{F} = \begin{bmatrix} \mathbf{F}_x \\ \mathbf{F}_y \\ 0 \end{bmatrix} \longrightarrow \mathbf{F}_x = \left[\int_{\Omega} f_x \phi_i dx dy \right]_{i=1}^{N_b}, \quad \mathbf{F}_y = \left[\int_{\Omega} f_y \phi_i dx dy \right]_{i=1}^{N_b} \quad (5.97)$$

这里 0 矩阵大小为 $N_{bp} \times 1$, 由此可以得到载荷向量阵 \mathbf{F} , 载荷向量可以采用算法 7 得到, 详细过程如下:

- 令算法 7 中 $f = f_x$, $p = q = 0$, 得到载荷向量阵 \mathbf{F}_x ;
 - 令算法 7 中 $f = f_y$, $p = q = 0$, 得到载荷向量阵 \mathbf{F}_y ;
- 非线性部分载荷向量定义为:

$$\mathbf{FN} = \begin{bmatrix} \mathbf{FN}_1 + \mathbf{FN}_2 \\ \mathbf{FN}_3 + \mathbf{FN}_4 \\ 0 \end{bmatrix} \quad (5.98)$$

其中, 0 矩阵大小为 $N_{bp} \times 1$, 其余各矩阵定义分别如下:

$$\begin{aligned} \mathbf{FN}_1 &= \left[\int_{\Omega} u_{xh}^{n-1} \frac{\partial u_{xh}^{n-1}}{\partial x} \phi_i dx dy \right]_{i,j=1}^{N_b}, & \mathbf{FN}_2 &= \left[\int_{\Omega} u_{yh}^{n-1} \frac{\partial u_{xh}^{n-1}}{\partial y} \phi_i dx dy \right]_{i,j=1}^{N_b} \\ \mathbf{FN}_3 &= \left[\int_{\Omega} u_{xh}^{n-1} \frac{\partial u_{yh}^{n-1}}{\partial x} \phi_i dx dy \right]_{i,j=1}^{N_b}, & \mathbf{FN}_4 &= \left[\int_{\Omega} u_{yh}^{n-1} \frac{\partial u_{yh}^{n-1}}{\partial y} \phi_i dx dy \right]_{i,j=1}^{N_b} \end{aligned} \quad (5.99)$$

令算法 7 中 $f = \frac{\partial^{d+e} f_{1h}}{\partial x^d \partial y^e} \frac{\partial^{r+s} f_{2h}}{\partial x^r \partial y^s}$, 则可以得到计算非线性部分载荷向量算法 13, 非线性部分载荷向量组中各个矩阵可通过该算法分别得到, 详细过程如下:

- 令算法 13 中 $f_{1h} = u_{xh}^{n-1}$, $f_{2h} = u_{yh}^{n-1}$, $r = 1$, $d = e = s = p = q = 0$, 得到 \mathbf{FN}_1 ;
- 令算法 13 中 $f_{1h} = u_{yh}^{n-1}$, $f_{2h} = u_{xh}^{n-1}$, $s = 1$, $d = e = r = p = q = 0$, 得到 \mathbf{FN}_2 ;
- 令算法 13 中 $f_{1h} = u_{xh}^{n-1}$, $f_{2h} = u_{yh}^{n-1}$, $r = 1$, $d = e = s = p = q = 0$, 得到 \mathbf{FN}_3 ;
- 令算法 13 中 $f_{1h} = u_{yh}^{n-1}$, $f_{2h} = u_{yh}^{n-1}$, $s = 1$, $d = e = r = p = q = 0$, 得到 \mathbf{FN}_4 ;

由此可以得到对应非线性载荷向量 \mathbf{FN} , 结合已经得到的载荷向量 \mathbf{F} , 进而可以得到每次 Newton 迭代中对应的载荷向量:

$$\mathbf{F}^n = \mathbf{F} + \mathbf{FN} \quad (5.100)$$

Algorithm 13 : 2D NS 方程非线性载荷向量组装

Require: $\mathbf{F} = \text{sparse}(N_b, 1)$

- 1: **for** $n = 1, \dots, N$ **do**
- 2: **for** $\beta = 1, \dots, N_{lb}$ **do**
- 3: Compute $r = \int_{E_n} \frac{\partial^{d+e} f_{xh}}{\partial x^d \partial y^e} \frac{\partial^{r+s} f_{yh}}{\partial x^r \partial y^s} \frac{\partial^{p+q} \varphi_{n\beta}}{\partial x^p \partial y^q} dx dy$
- 4: $\mathbf{F}(T_b^{test}(\beta, n), 1) += r$
- 5: **end for**
- 6: **end for**

(3) 未知向量

定义待求向量如下：

$$\mathbf{u}^n = \begin{bmatrix} \mathbf{u}_1^n \\ \mathbf{u}_2^n \\ \mathbf{u}_3^n \end{bmatrix} \longrightarrow \mathbf{u}_1^n = [u_{1j}^n]_{i=1}^{N_b}, \quad \mathbf{u}_2^n = [u_{2j}^n]_{i=1}^{N_b}, \quad \mathbf{u}_3^n = [p_j^n]_{i=1}^{N_{bp}} \quad (5.101)$$

(4) 线性系统组成

由此可以得到每一次 Newton 迭代中待求线性系统：

$$\mathbf{K}^n \mathbf{u}^n = \mathbf{F}^n \quad (5.102)$$

得到线性系统后根据所给问题模型的边界条件进行处理，得到需要求解的线性系统，然后求解该线性系统即可得到对应 Newton 迭代中有限元解，根据计算精度设置重复该迭代过程即可得到最终有限元解，其流程如算法 14 所示。

Algorithm 14 : 2D 稳态 NS 方程有限元求解流程

Require: ν , $\mathbf{f} = \mathbf{f}(x, y)$, $\mathbf{g} = \mathbf{g}(x, y)$

Ensure: $\mathbf{u}(x, y)$ 、 $p(x, y)$

- 1: 由算法 6 刚度矩阵 \mathbf{K}
 - 2: 由算法 7 计算载荷向量 \mathbf{F}
 - 3: **for** $l = 1, \dots, L$ **do**
 - 4: 由算法 12 计算矩阵 \mathbf{KN}
 - 5: 由算法 13 计算向量 \mathbf{FN}
 - 6: 计算: $\mathbf{K}^{(l)} = \mathbf{K} + \mathbf{KN}$ 和 $\mathbf{F}^{(l)} = \mathbf{F} + \mathbf{FN}$
 - 7: 边界条件处理
 - 8: 给定参考点压力
 - 9: 计算等效线性系统 $\mathbf{K}^{(l)} \mathbf{u}^{(l)} = \mathbf{F}^{(l)}$, 得到 $\mathbf{u}^{(l)}$
 - 10: **end for**
-

5.3.6 不同边界条件处理

(1) 第 1 类边界条件

在 Navier-Stokes 方程中，第一类边界条件为速度边界： $\mathbf{u} = \mathbf{g}$ ，也称为无滑移边界 (No-slip)，常用固体法向。由于 Navier-Stokes 问题中速度为一矢量，所以在进行边界处

理时需要同时对矢量分量边界进行处理，这里处理方式仍然采用置 1 法，详细算法流程如算法 11 所示。与 Stokes 方程类似，根据定解条件，需要在计算域内指定参考点压力，以便得到唯一解。

(2) 第 2 类边界条件

Neumann 边界

在 Navier-Stokes 方程中，第 2 类边界为压力边界，其满足如下控制方程：

$$\begin{cases} (\mathbf{u} \cdot \nabla) \mathbf{u} - \nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, & \text{in } \Omega \\ \nabla \cdot \mathbf{u} = 0, & \text{in } \Omega \\ \mathbb{T}(\mathbf{u}, p) \mathbf{n} = \mathbf{p}, & \text{on } \partial\Omega \end{cases} \quad (5.103)$$

其中， $\mathbf{u}(x, y) = (u_x, u_y)^T$ 为速度场， p 为压力场， $\mathbf{f}(x, y) = (f_x, f_y)^T$ 为体积力， $\mathbf{n}(x, y) = (n_x, n_y)^T$ 为边界 $\partial\Omega$ 法向， $\mathbb{T}(\mathbf{u}, p)$ 为压力应力张量， $\mathbf{p}(x, y) = (p_x, p_y)^T$ 为边界 $\partial\Omega$ 上的压力。

由前述推导可知：

$$\begin{aligned} & \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy \\ & - \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \\ & - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \end{aligned} \quad (5.104)$$

将边界条件代入上式可以得到：

$$\begin{aligned} & \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy \\ & = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy + \int_{\partial\Omega} \mathbf{p} \cdot \mathbf{v} ds \\ & - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \end{aligned} \quad (5.105)$$

可以看到，由于非线性项 $\int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy$ 的存在，纯压力边界条件（Neumann）下，对于 Navier-Stokes 方程而言，其所得解是唯一的，这是与 Stokes 方程不同之处。

Dirichlet + Neumann 边界

考虑如下 Navier-Stokes 方程：

$$\begin{cases} (\mathbf{u} \cdot \nabla) \mathbf{u} - \nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, & \text{in } \Omega \\ \nabla \cdot \mathbf{u} = 0, & \text{in } \Omega \\ \mathbb{T}(\mathbf{u}, p) \mathbf{n} = \mathbf{p}, & \text{on } \Gamma_S \subset \partial\Omega \\ \mathbf{u} = \mathbf{g}, & \text{on } \Gamma_D = \partial\Omega / \Gamma_S \end{cases} \quad (5.106)$$

由前述推导可知：

$$\begin{aligned} & \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy \\ & - \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \\ & - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \end{aligned} \quad (5.107)$$

由于边界 $\Gamma_D = \partial\Omega / \Gamma_S$ 上的解由式 $\mathbf{u} = \mathbf{g}$ 确定，所以，可以选择合适的测试函数使其在边界上满足 $\mathbf{v} = 0$ ，边界处控制方程满足：

$$\int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Gamma_S} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds + \int_{\partial\Omega / \Gamma_S} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds \quad (5.108)$$

因此，方程的弱格式就是找到 $\mathbf{u} \in H^1(\Omega) \times H^1(\Omega)$ 和 $p \in L^2(\Omega)$ 使其对任何测试函数 $\mathbf{v} \in H_{0D}^1(\Omega) \times H_{0D}^1(\Omega)$ 和 $q \in L^2(\Omega)$ 满足：

$$\begin{aligned} & \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy \\ & = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy + \int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds \\ & - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \end{aligned} \quad (5.109)$$

其中， $\int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds = \int_{\Gamma_S} p_x v_x ds + \int_{\Gamma_S} p_y v_y ds$ ， $H_{0D}^1(\Omega) = \{v \in H^1(\Omega) : v = 0 \text{ on } \Gamma_D\}$ 。

综上所述，当 NS 方程边界条件包含 Neumann 边界时，仅需要将对应 Neumann 边界对应的载荷向量处增加 $\int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds$ 即可。

(3) 第 3 类边界条件

考虑包含如下边界条件的 NS 方程：

$$\begin{cases} (\mathbf{u} \cdot \nabla) \mathbf{u} - \nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, & \text{in } \Omega \\ \nabla \cdot \mathbf{u} = 0, & \text{in } \Omega \\ \mathbb{T}(\mathbf{u}, p) \mathbf{n} + r\mathbf{u} = \mathbf{q}, & \text{on } \Gamma_R \subseteq \partial\Omega \\ \mathbf{u} = \mathbf{g}, & \text{on } \Gamma_D = \partial\Omega / \Gamma_R \end{cases} \quad (5.110)$$

由前述推导可知：

$$\begin{aligned} & \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy \\ & - \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \\ & - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \end{aligned} \quad (5.111)$$

由于边界 $\Gamma_D = \partial\Omega / \Gamma_R$ 上的解由式 $\mathbf{u} = \mathbf{g}$ 确定，所以，可以选择合适的测试函数使其在边界上满足 $\mathbf{v} = 0$ ，边界处控制方程满足：

$$\begin{aligned} \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds &= \int_{\Gamma_R} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds + \int_{\partial\Omega / \Gamma_R} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds \\ &= \int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds - \int_{\Gamma_R} r\mathbf{u} \cdot \mathbf{v} ds \end{aligned} \quad (5.112)$$

因此，方程的弱格式就是找到 $\mathbf{u} \in H^1(\Omega) \times H^1(\Omega)$ 和 $p \in L^2(\Omega)$ 使其对任何测试函数 $\mathbf{v} \in H_{0D}^1(\Omega) \times H_{0D}^1(\Omega)$ 和 $q \in L^2(\Omega)$ 满足：

$$\begin{aligned} & \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy \\ & + \int_{\Gamma_R} r\mathbf{u} \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy + \int_{\Gamma_R} (\mathbf{q} \cdot \mathbf{v}) ds \\ & - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \end{aligned} \quad (5.113)$$

其中， $\int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds = \int_{\Gamma_R} q_x v_x ds + \int_{\Gamma_R} q_y v_y ds$ ， $\int_{\Gamma_R} r\mathbf{u} \cdot \mathbf{v} ds = \int_{\Gamma_R} r_x u_x v_x ds + \int_{\Gamma_R} r_y u_y v_y ds$ ， $H_{0D}^1(\Omega) = \{\mathbf{v} \in H^1(\Omega) : \mathbf{v} = 0 \text{ on } \Gamma_D\}$ 。

所以，上式即为包含第 3 类边界条件时 NS 方程对应弱格式，需要在 Robin 边界条件对应刚度矩阵处增加 $\int_{\Gamma_R} r\mathbf{u} \cdot \mathbf{v} ds$ ，在载荷向量处增加 $\int_{\Gamma_R} (\mathbf{q} \cdot \mathbf{v}) ds$ 。

(4) 混合边界条件

考虑包含如下边界条件的 NS 方程：

$$\left\{ \begin{array}{l} (\mathbf{u} \cdot \nabla) \mathbf{u} - \nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, \quad \text{in } \Omega \\ \nabla \cdot \mathbf{u} = 0, \quad \text{in } \Omega \\ \mathbb{T}(\mathbf{u}, p) \mathbf{n} = \mathbf{p}, \quad \text{on } \Gamma_S \subset \partial\Omega \\ \mathbb{T}(\mathbf{u}, p) \mathbf{n} + r\mathbf{u} = \mathbf{q}, \quad \text{on } \Gamma_R \subseteq \partial\Omega \\ \mathbf{u} = \mathbf{g}, \quad \text{on } \Gamma_D = \partial\Omega / (\Gamma_S \cup \Gamma_R) \end{array} \right. \quad (5.114)$$

由前述推导可知：

$$\begin{aligned} & \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy \\ & - \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \\ & \quad - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \end{aligned} \quad (5.115)$$

由于边界 $\Gamma_D = \partial\Omega / (\Gamma_S \cup \Gamma_R)$ 上的解由式 $\mathbf{u} = \mathbf{g}$ 确定，所以，可以选择合适的测试函数使其在该边界上满足 $\mathbf{v} = 0$ ，边界处控制方程满足：

$$\begin{aligned} & \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Gamma_S} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds + \int_{\Gamma_R} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds \\ & + \int_{\partial\Omega / (\Gamma_S \cup \Gamma_R)} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} + \int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds - \int_{\Gamma_R} r\mathbf{u} \cdot \mathbf{v} ds \end{aligned} \quad (5.116)$$

因此，方程的弱格式就是找到 $\mathbf{u} \in H^1(\Omega) \times H^1(\Omega)$ 和 $p \in L^2(\Omega)$ 使其对任何测试函数 $\mathbf{v} \in H_{0D}^1(\Omega) \times H_{0D}^1(\Omega)$ 和 $q \in L^2(\Omega)$ 满足：

$$\begin{aligned} & \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy \\ & + \int_{\Gamma_R} r\mathbf{u} \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy + \int_{\Gamma_R} (\mathbf{q} \cdot \mathbf{v}) ds + \int_{\Gamma_S} (\mathbf{p} \cdot \mathbf{v}) ds \\ & \quad - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \end{aligned} \quad (5.117)$$

其中， $\int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds = \int_{\Gamma_R} q_x v_x ds + \int_{\Gamma_R} q_y v_y ds$ ， $\int_{\Gamma_R} r\mathbf{u} \cdot \mathbf{v} ds = \int_{\Gamma_R} r_x u_x v_x ds + \int_{\Gamma_R} r_y u_y v_y ds$ ， $H_{0D}^1(\Omega) = \{\mathbf{v} \in H^1(\Omega) : \mathbf{v} = 0 \text{ on } \Gamma_D\}$ 。

在求解该系统时，依次按照 Robin 型、Neumann 型、Dirichlet 型边界条件进行处理，始终保证 Dirichlet 边界发挥作用，具体而言为：在 Robin 型对应边界刚度矩阵增加 $\int_{\Gamma_R} r\mathbf{u} \cdot \mathbf{v} ds$ ，载荷向量增加 $\int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds$ ；在 Neumann 型对应边界载荷向量增加 $\int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds$ ；对 Dirichlet 类型边界对应刚度矩阵采用“置 1”法，载荷向量为给定边界条件值。

5.3.7 误差计算

Navier-Stokes 方程有限元解的正确性验证方法与 Stokes 方程正确性验证方式相同，主要通过有限元解与解析解（如果存在或者较为方便得到的话）的误差进行衡量。这里通过：1、最大节点值误差；2、无穷范数误差 L^∞ ；3、 L^2 范数误差；4、 H^1 范数误差进行恒量。与前述问题模型误差不同的是，Stokes 方程需要对速度场、压力场误差分别进行计算，且由于采用“Taylor-Hood”元，速度场和压力场的误差对应计算方法及精度有所区别。

(1) 最大节点误差

分别计算速度 x 、 y 方向上最大节点误差，随后取二者中最大值作为计算域中速度的最大节点值误差，同时计算压力的最大节点值误差。

$$\begin{aligned} maxNodeErrorU &= \max(maxNodeErrorU_x, maxNodeErrorU_y) \\ maxNodeErrorP &= \max(resP(i) - exactP(i)) \end{aligned} \quad (5.118)$$

(2) 无穷范数误差 L^∞

$$\begin{aligned} \|u_x - u_{xh}\|_\infty &= \sup_{\Omega} |u_x - u_{xh}| \\ \|u_y - u_{yh}\|_\infty &= \sup_{\Omega} |u_y - u_{yh}| \\ \rightarrow \|\mathbf{u} - \mathbf{u}_h\|_\infty &= \max(\|u_x - u_{xh}\|_\infty, \|u_y - u_{yh}\|_\infty) \\ \rightarrow \|p - p_h\|_\infty &= \sup_{\Omega} |p - p_h| \end{aligned} \quad (5.119)$$

(3) L^2 范数误差

$$\begin{aligned} \|u_x - u_{xh}\|_0 &= \sqrt{\int_{\Omega} (u_x - u_{xh})^2 dx dy} \\ \|u_y - u_{yh}\|_0 &= \sqrt{\int_{\Omega} (u_y - u_{yh})^2 dx dy} \\ \rightarrow \|\mathbf{u} - \mathbf{u}_h\|_0 &= \sqrt{\|u_x - u_{xh}\|_0^2 + \|u_y - u_{yh}\|_0^2} \\ \rightarrow \|p - p_h\|_0 &= \sqrt{\int_{\Omega} (p - p_h)^2 dx dy} \end{aligned} \quad (5.120)$$

(4) H^1 范数误差

$$\begin{aligned}
 \|u_x - u_{xh}\|_1 &= \sqrt{\int_{\Omega} \left(\frac{\partial (u_x - u_{xh})}{\partial x} \right)^2 + \left(\frac{\partial (u_x - u_{xh})}{\partial y} \right)^2 dx dy} \\
 \|u_y - u_{yh}\|_1 &= \sqrt{\int_{\Omega} \left(\frac{\partial (u_y - u_{yh})}{\partial x} \right)^2 + \left(\frac{\partial (u_y - u_{yh})}{\partial y} \right)^2 dx dy} \\
 \rightarrow \|u - u_h\|_1 &= \sqrt{\|u_x - u_{xh}\|_1^2 + \|u_y - u_{yh}\|_1^2} \\
 \rightarrow \|p - p_h\|_1 &= \sqrt{\int_{\Omega} \left(\frac{\partial (p - p_h)}{\partial x} \right)^2 + \left(\frac{\partial (p - p_h)}{\partial y} \right)^2 dx dy}
 \end{aligned} \tag{5.121}$$

5.4 应用实例及程序实现

5.4.1 Example 1

例 5.1 【问题描述】以下述稳态 2D Stokes 方程为例对前述有限元方法进行验证，模型计算域为： $\Omega = [0, 1] \times [-0.25, 0]$ ，取 $\nu = 1$ 。

$$\begin{cases} -\nabla \cdot \mathbf{T}(\mathbf{u}, p) = \mathbf{f}, & (x, y) \in \Omega \\ \nabla \cdot \mathbf{u} = 0, & (x, y) \in \Omega \\ u_x = e^{-y}, x = 0; \quad u_x = y^2 + e^{-y}, x = 1 \\ u_x = \frac{1}{16}x^2 + e^{0.25}, y = -0.25; \quad u_x = 1, y = 0 \\ u_y = 2, x = 0; \quad u_y = -\frac{2}{3}y^3 + 2, x = 1 \\ u_y = \frac{1}{96}x + 2 - \pi \sin(\pi x), y = -0.25; \quad u_y = 2 - \pi \sin(\pi x), y = 0 \end{cases}$$

其中：

$$\begin{aligned}
 f_x &= -2\nu x^2 - 2\nu y^2 - \nu e^{-y} + \pi^2 \cos(\pi x) \cos(2\pi y) \\
 f_y &= 4\nu x y - \nu \pi^3 \sin(\pi x) + 2\pi(2 - \pi \sin(\pi x)) \sin(2\pi y)
 \end{aligned}$$

【说明】 上述问题的解析解为：

$$\begin{cases} u_x = x^2 y^2 + e^{-y}, \quad u_y = -\frac{2}{3}x y^3 + 2 - \pi \sin(\pi x) \\ p = -(2 - \pi \sin(\pi x)) \cos(2\pi y) \end{cases}$$

(1) 网格划分与信息矩阵生成

根据 Stokes 方程有限元方法对网格的要求，速度场需要使用较压力场更高阶的网格单元以确保仿真结果的有效性。本算例中压力场采用线性三角形单元，如图 5-1a 所示；速度场采用二次三角形单元，如图 5-1b 所示。

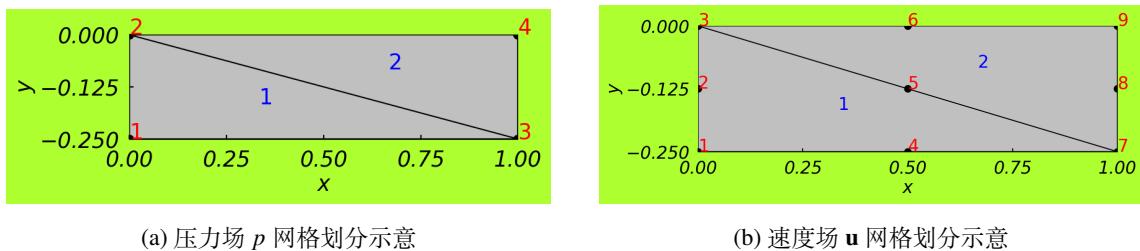


图 5-1 Stokes 方程网格划分示意

(2) 刚度矩阵组装

根据算法 6 计算式 (5.31) 中矩阵 \mathbf{K}_1 、 \mathbf{K}_2 、 \mathbf{K}_3 、 \mathbf{K}_5 、 \mathbf{K}_6 ，之后由式 (5.32) 得到刚度矩阵 \mathbf{K} ，整个待求线性系统矩阵规模为： $NN \times NN = 22 \times 22$ 。各刚度矩阵分别如下：

$$\mathbf{K}_1 = \begin{bmatrix} 0.1250 & 0 & 0 & -0.1667 & 0 & 0 & 0.0417 & 0 & 0 \\ 0 & 0.3333 & 0 & 0 & -0.3333 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1250 & 0 & 0 & -0.1667 & 0 & 0 & 0.0417 \\ -0.1667 & 0 & 0 & 0.3333 & 0 & 0 & -0.1667 & 0 & 0 \\ 0 & -0.3333 & 0 & 0 & 0.6667 & 0 & 0 & -0.3333 & 0 \\ 0 & 0 & -0.1667 & 0 & 0 & 0.3333 & 0 & 0 & -0.1667 \\ 0.0417 & 0 & 0 & -0.1667 & 0 & 0 & 0.1250 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.3333 & 0 & 0 & 0.3333 & 0 \\ 0 & 0 & 0.0417 & 0 & 0 & -0.1667 & 0 & 0 & 0.1250 \end{bmatrix} \quad (5.122)$$

$$\mathbf{K}_2 = \begin{bmatrix} 2.0000 & -2.6667 & 0.6667 & 0 & 0 & 0 & 0 & 0 & 0 \\ -2.6667 & 5.3333 & -2.6667 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.6667 & -2.6667 & 2.0000 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5.3333 & -5.3333 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -5.3333 & 10.6667 & -5.3333 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -5.3333 & 5.3333 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2.0000 & -2.6667 & 0.6667 \\ 0 & 0 & 0 & 0 & 0 & 0 & -2.6667 & 5.3333 & -2.6667 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.6667 & -2.6667 & 2.0000 \end{bmatrix} \quad (5.123)$$

$$\mathbf{K}_3 = \begin{bmatrix} 0.5000 & 0 & 0 & -0.6667 & 0 & 0 & 0.1667 & 0 & 0 \\ -0.6667 & 0.6667 & 0 & 0.6667 & -0.6667 & 0 & 0 & 0 & 0 \\ 0.1667 & -0.6667 & 0 & 0 & 0.6667 & 0 & -0.1667 & 0 & 0 \\ 0 & 0.6667 & 0 & 0.6667 & -0.6667 & 0 & -0.6667 & 0 & 0 \\ 0 & -0.6667 & 0.6667 & -0.6667 & 1.3333 & -0.6667 & 0.6667 & -0.6667 & 0 \\ 0 & 0 & -0.6667 & 0 & -0.6667 & 0.6667 & 0 & 0.6667 & 0 \\ 0 & 0 & -0.1667 & 0 & 0.6667 & 0 & 0 & -0.6667 & 0.1667 \\ 0 & 0 & 0 & 0 & -0.6667 & 0.6667 & 0 & 0.6667 & -0.6667 \\ 0 & 0 & 0.1667 & 0 & 0 & -0.6667 & 0 & 0 & 0.5000 \end{bmatrix} \quad (5.124)$$

$$\mathbf{K}_5 = \begin{bmatrix} 0.0417 & 0 & 0 & 0 \\ 0.0417 & 0.0833 & 0.0417 & 0 \\ 0 & 0.0417 & 0 & 0 \\ -0.0417 & 0 & 0.0417 & 0 \\ -0.0417 & -0.0417 & 0.0417 & 0.0417 \\ 0 & -0.0417 & 0 & 0.0417 \\ 0 & 0 & -0.0417 & 0 \\ 0 & -0.0417 & -0.0833 & -0.0417 \\ 0 & 0 & 0 & -0.0417 \end{bmatrix}, \quad \mathbf{K}_6 = \begin{bmatrix} 0.1667 & 0 & 0 & 0 \\ -0.1667 & 0.1667 & 0 & 0 \\ 0 & -0.1667 & 0 & 0 \\ 0.1667 & 0.1667 & 0.3333 & 0 \\ -0.1667 & 0.1667 & -0.1667 & 0.1667 \\ 0 & -0.3333 & -0.1667 & -0.1667 \\ 0 & 0 & 0.1667 & 0 \\ 0 & 0 & -0.1667 & 0.1667 \\ 0 & 0 & 0 & -0.1667 \end{bmatrix} \quad (5.125)$$

将所得各矩阵按照式 (5.32) 组装得到整体刚度矩阵 \mathbf{K} 。

(3) 载荷向量组装

载荷向量与矩阵相对应，根据算法 7 计算式 (5.34) 中载荷向量 \mathbf{F}_x 、 \mathbf{F}_y ，将所得载荷向量进行合并即可得到整体载荷向量 \mathbf{F} ，根据当前划分网格节点数量可知载荷向量的维度为： $NN = 22$ 。所得载荷向量分别如下：

$$\begin{aligned} \mathbf{F}_x &= \begin{bmatrix} -0.0069 & 0.1269 & 0.1677 & -0.0238 & -0.1251 & -0.1662 & -0.0530 & -0.3221 & -0.0623 \end{bmatrix}^T \\ \mathbf{F}_y &= \begin{bmatrix} 0.0319 & -0.7162 & 0.0739 & -0.9040 & -1.8521 & -0.9874 & 0.0202 & -0.7419 & 0.0837 \end{bmatrix}^T \end{aligned} \quad (5.126)$$

将所得载荷向量进行组装即可得到系统的载荷向量 \mathbf{F} 。

(4) 边界条件处理

将所得待求系统刚度矩阵及载荷向量进行边界条件处理即可得到待求线性系统，本问题中边界类型为 Dirichlet，采用前述章节中“置 1”法进行处理，同时注意在计算域中选定一点作为压力参考点以得到唯一解。

(5) 线性系统求解及后处理

求解边界处理后的线性系统，可以得到本问题的有限元解，采用前述网格划分得到的速度场与压力场的解分别为：

$$\begin{aligned} u_x &= \begin{bmatrix} 1.2840 & 1.1331 & 1 & 1.2992 & 1.1396 & 1 & 1.3465 & 1.1488 & 1 \end{bmatrix}^T \\ u_y &= \begin{bmatrix} 2 & 2 & 2 & -1.1367 & -1.1411 & -1.1416 & 2.0104 & 2.0013 & 2 \end{bmatrix}^T \\ p &= \begin{bmatrix} 0 & -2 & 0 & -2 \end{bmatrix}^T \end{aligned} \quad (5.127)$$

为方便验证本章所述方法的正确性，将本问题中网格细化为 8×2 ，将所得有限元解与解析解进行比对。图 5–2a 和 5–2b 所示分别为速度场 u_x 的有限元解与解析解，图 5–2c 和 5–2d 所示分别为速度场 u_y 的有限元解与解析解，图 5–3a 和 5–3b 所示分别为压力场 p 的有限元解与解析解。可以看到对于速度场而言由于采用 2 阶三角形单元在有限网格单元情况下与解析解吻合较好，压力场由于采用线性三角形单元其求解精度与解析解具有较大差距，可以通过细化网格或者使用更高阶的网格单元减小这种差距。

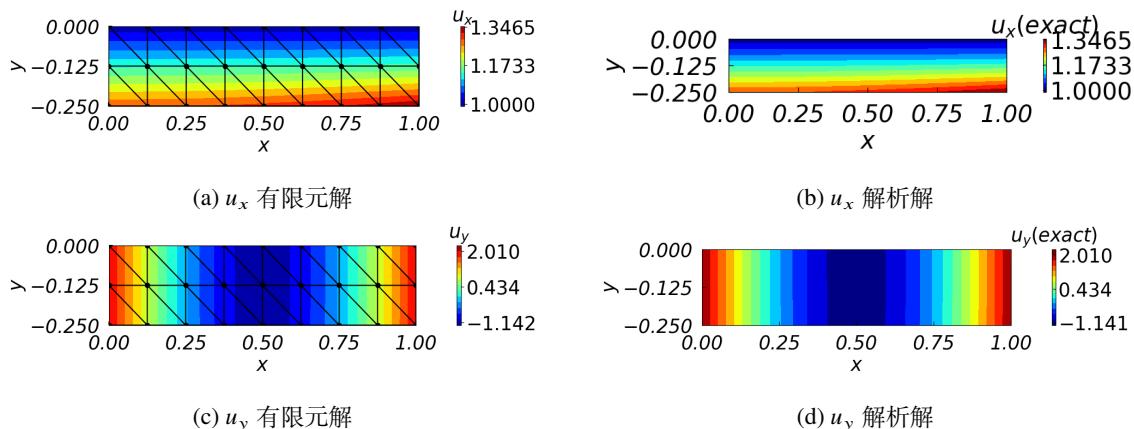


图 5–2 Stokes 方程速度场 \mathbf{u} 有限元解与解析解

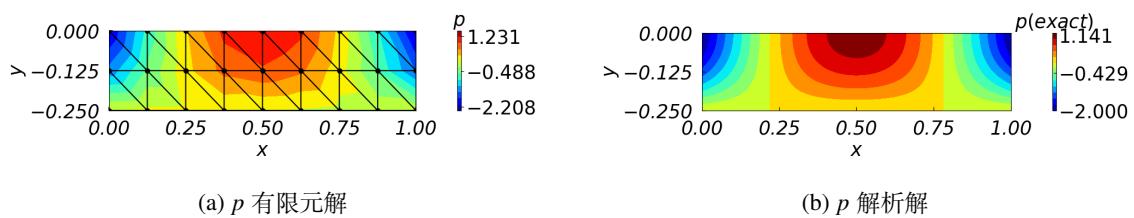


图 5–3 Stokes 方程压力场 p 有限元解与解析解

表 5–1 和表 5–2 分别给出了不同空间步长下，采用三角形网格单元时 Stokes 方程速度场 \mathbf{u} 和压力场 p 的各类误差范数参考：

表 5-1 不同空间步长下速度场 \mathbf{u} 各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	4.9361×10^{-4}	1.6765×10^{-3}	3.5687×10^{-4}	2.0424×10^{-2}
1/16	3.6378×10^{-5}	2.0256×10^{-4}	4.4059×10^{-5}	5.0674×10^{-3}
1/32	2.3274×10^{-6}	2.5182×10^{-5}	5.4832×10^{-6}	1.2623×10^{-3}
1/64	1.8147×10^{-7}	3.1057×10^{-6}	6.8444×10^{-7}	3.1522×10^{-4}

表 5-2 不同空间步长下压力场 p 各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	2.0794×10^{-1}	1.3124×10^{-1}	2.1810×10^{-2}	1.2651×10^0
1/16	5.5796×10^{-2}	4.5401×10^{-2}	8.4643×10^{-3}	6.3072×10^{-1}
1/32	1.3930×10^{-2}	1.2473×10^{-2}	2.4475×10^{-3}	3.1369×10^{-1}
1/64	3.4685×10^{-3}	3.2434×10^{-3}	6.5205×10^{-4}	1.5658×10^{-1}

5.4.2 Example 2

例 5.2 【问题描述】以下述稳态 2D Navier-Stokes 方程为例，对前述有限元方法进行验证，模型计算域为： $\Omega = [0, 1] \times [-0.25, 0]$ ，取 $\nu = 1$ 。

$$\begin{cases} (\mathbf{u} \cdot \nabla) \mathbf{u} - \nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, \quad (x, y) \in \Omega \\ \nabla \cdot \mathbf{u} = 0, \quad (x, y) \in \Omega \\ u_x = e^{-y}, \quad x = 0; \quad u_x = y^2 + e^{-y}, \quad x = 1 \\ u_x = \frac{1}{16}x^2 + e^{0.25}, \quad y = -0.25; \quad u_x = 1, \quad y = 0 \\ u_y = 2, \quad x = 0; \quad u_y = -\frac{2}{3}y^3 + 2, \quad x = 1 \\ u_y = \frac{1}{96}x + 2 - \pi \sin(\pi x), \quad y = -0.25; \quad u_y = 2 - \pi \sin(\pi x), \quad y = 0 \end{cases}$$

其中：

$$\begin{aligned} f_x &= -2\nu x^2 - 2\nu y^2 - \nu e^{-y} + \pi^2 \cos(\pi x) \cos(2\pi y) \\ &\quad + 2xy^2(x^2y^2 + e^{-y}) + (-2xy^3/3 + 2 - \pi \sin(\pi x))(2x^2y - e^{-y}) \\ f_y &= 4\nu xy - \nu \pi^3 \sin(\pi x) + 2\pi(2 - \pi \sin(\pi x)) \sin(2\pi y) \\ &\quad + (x^2y^2 + e^{-y})(-2y^3/3 - \pi^2 \cos(\pi x)) + (-2xy^3/3 + 2 - \pi \sin(\pi x))(-2xy^2) \end{aligned}$$

【说明】上述问题的解析解为：

$$\begin{cases} u_x = x^2y^2 + e^{-y}, & u_y = -\frac{2}{3}xy^3 + 2 - \pi \sin(\pi x) \\ p = -(2 - \pi \sin(\pi x)) \cos(2\pi y) \end{cases}$$

(1) 网格划分与信息矩阵生成

根据 Navier-Stokes 方程有限元方法对网格的要求，速度场需要使用较压力场更高阶的网格单元以确保仿真结果的有效性。本算例中压力场采用线性三角形单元，如图 5-1a 所示；速度场采用二次三角形单元，如图 5-1b 所示。

(2) 线性部分刚度矩阵组装

根据算法 6 计算式 (5.91) 中矩阵 \mathbf{K}_1 、 \mathbf{K}_2 、 \mathbf{K}_3 、 \mathbf{K}_5 、 \mathbf{K}_6 ，之后由式 (5.92) 得到线性部分刚度矩阵 \mathbf{K} ，矩阵规模为： $NN \times NN = 22 \times 22$ 。由前所述可以知道，Navier-Stokes 方程线性部分刚度矩阵与 Stokes 方程中各刚度矩阵相同，故此处不再重复给出。

(3) 线性部分载荷向量组装

Navier-Stokes 方程中线性部分载荷向量 \mathbf{F}_x 、 \mathbf{F}_y 可根据算法 7 通过式 (5.97) 得到，将所得载荷向量进行合并即可得到线性部分载荷向量 \mathbf{F} ，根据当前划分网格节点数量可知载荷向量的维度为： $NN = 22$ 。由于载荷向量与控制方程中源项相关，Navier-Stokes 方程中源项与 Stokes 方程中源项不同，故所得线性部分载荷向量不同，一种参考解如下：

$$\begin{aligned} \mathbf{F}_x &= \begin{bmatrix} -0.0194 & 0.1071 & 0.1565 & 0.0045 & -0.0779 & -0.1469 & -0.0681 & -0.3470 & -0.0719 \end{bmatrix}^T \\ \mathbf{F}_y &= \begin{bmatrix} -0.0347 & -1.0596 & -0.0926 & -1.0197 & -1.8257 & -0.8694 & 0.2021 & -0.4060 & 0.1275 \end{bmatrix}^T \end{aligned} \quad (5.128)$$

将所得载荷向量进行组装即可得到系统的载荷向量 \mathbf{F} 。

(4) 非线性部分刚度矩阵组装

在 Navier-Stokes 方程中，存在非线性项，非线性部分刚度矩阵需要在每次牛顿迭代中根据前一步中速度场重新计算：通过前一步速度场 \mathbf{u}^{n-1} ，根据式 (5.94) 计算非线性部分各刚度矩阵 \mathbf{KN}_1 、 \mathbf{KN}_2 、 \mathbf{KN}_3 、 \mathbf{KN}_4 、 \mathbf{KN}_4 、 \mathbf{KN}_6 ，之后由式 (5.95) 得到当前牛顿迭代中非线性部分刚度矩阵 \mathbf{KN} 。

假设开始牛顿迭代前，速度场为 0 场，则在第一次牛顿迭代中得到的非线性部分各刚度矩阵均为 0 矩阵。

(5) 非线性部分载荷向量组装

与 Navier-Stokes 方程中非线性部分刚度矩阵类似，非线性部分载荷向量需要在每次牛顿迭代中根据前一步中速度场重新计算：通过前一步速度场 \mathbf{u}^{n-1} ，根据式 (5.99) 计算非线性部分载荷向量 \mathbf{FN}_1 、 \mathbf{FN}_2 、 \mathbf{FN}_3 、 \mathbf{FN}_4 ，之后由式 (5.98) 得到当前牛顿迭代中非线性部分载荷向量 \mathbf{FN} 。

初始速度场为 0 时所得非线性部分载荷向量也均为 0 矩阵。

(6) 边界条件处理

在求解 Navier-Stokes 方程时，由前述方法得到非线性部分刚度矩阵及载荷向量后，需要将其与线性部分刚度矩阵及载荷向量进行组装，得到当前迭代步中系统的刚度矩阵及载荷向量，之后进行边界条件处理。本问题中边界条件为 Dirichlet 类型，采用前述章节中“置 1”法进行处理，同时注意在计算域中选定一点作为压力参考点以得到唯一解。

(7) 线性系统求解及后处理

求解边界处理后的线性系统，可以得到当前迭代步下系统的有限元解，之后根据收敛性条件对该解进行判断，若符合条件则所得解为系统的解；若不符合条件则将其作为下一步迭代的初值，继续迭代，详细算法流程如算法 14 所示。

为方便验证本章所述方法的正确性，将本问题中网格细化为 8×2 ，将所得有限元解与解析解进行比对。图 5-4a、5-4b 和 5-4c 分别为速度场 u_x 、 u_y 和压力场 p 的有限元解。该问题解析解与前述算例中一致，其结果分别如图 5-2b、5-2d 和 5-3b 所示，在本算例中各物理场最大节点值误差设置为 1×10^{-5} ，牛顿迭代最大次数设置为 20。

可以看到对于速度场而言由于采用 2 阶三角形单元在有限网格单元情况下与解析解吻合较好，压力场由于采用线性三角形单元其求解精度与解析解具有较大差距，可以通过细化网格或者使用更高阶的网格单元减小这种差距。

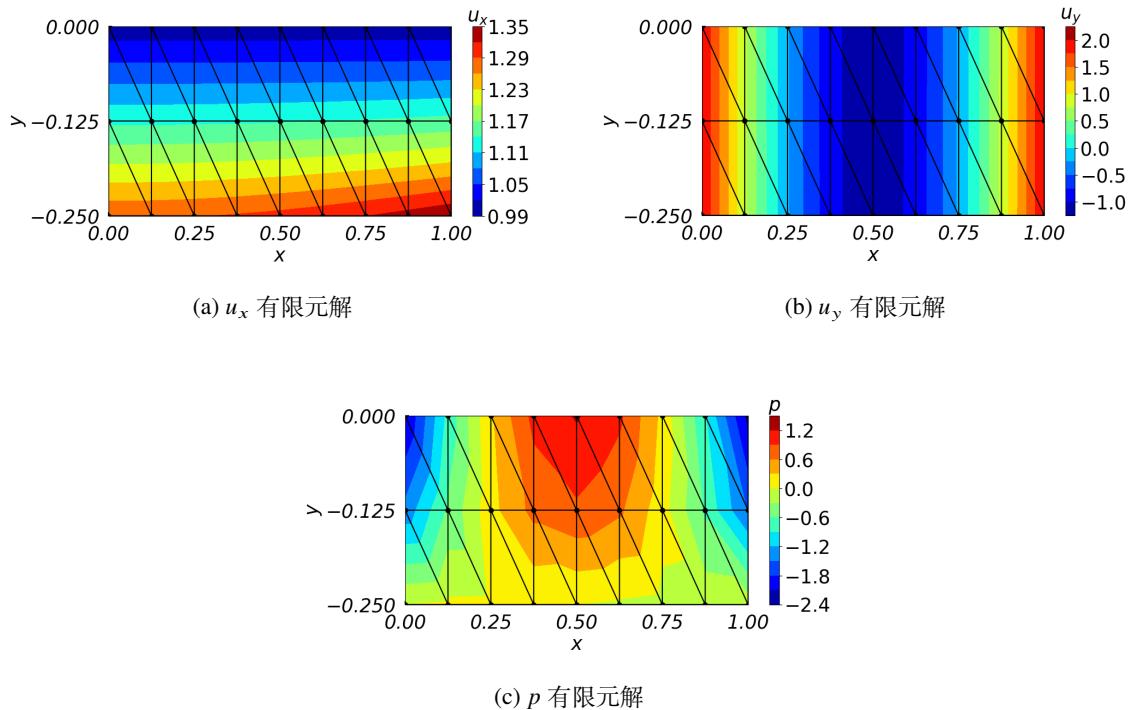


图 5-4 Navier-Stokes 方程有限元解

表 5-3 和表 5-4 分别给出了不同空间步长下，采用三角形网格单元时 Stokes 方程速度场 \mathbf{u} 和压力场 p 的各类误差范数参考：

表 5-3 不同空间步长下速度场 \mathbf{u} 各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	4.5401×10^{-4}	1.6853×10^{-3}	3.5640×10^{-4}	2.0429×10^{-2}
1/16	3.7082×10^{-5}	2.0224×10^{-4}	4.4016×10^{-5}	5.0681×10^{-3}
1/32	2.3809×10^{-6}	2.5167×10^{-5}	5.4798×10^{-6}	1.2623×10^{-3}
1/64	1.8440×10^{-7}	3.1048×10^{-6}	6.8421×10^{-7}	3.1523×10^{-4}

表 5-4 不同空间步长下压力场 p 各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	2.6107×10^0	1.3616×10^{-1}	2.2577×10^{-2}	1.2648×10^0
1/16	2.4299×10^0	4.5862×10^{-2}	8.6669×10^{-3}	6.3069×10^{-1}
1/32	2.3320×10^0	1.2533×10^{-2}	2.4764×10^{-3}	3.1369×10^{-1}
1/64	2.2812×10^0	3.2510×10^{-3}	6.5584×10^{-4}	1.5658×10^{-1}

习题

1. 分别采用三角形、四边形单元有限元方法求解如下方程，并计算所得有限元解的各类误差（最大节点误差、 L^∞ 误差、 L^2 误差、 H^1 误差）。计算域为 $\Omega = [-1, 1]^2$ ，网格规模为 1×1 。

$$\begin{cases} -\nabla \cdot \mathbf{T}(\mathbf{u}, p) = 0, & (x, y) \in \Omega \\ \nabla \cdot \mathbf{u} = 0, & (x, y) \in \Omega \\ u_x(-1, y) = -20y^3, \quad u_x(1, y) = 20y^3 \\ u_x(x, -1) = -20x, \quad u_x(x, 1) = 20x \\ u_y(-1, y) = 5 - yx^4, \quad u_y(1, y) = 5 - 5y^4 \\ u_y(x, -1) = 5x^4 - 5, \quad u_y(x, 1) = 5x^4 - 5 \\ p(0, 0) = 0 \end{cases}$$

该问题解析解为： $u_x = 20xy^3$, $u_y = 5x^4 - 5y^4$, $p = 60x^2y - 20y^3$ 。

6

2D 抛物及双曲方程有限元方法

6.1 引言

本章以 2D2 阶抛物方程和双曲方程^[30-33] 为例，对非稳态问题的有限元方法进行介绍，主要包括：有限元方法中非稳态项的处理方法，给定问题模型弱格式推导、线性系统构建、不同边界条件下线性系统的处理及误差计算等。

6.2 2D2 阶抛物方程

6.2.1 问题模型

生产、生活中许多现象可以用抛物方程描述，如瞬态热传导、扩散现象等。在二维情况下，抛物方程的一般形式如下：

$$\begin{cases} u_t - \nabla \cdot (c \nabla u) = f, & \text{in } \Omega \times [0, T] \\ u = g, & \text{on } \partial\Omega \times [0, T] \\ u = u_0, & \text{at } t = 0 \text{ and in } \Omega \end{cases} \quad (6.1)$$

其中， Ω 为计算域， $[0, T]$ 为时间范围， $f(x, y, t)$ 、 $c(x, y, t)$ 为给定计算域 $\Omega \times [0, T]$ 上的函数， $g(x, y, t)$ 为边界 $\partial\Omega \times [0, T]$ 上的函数， $u_0(x, y)$ 为时间 $t = 0$ 时计算域 Ω 中的函数值， $u(x, y, t)$ 为待求函数值。

6.2.2 弱格式

(1) 方程两端同乘以任意测试函数 (Test Function) $v(x, y)$ 做内积 (Inner Product), 并在整个求解域 Ω 上积分

$$\begin{aligned} u_t - \nabla \cdot (c \nabla u) &= f, \quad \text{in } \Omega \\ \rightarrow u_t v - \nabla \cdot (c \nabla u) v &= f v, \quad \text{in } \Omega \\ \rightarrow \int_{\Omega} u_t v dx dy - \int_{\Omega} \nabla \cdot (c \nabla u) v dx dy &= \int_{\Omega} f v dx dy \end{aligned} \quad (6.2)$$

在上述方程中, $u(x, y, t)$ 为试探函数 (Trial Function), $v(x, y)$ 为测试函数 (Test Function)。

(2) 根据多元分部积分法 (散度定理) 对方程等号左边扩散项进行变换

$$\int_{\Omega} \nabla \cdot (c \nabla u) v dx dy = \int_{\partial\Omega} (c \nabla u \cdot \vec{n}) v ds - \int_{\Omega} c \nabla u \cdot \nabla v dx dy \quad (6.3)$$

其中, \vec{n} 为计算域边界 $\partial\Omega$ 的单位外法向量 (Unit Outer Normal Vector), 由此可以得到:

$$\int_{\Omega} u_t v dx dy + \int_{\Omega} c \nabla u \cdot \nabla v dx dy - \int_{\partial\Omega} (c \nabla u \cdot \vec{n}) v ds = \int_{\Omega} f v dx dy \quad (6.4)$$

待求函数在计算域边界 $\partial\Omega$ 处的值由 $u(x, y, t) = g(x, y, t)$ 给定, 因此可以选择合适的测试函数 $v(x, y)$ 使其在边界 $\partial\Omega$ 处满足 $v(x, y) = 0$, 此时方程 (6.4) 等号左边第三项为零, 可以得到:

$$\int_{\Omega} u_t v dx dy + \int_{\Omega} c \nabla u \cdot \nabla v dx dy = \int_{\Omega} f v dx dy \quad (6.5)$$

方程 (6.5) 即为待求问题的弱格式 (Weak Formulation)。该方程的求解就是找到 $u \in H^1(0, T; H^1(\Omega))$ 对任意 $v \in H_0^1(\Omega)$ 都成立, 其中: $H^1(0, T; H^1(\Omega)) = \left\{ v(t, \cdot), \frac{\partial v}{\partial t}(t, \cdot) \in H^1(\Omega), \forall t \in [0, T] \right\}$ 。

6.2.3 方程离散

令 $a(u, v) = \int_{\Omega} c \nabla u \cdot \nabla v dx dy$, $(f, v) = \int_{\Omega} f v dx dy$, 则式 (6.5) 可改写为:

$$(u_t, v) + a(u, v) = (f, v) \quad (6.6)$$

将某一时刻待求函数中未知变量用有限元基函数展开, 得到:

$$u_h(x, y, t) = \sum_{j=1}^{N_b} u_j(t) \phi_j(x, y) \quad (6.7)$$

选取合适的基函数，得到某一时刻关于 $u_j(t)$ 的线性系统，然后求解该线性系统即可得到有限元解： $u_h(x, y, t)$ 。令测试函数 $v_h = \phi_i (i = 1, \dots, N_b)$ ，则有：

$$\begin{aligned} & \int_{\Omega} \left(\sum_{j=1}^{N_b} u_j(t) \phi_j \right)_t \phi_i dx dy + \int_{\Omega} c \nabla \left(\sum_{j=1}^{N_b} u_j(t) \phi_j \right) \cdot \nabla \phi_i dx dy \\ &= \int_{\Omega} f_x \phi_i dx dy, \quad i = 1, \dots, N_b \end{aligned} \quad (6.8)$$

$$\begin{aligned} & \rightarrow \sum_{j=1}^{N_b} u'_j(t) \left[\int_{\Omega} \phi_j \phi_i dx dy \right] + \sum_{j=1}^{N_b} u_j(t) \left[\int_{\Omega} c \nabla \phi_j \cdot \nabla \phi_i dx dy \right] \\ &= \int_{\Omega} f_x \phi_i dx dy, \quad i = 1, \dots, N_b \end{aligned} \quad (6.9)$$

上式中基函数 $\phi_i (i = 1, \dots, N_b)$ 仅依赖于空间 (x, y) ，而函数参数 c, f 依赖于时间 t 与空间 (x, y) 。

某一时刻刚度矩阵为：

$$\mathbf{K}(t) = [K_{ij}]_{i,j=1}^{N_b} = \left[\int_{\Omega} c \nabla \phi_j \cdot \nabla \phi_i dx dy \right]_{i,j=1}^{N_b} \quad (6.10)$$

定义质量矩阵：

$$\mathbf{M} = [m_{ij}]_{i,j=1}^{N_b} = \left[\int_{\Omega} \phi_j \phi_i dx dy \right]_{i,j=1}^{N_b} \quad (6.11)$$

载荷向量：

$$\mathbf{F}(t) = [f_i]_{i=1}^{N_b} = \left[\int_{\Omega} f \phi_i dx dy \right]_{i=1}^{N_b} \quad (6.12)$$

6.2.4 非稳态项处理

对于非稳态项，通常的离散格式有向前差分（Forward Difference）、向后差分（Backward Difference）、中心差分（Central Difference），这些离散格式都是通过泰勒级数（Taylor Series）变化得到的。以对应的离散格式对方程中非稳态项进行离散，所得对应离散格式分别为向前欧拉（Forward Euler）、向后欧拉（Backward Euler）、Crank-Nicolson 格式。

现以如下表达式为例，说明非稳态项各离散格式推导过程：

$$y'(t) = f(t, y(t)), \quad (a \leq t \leq b), \quad y(a) = g_a \quad (6.13)$$

假设区间 $[a, b]$ 被 N_t 等分，且每个区间大小为 h ，则各个节点可表示为 $t_i = a + ih$ ，其中 $i = 0, \dots, N_t$ ， t_i 处的函数值表示为 $y_i = y(t_i)$ 。

(1) 泰勒级数

空间位置 $x + h$ 处的函数值可由空间位置为 x 处的函数值及其导数的组合得到, 即:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \cdots + \frac{h^n}{n!}f^n(x) \quad (6.14)$$

其中, $f'(x)$ 为函数 $f(x)$ 在 x 处的一阶导数, $f''(x)$ 为函数 $f(x)$ 在 x 处的二阶导数, $f^n(x)$ 为函数 $f(x)$ 在 x 处的 n 阶导数, $R_n = \frac{h^n}{n!}f^n(x)$ 为余项。

同理可以得到 $x - h$ 处的函数值:

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) + \cdots + \frac{h^n}{n!}f^n(x) \quad (6.15)$$

(2) 向前欧拉

将式 (6.14) 进行变换: 用 $f(x+h)$ 及 $f(x)$ 表示该函数在 x 处的一阶导数, 即:

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h) \approx \frac{f(x+h) - f(x)}{h} \quad (6.16)$$

因为该差分格式中使用了 $f(x+h)$ 的值, 因此该差分格式为向前差分。将该差分格式应用与式 (6.13), 可以得到非稳态项的向前欧拉离散格式:

$$\begin{aligned} y'(t) &= f(t, y(t)) \\ \rightarrow y'(t_i) &= f(t_i, y(t_i)), \quad i = 0, \dots, N_t - 1 \\ \rightarrow \frac{y(t_{i+1}) - y(t_i)}{h} + O(h) &= f(t_i, y(t_i)), \quad i = 0, \dots, N_t - 1 \\ \rightarrow \frac{y_{i+1} - y_i}{h} &= f(t_i, y_i), \quad i = 0, \dots, N_t - 1 \\ \rightarrow y_{i+1} &= y_i + hf(t_i, y_i), \quad i = 0, \dots, N_t - 1 \\ y_0 &= y(a) = g_a \end{aligned} \quad (6.17)$$

(3) 向后欧拉

将式 (6.15) 进行变换: 用 $f(x)$ 及 $f(x-h)$ 表示该函数在 x 处的一阶导数, 即:

$$f'(x) = \frac{f(x) - f(x-h)}{h} + O(h) \approx \frac{f(x) - f(x-h)}{h} \quad (6.18)$$

因为该差分格式中使用了 $f(x-h)$ 的值, 因此该差分格式为向后差分。将该差分格式应用与式 (6.13), 可以得到非稳态项的向后欧拉离散格式:

$$y'(t) = f(t, y(t))$$

$$\begin{aligned}
&\rightarrow y'(t_i) = f(t_i, y(t_i)), \quad i = 1, \dots, N_t \\
&\rightarrow \frac{y(t_i) - y(t_{i-1})}{h} + O(h) = f(t_i, y(t_i)), \quad i = 1, \dots, N_t \\
&\rightarrow \frac{y_i - y_{i-1}}{h} = f(t_i, y_i), \quad i = 1, \dots, N_t \\
&\rightarrow \frac{y_{i+1} - y_i}{h} = f(t_{i+1}, y_{i+1}), \quad i = 0, \dots, N_t - 1 \\
&\rightarrow y_{i+1} = y_i + h f(t_{i+1}, y_{i+1}), \quad i = 0, \dots, N_t - 1 \\
y_0 &= y(a) = g_a
\end{aligned} \tag{6.19}$$

(4) Crank-Nicolson 格式

前述向前差分与向后差分均为一阶精度，为提高精度，将式 (6.14) 及 (6.15) 相减后进行变换可以得到：

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2) \approx \frac{f(x+h) - f(x-h)}{2h} \tag{6.20}$$

因为该差分格式中使用了 $f(x+h)$ 及 $f(x-h)$ 的平均值，因此该差分格式为重心差分，该格式具有二阶精度。将该差分格式应用与式 (6.13)，可以得到非稳态项的 Crank-Nicolson 格式：

$$\begin{aligned}
\frac{y_{i+1} - y_i}{h} &= \frac{f(t_{i+1}, y_{i+1}) + f(t_i, y_i)}{2}, \quad i = 0, \dots, N_t - 1 \\
\rightarrow y_{i+1} &= y_i + h \frac{f(t_{i+1}, y_{i+1}) + f(t_i, y_i)}{2}, \quad i = 0, \dots, N_t - 1 \\
y_0 &= y(a) = g_a
\end{aligned} \tag{6.21}$$

(5) 通用离散格式

观察上述向前、向后欧拉及 Crank-Nicolson 格式，可以发现这三种格式均可表示为通用 θ 格式：

$$\frac{y_{i+1} - y_i}{h} = \theta f(t_{i+1}, y_{i+1}) + (1 - \theta) f(t_i, y_i), \quad i = 0, \dots, N_t - 1 \tag{6.22}$$

其中， θ 为权重系数，当 $\theta = 0$ 时为向前欧拉格式，当 $\theta = 1$ 时为向后欧拉格式，当 $\theta = 0.5$ 时为 Crank-Nicolson 格式。

6.2.5 线性系统构建

(1) 刚度矩阵

为方便各元素计算，将各单元局部刚度阵的计算转换至参考单元下进行计算：

$$K_{ij}^e(t) = \int_{\Omega} c \nabla \phi_j \cdot \nabla \phi_i dx dy = \int_{\Omega} c(\xi, \eta) \left(\frac{\partial \psi_j}{\partial x} \frac{\partial \psi_i}{\partial x} + \frac{\partial \psi_j}{\partial y} \frac{\partial \psi_i}{\partial y} \right) |J| d\xi d\eta \quad (6.23)$$

刚度矩阵各元素主要由 x 、 y 分量组成，因此，可以分别计算对应分量刚度矩阵之后再将其合并得到总刚度矩阵，详细过程如下：

► 令算法 15 中 $c = c(c, y, t)$, $r = 1$, $s = 0$, $p = 1$, $q = 0$, 得到 x 分量；

► 令算法 15 中 $c = c(c, y, t)$, $r = 0$, $s = 1$, $p = 0$, $q = 1$, 得到 y 分量；

将得到的分量相加得到总刚度矩阵各元素的值： $K_{ij}^e(t) = K_{ij,x}^e(t) + K_{ij,y}^e(t)$ 。

质量矩阵各元素，可类比刚度矩阵元素计算方法得到：

$$m_{ij}^e = \int_{\Omega} \phi_j \phi_i dx dy = \int_{\Omega} (\psi_j \psi_i) |J| d\xi d\eta \quad (6.24)$$

令算法 15 中 $c = 1$, $r = s = p = q = 0$ 即可。

Algorithm 15 : 2D 抛物方程刚度矩阵组装

Require: $\mathbf{K} = \text{sparse}(N_b^{test}, N_b^{trial})$

1: **for** $n = 1, \dots, N$ **do**

2: **for** $\alpha = 1, \dots, N_{lb}^{trial}$ **do**

3: **for** $\beta = 1, \dots, N_{lb}^{test}$ **do**

4: Compute $r = \int_{E_n} c(t) \frac{\partial^{r+s} \varphi_{n\alpha}}{\partial x^r \partial y^s} \frac{\partial^{p+q} \varphi_{n\beta}}{\partial x^p \partial y^q} dx dy$

5: $\mathbf{K}(T_b^{test}(\beta, n), T_b^{trial}(\alpha, n)) += r$

6: **end for**

7: **end for**

8: **end for**

(2) 载荷向量

某一时刻载荷向量各元素计算方法与刚度矩阵各元素计算方式类似：

$$f_i^e(t) = \int_{\Omega} f(\xi, \eta) \psi_i(\xi, \eta) |J| d\xi d\eta \quad (6.25)$$

令算法 16 中 $f = f(x, y, t)$, $p = q = 0$, 得到载荷向量阵各元素。

Algorithm 16 : 2D 抛物方程载荷向量组装

Require: $\mathbf{F} = \text{sparse}(N_b, 1)$

```

1: for  $n = 1, \dots, N$  do
2:   for  $\beta = 1, \dots, N_{lb}$  do
3:     Compute  $r = \int_{E_n} f(t) \frac{\partial^{p+q} \varphi_{n\beta}}{\partial x^p \partial y^q} dx dy$ 
4:      $\mathbf{F}(T_b^{test}(\beta, n), 1) += r$ 
5:   end for
6: end for

```

(3) 未知向量

定义待求向量如下：

$$\mathbf{u}(t) = [u_j(t)]_{i=1}^{N_b} \quad (6.26)$$

(4) 线性系统组成

根据得到的刚度矩阵、载荷向量和未知向量，可以得到线性系统：

$$\mathbf{M}\dot{\mathbf{u}}(t) + \mathbf{K}(t)\mathbf{u}(t) = \mathbf{F}(t) \quad (6.27)$$

采用前述 θ 格式离散非稳态项，可以得到线性系统的离散格式：

$$\begin{aligned} & \mathbf{M} \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + \theta \mathbf{K}(t_{n+1}) \mathbf{u}^{n+1} + (1 - \theta) \mathbf{K}(t_n) \mathbf{u}^n \\ &= \theta \mathbf{F}(t_{n+1}) + (1 - \theta) \mathbf{F}(t_n), \quad n = 0, \dots, N_t - 1 \end{aligned} \quad (6.28)$$

其中， N_t 为时间分割数， Δt 为时间步长。

将所得式进行整理变换可以得到：

$$\begin{aligned} & \left[\frac{\mathbf{M}}{\Delta t} + \theta \mathbf{K}(t_{n+1}) \right] \mathbf{u}^{n+1} \\ &= \theta \mathbf{F}(t_{n+1}) + (1 - \theta) \mathbf{F}(t_n) + \left[\frac{\mathbf{M}}{\Delta t} - (1 - \theta) \mathbf{K}(t_n) \right] \mathbf{u}^n, \quad n = 0, \dots, N_t - 1 \end{aligned} \quad (6.29)$$

进一步简化得到等效线性系统表达式如下：

$$\tilde{\mathbf{K}}\mathbf{u}^{n+1} = \tilde{\mathbf{F}}(t), \quad n = 0, \dots, N_t - 1 \quad (6.30)$$

其中， $\tilde{\mathbf{K}} = \frac{\mathbf{M}}{\Delta t} + \theta \mathbf{K}(t_{n+1})$ ， $\tilde{\mathbf{F}} = \theta \mathbf{F}(t_{n+1}) + (1 - \theta) \mathbf{F}(t_n) + \left[\frac{\mathbf{M}}{\Delta t} - (1 - \theta) \mathbf{K}(t_n) \right] \mathbf{u}^n$ 。

因此，抛物型方程的有限元求解算法流程如算法 17 所示。

Algorithm 17 : 2D 抛物方程有限元求解算法流程

Require: $c = c(x, y, t)$, $f = f(x, y, t)$, $g = g(x, y, t)$, $u_0 = u_0(x, y)$

Ensure: $u(x, y, t)$

- 1: 网格剖分, 待求函数初始化 u_0
 - 2: 根据算法 15 计算质量矩阵 \mathbf{M}
 - 3: 设置时间步长 Δt , 时间分割数 N_t
 - 4: **for** $n = 0, \dots, N_t - 1$ **do**
 - 5: $t_{n+1} = (n + 1)\Delta t$, $t_n = n\Delta t$
 - 6: 由算法 15 计算 t_{n+1} 与 t_n 时刻刚度矩阵 \mathbf{K}^{n+1} 和 \mathbf{K}^n
 - 7: 由算法 16 计算 t_{n+1} 与 t_n 时刻载荷向量 \mathbf{F}^{n+1} 和 \mathbf{F}^n
 - 8: 边界条件处理
 - 9: 计算等效线性系统 $\tilde{\mathbf{K}}\mathbf{u}^{n+1} = \tilde{\mathbf{F}}(t)$, 得到 \mathbf{u}^{n+1}
 - 10: **end for**
-

如果方程中函数参数 c 与时间无关, 则仅需要在时间迭代前计算一次刚度矩阵 \mathbf{K} , 在时间迭代内刚度矩阵无需再次计算 (算法 17 第 6 步), 可以大幅减少计算量。进一步令 $\mathbf{u}^{n+\theta} = \theta\mathbf{u}^{n+1} + (1 - \theta)\mathbf{u}^n$, 如果 $\theta \neq 0$, 则有 $\mathbf{u}^{n+1} - \mathbf{u}^n = \frac{\mathbf{u}^{n+\theta} - \mathbf{u}^n}{\theta}$, 由前所述可知:

$$\begin{aligned} & \mathbf{M} \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + \theta \mathbf{K} \mathbf{u}^{n+1} + (1 - \theta) \mathbf{K} \mathbf{u}^n = \theta \mathbf{F}(t_{n+1}) + (1 - \theta) \mathbf{F}(t_n) \\ & \mathbf{M} \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + \mathbf{K} [\theta \mathbf{u}^{n+1} + (1 - \theta) \mathbf{u}^n] = \theta \mathbf{F}(t_{n+1}) + (1 - \theta) \mathbf{F}(t_n) \\ & \mathbf{M} \frac{\mathbf{u}^{n+\theta} - \mathbf{u}^n}{\theta \Delta t} + \mathbf{K} \mathbf{u}^{n+\theta} = \theta \mathbf{F}(t_{n+1}) + (1 - \theta) \mathbf{F}(t_n) \\ & \rightarrow \left[\frac{\mathbf{M}}{\theta \Delta t} + \mathbf{K} \right] \mathbf{u}^{n+\theta} = \theta \mathbf{F}(t_{n+1}) + (1 - \theta) \mathbf{F}(t_n) + \frac{\mathbf{M} \mathbf{u}^n}{\theta \Delta t} \end{aligned} \quad (6.31)$$

所以可以得到等效线性系统的另一种表达形式:

$$\tilde{\mathbf{K}}^\theta \mathbf{u}^{n+\theta} = \tilde{\mathbf{F}}^\theta(t), \quad n = 0, \dots, N_t - 1 \quad (6.32)$$

其中, $\tilde{\mathbf{K}}^\theta = \frac{\mathbf{M}}{\theta \Delta t} + \mathbf{K}$, $\tilde{\mathbf{F}}^{n+\theta} = \theta \mathbf{F}(t_{n+1}) + (1 - \theta) \mathbf{F}(t_n) + \frac{\mathbf{M} \mathbf{u}^n}{\Delta t}$, $\mathbf{u}^{n+1} = \frac{\mathbf{u}^{n+\theta} - \mathbf{u}^n}{\theta} + \mathbf{u}^n$ 。

6.2.6 不同边界条件处理

考虑如下包含第 1、2、3 类边界条件的混合边界条件抛物型问题模型：

$$\begin{cases} u_t - \nabla \cdot (c \nabla u) = f, & \text{in } \Omega \times [0, T] \\ \nabla u \cdot \vec{n} = p, & \text{on } \Gamma_N \times [0, T] \\ \nabla u \cdot \vec{n} + ru = q, & \text{on } \Gamma_R \times [0, T] \\ u = g, & \text{on } \partial\Omega / (\Gamma_N \cup \Gamma_R) \times [0, T] \\ u = u_0, & \text{at } t = 0 \text{ and in } \Omega \end{cases} \quad (6.33)$$

由前述推导可知：

$$\int_{\Omega} u_t v dx dy + \int_{\Omega} c \nabla u \cdot \nabla v dx dy - \int_{\partial\Omega} (c \nabla u \cdot \vec{n}) v ds = \int_{\Omega} f v dx dy \quad (6.34)$$

由于边界 $\partial\Omega / (\Gamma_N \cup \Gamma_R)$ 上的解由式 $u(x, y, t) = g(x, y, t)$ 给定，因此可以选择合适的测试函数 $v(x, y)$ 使其在边界 $\partial\Omega / (\Gamma_N \cup \Gamma_R)$ 处满足 $v(x, y) = 0$ ，由此可以得到待求方程在混合边界条件下的弱格式：

$$\int_{\Omega} u_t v dx dy + \int_{\Omega} c \nabla u \cdot \nabla v dx dy + \int_{\Gamma_R} \textcolor{red}{cruv ds} = \int_{\Omega} f v dx dy + \int_{\Gamma_N} \textcolor{red}{cpv ds} + \int_{\Gamma_R} \textcolor{red}{cqv ds} \quad (6.35)$$

在求解该系统时，依次按照 Robin 型、Neumann 型、Dirichlet 型边界条件进行处理，始终保证 Dirichlet 边界发挥作用，具体而言为：在 Robin 型对应边界刚度矩阵增加 $\int_{\Gamma_R} \textcolor{red}{cruv ds}$ ，载荷向量增加 $\int_{\Gamma_R} \textcolor{red}{cqv ds}$ ；在 Neumann 型对应边界载荷向量增加 $\int_{\Gamma_N} \textcolor{red}{cpv ds}$ ；对 Dirichlet 类型边界对应刚度矩阵采用“置 1”法，载荷向量为给定边界条件值。

6.2.7 各项异性抛物型方程

考虑如下包含第 1、2、3 类边界条件的混合边界条件抛物型问题模型：

$$\begin{cases} u_t - \nabla \cdot (c \nabla u) = f, & \text{in } \Omega \times [0, T] \\ \nabla u \cdot \vec{n} = p, & \text{on } \Gamma_N \times [0, T] \\ \nabla u \cdot \vec{n} + ru = q, & \text{on } \Gamma_R \times [0, T] \\ u = g, & \text{on } \partial\Omega / (\Gamma_N \cup \Gamma_R) \times [0, T] \\ u = u_0, & \text{at } t = 0 \text{ and in } \Omega \end{cases} \quad (6.36)$$

其中， $c = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$ 为各项异性系数矩阵，其处理方法可参考第 三 章相关内容。

6.2.8 误差计算

本章所述抛物型方程待求函数值为一标量，其误差计算可参考第 三 章中椭圆方程误差计算方法。

6.3 2D2 阶双曲方程

6.3.1 问题模型

2D2 阶双曲方程 (2D second order hyperbolic) 具有如下形式：

$$\begin{cases} u_{tt} - \nabla \cdot (c \nabla u) = f, & \text{in } \Omega \times [0, T] \\ u = g, & \text{on } \partial\Omega \times [0, T] \\ u = u_0, \frac{\partial u}{\partial t} = u_{00}, & \text{at } t = 0 \text{ and in } \Omega \end{cases} \quad (6.37)$$

其中， Ω 为计算域， $[0, T]$ 为时间范围， $f(x, y, t)$ 、 $c(x, y, t)$ 为给定计算域 $\Omega \times [0, T]$ 上函数参数， $g(x, y, t)$ 为给定边界 $\partial\Omega \times [0, T]$ 上函数， $u_0(x, y)$ 、 $u_{00}(x, y)$ 为初始时刻 $t = 0$ 时计算域 Ω 内函数值， $u(x, y, t)$ 为待求函数。

6.3.2 弱格式

(1) 方程两端同乘以任意测试函数 (Test Function) $v(x, y)$ ，并在整个求解域 Ω 上积分

$$\begin{aligned} u_{tt} - \nabla \cdot (c \nabla u) &= f, \quad \text{in } \Omega \\ \rightarrow u_{tt} v - \nabla \cdot (c \nabla u) v &= f v, \quad \text{in } \Omega \\ \rightarrow \int_{\Omega} u_{tt} v dx dy - \int_{\Omega} \nabla \cdot (c \nabla u) v dx dy &= \int_{\Omega} f v dx dy \end{aligned} \quad (6.38)$$

在上述方程中， $u(x, y, t)$ 为试探函数 (Trial Function)， $v(x, y)$ 为测试函数 (Test Function)。

(2) 根据多元分部积分法 (散度定理) 对方程等号左边项进行变换

$$\int_{\Omega} \nabla \cdot (c \nabla u) v dx dy = \int_{\partial\Omega} (c \nabla u \cdot \vec{n}) v ds - \int_{\Omega} c \nabla u \cdot \nabla v dx dy \quad (6.39)$$

其中， \vec{n} 为计算域边界 $\partial\Omega$ 的单位外法向量 (Unit Outer Normal Vector)，由此可以得到：

$$\int_{\Omega} u_{tt} v dx dy + \int_{\Omega} c \nabla u \cdot \nabla v dx dy - \int_{\partial\Omega} (c \nabla u \cdot \vec{n}) v ds = \int_{\Omega} f v dx dy \quad (6.40)$$

待求函数在计算域边界 $\partial\Omega$ 处的值由 $u(x, y, t) = g(x, y, t)$ 给定，因此可以选择合适的测试函数 $v(x, y)$ 使其在边界 $\partial\Omega$ 处满足 $v(x, y) = 0$ ，可以得到：

$$\int_{\Omega} u_{tt} v dx dy + \int_{\Omega} c \nabla u \cdot \nabla v dx dy = \int_{\Omega} f v dx dy \quad (6.41)$$

方程 (6.41) 即为待求问题的弱格式 (Weak Formulation)。该方程的求解就是找到 $u \in H^2(0, T; H^2(\Omega))$ 对任意 $v \in H_0^1(\Omega)$ 都成立，其中： $H^2(0, T; H^2(\Omega)) = \left\{ v(t, \cdot), \frac{\partial v}{\partial t}(t, \cdot), \frac{\partial^2 v}{\partial t^2}(t, \cdot) \in H^2(\Omega), \forall t \in [0, T] \right\}$ 。

6.3.3 方程离散

令 $a(u, v) = \int_{\Omega} c \nabla u \cdot \nabla v dx dy$, $(f, v) = \int_{\Omega} f v dx dy$, 则式 (6.41) 可以表示为：

$$(u_{tt}, v) + a(u, v) = (f, v) \quad (6.42)$$

将某一时刻待求函数中未知变量用有限元基函数展开，得到：

$$u_h(x, y, t) = \sum_{j=1}^{N_b} u_j(t) \varphi_j(x, y) \quad (6.43)$$

选择合适的基函数，得到某一时刻关于 $u_j(t)$ 的线性系统，然后求解该线性系统即可得到有限元解： $u_h(x, y, t)$ 。令测试函数 $v(x, y) = \varphi_i(x, y)$ ，可以得到：

$$\begin{aligned} & \int_{\Omega} \left(\sum_{j=1}^{N_b} u_j(t) \phi_j \right)_{tt} \phi_i dx dy + \int_{\Omega} c \nabla \left(\sum_{j=1}^{N_b} u_j(t) \phi_j \right) \cdot \nabla \phi_i dx dy \\ &= \int_{\Omega} f_x \phi_i dx dy, \quad i = 1, \dots, N_b \end{aligned} \quad (6.44)$$

$$\begin{aligned} & \rightarrow \sum_{j=1}^{N_b} u_j''(t) \left[\int_{\Omega} \phi_j \phi_i dx dy \right] + \sum_{j=1}^{N_b} u_j(t) \left[\int_{\Omega} c \nabla \phi_j \cdot \nabla \phi_i dx dy \right] \\ &= \int_{\Omega} f_x \phi_i dx dy, \quad i = 1, \dots, N_b \end{aligned} \quad (6.45)$$

上式中基函数 ϕ_i , ($i = 1, \dots, N_b$) 仅依赖于空间 (x, y) ，而函数参数 c 、 f 可能依赖于时间 t 与空间 (x, y) 。

某一时刻刚度矩阵为：

$$\mathbf{K}(t) = [K_{ij}]_{i,j=1}^{N_b} = \left[\int_{\Omega} c \nabla \phi_j \cdot \nabla \phi_i dx dy \right]_{i,j=1}^{N_b} \quad (6.46)$$

定义质量矩阵：

$$\mathbf{M} = [m_{ij}]_{i,j=1}^{N_b} = \left[\int_{\Omega} \phi_j \phi_i dx dy \right]_{i,j=1}^{N_b} \quad (6.47)$$

载荷向量：

$$\mathbf{F}(t) = [f_i]_{i=1}^{N_b} = \left[\int_{\Omega} f \phi_i dx dy \right]_{i=1}^{N_b} \quad (6.48)$$

6.3.4 构建线性系统

可以看到双曲方程所得各矩阵系统与抛物方程完全一致，因此，该问题模型中各矩阵、载荷向量可以直接使用抛物方程中对应构建方法。其区别在于，双曲方程中非稳态项具有二阶导数，所得线性系统具有如下形式：

$$\mathbf{M} \mathbf{u}''(t) + \mathbf{K}(t) \mathbf{u}(t) = \mathbf{F}(t) \quad (6.49)$$

根据前述抛物方程中对于非稳态项的处理方式，对双曲方程中非稳态项进行处理：对非稳态项采用中心差分进行离散，得到：

$$\mathbf{M} \frac{\mathbf{u}^{n+1} - 2\mathbf{u}^n + \mathbf{u}^{n-1}}{\Delta t^2} + \mathbf{K} \frac{\mathbf{u}^{n-1} + 2\mathbf{u}^n + \mathbf{u}^{n+1}}{4} = \mathbf{F}(t_n), \quad n = 1, 2, \dots, N_t - 1 \quad (6.50)$$

对上式进行整理变换，可以得到双曲方程待求线性系统：

$$\tilde{\mathbf{K}} \mathbf{u}^{n+1} = \tilde{\mathbf{F}}^{n+1}, \quad n = 1, 2, \dots, N_t - 1 \quad (6.51)$$

其中， $\tilde{\mathbf{K}} = \frac{\mathbf{M}}{\Delta t^2} + \frac{\mathbf{K}}{4}$ ， $\tilde{\mathbf{F}}^{n+1} = \mathbf{F}(t_n) + \left[\frac{2\mathbf{M}}{\Delta t^2} - \frac{\mathbf{K}}{2} \right] \mathbf{u}^n - \left[\frac{\mathbf{M}}{\Delta t^2} + \frac{\mathbf{K}}{4} \right] \mathbf{u}^{n-1}$ 。

6.3.5 不同边界条件处理

考虑如下包含第 1、2、3 类边界条件的混合边界条件双曲型问题模型：

$$\begin{cases} u_{tt} - \nabla \cdot (c \nabla u) = f, & \text{in } \Omega \times [0, T] \\ \nabla u \cdot \vec{n} = p, & \text{on } \Gamma_N \times [0, T] \\ \nabla u \cdot \vec{n} + ru = q, & \text{on } \Gamma_R \times [0, T] \\ u = g, & \text{on } \partial\Omega / (\Gamma_N \cup \Gamma_R) \times [0, T] \\ u = u_0, \quad \frac{\partial u}{\partial t} = u_{00}, & \text{at } t = 0 \text{ and in } \Omega \end{cases} \quad (6.52)$$

由前述推导可知：

$$\int_{\Omega} u_{tt} v dx dy + \int_{\Omega} c \nabla u \cdot \nabla v dx dy - \int_{\partial\Omega} (c \nabla u \cdot \vec{n}) v ds = \int_{\Omega} f v dx dy \quad (6.53)$$

由于边界 $\partial\Omega/(\Gamma_N \cup \Gamma_R)$ 上的解由式 $u(x, y, t) = g(x, y, t)$ 给定，因此可以选择合适的测试函数 $v(x, y)$ 使其在边界 $\partial\Omega/(\Gamma_N \cup \Gamma_R)$ 处满足 $v(x, y) = 0$ ，由此可以得到待求方程在混合边界条件下的弱格式：

$$\begin{aligned} & \int_{\Omega} u_{tt} v dx dy + \int_{\Omega} c \nabla u \cdot \nabla v dx dy + \int_{\Gamma_R} \textcolor{red}{cruv} ds \\ &= \int_{\Omega} f v dx dy + \int_{\Gamma_N} \textcolor{red}{cpv} ds + \int_{\Gamma_R} \textcolor{red}{cqv} ds \end{aligned} \quad (6.54)$$

在求解该系统时，依次按照 Robin 型、Neumann 型、Dirichlet 型边界条件进行处理，始终保证 Dirichlet 边界发挥作用，具体而言为：在 Robin 型对应边界刚度矩阵增加 $\int_{\Gamma_R} \textcolor{red}{cruv} ds$ ，载荷向量增加 $\int_{\Gamma_R} \textcolor{red}{cqv} ds$ ；在 Neumann 型对应边界载荷向量增加 $\int_{\Gamma_N} \textcolor{red}{cpv} ds$ ；对 Dirichlet 类型边界对应刚度矩阵采用“置 1”法，载荷向量为给定边界条件值。

6.3.6 各项异性双曲方程

考虑如下包含第 1、2、3 类边界条件的混合边界条件双曲型问题模型：

$$\left\{ \begin{array}{l} u_{tt} - \nabla \cdot (c \nabla u) = f, \quad \text{in } \Omega \times [0, T] \\ \nabla u \cdot \vec{n} = p, \quad \text{on } \Gamma_N \times [0, T] \\ \nabla u \cdot \vec{n} + ru = q, \quad \text{on } \Gamma_R \times [0, T] \\ u = g, \quad \text{on } \partial\Omega/(\Gamma_N \cup \Gamma_R) \times [0, T] \\ u = u_0, \quad \frac{\partial u}{\partial t} = u_{00}, \quad \text{at } t = 0 \text{ and in } \Omega \end{array} \right. \quad (6.55)$$

其中， $c = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$ ，其处理方式与第 三 章中各向异性椭圆方程一致。

6.3.7 误差计算

双曲方程误差计算思路与抛物型方程误差计算思路一致，可参考第 三 章中椭圆方程误差计算方法。

6.4 应用实例及程序实现

6.4.1 Example 1

例 6.1 【问题描述】以下述 2D 抛物型方程为例对前述有限元方法进行验证，模型计算域为： $\Omega = [0, 2] \times [0, 1]$ ，取 $c = 2$ 。

$$\begin{cases} u_t - \nabla \cdot (2u) = -3e^{x+y+t}, & \text{on } \Omega \times [0, 1] \\ u(x, y, 0) = e^{x+y}, & \text{on } \partial\Omega \\ u = e^{y+t}, x = 0; \quad u = e^{2+y+t}, x = 2 \\ u = e^{x+t}, y = 0; \quad u = e^{x+1+t}, y = 1 \end{cases}$$

【说明】上述问题的解析解为： $u = e^{x+y+t}$ 。

(1) 2D3 节点单元求解

1、网格划分

为方便计算演示，网格划分如图 6-1 所示。

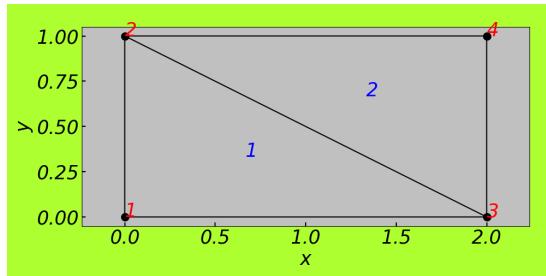


图 6-1 2D 三节点单元网格划分示意

2、刚度矩阵组装

刚度矩阵由质量矩阵 \mathbf{M} 和刚度矩阵 \mathbf{K} 组成，其中质量矩阵仅与基函数有关，不随时间发生变化，刚度矩阵与方程中函数参数 c 有关，对于本算例模型而言， $c = 2$ 为一常数，因此刚度矩阵也为常矩阵。二者详细组装过程可参考前述章节对应刚度矩阵方法，这里直接给出质量矩阵 \mathbf{M} 和刚度矩阵 \mathbf{K} 结果及所得系统刚度矩阵。

$$\mathbf{K} = \begin{bmatrix} 2.5 & -2 & -0.5 & 0 \\ -2 & 2.5 & 0 & -0.5 \\ -0.5 & 0 & 2.5 & -2 \\ 0 & -0.5 & -2 & 2.5 \end{bmatrix}, \quad \mathbf{M} = \begin{bmatrix} 0.1667 & 0.0833 & 0.0833 & 0 \\ 0.0833 & 0.3333 & 0.1667 & 0.0833 \\ 0.0833 & 0.1667 & 0.3333 & 0.0833 \\ 0 & 0.0833 & 0.0833 & 0.1667 \end{bmatrix} \quad (6.56)$$

3、载荷向量组装

抛物型方程中载荷向量与函数参数 f 有关, 本算例模型中 $f = -3e^{x+y+t}$, 因此每次时间迭代中对应载荷向量均不同, 这里分别给出 $t = 0$ 和 $t = 1$ 时的载荷向量供参考。

$$\mathbf{F}_{t=0} = \begin{bmatrix} -6.1788 \\ -24.5690 \\ -31.2480 \\ -27.5300 \end{bmatrix}, \quad \mathbf{F}_{t=1} = \begin{bmatrix} -2.2731 \\ -9.0385 \\ -11.4950 \\ -10.1280 \end{bmatrix} \quad (6.57)$$

4、边界条件处理及线性系统求解

假设当前 $\theta = 0.5$ 、 $\Delta t = 1$, 由式 (6.30) 可得到对应刚度矩阵及载荷向量分别为:

$$\tilde{\mathbf{K}} = \begin{bmatrix} 2.8333 & -1.8333 & -0.3333 & 0 \\ -1.8333 & 3.1667 & 0.3333 & -0.3333 \\ -0.3333 & 0.3333 & 3.1667 & -1.8333 \\ 0 & -0.3333 & -1.8333 & 2.8333 \end{bmatrix}, \quad \tilde{\mathbf{F}} = \begin{bmatrix} -2.2081 \\ -9.0144 \\ -12.0250 \\ -10.4490 \end{bmatrix} \quad (6.58)$$

采用“置 1”法分别对刚度矩阵及载荷向量进行处理, 经过边界处理后可以得到线性系统及各节点函数值, 这里 $\theta = 0.5$, 由线性系统解得未知向量 \mathbf{u} 需要由 $\mathbf{u} = \frac{\mathbf{u}^{n+\theta} - \mathbf{u}^n}{\theta} + \mathbf{u}^n$ 得到最终解 (\mathbf{u}_0) 为初始条件。

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} tmp_1 \\ tmp_2 \\ tmp_3 \\ tmp_4 \end{bmatrix} = \begin{bmatrix} 1.8591 \\ 5.0537 \\ 13.7370 \\ 37.3420 \end{bmatrix} \rightarrow \mathbf{u} = \begin{bmatrix} 2.7183 \\ 7.3891 \\ 20.0855 \\ 54.5982 \end{bmatrix} \quad (6.59)$$

5、后处理

求解边界处理后的线性系统, 所得结果与解析解如图 6-2 所示。

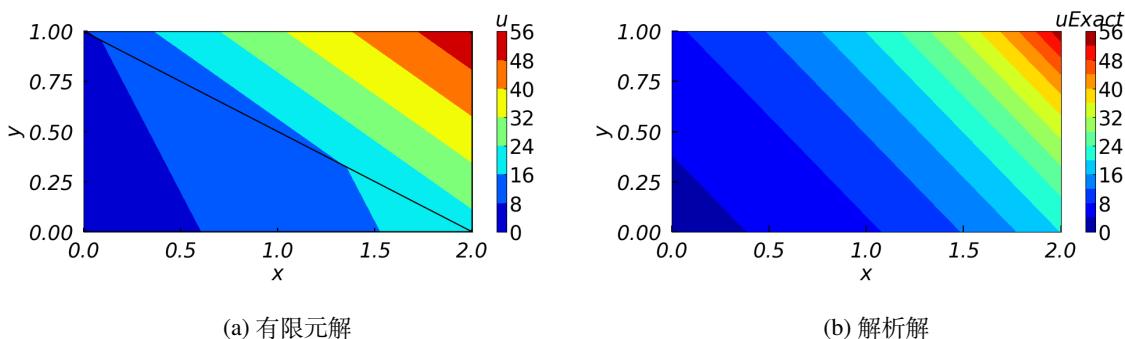


图 6-2 Example 1 三节点单元有限元解与解析解

表 6-1 给出了采用 Crank-Nicolson 离散格式、2D 三节点单元，不同空间、时间步长 ($\Delta t = \Delta x$) 下到达给定计算时间 ($t = 1$) 时各类误差范数结果参考。

表 6-1 $\theta = 0.5$, $\Delta t = \Delta x$, $t = 1$ 三节点单元不同空间步长各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	1.2042×10^{-6}	9.8704×10^{-2}	3.5921×10^{-2}	1.2845×10^0
1/16	7.6275×10^{-8}	2.5483×10^{-2}	8.9715×10^{-3}	6.4187×10^{-1}
1/32	4.7847×10^{-9}	6.4745×10^{-3}	2.2423×10^{-3}	3.2089×10^{-1}
1/64	3.0040×10^{-10}	1.6318×10^{-3}	5.6055×10^{-4}	1.6044×10^{-1}

令 $\theta = 1$, 此时为向后欧拉离散格式, $\Delta t = 4\Delta x^2$, 采用三节点单元重新计算不同空间、时间步长下到达给定计算时间 ($t = 1$) 时仿真结果, 所得各类误差范数结果参考如表 6-2 所示。

表 6-2 $\theta = 1$, $\Delta t = 4\Delta x^2$, $t = 1$ 三节点单元不同空间步长各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	2.5937×10^{-2}	9.8704×10^{-2}	5.0853×10^{-2}	1.2865×10^0
1/16	6.7094×10^{-3}	2.5483×10^{-2}	1.2871×10^{-2}	6.4214×10^{-1}
1/32	1.6886×10^{-3}	6.4745×10^{-3}	3.2279×10^{-3}	3.2092×10^{-1}
1/64	4.2269×10^{-4}	1.6318×10^{-3}	8.0763×10^{-4}	1.6044×10^{-1}

(2) 2D6 节点单元求解

1、网格划分

2D 六节点单元网格划分示意如图 6-3 所示。

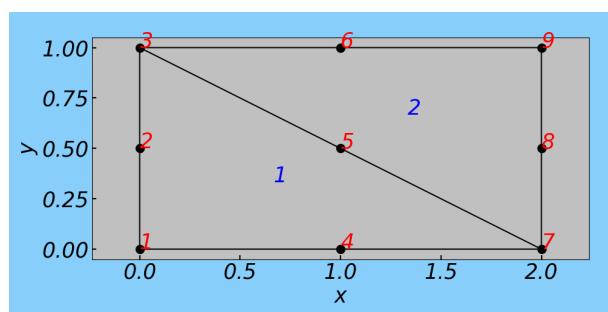


图 6-3 2D 六节点单元网格划分示意

2、刚度矩阵组装

$$\mathbf{K} = \begin{bmatrix} 2.5 & -2.6667 & 0.6667 & -0.6667 & 0 & 0 & 0.1667 & 0 & 0 \\ -2.6667 & 6.6667 & -2.6667 & 0 & -1.3333 & 0 & 0 & 0 & 0 \\ 0.6667 & -2.6667 & 2.5 & 0 & 0 & -0.6667 & 0 & 0 & 0.1667 \\ -0.6667 & 0 & 0 & 6.6667 & -5.3333 & 0 & -0.6667 & 0 & 0 \\ 0 & -1.3333 & 0 & -5.3333 & 13.333 & -5.3333 & 0 & -1.3333 & 0 \\ 0 & 0 & -0.6667 & 0 & -5.3333 & 6.6667 & 0 & 0 & -0.6667 \\ 0.1667 & 0 & 0 & -0.6667 & 0 & 0 & 2.5 & -2.6667 & 0.6667 \\ 0 & 0 & 0 & 0 & -1.3333 & 0 & -2.6667 & 6.6667 & -2.6667 \\ 0 & 0 & 0.1667 & 0 & 0 & -0.6667 & 0.6667 & -2.6667 & 2.5 \end{bmatrix} \quad (6.60)$$

$$\mathbf{M} = \begin{bmatrix} 0.0333 & 0 & -0.0056 & 0 & -0.0222 & 0 & -0.0056 & 0 & 0 \\ 0 & 0.1778 & 0 & 0.0889 & 0.0889 & 0 & -0.0222 & 0 & 0 \\ -0.0056 & 0 & 0.0667 & -0.0222 & 0 & 0 & -0.0111 & -0.0222 & -0.0056 \\ 0 & 0.0889 & -0.0222 & 0.1778 & 0.0889 & 0 & 0 & 0 & 0 \\ -0.0222 & 0.0889 & 0 & 0.0889 & 0.3556 & 0.0889 & 0 & 0.0889 & -0.0222 \\ 0 & 0 & 0 & 0 & 0.0889 & 0.1778 & -0.0222 & 0.0889 & 0 \\ -0.0056 & -0.0222 & -0.0111 & 0 & 0 & -0.0222 & 0.0667 & 0 & -0.0056 \\ 0 & 0 & -0.0222 & 0 & 0.0889 & 0.0889 & 0 & 0.1778 & 0 \\ 0 & 0 & -0.0056 & 0 & -0.0222 & 0 & -0.0056 & 0 & 0.0333 \end{bmatrix} \quad (6.61)$$

3、载荷向量组装

载荷向量与函数参数相关，本算例中载荷向量与时间相关，为 $f = -3e^{x+y+t}$ ，这里给出 $t = 0$ 和 $t = 1$ 时的载荷向量供参考。

$$\mathbf{F}_{t=0} = \begin{bmatrix} 0.3428 \\ -2.3360 \\ 0.9866 \\ -2.8957 \\ -9.8140 \\ -7.9002 \\ -0.4174 \\ -9.4463 \\ -1.4546 \end{bmatrix}, \quad \mathbf{F}_{t=1} = \begin{bmatrix} 0.9317 \\ -6.3498 \\ 2.6818 \\ -7.8714 \\ -26.6773 \\ -21.4751 \\ -1.1347 \\ -25.6777 \\ -3.9540 \end{bmatrix} \quad (6.62)$$

4、边界条件处理及线性系统求解

假设当前 $\theta = 0.5$ 、 $\Delta t = 1$ ，由式 (6.30) 可得到对应刚度矩阵及载荷向量，根据本问

题给定边界条件对所得刚度矩阵与载荷向量进行处理后可得到待求线性系统：

$$\left[\begin{array}{ccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -0.0444 & -1.1556 & 0 & -5.1556 & 14.0444 & -5.1556 & 0 & -1.1556 & -0.0444 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right] \begin{pmatrix} tmp_1 \\ tmp_2 \\ tmp_3 \\ tmp_4 \\ tmp_5 \\ tmp_6 \\ tmp_7 \\ tmp_8 \\ tmp_9 \end{pmatrix} = \begin{pmatrix} 1.8591 \\ 3.0652 \\ 5.0537 \\ 5.0537 \\ -11.7401 \\ 13.7373 \\ 13.7373 \\ 22.6490 \\ 37.3418 \end{pmatrix} \quad (6.63)$$

$$\mathbf{u} = \begin{bmatrix} 2.7183 & 4.4817 & 7.3891 & 7.3891 & 12.1219 & 20.0855 & 20.0855 & 33.1155 & 54.5982 \end{bmatrix}^T \quad (6.64)$$

5、后处理

所得结果与解析解如图 6-4 所示，表 6-3 给出了不同空间步长下 2D 六节点单元各类误差范数参考。

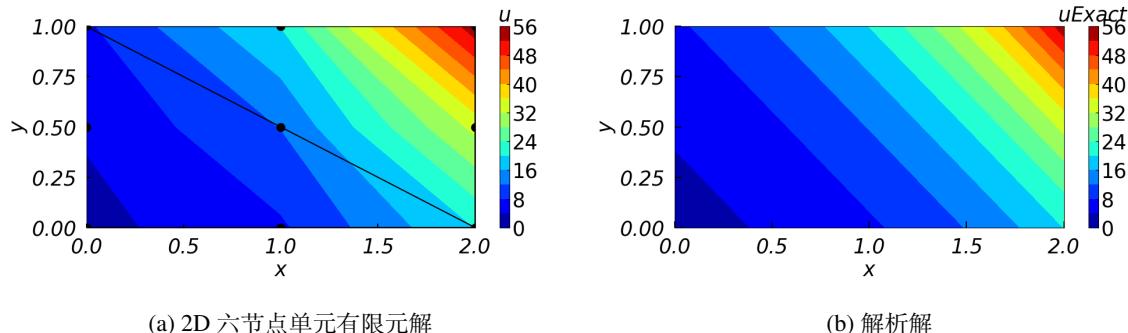


图 6-4 Example 1 六节点单元有限元解与解析解

表 6-3 $\theta = 0.5$, $\Delta t^2 \approx \Delta x^3$, $t = 1$ 六节点单元不同空间步长各类误差结果参考

Cell Size	Δt	maxNodeError	L^∞	L^2	H^1
1/8	1/23	1.5408×10^{-4}	8.1024×10^{-4}	2.8702×10^{-4}	2.0725×10^{-2}
1/16	1/64	1.8672×10^{-5}	1.0403×10^{-4}	3.6236×10^{-5}	5.1789×10^{-3}
1/32	1/181	2.3399×10^{-6}	1.3179×10^{-5}	4.5451×10^{-6}	1.2946×10^{-3}
1/64	1/512	2.8912×10^{-7}	1.6587×10^{-6}	5.6913×10^{-7}	3.2363×10^{-4}

习题

1. 对于初边值问题：

$$\begin{cases} \frac{\partial u}{\partial t} - \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) = -\frac{3}{2} e^{(x+y)/2-t}, & (x, y) \in [0, 1] \times [0, 1], \quad t \in (0, 1] \\ u(x, y, 0) = e^{(x+y)/2}, & (x, y) \in [0, 1] \times [0, 1] \\ u(0, y, t) = e^{y/2-t}, \quad u(1, y, t) = e^{(1+y)/2-t}, & y \in [0, 1], \quad t \in (0, 1] \\ u(x, 0, t) = e^{x/2-t}, \quad u(x, 1, t) = e^{(x+1)/2-t}, & x \in [0, 1], \quad t \in (0, 1] \end{cases}$$

用向前欧拉、向后欧拉、Crank-Nicolson 方法分别求解，取 $\Delta x = \frac{1}{60}$, $\Delta y = \frac{1}{40}$, $\Delta t = \frac{1}{20}$, 并与解析解 $u(x, y, t) = e^{(x+y)/2-t}$ 进行比较。

2. 对于初边值问题：

$$\begin{cases} \frac{\partial u}{\partial t} - \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) = f, & (x, y) \in [0, 1] \times [0, 0.75], \quad t \in (0, 1] \\ u(x, y, 0) = (2 - \pi \sin(\pi x))(-y + \cos(\pi(1 - y))), & (x, y) \in [0, 1] \times [0, 0.75] \\ u(0, y, t) = u(1, y, t) = 2(-y + \cos(\pi(1 - y))) \cos(2\pi t), & y \in [0, 0.75], \quad t \in (0, 1] \\ u(x, 0, t) = (\pi \sin(\pi x) - 2) \cos(2\pi t), & x \in [0, 1], \quad t \in (0, 1] \\ u(x, 0.75, t) = (\sqrt{2}/2 - 0.75)(2 - \pi \sin(\pi x)) \cos(2\pi t), & x \in [0, 1], \quad t \in (0, 1] \end{cases}$$

其中， $f(x, y) = (-\pi^3 \sin(\pi x)(-y + \cos(\pi(1 - y))) - (2 - \pi \sin(\pi x))(-\pi^2 \cos(\pi(1 - y))) \cos(2\pi t) - 2\pi(2 - \pi \sin(\pi x))(-y + \cos(\pi(1 - y))) \sin(2\pi t))$

用向前欧拉、向后欧拉、Crank-Nicolson 方法分别求解，取 $\Delta x = \Delta y = \Delta t = \frac{1}{4}$, 并与解析解 $u(x, y, t) = (2 - \pi \sin(\pi x))(-y + \cos(\pi(1 - y))) \cos(2\pi t)$ 进行比较。

7

2D 非稳态流体与固体问题有限元方法

7.1 引言

本章主要介绍了 2D 非稳态不可压缩流体（大多数液体和气体）有限元方法，主要包括 Stokes 方程和 Navier-Stokes 方程有限元方法及 2D 非稳态线弹性方程。

7.2 非稳态 Stokes 方程

7.2.1 问题模型

非稳态 Stokes 方程具有如下形式：

$$\begin{cases} \mathbf{u}_t - \nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, & \text{in } \Omega \times [0, T] \\ \nabla \cdot \mathbf{u} = 0, & \text{in } \Omega \times [0, T] \\ \mathbf{u} = \mathbf{g}, & \text{on } \partial\Omega \times [0, T] \\ \mathbf{u} = \mathbf{u}_0, \quad p = p_0, & \text{at } t = 0 \text{ and in } \Omega \end{cases} \quad (7.1)$$

其中， Ω 为计算域， $[0, T]$ 为时间范围， $\mathbf{u}(x, y, t) = (u_x, u_y)^T$ 和 $p(x, y, t) = (p_x, p_y)^T$ 分别为待求速度场及压力场， $\mathbf{f}(x, y, t) = (f_x, f_y)^T$ 为给定计算域 $\Omega \times [0, T]$ 上的函数， $\mathbf{g}(x, y, t) = (g_x, g_y)^T$ 为边界 $\partial\Omega \times [0, T]$ 上的函数， $\mathbf{u}_0 = (u_{x0}, u_{y0})^T$ 和 p_0 为 $t = 0$ 时计算域 Ω 中场量。 $\mathbb{T}(\mathbf{u}, p)$ 为应力张量，定义如下：

$$\mathbb{T}(\mathbf{u}, p) = 2\nu\mathbb{D}(\mathbf{u}) - p\mathbb{I} \quad (7.2)$$

其中, ν 为粘度, $\mathbb{D}(\mathbf{u})$ 为应变张量, 定义如下:

$$\mathbb{D}(\mathbf{u}) = \frac{1}{2} (\nabla \mathbf{u} + \nabla \mathbf{u}^T) \quad (7.3)$$

将速度各分量及 ∇ 算符代入可以得到, 应变张量具体形式为:

$$\mathbb{D}(\mathbf{u}) = \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \\ \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) & \frac{\partial u_y}{\partial y} \end{bmatrix} \quad (7.4)$$

将应变张量代入应力张量中可以得到, 应力张量具体形式为:

$$\mathbb{T}(\mathbf{u}, p) = \begin{bmatrix} 2\nu \frac{\partial u_x}{\partial x} - p & \nu \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \\ \nu \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) & 2\nu \frac{\partial u_y}{\partial y} - p \end{bmatrix} \quad (7.5)$$

7.2.2 弱格式

- (1) 方程两端同乘以任意测试函数 (Test Function) $\mathbf{v}(x, y) = (v_x, v_y)^T$ 做内积 (Inner Product), 并在整个求解域 Ω 上积分

$$\mathbf{u}_t - \nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, \quad \text{in } \Omega$$

$$\mathbf{u}_t \cdot \mathbf{v} - (\nabla \cdot \mathbb{T}(\mathbf{u}, p)) \cdot \mathbf{v} = \mathbf{f} \cdot \mathbf{v}, \quad \text{in } \Omega$$

$$\int_{\Omega} \mathbf{u}_t \cdot \mathbf{v} dx dy - \int_{\Omega} (\nabla \cdot \mathbb{T}(\mathbf{u}, p)) \cdot \mathbf{v} dx dy = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \quad (7.6)$$

- (2) 在无散度项乘以函数 $q(x, y)$, 并在计算域上积分

$$\nabla \cdot \mathbf{u} = 0, \quad \text{in } \Omega \rightarrow (\nabla \cdot \mathbf{u}) q = 0, \quad \text{in } \Omega \rightarrow \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \quad (7.7)$$

在上述方程中, $\mathbf{u}(x, y, t)$ 、 $p(x, y, t)$ 为试探函数 (Trial Function), $\mathbf{v}(x, y)$ 、 $q(x, y)$ 为测试函数 (Test Function)。

(3) 根据多元分部积分法（散度定理）对方程等号左边项进行变换

$$\int_{\Omega} (\nabla \cdot \mathbb{T}) \cdot \mathbf{v} dx dy = \int_{\partial\Omega} (\mathbb{T} \mathbf{n}) \cdot \mathbf{v} ds - \int_{\Omega} \mathbb{T} : \nabla \mathbf{v} dx dy \quad (7.8)$$

其中， $\mathbf{n} = (n_1, n_2)^T$ 为计算域边界 $\partial\Omega$ 的单位外法向量 (Unit Outer Normal Vector)，由此可以得到：

$$\int_{\Omega} \mathbf{u}_t \cdot \mathbf{v} dx dy + \int_{\Omega} \mathbb{T}(\mathbf{u}, p) : \nabla \mathbf{v} dx dy - \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \quad (7.9)$$

对于两个张量，其内积的定义为：

$$A : B = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} : \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = a_{12}b_{11} + a_{12}b_{12} + a_{21}b_{21} + a_{22}b_{22} \quad (7.10)$$

使用上述张量内积的定义，可以得到：

$$\mathbb{T}(\mathbf{u}, p) : \nabla \mathbf{v} = (2\nu \mathbb{D}(\mathbf{u}) - p \mathbb{I}) : \nabla \mathbf{v} = 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) - p (\nabla \cdot \mathbf{v}) \quad (7.11)$$

由此可以进一步得到：

$$\begin{aligned} & \int_{\Omega} \mathbf{u}_t \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy \\ & - \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \\ & - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \end{aligned} \quad (7.12)$$

【说明】：这里在式 (7.7) 中乘以 -1 用以保证矩阵对称性。

待求函数在计算域边界 $\partial\Omega$ 处的值由 $\mathbf{u}(x, y, t) = \mathbf{g}(x, y, t)$ 给定，因此，可以选择合适的测试函数 $\mathbf{v}(x, y)$ 使其在边界 $\partial\Omega$ 处满足 $\mathbf{v}(x, y) = 0$ ，此时方程 (7.9) 等号左边第三项为零，可以得到：

$$\begin{aligned} & \int_{\Omega} \mathbf{u}_t \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \\ & - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \end{aligned} \quad (7.13)$$

方程 (7.13) 即为待求问题的弱格式 (Weak Formulation)。该方程的求解就是找到 $\mathbf{u} \in H^1(0, T; [H^1(\Omega)]^2)$ 和 $p \in L^2(0, T; [L^2(\Omega)]^2)$ 使得方程 (7.13) 对任意 $\mathbf{v} \in [H_0^1(\Omega)]^2$ 和 $q \in L^2(\Omega)$ 都成立。其中， $H^1(0, T; [H^1(\Omega)]^2) = \left\{ \mathbf{v}(\cdot, t), \frac{\partial \mathbf{v}}{\partial t}(\cdot, t) \in [H^1(\Omega)]^2, \forall t \in [0, T] \right\}$ ， $L^2(0, T; L^2(\Omega)) = \{q(\cdot, t) \in L^2(\Omega), \forall t \in [0, T]\}$ 。

7.2.3 方程离散

令 $a(\mathbf{u}, \mathbf{v}) = \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy$, $b(\mathbf{u}, q) = - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy$, $(\mathbf{f}, \mathbf{v}) = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy$, 待求问题为找到 $\mathbf{u} \in H^1(0, T; [H^1(\Omega)]^2)$ 和 $p \in L^2(0, T; [L^2(\Omega)]^2)$ 对任意 $\mathbf{v} \in [H_0^1(\Omega)]^2$ 和 $q \in L^2(\Omega)$ 都成立。

$$\begin{aligned} & (\mathbf{u}_t, \mathbf{v}) + a(\mathbf{u}, \mathbf{v}) + b(\mathbf{v}, p) = (\mathbf{f}, \mathbf{v}) \\ & b(\mathbf{u}, q) = 0 \end{aligned} \quad (7.14)$$

根据张量双点积定义可以得到：

$$\begin{aligned} & \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) \\ &= \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \\ \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) & \frac{\partial u_y}{\partial y} \end{bmatrix} : \begin{bmatrix} \frac{\partial v_x}{\partial x} & \frac{1}{2} \left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x} \right) \\ \frac{1}{2} \left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x} \right) & \frac{\partial v_y}{\partial y} \end{bmatrix} \\ &= \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x} \right) + \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} \end{aligned} \quad (7.15)$$

进一步对上式进行简化，可以得到：

$$\begin{aligned} & \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) \\ &= \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + \frac{1}{2} \left(\frac{\partial u_x}{\partial y} \frac{\partial v_x}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_y}{\partial x} + \frac{\partial u_y}{\partial x} \frac{\partial v_x}{\partial y} + \frac{\partial u_y}{\partial x} \frac{\partial v_y}{\partial x} \right) + \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} \end{aligned} \quad (7.16)$$

所以，

$$\begin{aligned} & \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy \\ &= \int_{\Omega} \nu \left(2 \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + 2 \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_x}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_y}{\partial x} + \frac{\partial u_y}{\partial x} \frac{\partial v_x}{\partial y} + \frac{\partial u_y}{\partial x} \frac{\partial v_y}{\partial x} \right) dx dy \end{aligned} \quad (7.17)$$

同理可以得到：

$$\int_{\Omega} \mathbf{u}_t \cdot \mathbf{v} dx dy = \int_{\Omega} \left(\frac{\partial u_x}{\partial t} v_x + \frac{\partial v_y}{\partial t} v_y \right) dx dy \quad (7.18)$$

$$\int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy = \int_{\Omega} p \left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right) dx dy \quad (7.19)$$

$$\int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy = \int_{\Omega} (f_x v_x + f_y v_y) dx dy \quad (7.20)$$

$$\int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = \int_{\Omega} \left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} \right) q dx dy \quad (7.21)$$

所以，非稳态 Stokes 方程弱格式的标量形式为找到 $u_x \in H^1(0, T; [H^1(\Omega)]^2)$ 、 $u_y \in H^1(0, T; [H^1(\Omega)]^2)$ 和 $p \in L^2(0, T; L^2(\Omega))$ ，使得对任意 $v_x \in H_0^1(\Omega)$ 、 $v_y \in H_0^1(\Omega)$ 和 $q \in L^2(\Omega)$ 都成立：

$$\begin{aligned} & \int_{\Omega} \frac{\partial u_x}{\partial t} v_x dx dy + \int_{\Omega} \frac{\partial u_y}{\partial t} v_y dx dy + \int_{\Omega} \nu \left(2 \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + 2 \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_x}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_y}{\partial x} \right. \\ & \left. + \frac{\partial u_y}{\partial x} \frac{\partial v_x}{\partial y} + \frac{\partial u_y}{\partial x} \frac{\partial v_y}{\partial x} \right) dx dy - \int_{\Omega} p \left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right) dx dy = \int_{\Omega} (f_x v_x + f_y v_y) dx dy \\ & - \int_{\Omega} \left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} \right) q dx dy = 0 \end{aligned} \quad (7.22)$$

所以在有限元空间上，对任意 $\mathbf{v}_h \in U_h \times U_h$ 和 $q_h \in W_h$ 有：

$$\begin{aligned} (\mathbf{u}_{ht}, \mathbf{v}) + a(\mathbf{u}_h, \mathbf{v}_h) + b(\mathbf{v}_h, p_h) &= (\mathbf{f}, \mathbf{v}_h) \\ b(\mathbf{u}_h, q_h) &= 0 \end{aligned} \quad (7.23)$$

在采用有限元方法对速度场与压力场进行求解时，基函数不能任意选取，必须满足一些条件，因为动量方程与连续性方程的耦合会产生稳定性问题，即使程序运行无误也将产生无效结果。该问题与“inf-sup”或“Ladyzhenskaya-Babuska-Brezzi, LBB”有关。

$$\inf_{0 \neq q_h \in W_h} \sup_{0 \neq \mathbf{u}_h \in U_h \times U_h} \frac{b(\mathbf{u}_h, q_h)}{\|\nabla \mathbf{u}_h\|_0 \|q_h\|_0} > \beta \quad (7.24)$$

其中 $\beta > 0$ 为一无关单元大小的常数。

为避免出现稳定性问题，在求解时通常采用“Taylor-Hood”或“Hilbert-Schmidt”基函数对分别处理速度与压力，在“Taylor-Hood”基函数对中，速度基函数阶数通常较压力基函数阶数更高，因此，这里选择速度对应有限单元为 2 次元，其对应基函数空间为 $V_h = \{v \in [C^0(\Omega)]^d : v|_k \in [P^2(K)]^d\}$ ，压力对应有限元基函数为线性元，其基函数空间为 $Q_h = \{q \in [C^0(\Omega)]^1 : q|_k \in P^1(K)^1\}$ 。

7.2.4 线性系统构建

(1) 刚度矩阵

将待求函数中未知变量用有限元基函数展开，得到：

$$u_{xh}(x, y, t) = \sum_{j=1}^{N_b} u_{xj}(t) \phi_j, \quad u_{yh}(x, y, t) = \sum_{j=1}^{N_b} u_{yj}(t) \phi_j, \quad p_h(x, y, t) = \sum_{j=1}^{N_{bp}} p_j(t) \psi_j \quad (7.25)$$

选取合适的基函数，得到关于 $u_{xj}(t)$ 、 $u_{yj}(t)$ 和 $p_j(t)$ 的线性系统，然后求解该线性系统即可得到某一时刻有限元解： $\mathbf{u}_h = (u_{xh}, u_{yh})^T$ 和 p_h 。

对于第一个方程，将 $\mathbf{v}_h = (\phi_i, 0)^T$ ($i = 1, \dots, N_b$) 和 $\mathbf{v}_h = (0, \phi_i)^T$ ($i = 1, \dots, N_b$)^T 代入弱格式，即对第一组测试函数取 $v_{xh} = \phi_i$ ($i = 1, \dots, N_b$) 和 $v_{yh} = 0$ ，对第二组测试函数取 $v_{xh} = 0$ 和 $v_{yh} = \phi_i$ ($i = 1, \dots, N_b$)。

对于第二个方程，取 $q_h = \psi_i, 0$ ($i = 1, \dots, N_{bp}$)。

► 令 $\mathbf{v}_h = (\phi_i, 0)^T$ ，即 $v_{xh} = \phi_i$ 和 $v_{yh} = 0$ ($i = 1, \dots, N_b$)，得到：

$$\begin{aligned} & \int_{\Omega} \left(\sum_{j=1}^{N_b} u_{xj}(t) \phi_j \right)_t \phi_i dx dy + 2 \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{xj}(t) \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial x} dx dy \\ & + \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{xj}(t) \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial y} dx dy + \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{yj}(t) \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial y} dx dy \quad (7.26) \\ & - \int_{\Omega} \left(\sum_{j=1}^{N_{bp}} p_j(t) \psi_j \right) \frac{\partial \phi_i}{\partial x} dx dy = \int_{\Omega} f_x \phi_i dx dy \end{aligned}$$

► 令 $\mathbf{v}_h = (0, \phi_i)^T$ ，即 $v_{xh} = 0$ 和 $v_{yh} = \phi_i$ ($i = 1, \dots, N_b$)，得到：

$$\begin{aligned} & \int_{\Omega} \left(\sum_{j=1}^{N_b} u_{yj}(t) \phi_j \right)_t \phi_i dx dy + 2 \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{yj}(t) \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial y} dx dy \\ & + \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{yj}(t) \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{xj}(t) \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial x} dx dy \quad (7.27) \\ & - \int_{\Omega} \left(\sum_{j=1}^{N_{bp}} p_j(t) \psi_j \right) \frac{\partial \phi_i}{\partial y} dx dy = \int_{\Omega} f_y \phi_i dx dy \end{aligned}$$

► 令 $q_h = \psi_i$ ($i = 1, \dots, N_{bp}$)，得到：

$$- \int_{\Omega} \left(\sum_{j=1}^{N_b} u_{xj}(t) \frac{\partial \phi_j}{\partial x} \right) \psi_i dx dy - \int_{\Omega} \left(\sum_{j=1}^{N_b} u_{yj}(t) \frac{\partial \phi_j}{\partial y} \right) \psi_i dx dy = 0 \quad (7.28)$$

将上述三式进行整理后可以得到：

$$\begin{aligned} & \sum_{j=1}^{N_b} u'_{xj}(t) \int_{\Omega} \phi_j \phi_i dx dy + \sum_{j=1}^{N_b} u_{xj}(t) \left(2 \int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy \right) \\ & + \sum_{j=1}^{N_b} u_{yj}(t) \int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial y} dx dy + \sum_{j=1}^{N_{bp}} p_j(t) \left(- \int_{\Omega} \psi_j \frac{\partial \phi_i}{\partial x} dx dy \right) = \int_{\Omega} f_x \phi_i dx dy \quad (7.29) \end{aligned}$$

$$\begin{aligned} & \sum_{j=1}^{N_b} u_{yj}'(t) \int_{\Omega} \phi_j \phi_i dx dy + \sum_{j=1}^{N_b} u_{xj}(t) \int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial x} dx dy + \sum_{j=1}^{N_b} u_{yj}(t) \left(2 \int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy \right. \\ & \left. + \int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy \right) + \sum_{j=1}^{N_{bp}} p_j(t) \left(- \int_{\Omega} \psi_j \frac{\partial \phi_i}{\partial y} dx dy \right) = \int_{\Omega} f_y \phi_i dx dy \end{aligned} \quad (7.30)$$

$$\sum_{j=1}^{N_b} u_{xj}(t) \left(- \int_{\Omega} \frac{\partial \phi_j}{\partial x} \psi_i dx dy \right) + \sum_{j=1}^{N_b} u_{yj}(t) \left(- \int_{\Omega} \frac{\partial \phi_j}{\partial y} \psi_i dx dy \right) + \sum_{j=1}^{N_{bp}} p_j(t) * 0 = 0 \quad (7.31)$$

为了方便描述, 定义如下矩阵组:

$$\begin{aligned} \mathbf{K}_1 &= \left[\int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b}, & \mathbf{K}_2 &= \left[\int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b} \\ \mathbf{K}_3 &= \left[\int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b}, & \mathbf{K}_4 &= \left[\int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b} \\ \mathbf{K}_5 &= \left[\int_{\Omega} -\psi_j \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_{bp}, N_{bp}}, & \mathbf{K}_6 &= \left[\int_{\Omega} -\psi_j \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_{bp}, N_{bp}} \\ \mathbf{K}_7 &= \left[\int_{\Omega} -\frac{\partial \phi_j}{\partial x} \psi_i dx dy \right]_{i,j=1}^{N_{bp}, N_b}, & \mathbf{K}_8 &= \left[\int_{\Omega} -\frac{\partial \phi_j}{\partial y} \psi_i dx dy \right]_{i,j=1}^{N_{bp}, N_b} \end{aligned} \quad (7.32)$$

定义 0 矩阵为 $\mathbb{O}_1 = [0]_{i,j=1}^{N_{bp}, N_{bp}}$, 该矩阵大小为 $N_{bp} \times N_{bp}$, 由此可以得到线性系统矩阵 \mathbf{K} :

$$\mathbf{K} = \begin{bmatrix} 2\mathbf{K}_1 + \mathbf{K}_2 & \mathbf{K}_3 & \mathbf{K}_5 \\ \mathbf{K}_4 & 2\mathbf{K}_2 + \mathbf{K}_1 & \mathbf{K}_6 \\ \mathbf{K}_7 & \mathbf{K}_8 & \mathbb{O}_1 \end{bmatrix} \quad (7.33)$$

其中, $\mathbf{K}_4 = \mathbf{K}_3^T$, $\mathbf{K}_7 = \mathbf{K}_5^T$, $\mathbf{K}_8 = \mathbf{K}_6^T$, 所以该矩阵为一对称矩阵。

- 令算法 6 中 $c = \nu$, $r = 1$, $s = 0$, $p = 1$, $q = 0$, 得到矩阵 \mathbf{K}_1 ;
- 令算法 6 中 $c = \nu$, $r = 0$, $s = 1$, $p = 0$, $q = 1$, 得到矩阵 \mathbf{K}_2 ;
- 令算法 6 中 $c = \nu$, $r = 1$, $s = 0$, $p = 0$, $q = 1$, 得到矩阵 \mathbf{K}_3 ;
- 令算法 6 中 $c = -1$, $r = 0$, $s = 0$, $p = 1$, $q = 0$, 得到矩阵 \mathbf{K}_5 ;
- 令算法 6 中 $c = -1$, $r = 0$, $s = 0$, $p = 0$, $q = 1$, 得到矩阵 \mathbf{K}_6 ;

根据线性系统对上述矩阵组进行整理可以得到:

$$\mathbf{K} = \begin{bmatrix} 2\mathbf{K}_1 + \mathbf{K}_2 & \mathbf{K}_3 & \mathbf{K}_5 \\ \mathbf{K}_3^T & 2\mathbf{K}_2 + \mathbf{K}_1 & \mathbf{K}_6 \\ \mathbf{K}_5^T & \mathbf{K}_6^T & \mathbb{O}_1 \end{bmatrix} \quad (7.34)$$

定义质量矩阵为：

$$\mathbf{M}_e = [m_{ij}]_{i,j=1}^{N_b} = \left[\int_{\Omega} \phi_j \phi_i dx dy \right]_{i,j=1}^{N_b} \quad (7.35)$$

令算法 6 中 $c = 1, r = s = p = q = 0$, 得到矩阵 \mathbf{M}_e 。

定义 0 矩阵为 $\mathbb{O}_2 = [0]_{i,j=1}^{N_b, N_{bp}}$, $\mathbb{O}_3 = [0]_{i,j=1}^{N_b, N_b}$ 由此可以得到质量矩阵 \mathbf{M} :

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_e & \mathbb{O}_3 & \mathbb{O}_2 \\ \mathbb{O}_3 & \mathbf{M}_e & \mathbb{O}_2 \\ \mathbb{O}_2^T & \mathbb{O}_2^T & \mathbb{O}_1 \end{bmatrix} \quad (7.36)$$

(2) 载荷向量

定义载荷向量如下：

$$\mathbf{F}(t) = \begin{bmatrix} \mathbf{F}_x(t) \\ \mathbf{F}_y(t) \\ 0 \end{bmatrix} \longrightarrow \mathbf{F}_x(t) = \left[\int_{\Omega} f_x \phi_i dx dy \right]_{i=1}^{N_b}, \quad \mathbf{F}_y(t) = \left[\int_{\Omega} f_y \phi_i dx dy \right]_{i=1}^{N_b} \quad (7.37)$$

这里 0 矩阵大小为 $N_{bp} \times 1$, 由此可以得到载荷向量阵 b , 载荷向量可以采用算法 16 得到, 详细过程如下:

- ▶ 令算法 16 中 $f = f_x, p = q = 0$, 得到载荷向量阵 \mathbf{F}_x ;
- ▶ 令算法 16 中 $f = f_y, p = q = 0$, 得到载荷向量阵 \mathbf{F}_y ;

(3) 未知向量

定义待求向量如下:

$$\mathbf{u} = \begin{bmatrix} \mathbf{u}_1(t) \\ \mathbf{u}_2(t) \\ \mathbf{u}_3(t) \end{bmatrix} \longrightarrow \mathbf{u}_1(t) = [u_{1j}(t)]_{i=1}^{N_b}, \quad \mathbf{u}_2(t) = [u_{2j}(t)]_{i=1}^{N_b}, \quad \mathbf{u}_3(t) = [p_j(t)]_{i=1}^{N_{bp}} \quad (7.38)$$

(4) 线性系统组成

非稳态 Stokes 方程包含一阶时间导数, 其求解过程可以类比前述抛物型方程, 将其转化为一阶常微分系统 (Ordinary Differential Equations, ODE) 的形式。

$$\mathbf{Mu}'(t) + \mathbf{Ku}(t) = \mathbf{F}(t) \quad (7.39)$$

所以，可以采用第 六 章中的方法，二者区别在于对应矩阵组的不同，即 \mathbf{M} 、 \mathbf{K} 和 \mathbf{F} 不同，参考抛物型方程 ODE 系统，可以得到：

$$\begin{aligned} & \mathbf{M} \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + \theta \mathbf{K} \mathbf{u}^{n+1} + (1 - \theta) \mathbf{K} \mathbf{u}^n = \theta \mathbf{F}(t_{n+1}) + (1 - \theta) \mathbf{F}(t_n) \\ \rightarrow & \left[\frac{\mathbf{M}}{\Delta t} + \theta \mathbf{K} \right] \mathbf{u}^{n+1} = \theta \mathbf{F}(t_{n+1}) + (1 - \theta) \mathbf{F}(t_n) + \left[\frac{\mathbf{M}}{\Delta t} - (1 - \theta) \mathbf{K}(t_n) \right] \mathbf{u}^n \end{aligned} \quad (7.40)$$

由此可以得到非稳态 Stokes 方程的待求线性系统：

$$\tilde{\mathbf{K}} \mathbf{u}^{n+1} = \tilde{\mathbf{F}}^{n+1}(t), \quad n = 0, \dots, N_t - 1 \quad (7.41)$$

其中， $\tilde{\mathbf{K}} = \frac{\mathbf{M}}{\Delta t} + \theta \mathbf{K}(t_{n+1})$ ， $\tilde{\mathbf{F}}^{n+1} = \theta \mathbf{F}(t_{n+1}) + (1 - \theta) \mathbf{F}(t_n) + \left[\frac{\mathbf{M}}{\Delta t} - (1 - \theta) \mathbf{K}(t_n) \right] \mathbf{u}^n$ 。

得到线性系统后根据算法 18 求解非稳态 Stokes 系统即可得到有限元解。

Algorithm 18 : 2D 非稳态 Stokes 方程有限元求解流程

Require: ν , $\mathbf{f} = \mathbf{f}(x, y, t)$, $\mathbf{g} = \mathbf{g}(x, y, t)$

Ensure: $\mathbf{u}(x, y, t)$ 、 $p(x, y, t)$

- 1: 网格剖分，待求函数初始化 \mathbf{u}_0 、 p_0
 - 2: 根据算法 6 计算质量矩阵 \mathbf{M} 和刚度矩阵 \mathbf{K}
 - 3: 设置时间步长 Δt ，时间分割数 N_t
 - 4: **for** $n = 0, \dots, N_t - 1$ **do**
 - 5: $t_{n+1} = (n + 1)\Delta t$, $t_n = n\Delta t$
 - 6: 由算法 16 计算 t_{n+1} 和 t_n 时刻的矩阵 \mathbf{F}_{n+1} 和 \mathbf{F}_n
 - 7: 计算： $\tilde{\mathbf{K}}$ 和 $\tilde{\mathbf{F}}$
 - 8: 边界条件处理
 - 9: 计算等效线性系统 $\tilde{\mathbf{K}} \mathbf{u}^{n+1} = \tilde{\mathbf{F}}^{n+1}(t)$ ，得到 \mathbf{u}^{n+1}
 - 10: **end for**
-

7.2.5 不同边界条件处理

对于第一类边界条件而言，在每次时间迭代中边界条件处理与稳态 Stokes 问题中边界处理方式相同，这里不再重复，现考虑如下混合边界问题：

$$\left\{ \begin{array}{l} \mathbf{u}_t - \nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, \quad \text{in } \Omega \times [0, T] \\ \nabla \cdot \mathbf{u} = 0, \quad \text{in } \Omega \times [0, T] \\ \mathbb{T}(\mathbf{u}, p) \mathbf{n} = \mathbf{p}, \quad \text{on } \Gamma_S \times [0, T] \\ \mathbb{T}(\mathbf{u}, p) \mathbf{n} + r\mathbf{u} = \mathbf{q}, \quad \text{on } \Gamma_R \times [0, T] \\ \mathbf{u} = \mathbf{g}, \quad \text{on } \Gamma_D \times [0, T] \\ \mathbf{u} = \mathbf{u}_0, \quad \text{at } t = 0 \text{ and in } \Omega \end{array} \right. \quad (7.42)$$

其中， $\Gamma_S, \Gamma_R \subset \partial\Omega$ ， $\Gamma_D = \partial/(\Gamma_S \cup \Gamma_R)$ 。

由前述推导可知：

$$\begin{aligned} & \int_{\Omega} \mathbf{u}_t \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy \\ & - \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \\ & - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \end{aligned} \quad (7.43)$$

由于边界 $\Gamma_D = \partial\Omega/(\Gamma_S \cup \Gamma_R)$ 上的解由式 $\mathbf{u} = \mathbf{g}$ 确定，所以，可以选择合适的测试函数使其在边界上满足 $\mathbf{v} = 0$ 。由边界条件可以得到：

$$\begin{aligned} & \int_{\Omega} \mathbf{u}_t \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy \\ & + \int_{\Gamma_R} r\mathbf{u} \cdot \mathbf{v} ds = \int_{\Omega} f \cdot \mathbf{v} dx dy + \int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds + \int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds \\ & - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \end{aligned} \quad (7.44)$$

在求解该系统时，依次按照 Robin 型、Neumann 型、Dirichlet 型边界条件进行处理，始终保证 Dirichlet 边界发挥作用，具体而言为：在 Robin 型对应边界刚度矩阵增加 $\int_{\Gamma_R} r\mathbf{u} \cdot \mathbf{v} ds$ ，载荷向量增加 $\int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds$ ；在 Neumann 型对应边界载荷向量增加 $\int_{\Gamma_N} \mathbf{p} \cdot \mathbf{v} ds$ ；对 Dirichlet 类型边界对应刚度矩阵采用“置 1”法，载荷向量为给定边界条件值。

7.2.6 误差计算

非稳态 Stokes 方程的误差计算方法与稳态 Stokes 误差计算方式一致，可参考第 五 章对应内容。

7.3 非稳态 Navier-Stokes 方程

7.3.1 问题模型

非稳态 Navier-Stokes 方程形式如下：

$$\begin{cases} \mathbf{u}_t + (\mathbf{u} \cdot \nabla) \mathbf{u} - \nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, & \text{in } \Omega \times [0, T] \\ \nabla \cdot \mathbf{u} = 0, & \text{in } \Omega \times [0, T] \\ \mathbf{u} = \mathbf{g}, & \text{on } \partial\Omega \times [0, T] \\ \mathbf{u} = \mathbf{u}_0, \quad p = p_0, & \text{at } t = 0 \text{ and in } \Omega \end{cases} \quad (7.45)$$

其中， Ω 为计算域， $[0, T]$ 为时间范围， $\mathbf{u}(x, y, t) = (u_x, u_y)^T$ 和 $p(x, y, t) = (p_x, p_y)^T$ 分别为待求速度场及压力场， $\mathbf{f}(x, y, t) = (f_x, f_y)^T$ 为给定计算域 $\Omega \times [0, T]$ 上的函数， $\mathbf{g}(x, y, t) = (g_x, g_y)^T$ 为边界 $\partial\Omega \times [0, T]$ 上的函数， $\mathbf{u}_0 = (u_{x0}, u_{y0})^T$ 和 p_0 为 $t = 0$ 时计算域 Ω 中场量。其中非线性对流项 $(\mathbf{u} \cdot \nabla) \mathbf{u}$ 定义如下：

$$(\mathbf{u} \cdot \nabla) \mathbf{u} = \begin{bmatrix} u_x \frac{\partial u_x}{\partial x} + u_y \frac{\partial u_x}{\partial y} \\ u_x \frac{\partial u_y}{\partial x} + u_y \frac{\partial u_y}{\partial y} \end{bmatrix} \quad (7.46)$$

$\mathbb{T}(\mathbf{u}, p)$ 为应力张量，定义如下：

$$\mathbb{T}(\mathbf{u}, p) = 2\nu \mathbb{D}(\mathbf{u}) - p \mathbb{I} \quad (7.47)$$

其中， ν 为粘度， $\mathbb{D}(\mathbf{u})$ 为应变张量，定义如下：

$$\mathbb{D}(\mathbf{u}) = \frac{1}{2} (\nabla \mathbf{u} + \nabla \mathbf{u}^T) \quad (7.48)$$

将速度各分量及 ∇ 算符代入可以得到，应变张量具体形式为：

$$\mathbb{D}(\mathbf{u}) = \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \\ \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) & \frac{\partial u_y}{\partial y} \end{bmatrix} \quad (7.49)$$

将应变张量代入应力张量中可以得到，应力张量具体形式为：

$$\mathbb{T}(\mathbf{u}, p) = \begin{bmatrix} 2\nu \frac{\partial u_x}{\partial x} - p & \nu \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \\ \nu \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) & 2\nu \frac{\partial u_y}{\partial y} - p \end{bmatrix} \quad (7.50)$$

7.3.2 弱格式

- (1) 方程两端同乘以任意测试函数 (Test Function) $\mathbf{v}(x, y) = (v_x, v_y)^T$ 做内积 (Inner Product), 并在整个求解域 Ω 上积分

$$\begin{aligned} \mathbf{u}_t + (\mathbf{u} \cdot \nabla) \mathbf{u} - \nabla \cdot \mathbb{T}(\mathbf{u}, p) &= \mathbf{f}, \quad \text{in } \Omega \\ \rightarrow \mathbf{u}_t \cdot \mathbf{v} + (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} - (\nabla \cdot \mathbb{T}(\mathbf{u}, p)) \cdot \mathbf{v} &= \mathbf{f} \cdot \mathbf{v}, \quad \text{in } \Omega \\ \rightarrow \int_{\Omega} \mathbf{u}_t \cdot \mathbf{v} dx dy + \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy - \int_{\Omega} (\nabla \cdot \mathbb{T}(\mathbf{u}, p)) \cdot \mathbf{v} dx dy &= \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \quad (7.51) \end{aligned}$$

- (2) 在无散度项乘以函数 $q(x, y)$

$$\nabla \cdot \mathbf{u} = 0, \quad \text{in } \Omega \rightarrow (\nabla \cdot \mathbf{u}) q = 0, \quad \text{in } \Omega \rightarrow \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \quad (7.52)$$

在上述方程中, $\mathbf{u}(x, y, t)$ 、 $p(x, y, t)$ 为试探函数 (Trial Function), $\mathbf{v}(x, y)$ 、 $q(x, y)$ 为测试函数 (Test Function)。

- (3) 根据多元分部积分法 (散度定理) 对方程等号左边项进行变换

$$\int_{\Omega} (\nabla \cdot \mathbb{T}) \cdot \mathbf{v} dx dy = \int_{\partial\Omega} (\mathbb{T} \mathbf{n}) \cdot \mathbf{v} ds - \int_{\Omega} \mathbb{T} : \nabla \mathbf{v} dx dy \quad (7.53)$$

其中, $\mathbf{n} = (n_1, n_2)^T$ 为计算域边界 $\partial\Omega$ 的单位外法向量 (Unit Outer Normal Vector), 由此可以得到:

$$\begin{aligned} \int_{\Omega} \mathbf{u}_t \cdot \mathbf{v} dx dy + \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy + \int_{\Omega} \mathbb{T}(\mathbf{u}, p) : \nabla \mathbf{v} dx dy \\ - \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \quad (7.54) \end{aligned}$$

对于两个张量, 其内积的定义为:

$$A : B = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} : \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = a_{12}b_{11} + a_{12}b_{12} + a_{21}b_{21} + a_{22}b_{22} \quad (7.55)$$

使用上述张量内积的定义, 可以得到:

$$\mathbb{T}(\mathbf{u}, p) : \nabla \mathbf{v} = (2\nu \mathbb{D}(\mathbf{u}) - p \mathbb{I}) : \nabla \mathbf{v} = 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) - p (\nabla \cdot \mathbf{v}) \quad (7.56)$$

由此可以进一步得到：

$$\begin{aligned} & \int_{\Omega} \mathbf{u}_t \cdot \mathbf{v} dx dy + \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy \\ & - \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \\ & - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \end{aligned} \quad (7.57)$$

【说明】：这里在式 (7.52) 中乘以 -1 用以保证矩阵对称性。

待求函数在计算域边界 $\partial\Omega$ 处的值由 $\mathbf{u}(x, y, t) = \mathbf{g}(x, y, t)$ 给定，因此可以选择合适的测试函数 $\mathbf{v}(x, y)$ 使其在边界 $\partial\Omega$ 处满足 $\mathbf{v}(x, y) = 0$ ，可以得到：

$$\begin{aligned} & \int_{\Omega} \mathbf{u}_t \cdot \mathbf{v} dx dy + \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \\ & - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \end{aligned} \quad (7.58)$$

方程 (7.58) 即为待求问题的弱格式 (Weak Formulation)。该方程的求解就是找到 $\mathbf{u} \in H^1(0, T; [H^1(\Omega)]^2)$ 和 $p \in L^2(0, T; [L^2(\Omega)]^2)$ 使得方程对任意 $\mathbf{v} \in [H_0^1(\Omega)]^2$ 和 $q \in L^2(\Omega)$ 都成立。其中， $H^1(0, T; [H^1(\Omega)]^2) = \left\{ \mathbf{v}(\cdot, t), \frac{\partial \mathbf{v}}{\partial t}(\cdot, t) \in [H^1(\Omega)]^2, \forall t \in [0, T] \right\}$ ， $L^2(0, T; L^2(\Omega)) = \{q(\cdot, t) \in L^2(\Omega), \forall t \in [0, T]\}$ 。

7.3.3 方程离散

令 $c(\mathbf{w}, \mathbf{u}, \mathbf{v}) = \int_{\Omega} (\mathbf{w} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy$ ， $a(\mathbf{u}, \mathbf{v}) = \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy$ ， $b(\mathbf{u}, q) = - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy$ ， $(\mathbf{f}, \mathbf{v}) = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy$ ，待求问题为： $\mathbf{u} \in H^1(0, T; [H^1(\Omega)]^2)$ 和 $p \in L^2(0, T; [L^2(\Omega)]^2)$ 对任意 $\mathbf{v} \in [H_0^1(\Omega)]^2$ 和 $q \in L^2(\Omega)$ 都成立。

$$\begin{aligned} & (\mathbf{u}_t, \mathbf{v}) + c(\mathbf{u}, \mathbf{u}, \mathbf{v}) + a(\mathbf{u}, \mathbf{v}) + b(\mathbf{v}, p) = (\mathbf{f}, \mathbf{v}) \\ & b(\mathbf{u}, q) = 0 \end{aligned} \quad (7.59)$$

根据张量双点积定义可以得到：

$$\begin{aligned} & \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) \\ &= \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \\ \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) & \frac{\partial u_y}{\partial y} \end{bmatrix} : \begin{bmatrix} \frac{\partial v_x}{\partial x} & \frac{1}{2} \left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x} \right) \\ \frac{1}{2} \left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x} \right) & \frac{\partial v_y}{\partial y} \end{bmatrix} \\ &= \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x} \right) + \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} \end{aligned} \quad (7.60)$$

进一步对上式进行简化，可以得到：

$$\begin{aligned} & \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) \\ &= \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + \frac{1}{2} \left(\frac{\partial u_x}{\partial y} \frac{\partial v_x}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_y}{\partial x} + \frac{\partial u_y}{\partial x} \frac{\partial v_x}{\partial y} + \frac{\partial u_y}{\partial x} \frac{\partial v_y}{\partial x} \right) + \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} \end{aligned} \quad (7.61)$$

所以，

$$\begin{aligned} & \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy \\ &= \int_{\Omega} \nu \left(2 \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + 2 \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_x}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_y}{\partial x} + \frac{\partial u_y}{\partial x} \frac{\partial v_x}{\partial y} + \frac{\partial u_y}{\partial x} \frac{\partial v_y}{\partial x} \right) dx dy \end{aligned} \quad (7.62)$$

同理可以得到：

$$\int_{\Omega} \mathbf{u}_t \cdot \mathbf{v} dx dy = \int_{\Omega} \left(\frac{\partial u_x}{\partial t} v_x + \frac{\partial u_y}{\partial t} v_y \right) dx dy \quad (7.63)$$

$$\int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy = \int_{\Omega} \left(u_x \frac{\partial u_x}{\partial x} v_x + u_y \frac{\partial u_x}{\partial y} v_x + u_x \frac{\partial u_y}{\partial x} v_y + u_y \frac{\partial u_y}{\partial y} v_y \right) dx dy \quad (7.64)$$

$$\int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy = \int_{\Omega} p \left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right) dx dy \quad (7.65)$$

$$\int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy = \int_{\Omega} (f_x v_x + f_y v_y) dx dy \quad (7.66)$$

$$\int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = \int_{\Omega} \left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} \right) q dx dy \quad (7.67)$$

所以 Navier-Stokes 方程弱格式的标量形式为：找到 $u_x \in H^1(\Omega)$ 、 $u_y \in H^1(\Omega)$ 和 $p \in L^2(\Omega)$ ，使得对任意 $v_x \in H_0^1(\Omega)$ 、 $v_y \in H_0^1(\Omega)$ 和 $q \in L^2(\Omega)$ 都成立：

$$\begin{aligned} & \int_{\Omega} \left(\frac{\partial u_x}{\partial t} v_x + \frac{\partial u_y}{\partial t} v_y \right) dx dy + \int_{\Omega} \left(u_x \frac{\partial u_x}{\partial x} v_x + u_y \frac{\partial u_x}{\partial y} v_x + u_x \frac{\partial u_y}{\partial x} v_y + u_y \frac{\partial u_y}{\partial y} v_y \right) dx dy + \\ & \int_{\Omega} \nu \left(2 \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + 2 \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_x}{\partial y} + \frac{\partial u_x}{\partial y} \frac{\partial v_y}{\partial x} + \frac{\partial u_y}{\partial x} \frac{\partial v_x}{\partial y} + \frac{\partial u_y}{\partial x} \frac{\partial v_y}{\partial x} \right) dx dy \\ & - \int_{\Omega} p \left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right) dx dy = \int_{\Omega} (f_x v_x + f_y v_y) dx dy \\ & \quad - \int_{\Omega} \left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} \right) q dx dy = 0 \end{aligned} \quad (7.68)$$

所以在有限元空间上，对任意 $\mathbf{v}_h \in U_h \times U_h$ 和 $q_h \in W_h$ 有：

$$(\mathbf{u}_{ht}, \mathbf{v}) + c(\mathbf{u}_h, \mathbf{u}_h, \mathbf{v}_h) + a(\mathbf{u}_h, \mathbf{v}_h) + b(\mathbf{v}_h, p_h) = (\mathbf{f}, \mathbf{v}_h)$$

$$b(\mathbf{u}_h, q_h) = 0 \quad (7.69)$$

在采用有限元方法对速度场与压力场进行求解时，基函数不能任意选取，必须满足一些条件，因为动量方程与连续性方程的耦合会产生稳定性问题，即使程序运行无误也将产生无效结果。该问题与“inf-sup”或“Ladyzhenskaya-Babuska-Brezzi, LBB”有关。

$$\inf_{0 \neq q_h \in W_h} \sup_{0 \neq \mathbf{u}_h \in U_h \times U_h} \frac{b(\mathbf{u}_h, q_h)}{\|\nabla \mathbf{u}_h\|_0 \|q_h\|_0} > \beta \quad (7.70)$$

其中 $\beta > 0$ 为一无关单元大小的常数。

为避免出现稳定性问题，在求解时通常采用“Taylor-Hood”或“Hilbert-Schmidt”基函数对分别处理速度与压力，在“Taylor-Hood”基函数对中，速度基函数阶数通常较压力基函数阶数更高，因此，这里选择速度对应有限单元为 2 次元，压力对应有限元基函数为线性元。

7.3.4 非稳态项处理

非稳态项可参考第 六 章中述 θ 离散格式进行处理，这里以向后欧拉格式为例进行说明，将式 (7.69) 使用向后欧拉对时间项进行离散可以得到：

$$\begin{aligned} \left(\frac{\mathbf{u}_h^{n+1} - \mathbf{u}_h^n}{\Delta t}, \mathbf{v} \right) + c(\mathbf{u}_h^{n+1}, \mathbf{u}_h^{n+1}, \mathbf{v}_h) + a(\mathbf{u}_h^{n+1}, \mathbf{v}_h) + b(\mathbf{v}_h, p_h^{n+1}) &= (\mathbf{f}(t_{n+1}), \mathbf{v}_h) \\ b(\mathbf{u}_h^{n+1}, q_h) &= 0 \end{aligned} \quad (7.71)$$

其中， $n \in (0, N_t]$ 为时间节点，所以有：

$$\begin{aligned} \int_{\Omega} \frac{\mathbf{u}_h^{n+1} - \mathbf{u}_h^n}{\Delta t} \cdot \mathbf{v} dx dy + \int_{\Omega} (\mathbf{u}_h^{n+1} \cdot \nabla) \mathbf{u}_h^{n+1} \cdot \mathbf{v}_h dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}_h^{n+1}) : \mathbb{D}(\mathbf{v}_h) dx dy \\ - \int_{\Omega} p_h^{n+1} (\nabla \cdot \mathbf{v}_h) dx dy = \int_{\Omega} \mathbf{f}(t_{n+1}) \cdot \mathbf{v}_h dx dy \\ - \int_{\Omega} (\nabla \cdot \mathbf{u}_h^{n+1}) q_h dx dy = 0 \end{aligned} \quad (7.72)$$

进一步可以得到：

$$\begin{aligned} \int_{\Omega} \left(\frac{u_{xh}^{n+1} - u_{xh}^n}{\Delta t} v_{xh} + \frac{u_{yh}^{n+1} - u_{yh}^n}{\Delta t} v_{yh} \right) dx dy + \int_{\Omega} \left(u_{xh}^{n+1} \frac{\partial u_{xh}^{n+1}}{\partial x} v_{xh} + u_{yh}^{n+1} \frac{\partial u_{xh}^{n+1}}{\partial y} v_{xh} \right. \\ \left. + u_{xh}^{n+1} \frac{\partial u_{yh}^{n+1}}{\partial x} v_{yh} + u_{yh}^{n+1} \frac{\partial u_{yh}^{n+1}}{\partial y} v_{yh} \right) dx dy + \int_{\Omega} \nu \left(2 \frac{\partial u_{xh}^{n+1}}{\partial x} \frac{\partial v_{xh}}{\partial x} + 2 \frac{\partial u_{yh}^{n+1}}{\partial y} \frac{\partial v_{yh}}{\partial y} \right. \\ \left. + \frac{\partial u_{xh}^{n+1}}{\partial y} \frac{\partial v_{xh}}{\partial y} + \frac{\partial u_{xh}^{n+1}}{\partial y} \frac{\partial v_{yh}}{\partial x} + \frac{\partial u_{yh}^{n+1}}{\partial x} \frac{\partial v_{xh}}{\partial x} + \frac{\partial u_{yh}^{n+1}}{\partial x} \frac{\partial v_{yh}}{\partial x} \right) dx dy \\ - \int_{\Omega} p_h^{n+1} \left(\frac{\partial v_{xh}}{\partial x} + \frac{\partial v_{yh}}{\partial y} \right) dx dy = \int_{\Omega} (f_x(t_{n+1}) v_{xh} + f_y(t_{n+1}) v_{yh}) dx dy \end{aligned}$$

$$-\int_{\Omega} \left(\frac{\partial u_{xh}^{n+1}}{\partial x} + \frac{\partial u_{yh}^{n+1}}{\partial y} \right) q_h dx dy = 0 \quad (7.73)$$

7.3.5 非线性项处理

在每一步时间迭代中需要非线性对流项进行处理，处理方式采用与稳态 NS 方程中相同的 Newton 迭代法：这里假设第 $n+1$ 时间步初始速度场为 $\mathbf{u}^{n+1,(0)}$ ，初始压力场为 $p^{n+1,(0)}$ ，对上述方程进行变换可以得到：

$$\begin{aligned} & \left(\frac{\mathbf{u}_h^{n+1,(l)} - \mathbf{u}_h^{n,l}}{\Delta t}, \mathbf{v} \right) + c \left(\mathbf{u}_h^{n+1,(l)}, \mathbf{u}_h^{n+1,(l-1)}, \mathbf{v}_h \right) + c \left(\mathbf{u}_h^{n+1,(l-1)}, \mathbf{u}_h^{n+1,(l)}, \mathbf{v}_h \right) \\ & + a(\mathbf{u}_h^{n+1,(l)}, \mathbf{v}_h) + b(\mathbf{v}_h, p_h^{n+1,(l)}) = (\mathbf{f}(t_{n+1}), \mathbf{v}_h) + c \left(\mathbf{u}_h^{n+1,(l-1)}, \mathbf{u}_h^{n+1,(l-1)}, \mathbf{v}_h \right) \\ & b \left(\mathbf{u}_h^{n+1,(l)}, q_h \right) = 0 \end{aligned} \quad (7.74)$$

其中， $l = 1, 2, \dots, L$ 为迭代次数，该式中 $\mathbf{u}^n \in H^1(\Omega) \times H^1(\Omega)$ ， $p^n \in L^2(\Omega)$ ，对任意 $\mathbf{v} \in H_0^1(\Omega) \times H_0^1(\Omega)$ 和 $q \in L^2(\Omega)$ 都成立。

上式矢量形式可表示为：

$$\begin{aligned} & \int_{\Omega} \frac{\mathbf{u}_h^{n+1,(l)} - \mathbf{u}_h^{n,(l)}}{\Delta t} \cdot \mathbf{v} dx dy + \int_{\Omega} (\mathbf{u}_h^{n+1,(l)} \cdot \nabla) \mathbf{u}_h^{n+1,(l-1)} \cdot \mathbf{v}_h dx dy \\ & + \int_{\Omega} (\mathbf{u}_h^{n+1,(l-1)} \cdot \nabla) \mathbf{u}_h^{n+1,(l)} \cdot \mathbf{v}_h dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}_h^{n+1,(l)}) : \mathbb{D}(\mathbf{v}_h) dx dy \\ & - \int_{\Omega} p_h^{n+1,(l)} (\nabla \cdot \mathbf{v}_h) dx dy = \int_{\Omega} \mathbf{f}(t_{n+1}) \cdot \mathbf{v}_h dx dy + \int_{\Omega} (\mathbf{u}_h^{n+1,(l-1)} \cdot \nabla) \mathbf{u}_h^{n+1,(l-1)} \cdot \mathbf{v}_h dx dy \\ & - \int_{\Omega} (\nabla \cdot \mathbf{u}_h^{n+1,(l)}) q_h dx dy = 0 \end{aligned} \quad (7.75)$$

为方便后续计算及程序实现进一步得到方程的标量形式如下：

$$\begin{aligned} & \int_{\Omega} \left(\frac{u_{xh}^{n+1,(l)} - u_{xh}^{n,(l)}}{\Delta t} v_{xh} + \frac{u_{yh}^{n+1,(l)} - u_{yh}^{n,(l)}}{\Delta t} v_{yh} \right) dx dy + \int_{\Omega} \left(u_{xh}^{n+1,(l)} \frac{\partial u_{xh}^{n+1,(l-1)}}{\partial x} v_{xh} \right. \\ & \left. + u_{yh}^{n+1,(l)} \frac{\partial u_{xh}^{n+1,(l-1)}}{\partial y} v_{xh} + u_{xh}^{n+1,(l)} \frac{\partial u_{yh}^{n+1,(l-1)}}{\partial x} v_{yh} + u_{yh}^{n+1,(l)} \frac{\partial u_{yh}^{n+1,(l-1)}}{\partial y} v_{yh} \right) dx dy \\ & + \int_{\Omega} \left(u_{xh}^{n+1,(l-1)} \frac{\partial u_{xh}^{n+1,(l)}}{\partial x} v_{xh} + u_{yh}^{n+1,(l-1)} \frac{\partial u_{xh}^{n+1,(l)}}{\partial y} v_{xh} + u_{xh}^{n+1,(l-1)} \frac{\partial u_{yh}^{n+1,(l)}}{\partial x} v_{yh} \right. \\ & \left. + u_{yh}^{n+1,(l-1)} \frac{\partial u_{yh}^{n+1,(l)}}{\partial y} v_{yh} \right) dx dy + \int_{\Omega} \nu \left(2 \frac{\partial u_{xh}^{n+1,(l)}}{\partial x} \frac{\partial v_{xh}}{\partial x} + 2 \frac{\partial u_{yh}^{n+1,(l)}}{\partial y} \frac{\partial v_{yh}}{\partial y} \right) dx dy \end{aligned}$$

$$\begin{aligned}
& + \frac{\partial u_{xh}^{n+1,(l)}}{\partial y} \frac{\partial v_{xh}}{\partial y} + \frac{\partial u_{xh}^{n+1,(l)}}{\partial y} \frac{\partial v_{yh}}{\partial x} + \frac{\partial u_{yh}^{n+1,(l)}}{\partial x} \frac{\partial v_{xh}}{\partial y} + \frac{\partial u_{yh}^{n+1,(l)}}{\partial x} \frac{\partial v_{yh}}{\partial x} \Big) dx dy \\
& - \int_{\Omega} p_h^{n+1,(l)} \left(\frac{\partial v_{xh}}{\partial x} + \frac{\partial v_{yh}}{\partial y} \right) dx dy = \int_{\Omega} (f_x(t_{n+1}) v_{xh} + f_y(t_{n+1}) v_{yh}) dx dy \\
& + \int_{\Omega} \left(u_{xh}^{n+1,(l-1)} \frac{\partial u_{xh}^{n+1,(l-1)}}{\partial x} v_{xh} + u_{yh}^{n+1,(l-1)} \frac{\partial u_{xh}^{n+1,(l-1)}}{\partial y} v_{xh} + u_{xh}^{n+1,(l-1)} \frac{\partial u_{yh}^{n+1,(l-1)}}{\partial x} v_{yh} \right. \\
& \left. + u_{yh}^{n+1,(l-1)} \frac{\partial u_{yh}^{n+1,(l-1)}}{\partial y} v_{yh} \right) dx dy \\
& - \int_{\Omega} \left(\frac{\partial u_{xh}^{n+1,(l)}}{\partial x} + \frac{\partial u_{yh}^{n+1,(l)}}{\partial y} \right) q_h dx dy = 0
\end{aligned} \tag{7.76}$$

7.3.6 构建线性系统

(1) 刚度矩阵

将待求函数中未知变量用有限元基函数展开, 得到:

$$\begin{aligned}
u_{xh}^{n+1,(l)} &= \sum_{j=1}^{N_b} u_{xj}^{n+1,(l)} \phi_j, \quad u_{xh}^n = \sum_{j=1}^{N_b} u_{xj}^n \phi_j \\
u_{yh}^{n+1,(l)} &= \sum_{j=1}^{N_b} u_{yj}^{n+1,(l)} \phi_j, \quad u_{yh}^n = \sum_{j=1}^{N_b} u_{yj}^n \phi_j \\
p_h^{n+1,(l)} &= \sum_{j=1}^{N_p} p_{xj}^{n+1,(l)} \psi_j, \quad p_h^n = \sum_{j=1}^{N_p} p_{xj}^n \psi_j
\end{aligned} \tag{7.77}$$

选取合适的基函数, 得到关于 $u_{xj}^{n+1,(l)}$ 、 $u_{yj}^{n+1,(l)}$ 和 $p_{xj}^{n+1,(l)}$ 的线性系统, 然后求解该线性系统即可得到 $n+1$ 时刻第 l 次 Newton 迭代的有限元解: $\mathbf{u}_h^{n+1,(l)}$ 和 $\mathbf{p}_h^{n+1,(l)}$ 。

在进行 Newton 迭代时, 对于第一个方程, 将 $\mathbf{v}_h = (\phi_i, 0)^T (i = 1, \dots, N_b)$ 和 $\mathbf{v}_h = (0, \phi_i)^T (i = 1, \dots, N_b)$ 代入弱格式, 即对第一组测试函数取 $v_{xh} = \phi_i (i = 1, \dots, N_b)$ 和 $v_{yh} = 0$, 对第二组测试函数取 $v_{xh} = 0$ 和 $v_{yh} = \phi_i (i = 1, \dots, N_b)$ 。

对于第二个方程, 取 $q_h = \psi_i, 0 (i = 1, \dots, N_{bp})$ 。

► 令 $\mathbf{v}_h = (\phi_i, 0)^T$, 即 $v_{xh} = \phi_i$ 和 $v_{yh} = 0 (i = 1, \dots, N_b)$, 得到:

$$\frac{1}{\Delta t} \int_{\Omega} \left(\sum_{j=1}^{N_b} u_{xj}^{n+1,(l)} \phi_j \right) \phi_i dx dy - \frac{1}{\Delta t} \int_{\Omega} \left(\sum_{j=1}^{N_p} u_{xj}^n \phi_j \right) \phi_i dx dy$$

$$\begin{aligned}
& \int_{\Omega} \frac{\partial u_{xh}^{n+1,(l-1)}}{\partial x} \left(\sum_{j=1}^{N_b} u_{xj}^{n+1,(l)} \phi_j \right) \phi_i dx dy + \int_{\Omega} \frac{\partial u_{xh}^{n+1,(l-1)}}{\partial y} \left(\sum_{j=1}^{N_b} u_{yj}^{n+1,(l)} \phi_j \right) \phi_i dx dy \\
& + \int_{\Omega} u_{xh}^{n+1,(l-1)} \left(\sum_{j=1}^{N_b} u_{xj}^n \frac{\partial \phi_j}{\partial x} \right) \phi_i dx dy + \int_{\Omega} u_{yh}^{n+1,(l)} \left(\sum_{j=1}^{N_b} u_{xj}^{n+1,(l)} \frac{\partial \phi_j}{\partial y} \right) \phi_i dx dy \\
& 2 \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{xj}^{n+1,(l)} \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{xj}^{n+1,(l)} \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial y} dx dy \\
& + \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{yj}^{n+1,(l)} \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial y} dx dy - \int_{\Omega} \left(\sum_{j=1}^{N_{bp}} p_j^{n+1,(l)} \psi_j \right) \frac{\partial \phi_i}{\partial x} dx dy \\
& = \int_{\Omega} f_x \phi_i dx dy + \int_{\Omega} u_{xh}^{n+1,(l-1)} \frac{\partial u_{xh}^{n+1,(l-1)}}{\partial x} \phi_i dx dy + \int_{\Omega} u_{yh}^{n+1,(l-1)} \frac{\partial u_{xh}^{n+1,(l-1)}}{\partial y} \phi_i dx dy
\end{aligned} \tag{7.78}$$

► 令 $\mathbf{v}_h = (0, \phi_i)^T$, 即 $v_{xh} = 0$ 和 $v_{yh} = \phi_i$ ($i = 1, \dots, N_b$), 得到:

$$\begin{aligned}
& \frac{1}{\Delta t} \int_{\Omega} \left(\sum_{j=1}^{N_b} u_{yh}^{n+1,(l)} \phi_j \right) \phi_i dx dy - \frac{1}{\Delta t} \int_{\Omega} \left(\sum_{j=1}^{N_b} u_{yh}^n \phi_j \right) \phi_i dx dy \\
& \int_{\Omega} \frac{\partial u_{yh}^{n+1,(l-1)}}{\partial x} \left(\sum_{j=1}^{N_b} u_{xj}^{n+1,(l)} \phi_j \right) \phi_i dx dy + \int_{\Omega} \frac{\partial u_{yh}^{n+1,(l-1)}}{\partial y} \left(\sum_{j=1}^{N_b} u_{yj}^{n+1,(l)} \phi_j \right) \phi_i dx dy \\
& + \int_{\Omega} u_{xh}^{n+1,(l-1)} \left(\sum_{j=1}^{N_b} u_{xj}^{n+1,(l)} \frac{\partial \phi_j}{\partial x} \right) \phi_i dx dy + \int_{\Omega} u_{yh}^{n+1,(l)} \left(\sum_{j=1}^{N_b} u_{xj}^{n+1,(l)} \frac{\partial \phi_j}{\partial y} \right) \phi_i dx dy \\
& 2 \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{yj}^{n+1,(l)} \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{xj}^{n+1,(l)} \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial x} dx dy \\
& + \int_{\Omega} \nu \left(\sum_{j=1}^{N_b} u_{yj}^{n+1,(l)} \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial x} dx dy - \int_{\Omega} \left(\sum_{j=1}^{N_{bp}} p_j^{n+1,(l)} \psi_j \right) \frac{\partial \phi_i}{\partial y} dx dy \\
& = \int_{\Omega} f_y \phi_i dx dy + \int_{\Omega} u_{xh}^{n+1,(l-1)} \frac{\partial u_{yh}^{n+1,(l-1)}}{\partial x} \phi_i dx dy + \int_{\Omega} u_{yh}^{n+1,(l-1)} \frac{\partial u_{yh}^{n+1,(l-1)}}{\partial y} \phi_i dx dy
\end{aligned} \tag{7.79}$$

► 令 $q_h = \psi_i$ ($i = 1, \dots, N_{bp}$), 得到:

$$-\int_{\Omega} \left(\sum_{j=1}^{N_b} u_{xj}^{n+1,(l)} \frac{\partial \phi_j}{\partial x} \right) \psi_i dx dy - \int_{\Omega} \left(\sum_{j=1}^{N_b} u_{yj}^{n+1,(l)} \frac{\partial \phi_j}{\partial y} \right) \psi_i dx dy = 0 \tag{7.80}$$

将上述三式进行整理后可以得到:

$$\sum_{j=1}^{N_b} u_{xj}^{n+1,(l)} \left(\frac{1}{\Delta t} \int_{\Omega} \phi_j \phi_i dx dy + 2 \int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy \right)$$

$$\begin{aligned}
& + \int_{\Omega} \frac{\partial u_{xh}^{n+1,(l-1)}}{\partial x} \phi_j \phi_i dx dy + \int_{\Omega} u_{xh}^{n+1,(l-1)} \frac{\partial \phi_j}{\partial x} \phi_i dx dy + \int_{\Omega} u_{yh}^{n+1,(l-1)} \frac{\partial \phi_j}{\partial y} \phi_i dx dy \Big) \\
& + \sum_{j=1}^{N_b} u_{yj}^{n+1,(l)} \left(\int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial y} dx dy + \int_{\Omega} \frac{\partial u_{xh}^{n-1}}{\partial y} \phi_j \phi_i dx dy \right) + \sum_{j=1}^{N_bp} p_j^{n+1,(l)} \left(- \int_{\Omega} \psi_j \frac{\partial \phi_i}{\partial x} dx dy \right) \\
& = \int_{\Omega} f_x \phi_i dx dy + \int_{\Omega} u_{xh}^{n+1,(l-1)} \frac{\partial u_{xh}^{n+1,(l-1)}}{\partial x} \phi_i dx dy + \int_{\Omega} u_{yh}^{n-1} \frac{\partial u_{xh}^{n-1}}{\partial y} \phi_i dx dy \\
& + \sum_{j=1}^{N_b} u_{xj}^{n,(l)} \left(\frac{1}{\Delta t} \int_{\Omega} \phi_j \phi_i dx dy \right)
\end{aligned} \tag{7.81}$$

$$\begin{aligned}
& \sum_{j=1}^{N_b} u_{xj}^{n+1,(l)} \left(\int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \frac{\partial u_{yh}^{n+1,(l-1)}}{\partial x} \phi_j \phi_i dx dy \right) \\
& + \sum_{j=1}^{N_b} u_{yj}^{n+1,(l)} \left(\frac{1}{\Delta t} \int_{\Omega} \phi_j \phi_i dx dy + 2 \int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy + \int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy \right. \\
& \left. + \int_{\Omega} \frac{\partial u_{yh}^{n+1,(l-1)}}{\partial y} \phi_j \phi_i dx dy + \int_{\Omega} u_{xh}^{n+1,(l-1)} \frac{\partial \phi_j}{\partial x} \phi_i dx dy + \int_{\Omega} u_{yh}^{n+1,(l-1)} \frac{\partial \phi_j}{\partial y} \phi_i dx dy \right)
\end{aligned} \tag{7.82}$$

$$\begin{aligned}
& + \sum_{j=1}^{N_bp} p_j^{n+1,(l)} \left(- \int_{\Omega} \psi_j \frac{\partial \phi_i}{\partial y} dx dy \right) = \int_{\Omega} f_y \phi_i dx dy + \int_{\Omega} u_{xh}^{n+1,(l-1)} \frac{\partial u_{yh}^{n+1,(l-1)}}{\partial x} \phi_i dx dy
\end{aligned}$$

$$+ \int_{\Omega} u_{yh}^{n-1} \frac{\partial u_{yh}^{n-1}}{\partial y} \phi_i dx dy + \sum_{j=1}^{N_b} u_{yj}^{n,(l)} \left(\frac{1}{\Delta t} \int_{\Omega} \phi_j \phi_i dx dy \right)$$

$$\sum_{j=1}^{N_b} u_{xj}^{n+1,(l)} \left(- \int_{\Omega} \frac{\partial \phi_j}{\partial x} \psi_i dx dy \right) + \sum_{j=1}^{N_b} u_{yj}^{n+1,(l)} \left(- \int_{\Omega} \frac{\partial \phi_j}{\partial y} \psi_i dx dy \right) + \sum_{j=1}^{N_bp} p_j^{n+1,(l)} * 0 = 0 \tag{7.83}$$

为了方便描述, 定义如下矩阵组:

$$\begin{aligned}
\mathbf{K}_1 &= \left[\int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b}, \quad \mathbf{K}_2 = \left[\int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b} \\
\mathbf{K}_3 &= \left[\int_{\Omega} \nu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b}, \quad \mathbf{K}_4 = \left[\int_{\Omega} \nu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b} \\
\mathbf{K}_5 &= \left[\int_{\Omega} -\psi_j \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b, N_bp}, \quad \mathbf{K}_6 = \left[\int_{\Omega} -\psi_j \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b, N_bp} \\
\mathbf{K}_7 &= \left[\int_{\Omega} -\frac{\partial \phi_j}{\partial x} \psi_i dx dy \right]_{i,j=1}^{N_bp, N_b}, \quad \mathbf{K}_8 = \left[\int_{\Omega} -\frac{\partial \phi_j}{\partial y} \psi_i dx dy \right]_{i,j=1}^{N_bp, N_b}
\end{aligned} \tag{7.84}$$

定义 0 矩阵为 $\mathbb{O}_1 = [0]_{i,j=1}^{N_{bp}, N_{bp}}$, 该矩阵大小为 $N_{bp} \times N_{bp}$, 由此可以得到线性系统矩阵 \mathbf{K} :

$$\mathbf{K} = \begin{bmatrix} 2\mathbf{K}_1 + \mathbf{K}_2 & \mathbf{K}_3 & \mathbf{K}_5 \\ \mathbf{K}_4 & 2\mathbf{K}_2 + \mathbf{K}_1 & \mathbf{K}_6 \\ \mathbf{K}_7 & \mathbf{K}_8 & \mathbb{O}_1 \end{bmatrix} \quad (7.85)$$

其中, $\mathbf{K}_4 = \mathbf{K}_3^T$, $\mathbf{K}_7 = \mathbf{K}_5^T$, $\mathbf{K}_8 = \mathbf{K}_6^T$, 所以该矩阵为一对称矩阵。

矩阵组中各个矩阵可通过算法 6 分别得到, 详细过程如下:

- 令算法 6 中 $c = v$, $r = 1$, $s = 0$, $p = 1$, $q = 0$, 得到矩阵 \mathbf{K}_1 ;
- 令算法 6 中 $c = v$, $r = 0$, $s = 1$, $p = 0$, $q = 1$, 得到矩阵 \mathbf{K}_2 ;
- 令算法 6 中 $c = v$, $r = 1$, $s = 0$, $p = 0$, $q = 1$, 得到矩阵 \mathbf{K}_3 ;
- 令算法 6 中 $c = -1$, $r = 0$, $s = 0$, $p = 1$, $q = 0$, 得到矩阵 \mathbf{K}_5 ;
- 令算法 6 中 $c = -1$, $r = 0$, $s = 0$, $p = 0$, $q = 1$, 得到矩阵 \mathbf{K}_6 ;

根据线性系统对上述矩阵组进行整理可以得到:

$$\mathbf{K} = \begin{bmatrix} 2\mathbf{K}_1 + \mathbf{K}_2 & \mathbf{K}_3 & \mathbf{K}_5 \\ \mathbf{K}_3^T & 2\mathbf{K}_2 + \mathbf{K}_1 & \mathbf{K}_6 \\ \mathbf{K}_5^T & \mathbf{K}_6^T & \mathbb{O}_1 \end{bmatrix} \quad (7.86)$$

定义质量矩阵为:

$$\mathbf{M}_e = [m_{ij}]_{i,j=1}^{N_b} = \left[\int_{\Omega} \phi_j \phi_i dx dy \right]_{i,j=1}^{N_b} \quad (7.87)$$

令算法 6 中 $c = 1$, $r = s = p = q = 0$, 得到矩阵 \mathbf{M}_e 。

定义 0 矩阵为 $\mathbb{O}_2 = [0]_{i,j=1}^{N_b, N_{bp}}$, $\mathbb{O}_3 = [0]_{i,j=1}^{N_b, N_b}$ 由此可以得到质量矩阵 \mathbf{M} :

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_e & \mathbb{O}_3 & \mathbb{O}_2 \\ \mathbb{O}_3 & \mathbf{M}_e & \mathbb{O}_2 \\ \mathbb{O}_2^T & \mathbb{O}_2^T & \mathbb{O}_1 \end{bmatrix} \quad (7.88)$$

将非线性项矩阵定义为:

$$\begin{aligned} \mathbf{KN}_1 &= \left[\int_{\Omega} \frac{\partial u_{xh}^{n+1,(l-1)}}{\partial x} \phi_j \phi_i dx dy \right]_{i,j=1}^{N_b}, \quad \mathbf{KN}_2 = \left[\int_{\Omega} u_{xh}^{n+1,(l-1)} \frac{\partial \phi_j}{\partial x} \phi_i dx dy \right]_{i,j=1}^{N_b} \\ \mathbf{KN}_3 &= \left[\int_{\Omega} u_{yh}^{n+1,(l-1)} \frac{\partial \phi_j}{\partial y} \phi_i dx dy \right]_{i,j=1}^{N_b}, \quad \mathbf{KN}_4 = \left[\int_{\Omega} \frac{\partial u_{xh}^{n+1,(l-1)}}{\partial y} \phi_j \phi_i dx dy \right]_{i,j=1}^{N_b} \\ \mathbf{KN}_5 &= \left[\int_{\Omega} \frac{\partial u_{yh}^{n+1,(l-1)}}{\partial x} \phi_j \phi_i dx dy \right]_{i,j=1}^{N_b}, \quad \mathbf{KN}_6 = \left[\int_{\Omega} \frac{\partial u_{yh}^{n+1,(l-1)}}{\partial y} \phi_j \phi_i dx dy \right]_{i,j=1}^{N_b} \end{aligned} \quad (7.89)$$

进一步得到非线性矩阵 \mathbf{KN} :

$$\mathbf{KN} = \begin{bmatrix} \mathbf{KN}_1 + \mathbf{KN}_2 + \mathbf{KN}_3 & \mathbf{KN}_4 & \mathbb{O}_2 \\ \mathbf{KN}_5 & \mathbf{KN}_6 + \mathbf{KN}_2 + \mathbf{KN}_3 & \mathbb{O}_2 \\ \mathbb{O}_2^T & \mathbb{O}_2^T & \mathbb{O}_1 \end{bmatrix} \quad (7.90)$$

- ▶ 令算法 12 中 $c_h = u_{xh}^{l-1}$, $d = 1$, $e = r = s = p = q = 0$, 得到矩阵 \mathbf{KN}_1 ;
- ▶ 令算法 12 中 $c_h = u_{xh}^{l-1}$, $r = 1$, $d = e = s = p = q = 0$, 得到矩阵 \mathbf{KN}_2 ;
- ▶ 令算法 12 中 $c_h = u_{yh}^{l-1}$, $s = 1$, $d = e = r = p = q = 0$, 得到矩阵 \mathbf{KN}_3 ;
- ▶ 令算法 12 中 $c_h = u_{xh}^{l-1}$, $e = 1$, $d = r = s = p = q = 0$, 得到矩阵 \mathbf{KN}_4 ;
- ▶ 令算法 12 中 $c_h = u_{yh}^{l-1}$, $d = 1$, $e = r = s = p = q = 0$, 得到矩阵 \mathbf{KN}_5 ;
- ▶ 令算法 12 中 $c_h = u_{yh}^{l-1}$, $e = 1$, $d = r = s = p = q = 0$, 得到矩阵 \mathbf{KN}_6 ;

(2) 载荷向量

定义载荷向量如下:

$$\mathbf{F} = \begin{bmatrix} \mathbf{F}_x \\ \mathbf{F}_y \\ 0 \end{bmatrix} \longrightarrow \mathbf{F}_x = \left[\int_{\Omega} f_x \phi_i dx dy \right]_{i=1}^{N_b}, \quad \mathbf{F}_y = \left[\int_{\Omega} f_y \phi_i dx dy \right]_{i=1}^{N_b} \quad (7.91)$$

这里 0 矩阵大小为 $N_{bp} \times 1$, 由此可以得到载荷向量阵 \mathbf{F} , 载荷向量可以采用算法 16 得到, 详细过程如下:

- ▶ 令算法 16 中 $f = f_x$, $p = q = 0$, 得到载荷向量阵 \mathbf{F}_x ;
- ▶ 令算法 16 中 $f = f_y$, $p = q = 0$, 得到载荷向量阵 \mathbf{F}_y ;

非线性部分载荷向量定义为:

$$\mathbf{FN} = \begin{bmatrix} \mathbf{FN}_1 + \mathbf{FN}_2 \\ \mathbf{FN}_3 + \mathbf{FN}_4 \\ 0 \end{bmatrix} \quad (7.92)$$

其中, 0 矩阵大小为 $N_{bp} \times 1$, 其余各矩阵定义分别如下:

$$\begin{aligned} \mathbf{FN}_1 &= \left[\int_{\Omega} u_{xh}^{n+1,(l-1)} \frac{\partial u_{xh}^{n+1,(l-1)}}{\partial x} \phi_i dx dy \right]_{i,j=1}^{N_b}, \quad \mathbf{FN}_2 = \left[\int_{\Omega} u_{yh}^{n+1,(l-1)} \frac{\partial u_{xh}^{n+1,(l-1)}}{\partial y} \phi_i dx dy \right]_{i,j=1}^{N_b} \\ \mathbf{FN}_3 &= \left[\int_{\Omega} u_{xh}^{n+1,(l-1)} \frac{\partial u_{yh}^{n+1,(l-1)}}{\partial x} \phi_i dx dy \right]_{i,j=1}^{N_b}, \quad \mathbf{FN}_4 = \left[\int_{\Omega} u_{yh}^{n+1,(l-1)} \frac{\partial u_{yh}^{n+1,(l-1)}}{\partial y} \phi_i dx dy \right]_{i,j=1}^{N_b} \end{aligned} \quad (7.93)$$

- ▶ 令算法 13 中 $f_{1h} = u_{xh}^{l-1}$, $f_{2h} = u_{yh}^{n-1}$, $r = 1$, $d = e = s = p = q = 0$, 得到 \mathbf{FN}_1 ;
- ▶ 令算法 13 中 $f_{1h} = u_{yh}^{l-1}$, $f_{2h} = u_{xh}^{n-1}$, $s = 1$, $d = e = r = p = q = 0$, 得到 \mathbf{FN}_2 ;
- ▶ 令算法 13 中 $f_{1h} = u_{xh}^{l-1}$, $f_{2h} = u_{yh}^{n-1}$, $r = 1$, $d = e = s = p = q = 0$, 得到 \mathbf{FN}_3 ;
- ▶ 令算法 13 中 $f_{1h} = u_{yh}^{l-1}$, $f_{2h} = u_{xh}^{n-1}$, $s = 1$, $d = e = r = p = q = 0$, 得到 \mathbf{FN}_4 ;

(3) 未知向量

定义前一时间步向量如下:

$$\mathbf{u}^n = \begin{bmatrix} \mathbf{u}_1^n \\ \mathbf{u}_2^n \\ \mathbf{u}_3^n \end{bmatrix} \longrightarrow \mathbf{u}_1^n = [u_{1j}^n]_{i=1}^{N_b}, \quad \mathbf{u}_2^n = [u_{2j}^n]_{i=1}^{N_b}, \quad \mathbf{u}_3^n = [p_j^n]_{i=1}^{N_{bp}} \quad (7.94)$$

定义待求解向量如下:

$$\mathbf{u}^{n+1,(l)} = \begin{bmatrix} \mathbf{u}_1^{n+1,(l)} \\ \mathbf{u}_2^{n+1,(l)} \\ \mathbf{u}_3^{n+1,(l)} \end{bmatrix} \rightarrow \mathbf{u}_1^{n+1,(l)} = [u_{1j}^{n+1,(l)}]_{i=1}^{N_b}, \quad \mathbf{u}_2^{n+1,(l)} = [u_{2j}^{n+1,(l)}]_{i=1}^{N_b}, \quad \mathbf{u}_3^{n+1,(l)} = [p_j^{n+1,(l)}]_{i=1}^{N_{bp}} \quad (7.95)$$

(4) 线性系统组成

本问题中包含非稳态时间项和非线性对流项, 因此对应待求线性系统中刚度矩阵、载荷向量分别为:

$$\mathbf{K}^{n+1,(l)} = \frac{\mathbf{M}}{\Delta t} + \mathbf{K} + \mathbf{kN} \quad (7.96)$$

$$\mathbf{F}^{n+1,(l)} = \frac{\mathbf{M}}{\Delta t} \mathbf{u}^n + \mathbf{F} + \mathbf{FN} \quad (7.97)$$

由此可以得到线性系统:

$$\mathbf{K}^{n+1,(l)} \mathbf{u}^{n+1,(l)} = \mathbf{F}^{n+1,(l)} \quad (7.98)$$

由此得到 n 时刻、 l 次 Newton 迭代线性系统, 根据所给问题模型的边界条件进行处理, 然后求解该线性系统即可得到对应时刻及对应 Newton 迭代中有限元解, 增加 Newton 迭代次数至给定计算精度, 然后向前推进时间步, 重复前述计算过程直至计算结束, 该求解过程如算法 19 所示。

Algorithm 19 : 2D 非稳态 NS 方程有限元求解流程

Require: ν , $\mathbf{f} = \mathbf{f}(x, y, t)$, $\mathbf{g} = \mathbf{g}(x, y, t)$

Ensure: $\mathbf{u}(x, y, t)$ 、 $p(x, y, t)$

- 1: 网格剖分, 待求函数初始化 \mathbf{u}_0 、 p_0
- 2: 根据算法 6 计算质量矩阵 \mathbf{M} 和刚度矩阵 \mathbf{K}
- 3: 设置时间步长 Δt , 时间分割数 N_t
- 4: **for** $n = 0, \dots, N_t - 1$ **do**
- 5: $t_{n+1} = (n + 1)\Delta t$, $t_n = n\Delta t$
- 6: 由算法 16 计算载荷向量 \mathbf{F}
- 7: **for** $l = 0, \dots, N_t - 1$ **do**
- 8: 由算法 12 计算矩阵 \mathbf{KN}
- 9: 由算法 13 计算载荷向量 \mathbf{FN}
- 10: 计算: $\mathbf{K}^{n+1,(l)} = \frac{\mathbf{M}}{\Delta t} + \mathbf{K} + \mathbf{kN}$ 和 $\mathbf{F}^{n+1,(l)} = \frac{\mathbf{M}}{\Delta t}\mathbf{u}^n + \mathbf{F} + \mathbf{FN}$
- 11: 边界条件处理
- 12: 计算等效线性系统 $\mathbf{K}^{n+1,(l)}\mathbf{u}^{n+1,(l)} = \mathbf{F}^{n+1,(l)}$, 得到 \mathbf{u}^{n+1}
- 13: **end for**
- 14: $\mathbf{u}^{n+1} = \mathbf{u}^{n+1,(l)}$
- 15: **end for**

7.3.7 不同边界条件处理

考虑包含如下混合边界的非稳态 NS 问题:

$$\begin{cases} \mathbf{u}_t + (\mathbf{u} \cdot \nabla) \mathbf{u} - \nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, & \text{in } \Omega \times [0, T] \\ \nabla \cdot \mathbf{u} = 0, & \text{in } \Omega \times [0, T] \\ \mathbb{T}(\mathbf{u}, p) \mathbf{n} = \mathbf{p}, & \text{on } \Gamma_S \times [0, T] \\ \mathbb{T}(\mathbf{u}, p) \mathbf{n} + r\mathbf{u} = \mathbf{q}, & \text{on } \Gamma_R \times [0, T] \\ \mathbf{u} = \mathbf{g}, & \text{on } \Gamma_D \times [0, T] \\ \mathbf{u} = \mathbf{u}_0, & \text{at } t = 0 \text{ and in } \Omega \end{cases} \quad (7.99)$$

其中, $\Gamma_S, \Gamma_R \subset \partial\Omega$ 且 $\Gamma_D = \partial\Omega / (\Gamma_S \cup \Gamma_R)$ 。

由前述推导可知:

$$\begin{aligned} & \int_{\Omega} \mathbf{u}_t \cdot \mathbf{v} dx dy + \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy \\ & - \int_{\partial\Omega} (\mathbb{T}(\mathbf{u}, p) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \end{aligned}$$

$$-\int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \quad (7.100)$$

由于边界 $\Gamma_D = \partial\Omega / (\Gamma_S \cup \Gamma_R)$ 上的解由式 $\mathbf{u} = \mathbf{g}$ 确定，所以，可以选择合适的测试函数使其在边界上满足 $\mathbf{v} = 0$ ，边界处控制方程满足：

$$\begin{aligned} & \int_{\Omega} \mathbf{u}_t \cdot \mathbf{v} dx dy + \int_{\Omega} (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} dx dy + \int_{\Omega} 2\nu \mathbb{D}(\mathbf{u}) : \mathbb{D}(\mathbf{v}) dx dy - \int_{\Omega} p (\nabla \cdot \mathbf{v}) dx dy \\ & + \int_{\Gamma_R} r \mathbf{u} \cdot \mathbf{v} ds = \int_{\Omega} f \cdot \mathbf{v} dx dy + \int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds + \int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds \\ & - \int_{\Omega} (\nabla \cdot \mathbf{u}) q dx dy = 0 \end{aligned} \quad (7.101)$$

在求解该系统时，依次按照 Robin 型、Neumann 型、Dirichlet 型边界条件进行处理，始终保证 Dirichlet 边界发挥作用，具体而言为：在 Robin 型对应边界刚度矩阵增加 $\int_{\Gamma_R} r \mathbf{u} \cdot \mathbf{v} ds$ ，载荷向量增加 $\int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds$ ；在 Neumann 型对应边界载荷向量增加 $\int_{\Gamma_N} \mathbf{p} \cdot \mathbf{v} ds$ ；对 Dirichlet 类型边界对应刚度矩阵采用“置 1”法，载荷向量为给定边界条件值。

7.3.8 误差计算

非稳态 NS 方程的误差计算方法与稳态 NS 误差计算方式一致，可参考第 五 章对应内容。

7.4 非稳态线弹性方程

非稳态线弹性问题具有如下形式：

$$\begin{cases} \mathbf{u}_{tt} - \nabla \cdot \sigma(\mathbf{u}) = \mathbf{f}, & \text{in } \Omega \times [0, T] \\ \mathbf{u} = \mathbf{g}, & \text{on } \partial\Omega \times [0, T] \\ \mathbf{u} = \mathbf{u}_0, \quad \frac{\partial \mathbf{u}}{\partial t} = \mathbf{u}_{00}, & \text{at } t = 0 \text{ and in } \Omega \end{cases} \quad (7.102)$$

σ 为应力张量，定义如下：

$$\sigma(\mathbf{u}) = \begin{bmatrix} \sigma_{11}(\mathbf{u}) & \sigma_{12}(\mathbf{u}) \\ \sigma_{21}(\mathbf{u}) & \sigma_{22}(\mathbf{u}) \end{bmatrix}, \quad \sigma_{ij}(\mathbf{u}) = \lambda (\nabla \cdot \mathbf{u}) \delta_{ij} + 2\mu \varepsilon_{ij}(\mathbf{u}) \quad (7.103)$$

其中， λ 和 μ 为拉梅常数， $\delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$ ， ε_{ij} 为应变张量，定义如下：

$$\varepsilon = \begin{bmatrix} \varepsilon_{11}(\mathbf{u}) & \varepsilon_{12}(\mathbf{u}) \\ \varepsilon_{21}(\mathbf{u}) & \varepsilon_{22}(\mathbf{u}) \end{bmatrix}, \quad \varepsilon_{ij}(\mathbf{u}) = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad (7.104)$$

由此可以得到应力张量的具体形式为：

$$\sigma(\mathbf{u}) = \begin{bmatrix} \lambda \frac{\partial u_x}{\partial x} + 2\mu \frac{\partial u_x}{\partial x} + \lambda \frac{\partial u_y}{\partial y} & \mu \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \\ \mu \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) & \lambda \frac{\partial u_x}{\partial x} + \lambda \frac{\partial u_y}{\partial y} + 2\mu \frac{\partial u_y}{\partial y} \end{bmatrix} \quad (7.105)$$

7.4.1 弱格式

(1) 在方程两端同乘以任意的测试函数 (Test Function) $\mathbf{v}(x, y) = (v_x, v_y)^T$ 做内积 (Inner Product)，并在整个求解域 Ω 上积分

$$\begin{aligned} \mathbf{u}_{tt} - \nabla \cdot \sigma(\mathbf{u}) &= \mathbf{f}, \quad (x, y) \in \Omega \\ \rightarrow \mathbf{u}_{tt} \cdot \mathbf{v} - (\nabla \cdot \sigma(\mathbf{u})) \cdot \mathbf{v} &= \mathbf{f} \cdot \mathbf{v}, \quad (x, y) \in \Omega \\ \rightarrow \int_{\Omega} \mathbf{u}_{tt} \cdot \mathbf{v} dx dy - \int_{\Omega} (\nabla \cdot \sigma(\mathbf{u})) \cdot \mathbf{v} dx dy &= \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \end{aligned} \quad (7.106)$$

(2) 根据多元分部积分法 (散度定理) 对方程等号左边项进行变换

$$\int_{\Omega} (\nabla \cdot \sigma(\mathbf{u})) \cdot \mathbf{v} dx dy = \int_{\partial\Omega} (\sigma(\mathbf{u}) \mathbf{n}) \cdot \mathbf{v} ds - \int_{\Omega} \sigma(\mathbf{u}) : \nabla \mathbf{v} dx dy \quad (7.107)$$

其中， $\mathbf{n} = (n_1, n_2)^T$ 为计算域边界 $\partial\Omega$ 的单位外法向量 (Unit Outer Normal Vector)，由此可以得到：

$$\int_{\Omega} \mathbf{u}_{tt} \cdot \mathbf{v} dx dy + \int_{\Omega} \sigma(\mathbf{u}) : \nabla \mathbf{v} dx dy - \int_{\partial\Omega} (\sigma(\mathbf{u}) \mathbf{n}) \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \quad (7.108)$$

对于两个张量，其内积的定义为：

$$A : B = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} : \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = a_{12}b_{11} + a_{12}b_{12} + a_{21}b_{21} + a_{22}b_{22} \quad (7.109)$$

且有：

$$\nabla \mathbf{v} = \begin{bmatrix} \frac{\partial v_x}{\partial x} & \frac{\partial v_x}{\partial y} \\ \frac{\partial v_y}{\partial x} & \frac{\partial v_y}{\partial y} \end{bmatrix} \quad (7.110)$$

待求函数在计算域边界 $\partial\Omega$ 处的值由 $\mathbf{u}(x, y, t) = \mathbf{g}(x, y, t)$ 给定，因此可以选择合适的测试函数 $\mathbf{v}(x, y)$ 使其在边界 $\partial\Omega$ 处满足 $\mathbf{v}(x, y) = 0$ ，可以得到：

$$\int_{\Omega} \mathbf{u}_{tt} \cdot \mathbf{v} dx dy + \int_{\Omega} \sigma(\mathbf{u}) : \nabla \mathbf{v} dx dy = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy \quad (7.111)$$

方程 (7.111) 即为待求问题的弱格式 (Weak Formulation)。

$$\begin{aligned}
 \sigma(\mathbf{u}) : \nabla \mathbf{v} &= \begin{bmatrix} \sigma_{11}(\mathbf{u}) & \sigma_{12}(\mathbf{u}) \\ \sigma_{21}(\mathbf{u}) & \sigma_{22}(\mathbf{u}) \end{bmatrix} : \begin{bmatrix} \frac{\partial v_x}{\partial x} & \frac{\partial v_x}{\partial y} \\ \frac{\partial v_y}{\partial x} & \frac{\partial v_y}{\partial y} \end{bmatrix} \\
 &= \sigma_{11}(\mathbf{u}) \frac{\partial v_x}{\partial x} + \sigma_{12}(\mathbf{u}) \frac{\partial v_x}{\partial y} + \sigma_{21}(\mathbf{u}) \frac{\partial v_y}{\partial x} + \sigma_{22}(\mathbf{u}) \frac{\partial v_y}{\partial y} \\
 &= \left(\lambda \frac{\partial u_x}{\partial x} + 2\mu \frac{\partial u_x}{\partial x} + \lambda \frac{\partial u_y}{\partial y} \right) \frac{\partial v_x}{\partial x} \\
 &\quad + \left(\mu \frac{\partial u_x}{\partial y} + \mu \frac{\partial u_y}{\partial x} \right) \frac{\partial v_x}{\partial y} + \left(\mu \frac{\partial u_x}{\partial y} + \mu \frac{\partial u_y}{\partial x} \right) \frac{\partial v_y}{\partial x} \\
 &\quad + \left(\lambda \frac{\partial u_x}{\partial x} + \lambda \frac{\partial u_y}{\partial y} + 2\mu \frac{\partial u_y}{\partial y} \right) \frac{\partial v_y}{\partial y}
 \end{aligned} \tag{7.112}$$

所以可以得到方程等号左边为：

$$\begin{aligned}
 &\int_{\Omega} \sigma(\mathbf{u}) : \nabla \mathbf{v} dx dy \\
 &= \int_{\Omega} \left(\lambda \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + 2\mu \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial y} + \lambda \frac{\partial u_y}{\partial y} \frac{\partial v_x}{\partial x} \right. \\
 &\quad \left. + \mu \frac{\partial u_x}{\partial y} \frac{\partial v_x}{\partial y} + \mu \frac{\partial u_y}{\partial x} \frac{\partial v_x}{\partial y} + \mu \frac{\partial u_x}{\partial y} \frac{\partial v_y}{\partial x} + \mu \frac{\partial u_y}{\partial x} \frac{\partial v_y}{\partial x} \right. \\
 &\quad \left. + \lambda \frac{\partial u_x}{\partial x} \frac{\partial v_y}{\partial y} + \lambda \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} + 2\mu \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} \right) dx dy
 \end{aligned} \tag{7.113}$$

同理可以得到方程等号右边为：

$$\int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy = \int_{\Omega} (f_x v_x + f_y v_y) dx dy \tag{7.114}$$

$$\int_{\Omega} \mathbf{u}_{tt} \cdot \mathbf{v} dx dy = \int_{\Omega} \left(\frac{\partial^2 u_x}{\partial t^2} v_x + \frac{\partial^2 u_y}{\partial t^2} v_y \right) dx dy \tag{7.115}$$

所以可以得到方程 (7.111) 的标量形式：

$$\begin{aligned}
 &\int_{\Omega} \left(\frac{\partial^2 u_x}{\partial t^2} v_x + \frac{\partial^2 u_y}{\partial t^2} v_y \right) dx dy + \int_{\Omega} \left(\lambda \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial x} + 2\mu \frac{\partial u_x}{\partial x} \frac{\partial v_x}{\partial y} + \lambda \frac{\partial u_y}{\partial y} \frac{\partial v_x}{\partial x} \right. \\
 &\quad \left. + \mu \frac{\partial u_x}{\partial y} \frac{\partial v_x}{\partial y} + \mu \frac{\partial u_y}{\partial x} \frac{\partial v_x}{\partial y} + \mu \frac{\partial u_x}{\partial y} \frac{\partial v_y}{\partial x} + \mu \frac{\partial u_y}{\partial x} \frac{\partial v_y}{\partial x} \right. \\
 &\quad \left. + \lambda \frac{\partial u_x}{\partial x} \frac{\partial v_y}{\partial y} + \lambda \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} + 2\mu \frac{\partial u_y}{\partial y} \frac{\partial v_y}{\partial y} \right) dx dy = \int_{\Omega} (f_x v_x + f_y v_y) dx dy
 \end{aligned} \tag{7.116}$$

其中， $u_x, u_y \in H^2(0, T; H^1(\Omega))$ 对任意 $v_x, v_y \in H_0^1(\Omega) \times H_0^1(\Omega)$ 成立。

7.4.2 方程离散

令 $a(\mathbf{u}, \mathbf{v}) = \int_{\Omega} \sigma(\mathbf{u}) : \nabla \mathbf{v} dx dy$, $(\mathbf{f}, \mathbf{v}) = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy$, 待求问题为找到 $\mathbf{u} \in H^1(\Omega) \times H^1(\Omega)$ 对任意 $\mathbf{v} \in H_0^1(\Omega) \times H_0^1(\Omega)$ 都成立:

$$(\mathbf{u}_{tt}, \mathbf{v}) + a(\mathbf{u}, \mathbf{v}) = (\mathbf{f}, \mathbf{v}) \quad (7.117)$$

所以在有限元空间上, 对任意 $\mathbf{v}_h \in U_h \times U_h$ 有:

$$\begin{aligned} & (\mathbf{u}_{htt}, \mathbf{v}) + a(\mathbf{u}_h, \mathbf{v}_h) = (\mathbf{f}, \mathbf{v}_h) \\ \longleftrightarrow & \int_{\Omega} \sigma(\mathbf{u}_h) : \nabla \mathbf{v}_h dx dy = \int_{\Omega} \mathbf{f} \cdot \mathbf{v}_h dx dy \end{aligned} \quad (7.118)$$

7.4.3 线性系统构建

(1) 刚度矩阵组装

将待求函数中未知变量用有限元基函数展开, 得到:

$$u_{xh}(x, y, t) = \sum_{i=1}^{N_b} u_{xj}(t) \phi_j, \quad u_{yh}(x, y, t) = \sum_{j=1}^{N_b} u_{yj}(t) \phi_j \quad (7.119)$$

选取合适的基函数, 得到关于 $u_{xj}(t)$ 和 $u_{yj}(t)$ 的线性系统, 然后求解该线性系统即可得到有限元解: $\mathbf{u}_h = (u_{xh}, u_{yh})^T$ 。

在选择基函数时, 分别对所得弱格式的两个分量进行处理: 分别将 $\mathbf{v}_h = (\phi_i, 0)^T$ ($i = 1, \dots, N_b$) 和 $\mathbf{v}_h = (0, \phi_i)^T$ ($i = 1, \dots, N_b$) 代入弱格式, 即对第一组测试函数取 $v_{xh} = \phi_i$ ($i = 1, \dots, N_b$) 和 $v_{yh} = 0$, 对第二组测试函数取 $v_{xh} = 0$ 和 $v_{yh} = \phi_i$ ($i = 1, \dots, N_b$), 两组测试函数对应弱格式如下:

► 令 $\mathbf{v}_h = (\phi_i, 0)^T$, 即 $v_{xh} = \phi_i$ 和 $v_{yh} = 0$ ($i = 1, \dots, N_b$), 得到:

$$\begin{aligned} & \int_{\Omega} \left(\sum_{j=1}^{N_b} u_{xj}(t) \phi_j \right)_{tt} \phi_i dx dy + \int_{\Omega} \lambda \left(\sum_{j=1}^{N_b} u_{xj} \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial x} dx dy \\ & + 2 \int_{\Omega} \mu \left(\sum_{j=1}^{N_b} u_{xj} \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \lambda \left(\sum_{j=1}^{N_b} u_{yj} \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial x} dx dy \\ & + \int_{\Omega} \mu \left(\sum_{j=1}^{N_b} u_{xj} \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial y} dx dy + \int_{\Omega} \mu \left(\sum_{j=1}^{N_b} u_{yj} \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial y} dx dy = \int_{\Omega} f_x \phi_i dx dy \end{aligned} \quad (7.120)$$

► 令 $\mathbf{v}_h = (0, \phi_i)^T$, 即 $v_{xh} = 0$ 和 $v_{yh} = \phi_i (i = 1, \dots, N_b)$, 得到:

$$\begin{aligned} & \int_{\Omega} \left(\sum_{j=1}^{N_b} u_{yj}(t) \phi_j \right)_{tt} \phi_i dx dy + \int_{\Omega} \mu \left(\sum_{j=1}^{N_b} u_{xj} \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial x} dx dy \\ & + \int_{\Omega} \mu \left(\sum_{j=1}^{N_b} u_{yj} \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \lambda \left(\sum_{j=1}^{N_b} u_{xj} \frac{\partial \phi_j}{\partial x} \right) \frac{\partial \phi_i}{\partial y} dx dy \\ & + \int_{\Omega} \lambda \left(\sum_{j=1}^{N_b} u_{yj} \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial y} dx dy + 2 \int_{\Omega} \mu \left(\sum_{j=1}^{N_b} u_{yj} \frac{\partial \phi_j}{\partial y} \right) \frac{\partial \phi_i}{\partial y} dx dy = \int_{\Omega} f_y \phi_i dx dy \end{aligned} \quad (7.121)$$

将上述两式进行整理后可以得到:

$$\begin{aligned} & \sum_{j=1}^{N_b} u''_{xj}(t) \int_{\Omega} \phi_j \phi_i dx dy + \sum_{j=1}^{N_b} u_{xj} \left(\int_{\Omega} \lambda \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy + 2 \int_{\Omega} \mu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy \right. \\ & \left. + \int_{\Omega} \mu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy \right) + \sum_{j=1}^{N_b} u_{yj} \left(\int_{\Omega} \lambda \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial x} dx dy + \int_{\Omega} \mu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial y} dx dy \right) \\ & = \int_{\Omega} f_x \phi_i dx dy \end{aligned} \quad (7.122)$$

$$\begin{aligned} & \sum_{j=1}^{N_b} u''_{yj}(t) \int_{\Omega} \phi_j \phi_i dx dy + \sum_{j=1}^{N_b} u_{xj} \left(\int_{\Omega} \lambda \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial y} dx dy + \int_{\Omega} \mu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial x} dx dy \right) \\ & \sum_{j=1}^{N_b} u_{yj} \left(\int_{\Omega} \lambda \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy + 2 \int_{\Omega} \mu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy + \int_{\Omega} \mu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy \right) \\ & = \int_{\Omega} f_y \phi_i dx dy \end{aligned} \quad (7.123)$$

为了方便描述, 定义如下矩阵组:

$$\begin{aligned} \mathbf{K}_1 &= \left[\int_{\Omega} \lambda \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b}, \quad \mathbf{K}_2 = \left[\int_{\Omega} \mu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b} \\ \mathbf{K}_3 &= \left[\int_{\Omega} \mu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b}, \quad \mathbf{K}_4 = \left[\int_{\Omega} \lambda \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b} \\ \mathbf{K}_5 &= \left[\int_{\Omega} \mu \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b}, \quad \mathbf{K}_6 = \left[\int_{\Omega} \lambda \frac{\partial \phi_j}{\partial x} \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b} \\ \mathbf{K}_7 &= \left[\int_{\Omega} \mu \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial x} dx dy \right]_{i,j=1}^{N_b}, \quad \mathbf{K}_8 = \left[\int_{\Omega} \lambda \frac{\partial \phi_j}{\partial y} \frac{\partial \phi_i}{\partial y} dx dy \right]_{i,j=1}^{N_b} \end{aligned} \quad (7.124)$$

► 令算法 6 中 $c = \lambda$, $r = 1$, $s = 0$, $p = 1$, $q = 0$, 得到矩阵 \mathbf{K}_1 ;

- 令算法 6 中 $c = \mu, r = 1, s = 0, p = 1, q = 0$, 得到矩阵 \mathbf{K}_2 ;
- 令算法 6 中 $c = \mu, r = 0, s = 1, p = 0, q = 1$, 得到矩阵 \mathbf{K}_3 ;
- 令算法 6 中 $c = \lambda, r = 0, s = 1, p = 1, q = 0$, 得到矩阵 \mathbf{K}_4 ;
- 令算法 6 中 $c = \mu, r = 1, s = 0, p = 0, q = 1$, 得到矩阵 \mathbf{K}_5 ;
- 令算法 6 中 $c = \lambda, r = 1, s = 0, p = 0, q = 1$, 得到矩阵 \mathbf{K}_6 ;
- 令算法 6 中 $c = \mu, r = 0, s = 1, p = 1, q = 0$, 得到矩阵 \mathbf{K}_7 ;
- 令算法 6 中 $c = \lambda, r = 0, s = 1, p = 0, q = 1$, 得到矩阵 \mathbf{K}_8 ;

根据线性系统对上述矩阵组进行整理可以得到:

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_1 + 2\mathbf{K}_2 + \mathbf{K}_3 & \mathbf{K}_4 + \mathbf{K}_5 \\ \mathbf{K}_6 + \mathbf{K}_7 & \mathbf{K}_8 + 2\mathbf{K}_3 + \mathbf{K}_2 \end{bmatrix} \quad (7.125)$$

定义基本质量矩阵为:

$$\mathbf{M}_e = [m_{ij}]_{i,j=1}^{N_b} = \left[\int_{\Omega} \phi_i \phi_j dx dy \right]_{i,j=1}^{N_b} \quad (7.126)$$

- 令算法 6 中 $c = 1, r = s = p = q = 0$, 得到矩阵 \mathbf{M} ;

定义 0 矩阵 $\mathbb{O}_4 = [0]_{i,j=1}^{N_b}$, 由此得到总质量矩阵为:

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_e & \mathbb{O}_4 \\ \mathbb{O}_4 & \mathbf{M}_e \end{bmatrix} \quad (7.127)$$

(2) 载荷向量组装

定义载荷向量如下:

$$\mathbf{F}(t) = \begin{bmatrix} \mathbf{F}_x(t) \\ \mathbf{F}_y(t) \end{bmatrix} \longrightarrow \mathbf{F}_x(t) = \left[\int_{\Omega} f_x \phi_i dx dy \right]_{i=1}^{N_b}, \quad \mathbf{F}_y(t) = \left[\int_{\Omega} f_y \phi_i dx dy \right]_{i=1}^{N_b} \quad (7.128)$$

- 令算法 16 中 $f = f_x, p = q = 0$, 得到载荷向量阵 \mathbf{F}_x ;
- 令算法 16 中 $f = f_y, p = q = 0$, 得到载荷向量阵 \mathbf{F}_y ;

(3) 未知向量

定义待求向量如下:

$$\mathbf{u}(t) = \begin{bmatrix} \mathbf{u}_x(t) \\ \mathbf{u}_y(t) \end{bmatrix} \longrightarrow \mathbf{u}_x(t) = [u_{xi}(t)]_{i=1}^{N_b}, \quad u_y(t) = [\mathbf{u}_{yi}(t)]_{i=1}^{N_b} \quad (7.129)$$

(4) 线性系统组成

本问题为二阶常微分方程，其待求线性系统可参考第 六 章双曲型方程进行构造，这里直接给出对应线性系统：

$$\tilde{\mathbf{K}}\mathbf{u}^{n+1} = \tilde{\mathbf{F}}^{n+1}, \quad n = 1, \dots, N_t \quad (7.130)$$

其中， $\tilde{\mathbf{K}} = \frac{\mathbf{M}}{\Delta t^2} + \frac{\mathbf{K}}{4}$ ， $\tilde{\mathbf{F}}^{n+1} = \left[\frac{2\mathbf{M}}{\Delta t^2} - \frac{\mathbf{K}}{2} \right] \mathbf{u}^n - \left[\frac{\mathbf{M}}{\Delta t^2} + \frac{\mathbf{K}}{4} \right] \mathbf{u}^{n-1} + \mathbf{F}^n$ 。

以上为某一时间步内非稳态线弹性方程线性系统构造过程，之后根据给定问题边界条件对刚度矩阵及载荷向量进行处理并求解，即可得到对应时间步中各节点值，随后对下一时间步重复上述过程至给定仿真时间，该仿真过程如算法 20 所示。

Algorithm 20 : 2D 非稳态弹性方程有限元求解算法流程

Require: $\lambda, \mu, f = f(x, y, t), g = g(x, y, t)$

Ensure: $\mathbf{u}(x, y, t)$

- 1: 网格剖分，待求函数初始化 $\mathbf{u}_0, \mathbf{u}_1$
 - 2: 根据算法 15 计算质量矩阵 \mathbf{M} 和刚度矩阵 \mathbf{K}
 - 3: 设置时间步长 Δt ，时间分割数 N_t
 - 4: **for** $n = 0, \dots, N_t - 1$ **do**
 - 5: $t_n = n\Delta t$
 - 6: 由算法 16 计算 t_n 时刻载荷向量 \mathbf{F}^n
 - 7: 边界条件处理
 - 8: 计算等效线性系统 $\tilde{\mathbf{K}}\mathbf{u}^{n+1} = \tilde{\mathbf{F}}(t)$ ，得到 \mathbf{u}^{n+1}
 - 9: **end for**
-

7.4.4 不同边界条件处理

考虑包含如下混合边界条件的非稳态线弹性方程：

$$\begin{cases} \mathbf{u}_{tt} - \nabla \cdot \sigma(\mathbf{u}) = \mathbf{f}, & \text{in } \Omega \times [0, T] \\ \sigma(\mathbf{u}) \mathbf{n} = \mathbf{p}, & \text{on } \Gamma_S \times [0, T] \\ \sigma(\mathbf{u}) \mathbf{n} + r\mathbf{u} = \mathbf{q}, & \text{on } \Gamma_R \times [0, T] \\ \mathbf{u} = \mathbf{g}, & \text{on } \Gamma_D \times [0, T] \\ \mathbf{u} = \mathbf{u}_0, \quad \frac{\partial \mathbf{u}}{\partial t} = \mathbf{u}_{00}, & \text{at } t = 0 \text{ and in } \Omega \end{cases} \quad (7.131)$$

其中， $\Gamma_S, \Gamma_R \subset \partial\Omega$ 且 $\Gamma_D = \partial\Omega / (\Gamma_S \cup \Gamma_R)$ 。

由于边界 $\Gamma_D = \partial\Omega / (\Gamma_S \cup \Gamma_R)$ 上的解由式 $\mathbf{u} = \mathbf{g}$ 确定，所以，可以选择合适的测试函数使其在该边界上满足 $\mathbf{v} = 0$ ，可以得到：

$$\int_{\Omega} \mathbf{u}_{tt} \cdot \mathbf{v} dx dy + \int_{\Omega} \sigma(\mathbf{u}) : \nabla \mathbf{v} dx dy + \int_{\Gamma_R} r \mathbf{u} \cdot \mathbf{v} ds = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dx dy + \int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds + \int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds \quad (7.132)$$

在求解该系统时，依次按照 Robin 型、Neumann 型、Dirichlet 型边界条件进行处理，始终保证 Dirichlet 边界发挥作用，具体而言为：在 Robin 型对应边界刚度矩阵增加 $\int_{\Gamma_R} r \mathbf{u} \cdot \mathbf{v} ds$ ，载荷向量增加 $\int_{\Gamma_R} \mathbf{q} \cdot \mathbf{v} ds$ ；在 Neumann 型对应边界载荷向量增加 $\int_{\Gamma_S} \mathbf{p} \cdot \mathbf{v} ds$ ；对 Dirichlet 类型边界对应刚度矩阵采用“置 1”法，载荷向量为给定边界条件值。

7.4.5 误差计算

非稳态线弹性误差计算方法与稳态线弹性计算方法相似，详细过程可参考第 四 章。

7.5 应用实例及程序实现

7.5.1 Example 1

例 7.1 【问题描述】以下述 2D 非稳态 Stokes 方程为例对前述有限元方法进行验证，模型计算域为： $\Omega = [0, 1] \times [-0.25, 0]$ ，取 $\nu = 1$ 。

$$\left\{ \begin{array}{l} \mathbf{u}_t - \nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, \quad (x, y) \in \Omega \times [0, 1] \\ \nabla \cdot \mathbf{u} = 0, \quad (x, y) \in \Omega \times [0, 1] \\ u_x = x^2 y^2 + e^{-y}, \quad \text{at } t = 0 \text{ and in } \Omega \\ u_y = -\frac{2}{3} x y^3 + 2 - \pi \sin(\pi x), \quad \text{at } t = 0 \text{ and in } \Omega \\ p = -[2 - \pi \sin(\pi x)] \cos(2\pi y), \quad \text{at } t = 0 \text{ and in } \Omega \\ u_x(0, y) = e^{-y} \cos(2\pi t), \quad u_x(1, y) = (y^2 + e^{-y}) \cos(2\pi t) \\ u_x(x, -0.25) = \left(\frac{1}{16} x^2 + e^{0.25} \right) \cos(2\pi t), \quad u_x(x, 0) = \cos(2\pi t) \\ u_y(0, y) = 2 \cos(2\pi t), \quad u_y(1, y) = \left(-\frac{2}{3} y^3 + 2 \right) \cos(2\pi t) \\ u_y(x, -0.25) = \left(\frac{1}{96} x^2 + 2 - \pi \sin(\pi x) \right) \cos(2\pi t), \quad u_y(x, 0) = [2 - \pi \sin(\pi x)] \cos(2\pi t) \end{array} \right.$$

其中：

$$\begin{aligned} f_x &= -2\pi(x^2 y^2 + e^{-y}) \sin(2\pi t) + [-2\nu x^2 - 2\nu y^2 - \nu e^{-y} + \pi^2 \cos(\pi x) \cos(2\pi y)] \cos(2\pi t) \\ f_y &= -2\pi \left[-\frac{2}{3} x y^3 + 2 - \pi \sin(\pi x) \right] \sin(2\pi t) + 4\nu x y - \nu \pi^3 \sin(\pi x) + 2\pi(2 - \pi \sin(\pi x)) \sin(2\pi y) \end{aligned}$$

【说明】 上述问题的解析解为：

$$\begin{cases} u_x = (x^2y^2 + e^{-y}) \cos(2\pi t), & u_y = \left[-\frac{2}{3}xy^3 + 2 - \pi \sin(\pi x) \right] \cos(2\pi t) \\ p = -[2 - \pi \sin(\pi x)] \cos(2\pi y) \cos(2\pi t) \end{cases}$$

(1) 网格划分与信息矩阵生成

本算例中压力场采用线性三角形单元，如图 5-1a 所示；速度场采用二次三角形单元，如图 5-1b 所示。

(2) 刚度矩阵组装

根据式 (7.32) 计算矩阵 $\mathbf{K}_1, \mathbf{K}_2, \mathbf{K}_3, \mathbf{K}_5, \mathbf{K}_6$ ，之后由式 (7.33) 得到刚度矩阵 \mathbf{K} ，矩阵规模为： $NN \times NN = 22 \times 22$ 。非稳态 Stokes 方程线性部分刚度矩阵与 Stokes 方程中各刚度矩阵相同，故此处不再重复给出。

(3) 质量矩阵组装

非稳态 Stokes 中还需要计算质量矩阵，由式 (7.35) 计算矩阵 \mathbf{M}_e ，之后由式 (7.36) 得到质量矩阵 \mathbf{M} 。在节算例网格划分情况下，所得质量矩阵 \mathbf{M}_e 的一种参考如下：

$$\mathbf{M}_e = \begin{bmatrix} 0.0042 & 0 & -0.0007 & 0 & -0.0028 & 0 & -0.0007 & 0 & 0 \\ 0 & 0.0222 & 0 & 0.0111 & 0.0111 & 0 & -0.0028 & 0 & 0 \\ -0.0007 & 0 & 0.0083 & -0.0028 & 0 & 0 & -0.0014 & -0.0028 & -0.0007 \\ 0 & 0.0111 & -0.0028 & 0.0222 & 0.0111 & 0 & 0 & 0 & 0 \\ -0.0028 & 0.0111 & 0 & 0.0111 & 0.0444 & 0.0111 & 0 & 0.0111 & -0.0028 \\ 0 & 0 & 0 & 0 & 0.0111 & 0.0222 & -0.0028 & 0.0111 & 0 \\ -0.0007 & -0.0028 & -0.0014 & 0 & 0 & -0.0028 & 0.0083 & 0 & -0.0007 \\ 0 & 0 & -0.0028 & 0 & 0.0111 & 0.0111 & 0 & 0.0222 & 0 \\ 0 & 0 & -0.0007 & 0 & -0.0028 & 0 & -0.0007 & 0 & 0.0042 \end{bmatrix} \quad (7.133)$$

(4) 载荷向量组装

非稳态 Stokes 方程中载荷向量与控制方程源项相关，且源项随时间变化，因此每次时间迭代中对应载荷向量均不同， $t = 0$ 时由式 (7.37) 所得载荷向量 $\mathbf{F}_x, \mathbf{F}_y$ 与第 五 章中稳态 Stokes 方程中载荷向量结果一致。

(5) 线性系统组装与边界条件处理

根据式(7.41)得到当前迭代等效线性系统，之后进行边界条件处理即可得到待求线性系统，边界条件处理可参考稳态 Stokes 方程的边界条件处理方法。

完成当前时间步迭代后，将当前时间步速度场与压力场作为下一时间步的初值，重复上述过程直至仿真结束，详细过程可参考算法 18。

(6) 线性系统求解及后处理

将本问题中网格细化为 8×2 ，取 $\theta = 1$ 、 $\Delta t = 1/64$ ，将仿真终止时刻的速度场与压力场所得有限元解与解析解进行比对。

图 7-1a 和 7-1b 所示分别为速度场 u_x 的有限元解与解析解，图 7-1c 和 7-1d 所示分别为速度场 u_y 的有限元解与解析解，图 7-2a 和 7-2b 所示分别为压力场 p 的有限元解与解析解。可以看到对于速度场而言由于采用 2 阶三角形单元在有限网格单元情况下与解析解吻合较好，压力场由于采用线性三角形单元其求解精度与解析解具有较大差距，可以通过细化网格或者使用更高阶的网格单元减小这种差距。

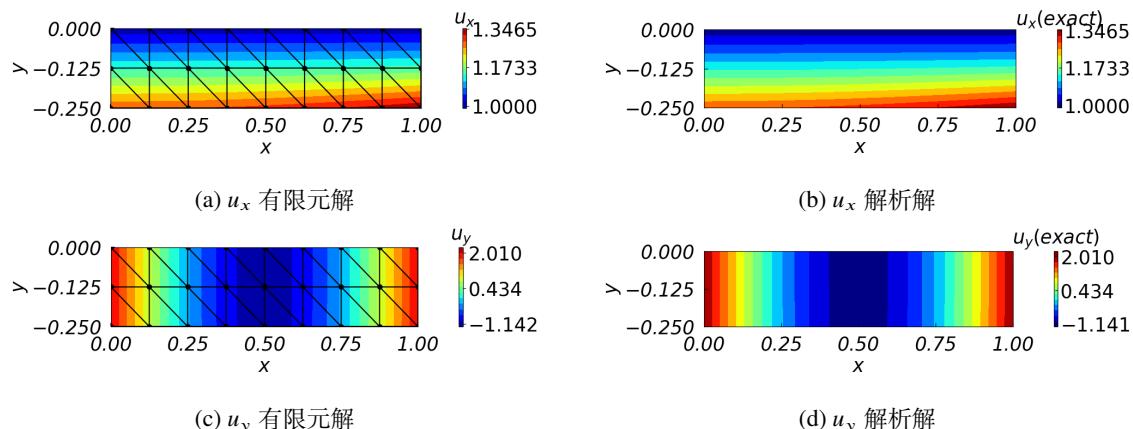


图 7-1 非稳态 Stokes 方程速度场 \mathbf{u} 有限元解与解析解

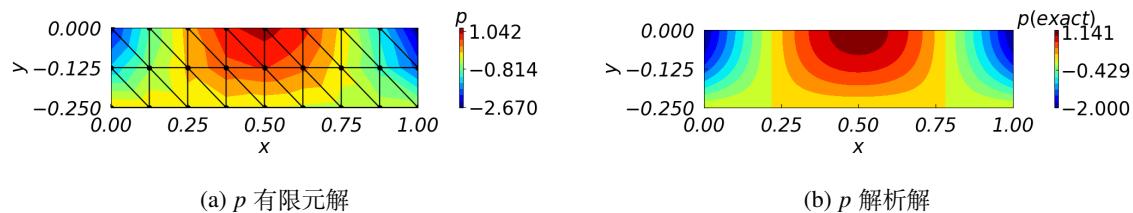


图 7-2 非稳态 Stokes 方程压力场 p 有限元解与解析解

表 7-1 和表 7-2 分别给出了 $\theta = 1$ 、 $\Delta t = 1/64$ ，不同空间步长下到达给定计算时间 ($t = 1$)，三角形单元非稳态 Stokes 方程速度场 \mathbf{u} 和压力场 p 的各类误差范数参考：

表 7-1 $\theta = 1$ ， $\Delta t = 1/64$ ，不同空间步长下速度场 \mathbf{u} 各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	5.6948×10^{-4}	1.6676×10^{-3}	3.6290×10^{-4}	2.0487×10^{-2}
1/16	5.5440×10^{-5}	2.1848×10^{-4}	4.5026×10^{-5}	5.0726×10^{-3}
1/32	5.9783×10^{-6}	2.7448×10^{-5}	5.6114×10^{-6}	1.2626×10^{-3}
1/64	6.9185×10^{-7}	3.3781×10^{-6}	7.0079×10^{-7}	3.1525×10^{-4}

表 7-2 $\theta = 1$ ， $\Delta t = 1/64$ ，不同空间步长下压力场 p 各类误差结果参考

Cell Size	maxNodeError	L^∞	L^2	H^1
1/8	6.6973×10^{-1}	5.7967×10^{-1}	1.3909×10^{-1}	1.3489×10^0
1/16	1.1329×10^{-1}	9.4258×10^{-2}	2.3063×10^{-2}	6.3538×10^{-1}
1/32	2.1108×10^{-2}	1.8080×10^{-2}	4.2194×10^{-3}	3.1396×10^{-1}
1/64	4.3653×10^{-3}	3.8072×10^{-3}	8.6779×10^{-4}	1.5660×10^{-1}

7.5.2 Example 2

例 7.2 【问题描述】以下述 2D 非稳态 Navier-Stokes 方程为例，对前述有限元方法进行验证，模型计算域为： $\Omega = [0, 1] \times [-0.25, 0]$ ，取 $\nu = 1$ 。

$$\left\{ \begin{array}{l} \mathbf{u}_t + (\mathbf{u} \cdot \nabla) \mathbf{u} - \nabla \cdot \mathbb{T}(\mathbf{u}, p) = \mathbf{f}, \quad \text{in } \Omega \times [0, 1] \\ \nabla \cdot \mathbf{u} = 0, \quad \text{in } \Omega \times [0, 1] \\ u_x = x^2 y^2 + e^{-y}, \quad \text{at } t = 0 \text{ and in } \Omega \\ u_y = -\frac{2}{3} x y^3 + 2 - \pi \sin(\pi x), \quad \text{at } t = 0 \text{ and in } \Omega \\ p = -[2 - \pi \sin(\pi x)] \cos(2\pi y), \quad \text{at } t = 0 \text{ and in } \Omega \\ u_x(0, y) = e^{-y} \cos(2\pi t), \quad u_x(1, y) = (y^2 + e^{-y}) \cos(2\pi t) \\ u_x(x, -0.25) = \left(\frac{1}{16} x^2 + e^{0.25} \right) \cos(2\pi t), \quad u_x(x, 0) = \cos(2\pi t) \\ u_y(0, y) = 2 \cos(2\pi t), \quad u_y(1, y) = \left(-\frac{2}{3} y^3 + 2 \right) \cos(2\pi t) \\ u_y(x, -0.25) = \left(\frac{1}{96} x^2 + 2 - \pi \sin(\pi x) \right) \cos(2\pi t), \quad u_y(x, 0) = [2 - \pi \sin(\pi x)] \cos(2\pi t) \end{array} \right.$$

其中：

$$\begin{aligned} f_x &= -2\pi(x^2y^2 + e^{-y}) \sin(2\pi t) + [-2\nu x^2 - 2\nu y^2 - \nu e^{-y} + \pi^2 \cos(\pi x) \cos(2\pi y)] \cos(2\pi t) \\ f_y &= -2\pi \left[-\frac{2}{3}xy^3 + 2 - \pi \sin(\pi x) \right] \sin(2\pi t) + 4\nu xy - \nu \pi^3 \sin(\pi x) + 2\pi(2 - \pi \sin(\pi x)) \sin(2\pi y) \end{aligned}$$

【说明】 上述问题的解析解为：

$$\begin{cases} u_x = (x^2y^2 + e^{-y}) \cos(2\pi t), & u_y = \left[-\frac{2}{3}xy^3 + 2 - \pi \sin(\pi x) \right] \cos(2\pi t) \\ p = -[2 - \pi \sin(\pi x)] \cos(2\pi y) \cos(2\pi t) \end{cases}$$

本问题的求解过程可结合前述非稳态 Stokes 方程及稳态 Navier-Stokes 方程的求解过程得到，这里不再赘述，留给读者自行完成。

习题

- 分别采用三角形、四边形单元有限元方法求解如下方程，并计算所得有限元解的各类误差（最大节点误差、 L^∞ 误差、 L^2 误差、 H^1 误差）。计算域为 $\Omega = [0, 1]^2$ ，网格规模为 1×1 。

$$\begin{cases} \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla) \mathbf{u} = -\nabla p + \Delta \mathbf{u}, & \text{in } \Omega \times [0, T] \\ \nabla \cdot \mathbf{u} = 0, & \text{in } \Omega \times [0, T] \\ \mathbf{u}_0(x, y) = 0, & \text{on } \Omega \times 0 \\ \mathbf{u}(x, y) = 0, & \text{on } \Omega \times [0, T] \\ p(x, y) = 8, & \text{on } [0] \times [0, 1] \times [0, T] \\ p(x, y) = 0, & \text{on } [1] \times [0, 1] \times [0, T] \end{cases}$$

其中， $\mathbf{f} = 0$ ，该问题解析解为： $u_y = (4y(1-y), 0)$ ， $p = 8(1-x)$ 。

8

3D 有限元方法简介

8.1 引言

在前述章节中已经对 2D 情况下各类 PDE 的有限元方法进行了介绍，本章将以 3D Poisson 方程为例，对 3D 情况有限元方法进行简要介绍，主要包括：方程弱格式推导、线性系统构建、边界条件处理及算例演示。本章内容仅作为 3D 有限元方法的入门介绍，在实际应用中可结合本章内容及前述章节内容进行更深入的研究。

8.2 问题模型

以下述 3D 2 阶椭圆方程为例对 3D FEM 方法求解方程详细过程进行介绍：

$$\begin{cases} -\nabla \cdot (c \nabla u) = f, & (x, y, z) \in \Omega \\ u = g, & (x, y, z) \in \partial\Omega \end{cases} \quad (8.1)$$

其中， Ω 为一个 3D 空间域， $f(x, y, z)$ 和 $c(x, y, z)$ 为空间 Ω 上已知函数， $g(x, y, z)$ 为边界 $\partial\Omega$ 上已知函数， $u(x, y, z)$ 为待求函数——试探函数（Trial Function）。

8.2.1 弱格式

(1) 方程两边同时乘以一个测试函数 $v(x, y, z)$

$$-\nabla \cdot (c \nabla u) v = fv \quad (8.2)$$

(2) 对方程两边进行积分

$$-\int_{\Omega} \nabla \cdot (c \nabla u) v d\Omega = \int_{\Omega} f v d\Omega \quad (8.3)$$

(3) 积分变换

$$\int_{\Omega} \nabla \cdot (c \nabla u) v dx dy dz = \int_{\partial\Omega} (c \nabla u \cdot \vec{n}) v ds - \int_{\Omega} c \nabla u \cdot \nabla v dx dy dz \quad (8.4)$$

$$\int_{\Omega} c \nabla u \cdot \nabla v dx dy dz - \int_{\partial\Omega} (c \nabla u \cdot \vec{n}) v ds = \int_{\Omega} f v dx dy dz \quad (8.5)$$

在边界 $\partial\Omega$ 处试探函数值 $u(x, y, z) = g(x, y, z)$ 为已知, 由于测试函数 $v(x, y, z)$ 为任意函数, 故此可以令其在边界处为 0, 即 $v(x, y, z) = 0, (x, y, z) \in \partial\Omega$ 。由此可以将上式进一步简化得到:

$$\int_{\Omega} c \nabla u \cdot \nabla v dx dy dz = \int_{\Omega} f v dx dy dz \quad (8.6)$$

方程 (8.6) 即为 3D2 阶椭圆方程的弱格式。

8.2.2 方程离散

令 $a(u, v) = \int_{\Omega} c \nabla u \cdot \nabla v dx dy dz$, $(f, v) = \int_{\Omega} f v dx dy dz$, 求解试探函数, 就是在空间 Ω 中寻找 u , 使其对任意 $v \in H_0^1(\Omega)$ 满足:

$$a(u, v) = (f, v) \quad (8.7)$$

上述弱格式就是在空间划分后的子空间内 ($U_h \in H^1(\Omega)$), 得到各个子空间内试探函数的解 u_h , 随后将各个子空间得解合并得到整个空间的解。

$$\begin{aligned} a(u_h, v_h) &= (f, v_h) \\ \Leftrightarrow \int_{\Omega} c \nabla u_h \cdot \nabla v_h dx dy dz &= \int_{\Omega} f v_h dx dy dz \end{aligned} \quad (8.8)$$

将离散后的各节点函数值按照基底展开:

$$u_h(x_k, y_k, z_k) = \sum_{j=1}^{N_b} u_j \phi_j(\xi_k, \eta_k, \zeta_k) = u_k \quad (8.9)$$

其中, N_b 为有限元基函数节点数。

选取合适测试函数：较为简单的操作就是将基函数作为测试函数，在各个离散空间上选择基函数作为测试函数： $v_h = \phi_i, (i = 1, \dots, N_b)$ ，可以得到有限元方程格式为：

$$\begin{aligned} & \int_{\Omega} c \nabla \left(\sum_{j=1}^{N_b} u_j \phi_j \right) \cdot \nabla \phi_i dx dy dz = \int_{\Omega} f \phi_i dx dy dz \\ & \rightarrow \sum_{j=1}^{N_b} u_j \left[\int_{\Omega} c \nabla \phi_j \cdot \nabla \phi_i dx dy dz \right] = \int_{\Omega} f \phi_i dx dy dz, (i = 1, \dots, N_b) \end{aligned} \quad (8.10)$$

类比 1D、2D 情况可以知道，刚度矩阵为：

$$\mathbf{K} = [K_{ij}]_{i,j=1}^{N_b} = \left[\int_{\Omega} c \nabla \phi_j \cdot \nabla \phi_i dx dy dz \right]_{i,j=1}^{N_b} \quad (8.11)$$

载荷向量：

$$\mathbf{F} = [b_i]_{i=1}^{N_b} = \left[\int_{\Omega} f \phi_i dx dy dz \right]_{i=1}^{N_b} \quad (8.12)$$

8.2.3 线性系统构建

(1) 组装刚度矩阵

为了方便对刚度矩阵中各个元素进行计算，通常先计算各个单元的局部刚度阵，可以看到当前刚度矩阵各个元素的计算是在全局坐标系下进行，如前所述为了避免复杂的坐标转换，通常将刚度矩阵元素的计算由全局坐标转换至参考单元坐标系下进行。其转换关系如下：

$$\begin{aligned} K_{ij}^e &= \int_{\Omega} c \nabla \phi_j \cdot \nabla \phi_i dV \\ &= \int_{\Omega} c(\xi, \eta, \zeta) \nabla_{(x,y,z)} \psi_j \cdot \nabla_{(x,y,z)} \psi_i \left| \frac{\partial(x, y, z)}{\partial(\xi, \eta, \zeta)} \right| d\xi d\eta d\zeta \\ &= \int_{\Omega} c(\xi, \eta, \zeta) \left(\frac{\partial \psi_j}{\partial x} \frac{\partial \psi_i}{\partial x} + \frac{\partial \psi_j}{\partial y} \frac{\partial \psi_i}{\partial y} + \frac{\partial \psi_j}{\partial z} \frac{\partial \psi_i}{\partial z} \right) |J| d\xi d\eta d\zeta \end{aligned} \quad (8.13)$$

可以看到矩阵系统中各元素主要由 x, y, z 三部分组成，所以可以将其分为三部分分别计算，每一部分可通过算法 21 分别得到，详细过程如下：

- ▶ 令算法 21 中 $r = 1, s = 0, m = 0; p = 1, q = 0, n = 0$ ，得到 x 分量；
- ▶ 令算法 21 中 $r = 0, s = 1, m = 0; p = 0, q = 1, n = 0$ ，得到 y 分量；
- ▶ 令算法 21 中 $r = 0, s = 0, m = 1; p = 0, q = 0, n = 1$ ，得到 z 分量；

将得到的分量相加即可得到矩阵系统中各元素的值， $K_{ij}^e = K_{ij,x}^e + K_{ij,y}^e + K_{ij,z}^e$ 。

Algorithm 21 3D2 阶椭圆方程刚度矩阵组装

Require: $\mathbf{K} = \text{sparse}(N_b^{test}, N_b^{trial})$

```

1: for  $n = 1, \dots, N$  do
2:   for  $\alpha = 1, \dots, N_{lb}^{trial}$  do
3:     for  $\beta = 1, \dots, N_{lb}^{test}$  do
4:       Compute  $r = \int_{E_n} c \frac{\partial^{r+s+m} \varphi_{n\alpha}}{\partial x^r \partial y^s \partial z^m} \frac{\partial^{p+q+n} \varphi_{n\beta}}{\partial x^p \partial y^q \partial z^n} dx dy dz$ 
5:        $\mathbf{K}(T_b^{test}(\beta, n), T_b^{trial}(\alpha, n)) += r$ 
6:     end for
7:   end for
8: end for

```

(2) 载荷向量组装

载荷向量中各元素的计算方式与刚度阵中各元素的计算方法相似。

$$\begin{aligned} \int_{\Omega} f \phi_i dV &= \int_{\Omega} f(\xi, \eta, \zeta) \psi_i(\xi, \eta, \zeta) \left| \frac{\partial(x, y, z)}{\partial(\xi, \eta, \zeta)} \right| d\xi d\eta d\zeta \\ &= \int_{\Omega} f(\xi, \eta, \zeta) \psi_i(\xi, \eta, \zeta) |J| d\xi d\eta d\zeta \end{aligned} \quad (8.14)$$

其中, J 为雅可比矩阵, 该行列式的值在 3D FEM 中为有限元单元体积与参考单元的体积比, 其具有如下形式:

$$\begin{aligned} J &= \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{NN} \frac{\partial N_i(\xi, \eta, \zeta)}{\partial \xi} x_i & \sum_{i=1}^{NN} \frac{\partial N_i(\xi, \eta, \zeta)}{\partial \xi} y_i & \sum_{i=1}^{NN} \frac{\partial N_i(\xi, \eta, \zeta)}{\partial \xi} z_i \\ \sum_{i=1}^{NN} \frac{\partial N_i(\xi, \eta, \zeta)}{\partial \eta} x_i & \sum_{i=1}^{NN} \frac{\partial N_i(\xi, \eta, \zeta)}{\partial \eta} y_i & \sum_{i=1}^{NN} \frac{\partial N_i(\xi, \eta, \zeta)}{\partial \eta} z_i \\ \sum_{i=1}^{NN} \frac{\partial N_i(\xi, \eta, \zeta)}{\partial \zeta} x_i & \sum_{i=1}^{NN} \frac{\partial N_i(\xi, \eta, \zeta)}{\partial \zeta} y_i & \sum_{i=1}^{NN} \frac{\partial N_i(\xi, \eta, \zeta)}{\partial \zeta} z_i \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \xi} & \dots & \frac{\partial N_{NN}}{\partial \xi} \\ \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \dots & \frac{\partial N_{NN}}{\partial \eta} \\ \frac{\partial N_1}{\partial \zeta} & \frac{\partial N_2}{\partial \zeta} & \dots & \frac{\partial N_{NN}}{\partial \zeta} \end{bmatrix} \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_{NN} & y_{NN} & z_{NN} \end{bmatrix} \end{aligned} \quad (8.15)$$

各项节点基函数对参考单元的导数为:

$$\left\{ \begin{array}{l} \frac{\partial N_i}{\partial \xi} = \frac{\partial N_i}{\partial x} \frac{\partial x}{\partial \xi} + \frac{\partial N_i}{\partial y} \frac{\partial y}{\partial \xi} + \frac{\partial N_i}{\partial z} \frac{\partial z}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} = \frac{\partial N_i}{\partial x} \frac{\partial x}{\partial \eta} + \frac{\partial N_i}{\partial y} \frac{\partial y}{\partial \eta} + \frac{\partial N_i}{\partial z} \frac{\partial z}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} = \frac{\partial N_i}{\partial x} \frac{\partial x}{\partial \zeta} + \frac{\partial N_i}{\partial y} \frac{\partial y}{\partial \zeta} + \frac{\partial N_i}{\partial z} \frac{\partial z}{\partial \zeta} \end{array} \right. \rightarrow \left[\begin{array}{c} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{array} \right] = \left[\begin{array}{ccc} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{array} \right] \left[\begin{array}{c} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{array} \right]$$

$$\rightarrow \begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{bmatrix} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} & \frac{\partial \zeta}{\partial x} \\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} & \frac{\partial \zeta}{\partial y} \\ \frac{\partial \xi}{\partial z} & \frac{\partial \eta}{\partial z} & \frac{\partial \zeta}{\partial z} \end{bmatrix} \begin{bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{bmatrix} = J^{-1} \begin{bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{bmatrix} \quad (8.16)$$

载荷向量的组装方法如算法 22 所示，令 $p = q = n = 0$ 即可得到 \mathbf{F} 。

Algorithm 22 3D 椭圆方程载荷向量组装

Require: $\mathbf{F} = \text{sparse}(N_b, 1)$

- 1: **for** $n = 1, \dots, N$ **do**
 - 2: **for** $\beta = 1, \dots, N_{lb}$ **do**
 - 3: Compute $r = \int_{E_n} f \frac{\partial^{p+q+n} \varphi_{n\beta}}{\partial x^p \partial y^q \partial z^n} dx dy dz$
 - 4: $\mathbf{F}(T_b^{test}(\beta, n), 1) += r$
 - 5: **end for**
 - 6: **end for**
-

(3) 待求函数值

$$\mathbf{u} = [u_j]_{j=1}^{N_b} \quad (8.17)$$

到此，已经得到了 3D FEM 方法需要求解的线性系统，随后根据待求问题给定的边界条件，参考 1D、2D FEM 方法中介绍的不同边界条件下刚度矩阵及载荷向量的处理方式得到边界处理后的线性系统，并对其进行求解即可得到最终结果，最后根据需要进行相应后处理即可。

8.3 3D 基函数与有限元空间

由前所述，在实际进行计算时，为了方便通常将局部单元的刚度阵、载荷向量的计算转换至参考单元下进行计算。随着空间维度的提升，能够用于表示该维度空间的有限元几何形状也逐渐丰富，对于 3D 有限元方法而言常用的单元类型主要由四面体单元、六面体单元、金字塔形单元、棱柱形单元等。

本节主要对常见四面体单元与六面体单元的参考单元基函数进行介绍。

8.3.1 六面体单元

(1) 六面体参考单元基函数

对于六面体单元而言，其参考单元通常为一立方体，该单元与有限元单元的映射关系如图 8-1 所示。

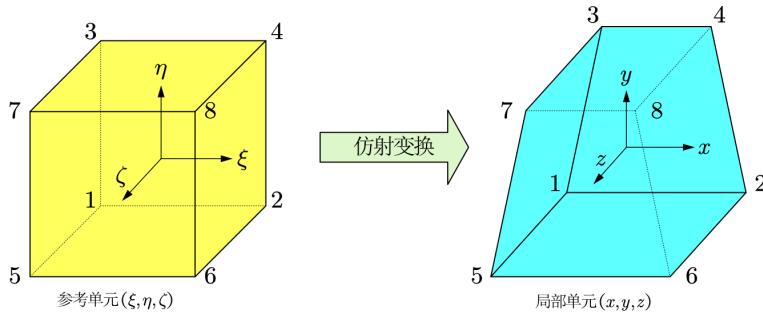


图 8-1 3D 六面体单元参考到局部变换 (*reference* \rightarrow *local*)

在该参考单元中各参考节点坐标及其对应基函数如表 8-1 所示。

表 8-1 3D 六面体单元参考节点坐标及基函数

参考节点编号	参考节点坐标	参考节点对应基函数
1	(-1,-1,-1)	$N_1(\xi, \eta, \zeta) = \frac{1}{8}(1-\xi)(1-\eta)(1-\zeta)$
2	(+1,-1,-1)	$N_2(\xi, \eta, \zeta) = \frac{1}{8}(1+\xi)(1-\eta)(1-\zeta)$
3	(-1,+1,-1)	$N_3(\xi, \eta, \zeta) = \frac{1}{8}(1-\xi)(1+\eta)(1-\zeta)$
4	(+1,+1,-1)	$N_4(\xi, \eta, \zeta) = \frac{1}{8}(1+\xi)(1+\eta)(1-\zeta)$
5	(-1,-1,+1)	$N_5(\xi, \eta, \zeta) = \frac{1}{8}(1-\xi)(1-\eta)(1+\zeta)$
6	(+1,-1,+1)	$N_6(\xi, \eta, \zeta) = \frac{1}{8}(1+\xi)(1-\eta)(1+\zeta)$
7	(-1,+1,+1)	$N_7(\xi, \eta, \zeta) = \frac{1}{8}(1-\xi)(1+\eta)(1+\zeta)$
8	(+1,+1,+1)	$N_8(\xi, \eta, \zeta) = \frac{1}{8}(1+\xi)(1+\eta)(1+\zeta)$

上述各个节点的形函数也可以表示为: $N_i(\xi, \eta, \zeta) = \frac{1}{8}(1 + \xi_i \xi)(1 + \eta_i \eta)(1 + \zeta_i \zeta)$, 其中, (ξ_i, η_i, ζ_i) 为参考单元节点 i 的坐标, 并且 $i = 1, \dots, 8$ 。

(2) 六面体参考单元高斯积分点及其权重

在得到参考单元基函数后, 需要进行仿射变换得到局部单元基函数, 随后对单元刚度阵各个元素进行计算, 计算时采用 gauss 积分, 本问题为 3D, 因此需要对 3 个方向进行积分, 按照如图 8-2 所示将 3D 积分转换为 3 个方向上对应的 1D 积分即可。积分时各个积分点及其对应权重系数如表 8-2 所示。

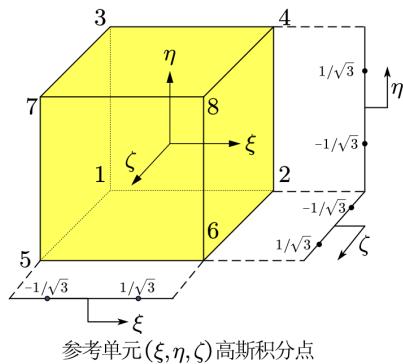


图 8-2 3D 六面体单元高斯积分点

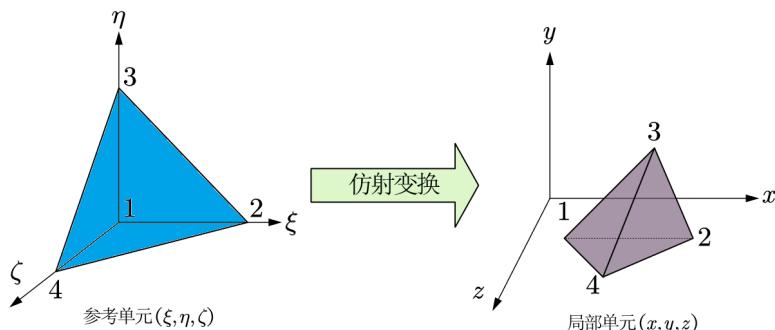
表 8-2 3D 六面体单元高斯积分点及权重

参考节点积分点编号	参考节点积分点坐标	权重系数
1	$(-1/\sqrt{3}, -1/\sqrt{3}, -1/\sqrt{3})$	1
2	$(+1/\sqrt{3}, -1/\sqrt{3}, -1/\sqrt{3})$	1
3	$(-1/\sqrt{3}, +1/\sqrt{3}, -1/\sqrt{3})$	1
4	$(+1/\sqrt{3}, +1/\sqrt{3}, -1/\sqrt{3})$	1
5	$(-1/\sqrt{3}, -1/\sqrt{3}, +1/\sqrt{3})$	1
6	$(+1/\sqrt{3}, -1/\sqrt{3}, +1/\sqrt{3})$	1
7	$(-1/\sqrt{3}, +1/\sqrt{3}, +1/\sqrt{3})$	1
8	$(+1/\sqrt{3}, +1/\sqrt{3}, +1/\sqrt{3})$	1

8.3.2 四面体单元

(1) 四面体参考单元基函数

对于四面体单元而言，其参考单元通常为一标准四面体，该单元与有限元单元的映射关系如图 8-3 所示。

图 8-3 3D 四面体单元参考到局部变换 (*reference* \rightarrow *local*)

在该参考单元中各参考节点坐标及其对应基函数如表 8-3 所示。

表 8-3 3D 四面体单元参考节点坐标及基函数

参考节点编号	参考节点坐标	参考节点对应基函数
1	(0,0,0)	$N_1(\xi, \eta, \zeta) = 1 - \xi - \eta - \zeta$
2	(1,0,0)	$N_2(\xi, \eta, \zeta) = \xi$
3	(0,1,0)	$N_3(\xi, \eta, \zeta) = \eta$
4	(0,0,1)	$N_4(\xi, \eta, \zeta) = \zeta$

(2) 四面体参考单元高斯积分点及其权重

四面体参考单元各积分点及其对应权重系数如表 8-4 所示。

表 8-4 3D 四面体单元高斯积分点及权重

参考节点积分点编号	参考节点积分点坐标	权重系数
1	(0.58541020, 0.13819660, 0.13819660)	1/24
2	(0.13819660, 0.58541020, 0.13819660)	1/24
3	(0.13819660, 0.13819660, 0.58541020)	1/24
4	(0.13819660, 0.13819660, 0.13819660)	1/24

8.4 应用实例

8.4.1 Example 1

例 8.1 【问题描述】以下述 3D 泊松方程的有限元求解对前述有限元方法进行验证，模型计算域为： $\Omega = [0, 0, 0] \times [0.1, 0.1, 0.01]$ 。

$$\begin{cases} -\nabla \cdot (\nabla u) = 0 \in \Omega \\ u(x, y, 0) = u(x, y, 0.01) = u(0, y, z) = u(0.1, y, z) = u(x, 0, z) = 0 \\ u(x, 0.01, z) = 100 \end{cases}$$

这里以 3D 六面体单元的求解过程进行说明，其他单元类型求解过程类似。

1、网格划分

对上述问题计算域进行网格划分，为了方便检验这里网格划分数较少，以说明 3D 有限元方法的计算过程，如图 8-4 所示。

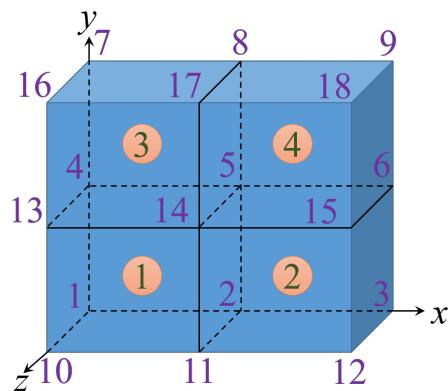


图 8-4 3D 六面体单元网格划分示意

根据划分的网格，产生对应的网格信息矩阵，其基本思路与 2D 有限元方法相同。

2、刚度矩阵组装

根据当前划分网格节点数量可知矩阵的规模为： $NN \times NN = 18 \times 18 = 324$ ，对该刚度阵进行组装时，仍然先对各个单元刚度阵进行计算，之后将其合并得到总体刚度矩阵。

对本问题中六面体单元而言，组成该单元的节点数为 8，单元刚度矩阵规模为： $8 \times 8 = 64$ ，这里以第 1 个单元刚度矩阵计算为例对上述过程进行说明。

(1) Jacobian 矩阵

$$J^1 = \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \xi} & \dots & \frac{\partial N_8}{\partial \xi} \\ \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \dots & \frac{\partial N_8}{\partial \eta} \\ \frac{\partial N_1}{\partial \zeta} & \frac{\partial N_2}{\partial \zeta} & \dots & \frac{\partial N_8}{\partial \zeta} \end{bmatrix} \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_8 & y_8 & z_8 \end{bmatrix} = \frac{1}{8} \begin{bmatrix} 0.2 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.04 \end{bmatrix} = \begin{bmatrix} 0.025 & 0 & 0 \\ 0 & 0.025 & 0 \\ 0 & 0 & 0.005 \end{bmatrix} \quad (8.18)$$

(2) Jacobian 矩阵的行列式：

$$|J^1| = 0.025 \times 0.025 \times 0.005 = 0.000003125 \quad (8.19)$$

(3) Jacobian 矩阵的逆

$$[J^1]^{-1} = \begin{bmatrix} 40 & 0 & 0 \\ 0 & 40 & 0 \\ 0 & 0 & 200 \end{bmatrix} \quad (8.20)$$

(4) 组成该单元的各个元素

$$\begin{aligned}
 K_{11}^1 &= \int_{\Omega^1} \left(\frac{\partial \psi_1}{\partial x} \frac{\partial \psi_1}{\partial x} + \frac{\partial \psi_1}{\partial y} \frac{\partial \psi_1}{\partial y} + \frac{\partial \psi_1}{\partial z} \frac{\partial \psi_1}{\partial z} \right) |J| d\xi d\eta d\zeta \\
 &= \int_{\Omega^1} \left\{ \left(J_{11}^{-1} \frac{\partial N_1}{\partial \xi} + J_{12}^{-1} \frac{\partial N_1}{\partial \eta} + J_{13}^{-1} \frac{\partial N_1}{\partial \zeta} \right) \left(J_{11}^{-1} \frac{\partial N_1}{\partial \xi} + J_{12}^{-1} \frac{\partial N_1}{\partial \eta} + J_{13}^{-1} \frac{\partial N_1}{\partial \zeta} \right) \right. \\
 &\quad + \left(J_{21}^{-1} \frac{\partial N_1}{\partial \xi} + J_{22}^{-1} \frac{\partial N_1}{\partial \eta} + J_{23}^{-1} \frac{\partial N_1}{\partial \zeta} \right) \left(J_{21}^{-1} \frac{\partial N_1}{\partial \xi} + J_{22}^{-1} \frac{\partial N_1}{\partial \eta} + J_{23}^{-1} \frac{\partial N_1}{\partial \zeta} \right) \\
 &\quad \left. + \left(J_{31}^{-1} \frac{\partial N_1}{\partial \xi} + J_{32}^{-1} \frac{\partial N_1}{\partial \eta} + J_{33}^{-1} \frac{\partial N_1}{\partial \zeta} \right) \left(J_{31}^{-1} \frac{\partial N_1}{\partial \xi} + J_{32}^{-1} \frac{\partial N_1}{\partial \eta} + J_{33}^{-1} \frac{\partial N_1}{\partial \zeta} \right) \right\} |J| d\xi d\eta d\zeta \\
 &= 3.125 \times 10^{-6} \times \int_{-1}^1 \int_{-1}^1 \int_{-1}^1 [-5(1-\eta)(1-\zeta)] [-5(1-\eta)(1-\zeta)] \\
 &\quad + [-5(1-\xi)(1-\zeta)] [-5(1-\xi)(1-\zeta)] + [-25(1-\xi)(1-\eta)] [-25(1-\xi)(1-\eta)] d\xi d\eta d\zeta \\
 &= 0.0300
 \end{aligned} \tag{8.21}$$

$$\begin{aligned}
 K_{12}^1 = K_{21}^1 &= \int_{\Omega^1} \left(\frac{\partial \psi_2}{\partial x} \frac{\partial \psi_1}{\partial x} + \frac{\partial \psi_2}{\partial y} \frac{\partial \psi_1}{\partial y} + \frac{\partial \psi_2}{\partial z} \frac{\partial \psi_1}{\partial z} \right) |J| d\xi d\eta d\zeta \\
 &= 3.125 \times 10^{-6} \times \int_{-1}^1 \int_{-1}^1 \int_{-1}^1 [5(1-\eta)(1-\zeta)] [-5(1-\eta)(1-\zeta)] \\
 &\quad + [-5(1-\xi)(1-\zeta)] [-5(1+\xi)(1-\zeta)] + [-25(1-\xi)(1-\eta)] [-25(1+\xi)(1-\eta)] d\xi d\eta d\zeta \\
 &= 0.0133
 \end{aligned} \tag{8.22}$$

$$\begin{aligned}
 K_{13}^e = K_{31}^1 &= \int_{\Omega^e} \left(\frac{\partial \psi_3}{\partial x} \frac{\partial \psi_1}{\partial x} + \frac{\partial \psi_3}{\partial y} \frac{\partial \psi_1}{\partial y} + \frac{\partial \psi_3}{\partial z} \frac{\partial \psi_1}{\partial z} \right) |J| d\xi d\eta d\zeta \\
 &= 3.125 \times 10^{-6} \times \int_{-1}^1 \int_{-1}^1 \int_{-1}^1 [-5(1+\eta)(1-\zeta)] [-5(1-\eta)(1-\zeta)] \\
 &\quad + [5(1-\xi)(1-\zeta)] [-5(1-\xi)(1-\zeta)] + [-25(1-\xi)(1+\eta)] [-25(1-\xi)(1-\eta)] d\xi d\eta d\zeta \\
 &= 0.0133
 \end{aligned} \tag{8.23}$$

同理可以得到该单元刚度矩阵的其他元素，最终得到单元 1 的局部刚度矩阵：

$$K_{8 \times 8}^1 = \begin{bmatrix} 0.03 & 0.0133 & 0.0133 & 0.0058 & -0.0267 & -0.0142 & -0.0142 & -0.0075 \\ 0.0133 & 0.03 & 0.0058 & 0.0133 & -0.0142 & -0.0267 & -0.0075 & -0.0142 \\ 0.0133 & 0.0058 & 0.03 & 0.0133 & -0.0142 & -0.0075 & -0.0267 & -0.0142 \\ 0.0058 & 0.0133 & 0.0133 & 0.03 & -0.0075 & -0.0142 & -0.0142 & -0.0267 \\ -0.0267 & -0.0142 & -0.0142 & -0.0075 & 0.03 & 0.0133 & 0.0133 & 0.0058 \\ -0.0142 & -0.0267 & -0.0075 & -0.0142 & 0.0133 & 0.03 & 0.0058 & 0.0133 \\ -0.0142 & -0.0075 & -0.0267 & -0.0142 & 0.0133 & 0.0058 & 0.03 & 0.0133 \\ -0.0075 & -0.0142 & -0.0142 & -0.0267 & 0.0058 & 0.0133 & 0.0133 & 0.03 \end{bmatrix} \tag{8.24}$$

按照上述方法计算其他单元刚度矩阵，之后将其添加至全局刚度矩阵，得到完整的刚度矩阵 \mathbf{K} 。

3、载荷向量组装

当前算例中载荷向量函数为： $\mathbf{F}(x, y, z) = 100$ ，由于网格为正规网格所以得到的载荷向量都相同，

$$\begin{aligned} F_1^1 &= \int_{\Omega^1} f(\xi, \eta, \zeta) \psi_1(\xi, \eta, \zeta) |J| d\xi d\eta d\zeta \\ &= 3.125 \times 10^{-6} \times \int_{-1}^1 \int_{-1}^1 \int_{-1}^1 12.5 (1 - \xi) (1 - \eta) (1 - \zeta) d\xi d\eta d\zeta \\ &= 0.0003125 \end{aligned} \quad (8.25)$$

根据上述方法依次得到其他单元的载荷向量，最终得到完整的载荷向量 \mathbf{F} 。

4、边界条件处理、线性系统求解及后处理

采用前述章节中“置 1”法对本算例中边界进行处理，经过边界处理后可以得到线性系统及各节点函数值。细化网格后可得到本算例模型有限元结果如图 8-5 所示。

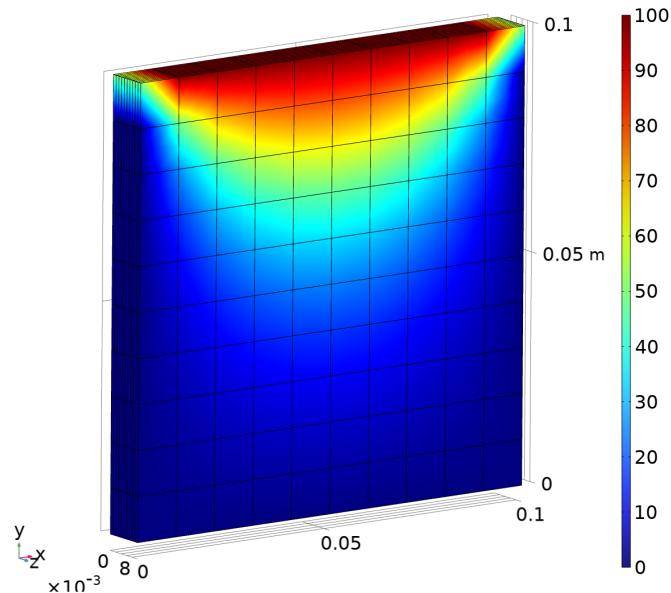


图 8-5 3D Example 1 算例模型有限元结果

以上为 3D 有限元方法的一般过程，其他 3D 问题模型均可参考上述方法及前述章节内容得到对应问题模型的有限元解。

参考文献

- [1] R. BYRON BIRD E N L, Warren E. Stewart. Transport Phenomena, Revised 2nd Edition [M]. USA: Wiley, 2006.
- [2] EMANUEL G. ANALYTICAL FLUID DYNAMICS[M]. Boca Raton London New York: CRC Press, 2016.
- [3] HASS J, HEIL C, WEIR M D, et al. Thomas' Calculus: Early Transcendentals[M]. Fourteenth edition. Boston: Pearson, 2018.
- [4] SURANA K S. Numerical Methods and Methods of Approximation in Science and Engineering[M]. CRC Press, 2018.
- [5] 张宏伟, 金光日, 施吉林, 等. 计算机科学计算[M]. 北京: 高等教育出版社, 2013.
- [6] LARSON M G, BENGZON F. The Finite Element Method: Theory, Implementation, and Applications: vol. 10[M]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013 [2023-05-30].
- [7] POLYCARPOU A C. Introduction to the Finite Element Method in Electromagnetics[M]. Cham: Springer International Publishing, 2006.
- [8] SMITH I M, GRIFFITHS D V, MARGETTS L. Programming the Finite Element Method [M]. Hoboken, New Jersey: John Wiley & Sons, Ltd, 2014.
- [9] CHEN Z. Finite Element Methods and Their Applications[M]. Berlin ; New York: Springer, 2005.
- [10] JIN J M. The Finite Element Method in Electromagnetics, 3rd Edition[M]. New York: Wiley, 2014,3.
- [11] CHAPRA S C, CANALE R P. Numerical Methods for Engineers[M]. Seventh edition. New York, NY: McGraw-Hill Education, 2015.
- [12] PLEWA T, LINDE T J, WEIRS V G. Adaptive Mesh Refinement, Theory and Applications: Proceedings of the Chicago Workshop on Adaptive Mesh Refinement Methods, Sept. 3-5, 2003[M]. Berlin ; New York: Springer, 2005.

- [13] LO D S H. Finite Element Mesh Generation[M]. London: CRC Press, 2015.
- [14] DARYL L. L. A First Course in the Finite Element Method (6th Edition)[M]. Boston, MA: Cengage Learning, 2022.
- [15] FISH J, BELYTSCHKO T. A First Course in Finite Elements[M]. Chichester, England ; Hoboken, NJ: John Wiley & Sons Ltd, 2007.
- [16] 王小如. A Virtual Element Method Based on 2D Linear Elastic Plate with Hole[J]. Advances in Applied Mathematics, 2022, 11: 3839-3848.
- [17] De BORST R, CRISFIELD M A. Nonlinear Finite Element Analysis of Solids and Structures[M]. 2nd ed. Hoboken, NJ: Wiley, 2012.
- [18] 刘鸿文, 林建兴, 曹曼玲. 材料力学[M]. 北京: 高等教育出版社, 2017.
- [19] 徐芝纶. 弹性力学[M]. 北京: 高等教育出版社, 2016.
- [20] HUO Z, MEI G, XU N. juSFEM: A Julia-based Open-Source Package of Parallel Smoothed Finite Element Method (S-FEM) for Elastic Problems[J]. Computers & Mathematics with Applications. Development and Application of Open-source Software for Problems with Numerical PDEs 2021, 81: 459-477.
- [21] ZHOU M, QIN J, HUO Z, et al. epSFEM: A Julia-Based Software Package of Parallel Incremental Smoothed Finite Element Method (S-FEM) for Elastic-Plastic Problems[J]. Mathematics, 2022, 10(12).
- [22] CINATL E. Finite Element Discretizations for Linear Elasticity[C/OL]//. 2018. <https://api.semanticscholar.org/CorpusID:146101139>.
- [23] 侯新录. 结构分析中的有限元法与程序设计[M]. 北京: 中国建材工业出版社.
- [24] 周光炯. 流体力学[M]. 高等教育出版社, 2000.
- [25] D.ANDERSON J. 计算流体力学基础及其应用[M]. 机械工业出版社, 2007.
- [26] 高歌. 计算流体力学典型算法与算例[M]. 机械工业出版社, 2015.
- [27] ÇENGEL Y A, CIMBALA J M. Fluid Mechanics: Fundamentals and Applications[M]. Fourth edition. New York, NY: McGraw-Hill Education, 2018.
- [28] PANTON R L. Incompressible Flow[M]. John Wiley & Sons, Ltd, 2013.
- [29] LOGG A, MARDAL K A, WELLS G. Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book: vol. 84[M]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

- [30] 陈金甫. 偏微分方程数值解法 (第 2 版) [M]. 北京: 清华大学出版社, 2004,1.
- [31] MAZUMDER S. Numerical Methods for Partial Differential Equations: Finite Difference and Finite Volume Methods[M]. Amsterdam Boston Heidelberg London: Elsevier, AP, 2016.
- [32] 李荣华, 李播. 微分方程数值解法[M]. 北京: 高等教育出版社.
- [33] 华冬英, 李祥贵. 微分方程的数值解法与程序实现[M]. 北京: 电子工业出版社, 2016.

A

2D 常用高斯积分点及权重

2D 三角形单元高斯积分点及权重

附录-表 A.1 2D 三角形单元 Gauss 积分表

Number of Points	Location ξ	Location η	Weight w
3(5)	0.5	0	1/6
	0	0.5	1/6
	0.5	0.5	1/6
4(7)	1/3	1/3	-27/96
	0.6	0.2	25/96
	0.2	0.6	25/96
	0.2	0.2	25/96
	1/2	1/4	8/81
9(17)	$(1 + \sqrt{3/5})/2$	$(1 - \sqrt{3/5}) \times (1 + \sqrt{3/5})/4$	$100/324 \times (1 - \sqrt{3/5})/8$
	$(1 + \sqrt{3/5})/2$	$(1 - \sqrt{3/5}) \times (1 - \sqrt{3/5})/4$	$100/324 \times (1 - \sqrt{3/5})/8$
	$(1 - \sqrt{3/5})/2$	$(1 + \sqrt{3/5}) \times (1 + \sqrt{3/5})/4$	$100/324 \times (1 + \sqrt{3/5})/8$
	$(1 - \sqrt{3/5})/2$	$(1 + \sqrt{3/5}) \times (1 - \sqrt{3/5})/4$	$100/324 \times (1 + \sqrt{3/5})/8$
	1/2	$(1 + \sqrt{3/5})/4$	5/81
	1/2	$(1 - \sqrt{3/5})/4$	5/81
	$(1 + \sqrt{3/5})/2$	$(1 - \sqrt{3/5})/4$	$5/81 \times (1 - \sqrt{3/5})$
	$(1 - \sqrt{3/5})/2$	$(1 + \sqrt{3/5})/4$	$5/81 \times (1 + \sqrt{3/5})$

注：表中括号内数字为对应高斯点数下计算精度。

2D 四边形单元高斯积分点及权重

附录-表 A.2 2D 四边形单元 Gauss 积分表

	Number of Points	Location ξ	Location η	Weight w
4(7)		$-\sqrt{1/3}$	$-\sqrt{1/3}$	1
		$\sqrt{1/3}$	$-\sqrt{1/3}$	1
		$-\sqrt{1/3}$	$\sqrt{1/3}$	1
		$\sqrt{1/3}$	$\sqrt{1/3}$	1
9(17)		$-\sqrt{3/5}$	$-\sqrt{3/5}$	$25/81$
		0	$-\sqrt{3/5}$	$40/81$
		$\sqrt{3/5}$	$-\sqrt{3/5}$	$25/81$
		$-\sqrt{3/5}$	0	$40/81$
		0	0	$64/81$
		$\sqrt{3/5}$	0	$40/81$
		$-\sqrt{3/5}$	$\sqrt{3/5}$	$25/81$
		0	$\sqrt{3/5}$	$40/81$
		$\sqrt{3/5}$	$\sqrt{3/5}$	$25/81$

B

本书所述有限元方法参考程序（Python）

参考程序使用 Python 语言编写，主要使用了 numpy 科学计算库和 matplotlib 绘图库。附录中提供了本书中所述问题模型基础程序，其余问题模型程序可由所提供基础程序修改得到，参考程序可在 Windows、Linux 等系统的 Python 环境下运行。

参考程序中主要程序文件及其功能如下：

- ▶ `nD_FEM.py`: n 维有限元主程序，执行有限元方法求解流程。
- ▶ `Mesh.py`: 网格划分、边界条件定义。
- ▶ `linearSystem.py`: 刚度矩阵、载荷向量组装，边界条件处理，线性系统求解，误差计算。
- ▶ `functions.py`: 问题模型中涉及各函数参数。
- ▶ `postProcess.py`: 仿真结果可视化。

B.1 第二章参考程序

B.1.1 1D_FEM.py

```
1 import Mesh
2 import linearSystem
3 import postProcess
4 import argparse
5
```

```
6  """
7  运行指令示范:
8  1. 默认参数: python .\1D_FEM.py
9  2. 线性元 + Dirichlet 边界条件: python .\1D_FEM.py --femInfo Linear,
   Dirichlet,Dirichlet,4
10 3. 2 次元 + Neumann 边界条件: python .\1D_FEM.py --femInfo Quadratic,
   Dirichlet,Neumann,4
11 4. 2 次元 + Robin 边界条件: python .\1D_FEM.py --femInfo Quadratic,Robin
   ,Dirichlet,4
12 """
13
14 if __name__ == "__main__":
15     parser = argparse.ArgumentParser(prog='1D_FEM',description='Finite
   Element Method For 1D
   Template')
16     parser.add_argument("--femInfo", default="Linear,Dirichlet,
   Dirichlet,4")
17     inputInfo = parser.parse_args().femInfo
18     inputInfo = inputInfo.split(',')
19     basisType = inputInfo[0]
20     BC = [inputInfo[1], inputInfo[2]]
21     cellN = int(inputInfo[3])
22
23     mesh = Mesh.Mesh(cellN, basisType)
24     mesh.setExtents(0, 1, BC=BC)
25
26     print("P\n", mesh.P)
27     print("\nT\n", mesh.T)
28     print("\nPb\n", mesh.Pb)
29     print("\nTb\n", mesh.Tb)
30     print("\nBoundary Condition\n", mesh.BC)
31
32     system = linearSystem.LinearSystem(mesh, 4)
33     system.assembleMtxA()
34     system.assembleVectorB()
35     print("\nmatrix\n", system mtxA)
36     print("\nvector\n", system vecB)
37
38     system.boundaryTreatment()
39     print("\nmatrix after BC treatment\n", system mtxA)
40     print("\nvector after BC treatment\n", system vecB)
41
42     system.solveLinearSystem()
43     system.errorCalculation()
```

```
44     print("\nresult\n", system.res)
45
46     print("\nmaxError = %.4e" % system.maxError)
47     print("Linfinity = %.4e" % system.Linfinity)
48     print("L2 = %.4e" % system.L2)
49     print("H1 = %.4e" % system.H1)
50
51     postProcess.linePlot(mesh, system)
```

B.1.2 Mesh.py

```
1 import numpy as np
2
3 class Mesh:
4     def __init__(self, NE, bTp):
5         self.x0 = 0
6         self.xm = 0
7         self.dh = 0
8
9         self.NE = NE
10        self.NN = NE + 1
11
12        self.bTp = bTp
13        self.P = np.zeros(NE + 1)
14        self.T = np.zeros((2, NE), dtype=int)
15
16        self.BC = np.zeros((3, 2), dtype=int)
17
18        if bTp == "Linear":
19            print("\n<< Using 1D Linear Finite Element Method >>")
20            self.Pb = self.P
21            self.Tb = self.T
22        elif bTp == "Quadratic":
23            print("\n<< Using 1D Quadratic Finite Element Method >>")
24            self.Pb = np.zeros(2 * NE + 1)
25            self.Tb = np.zeros((3, NE), dtype=int)
26
27
28    def setExtents(self, x0, xm, BC = ["Dirichlet", "Dirichlet"]):
29        self.x0 = x0
30        self.xm = xm
31        self.dh = (self.xm - self.x0) / self.NE
```

```
32     self.generatePT()
33     self.generatePbTb()
34     self.boundaryGeneration(BC)
35
36
37     def generatePT(self):
38         for i in range(self.NN):
39             self.P[i] = self.x0 + i * self.dh
40         for j in range(self.NE):
41             self.T[0][j] = j
42             self.T[1][j] = j + 1
43
44
45     def generatePbTb(self):
46         if self.bTp == "Linear":
47             self.Pb = self.P
48             self.Tb = self.T
49         elif self.bTp == "Quadratic":
50             for i in range(self.Pb.shape[0]):
51                 self.Pb[i] = self.x0 + 0.5 * i * self.dh
52             for i in range(self.NE):
53                 self.Tb[0][i] = 2 * i
54                 self.Tb[1][i] = 2 * (i + 1)
55                 self.Tb[2][i] = 2 * i + 1
56
57
58     def boundaryGeneration(self, BC):
59         self.BC[1][0] = 0
60         self.BC[1][1] = self.Pb.shape[0] - 1
61
62         # 左边界: 默认为第 1 类边界
63         if BC[0] == "Neumann":
64             self.BC[0][0] = -2
65             self.BC[2][0] = -1
66         elif BC[0] == "Robin":
67             self.BC[0][0] = -3
68             self.BC[2][0] = -1
69         else:
70             self.BC[0][0] = -1
71             self.BC[2][0] = 0
72
73         # 右边边界: 默认为第 1 类边界
74         if BC[1] == "Neumann":
75             self.BC[0][1] = -2
```

```
76         self.BC[2][1] = 1
77     elif BC[1] == "Robin":
78         self.BC[0][1] = -3
79         self.BC[2][1] = -1
80     else:
81         self.BC[0][1] = -1
82         self.BC[2][1] = 0
```

B.1.3 linearSystem.py

```
1 import numpy as np
2 from math import *
3 import functions as fun
4 np.set_printoptions(precision=6, suppress=True)
5
6 class LinearSystem:
7     def __init__(self, mesh, gaussPointNum) -> None:
8         self.mesh = mesh
9         self.gPtN = gaussPointNum
10
11     self.gaussWeightRef = np.zeros(gaussPointNum)
12     self.gaussPointRef = np.zeros(gaussPointNum)
13
14     self.mtxA = np.zeros((mesh.Pb.shape[0], mesh.Pb.shape[0]))
15     self.vecB = np.zeros(mesh.Pb.shape[0])
16     self.res = np.zeros(mesh.Pb.shape[0])
17
18     self.xAnalytic = [mesh.x0 + i *
19                         (mesh.xm - mesh.x0) / 10000 for i in range(
20                                         10001)]
21     self.resAnalytic = [fun.analyticSolution(i) for i in self.
22                         xAnalytic]
23
24     self.maxError = 0
25     self.Linfinity = 0
26     self.L2 = 0
27     self.H1 = 0
28
29     if mesh.bTp == "Linear":
30         self.baseNum = 2
31     elif mesh.bTp == "Quadratic":
32         self.baseNum = 3
```

```
31     self.gaussWeightAndPointRef()  
32  
33  
34     def gaussWeightAndPointRef(self):  
35         if self.gPtN == 2:  
36             self.gaussWeightRef[0] = 1.0  
37             self.gaussWeightRef[1] = 1.0  
38  
39             self.gaussPointRef[0] = -1.0 / sqrt(3)  
40             self.gaussPointRef[1] = +1.0 / sqrt(3)  
41         elif self.gPtN == 4:  
42             self.gaussWeightRef[0] = 0.3478548451  
43             self.gaussWeightRef[1] = 0.3478548451  
44             self.gaussWeightRef[2] = 0.6521451549  
45             self.gaussWeightRef[3] = 0.6521451549  
46  
47             self.gaussPointRef[0] = +0.8611363116  
48             self.gaussPointRef[1] = -0.8611363116  
49             self.gaussPointRef[2] = +0.3399810436  
50             self.gaussPointRef[3] = -0.3399810436  
51         elif self.gPtN == 8:  
52             self.gaussWeightRef[0] = 0.1012285363  
53             self.gaussWeightRef[1] = 0.1012285363  
54             self.gaussWeightRef[2] = 0.2223810345  
55             self.gaussWeightRef[3] = 0.2223810345  
56             self.gaussWeightRef[4] = 0.3137066459  
57             self.gaussWeightRef[5] = 0.3137066459  
58             self.gaussWeightRef[6] = 0.3626837834  
59             self.gaussWeightRef[7] = 0.3626837834  
60  
61             self.gaussPointRef[0] = +0.9602898565  
62             self.gaussPointRef[1] = -0.9602898565  
63             self.gaussPointRef[2] = +0.7966664774  
64             self.gaussPointRef[3] = -0.7966664774  
65             self.gaussPointRef[4] = +0.5255324099  
66             self.gaussPointRef[5] = -0.5255324099  
67             self.gaussPointRef[6] = +0.1834346425  
68             self.gaussPointRef[7] = -0.1834346425  
69  
70  
71     def refBaseFunction(self, x, ver, dD, bIx):  
72         if self.mesh.bTp == "Linear":  
73             if dD == 0:  
74                 if bIx == 0:
```

```
75         return (ver[1] - x) / (ver[1] - ver[0])
76     elif bIx == 1:
77         return (x - ver[0]) / (ver[1] - ver[0])
78     elif dD == 1:
79         if bIx == 0:
80             return 1.0 / (ver[0] - ver[1])
81         elif bIx == 1:
82             return 1.0 / (ver[1] - ver[0])
83     else:
84         return 0
85     elif self.mesh.bTp == "Quadratic":
86         upper = ver[1]
87         lower = ver[0]
88         res = 0
89         bottom = (upper - lower) ** 2
90         if dD == 0:
91             if bIx == 0:
92                 res = 2.0 * pow(x, 2) - (lower + 3.0 * upper) * \
93                     x + pow(upper, 2) + lower * upper
94             elif bIx == 1:
95                 res = 2.0 * pow(x, 2) - (3.0 * lower + upper) * \
96                     x + pow(lower, 2) + lower * upper
97             elif bIx == 2:
98                 res = -4.0 * pow(x, 2) + 4.0 * \
99                     (lower + upper) * x - 4.0 * lower * upper
100        elif dD == 1:
101            if bIx == 0:
102                res = 4.0 * x - (lower + 3.0 * upper)
103            elif bIx == 1:
104                res = 4.0 * x - (3.0 * lower + upper)
105            elif bIx == 2:
106                res = -8.0 * x + 4.0 * (lower + upper)
107        elif dD == 2:
108            if bIx == 0:
109                res = 4.0
110            elif bIx == 1:
111                res = 4.0
112            elif bIx == 2:
113                res = -8.0
114        elif dD > 2:
115            res = 0
116
117        res /= bottom
118        return res
```

```
119
120
121     def gaussInteTrialAndTest(self, ver, bIx, function):
122         gaussWeightLocal = np.zeros(self.gPtN)
123         gaussPointLocal = np.zeros(self.gPtN)
124         res = 0
125
126         for i in range(self.gPtN):
127             gaussWeightLocal[i] = (ver[1] - ver[0]) * \
128                 self.gaussWeightRef[i] / 2.0
129             gaussPointLocal[i] = (
130                 ver[1] - ver[0]) * self.gaussPointRef[i] / 2.0 + (ver[
131                     0] + ver[1]) / 2.0
132             res += (gaussWeightLocal[i] * function(gaussPointLocal[i])
133                     * self.refBaseFunction(
134                         gaussPointLocal[i], ver, 1, bIx[0]) * self.
135                         refBaseFunction(
136                             gaussPointLocal[i], ver, 1,
137                             bIx[1]))
138
139         return res
140
141
142
143     def assembleMtxA(self):
144         ver = [0, 0]
145         for n in range(self.mesh.NE):
146             for i in range(len(ver)):
147                 ver[i] = self.mesh.P[self.mesh.T[i][n]]
148             if ver[0] > ver[1]:
149                 ver[0], ver[1] = ver[1], ver[0]
150             for alpha in range(self.baseNum):
151                 for beta in range(self.baseNum):
152                     tmp = self.gaussInteTrialAndTest(
153                         ver, [alpha, beta], fun.funA)
154                     self.mtxA[self.mesh.Tb[beta][n]
155                         ][self.mesh.Tb[alpha][n]] += tmp
156
157
158     def gaussInteTest(self, ver, bIx, function):
159         gaussWeightLocal = np.zeros(self.gPtN)
160         gaussPointLocal = np.zeros(self.gPtN)
161         res = 0
162         for i in range(self.gPtN):
163             gaussWeightLocal[i] = (ver[1] - ver[0]) * self.
164                 gaussWeightRef[i] / 2.0
```

```
157     gaussPointLocal[i] = (ver[1] - ver[0]) * self.
158                               gaussPointRef[i] / 2.0 + (
159                                 ver[0] + ver[1]) / 2.0
160
161     res += (gaussWeightLocal[i] * function(gaussPointLocal[i])
162               * self.refBaseFunction(
163                 gaussPointLocal[i], ver, 0,
164                 bIx))
165
166     return res
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
```

```
gaussPointLocal[i] = (ver[1] - ver[0]) * self.
gaussPointRef[i] / 2.0 + (
ver[0] + ver[1]) / 2.0
res += (gaussWeightLocal[i] * function(gaussPointLocal[i])
* self.refBaseFunction(
gaussPointLocal[i], ver, 0,
bIx))

return res

def assembleVectorB(self):
    ver = [0, 0]
    for n in range(self.mesh.NE):
        for i in range(len(ver)):
            ver[i] = self.mesh.P[self.mesh.T[i][n]]
        if ver[0] > ver[1]:
            ver[0], ver[1] = ver[1], ver[0]

        for beta in range(self.baseNum):
            tmp = self.gaussInteTest(ver, beta, fun.funF)
            self.vecB[self.mesh.Tb[beta][n]] += tmp

def boundaryTreatment(self):
    for k in range(self.mesh.BC.shape[1]):
        i = self.mesh.BC[1][k]
        if self.mesh.BC[0][k] == -1:
            print("Treat < Dirichlet > Boundary Condition: %d" % i)
            self.mtxA[i, :] = 0
            self.mtxA[i][i] = 1
            self.vecB[i] = fun.funG(self.mesh.Pb[i])
        elif self.mesh.BC[0][k] == -2:
            print("Treat < Neumann > Boundary Condition: %d" % i)
            self.vecB[i] += self.mesh.BC[2][k] * \
                fun.funNeumann(self.mesh.Pb[i], self.mesh.BC)
        elif self.mesh.BC[0][k] == -3:
            print("Treat < Robin > Boundary Condition: %d" % i)
            self.vecB[i] += self.mesh.BC[2][k] * \
                fun.funNeumann2(self.mesh.Pb[i], self.mesh.BC)
            self.mtxA[i][i] += self.mesh.BC[2][k] * \
                fun.funRobin(self.mesh.Pb[i], self.mesh.BC)
```

```

195     def solveLinearSystem(self):
196         self.res = np.linalg.solve(self mtxA, self.vecB)
197
198     """
199     误差分析
200     """
201
202     def infinityFeRes(self, x, uhLocal, ver, dD):
203         res = 0
204         for i in range(self.baseNum):
205             res += uhLocal[i] * self.refBaseFunction(x, ver, dD, i)
206         return res
207
208
209     def gaussIntegralFeRes(self, uhLocal, ver, function, dD):
210         res = 0
211         gaussWeightLocal = np.zeros(self.gPtN)
212         gaussPointLocal = np.zeros(self.gPtN)
213         for i in range(self.gPtN):
214             gaussWeightLocal[i] = (ver[1] - ver[0]) * \
215                 self.gaussWeightRef[i] / 2.0
216             gaussPointLocal[i] = (
217                 ver[1] - ver[0]) * self.gaussPointRef[i] / 2.0 + (ver[
218                     0] + ver[1]) / 2.0
219             res += gaussWeightLocal[i] * pow(function(gaussPointLocal[
220                         i]) -
221                                         self.infinityFeRes(
222                                         gaussPointLocal[i], uhLocal,
223                                         ver, dD), 2)
224
225         return res
226
227
228     def maxErrorCompute(self):
229         res = abs(fun.analyticSolution(self.mesh.Pb[0]) - self.res[0])
230         for i in range(1, self.mesh.Pb.shape[0]):
231             tmp = abs(fun.analyticSolution(self.mesh.Pb[i]) - self.res
232                         [i])
233             if tmp > res:
234                 res = tmp
235         return res
236
237
238     def errorInfinityNorm(self, dD):
239         gaussPointLocal = np.zeros(self.gPtN)

```

```
234     ver = [0, 0]
235     res = 0
236     uhLocal = [0 for i in range(self.baseNum)]
237     for n in range(self.mesh.NE):
238         for i in range(len(ver)):
239             ver[i] = self.mesh.P[self.mesh.T[i][n]]
240         if ver[0] > ver[1]:
241             ver[0], ver[1] = ver[1], ver[0]
242         for i in range(self.baseNum):
243             uhLocal[i] = self.res[self.mesh.Tb[i][n]]
244         tmp = 0
245         for i in range(self.gPtN):
246             gaussPointLocal[i] = (
247                 ver[1] - ver[0]) * self.gaussPointRef[i] / 2.0 + (
248                     ver[0] + ver[1]) / 2.0
249             value = abs(fun.analyticSolution(
250                 gaussPointLocal[i]) - self.infinityFeRes(
251                     gaussPointLocal[i], uhLocal,
252                     ver, dD))
253             if tmp < value:
254                 tmp = value
255             if res < tmp:
256                 res = tmp
257     return res
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
def L2AndH1NormError(self, function, dD):
    ver = [0, 0]
    uhLocal = [0 for i in range(self.baseNum)]
    res = 0
    for n in range(self.mesh.NE):
        for i in range(len(ver)):
            ver[i] = self.mesh.P[self.mesh.T[i][n]]
        if ver[0] > ver[1]:
            ver[0], ver[1] = ver[1], ver[0]
        for i in range(self.baseNum):
            uhLocal[i] = self.res[self.mesh.Tb[i][n]]
        res += self.gaussIntegralFeRes(uhLocal, ver, function, dD)
    res = sqrt(res)
    return res

def errorCalculation(self):
    self.maxError = self.maxErrorCompute()
```

```
275     self.Linfinity = self.errorInfinityNorm(0)
276     self.L2 = self.L2AndH1NormError(fun.analyticSolution, 0)
277     self.H1 = self.L2AndH1NormError(fun.solutionDerivative, 1)
```

B.1.4 functions.py

```
1 import Mesh
2 from math import *
3
4 def funA(x):
5     return exp(x)
6
7
8 def funF(x):
9     return -exp(x) * (cos(x) - 2 * sin(x) - x * cos(x) - x * sin(x))
10
11
12 def funG(x):
13     left = 0
14     right = 1
15     if x == left:
16         return 0
17     elif x == right:
18         return cos(1)
19     return 0
20
21
22 def analyticSolution(x):
23     return x * cos(x)
24
25
26 def solutionDerivative(x):
27     return cos(x) - x * sin(x)
28
29
30 def funNeumann(x, BC):
31     res = 0
32     if BC[0][1] == -2:
33         res = funA(x) * solutionDerivative(x)
34     return res
35
36
```

```

37 def funRobin(x, BC):
38     res = 0
39     if BC[0][0] == -3:
40         res = 1.0
41     return res
42
43
44 def funNeumann2(x, BC):
45     res = 0
46     if BC[0][0] == -3:
47         res = funA(x) * solutionDerivative(x) + funRobin(x, BC) *
48                                         analyticSolution(x)
49     return res

```

B.1.5 postProcess.py

```

1 import matplotlib.pyplot as plt
2 fontStyle = {'family': 'SimHei', 'style': 'italic', 'weight': 'normal',
3               'size': 20}
4
5
6 def linePlot(mesh, system):
7     fig = plt.figure()
8     sub = fig.add_subplot(111)
9     sub.plot(mesh.Pb, system.res, "r*-", label="FEM")
10    sub.plot(system.xAnalytic, system.resAnalytic, "b--", label="Exact")
11    sub.set_xlabel("x", fontproperties=fontStyle)
12    sub.set_ylabel("u(x)", fontproperties=fontStyle)
13    plt.xticks(fontsize=18)
14    plt.yticks(fontsize=18)
15    plt.tick_params("both", which='major', length=5, width=1.5, colors=
16                           'k', direction='in')
17    plt.tick_params("both", which='minor', length=3, width=1.5, colors=
18                           'r', direction='in')
19    plt.legend(loc=0, prop=fontStyle)
20    plt.grid()
21    errorInfo = "maxError = %e\nL∞ = %e\nL2 = %e\nH1 = %e" % (system.
22                           maxError, system.Linfinity,
23                           system.L2, system.H1)
24    sub.text(0.4, 0.2, errorInfo, fontsize=15, color="k",
25            horizontalalignment="left",
26            verticalalignment="center",
27            style="italic")

```

```

        transform=sub.transAxes,
        rotation=0, bbox=dict(
            edgecolor="r", facecolor="w",
            alpha=0.9, linestyle="--",
            lw=1.5))
19    plt.annotate("notation", xy=(3, 0.1), xytext=(3.5, 0.25),
                 arrowprops=dict(facecolor="k",
                                 shrink=0.05), fontsize=18
                )
20    plt.tight_layout()
21    plt.show()

```

B.2 第三章参考程序

本章为包含 Robin 边界条件的有限元参考程序，该边界包含了 Dirichlet 与 Neumann 边界条件的处理，读者可类比修改得到包含其他边界类型的有限元程序。

B.2.1 2D_FEM.py

```

1 import Mesh
2 import linearSystem
3 import postProcess
4 import argparse
5 import time
6
7 """
8 运行指令示范：
9 python .\2D_FEM.py --femInfo LinearTri,2,3
10 python .\2D_FEM.py --femInfo QuadraticTri,2,3
11 """
12
13 if __name__ == "__main__":
14     parser = argparse.ArgumentParser(prog='2D_FEM',description='Finite
15                                     Element Method For 2D
16                                     Template')
17
18     parser.add_argument("--femInfo", default="QuadraticTri,2,3")
19     args = parser.parse_args()
20     inputInfo = parser.parse_args().femInfo
21     inputInfo = inputInfo.split(',')
22     basisType = inputInfo[0]

```

```
21  cellN = [int(inputInfo[1]), int(inputInfo[2])]  
22  mesh = Mesh.Mesh(cellN, basisType)  
23  mesh.setExtents([-1, -1], [1, 1])  
24  
25  print("P\n", mesh.P)  
26  print("T\n", mesh.T)  
27  print("Pb\n", mesh.Pb)  
28  print("Tb\n", mesh.Tb)  
29  print("----> boundary Condition <---")  
30  print("Boundary Nodes Condition\n", mesh.nodeBC[:, 0], "\n", mesh.  
           nodeBC[:, 1])  
31  print("\nBoundary Edges Condition\n", mesh.edgeBC[:, 0], "\n",  
           mesh.edgeBC[:, 1], "\n",  
           mesh.edgeBC[:, 2], "\n",  
           mesh.edgeBC[:, 3])  
32  
33  system = linearSystem.LinearSystem(mesh, 9)  
34  system.assembleMtxA2()  
35  system.assembleVectorB()  
36  print("----> before boundary treatment <---")  
37  print("matrix\n", system mtxA)  
38  print("vector\n", system vecB)  
39  
40  system.boundaryTreatment()  
41  print("----> after boundary treatment <---")  
42  print("matrix\n", system mtxA)  
43  print("vector\n", system vecB)  
44  
45  timeStart = time.time()  
46  system.solveLinearSystem()  
47  system.errorCalculation()  
48  timeEnd = time.time()  
49  print("\ncalculationTime = %f s\n" % (timeEnd - timeStart))  
50  
51  print("maxError = %e" % system.maxError)  
52  print("Linfinity = %e" % system.Linfinity)  
53  print("L2 = %e" % system.L2)  
54  print("H1 = %e" % system.H1)  
55  
56  print("result\n", system.res)  
57  
58  postProcess.PlotResNoMesh(mesh, system)  
59  postProcess.PlotAnalyticalRes(mesh)
```

B.2.2 Mesh.py

```

1 import numpy as np
2
3 class Mesh:
4     def __init__(self, NE, bTp):
5         self.x0 = [0, 0]
6         self.xm = [0, 0]
7         self.dh = [0, 0]
8
9         self.NE = NE
10        self.NN = [NE[0] + 1, NE[1] + 1]
11
12        self.P = np.zeros((NE[0] + 1) * (NE[1] + 1), 2)
13
14        self.nodeBC = np.zeros(2 * (NE[0] + NE[1]), 2, dtype=int)
15
16        self.edgeBC = np.zeros(2 * (NE[0] + NE[1]), 4, dtype=int)
17
18        self.bTp = bTp
19
20        if bTp == "LinearTri":
21            print("\n<< Using 2D Linear Triangular Finite Element
22                  Method >>")
23
24            self.T = np.zeros(2 * NE[0] * NE[1], 3, dtype=int)
25
26            self.Pb = self.P
27            self.Tb = self.T
28        elif bTp == "QuadraticTri":
29            print("\n<< Using 2D Quadratic Triangular Finite Element
30                  Method >>")
31
32            self.T = np.zeros(2 * NE[0] * NE[1], 3, dtype=int)
33
34            self.Pb = np.zeros((2 * NE[0] + 1) * (2 * NE[1] + 1), 2)
35            self.Tb = np.zeros(2 * NE[0] * NE[1], 6, dtype=int)
36
37            self.nodeBC = np.zeros(4 * (NE[0] + NE[1]), 3, dtype=int
38                                )
38
39        def setExtents(self, x0, xm):

```

```
39     self.x0 = x0
40     self.xm = xm
41     for i in range(len(self.dh)):
42         self.dh[i] = (self.xm[i] - self.x0[i]) / self.NE[i]
43
44     self.generatePTVertical()
45     self.generatePbTb()
46     self.boundaryGeneration()
47
48
49     def generatePTVertical(self):
50         for i in range(self.NN[0]):
51             for j in range(self.NN[1]):
52                 self.P[i * self.NN[1] + j][0] = self.x0[0] + i * self.
53                                         dh[0]
54                 self.P[i * self.NN[1] + j][1] = self.x0[1] + j * self.
55                                         dh[1]
56
57
58         if self.bTp == "LinearTri" or self.bTp == "QuadraticTri":
59             tmp = np.zeros((self.NN[0], self.NN[1]))
60
61
62             for i in range(self.NN[0]):
63                 for j in range(self.NN[1]):
64                     tmp[i][j] = i * self.NN[1] + j
65
66
67             for cellI in range(1, self.NE[0] * self.NE[1] + 1):
68                 row, col = 0, 0
69                 if cellI % self.NE[1] == 0:
70                     row = self.NE[1]
71                     col = cellI // self.NE[1]
72                 else:
73                     row = cellI % self.NE[1]
74                     col = cellI // self.NE[1] + 1
75
76
77                 if row - 1 < 0 or col - 1 < 0:
78                     return
79
80
81                 if row >= 1 and col >= 1:
82                     self.T[2 * (cellI - 1)][0] = tmp[col - 1][row - 1]
83                     self.T[2 * (cellI - 1)][1] = tmp[col][row - 1]
84                     self.T[2 * (cellI - 1)][2] = tmp[col - 1][row]
85
86
87                     self.T[2 * cellI - 1][0] = tmp[col - 1][row]
88                     self.T[2 * cellI - 1][1] = tmp[col][row - 1]
```

```

81             self.T[2 * cellI - 1][2] = tmp[col][row]
82
83
84     def generatePbTb(self):
85         if self.bTp == "LinearTri":
86             self.Pb = self.P
87             self.Tb = self.T
88         elif self.bTp == "QuadraticTri":
89             dh = [dx / 2 for dx in self.dh]
90             # 纵向分布
91             for nodeI in range(1, self.Pb.shape[0] + 1):
92                 if nodeI % (2 * self.NE[1] + 1) == 0:
93                     self.Pb[nodeI - 1][0] = self.x0[0] + (nodeI / (2 *
94                                         self.NE[1] + 1) - 1) * dh[0]
95
96                     ]
97                     self.Pb[nodeI - 1][1] = self.xm[1]
98                 else:
99                     self.Pb[nodeI - 1][0] = self.x0[0] + nodeI // (2 *
100                                         self.NE[1] + 1) * dh[0]
101                     self.Pb[nodeI - 1][1] = self.x0[1] + (nodeI % (2 *
102                                         self.NE[1] + 1) - 1) * dh[1]
103
104
105             tmp = np.zeros((2 * self.NE[0] + 1, 2 * self.NE[1] + 1))
106             for i in range(tmp.shape[0]):
107                 for j in range(tmp.shape[1]):
108                     tmp[i][j] = i * (2 * self.NE[1] + 1) + j
109
110             for cellI in range(1, self.NE[0] * self.NE[1] + 1):
111                 row, col = 0, 0
112                 if cellI % self.NE[1] == 0:
113                     row = self.NE[1]
114                     col = cellI // self.NE[1]
115                 else:
116                     row = cellI % self.NE[1]
117                     col = cellI // self.NE[1] + 1
118
119                 if row - 1 < 0 or col - 1 < 0:
120                     return
121
122                 if row >= 1 and col >= 1:
123                     self.Tb[2 * (cellI - 1)][0] = tmp[2 * col - 2][2 *
124                                         row - 2]

```

```
118         self.Tb[2 * (cellI - 1)][1] = tmp[2 * col][2 * row
119                               - 2]
120         self.Tb[2 * (cellI - 1)][2] = tmp[2 * col - 2][2 *
121                               row]
122         self.Tb[2 * (cellI - 1)][3] = tmp[2 * col - 1][2 *
123                               row - 2]
124         self.Tb[2 * (cellI - 1)][4] = tmp[2 * col - 1][2 *
125                               row - 1]
126         self.Tb[2 * (cellI - 1)][5] = tmp[2 * col - 2][2 *
127                               row - 1]
128
129         self.Tb[2 * cellI - 1][0] = tmp[2 * col - 2][2 *
130                               row]
131         self.Tb[2 * cellI - 1][1] = tmp[2 * col][2 * row -
132                               2]
133         self.Tb[2 * cellI - 1][2] = tmp[2 * col][2 * row]
134         self.Tb[2 * cellI - 1][3] = tmp[2 * col - 1][2 *
135                               row - 1]
136         self.Tb[2 * cellI - 1][4] = tmp[2 * col][2 * row -
137                               1]
138         self.Tb[2 * cellI - 1][5] = tmp[2 * col - 1][2 *
139                               row]
140
141     def boundaryGeneration(self):
142         if self.bTp == "LinearTri":
143             NE = self.NE
144             NN = self.NN
145         elif self.bTp == "QuadraticTri":
146             NE = [2 * self.NE[0], 2 * self.NE[1]]
147             NN = [ne + 1 for ne in NE]
148
149             # ---> 边界网格节点的定义 <---#
150             # 将所有边界节点设置为 Dirichlet 边界类型
151             self.nodeBC[:, 0] = -1
152
153             # 针对本问题将下边界上节点设置为 Robin 类型
154             for i in range(1, NE[0]):
155                 self.nodeBC[i][0] = -3;
156
157             # bottom nodes
158             for i in range(NE[0]):
159                 self.nodeBC[i][1] = i * NN[1]
```

```

152     # right nodes
153     for i in range(NE[0], NE[0] + NE[1]):
154         self.nodeBC[i][1] = NE[0] * NN[1] + i - NE[0]
155
156     # top nodes
157     for i in range(NE[0] + NE[1], 2 * NE[0] + NE[1]):
158         self.nodeBC[i][1] = (2 * NE[0] + NE[1] + 1 - i) * NN[1] -
159                         1
160
161     # left nodes
162     for i in range(2 * NE[0] + NE[1], self.nodeBC.shape[0]):
163         self.nodeBC[i][1] = 2 * (NE[0] + NE[1]) - i
164
165     # ---> 边界网格边的定义 <---
166     # 初始化所有边界类型为 Dirichlet 类型边
167     self.edgeBC[:, 0] = -1
168
169     # 针对本问题将下边界的边设置为 Neumann 类型
170     NE = self.NE
171     for i in range(NE[0]):
172         self.edgeBC[i][0] = -3
173
174     # bottom edge
175     for i in range(1, NE[0] + 1):
176         self.edgeBC[i - 1][1] = (i - 1) * 2 * NE[1]
177         self.edgeBC[i - 1][2] = (i - 1) * (NE[1] + 1)
178         self.edgeBC[i - 1][3] = i * (NE[1] + 1)
179
180     # right edge
181     for i in range(NE[0] + 1, NE[0] + NE[1] + 1):
182         self.edgeBC[i - 1][1] = (NE[0] - 1) * 2 * NE[1] + 2 * (i -
183                                         NE[0]) - 1
184         self.edgeBC[i - 1][2] = NE[0] * (NE[1] + 1) + i - NE[0] -
185                                         1
186         self.edgeBC[i - 1][3] = NE[0] * (NE[1] + 1) + i - NE[0]
187
188     # top edge
189     for i in range(NE[0] + NE[1] + 1, 2 * NE[0] + NE[1] + 1):
190         self.edgeBC[i - 1][1] = (2 * NE[0] + NE[1] + 1 - i) * 2 *
191                         NE[1] - 1
192         self.edgeBC[i - 1][2] = (2 * NE[0] + NE[1] + 2 - i) * (NE[1] +
193                         1) - 1
194         self.edgeBC[i - 1][3] = (2 * NE[0] + NE[1] + 1 - i) * (NE[1] +
195                         1) - 1

```

```

190
191     # left edge
192     for i in range(2 * NE[0] + NE[1] + 1, self.edgeBC.shape[0] + 1
193                     ):
193         self.edgeBC[i - 1][1] = 2 * (2 * NE[0] + 2 * NE[1] + 1 - i
194                               ) - 2
194         self.edgeBC[i - 1][2] = 2 * NE[0] + 2 * NE[1] + 1 - i
195         self.edgeBC[i - 1][3] = 2 * NE[0] + 2 * NE[1] - i

```

B.2.3 linearSystem.py

```

1 import numpy as np
2 from math import *
3 import functions as fun
4 np.set_printoptions(precision=6, suppress=True)
5
6 class LinearSystem:
7     def __init__(self, mesh, gaussPointNum) -> None:
8         self.mesh = mesh
9         self.gPtN = gaussPointNum
10        self.gPtN1D = 4
11
12        self.gaussWeightRef = np.zeros(gaussPointNum)
13        self.gaussPointRef = np.zeros((gaussPointNum, 2))
14        self.gaussWeightRef1D = np.zeros(self.gPtN1D)
15        self.gaussPointRef1D = np.zeros(self.gPtN1D)
16
17        self.gaussWeightLocal = np.zeros(gaussPointNum)
18        self.gaussPointLocal = np.zeros((gaussPointNum, 2))
19        self.gaussWeightLocal1D = np.zeros(self.gPtN1D)
20        self.gaussPointLocal1D = np.zeros(self.gPtN1D)
21
22        self.Jacobian = np.zeros((2, 2))
23        self.JacobianInv = np.zeros((2, 2))
24        self.Jdet = 0
25
26        self mtxA = np.zeros((mesh.Pb.shape[0], mesh.Pb.shape[0]))
27        self vecB = np.zeros(mesh.Pb.shape[0])
28        self.res = np.zeros(mesh.Pb.shape[0])
29
30        self.maxError = 0
31        self.Linfinity = 0

```

```
32     self.L2 = 0
33     self.H1 = 0
34
35     if mesh.bTp == "LinearTri":
36         self.baseNum = 3
37     elif mesh.bTp == "LinearQua":
38         self.baseNum = 4
39     elif mesh.bTp == "QuadraticTri":
40         self.baseNum = 6
41     elif mesh.bTp == "QuadraticQua":
42         self.baseNum = 8
43
44     self.gaussWeightAndPointRef()
45     self.gaussWeightAndPointRef1D()
46
47
48 def gaussWeightAndPointRef(self):
49     if self.gPtN == 3:
50         self.gaussWeightRef[0] = 1 / 6
51         self.gaussWeightRef[1] = 1 / 6
52         self.gaussWeightRef[2] = 1 / 6
53
54         self.gaussPointRef[0][0] = 0.5
55         self.gaussPointRef[0][1] = 0
56
57         self.gaussPointRef[1][0] = 0.5
58         self.gaussPointRef[1][1] = 0.5
59
60         self.gaussPointRef[2][0] = 0
61         self.gaussPointRef[2][1] = 0.5
62     elif self.gPtN == 4:
63         self.gaussWeightRef[0] = (1 - 1 / sqrt(3)) / 8
64         self.gaussWeightRef[1] = (1 - 1 / sqrt(3)) / 8
65         self.gaussWeightRef[2] = (1 + 1 / sqrt(3)) / 8
66         self.gaussWeightRef[3] = (1 + 1 / sqrt(3.0)) / 8
67
68         self.gaussPointRef[0][0] = (1 / sqrt(3) + 1) / 2
69         self.gaussPointRef[0][1] = (
70             1 - 1 / sqrt(3) * (1 + 1 / sqrt(3)) / 4
71
72         self.gaussPointRef[1][0] = (1 / sqrt(3) + 1) / 2
73         self.gaussPointRef[1][1] = (
74             1 - 1 / sqrt(3) * (1 - 1 / sqrt(3)) / 4
75
```

```
76     self.gaussPointRef[2][0] = (-1 / sqrt(3) + 1) / 2
77     self.gaussPointRef[2][1] = (
78         1 + 1 / sqrt(3)) * (1 + 1 / sqrt(3)) / 4
79
80     self.gaussPointRef[3][0] = (-1 / sqrt(3) + 1) / 2
81     self.gaussPointRef[3][1] = (
82         1 + 1 / sqrt(3)) * (1 - 1 / sqrt(3)) / 4
83 elif self.gPtN == 9:
84     self.gaussWeightRef[0] = 64 / 81 * 1 / 8
85     self.gaussWeightRef[1] = 100 / 324 * (1 - sqrt(3 / 5)) / 8
86     self.gaussWeightRef[2] = 100 / 324 * (1 - sqrt(3 / 5)) / 8
87     self.gaussWeightRef[3] = 100 / 324 * (1 + sqrt(3 / 5)) / 8
88     self.gaussWeightRef[4] = 100 / 324 * (1 + sqrt(3 / 5)) / 8
89     self.gaussWeightRef[5] = 40 / 81 * 1 / 8
90     self.gaussWeightRef[6] = 40 / 81 * 1 / 8
91     self.gaussWeightRef[7] = 40 / 81 * (1 - sqrt(3 / 5)) / 8
92     self.gaussWeightRef[8] = 40 / 81 * (1 + sqrt(3 / 5)) / 8
93
94     self.gaussPointRef[0][0] = 1 / 2
95     self.gaussPointRef[0][1] = 1 / 4
96
97     self.gaussPointRef[1][0] = (1.0 + sqrt(3.0 / 5.0)) / 2.0
98     self.gaussPointRef[1][1] = (
99         1.0 - sqrt(3.0 / 5.0)) * (1.0 + sqrt(3.0 / 5.0)) / 4.0
100
101    self.gaussPointRef[2][0] = (1.0 + sqrt(3.0 / 5.0)) / 2.0
102    self.gaussPointRef[2][1] = (
103        1.0 - sqrt(3.0 / 5.0)) * (1.0 - sqrt(3.0 / 5.0)) / 4.0
104
105    self.gaussPointRef[3][0] = (1.0 - sqrt(3.0 / 5.0)) / 2.0
106    self.gaussPointRef[3][1] = (
107        1.0 + sqrt(3.0 / 5.0)) * (1.0 + sqrt(3.0 / 5.0)) / 4.0
108
109    self.gaussPointRef[4][0] = (1.0 - sqrt(3.0 / 5.0)) / 2.0
110    self.gaussPointRef[4][1] = (
111        1.0 + sqrt(3.0 / 5.0)) * (1.0 - sqrt(3.0 / 5.0)) / 4.0
112
113    self.gaussPointRef[5][0] = (1.0 + 0.0) / 2.0
114    self.gaussPointRef[5][1] = (
115        1.0 - 0.0) * (1.0 + sqrt(3.0 / 5.0)) / 4.0
116
117    self.gaussPointRef[6][0] = (1.0 + 0.0) / 2.0
118    self.gaussPointRef[6][1] = (
119        1.0 - 0.0) * (1.0 - sqrt(3.0 / 5.0)) / 4.0
```

```
120     self.gaussPointRef[7][0] = (1.0 + sqrt(3.0 / 5.0)) / 2.0
121     self.gaussPointRef[7][1] = (
122         1.0 - sqrt(3.0 / 5.0)) * (1.0 + 0.0) / 4.0
123
124
125     self.gaussPointRef[8][0] = (1.0 - sqrt(3.0 / 5.0)) / 2.0
126     self.gaussPointRef[8][1] = (
127         1.0 + sqrt(3.0 / 5.0)) * (1.0 + 0.0) / 4.0
128
129
130 def gaussWeightAndPointRef1D(self):
131     if self.gPtN1D == 2:
132         self.gaussWeightRef1D[0] = 1.0
133         self.gaussWeightRef1D[1] = 1.0
134
135         self.gaussPointRef1D[0] = -1.0 / sqrt(3)
136         self.gaussPointRef1D[1] = +1.0 / sqrt(3)
137     elif self.gPtN1D == 4:
138         self.gaussWeightRef1D[0] = 0.3478548451
139         self.gaussWeightRef1D[1] = 0.3478548451
140         self.gaussWeightRef1D[2] = 0.6521451549
141         self.gaussWeightRef1D[3] = 0.6521451549
142
143         self.gaussPointRef1D[0] = +0.8611363116
144         self.gaussPointRef1D[1] = -0.8611363116
145         self.gaussPointRef1D[2] = +0.3399810436
146         self.gaussPointRef1D[3] = -0.3399810436
147     elif self.gPtN1D == 8:
148         self.gaussWeightRef1D[0] = 0.1012285363
149         self.gaussWeightRef1D[1] = 0.1012285363
150         self.gaussWeightRef1D[2] = 0.2223810345
151         self.gaussWeightRef1D[3] = 0.2223810345
152         self.gaussWeightRef1D[4] = 0.3137066459
153         self.gaussWeightRef1D[5] = 0.3137066459
154         self.gaussWeightRef1D[6] = 0.3626837834
155         self.gaussWeightRef1D[7] = 0.3626837834
156
157         self.gaussPointRef1D[0] = +0.9602898565
158         self.gaussPointRef1D[1] = -0.9602898565
159         self.gaussPointRef1D[2] = +0.7966664774
160         self.gaussPointRef1D[3] = -0.7966664774
161         self.gaussPointRef1D[4] = +0.5255324099
162         self.gaussPointRef1D[5] = -0.5255324099
163         self.gaussPointRef1D[6] = +0.1834346425
```

```
164     self.gaussPointRef1D[7] = -0.1834346425
165
166
167     def gaussWeightAndPointLocal(self, ver):
168         x1, y1 = ver[0][0], ver[0][1]
169         x2, y2 = ver[1][0], ver[1][1]
170         x3, y3 = ver[2][0], ver[2][1]
171         self.Jdet = abs((x2 - x1) * (y3 - y1) - (x3 - x1) * (y2 - y1))
172
173         for i in range(self.gPtN):
174             self.gaussWeightLocal[i] = self.Jdet * self.gaussWeightRef
175                         [i]
176             self.gaussPointLocal[i][0] = x1 + (x2 - x1) * self.
177                         gaussPointRef[i][0] + (
178                 x3 - x1) * self.gaussPointRef[i][1]
179             self.gaussPointLocal[i][1] = y1 + (y2 - y1) * self.
180                         gaussPointRef[i][0] + (
181                 y3 - y1) * self.gaussPointRef[i][1]
182
183
184
185
186     def gaussWeightAndPointLocal1D(self, lower, upper):
187         self.gaussWeightLocal1D = (upper - lower) * self.
188                         gaussWeightRef1D / 2
189         self.gaussPointLocal1D = (upper - lower) * self.
190                         gaussPointRef1D / 2 + (upper
191                         + lower) / 2
192
193
194
195
196     def refBaseFunction(self, x, dD, bIx):
197         res = 0
198
199         if self.mesh.bTp == "LinearTri":
200
201             if dD[0] == 0 and dD[1] == 0:
202                 if bIx == 0:
203                     res = 1 - x[0] - x[1]
204                 elif bIx == 1:
205                     res = x[0]
206                 elif bIx == 2:
207                     res = x[1]
208             elif dD[0] == 1 and dD[1] == 0:
209                 if bIx == 0:
210                     res = -1
211                 elif bIx == 1:
212                     res = 1
213                 elif bIx == 2:
```

```

202             res = 0
203         elif dD[0] == 0 and dD[1] == 1:
204             if bIx == 0:
205                 res = -1
206             elif bIx == 1:
207                 res = 0
208             elif bIx == 2:
209                 res = 1
210         else:
211             return 0
212     return res
213 elif self.mesh.bTp == "QuadraticTri":
214     if dD[0] == 0 and dD[1] == 0:
215         if bIx == 0:
216             res = 2 * pow(x[0], 2) + 2 * pow(x[1], 2) + 4 * x[
217                 0] * x[1] - 3 * x[0] - 3 * x
218                 [1] + 1
219             elif bIx == 1:
220                 res = 2 * pow(x[0], 2) - x[0]
221             elif bIx == 2:
222                 res = 2 * pow(x[1], 2) - x[1]
223             elif bIx == 3:
224                 res = -4 * pow(x[0], 2) - 4 * x[0] * x[1] + 4 * x[
225                     0]
226             elif bIx == 4:
227                 res = 4 * x[0] * x[1]
228             elif bIx == 5:
229                 res = -4 * pow(x[1], 2) - 4 * x[0] * x[1] + 4 * x[
230                     1]
231     elif dD[0] == 1 and dD[1] == 0:
232         if bIx == 0:
233             res = 4 * x[0] + 4 * x[1] - 3
234         elif bIx == 1:
235             res = 4 * x[0] - 1
236         elif bIx == 2:
237             res = 0
238         elif bIx == 3:
239             res = -8 * x[0] - 4 * x[1] + 4
240         elif bIx == 4:
241             res = 4 * x[1]
242         elif bIx == 5:
243             res = -4 * x[1]
244     elif dD[0] == 0 and dD[1] == 1:
245         if bIx == 0:

```

```
242             res = 4 * x[1] + 4 * x[0] - 3
243     elif bIx == 1:
244         res = 0
245     elif bIx == 2:
246         res = 4 * x[1] - 1
247     elif bIx == 3:
248         res = -4 * x[0]
249     elif bIx == 4:
250         res = 4 * x[0]
251     elif bIx == 5:
252         res = -8 * x[1] - 4 * x[0] + 4
253 elif dD[0] == 2 and dD[1] == 0:
254     if bIx == 0:
255         res = 4
256     elif bIx == 1:
257         res = 4
258     elif bIx == 2:
259         res = 0
260     elif bIx == 3:
261         res = -8
262     elif bIx == 4:
263         res = 0
264     elif bIx == 5:
265         res = 0
266 elif dD[0] == 0 and dD[1] == 2:
267     if bIx == 0:
268         res = 4
269     elif bIx == 1:
270         res = 0
271     elif bIx == 2:
272         res = 4
273     elif bIx == 3:
274         res = 0
275     elif bIx == 4:
276         res = 0
277     elif bIx == 5:
278         res = -8
279 elif dD[0] == 1 and dD[1] == 1:
280     if bIx == 0:
281         res = 4
282     elif bIx == 1:
283         res = 0
284     elif bIx == 2:
285         res = 0
```

```

286         elif bIx == 3:
287             res = -4
288         elif bIx == 4:
289             res = 4
290         elif bIx == 5:
291             res = -4
292     return res
293 elif self.mesh.bTp == "LinearQua":
294     if dD[0] == 0 and dD[1] == 0:
295         if bIx == 0:
296             res = 0.25 * (1 - x[0]) * (1 - x[1])
297         elif bIx == 1:
298             res = 0.25 * (1 + x[0]) * (1 - x[1])
299         elif bIx == 2:
300             res = 0.25 * (1 + x[0]) * (1 + x[1])
301         elif bIx == 3:
302             res = 0.25 * (1 - x[0]) * (1 + x[1])
303     elif dD[0] == 1 and dD[1] == 0:
304         if bIx == 0:
305             res = -0.25 * (1 - x[1])
306         elif bIx == 1:
307             res = 0.25 * (1 - x[1])
308         elif bIx == 2:
309             res = 0.25 * (1 + x[1])
310         elif bIx == 3:
311             res = -0.25 * (1 + x[1])
312     elif dD[0] == 0 and dD[1] == 1:
313         if bIx == 0:
314             res = -0.25 * (1 - x[0])
315         elif bIx == 1:
316             res = -0.25 * (1 + x[0])
317         elif bIx == 2:
318             res = 0.25 * (1 + x[0])
319         elif bIx == 3:
320             res = 0.25 * (1 - x[0])
321     else:
322         return 0
323     return res
324
325
326 def localBaseFun(self, x, ver, dD, bIx):
327     J00 = ver[1][0] - ver[0][0]
328     J01 = ver[2][0] - ver[0][0]
329     J10 = ver[1][1] - ver[0][1]

```

```
330     J11 = ver[2][1] - ver[0][1]
331     Jdet = J00 * J11 - J01 * J10
332
333     xHat = (J11 * (x[0] - ver[0][0]) - J01 * (x[1] - ver[0][1])) /
334                               Jdet
335     yHat = (-J10 * (x[0] - ver[0][0]) + J00 * (x[1] - ver[0][1])) /
336                               Jdet
337
338     res = 0
339     if dD[0] == 0 and dD[1] == 0:
340         res = self.refBaseFunction([xHat, yHat], [0, 0], bIx)
341     elif dD[0] == 1 and dD[1] == 0:
342         res = (self.refBaseFunction([xHat, yHat], [
343             1, 0], bIx) * J11 + self.refBaseFunction([xHat,
344                                         yHat], [0, 1], bIx) * (-J10)
345                                         ) / Jdet
346     elif dD[0] == 0 and dD[1] == 1:
347         res = (self.refBaseFunction([xHat, yHat], [
348             1, 0], bIx) * (-J01) + self.refBaseFunction([xHat,
349                                         yHat], [0, 1], bIx) * J00) /
350                                         Jdet
351     elif dD[1] == 2 and dD[1] == 0:
352         res = (self.refBaseFunction([xHat, yHat], [2, 0], bIx) *
353                         pow(J11, 2) + self.
354                         refBaseFunction([xHat, yHat],
355                                         ,
356                                         [0, 2], bIx) * pow(J10, 2) + self.refBaseFunction([
357                                         xHat, yHat], [1, 1], bIx) *
358                                         (-2 * J10 * J11)) / pow(Jdet
359                                         , 2)
360     elif dD[0] == 0 and dD[1] == 2:
361         res = (self.refBaseFunction([xHat, yHat], [2, 0], bIx) *
362                         pow(J01, 2) + self.
363                         refBaseFun([xHat, yHat], [
364                                         0, 2], bIx) * pow(J00, 2) + self.refBaseFunction([
365                                         xHat, yHat], [1, 1], bIx) *
366                                         (-2 * J00 * J01)) / pow(Jdet
367                                         , 2)
368     elif dD[0] == 1 and dD[1] == 1:
369         res = (self.refBaseFunction([xHat, yHat], [2, 0], bIx) *
370                         (-J11 * J01) + self.
371                         refBaseFunction([xHat, yHat],
372                                         ,
373                                         [
```



```
393     tmp = self.gaussInteTrialAndTest(
394         [alpha, beta], 0, fun.funA) + self.
395             gaussInteTrialAndTest([alpha
396             , beta], 1, fun.funA)
397     row = self.mesh.Tb[n][beta]
398     col = self.mesh.Tb[n][alpha]
399     self.mtxA[row][col] += tmp
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
tmp = self.gaussInteTrialAndTest(
    [alpha, beta], 0, fun.funA) + self.
        gaussInteTrialAndTest([alpha
    , beta], 1, fun.funA)
row = self.mesh.Tb[n][beta]
col = self.mesh.Tb[n][alpha]
self.mtxA[row][col] += tmp

def gaussInteTrialAndTest2(self, ver, bIx, trialdD, testdD,
                           function):
    res = 0
    self.gaussWeightAndPointLocal(ver)
    for i in range(self.gPtN):
        res += self.gaussWeightLocal[i] * function(self.
            gaussPointLocal[i]) * self.
            localBaseFun(self.
            gaussPointLocal[i], ver,
            trialdD, bIx[0]) * self.
            localBaseFun(self.
            gaussPointLocal[i], ver,
            testdD, bIx[1])

    return res

def assembleMtxA2(self):
    for n in range(self.mesh.T.shape[0]):
        ver = np.zeros((self.mesh.T.shape[1], 2))
        for i in range(self.mesh.T.shape[1]):
            ver[i][0] = self.mesh.P[self.mesh.T[n][i]][0]
            ver[i][1] = self.mesh.P[self.mesh.T[n][i]][1]
        for alpha in range(self.baseNum):
            for beta in range(self.baseNum):
                tmp = self.gaussInteTrialAndTest2(ver, [alpha,
                    beta], [1, 0], [1, 0], fun.
                    funA) + self.
                    gaussInteTrialAndTest2(ver,
                    [alpha, beta], [0, 1], [0, 1],
                    fun.funA)
                row = self.mesh.Tb[n][beta]
                col = self.mesh.Tb[n][alpha]
                self.mtxA[row][col] += tmp
```

```

422     def gaussInteTest(self, ver, teBIdx, function):
423         res = 0
424         self.gaussWeightAndPointLocal(ver)
425         for i in range(self.gPtN):
426             tmp = function(self.gaussPointLocal[i])
427             tmpBase = self.localBaseFun(
428                 self.gaussPointLocal[i], ver, [0, 0], teBIdx)
429             res += self.gaussWeightLocal[i] * tmp * tmpBase
430         return res
431
432
433     def assembleVectorB(self):
434         ND = np.array([[-1, 1, 0], [-1, 0, 1]])
435
436         for n in range(self.mesh.Tb.shape[0]):
437             ver = np.zeros((self.mesh.Tb.shape[1], 2))
438             for i in range(self.mesh.Tb.shape[1]):
439                 ver[i][0] = self.mesh.Pb[self.mesh.Tb[n][i]][0]
440                 ver[i][1] = self.mesh.Pb[self.mesh.Tb[n][i]][1]
441
442             for beta in range(self.baseNum):
443                 tmp = self.gaussInteTest(ver, beta, fun.funF)
444                 self.vecB[self.mesh.Tb[n][beta]] += tmp
445
446
447     def gaussInteTrialTestLine(self, point1, point2, ver, bIx,
448                               function, trialdD, testdD):
449         res = 0
450         if point1[1] == point2[1]:
451             lower = min(point1[0], point2[0])
452             upper = max(point1[0], point2[0])
453             self.gaussWeightAndPointLocal1D(lower, upper)
454             for i in range(self.gPtN1D):
455                 res += self.gaussWeightLocal1D[i] * function([
456                     self.
457                     gaussPointLocal1D[i], point1
458                     [1]]) * self.localBaseFun([
459                         self.gaussPointLocal1D[i],
460                         point1[1]], ver, trialdD,
461                         bIx[0]) * self.localBaseFun([
462                             self.gaussPointLocal1D[i],
463                             point1[1]], ver, testdD, bIx
464                             [1])
465
466         return res
467
468

```



```
532
533     def errorInfinityNorm(self, dD):
534         res = 0
535         uhLocal = [0 for i in range(self.baseNum)]
536         for n in range(self.mesh.Tb.shape[0]):
537             ver = np.zeros((self.mesh.Tb.shape[1], 2))
538             for i in range(self.mesh.Tb.shape[1]):
539                 ver[i][0] = self.mesh.Pb[self.mesh.Tb[n][i]][0]
540                 ver[i][1] = self.mesh.Pb[self.mesh.Tb[n][i]][1]
541             self.gaussWeightAndPointLocal(ver)
542             for i in range(self.baseNum):
543                 uhLocal[i] = self.res[self.mesh.Tb[n][i]]
544             tmp = 0
545             for i in range(self.gPtN):
546                 value = abs(fun.analyticSolution(
547                     self.gaussPointLocal[i]) - self.infinityFeRes(self
548                     .gaussPointLocal[i], uhLocal
549                     , ver, dD))
550                 if tmp < value:
551                     tmp = value
552             if res < tmp:
553                 res = tmp
554
555         return res
556
557
558     def L2AndH1NormError(self, function, dD):
559         uhLocal = [0 for i in range(self.baseNum)]
560         res = 0
561         for n in range(self.mesh.Tb.shape[0]):
562             ver = np.zeros((self.mesh.Tb.shape[1], 2))
563             for i in range(self.mesh.Tb.shape[1]):
564                 ver[i][0] = self.mesh.Pb[self.mesh.Tb[n][i]][0]
565                 ver[i][1] = self.mesh.Pb[self.mesh.Tb[n][i]][1]
566             for i in range(self.baseNum):
567                 uhLocal[i] = self.res[self.mesh.Tb[n][i]]
568             res += self.gaussIntegralFeRes(uhLocal, ver, function, dD)
569         res = sqrt(res)
570         return res
571
572     def errorCalculation(self):
573         self.maxError = self.maxErrorCompute()
574         self.Linfinity = self.errorInfinityNorm([0, 0])
575         self.L2 = self.L2AndH1NormError(fun.analyticSolution, [0, 0])
```

```
574     self.H1 = sqrt(pow(self.L2AndH1NormError(fun.  
                      solutionDerivativeX, [1, 0])  
                      , 2) + pow(self.  
                      L2AndH1NormError(fun.  
                      solutionDerivativeY, [0, 1])  
                      , 2))
```

B.2.4 functions.py

```
1 import Mesh  
2 from math import *  
3  
4  
5 def funA(x):  
6     return 1  
7  
8  
9 def funF(x):  
10    return -2 * exp(x[0] + x[1])  
11  
12  
13 def funP(x):  
14    return 1  
15  
16  
17 def funQ(x):  
18    return 0  
19  
20  
21 def funG(x):  
22    return exp(x[0] + x[1])  
23  
24  
25 def analyticSolution(x):  
26    return exp(x[0] + x[1])  
27  
28  
29 def solutionDerivativeX(x):  
30    return exp(x[0] + x[1])  
31  
32  
33 def solutionDerivativeY(x):
```

34 **return exp(x[0] + x[1])**

B.2.5 postProcess.py

```
1 import matplotlib.pyplot as plt
2 import matplotlib.tri as mtri
3 import functions as fun
4 import numpy as np
5 fontStyle = {'family':'DejaVu Sans', 'style':'italic', 'weight':'
6               'normal', 'size':30}
7
8 def PlotResNoMesh(mesh, system):
9     fig = plt.figure(figsize=(7.8,7))
10    ax = fig.add_subplot(111)
11    plt.xlabel('x', fontproperties = fontStyle)
12    plt.ylabel('y', fontproperties = fontStyle)
13    plt.rcParams['xtick.direction'] = 'in'
14    plt.rcParams['ytick.direction'] = 'in'
15    tri = mtri.Triangulation(mesh.P[:, 0], mesh.P[:, 1], mesh.T)
16    if mesh.bTp == 'LinearTri':
17        plot = plt.tricontourf(tri, system.res, cmap= 'jet')
18    else:
19        x = [mesh.x0[0] + i * mesh.dh[0] / 2 for i in range(2 * mesh.
20                           NE[0] + 1)]
21        y = [mesh.x0[1] + j * mesh.dh[1] / 2 for j in range(2 * mesh.
22                           NE[1] + 1)]
23        res = []
24        for j in range(2 * mesh.NE[1] + 1):
25            tmp = []
26            for i in range(2 * mesh.NE[0] + 1):
27                tmp.append(system.res[j + i * (2 * mesh.NE[1] + 1)])
28            res.append(tmp)
29        x, y = np.meshgrid(x, y)
30        plot = plt.contourf(x, y, res,12, cmap='jet')
31        colorBar = fig.colorbar(plot)
32        colorBar.ax.set_title("u", fontproperties=fontStyle)
33        plt.autoscale(enable=True, axis='both', tight=True)
34        plt.tight_layout()
35        plt.show()
36        plt.ioff()
```

```

36 def PlotAnalyticalRes(mesh):
37     nCells = [200, 50]
38     x=np.linspace(-1, 1, nCells[0])
39     y=np.linspace(-1, 1, nCells[1])
40     res = []
41     for j in y:
42         tmp = []
43         for i in x:
44             value = fun.analyticSolution([i, j])
45             tmp.append(value)
46         res.append(tmp)
47
48     x,y=np.meshgrid(x,y)
49     fig, ax = plt.subplots(figsize=(8,7))
50     ax.tick_params("both", which='major', length=8, width=1.5, colors=
51                           'k', direction='in', pad=12)
52                           # "y", 'x',
53                           'both'
54     plot = plt.contourf(x,y,res,12,cmap='jet')
55     ax.set_xlabel('x', fontproperties=fontStyle)
56     ax.set_ylabel('y', fontproperties=fontStyle)
57     xTicks=np.linspace(-1,1,5)
58     yTicks=np.linspace(-1,1,5)
59     plt.xlim([min(xTicks), max(xTicks)])
60     plt.ylim([min(yTicks), max(yTicks)])
61     plt.xticks(xTicks)
62     plt.yticks(yTicks)
63     plt.setp(ax.get_xticklabels(), fontproperties=fontStyle)
64     plt.setp(ax.get_yticklabels(), fontproperties=fontStyle)
65     colorBar = fig.colorbar(plot)
66     colorBar.ax.set_title("uExact", fontproperties=fontStyle)
67     colorBar.ax.tick_params(direction='in', labelsize=fontStyle['size']
68                           ])
69     plt.tight_layout()
70     plt.show()

```

B.3 第四章参考程序

本章为包含张量问题模型的有限元参考程序，该问题模型为二阶张量问题，读者可类比修改得到其他张量问题模型的有限元程序。

B.3.1 2D_FEM.py

```
1 import Mesh
2 import linearSystem
3 import argparse
4 import json
5 import matplotlib.pyplot as plt
6 import postProcess
7 import functions
8
9 """
10 运行指令示范:
11 python .\2D_FEM.py
12 或者
13 python .\2D_FEM.py -type QuadraticTri -nCells 2 3 -simDomain 0 1 0 1 -
   showMeshInfo yes -
   showMtxAndVecBeforeBC yes -
   showMtxAndVecAfterBC yes -
   printRes yes -printError yes
   -plotMesh yes -
   plotResWithMesh yes -
   plotResNoMesh yes -
   plotAnalyticalRes yes -
   gaussPointNum 9
14 """
15
16 if __name__ == "__main__":
17     plt.ion()
18     simulationInfo = {
19         "description": "2DLinearElasticHe",
20         "basisType": "QuadraticTri",
21         "nCells": [2, 3],
22         "simDomain": [[0, 0], [1, 1]],
23         "leftBC": "Dirichlet",
24         "rightBC": "Dirichlet",
25         "showMeshInfo": "no",
26         "showMtxBeforeBC": "no",
27         "showVecBeforeBC": "no",
28         "showMtxAfterBC": "no",
29         "showVecAfterBC": "no",
30         "printRes": "no",
31         "printError": "yes",
32         "plotResNoMesh": "yes",
```

```
33     "plotAnalyticalRes": "yes",
34     "gaussPointNum": 9}
35
36 parser = argparse.ArgumentParser(prog='2D_FEM', description=
37                               simulationInfo["description"])
38
39 parser.add_argument('-type', choices=('LinearTri', 'LinearQua',
40                      'QuadraticTri', 'QuadraticQua',
41                      '), help='BasisType',
42                      default=simulationInfo['basisType'])
43
44 parser.add_argument('-nCells', nargs='+', type=int, help='No of
45                     cells in x and y direction',
46                     default=simulationInfo['nCells'])
47
48 parser.add_argument('-simDomain', type=list, help='simulation
49                     domain', default=
50                     simulationInfo['simDomain'])
51
52 parser.add_argument('-showMeshInfo', choices=('no', 'yes'), help='
53                     print mesh', default=
54                     simulationInfo['showMeshInfo'])
55
56 parser.add_argument('-showMtxBeforeBC', choices=('no', 'yes'),
57                     help='print matrix before BC
58                     treatment', default=
59                     simulationInfo['
60                     showMtxBeforeBC'])
61
62 parser.add_argument('-showVecBeforeBC', choices=('no', 'yes'),
63                     help='print vector before BC
64                     treatment', default=
65                     simulationInfo['
66                     showVecBeforeBC'])
67
68 parser.add_argument('-showMtxAfterBC', choices=('no', 'yes'), help
69                     ='print matrix after BC
70                     treatment', default=
71                     simulationInfo['
72                     showMtxAfterBC'])
```

```
51
52     parser.add_argument('-showVecAfterBC', choices=('no', 'yes'), help=
53                         ='print vector after BC
54                         treatment', default=
55                         simulationInfo['
56                         showVecAfterBC'])
57
58     parser.add_argument('-printRes', choices=('no', 'yes'), help='
59                         print result', default=
60                         simulationInfo['printRes'])
61
62     parser.add_argument('-printError', choices=('no', 'yes'), help='
63                         print error', default=
64                         simulationInfo['printError']
65                         )
66
67     parser.add_argument('-plotResNoMesh', choices=('no', 'yes'), help='
68                         plot result no mesh',
69                         default=simulationInfo['
70                         plotResNoMesh'])
71
72     parser.add_argument('-plotAnalyticalRes', choices=('no', 'yes'),
73                         help='plot analytical result
74                         ', default=simulationInfo['
75                         plotAnalyticalRes'])
76
77     parser.add_argument('-gaussPointNum', type=int, help='gauss point
78                         number', default=
79                         simulationInfo['
80                         gaussPointNum'])
81
82     args = parser.parse_args()
83
84
85     mesh = Mesh.Mesh(args)
86     system = linearSystem.LinearSystem(mesh, args)
87     system.assembleAllMtxA()
88     system.assembleAllVectorB()
89     system.boundaryTreatment()
90     system.solveLinearSystem()
91
92     # 数据后处理
93     if args.plotResNoMesh == 'yes':
94         postProcess.PlotResNoMesh(mesh, system.res1, "ux")
95         postProcess.PlotResNoMesh(mesh, system.res2, "uy")
96     if args.plotAnalyticalRes == 'yes':
```

```

77     postProcess.PlotAnalyticalRes(functions.analyticSolutionX,
78                                     args, fileName="u_x(exact)")
79     postProcess.PlotAnalyticalRes(functions.analyticSolutionY,
80                                     args, fileName="u_y(exact)")
81     plt.show()

```

B.3.2 Mesh.py

```

1 import numpy as np
2
3 class Mesh:
4     def __init__(self, args):
5         self.x0 = [0, 0]
6         self.xm = [0, 0]
7         self.dh = [0, 0]
8
9         self.NE = args.nCells
10        self.NN = [args.nCells[0]+1, args.nCells[1]+1]
11
12        self.P = np.zeros(((args.nCells[0] + 1) * (args.nCells[1] + 1),
13                          2))
14
15        self.nodeBC = np.zeros((2 * (args.nCells[0] + args.nCells[1]),
16                               7))
17
18        self.edgeBC = np.zeros((2 * (args.nCells[0] + args.nCells[1]),
19                               9), dtype=int)
20
21        self.bTp = args.type
22
23        if args.type == "LinearTri":
24            print("\n<< Using 2D Linear Triangular Finite Element
25                  Method >>")
26
27            self.T = np.zeros((2 * args.nCells[0] * args.nCells[1], 3)
28                              , dtype=int)
29
30            self.Pb = self.P
31            self.Tb = self.T
32        elif args.type == "QuadraticTri":
33            print("\n<< Using 2D Quadratic Triangular Finite Element
34                  Method >>")

```

```
29         self.T = np.zeros((2 * args.nCells[0] * args.nCells[1], 3)
30                             , dtype=int)
31
32         self.Pb = np.zeros(((2 * args.nCells[0] + 1) * (2 * args.
33                             nCells[1] + 1), 2))
34         self.Tb = np.zeros((2 * args.nCells[0] * args.nCells[1], 6
35                             ), dtype=int)
36
37         self.nodeBC = np.zeros((4 * (args.nCells[0] + args.nCells[
38                         1]), 7))
39
40         print("-->> Mesh info: nCellX = %d, nCellY = %d <<---" % (args
41                         .nCells[0], args.nCells[1]))
42         self.setExtents(args.simDomain[0], args.simDomain[1])
43
44         if args.showMeshInfo == 'yes':
45             print("P\n", self.P)
46             print("T\n", self.T)
47             print("Pb\n", self.Pb)
48             print("Tb\n", self.Tb)
49             print("Boundary Nodes\n", self.nodeBC)
50             print("Boundary Edges\n", self.edgeBC)
51
52
53     def setExtents(self, x0, xm):
54         self.x0 = x0
55         self.xm = xm
56         for i in range(len(self.dh)):
57             self.dh[i] = (self.xm[i] - self.x0[i]) / self.NE[i]
58
59         self.generatePTVertical()
60         self.generatePbTb()
61         self.boundaryGeneration()
62
63
64     def generatePTVertical(self):
65         for i in range(self.NN[0]):
66             for j in range(self.NN[1]):
67                 self.P[i * self.NN[1] + j][0] = self.x0[0] + i * self.
68                     dh[0]
69                 self.P[i * self.NN[1] + j][1] = self.x0[1] + j * self.
70                     dh[1]
```



```
107         self.Pb[nodeI - 1][1] = self.x0[1] + (nodeI % (2 *  
108             self.NE[1] + 1) - 1) * dh[1]  
109     ]  
110  
111     tmp = np.zeros((2 * self.NE[0] + 1, 2 * self.NE[1] + 1))  
112     for i in range(tmp.shape[0]):  
113         for j in range(tmp.shape[1]):  
114             tmp[i][j] = i * (2 * self.NE[1] + 1) + j  
115  
116     for cellI in range(1, self.NE[0] * self.NE[1] + 1):  
117         row, col = 0, 0  
118         if cellI % self.NE[1] == 0:  
119             row = self.NE[1]  
120             col = cellI // self.NE[1]  
121         else:  
122             row = cellI % self.NE[1]  
123             col = cellI // self.NE[1] + 1  
124  
125         if row - 1 < 0 or col - 1 < 0:  
126             return  
127  
128         if row >= 1 and col >= 1:  
129             self.Tb[2 * (cellI - 1)][0] = tmp[2 * col - 2][2 *  
130                 row - 2]  
131             self.Tb[2 * (cellI - 1)][1] = tmp[2 * col][2 * row  
132                 - 2]  
133             self.Tb[2 * (cellI - 1)][2] = tmp[2 * col - 2][2 *  
134                 row]  
135             self.Tb[2 * (cellI - 1)][3] = tmp[2 * col - 1][2 *  
136                 row - 2]  
137             self.Tb[2 * (cellI - 1)][4] = tmp[2 * col - 1][2 *  
138                 row - 1]  
139             self.Tb[2 * (cellI - 1)][5] = tmp[2 * col - 2][2 *  
140                 row - 1]  
141  
142             self.Tb[2 * cellI - 1][0] = tmp[2 * col - 2][2 *  
143                 row]  
144             self.Tb[2 * cellI - 1][1] = tmp[2 * col][2 * row -  
145                 2]  
146             self.Tb[2 * cellI - 1][2] = tmp[2 * col][2 * row]  
147             self.Tb[2 * cellI - 1][3] = tmp[2 * col - 1][2 *  
148                 row - 1]  
149             self.Tb[2 * cellI - 1][4] = tmp[2 * col][2 * row -  
150                 1]
```

```

139         self.Tb[2 * cellI - 1][5] = tmp[2 * col - 1][2 *
140                                     row]
141
142     def boundaryGeneration(self):
143         if self.bTp == "LinearTri":
144             NE = self.NE
145             NN = self.NN
146         elif self.bTp == "QuadraticTri":
147             NE = [2 * self.NE[0], 2 * self.NE[1]]
148             NN = [ne + 1 for ne in NE]
149
150
151         # ---> 边界网格节点的定义 <---#
152         # 将所有边界节点设置为 Dirichlet 边界类型
153         self.nodeBC[:, 0] = -1
154         self.nodeBC[:, 1] = -1
155
156         # bottom nodes
157         for i in range(NE[0]):
158             self.nodeBC[i][2] = i * NN[1]
159             self.nodeBC[i][3] = 0
160             self.nodeBC[i][4] = -1
161
162         # right nodes
163         for i in range(NE[0], NE[0] + NE[1]):
164             self.nodeBC[i][2] = NE[0] * NN[1] + i - NE[0]
165             self.nodeBC[i][3] = 1
166             self.nodeBC[i][4] = 0
167
168         # top nodes
169         for i in range(NE[0] + NE[1], 2 * NE[0] + NE[1]):
170             self.nodeBC[i][2] = (2 * NE[0] + NE[1] + 1 - i) * NN[1] -
171                                         1
172             self.nodeBC[i][3] = 0
173             self.nodeBC[i][4] = 1
174
175         # left nodes
176         for i in range(2 * NE[0] + NE[1], self.nodeBC.shape[0]):
177             self.nodeBC[i][2] = 2 * (NE[0] + NE[1]) - i
178             self.nodeBC[i][3] = -1
179             self.nodeBC[i][4] = 0
180
# left-bottom corner

```

```
181     self.nodeBC[0][3] = -1 / 2 ** 0.5
182     self.nodeBC[0][4] = -1 / 2 ** 0.5
183
184     # right-bottom corner
185     self.nodeBC[NE[0]][3] = 1 / 2 ** 0.5
186     self.nodeBC[NE[0]][4] = -1 / 2 ** 0.5
187
188     # right-top corner
189     self.nodeBC[NE[0]+NE[1]][3] = 1 / 2 ** 0.5
190     self.nodeBC[NE[0]+NE[1]][4] = 1 / 2 ** 0.5
191
192     # left-top corner
193     self.nodeBC[2*NE[0]+NE[1]][3] = -1 / 2 ** 0.5
194     self.nodeBC[2*NE[0]+NE[1]][4] = 1 / 2 ** 0.5
195
196     self.nodeBC[:,5] = -self.nodeBC[:,4]
197     self.nodeBC[:,6] = self.nodeBC[:,3]
198
199
200     # ---> 边界网格边的定义 <---
201     # 初始化所有边界类型为 Dirichlet 类型边
202     self.edgeBC[:, 0] = -1
203     self.edgeBC[:, 1] = -1
204
205     # bottom edge
206     for i in range(1, self.NE[0] + 1):
207         self.edgeBC[i - 1][2] = (i - 1) * 2 * self.NE[1]
208         self.edgeBC[i - 1][3] = (i - 1) * (self.NE[1] + 1)
209         self.edgeBC[i - 1][4] = i * (self.NE[1] + 1)
210         self.edgeBC[i - 1][5] = 0
211         self.edgeBC[i - 1][6] = -1
212
213     # right edge
214     for i in range(self.NE[0] + 1, self.NE[0] + self.NE[1] + 1):
215         self.edgeBC[i - 1][2] = (self.NE[0] - 1) * 2 * self.NE[1] +
216                               2 * (i - self.NE[0]) - 1
217         self.edgeBC[i - 1][3] = self.NE[0] * (self.NE[1] + 1) + i
218                               - self.NE[0] - 1
219         self.edgeBC[i - 1][4] = self.NE[0] * (self.NE[1] + 1) + i
220                               - self.NE[0]
221         self.edgeBC[i - 1][5] = 1
222         self.edgeBC[i - 1][6] = 0
223
224     # top edge
```

```

222     for i in range(self.NE[0] + self.NE[1] + 1, 2 * self.NE[0] +
223                     self.NE[1] + 1):
224         self.edgeBC[i - 1][2] = (2 * self.NE[0] + self.NE[1] + 1 -
225                                   i) * 2 * self.NE[1] - 1
226         self.edgeBC[i - 1][3] = (2 * self.NE[0] + self.NE[1] + 2 -
227                                   i) * (self.NE[1] + 1) - 1
228         self.edgeBC[i - 1][4] = (2 * self.NE[0] + self.NE[1] + 1 -
229                                   i) * (self.NE[1] + 1) - 1
230         self.edgeBC[i - 1][5] = 0
231         self.edgeBC[i - 1][6] = 1
232
233     # left edge
234     for i in range(2 * self.NE[0] + self.NE[1] + 1, self.edgeBC.
235                     shape[0] + 1):
236         self.edgeBC[i - 1][2] = 2 * (2 * self.NE[0] + 2 * self.NE[
237                                       1] + 1 - i) - 2
238         self.edgeBC[i - 1][3] = 2 * self.NE[0] + 2 * self.NE[1] +
239                               1 - i
240         self.edgeBC[i - 1][4] = 2 * self.NE[0] + 2 * self.NE[1] -
241                               i
242         self.edgeBC[i - 1][5] = -1
243         self.edgeBC[i - 1][6] = 0
244
245     self.edgeBC[:,7] = -self.edgeBC[:,6]
246     self.edgeBC[:,8] = self.edgeBC[:,5]

```

B.3.3 linearSystem.py

```

1 import numpy as np
2 import scipy.sparse as sp
3 from math import *
4 import functions as fun
5 import time
6 np.set_printoptions(precision=6, suppress=True)
7
8 class LinearSystem:
9     def __init__(self, mesh, args) -> None:
10         self.mesh = mesh
11         self.args = args
12         self.gPtN = args.gaussPointNum
13
14         self.gaussWeightRef = np.zeros(args.gaussPointNum)

```

```
15     self.gaussPointRef = np.zeros((args.gaussPointNum, 2))
16
17     self.gaussWeightLocal = np.zeros(args.gaussPointNum)
18     self.gaussPointLocal = np.zeros((args.gaussPointNum, 2))
19
20     self.Jacobian = np.zeros((2, 2))
21     self.JacobianInv = np.zeros((2, 2))
22     self.Jdet = 0
23
24     self mtxA = None
25     self mtxA1 = np.zeros((mesh.Pb.shape[0], mesh.Pb.shape[0]))
26     self mtxA2 = np.zeros((mesh.Pb.shape[0], mesh.Pb.shape[0]))
27     self mtxA3 = np.zeros((mesh.Pb.shape[0], mesh.Pb.shape[0]))
28     self mtxA4 = np.zeros((mesh.Pb.shape[0], mesh.Pb.shape[0]))
29     self mtxA5 = np.zeros((mesh.Pb.shape[0], mesh.Pb.shape[0]))
30     self mtxA6 = np.zeros((mesh.Pb.shape[0], mesh.Pb.shape[0]))
31     self mtxA7 = np.zeros((mesh.Pb.shape[0], mesh.Pb.shape[0]))
32     self mtxA8 = np.zeros((mesh.Pb.shape[0], mesh.Pb.shape[0]))
33     self vecB = None
34     self vecB1 = np.zeros(mesh.Pb.shape[0])
35     self vecB2 = np.zeros(mesh.Pb.shape[0])
36     self.res = None
37     self.res1 = np.zeros(mesh.Pb.shape[0])
38     self.res2 = np.zeros(mesh.Pb.shape[0])
39
40     self.maxError = 0
41     self.Linfinity = 0
42     self.L2 = 0
43     self.H1 = 0
44
45     if mesh.bTp == "LinearTri":
46         self.baseNum = 3
47     elif mesh.bTp == "QuadraticTri":
48         self.baseNum = 6
49
50     self.gaussWeightAndPointRef()
51
52
53 def gaussWeightAndPointRef(self):
54     if self.mesh.bTp == "LinearTri" or "QuadraticTri":
55         if self.gPtN == 3:
56             self.gaussWeightRef[0] = 1 / 6
57             self.gaussWeightRef[1] = 1 / 6
58             self.gaussWeightRef[2] = 1 / 6
```

```
59         self.gaussPointRef[0][0] = 0.5
60         self.gaussPointRef[0][1] = 0
61
62         self.gaussPointRef[1][0] = 0.5
63         self.gaussPointRef[1][1] = 0.5
64
65         self.gaussPointRef[2][0] = 0
66         self.gaussPointRef[2][1] = 0.5
67
68     elif self.gPtN == 4:
69         self.gaussWeightRef[0] = (1 - 1 / sqrt(3)) / 8
70         self.gaussWeightRef[1] = (1 - 1 / sqrt(3)) / 8
71         self.gaussWeightRef[2] = (1 + 1 / sqrt(3)) / 8
72         self.gaussWeightRef[3] = (1 + 1 / sqrt(3.0)) / 8
73
74         self.gaussPointRef[0][0] = (1 / sqrt(3) + 1) / 2
75         self.gaussPointRef[0][1] = (
76             1 - 1 / sqrt(3)) * (1 + 1 / sqrt(3)) / 4
77
78         self.gaussPointRef[1][0] = (1 / sqrt(3) + 1) / 2
79         self.gaussPointRef[1][1] = (
80             1 - 1 / sqrt(3)) * (1 - 1 / sqrt(3)) / 4
81
82         self.gaussPointRef[2][0] = (-1 / sqrt(3) + 1) / 2
83         self.gaussPointRef[2][1] = (
84             1 + 1 / sqrt(3)) * (1 + 1 / sqrt(3)) / 4
85
86         self.gaussPointRef[3][0] = (-1 / sqrt(3) + 1) / 2
87         self.gaussPointRef[3][1] = (
88             1 + 1 / sqrt(3)) * (1 - 1 / sqrt(3)) / 4
89
90     elif self.gPtN == 9:
91         self.gaussWeightRef[0] = 64 / 81 * 1 / 8
92         self.gaussWeightRef[1] = 100 / 324 * (1 - sqrt(3 / 5))
93                         / 8
94         self.gaussWeightRef[2] = 100 / 324 * (1 - sqrt(3 / 5))
95                         / 8
96         self.gaussWeightRef[3] = 100 / 324 * (1 + sqrt(3 / 5))
97                         / 8
98         self.gaussWeightRef[4] = 100 / 324 * (1 + sqrt(3 / 5))
99                         / 8
100        self.gaussWeightRef[5] = 40 / 81 * 1 / 8
101        self.gaussWeightRef[6] = 40 / 81 * 1 / 8
102        self.gaussWeightRef[7] = 40 / 81 * (1 - sqrt(3 / 5)) /
```

```
98     self.gaussWeightRef[8] = 40 / 81 * (1 + sqrt(3 / 5)) /
99                                     8
100
100    self.gaussPointRef[0][0] = 1 / 2
101    self.gaussPointRef[0][1] = 1 / 4
102
103    self.gaussPointRef[1][0] = (1.0 + sqrt(3.0 / 5.0)) / 2
104                                     .
104    self.gaussPointRef[1][1] = (
105        1.0 - sqrt(3.0 / 5.0)) * (1.0 + sqrt(3.0 / 5.0)) /
106                                     4.0
107
107    self.gaussPointRef[2][0] = (1.0 + sqrt(3.0 / 5.0)) / 2
108                                     .
108    self.gaussPointRef[2][1] = (
109        1.0 - sqrt(3.0 / 5.0)) * (1.0 - sqrt(3.0 / 5.0)) /
110                                     4.0
111
111    self.gaussPointRef[3][0] = (1.0 - sqrt(3.0 / 5.0)) / 2
112                                     .
112    self.gaussPointRef[3][1] = (
113        1.0 + sqrt(3.0 / 5.0)) * (1.0 + sqrt(3.0 / 5.0)) /
114                                     4.0
115
115    self.gaussPointRef[4][0] = (1.0 - sqrt(3.0 / 5.0)) / 2
116                                     .
116    self.gaussPointRef[4][1] = (
117        1.0 + sqrt(3.0 / 5.0)) * (1.0 - sqrt(3.0 / 5.0)) /
118                                     4.0
119
119    self.gaussPointRef[5][0] = (1.0 + 0.0) / 2.0
120    self.gaussPointRef[5][1] = (
121        1.0 - 0.0) * (1.0 + sqrt(3.0 / 5.0)) / 4.0
122
122    self.gaussPointRef[6][0] = (1.0 + 0.0) / 2.0
123    self.gaussPointRef[6][1] = (
124        1.0 - 0.0) * (1.0 - sqrt(3.0 / 5.0)) / 4.0
125
125    self.gaussPointRef[7][0] = (1.0 + sqrt(3.0 / 5.0)) / 2
126                                     .
126    self.gaussPointRef[7][1] = (
127        1.0 - sqrt(3.0 / 5.0)) * (1.0 + 0.0) / 4.0
128
128
```

```

131         self.gaussPointRef[8][0] = (1.0 - sqrt(3.0 / 5.0)) / 2
132                                         .0
133         self.gaussPointRef[8][1] = (
134             1.0 + sqrt(3.0 / 5.0)) * (1.0 + 0.0) / 4.0
135
136     def gaussWeightAndPointLocal(self, ver):
137         x1, y1 = ver[0][0], ver[0][1]
138         x2, y2 = ver[1][0], ver[1][1]
139         x3, y3 = ver[2][0], ver[2][1]
140         self.Jdet = abs((x2 - x1) * (y3 - y1) - (x3 - x1) * (y2 - y1))
141
142     for i in range(self.gPtN):
143         self.gaussWeightLocal[i] = self.Jdet * self.gaussWeightRef
144                                         [i]
145         self.gaussPointLocal[i][0] = x1 + (x2 - x1) * self.
146                                         gaussPointRef[i][0] + (
147                                         x3 - x1) * self.gaussPointRef[i][1]
148         self.gaussPointLocal[i][1] = y1 + (y2 - y1) * self.
149                                         gaussPointRef[i][0] + (
150                                         y3 - y1) * self.gaussPointRef[i][1]
151
152     def refBaseFunction(self, x, dD, bIx):
153         res = 0
154         if self.mesh.bTp == "LinearTri":
155             if dD[0] == 0 and dD[1] == 0:
156                 if bIx == 0:
157                     res = 1 - x[0] - x[1]
158                 elif bIx == 1:
159                     res = x[0]
160                 elif bIx == 2:
161                     res = x[1]
162             elif dD[0] == 1 and dD[1] == 0:
163                 if bIx == 0:
164                     res = -1
165                 elif bIx == 1:
166                     res = 1
167                 elif bIx == 2:
168                     res = 0
169             elif dD[0] == 0 and dD[1] == 1:
170                 if bIx == 0:
171                     res = -1
172                 elif bIx == 1:
173

```

```
171             res = 0
172         elif bIx == 2:
173             res = 1
174     else:
175         return 0
176     return res
177 elif self.mesh.bTp == "QuadraticTri":
178     if dD[0] == 0 and dD[1] == 0:
179         if bIx == 0:
180             res = 2 * pow(x[0], 2) + 2 * pow(x[1], 2) + 4 * x[
181                 0] * x[1] - 3 * x[0] - 3 * x
182                 [1] + 1
183         elif bIx == 1:
184             res = 2 * pow(x[0], 2) - x[0]
185         elif bIx == 2:
186             res = 2 * pow(x[1], 2) - x[1]
187         elif bIx == 3:
188             res = -4 * pow(x[0], 2) - 4 * x[0] * x[1] + 4 * x[
189                 0]
190         elif bIx == 4:
191             res = 4 * x[0] * x[1]
192         elif bIx == 5:
193             res = -4 * pow(x[1], 2) - 4 * x[0] * x[1] + 4 * x[
194                 1]
195     elif dD[0] == 1 and dD[1] == 0:
196         if bIx == 0:
197             res = 4 * x[0] + 4 * x[1] - 3
198         elif bIx == 1:
199             res = 4 * x[0] - 1
200         elif bIx == 2:
201             res = 0
202         elif bIx == 3:
203             res = -8 * x[0] - 4 * x[1] + 4
204         elif bIx == 4:
205             res = 4 * x[1]
206         elif bIx == 5:
207             res = -4 * x[1]
208     elif dD[0] == 0 and dD[1] == 1:
209         if bIx == 0:
210             res = 4 * x[1] + 4 * x[0] - 3
```

```
211     elif bIx == 3:  
212         res = -4 * x[0]  
213     elif bIx == 4:  
214         res = 4 * x[0]  
215     elif bIx == 5:  
216         res = -8 * x[1] - 4 * x[0] + 4  
217     elif dD[0] == 2 and dD[1] == 0:  
218         if bIx == 0:  
219             res = 4  
220         elif bIx == 1:  
221             res = 4  
222         elif bIx == 2:  
223             res = 0  
224         elif bIx == 3:  
225             res = -8  
226         elif bIx == 4:  
227             res = 0  
228         elif bIx == 5:  
229             res = 0  
230     elif dD[0] == 0 and dD[1] == 2:  
231         if bIx == 0:  
232             res = 4  
233         elif bIx == 1:  
234             res = 0  
235         elif bIx == 2:  
236             res = 4  
237         elif bIx == 3:  
238             res = 0  
239         elif bIx == 4:  
240             res = 0  
241         elif bIx == 5:  
242             res = -8  
243     elif dD[0] == 1 and dD[1] == 1:  
244         if bIx == 0:  
245             res = 4  
246         elif bIx == 1:  
247             res = 0  
248         elif bIx == 2:  
249             res = 0  
250         elif bIx == 3:  
251             res = -4  
252         elif bIx == 4:  
253             res = 4  
254         elif bIx == 5:
```

```

255             res = -4
256         return res
257
258
259     def localBaseFun(self, x, ver, dD, bIx):
260         J00 = ver[1][0] - ver[0][0]
261         J01 = ver[2][0] - ver[0][0]
262         J10 = ver[1][1] - ver[0][1]
263         J11 = ver[2][1] - ver[0][1]
264         Jdet = J00 * J11 - J01 * J10
265
266         xHat = (J11 * (x[0] - ver[0][0]) - J01 * (x[1] - ver[0][1])) /
267                                         Jdet
268         yHat = (-J10 * (x[0] - ver[0][0]) + J00 * (x[1] - ver[0][1])) /
269                                         Jdet
270
271         res = 0
272         if dD[0] == 0 and dD[1] == 0:
273             res = self.refBaseFunction([xHat, yHat], [0, 0], bIx)
274         elif dD[0] == 1 and dD[1] == 0:
275             res = (self.refBaseFunction([xHat, yHat], [
276                 1, 0], bIx) * J11 + self.refBaseFunction([xHat,
277                                                 yHat], [0, 1], bIx) * (-J10)
278                                         ) / Jdet
279         elif dD[0] == 0 and dD[1] == 1:
280             res = (self.refBaseFunction([xHat, yHat], [
281                 1, 0], bIx) * (-J01) + self.refBaseFunction([xHat,
282                                                 yHat], [0, 1], bIx) * J00) /
283                                         Jdet
284         elif dD[1] == 2 and dD[1] == 0:
285             res = (self.refBaseFunction([xHat, yHat], [2, 0], bIx) *
286                     pow(J11, 2) + self.
287                     refBaseFunction([xHat, yHat]
288                                     ,
289                                     [0, 2], bIx) * pow(J10, 2) + self.refBaseFunction([
290                                         xHat, yHat], [1, 1], bIx) *
291                                         (-2 * J10 * J11)) / pow(Jdet
292                                         , 2)
293         elif dD[0] == 0 and dD[1] == 2:
294             res = (self.refBaseFunction([xHat, yHat], [2, 0], bIx) *
295                     pow(J01, 2) + self.
296                     refBaseFun([xHat, yHat], [
297                                     0, 2], bIx) * pow(J00, 2) + self.refBaseFunction([
298                                         xHat, yHat], [1, 1], bIx) *
299                                         2)
300
301     def refBaseFunction(self, x, y, dD, bIx):
302         if dD[0] == 0 and dD[1] == 0:
303             return self.localBaseFun(x, y, dD, bIx)
304         else:
305             return self.refBaseFunction([x, y], dD, bIx)

```



```
311         tmp = self.gaussInteTrialAndTest(ver, [alpha, beta
312                                         ], tralD, testD, func)
313         row = self.mesh.Tb[n][beta]
314         col = self.mesh.Tb[n][alpha]
315         tmpMtxA[row][col] += tmp
316     return tmpMtxA
317
318 def assembleAllMtxA(self):
319     print("\n-->> Assembling Matrix <<---")
320     timeStart = time.time()
321     self mtxA1 = self.assembleMtxA(fun.funLamda, [1, 0], [1, 0])
322     self mtxA2 = self.assembleMtxA(fun.funMu, [1, 0], [1, 0])
323     self mtxA3 = self.assembleMtxA(fun.funMu, [0, 1], [0, 1])
324     self mtxA4 = self.assembleMtxA(fun.funLamda, [0, 1], [1, 0])
325     self mtxA5 = self.assembleMtxA(fun.funMu, [1, 0], [0, 1])
326     self mtxA6 = self.assembleMtxA(fun.funLamda, [1, 0], [0, 1])
327     self mtxA7 = self.assembleMtxA(fun.funMu, [0, 1], [1, 0])
328     self mtxA8 = self.assembleMtxA(fun.funLamda, [0, 1], [0, 1])
329     self mtxA = np.block([[self mtxA1+2*self mtxA2+self mtxA3,
330                           self mtxA4+self mtxA5],
331                           [self mtxA6+self mtxA7, self.
332                           mtxA8+2*self mtxA3+self.
333                           mtxA2]])
334
335
336
337     # 载荷向量组装功能函数
338     def gaussInteTest(self, ver, teBIX, function):
339         res = 0
340
341         self.gaussWeightAndPointLocal(ver)
342
343         for i in range(self.gPtN):
344             tmp = function(self.gaussPointLocal[i])
345             tmpBase = self.localBaseFun(
346                 self.gaussPointLocal[i], ver, [0, 0], teBIX)
347             res += self.gaussWeightLocal[i] * tmp * tmpBase
348
349     return res
```

```

350
351
352     def assembleVectorB(self, func):
353         tmpVecB = np.zeros(self.mesh.Pb.shape[0])
354         for n in range(self.mesh.Tb.shape[0]):
355             ver = np.zeros((self.mesh.Tb.shape[1], 2))
356             for i in range(self.mesh.Tb.shape[1]):
357                 ver[i][0] = self.mesh.Pb[self.mesh.Tb[n][i]][0]
358                 ver[i][1] = self.mesh.Pb[self.mesh.Tb[n][i]][1]
359
360             for beta in range(self.baseNum):
361                 tmp = self.gaussInteTest(ver, beta, func)
362                 tmpVecB[self.mesh.Tb[n][beta]] += tmp
363         return tmpVecB
364
365     def assembleAllVectorB(self):
366         print("\n-->> Assembling Vector <<---")
367         timeStart = time.time()
368         self.vecB1 = self.assembleVectorB(fun.funF1)
369         self.vecB2 = self.assembleVectorB(fun.funF2)
370         self.vecB = np.concatenate((self.vecB1, self.vecB2))
371         timeEnd = time.time()
372         print("-->> Time = %f s <<---" % (timeEnd - timeStart))
373         if self.args.showVecBeforeBC == 'yes':
374             print("-->> Vector Before BC treatment <<---")
375             print(self.vecB)
376
377
378     def boundaryTreatment(self):
379         print("\n-->> Boundary Treatment <<---")
380         timeStart = time.time()
381         for k in range(self.mesh.nodeBC.shape[0]):
382             i = int(self.mesh.nodeBC[k][2])
383             # 处理 x 方向的边界条件
384             if self.mesh.nodeBC[k][0] == -1:
385                 self.mtxA[i, :] = 0
386                 self.mtxA[i][i] = 1
387                 self.vecB[i] = fun.funG1(self.mesh.Pb[i])
388
389             # 处理 y 方向的边界条件
390             if self.mesh.nodeBC[k][1] == -1:
391                 iy = i + self.mesh.Pb.shape[0]
392                 self.mtxA[iy, :] = 0
393                 self.mtxA[iy][iy] = 1

```

```
394         self.vecB[iy] = fun.funG2(self.mesh.Pb[i])
395     timeEnd = time.time()
396     print("--->> Time = %f s <<---" % (timeEnd - timeStart))
397
398     if self.args.showMtxAfterBC == 'yes':
399         print("--->> Matrix After BC treatment <<---")
400         print(self mtxA)
401
402     if self.args.showVecAfterBC == 'yes':
403         print("\n--->> Vector After BC treatment <<---")
404         print(self.vecB)
405
406
407     def solveLinearSystem(self):
408         timeStart = time.time()
409         print("\n--->> Linear System Solving <<---")
410         self.res = np.linalg.solve(self mtxA, self.vecB)
411         self.res1 = self.res[:self.mesh.Pb.shape[0]]
412         self.res2 = self.res[self.mesh.Pb.shape[0]:]
413         timeEnd = time.time()
414         print("--->> Simulation Done !!! <<---")
415         print("--->> Time = %f s <<---" % (timeEnd - timeStart))
416
417         self.errorCalculation()
418
419         if self.args.printRes == 'yes':
420             print("\n--->> Result <<---")
421             print("ux")
422             print(self.res1)
423             print("\nuy")
424             print(self.res2)
425
426
427     """
428     误差分析
429     """
430     def infinityFeRes(self, x, uhLocal, ver, dD):
431         res = 0
432         for i in range(self.baseNum):
433             res += uhLocal[i] * self.localBaseFun(x, ver, dD, i)
434         return res
435
436
437     def gaussIntegralFeRes(self, uhLocal, ver, function, dD):
```

```

438     res = 0
439     self.gaussWeightAndPointLocal(ver)
440     for i in range(self.gPtN):
441         res += self.gaussWeightLocal[i] * pow(function(self.
442                                         gaussPointLocal[i]) - self.
443                                         infinityFeRes(self.
444                                         gaussPointLocal[i], uhLocal,
445                                         ver, dD), 2)
446
447     return res
448
449
450
451
452
453
454
455     def maxErrorCompute(self, resVec, func):
456         res = abs(func(self.mesh.Pb[0]) - resVec[0])
457         for i in range(1, self.mesh.Pb.shape[0]):
458             tmp = abs(func(self.mesh.Pb[i]) - resVec[i])
459             if tmp > res:
460                 res = tmp
461
462         return res
463
464
465
466
467     def errorInfinityNorm(self, resVec, dD, func):
468         res = 0
469         uhLocal = [0 for i in range(self.baseNum)]
470         for n in range(self.mesh.Tb.shape[0]):
471             ver = np.zeros((self.mesh.Tb.shape[1], 2))
472             for i in range(self.mesh.Tb.shape[1]):
473                 ver[i][0] = self.mesh.Pb[self.mesh.Tb[n][i]][0]
474                 ver[i][1] = self.mesh.Pb[self.mesh.Tb[n][i]][1]
475             self.gaussWeightAndPointLocal(ver)
476             for i in range(self.baseNum):
477                 uhLocal[i] = resVec[self.mesh.Tb[n][i]]
478             tmp = 0
479             for i in range(self.gPtN):
480                 value = abs(func(self.gaussPointLocal[i]) - self.
481                             infinityFeRes(self.
482                             gaussPointLocal[i], uhLocal,
483                             ver, dD))
484
485                 if tmp < value:
486                     tmp = value
487             if res < tmp:
488                 res = tmp
489
490         return res
491
492
493
494

```



```

509         ), 2) + pow(self.
510         L2AndH1NormError(self.res2,
511         fun.solutionYDerivativeY, [0
512         , 1]), 2))
513     self.H1 = (xH1 ** 2 + yH1 ** 2) ** 0.5
514
515     if self.args.printError == 'yes':
516         print("\n-->> Error Information <<---")
517         print("maxError = %.4e" % self.maxError)
518         print("Linfinity = %.4e" % self.Linfinity)
519         print("L2 = %.4e" % self.L2)
520         print("H1 = %.4e" % self.H1)

```

B.3.4 functions.py

```

1 import Mesh
2 from math import *
3
4 def funLamda(x):
5     return 1
6
7
8 def funMu(x):
9     return 2
10
11
12 def funF1(x):
13     lam = funLamda(x)
14     mu = funMu(x)
15     res = -(lam+2*mu)*(-pi**2*sin(pi*x[0])*sin(pi*x[1]))-(lam+mu)*((2*
16             x[0]-1)*(2*x[1]-1))-mu*(-pi
17             *2*sin(pi*x[0])*sin(pi*x[1])
18             ))
19     return res
20
21
22 def funF2(x):
23     lam = funLamda(x)
24     mu = funMu(x)
25     res = -(lam+2*mu)*(2*x[0]*(x[0]-1))-(lam+mu)*(pi**2*cos(pi*x[0])*
26             cos(pi*x[1]))-mu*(2*x[1]*(x[
27             1]-1))

```

```
23     return res
24
25
26 def funG1(x):
27     return 0
28
29
30 def funG2(x):
31     return 0
32
33
34 def analyticSolutionX(x):
35     return sin(pi*x[0])*sin(pi*x[1])
36
37
38 def analyticSolutionY(x):
39     return x[0] * (x[0] - 1) * x[1] * (x[1] - 1)
40
41
42 def solutionXDerivativeX(x):
43     return pi*cos(pi*x[0])*sin(pi*x[1])
44
45
46 def solutionXDerivativeY(x):
47     return pi*sin(pi*x[0])*cos(pi*x[1])
48
49
50 def solutionYDerivativeX(x):
51     return (2*x[0]-1)*x[1]*(x[1]-1)
52
53
54 def solutionYDerivativeY(x):
55     return x[0]*(x[0]-1)*(2*x[1]-1)
```

B.3.5 postProcess.py

```
1 import matplotlib.pyplot as plt
2 import matplotlib.tri as mtri
3 import functions as fun
4 import numpy as np
5 fontStyle = {'family':'DejaVu Sans', 'style':'italic', 'weight': 'normal', 'size':20}
```

```

6
7 def PlotResNoMesh(mesh, resVec, fileName="u"):
8     fig = plt.figure(figsize=(7.8, 7))
9     ax = fig.add_subplot(111)
10    plt.xlabel('x', fontproperties = fontStyle)
11    plt.ylabel('y', fontproperties = fontStyle)
12    ax.tick_params("both", which='major', length=8, width=1.5, colors=
13                  'k', direction='in', pad=12)
14    plt.xticks(fontsize = 18)
15    plt.yticks(fontsize = 18)
16    tri = mtri.Triangulation(mesh.P[:, 0], mesh.P[:, 1], mesh.T)
17    if mesh.bTp == 'LinearTri':
18        plot = plt.tricontourf(tri, resVec, cmap= 'jet')
19    else:
20        x = [mesh.x0[0] + i * mesh.dh[0] / 2 for i in range(2 * mesh.
21                                              NE[0] + 1)]
22        y = [mesh.x0[1] + j * mesh.dh[1] / 2 for j in range(2 * mesh.
23                                              NE[1] + 1)]
24        res = []
25        for j in range(2 * mesh.NE[1] + 1):
26            tmp = []
27            for i in range(2 * mesh.NE[0] + 1):
28                tmp.append(resVec[j + i * (2 * mesh.NE[1] + 1)])
29            res.append(tmp)
30        x, y = np.meshgrid(x, y)
31        plot = plt.contourf(x, y, res,12, cmap='jet')
32        xTicks=np.linspace(mesh.Pb[0][0],mesh.Pb[-1][0],5)
33        yTicks=np.linspace(mesh.Pb[0][1],mesh.Pb[-1][1],5)
34        plt.xticks(xTicks)
35        plt.yticks(yTicks)
36        colorBar = fig.colorbar(plot)
37        colorBar.ax.set_title(fileName, fontproperties=fontStyle)
38        colorBar.ax.tick_params(direction = 'in', labelsize=fontStyle['
39                                              size'])
40
41    plt.tight_layout()
42    plt.show()
43    plt.ioff()

44
45 def PlotAnalyticalRes(func, args, fileName="uexact"):
46     nCells = [150, 150]
47     x=np.linspace(args.simDomain[0][0], args.simDomain[1][0], nCells[0
48                  ])

```

```
44     y=np.linspace(args.simDomain[0][1], args.simDomain[1][1], nCells[1]
45                     ])
46     res = []
47     for j in y:
48         tmp = []
49         for i in x:
50             value = func([i, j])
51             tmp.append(value)
52         res.append(tmp)
53
53     x,y=np.meshgrid(x,y)
54     width, height = 7.8, 7
55     fig, ax = plt.subplots(figsize=(width, height))
56     ax.tick_params("both", which='major', length=5, colors='k',
57                    direction='in', pad=12)
57     plot = plt.contourf(x,y,res,12,cmap='jet')
58     ax.set_xlabel('x', fontproperties=fontStyle)
59     ax.set_ylabel('y', fontproperties=fontStyle)
60     xTicks=np.linspace(args.simDomain[0][0], args.simDomain[1][0], 5)
61     yTicks=np.linspace(args.simDomain[0][1], args.simDomain[1][1], 5)
62     plt.xticks(xTicks)
63     plt.yticks(yTicks)
64     plt.setp(ax.get_xticklabels(), fontproperties=fontStyle)
65     plt.setp(ax.get_yticklabels(), fontproperties=fontStyle)
66     colorBar = fig.colorbar(plot)
67     colorBar.ax.set_title(fileName, fontproperties=fontStyle)
68     colorBar.ax.tick_params(direction = 'in', labelsize=fontStyle['
69                           size'])
70     plt.tight_layout()
71     plt.show()
71     plt.ioff()
```

B.4 第五章参考程序

本章为包含张量、多物理场问题模型的有限元参考程序（稳态 Stokes 程序），读者可根据本书及前述章节参考程序实现稳态 Navier-Stokes 方程有限元程序。

B.4.1 2D_FEM.py

```
1 import Mesh
2 import linearSystem
```

```
3 import argparse
4 import matplotlib.pyplot as plt
5 import postProcess
6 import functions
7
8 """
9 运行指令示范:
10 python .\2D_FEM.py
11 或者
12 python .\2D_FEM.py -type QuadraticTri -nCells 2 3 -simDomain 0 1 0 1 -
13   showMeshInfo yes -
14   showMtxAndVecBeforeBC yes -
15   showMtxAndVecAfterBC yes -
16   printRes yes -printError yes
17   -plotMesh yes -
18   plotResWithMesh yes -
19   plotResNoMesh yes -
20   plotAnalyticalRes yes -
21   gaussPointNum 3
22 """
23
24 if __name__ == "__main__":
25     plt.ion()
26     simulationInfo = {
27         "description": "2DLinearElasticHe",
28         "nCells": [8, 2],
29         "simDomain": [[0, -0.25], [1, 0]],
30         "showMeshInfo": "no",
31         "showMtxBeforeBC": "no",
32         "showVecBeforeBC": "no",
33         "showMtxAfterBC": "no",
34         "showVecAfterBC": "no",
35         "printRes": "no",
36         "printError": "yes",
37         "plotResNoMesh": "no",
38         "plotAnalyticalRes": "no",
39         "gaussPointNum": 9}
40
41     parser = argparse.ArgumentParser(prog='2D_FEM', description=
42                                     simulationInfo["description"])
43
44     parser.add_argument('-nCells', nargs='+', type=int, help='No of
45                         cells in x and y direction',
46                         default=simulationInfo['
```

```
34
35     parser.add_argument('-nCells', type=int, help='number of cells in the domain', default=100)
36
37     parser.add_argument('-simDomain', type=list, help='simulation domain', default=[0, 1])
38
39     parser.add_argument('-showMeshInfo', choices=('no', 'yes'), help='print mesh information', default=False)
40
41     parser.add_argument('-showMtxBeforeBC', choices=('no', 'yes'), help='print matrix before BC treatment', default=False)
42
43     parser.add_argument('-showVecBeforeBC', choices=('no', 'yes'), help='print vector before BC treatment', default=False)
44
45     parser.add_argument('-showMtxAfterBC', choices=('no', 'yes'), help='print matrix after BC treatment', default=False)
46
47     parser.add_argument('-showVecAfterBC', choices=('no', 'yes'), help='print vector after BC treatment', default=False)
48
49     parser.add_argument('-printRes', choices=('no', 'yes'), help='print result', default=False)
50
51     parser.add_argument('-printError', choices=('no', 'yes'), help='print error', default=False)
```

```

51     parser.add_argument('-plotResNoMesh', choices=('no', 'yes'), help=
52                         'plot result no mesh',
53                         default=simulationInfo['
54                             plotResNoMesh'])
55
56     parser.add_argument('-plotAnalyticalRes', choices=('no', 'yes'),
57                         help='plot analytical result
58                         ', default=simulationInfo['
59                             plotAnalyticalRes'])
60
61     parser.add_argument('-gaussPointNum', type=int, help='gauss point
62                         number', default=
63                         simulationInfo['
64                             gaussPointNum'])
65
66     args = parser.parse_args()
67
68     mesh = Mesh.Mesh(args)
69     system = linearSystem.LinearSystem(mesh, args)
70     system.assembleAllMtxA()
71     system.assembleAllVectorB()
72     system.boundaryTreatment()
73     system.solveLinearSystem()
74
75     # 数据后处理
76     if args.plotResNoMesh == 'yes':
77         postProcess.PlotResNoMesh(mesh, system.res1, mesh.vPb, "u_x", "
78                                     velocity")
79         postProcess.PlotResNoMesh(mesh, system.res2, mesh.vPb, "u_y", "
80                                     velocity")
81         postProcess.PlotResNoMesh(mesh, system.res3, mesh.pPb, "p", "
82                                     pressure")
83
84     if args.plotAnalyticalRes == 'yes':
85         postProcess.PlotAnalyticalRes(functions.exactU, args, fileName
86                                         ="u_x(exact)")
87         postProcess.PlotAnalyticalRes(functions.exactV, args, fileName
88                                         ="u_y(exact)")
89         postProcess.PlotAnalyticalRes(functions.exactP, args, fileName
90                                         ="p(exact)")
91
92     plt.show()

```

B.4.2 Mesh.py

```
1 import numpy as np
2
3 class Mesh:
4     def __init__(self, args):
5         self.x0 = [0, 0]
6         self.xm = [0, 0]
7         self.dh = [0, 0]
8
9         self.NE = args.nCells
10        self.NN = [args.nCells[0]+1, args.nCells[1]+1]
11
12        self.P = np.zeros(((args.nCells[0] + 1) * (args.nCells[1] + 1),
13                           2))
13        self.T = np.zeros((2 * args.nCells[0] * args.nCells[1], 3),
14                           dtype=int)
14
15        self.nodeBCP = np.zeros((2 * (args.nCells[0] + args.nCells[1]),
16                               7))
16        self.nodeBCV = np.zeros((4 * (args.nCells[0] + args.nCells[1]),
17                               7))
17        self.edgeBC = np.zeros((2 * (args.nCells[0] + args.nCells[1]),
18                               9), dtype=int)
19
20        print("\n<< Using 2D Linear Triangular Mesh for Pressure >>")
21        self.pPb = self.P
22        self.pTb = self.T
23
24        print("\n<< Using 2D Quadratic Triangular Mesh for Velocity >>
25              ")
25        self.vPb = np.zeros(((2 * args.nCells[0] + 1) * (2 * args.
26                           nCells[1] + 1), 2))
27        self.vTb = np.zeros((2 * args.nCells[0] * args.nCells[1], 6),
28                           dtype=int)
29
30        print("-->> Mesh info: nCellX = %d, nCellY = %d <<---" % (args
31                           .nCells[0], args.nCells[1]))
32        self.setExtents(args.simDomain[0], args.simDomain[1])
33
34        if args.showMeshInfo == 'yes':
35            print("P\n", self.P)
36            print("T\n", self.T)
37            print("\n-->> Pressure Mesh <<---")
38            print("pPb\n", self.pPb)
```

```

35     print("pTb\n", self.pTb)
36     print("\n-->> Velocity Mesh <<---")
37     print("vPb\n", self.vPb)
38     print("vTb\n", self.vTb)
39     print("Boundary Nodes for Pressure\n", self.nodeBCP)
40     print("Boundary Nodes for Velocity\n", self.nodeBCV)
41     print("Boundary Edges\n", self.edgeBC)

42
43
44     def setExtents(self, x0, xm):
45         self.x0 = x0
46         self.xm = xm
47         for i in range(len(self.dh)):
48             self.dh[i] = (self.xm[i] - self.x0[i]) / self.NE[i]

49
50         self.generatePTVertical()
51         self.generatePbTb()
52         self.boundaryGeneration(self.nodeBCP, fieldType='pressure')
53         self.boundaryGeneration(self.nodeBCV, fieldType='velocity')

54
55
56     def generatePTVertical(self):
57         for i in range(self.NN[0]):
58             for j in range(self.NN[1]):
59                 self.P[i * self.NN[1] + j][0] = self.x0[0] + i * self.
59                               dh[0]
60                 self.P[i * self.NN[1] + j][1] = self.x0[1] + j * self.
60                               dh[1]

61
62         tmp = np.zeros((self.NN[0], self.NN[1]))

63
64         for i in range(self.NN[0]):
65             for j in range(self.NN[1]):
66                 tmp[i][j] = i * self.NN[1] + j

67
68         for cellI in range(1, self.NE[0] * self.NE[1] + 1):
69             row, col = 0, 0
70             if cellI % self.NE[1] == 0:
71                 row = self.NE[1]
72                 col = cellI // self.NE[1]
73             else:
74                 row = cellI % self.NE[1]
75                 col = cellI // self.NE[1] + 1
76

```

```
77     if row - 1 < 0 or col - 1 < 0:
78         return
79
80     if row >= 1 and col >= 1:
81         self.T[2 * (cellI - 1)][0] = tmp[col - 1][row - 1]
82         self.T[2 * (cellI - 1)][1] = tmp[col][row - 1]
83         self.T[2 * (cellI - 1)][2] = tmp[col - 1][row]
84
85         self.T[2 * cellI - 1][0] = tmp[col - 1][row]
86         self.T[2 * cellI - 1][1] = tmp[col][row - 1]
87         self.T[2 * cellI - 1][2] = tmp[col][row]
88
89
90     def generatePbTb(self):
91         self.pPb = self.P
92         self.pTb = self.T
93
94         dh = [dx / 2 for dx in self.dh]
95         for nodeI in range(1, self.vPb.shape[0] + 1):
96             if nodeI % (2 * self.NE[1] + 1) == 0:
97                 self.vPb[nodeI - 1][0] = self.x0[0] + (nodeI / (2 *
98                                                 self.NE[1] + 1) - 1) * dh[0]
99                 self.vPb[nodeI - 1][1] = self.xm[1]
100            else:
101                self.vPb[nodeI - 1][0] = self.x0[0] + nodeI // (2 *
102                                                 self.NE[1] + 1) * dh[0]
103                self.vPb[nodeI - 1][1] = self.x0[1] + (nodeI % (2 *
104                                                 self.NE[1] + 1) - 1) * dh[1]
105
106            tmp = np.zeros((2 * self.NE[0] + 1, 2 * self.NE[1] + 1))
107            for i in range(tmp.shape[0]):
108                for j in range(tmp.shape[1]):
109                    tmp[i][j] = i * (2 * self.NE[1] + 1) + j
110
111            for cellI in range(1, self.NE[0] * self.NE[1] + 1):
112                row, col = 0, 0
113                if cellI % self.NE[1] == 0:
114                    row = self.NE[1]
115                    col = cellI // self.NE[1]
116                else:
117                    row = cellI % self.NE[1]
118                    col = cellI // self.NE[1] + 1
119
120                if row - 1 < 0 or col - 1 < 0:
```

```

118         return
119
120     if row >= 1 and col >= 1:
121         self.vTb[2 * (cellI - 1)][0] = tmp[2 * col - 2][2 *
122                                     row - 2]
122         self.vTb[2 * (cellI - 1)][1] = tmp[2 * col][2 * row -
123                                     2]
123         self.vTb[2 * (cellI - 1)][2] = tmp[2 * col - 2][2 *
124                                     row]
124         self.vTb[2 * (cellI - 1)][3] = tmp[2 * col - 1][2 *
125                                     row - 2]
125         self.vTb[2 * (cellI - 1)][4] = tmp[2 * col - 1][2 *
126                                     row - 1]
126         self.vTb[2 * (cellI - 1)][5] = tmp[2 * col - 2][2 *
127                                     row - 1]
127
128         self.vTb[2 * cellI - 1][0] = tmp[2 * col - 2][2 * row]
129         self.vTb[2 * cellI - 1][1] = tmp[2 * col][2 * row - 2]
130         self.vTb[2 * cellI - 1][2] = tmp[2 * col][2 * row]
131         self.vTb[2 * cellI - 1][3] = tmp[2 * col - 1][2 * row
132                                     - 1]
132         self.vTb[2 * cellI - 1][4] = tmp[2 * col][2 * row - 1]
133         self.vTb[2 * cellI - 1][5] = tmp[2 * col - 1][2 * row]
134
135
136     def boundaryGeneration(self, nodeBC, fieldType='pressure'):
137         if fieldType == 'pressure':
138             NE = self.NE
139             NN = self.NN
140         elif fieldType == 'velocity':
141             NE = [2 * self.NE[0], 2 * self.NE[1]]
142             NN = [ne + 1 for ne in NE]
143
144         # ---> 边界网格节点的定义 <---
145         # 将所有边界节点设置为 Dirichlet 边界类型
146         nodeBC[:, 0] = -1
147         nodeBC[:, 1] = -1
148
149         # bottom nodes
150         for i in range(NE[0]):
151             nodeBC[i][2] = i * NN[1]
152             nodeBC[i][3] = 0
153             nodeBC[i][4] = -1
154

```

```
155     # right nodes
156     for i in range(NE[0], NE[0] + NE[1]):
157         nodeBC[i][2] = NE[0] * NN[1] + i - NE[0]
158         nodeBC[i][3] = 1
159         nodeBC[i][4] = 0
160
161     # top nodes
162     for i in range(NE[0] + NE[1], 2 * NE[0] + NE[1]):
163         nodeBC[i][2] = (2 * NE[0] + NE[1] + 1 - i) * NN[1] - 1
164         nodeBC[i][3] = 0
165         nodeBC[i][4] = 1
166
167     # left nodes
168     for i in range(2 * NE[0] + NE[1], nodeBC.shape[0]):
169         nodeBC[i][2] = 2 * (NE[0] + NE[1]) - i
170         nodeBC[i][3] = -1
171         nodeBC[i][4] = 0
172
173     # left-bottom corner
174     nodeBC[0][3] = -1 / 2 ** 0.5
175     nodeBC[0][4] = -1 / 2 ** 0.5
176
177     # right-bottom corner
178     nodeBC[NE[0]][3] = 1 / 2 ** 0.5
179     nodeBC[NE[0]][4] = -1 / 2 ** 0.5
180
181     # right-top corner
182     nodeBC[NE[0]+NE[1]][3] = 1 / 2 ** 0.5
183     nodeBC[NE[0]+NE[1]][4] = 1 / 2 ** 0.5
184
185     # left-top corner
186     nodeBC[2*NE[0]+NE[1]][3] = -1 / 2 ** 0.5
187     nodeBC[2*NE[0]+NE[1]][4] = 1 / 2 ** 0.5
188
189     nodeBC[:,5] = -nodeBC[:,4]
190     nodeBC[:,6] = nodeBC[:,3]
191
192
193     # ---> 边界网格边的定义 <---#
194     # 初始化所有边界类型为 Dirichlet 类型边
195     self.edgeBC[:, 0] = -1
196     self.edgeBC[:, 1] = -1
197
198     # bottom edge
```

```

199     for i in range(1, self.NE[0] + 1):
200         self.edgeBC[i - 1][2] = (i - 1) * 2 * self.NE[1]
201         self.edgeBC[i - 1][3] = (i - 1) * (self.NE[1] + 1)
202         self.edgeBC[i - 1][4] = i * (self.NE[1] + 1)
203         self.edgeBC[i - 1][5] = 0
204         self.edgeBC[i - 1][6] = -1
205
206     # right edge
207     for i in range(self.NE[0] + 1, self.NE[0] + self.NE[1] + 1):
208         self.edgeBC[i - 1][2] = (self.NE[0] - 1) * 2 * self.NE[1] +
209                             2 * (i - self.NE[0]) - 1
210         self.edgeBC[i - 1][3] = self.NE[0] * (self.NE[1] + 1) + i -
211                             self.NE[0] - 1
212         self.edgeBC[i - 1][4] = self.NE[0] * (self.NE[1] + 1) + i -
213                             self.NE[0]
214         self.edgeBC[i - 1][5] = 1
215         self.edgeBC[i - 1][6] = 0
216
217     # top edge
218     for i in range(self.NE[0] + self.NE[1] + 1, 2 * self.NE[0] +
219                             self.NE[1] + 1):
220         self.edgeBC[i - 1][2] = (2 * self.NE[0] + self.NE[1] + 1 -
221                             i) * 2 * self.NE[1] - 1
222         self.edgeBC[i - 1][3] = (2 * self.NE[0] + self.NE[1] + 2 -
223                             i) * (self.NE[1] + 1) - 1
224         self.edgeBC[i - 1][4] = (2 * self.NE[0] + self.NE[1] + 1 -
225                             i) * (self.NE[1] + 1) - 1
226         self.edgeBC[i - 1][5] = 0
227         self.edgeBC[i - 1][6] = 1
228
229     # left edge
230     for i in range(2 * self.NE[0] + self.NE[1] + 1, self.edgeBC.
231                             shape[0] + 1):
232         self.edgeBC[i - 1][2] = 2 * (2 * self.NE[0] + 2 * self.NE[1] +
233                                     1 - i) - 2
234         self.edgeBC[i - 1][3] = 2 * self.NE[0] + 2 * self.NE[1] +
235                                     1 - i
236         self.edgeBC[i - 1][4] = 2 * self.NE[0] + 2 * self.NE[1] -
237                                     i
238         self.edgeBC[i - 1][5] = -1
239         self.edgeBC[i - 1][6] = 0
240
241         self.edgeBC[:, 7] = -self.edgeBC[:, 6]
242         self.edgeBC[:, 8] = self.edgeBC[:, 5]

```

B.4.3 linearSystem.py

```
1 import numpy as np
2 import scipy.sparse as sp
3 from math import *
4 import functions as fun
5 import postProcess
6 import time
7 np.set_printoptions(precision=6, suppress=True)
8
9 class LinearSystem:
10     def __init__(self, mesh, args) -> None:
11         self.mesh = mesh
12         self.args = args
13         self.gaussWeightRef = np.zeros(args.gaussPointNum)
14         self.gaussPointRef = np.zeros((args.gaussPointNum, 2))
15
16         self.gaussWeightLocal = np.zeros(args.gaussPointNum)
17         self.gaussPointLocal = np.zeros((args.gaussPointNum, 2))
18
19         self.Jacobian = np.zeros((2, 2))
20         self.JacobianInv = np.zeros((2, 2))
21         self.Jdet = 0
22
23         self mtxA = None
24         self mtxA1 = np.zeros((mesh.vPb.shape[0], mesh.vPb.shape[0]))
25         self mtxA2 = np.zeros((mesh.vPb.shape[0], mesh.vPb.shape[0]))
26         self mtxA3 = np.zeros((mesh.vPb.shape[0], mesh.vPb.shape[0]))
27         self mtxA4 = np.zeros((mesh.vPb.shape[0], mesh.vPb.shape[0]))
28         self mtxA5 = np.zeros((mesh.vPb.shape[0], mesh.pPb.shape[0]))
29         self mtxA6 = np.zeros((mesh.vPb.shape[0], mesh.pPb.shape[0]))
30         self mtxA7 = np.zeros((mesh.pPb.shape[0], mesh.vPb.shape[0]))
31         self mtxA8 = np.zeros((mesh.pPb.shape[0], mesh.vPb.shape[0]))
32
33         self vecB = None
34         self vecB1 = np.zeros(mesh.vPb.shape[0])
35         self vecB2 = np.zeros(mesh.vPb.shape[0])
36         self vecB3 = np.zeros(mesh.pPb.shape[0])
37
38         self.res = None
39         self.res1 = np.zeros(mesh.vPb.shape[0])
40         self.res2 = np.zeros(mesh.vPb.shape[0])
41         self.res3 = np.zeros(mesh.pPb.shape[0])
```

```
42     self.maxErrorU = 0
43     self.maxErrorP = 0
44     self.LinfinityU = 0
45     self.LinfinityP = 0
46     self.L2U = 0
47     self.L2P = 0
48     self.H1U = 0
49     self.H1P = 0
50
51
52     self.baseNumP = 3
53     self.baseNumV = 6
54
55     self.gaussWeightAndPointRef()
56
57
58     def gaussWeightAndPointRef(self):
59         if self.args.gaussPointNum == 3:
60             self.gaussWeightRef[0] = 1 / 6
61             self.gaussWeightRef[1] = 1 / 6
62             self.gaussWeightRef[2] = 1 / 6
63
64             self.gaussPointRef[0][0] = 0.5
65             self.gaussPointRef[0][1] = 0
66
67             self.gaussPointRef[1][0] = 0.5
68             self.gaussPointRef[1][1] = 0.5
69
70             self.gaussPointRef[2][0] = 0
71             self.gaussPointRef[2][1] = 0.5
72         elif self.args.gaussPointNum == 4:
73             self.gaussWeightRef[0] = (1 - 1 / sqrt(3)) / 8
74             self.gaussWeightRef[1] = (1 - 1 / sqrt(3)) / 8
75             self.gaussWeightRef[2] = (1 + 1 / sqrt(3)) / 8
76             self.gaussWeightRef[3] = (1 + 1 / sqrt(3.0)) / 8
77
78             self.gaussPointRef[0][0] = (1 / sqrt(3) + 1) / 2
79             self.gaussPointRef[0][1] = (
80                 1 - 1 / sqrt(3) * (1 + 1 / sqrt(3)) / 4
81
82             self.gaussPointRef[1][0] = (1 / sqrt(3) + 1) / 2
83             self.gaussPointRef[1][1] = (
84                 1 - 1 / sqrt(3) * (1 - 1 / sqrt(3)) / 4
85
```

```
86     self.gaussPointRef[2][0] = (-1 / sqrt(3) + 1) / 2
87     self.gaussPointRef[2][1] = (
88         1 + 1 / sqrt(3)) * (1 + 1 / sqrt(3)) / 4
89
90     self.gaussPointRef[3][0] = (-1 / sqrt(3) + 1) / 2
91     self.gaussPointRef[3][1] = (
92         1 + 1 / sqrt(3)) * (1 - 1 / sqrt(3)) / 4
93 elif self.args.gaussPointNum == 9:
94     self.gaussWeightRef[0] = 64 / 81 * 1 / 8
95     self.gaussWeightRef[1] = 100 / 324 * (1 - sqrt(3 / 5)) / 8
96     self.gaussWeightRef[2] = 100 / 324 * (1 - sqrt(3 / 5)) / 8
97     self.gaussWeightRef[3] = 100 / 324 * (1 + sqrt(3 / 5)) / 8
98     self.gaussWeightRef[4] = 100 / 324 * (1 + sqrt(3 / 5)) / 8
99     self.gaussWeightRef[5] = 40 / 81 * 1 / 8
100    self.gaussWeightRef[6] = 40 / 81 * 1 / 8
101    self.gaussWeightRef[7] = 40 / 81 * (1 - sqrt(3 / 5)) / 8
102    self.gaussWeightRef[8] = 40 / 81 * (1 + sqrt(3 / 5)) / 8
103
104    self.gaussPointRef[0][0] = 1 / 2
105    self.gaussPointRef[0][1] = 1 / 4
106
107    self.gaussPointRef[1][0] = (1.0 + sqrt(3.0 / 5.0)) / 2.0
108    self.gaussPointRef[1][1] = (1.0 - sqrt(3.0 / 5.0)) * (1.0
109                                + sqrt(3.0 / 5.0)) / 4.0
110
111    self.gaussPointRef[2][0] = (1.0 + sqrt(3.0 / 5.0)) / 2.0
112    self.gaussPointRef[2][1] = (1.0 - sqrt(3.0 / 5.0)) * (1.0
113                                - sqrt(3.0 / 5.0)) / 4.0
114
115    self.gaussPointRef[3][0] = (1.0 - sqrt(3.0 / 5.0)) / 2.0
116    self.gaussPointRef[3][1] = (1.0 + sqrt(3.0 / 5.0)) * (1.0
117                                - sqrt(3.0 / 5.0)) / 4.0
118
119    self.gaussPointRef[4][0] = (1.0 - sqrt(3.0 / 5.0)) / 2.0
120    self.gaussPointRef[4][1] = (1.0 + sqrt(3.0 / 5.0)) * (1.0
121                                - sqrt(3.0 / 5.0)) / 4.0
122
123    self.gaussPointRef[5][0] = (1.0 + 0.0) / 2.0
124    self.gaussPointRef[5][1] = (1.0 - 0.0) * (1.0 + sqrt(3.0 /
125                                5.0)) / 4.0
126
127    self.gaussPointRef[6][0] = (1.0 + 0.0) / 2.0
128    self.gaussPointRef[6][1] = (1.0 - 0.0) * (1.0 - sqrt(3.0 /
129                                5.0)) / 4.0
```

```

124
125     self.gaussPointRef[7][0] = (1.0 + sqrt(3.0 / 5.0)) / 2.0
126     self.gaussPointRef[7][1] = (1.0 - sqrt(3.0 / 5.0)) * (1.0
127                               + 0.0) / 4.0
128
129     self.gaussPointRef[8][0] = (1.0 - sqrt(3.0 / 5.0)) / 2.0
130     self.gaussPointRef[8][1] = (1.0 + sqrt(3.0 / 5.0)) * (1.0
131                               + 0.0) / 4.0
132
133     def gaussWeightAndPointLocal(self, ver):
134         x1, y1 = ver[0][0], ver[0][1]
135         x2, y2 = ver[1][0], ver[1][1]
136         x3, y3 = ver[2][0], ver[2][1]
137         self.Jdet = abs((x2 - x1) * (y3 - y1) - (x3 - x1) * (y2 - y1))
138
139         for i in range(self.args.gaussPointNum):
140             self.gaussWeightLocal[i] = self.Jdet * self.gaussWeightRef
141                               [i]
142             self.gaussPointLocal[i][0] = x1 + (x2 - x1) * self.
143                               gaussPointRef[i][0] + (
144                               x3 - x1) * self.gaussPointRef[i][1]
145             self.gaussPointLocal[i][1] = y1 + (y2 - y1) * self.
146                               gaussPointRef[i][0] + (
147                               y3 - y1) * self.gaussPointRef[i][1]
148
149     def refBaseFunction(self, x, dD, bIx, bTp):
150         res = 0
151         if bTp == "pressure":
152             if dD[0] == 0 and dD[1] == 0:
153                 if bIx == 0:
154                     res = 1 - x[0] - x[1]
155                 elif bIx == 1:
156                     res = x[0]
157                 elif bIx == 2:
158                     res = x[1]
159             elif dD[0] == 1 and dD[1] == 0:
160                 if bIx == 0:
161                     res = -1
162                 elif bIx == 1:
163                     res = 1
164                 elif bIx == 2:
165                     res = 0

```

```
163         elif dD[0] == 0 and dD[1] == 1:
164             if bIx == 0:
165                 res = -1
166             elif bIx == 1:
167                 res = 0
168             elif bIx == 2:
169                 res = 1
170         else:
171             return 0
172     return res
173 elif bTp == "velocity":
174     if dD[0] == 0 and dD[1] == 0:
175         if bIx == 0:
176             res = 2 * pow(x[0], 2) + 2 * pow(x[1], 2) + 4 * x[
177                 0] * x[1] - 3 * x[0] - 3 * x
178                 [1] + 1
179         elif bIx == 1:
180             res = 2 * pow(x[0], 2) - x[0]
181         elif bIx == 2:
182             res = 2 * pow(x[1], 2) - x[1]
183         elif bIx == 3:
184             res = -4 * pow(x[0], 2) - 4 * x[0] * x[1] + 4 * x[
185                 0]
186         elif bIx == 4:
187             res = 4 * x[0] * x[1]
188         elif bIx == 5:
189             res = -4 * pow(x[1], 2) - 4 * x[0] * x[1] + 4 * x[
190                 1]
191     elif dD[0] == 1 and dD[1] == 0:
192         if bIx == 0:
193             res = 4 * x[0] + 4 * x[1] - 3
194         elif bIx == 1:
195             res = 4 * x[0] - 1
196         elif bIx == 2:
197             res = 0
198         elif bIx == 3:
199             res = -8 * x[0] - 4 * x[1] + 4
200         elif bIx == 4:
201             res = 4 * x[1]
202         elif bIx == 5:
203             res = -4 * x[1]
204     elif dD[0] == 0 and dD[1] == 1:
205         if bIx == 0:
206             res = 4 * x[1] + 4 * x[0] - 3
```

```
203     elif bIx == 1:  
204         res = 0  
205     elif bIx == 2:  
206         res = 4 * x[1] - 1  
207     elif bIx == 3:  
208         res = -4 * x[0]  
209     elif bIx == 4:  
210         res = 4 * x[0]  
211     elif bIx == 5:  
212         res = -8 * x[1] - 4 * x[0] + 4  
213 elif dD[0] == 2 and dD[1] == 0:  
214     if bIx == 0:  
215         res = 4  
216     elif bIx == 1:  
217         res = 4  
218     elif bIx == 2:  
219         res = 0  
220     elif bIx == 3:  
221         res = -8  
222     elif bIx == 4:  
223         res = 0  
224     elif bIx == 5:  
225         res = 0  
226 elif dD[0] == 0 and dD[1] == 2:  
227     if bIx == 0:  
228         res = 4  
229     elif bIx == 1:  
230         res = 0  
231     elif bIx == 2:  
232         res = 4  
233     elif bIx == 3:  
234         res = 0  
235     elif bIx == 4:  
236         res = 0  
237     elif bIx == 5:  
238         res = -8  
239 elif dD[0] == 1 and dD[1] == 1:  
240     if bIx == 0:  
241         res = 4  
242     elif bIx == 1:  
243         res = 0  
244     elif bIx == 2:  
245         res = 0  
246     elif bIx == 3:
```

```
247             res = -4
248         elif bIx == 4:
249             res = 4
250         elif bIx == 5:
251             res = -4
252     return res
253
254
255     def localBaseFun(self, x, ver, dD, bIx, bTp):
256         J00 = ver[1][0] - ver[0][0]
257         J01 = ver[2][0] - ver[0][0]
258         J10 = ver[1][1] - ver[0][1]
259         J11 = ver[2][1] - ver[0][1]
260         Jdet = J00 * J11 - J01 * J10
261
262         xHat = (J11 * (x[0] - ver[0][0]) - J01 * (x[1] - ver[0][1])) /
263                                     Jdet
264         yHat = (-J10 * (x[0] - ver[0][0]) + J00 * (x[1] - ver[0][1])) /
265                                     Jdet
266
267         res = 0
268         if dD[0] == 0 and dD[1] == 0:
269             res = self.refBaseFunction([xHat, yHat], [0, 0], bIx, bTp)
270         elif dD[0] == 1 and dD[1] == 0:
271             res = (self.refBaseFunction([xHat, yHat], [1, 0], bIx, bTp)
272                   * J11 + self.refBaseFunction
273                   ([xHat, yHat], [0, 1], bIx,
274                     bTp) * (-J10)) / Jdet
275         elif dD[0] == 0 and dD[1] == 1:
276             res = (self.refBaseFunction([xHat, yHat], [1, 0], bIx, bTp)
277                   * (-J01) + self.
278                   refBaseFunction([xHat, yHat]
279                   , [0, 1], bIx, bTp) * J00) /
280                         Jdet
281         elif dD[1] == 2 and dD[0] == 0:
282             res = (self.refBaseFunction([xHat, yHat], [2, 0], bIx, bTp)
283                   * pow(J11, 2) + self.
284                   refBaseFunction([xHat, yHat]
285                   , [0, 2], bIx, bTp) * pow(
286                   J10, 2) + self.
287                   refBaseFunction([xHat, yHat]
288                   , [1, 1], bIx, bTp) * (-2 *
289                     J10 * J11)) / pow(Jdet, 2)
290         elif dD[0] == 0 and dD[1] == 2:
```

```

275     res=(self.refBaseFunction([xHat, yHat], [2, 0], bIx, bTp)
276         * pow(J01, 2) + self.
277         refBaseFun([xHat, yHat], [0,
278             2], bIx, bTp) * pow(J00, 2)
279         + self.refBaseFunction([
280             xHat, yHat], [1, 1], bIx,
281             bTp) * (-2 * J00 * J01)) /
282             pow(Jdet, 2)
283
284     elif dD[0] == 1 and dD[1] == 1:
285         res = (self.refBaseFunction([xHat, yHat], [2, 0], bIx, bTp
286             ) * (-J11 * J01) + self.
287             refBaseFunction([xHat, yHat]
288                 , [0, 2], bIx, bTp) * (-J10
289                 * J00) + self.
290                 refBaseFunction([xHat, yHat]
291                     , [1, 1], bIx, bTp) * (J10 *
292                         J01 + J00 * J11)) / pow(
293                         Jdet, 2)
294
295     return res
296
297
298
299 # 刚度矩阵组装功能函数
300
301 def gaussInteTrialAndTest(self, ver, bIx, trialdD, testdD,
302                             function, bTp):
303
304     res = 0
305     self.gaussWeightAndPointLocal(ver)
306
307     for i in range(self.args.gaussPointNum):
308         res += self.gaussWeightLocal[i] * function(self.
309             gaussPointLocal[i]) * self.
310             localBaseFun(self.
311                 gaussPointLocal[i], ver,
312                 trialdD, bIx[0], bTp[0]) *
313                 self.localBaseFun(self.
314                     gaussPointLocal[i], ver,
315                     testdD, bIx[1], bTp[1])
316
317
318     return res
319
320
321
322 def assembleMtxA(self, func, trialdD, testdD, PbTrial, PbTest,
323                   TbTrial, TbTest,
324                   trialBasisNum, testBasisNum,
325                   bTp):

```



```
316         baseNumV, ["pressure", "velocity"])
317     self mtxA6 = self assembleMtxA(fun.funC, [0, 0], [0, 1], self.
318                                     mesh.pPb, self.mesh.vPb,
319                                     self.mesh.pTb, self.mesh.vTb
320                                     , self.baseNumP, self.
321                                     baseNumV, ["pressure", "velocity"])
322     self mtxA = np.block([[2*self mtxA1+self mtxA2, self mtxA3,
323                           self mtxA5], [self mtxA3.T,
324                           2*self mtxA2+self mtxA1,
325                           self mtxA6], [self mtxA5.T,
326                           self mtxA6.T, np.zeros((self.
327                                     .mesh.pPb.shape[0], self.
328                                     mesh.pPb.shape[0]))]])
328
329     timeEnd = time.time()
330     print("----> Time = %f s <----" % (timeEnd - timeStart))
331     if self.args.showMtxBeforeBC == 'yes':
332         print("----> Matrix Before BC treatment <----")
333         print("mtxA1.shape = ", self mtxA1.shape)
334         print(self mtxA1)
335         print()
336         print("mtxA2.shape = ", self mtxA2.shape)
337         print(self mtxA2)
338         print()
339         print("mtxA3.shape = ", self mtxA3.shape)
340         print(self mtxA3)
341         print()
342         print("mtxA5.shape = ", self mtxA5.shape)
343         print(self mtxA5)
344         print()
345         print("mtxA6.shape = ", self mtxA6.shape)
346         print(self mtxA6)
347         print()
348         print("mtxA.shape = ", self mtxA.shape)
349         print(self mtxA)
350
351
352     # 载荷向量组装功能函数
353     def gaussInteTest(self, ver, teBIX, function, bTp):
354         res = 0
355
356         self.gaussWeightAndPointLocal(ver)
357
```

```
348     for i in range(self.args.gaussPointNum):
349         tmp = function(self.gaussPointLocal[i])
350         tmpBase = self.localBaseFun(self.gaussPointLocal[i], ver,
351                                     [0, 0], teBIdx, bTp)
352         res += self.gaussWeightLocal[i] * tmp * tmpBase
353
354
355
356     def assembleVectorB(self, func, PbTest, TbTest, testBasisNum, bTp):
357         :
358         tmpVecB = np.zeros(PbTest.shape[0])
359         for n in range(TbTest.shape[0]):
360             ver = np.zeros((TbTest.shape[1], 2))
361             for i in range(TbTest.shape[1]):
362                 ver[i][0] = PbTest[TbTest[n][i]][0]
363                 ver[i][1] = PbTest[TbTest[n][i]][1]
364
365             for beta in range(testBasisNum):
366                 tmp = self.gaussInteTest(ver, beta, func, bTp)
367                 tmpVecB[TbTest[n][beta]] += tmp
368
369
370     def assembleAllVectorB(self):
371         print("\n-->> Assembling Vector <<---")
372         timeStart = time.time()
373         self.vecB1 = self.assembleVectorB(fun.funF1, self.mesh.vPb,
374                                         self.mesh.vTb, self.baseNumV
375                                         , "velocity")
376         self.vecB2 = self.assembleVectorB(fun.funF2, self.mesh.vPb,
377                                         self.mesh.vTb, self.baseNumV
378                                         , "velocity")
379         self.vecB = np.concatenate((self.vecB1, self.vecB2, np.zeros(
380                                         self.mesh.pPb.shape[0])))
381
382         timeEnd = time.time()
383         print("---->> Time = %f s <<---" % (timeEnd - timeStart))
384         if self.args.showVecBeforeBC == 'yes':
385             print("---->> Vector Before BC treatment <<---")
386             print("vecB1.shape = ", self.vecB1.shape)
387             print(self.vecB1)
388             print()
389             print("vecB2.shape = ", self.vecB2.shape)
390             print(self.vecB2)
```

```

385     print()
386     print("vecB.shape = ", self.vecB.shape)
387     print(self.vecB)
388     print()
389
390
391     def boundaryTreatment(self):
392         print("\n-->> Boundary Treatment <<---")
393         timeStart = time.time()
394         # 处理速度边界
395         for k in range(self.mesh.nodeBCV.shape[0]):
396             i = int(self.mesh.nodeBCV[k][2])
397             # 处理 x 方向速度的边界条件
398             if self.mesh.nodeBCV[k][0] == -1:
399                 self mtxA[i, :] = 0
400                 self mtxA[i][i] = 1
401                 self.vecB[i] = fun.exactU(self.mesh.vPb[i])
402
403             # 处理 y 方向速度的边界条件
404             if self.mesh.nodeBCV[k][1] == -1:
405                 iy = i + self.mesh.vPb.shape[0]
406                 self mtxA[iy, :] = 0
407                 self mtxA[iy][iy] = 1
408                 self.vecB[iy] = fun.exactV(self.mesh.vPb[i])
409
410         # 处理压力边界
411         for k in range(self.mesh.nodeBCP.shape[0]):
412             i = int(self.mesh.nodeBCP[k][2])
413             if self.mesh.nodeBCP[k][0] == -1 and self.mesh.pPb[i][0] ==
414                 = 0 and self.mesh.pPb[i][1]
415                 == 0:
416                 print("压力参考点: ", self.mesh.pPb[i])
417                 ip = i + 2 * self.mesh.vPb.shape[0]
418                 self mtxA[ip, :] = 0
419                 self mtxA[ip][ip] = 1
420                 self.vecB[ip] = fun.exactP(self.mesh.pPb[i])
421             timeEnd = time.time()
422             print("---->> Time = %f s <<---" % (timeEnd - timeStart))
423
424             if self.args.showMtxAfterBC == 'yes':
425                 print("---->> Matrix After BC treatment <<---")
426                 print(self mtxA)
427
428             if self.args.showVecAfterBC == 'yes':

```

```
427     print("\n--->> Vector After BC treatment <<---")
428     print(self.vecB)
429
430
431     def solveLinearSystem(self):
432         timeStart = time.time()
433         print("\n--->> Linear System Solving <<---")
434         self.res = np.linalg.solve(self mtxA, self.vecB)
435         self.res1 = self.res[:self.mesh.vPb.shape[0]]
436         self.res2 = self.res[self.mesh.vPb.shape[0]:self.mesh.vPb.
437                             shape[0]*2]
438         self.res3 = self.res[2*self.mesh.vPb.shape[0]:]
439         timeEnd = time.time()
440         print("--->> Simulation Done !!! <<---")
441         print("--->> Time = %f s <<---" % (timeEnd - timeStart))
442
443         self.errorCalculation()
444
445         if self.args.printRes == 'yes':
446             print("\n--->> Result <<---")
447             print("ux")
448             print(self.res1)
449             print("\nuy")
450             print(self.res2)
451             print("\npressure")
452             print(self.res3)
453
454     """
455     误差分析
456     """
457     def infinityFeRes(self, x, uhLocal, ver, dD, baseNum, bTp):
458         res = 0
459         for i in range(baseNum):
460             res += uhLocal[i] * self.localBaseFun(x, ver, dD, i, bTp)
461         return res
462
463
464     def gaussIntegralFeRes(self, uhLocal, ver, function, dD, baseNum,
465                           bTp):
466         res = 0
467         self.gaussWeightAndPointLocal(ver)
468         for i in range(self.args.gaussPointNum):
```

```

468     res += self.gaussWeightLocal[i] * pow(function(self.
469                                             gaussPointLocal[i]) - self.
470                                             infinityFeRes(self.
471                                             gaussPointLocal[i], uhLocal,
472                                             ver, dD, baseNum, bTp), 2)
473
474     return res
475
476
477
478 def maxErrorCompute(self, resVec, func, Pb):
479     res = abs(func(Pb[0]) - resVec[0])
480
481     for i in range(1, Pb.shape[0]):
482         tmp = abs(func(Pb[i]) - resVec[i])
483
484         if tmp > res:
485             res = tmp
486
487     return res
488
489
490
491 def errorInfinityNorm(self, resVec, dD, func, Pb, Tb, baseNum, bTp
492                         ):
493     res = 0
494     uhLocal = [0 for i in range(baseNum)]
495
496     for n in range(Tb.shape[0]):
497         ver = np.zeros((Tb.shape[1], 2))
498
499         for i in range(Tb.shape[1]):
500             ver[i][0] = Pb[Tb[n][i]][0]
501             ver[i][1] = Pb[Tb[n][i]][1]
502
503         self.gaussWeightAndPointLocal(ver)
504
505         for i in range(baseNum):
506             uhLocal[i] = resVec[Tb[n][i]]
507
508         tmp = 0
509
510         for i in range(self.args.gaussPointNum):
511             value = abs(func(self.gaussPointLocal[i]) - self.
512                         infinityFeRes(self.
513                                         gaussPointLocal[i], uhLocal,
514                                         ver, dD, baseNum, bTp))
515
516             if tmp < value:
517                 tmp = value
518
519             if res < tmp:
520                 res = tmp
521
522     return res
523
524
525 def L2AndH1NormError(self, resVec, function, dD, baseNum, Pb, Tb,
526                      bTp):

```

```

503 uhLocal = [0 for i in range(baseNum)]
504 res = 0
505 for n in range(Tb.shape[0]):
506     ver = np.zeros((Tb.shape[1], 2))
507     for i in range(Tb.shape[1]):
508         ver[i][0] = Pb[Tb[n][i]][0]
509         ver[i][1] = Pb[Tb[n][i]][1]
510     for i in range(baseNum):
511         uhLocal[i] = resVec[Tb[n][i]]
512     res += self.gaussIntegralFeRes(uhLocal, ver, function, dD,
513                                     baseNum, bTp)
514
515
516
517 def errorCalculation(self):
518     # 最大节点值误差
519     uxMax = self.maxErrorCompute(self.res1, fun.exactU, self.mesh.
520                                   vPb)
521     uyMax = self.maxErrorCompute(self.res2, fun.exactV, self.mesh.
522                                   vPb)
523     self.maxErrorU = max(uxMax, uyMax)
524     self.maxErrorP = self.maxErrorCompute(self.res3, fun.exactP,
525                                         self.mesh.pPb)
526
527     # 无穷范数误差
528     xLinfinity = self.errorInfinityNorm(self.res1, [0, 0], fun.
529                                           exactU, self.mesh.vPb, self.
530                                           mesh.vTb, self.baseNumV, "
531                                           velocity")
532     yLinfinity = self.errorInfinityNorm(self.res2, [0, 0], fun.
533                                           exactV, self.mesh.vPb, self.
534                                           mesh.vTb, self.baseNumV, "
535                                           velocity")
536     self.LinfinityU = max(xLinfinity, yLinfinity)
537     self.LinfinityP = self.errorInfinityNorm(self.res3, [0, 0],
538                                             fun.exactP, self.mesh.pPb,
539                                             self.mesh.pTb, self.baseNumP,
540                                             "pressure")
541
542     # L2 范数误差
543     xL2 = self.L2AndH1NormError(self.res1, fun.exactU, [0, 0],
544                                 baseNum=self.baseNumV, Pb=
545                                 self.mesh.vPb, Tb=self.mesh.
546                                 vTb)

```

```

532                                     vTb, bTp="velocity")
533     yL2 = self.L2AndH1NormError(self.res2, fun.exactV, [0, 0],
534                                   baseNum=self.baseNumV, Pb=
535                                   self.mesh.vPb, Tb=self.mesh.
536                                   vTb, bTp="velocity")
537     self.L2U = (xL2 ** 2 + yL2 ** 2) ** 0.5
538     self.L2P = self.L2AndH1NormError(self.res3, fun.exactP, [0, 0]
539                                   , baseNum=self.baseNumP, Pb=
540                                   self.mesh.pPb, Tb=self.mesh.
541                                   pTb, bTp="pressure")
542
543     # H1 范数误差
544     xH1 = sqrt(pow(self.L2AndH1NormError(self.res1, fun.exactUxDx,
545                      [1, 0], self.baseNumV, self.
546                      .mesh.vPb, self.mesh.vTb, "
547                      velocity"),2) + pow(self.
548                      L2AndH1NormError(self.res1,
549                      fun.exactUxDy, [0, 1], self.
550                      baseNumV, self.mesh.vPb,
551                      self.mesh.vTb, "velocity"),2
552                      ))
553     yH1 = sqrt(pow(self.L2AndH1NormError(self.res2, fun.exactUyDx,
554                      [1, 0], self.baseNumV, self.
555                      .mesh.vPb, self.mesh.vTb, "
556                      velocity"),2) + pow(self.
557                      L2AndH1NormError(self.res2,
558                      fun.exactUyDy, [0, 1], self.
559                      baseNumV, self.mesh.vPb,
560                      self.mesh.vTb, "velocity"),2
561                      ))
562     self.H1U = (xH1 ** 2 + yH1 ** 2) ** 0.5
563     self.H1P = sqrt(pow(self.L2AndH1NormError(self.res3, fun.
564                           exactPDx, [1, 0], self.
565                           baseNumP, self.mesh.pPb,
566                           self.mesh.pTb, "pressure"),2
567                           ) + pow(self.
568                           L2AndH1NormError(self.res3,
569                           fun.exactPDy, [0, 1], self.
570                           baseNumP, self.mesh.pPb,
571                           self.mesh.pTb, "pressure"),2
572                           ))
573
574     if self.args.printError == 'yes':
575         print("\n-->> Error Information <<---")

```

```
544     print("Error for Velocity:")
545     print("maxErrorU = %.4e" % self.maxErrorU)
546     print("LinfinityU = %.4e" % self.LinfinityU)
547     print("L2U = %.4e" % self.L2U)
548     print("H1U = %.4e" % self.H1U)
549
550     print("\nError for Pressure:")
551     print("maxErrorP = %.4e" % self.maxErrorP)
552     print("LinfinityP = %.4e" % self.LinfinityP)
553     print("L2P = %.4e" % self.L2P)
554     print("H1P = %.4e" % self.H1P)
```

B.4.4 functions.py

```
1 import Mesh
2 from math import *
3
4
5 def funNu(x):
6     return 1
7
8
9 def funC(x):
10    return -1
11
12
13 def funF1(x):
14    nu=funNu(x)
15    res=-2*nu*pow(x[0],2)-2*nu*pow(x[1],2)-nu*exp(-x[1])+pow(pi,2)*cos(pi*x[0])*cos(2*pi*x[1])
16    return res
17
18
19 def funF2(x):
20    nu=funNu(x)
21    res=4*nu*x[0]*x[1]-nu*pow(pi,3)*sin(pi*x[0])+2*pi*(2-pi*sin(pi*x[0]))*sin(2*pi*x[1])
22    return res
23
24
25 def exactU(x):
26     return pow(x[0],2)*pow(x[1],2)+exp(-x[1])
```

```

27
28
29 def exactV(x):
30     return -2.0/3.0*x[0]*pow(x[1],3)+2*pi*sin(pi*x[0])
31
32
33 def exactP(x):
34     return -(2*pi*sin(pi*x[0]))*cos(2*pi*x[1])
35
36
37 def exactUxDx(x):
38     return 2*x[0]*pow(x[1],2)
39
40
41 def exactUxDy(x):
42     return 2*pow(x[0],2)*x[1]-exp(-x[1])
43
44
45 def exactUyDx(x):
46     return -2.0/3.0*pow(x[1],3)-pow(pi,2)*cos(pi*x[0])
47
48
49 def exactUyDy(x):
50     return -2*x[0]*pow(x[1],2)
51
52
53 def exactPDx(x):
54     return pow(pi,2)*cos(pi*x[0])*cos(2*pi*x[1])
55
56
57 def exactPDy(x):
58     return -2*pi*(pi*sin(pi*x[0])-2)*sin(2*pi*x[1])

```

B.4.5 postProcess.py

```

1 import matplotlib.pyplot as plt
2 import matplotlib.tri as mtri
3 import numpy as np
4 fontStyle = {'family':'DejaVu Sans', 'style':'italic', 'weight':'
               'normal', 'size':25}
5
6 def PlotResNoMesh(mesh,resVec,Pb,fileNames="u",bTp='pressure'):

```



```

45     minRes,maxRes = 1e10,-1e10
46     for j in y:
47         tmp = []
48         for i in x:
49             value = func([i, j])
50             if value < minRes:
51                 minRes = value
52             if value > maxRes:
53                 maxRes = value
54             tmp.append(value)
55         res.append(tmp)
56     x,y=np.meshgrid(x,y)
57     fig, ax = plt.subplots(figsize=(10, 5))
58     ax.tick_params("both", which='major', length=5, colors='k',
59                     direction='in', pad=12)
60     plot = plt.contourf(x,y,res,12,cmap='jet',vmin=minRes,vmax=maxRes)
61     ax.set_xlabel('x', fontproperties=fontStyle)
62     ax.set_ylabel('y', fontproperties=fontStyle)
63     xTicks=np.linspace(args.simDomain[0][0], args.simDomain[1][0], 5)
64     yTicks=np.linspace(args.simDomain[0][1], args.simDomain[1][1], 5)
65     plt.xticks(xTicks)
66     plt.yticks(yTicks)
67     plt.setp(ax.get_xticklabels(), fontproperties=fontStyle)
68     plt.setp(ax.get_yticklabels(), fontproperties=fontStyle)
69     colorBar = fig.colorbar(plot)
70     colorBar.ax.set_title(fileName, fontproperties=fontStyle)
71     colorBar.ax.tick_params(direction = 'in', labelsize=fontStyle['
72                                         size'])
73     plt.tight_layout()
74     plt.show()
75     plt.ioff()

```

B.5 第六章参考程序

B.5.1 2D_FEM.py

```

1 import Mesh
2 import linearSystem
3 import postProcess
4
5 '''
6 运行指令示范:

```

```

7  python .\2D_FEM.py --femInfo LinearTri,2,3
8  python .\2D_FEM.py --femInfo QuadraticTri,2,3
9  '''
10
11 if __name__ == "__main__":
12     setInfo = {
13         "description": "2D-Parabolic, case-1, LinearTri Dirichlet",
14         "simDomain": [[0, 0], [2, 1]],
15         "simTime": [0, 1],
16         "nCells": [16, 8],
17         "basisType": "LinearTri",
18         "theta": 0.5,
19         "Nt": 8,
20         "showMtxBeforeBC": "no",
21         "showVecBeforeBC": "no",
22         "showMtxAfterBC": "no",
23         "showVecAfterBC": "no",
24         "printRes": "no",
25         "printError": "yes",
26         "plotResNoMesh": "no",
27         "plotAnalyticalRes": "no",
28         "gaussPointNum": 9}
29     mesh = Mesh.Mesh(setInfo["nCells"], setInfo["basisType"], setInfo)
30     system = linearSystem.LinearSystem(mesh, 9, setInfo)
31     system.assembleAllMtxA()
32     system.parabolicSolver()
33
34     # 数据后处理
35     if setInfo["plotResNoMesh"] == "yes":
36         postProcess.PlotResNoMesh(mesh, system)
37
38     if setInfo["plotAnalyticalRes"] == "yes":
39         postProcess.PlotAnalyticalRes(setInfo)

```

B.5.2 Mesh.py

```

1 import numpy as np
2
3 class Mesh:
4     def __init__(self, NE, bTp, setInfo):
5         self.x0 = [0, 0]
6         self.xm = [0, 0]

```

```

7     self.dh = [0, 0]
8     self.NE = NE
9     self.NN = [NE[0] + 1, NE[1] + 1]
10    self.P = np.zeros((NE[0] + 1) * (NE[1] + 1), 2)
11    self.nodeBC = np.zeros((2 * (NE[0] + NE[1]), 2), dtype=int)
12    self.edgeBC = np.zeros((2 * (NE[0] + NE[1]), 4), dtype=int)
13    self.bTp = bTp
14    print("<< Using " + setInfo["description"] + " <---")
15    print("--> Mesh info: nCell=[%d, %d] <---" % (setInfo[""
16                                              "nCells"][0], setInfo["nCells"
17                                              "[1]]))
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44

```

```

self.dh = [0, 0]
self.NE = NE
self.NN = [NE[0] + 1, NE[1] + 1]
self.P = np.zeros((NE[0] + 1) * (NE[1] + 1), 2)
self.nodeBC = np.zeros((2 * (NE[0] + NE[1]), 2), dtype=int)
self.edgeBC = np.zeros((2 * (NE[0] + NE[1]), 4), dtype=int)
self.bTp = bTp
print("<< Using " + setInfo["description"] + " <---")
print("--> Mesh info: nCell=[%d, %d] <---" % (setInfo[""
                                              "nCells"][0], setInfo["nCells"
                                              "[1]]))
if bTp == "LinearTri":
    self.T = np.zeros((2 * NE[0] * NE[1], 3), dtype=int)
    self.Pb = self.P
    self.Tb = self.T
elif bTp == "QuadraticTri":
    self.T = np.zeros((2 * NE[0] * NE[1], 3), dtype=int)
    self.Pb = np.zeros(((2 * NE[0] + 1) * (2 * NE[1] + 1), 2))
    self.Tb = np.zeros((2 * NE[0] * NE[1], 6), dtype=int)
    self.nodeBC = np.zeros((4 * (NE[0] + NE[1]), 3), dtype=int
                           )
self.setExtents(setInfo["simDomain"][0], setInfo["simDomain"]
               [1])
def setExtents(self, x0, xm):
    self.x0 = x0
    self.xm = xm
    for i in range(len(self.dh)):
        self.dh[i] = (self.xm[i] - self.x0[i]) / self.NE[i]
    self.generatePTVertical()
    self.generatePbTb()
    self.boundaryGeneration()

def generatePTVertical(self):
    for i in range(self.NN[0]):
        for j in range(self.NN[1]):
            self.P[i * self.NN[1] + j][0] = self.x0[0] + i * self.
                                             dh[0]
            self.P[i * self.NN[1] + j][1] = self.x0[1] + j * self.
                                             dh[1]
    if self.bTp == "LinearTri" or self.bTp == "QuadraticTri":
```

```
45     tmp = np.zeros((self.NN[0], self.NN[1]))
46     for i in range(self.NN[0]):
47         for j in range(self.NN[1]):
48             tmp[i][j] = i * self.NN[1] + j
49     for cellI in range(1, self.NE[0] * self.NE[1] + 1):
50         row, col = 0, 0
51         if cellI % self.NE[1] == 0:
52             row = self.NE[1]
53             col = cellI // self.NE[1]
54         else:
55             row = cellI % self.NE[1]
56             col = cellI // self.NE[1] + 1
57         if row - 1 < 0 or col - 1 < 0:
58             return
59         if row >= 1 and col >= 1:
60             self.T[2 * (cellI - 1)][0] = tmp[col - 1][row - 1]
61             self.T[2 * (cellI - 1)][1] = tmp[col][row - 1]
62             self.T[2 * (cellI - 1)][2] = tmp[col - 1][row]
63             self.T[2 * cellI - 1][0] = tmp[col - 1][row]
64             self.T[2 * cellI - 1][1] = tmp[col][row - 1]
65             self.T[2 * cellI - 1][2] = tmp[col][row]
66
67
68     def generatePbTb(self):
69         if self.bTp == "LinearTri":
70             self.Pb = self.P
71             self.Tb = self.T
72         elif self.bTp == "QuadraticTri":
73             dh = [dx / 2 for dx in self.dh]
74             for nodeI in range(1, self.Pb.shape[0] + 1):
75                 if nodeI % (2 * self.NE[1] + 1) == 0:
76                     self.Pb[nodeI - 1][0] = self.x0[0] + (nodeI / (2 *
77                                         self.NE[1] + 1) - 1) * dh[0]
78
79                     ]
80                     self.Pb[nodeI - 1][1] = self.xm[1]
81                 else:
82                     self.Pb[nodeI - 1][0] = self.x0[0] + nodeI // (2 *
83                                         self.NE[1] + 1) * dh[0]
84                     self.Pb[nodeI - 1][1] = self.x0[1] + (nodeI % (2 *
85                                         self.NE[1] + 1) - 1) * dh[1]
86
87
88     tmp = np.zeros((2 * self.NE[0] + 1, 2 * self.NE[1] + 1))
89     for i in range(tmp.shape[0]):
```

```

84     for j in range(tmp.shape[1]):
85         tmp[i][j] = i * (2 * self.NE[1] + 1) + j
86
87     for cellI in range(1, self.NE[0] * self.NE[1] + 1):
88         row, col = 0, 0
89         if cellI % self.NE[1] == 0:
90             row = self.NE[1]
91             col = cellI // self.NE[1]
92         else:
93             row = cellI % self.NE[1]
94             col = cellI // self.NE[1] + 1
95
96         if row - 1 < 0 or col - 1 < 0:
97             return
98
99         if row >= 1 and col >= 1:
100             self.Tb[2 * (cellI - 1)][0] = tmp[2 * col - 2][2 *
101                                         row - 2]
102             self.Tb[2 * (cellI - 1)][1] = tmp[2 * col][2 * row -
103                                         2]
104             self.Tb[2 * (cellI - 1)][2] = tmp[2 * col - 2][2 *
105                                         row]
106             self.Tb[2 * (cellI - 1)][3] = tmp[2 * col - 1][2 *
107                                         row - 2]
108             self.Tb[2 * (cellI - 1)][4] = tmp[2 * col - 1][2 *
109                                         row - 1]
110             self.Tb[2 * (cellI - 1)][5] = tmp[2 * col - 2][2 *
111                                         row - 1]
112
113             self.Tb[2 * cellI - 1][0] = tmp[2 * col - 2][2 *
114                                         row]
115             self.Tb[2 * cellI - 1][1] = tmp[2 * col][2 * row -
116                                         2]
117             self.Tb[2 * cellI - 1][2] = tmp[2 * col][2 * row]
118             self.Tb[2 * cellI - 1][3] = tmp[2 * col - 1][2 *
119                                         row - 1]
120             self.Tb[2 * cellI - 1][4] = tmp[2 * col][2 * row -
121                                         1]
122             self.Tb[2 * cellI - 1][5] = tmp[2 * col - 1][2 *
123                                         row]
124
125     def boundaryGeneration(self):
126         if self.bTp == "LinearTri":

```

```
117     NE = self.NE
118     NN = self.NN
119     elif self.bTp == "QuadraticTri":
120         NE = [2 * self.NE[0], 2 * self.NE[1]]
121         NN = [ne + 1 for ne in NE]
122
123     # ---> 边界网格节点的定义 <---
124     # 将所有边界节点设置为 Dirichlet 边界类型
125     self.nodeBC[:, 0] = -1
126
127     # 针对本问题将下边界上节点设置为 Robin 类型
128     for i in range(1, NE[0]):
129         self.nodeBC[i][0] = -1
130
131     # bottom nodes
132     for i in range(NE[0]):
133         self.nodeBC[i][1] = i * NN[1]
134
135     # right nodes
136     for i in range(NE[0], NE[0] + NE[1]):
137         self.nodeBC[i][1] = NE[0] * NN[1] + i - NE[0]
138
139     # top nodes
140     for i in range(NE[0] + NE[1], 2 * NE[0] + NE[1]):
141         self.nodeBC[i][1] = (2 * NE[0] + NE[1] + 1 - i) * NN[1] -
142                         1
143
144     # left nodes
145     for i in range(2 * NE[0] + NE[1], self.nodeBC.shape[0]):
146         self.nodeBC[i][1] = 2 * (NE[0] + NE[1]) - i
147
148     # ---> 边界网格边的定义 <---
149     # 初始化所有边界类型为 Dirichlet 类型边
150     self.edgeBC[:, 0] = -1
151
152     # 针对本问题将下边界的边设置为 Neumann 类型
153     NE = self.NE
154     for i in range(NE[0]):
155         self.edgeBC[i][0] = -1
156
157     # bottom edge
158     for i in range(1, NE[0] + 1):
159         self.edgeBC[i - 1][1] = (i - 1) * 2 * NE[1]
159         self.edgeBC[i - 1][2] = (i - 1) * (NE[1] + 1)
```

```

160         self.edgeBC[i - 1][3] = i * (NE[1] + 1)
161
162     # right edge
163     for i in range(NE[0] + 1, NE[0] + NE[1] + 1):
164         self.edgeBC[i - 1][1] = (NE[0] - 1) * 2 * NE[1] + 2 * (i -
165                                         NE[0]) - 1
166         self.edgeBC[i - 1][2] = NE[0] * (NE[1] + 1) + i - NE[0] -
167                                         1
168         self.edgeBC[i - 1][3] = NE[0] * (NE[1] + 1) + i - NE[0]
169
170     # top edge
171     for i in range(NE[0] + NE[1] + 1, 2 * NE[0] + NE[1] + 1):
172         self.edgeBC[i - 1][1] = (2 * NE[0] + NE[1] + 1 - i) * 2 *
173                                         NE[1] - 1
174         self.edgeBC[i - 1][2] = (2 * NE[0] + NE[1] + 2 - i) * (NE[
175                                         1] + 1) - 1
176         self.edgeBC[i - 1][3] = (2 * NE[0] + NE[1] + 1 - i) * (NE[
177                                         1] + 1) - 1
178
179     # left edge
180     for i in range(2 * NE[0] + NE[1] + 1, self.edgeBC.shape[0] + 1
181                     ):
182         self.edgeBC[i - 1][1] = 2 * (2 * NE[0] + 2 * NE[1] + 1 - i
183                                     ) - 2
184         self.edgeBC[i - 1][2] = 2 * NE[0] + 2 * NE[1] + 1 - i
185         self.edgeBC[i - 1][3] = 2 * NE[0] + 2 * NE[1] - i

```

B.5.3 linearSystem.py

```

1 import numpy as np
2 from math import *
3 import functions as fun
4 import time
5 import postProcess
6 np.set_printoptions(precision=6, suppress=True)
7
8 class LinearSystem:
9     def __init__(self, mesh, gaussPointNum, setInfo) -> None:
10         self.mesh = mesh
11         self.simSet = setInfo
12         self.gPtN = gaussPointNum
13         self.gPtN1D = 4

```

```
14     self.dt = (setInfo["simTime"][1] - setInfo["simTime"][0]) /  
15         setInfo["Nt"]  
16  
16     self.gaussWeightRef = np.zeros(gaussPointNum)  
17     self.gaussPointRef = np.zeros((gaussPointNum, 2))  
18     self.gaussWeightRef1D = np.zeros(self.gPtN1D)  
19     self.gaussPointRef1D = np.zeros(self.gPtN1D)  
20  
21     self.gaussWeightLocal = np.zeros(gaussPointNum)  
22     self.gaussPointLocal = np.zeros((gaussPointNum, 2))  
23     self.gaussWeightLocal1D = np.zeros(self.gPtN1D)  
24     self.gaussPointLocal1D = np.zeros(self.gPtN1D)  
25  
26     self.Jacobian = np.zeros((2, 2))  
27     self.JacobianInv = np.zeros((2, 2))  
28     self.Jdet = 0  
29  
30     self.mtxA = np.zeros((mesh.Pb.shape[0], mesh.Pb.shape[0]))  
31     self.massMtx = np.zeros((mesh.Pb.shape[0], mesh.Pb.shape[0]))  
32     self.mtxAFinal = np.zeros((mesh.Pb.shape[0], mesh.Pb.shape[0])  
33         )  
33     self.vecB = np.zeros(mesh.Pb.shape[0])  
34     self.res = np.zeros(mesh.Pb.shape[0])  
35  
36     self.maxError = 0  
37     self.Linfinity = 0  
38     self.L2 = 0  
39     self.H1 = 0  
40  
41     if mesh.bTp == "LinearTri":  
42         self.baseNum = 3  
43     elif mesh.bTp == "LinearQua":  
44         self.baseNum = 4  
45     elif mesh.bTp == "QuadraticTri":  
46         self.baseNum = 6  
47     elif mesh.bTp == "QuadraticQua":  
48         self.baseNum = 8  
49  
50     self.gaussWeightAndPointRef()  
51     self.gaussWeightAndPointRef1D()  
52  
53 def gaussWeightAndPointRef(self):  
54     if self.gPtN == 3:  
55         self.gaussWeightRef[0] = 1 / 6
```

```

56     self.gaussWeightRef[1] = 1 / 6
57     self.gaussWeightRef[2] = 1 / 6
58
59     self.gaussPointRef[0][0] = 0.5
60     self.gaussPointRef[0][1] = 0
61
62     self.gaussPointRef[1][0] = 0.5
63     self.gaussPointRef[1][1] = 0.5
64
65     self.gaussPointRef[2][0] = 0
66     self.gaussPointRef[2][1] = 0.5
67 elif self.gPtN == 4:
68     self.gaussWeightRef[0] = (1 - 1 / sqrt(3)) / 8
69     self.gaussWeightRef[1] = (1 - 1 / sqrt(3)) / 8
70     self.gaussWeightRef[2] = (1 + 1 / sqrt(3)) / 8
71     self.gaussWeightRef[3] = (1 + 1 / sqrt(3.0)) / 8
72
73     self.gaussPointRef[0][0] = (1 / sqrt(3) + 1) / 2
74     self.gaussPointRef[0][1] =
75         1 - 1 / sqrt(3) * (1 + 1 / sqrt(3)) / 4
76
77     self.gaussPointRef[1][0] = (1 / sqrt(3) + 1) / 2
78     self.gaussPointRef[1][1] =
79         1 - 1 / sqrt(3) * (1 - 1 / sqrt(3)) / 4
80
81     self.gaussPointRef[2][0] = (-1 / sqrt(3) + 1) / 2
82     self.gaussPointRef[2][1] =
83         1 + 1 / sqrt(3) * (1 + 1 / sqrt(3)) / 4
84
85     self.gaussPointRef[3][0] = (-1 / sqrt(3) + 1) / 2
86     self.gaussPointRef[3][1] =
87         1 + 1 / sqrt(3) * (1 - 1 / sqrt(3)) / 4
88 elif self.gPtN == 9:
89     self.gaussWeightRef[0] = 64 / 81 * 1 / 8
90     self.gaussWeightRef[1] = 100 / 324 * (1 - sqrt(3 / 5)) / 8
91     self.gaussWeightRef[2] = 100 / 324 * (1 - sqrt(3 / 5)) / 8
92     self.gaussWeightRef[3] = 100 / 324 * (1 + sqrt(3 / 5)) / 8
93     self.gaussWeightRef[4] = 100 / 324 * (1 + sqrt(3 / 5)) / 8
94     self.gaussWeightRef[5] = 40 / 81 * 1 / 8
95     self.gaussWeightRef[6] = 40 / 81 * 1 / 8
96     self.gaussWeightRef[7] = 40 / 81 * (1 - sqrt(3 / 5)) / 8
97     self.gaussWeightRef[8] = 40 / 81 * (1 + sqrt(3 / 5)) / 8
98
99     self.gaussPointRef[0][0] = 1 / 2

```

```
100     self.gaussPointRef[0][1] = 1 / 4
101
102     self.gaussPointRef[1][0] = (1.0 + sqrt(3.0 / 5.0)) / 2.0
103     self.gaussPointRef[1][1] = (
104         1.0 - sqrt(3.0 / 5.0)) * (1.0 + sqrt(3.0 / 5.0)) / 4.0
105
106     self.gaussPointRef[2][0] = (1.0 + sqrt(3.0 / 5.0)) / 2.0
107     self.gaussPointRef[2][1] = (
108         1.0 - sqrt(3.0 / 5.0)) * (1.0 - sqrt(3.0 / 5.0)) / 4.0
109
110     self.gaussPointRef[3][0] = (1.0 - sqrt(3.0 / 5.0)) / 2.0
111     self.gaussPointRef[3][1] = (
112         1.0 + sqrt(3.0 / 5.0)) * (1.0 + sqrt(3.0 / 5.0)) / 4.0
113
114     self.gaussPointRef[4][0] = (1.0 - sqrt(3.0 / 5.0)) / 2.0
115     self.gaussPointRef[4][1] = (
116         1.0 + sqrt(3.0 / 5.0)) * (1.0 - sqrt(3.0 / 5.0)) / 4.0
117
118     self.gaussPointRef[5][0] = (1.0 + 0.0) / 2.0
119     self.gaussPointRef[5][1] = (
120         1.0 - 0.0) * (1.0 + sqrt(3.0 / 5.0)) / 4.0
121
122     self.gaussPointRef[6][0] = (1.0 + 0.0) / 2.0
123     self.gaussPointRef[6][1] = (
124         1.0 - 0.0) * (1.0 - sqrt(3.0 / 5.0)) / 4.0
125
126     self.gaussPointRef[7][0] = (1.0 + sqrt(3.0 / 5.0)) / 2.0
127     self.gaussPointRef[7][1] = (
128         1.0 - sqrt(3.0 / 5.0)) * (1.0 + 0.0) / 4.0
129
130     self.gaussPointRef[8][0] = (1.0 - sqrt(3.0 / 5.0)) / 2.0
131     self.gaussPointRef[8][1] = (
132         1.0 + sqrt(3.0 / 5.0)) * (1.0 + 0.0) / 4.0
133
134 def gaussWeightAndPointRef1D(self):
135     if self.gPtN1D == 2:
136         self.gaussWeightRef1D[0] = 1.0
137         self.gaussWeightRef1D[1] = 1.0
138
139         self.gaussPointRef1D[0] = -1.0 / sqrt(3)
140         self.gaussPointRef1D[1] = +1.0 / sqrt(3)
141     elif self.gPtN1D == 4:
142         self.gaussWeightRef1D[0] = 0.3478548451
143         self.gaussWeightRef1D[1] = 0.3478548451
```

```

144     self.gaussWeightRef1D[2] = 0.6521451549
145     self.gaussWeightRef1D[3] = 0.6521451549
146
147     self.gaussPointRef1D[0] = +0.8611363116
148     self.gaussPointRef1D[1] = -0.8611363116
149     self.gaussPointRef1D[2] = +0.3399810436
150     self.gaussPointRef1D[3] = -0.3399810436
151     elif self.gPtN1D == 8:
152         self.gaussWeightRef1D[0] = 0.1012285363
153         self.gaussWeightRef1D[1] = 0.1012285363
154         self.gaussWeightRef1D[2] = 0.2223810345
155         self.gaussWeightRef1D[3] = 0.2223810345
156         self.gaussWeightRef1D[4] = 0.3137066459
157         self.gaussWeightRef1D[5] = 0.3137066459
158         self.gaussWeightRef1D[6] = 0.3626837834
159         self.gaussWeightRef1D[7] = 0.3626837834
160
161         self.gaussPointRef1D[0] = +0.9602898565
162         self.gaussPointRef1D[1] = -0.9602898565
163         self.gaussPointRef1D[2] = +0.7966664774
164         self.gaussPointRef1D[3] = -0.7966664774
165         self.gaussPointRef1D[4] = +0.5255324099
166         self.gaussPointRef1D[5] = -0.5255324099
167         self.gaussPointRef1D[6] = +0.1834346425
168         self.gaussPointRef1D[7] = -0.1834346425
169
170     def gaussWeightAndPointLocal(self, ver):
171         x1, y1 = ver[0][0], ver[0][1]
172         x2, y2 = ver[1][0], ver[1][1]
173         x3, y3 = ver[2][0], ver[2][1]
174         self.Jdet = abs((x2 - x1) * (y3 - y1) - (x3 - x1) * (y2 - y1))
175
176         for i in range(self.gPtN):
177             self.gaussWeightLocal[i] = self.Jdet * self.gaussWeightRef
178                             [i]
179             self.gaussPointLocal[i][0] = x1 + (x2 - x1) * self.
180                               gaussPointRef[i][0] + (
181                             x3 - x1) * self.gaussPointRef[i][1]
182             self.gaussPointLocal[i][1] = y1 + (y2 - y1) * self.
183                               gaussPointRef[i][0] + (
184                             y3 - y1) * self.gaussPointRef[i][1]
185
186     def gaussWeightAndPointLocal1D(self, lower, upper):

```

```
184     self.gaussWeightLocal1D = (upper - lower) * self.  
185                                         gaussWeightRef1D / 2  
186     self.gaussPointLocal1D = (upper - lower) * self.  
187                                         gaussPointRef1D / 2 + (upper  
188                                         + lower) / 2  
189  
190  
191     def refBaseFunction(self, x, dD, bIx, bTp):  
192         res = 0  
193         if bTp == "LinearTri":  
194             if dD[0] == 0 and dD[1] == 0:  
195                 if bIx == 0:  
196                     res = 1 - x[0] - x[1]  
197                 elif bIx == 1:  
198                     res = x[0]  
199                 elif bIx == 2:  
200                     res = x[1]  
201             elif dD[0] == 1 and dD[1] == 0:  
202                 if bIx == 0:  
203                     res = -1  
204                 elif bIx == 1:  
205                     res = 1  
206                 elif bIx == 2:  
207                     res = 0  
208             elif dD[0] == 0 and dD[1] == 1:  
209                 if bIx == 0:  
210                     res = -1  
211                 elif bIx == 1:  
212                     res = 0  
213                 elif bIx == 2:  
214                     res = 1  
215             else:  
216                 return 0  
217         return res  
218         elif bTp == "QuadraticTri":  
219             if dD[0] == 0 and dD[1] == 0:  
220                 if bIx == 0:  
221                     res = 2 * pow(x[0], 2) + 2 * pow(x[1], 2) + 4 * x[  
222                         0] * x[1] - 3 * x[0] - 3 * x[1] + 1  
223                 elif bIx == 1:  
224                     res = 2 * pow(x[0], 2) - x[0]  
225                 elif bIx == 2:  
226                     res = 2 * pow(x[1], 2) - x[1]
```

```
223     elif bIx == 3:  
224         res = -4 * pow(x[0], 2) - 4 * x[0] * x[1] + 4 * x[  
225                                         0]  
226     elif bIx == 4:  
227         res = 4 * x[0] * x[1]  
228     elif bIx == 5:  
229         res = -4 * pow(x[1], 2) - 4 * x[0] * x[1] + 4 * x[  
230                                         1]  
231     elif dD[0] == 1 and dD[1] == 0:  
232         if bIx == 0:  
233             res = 4 * x[0] + 4 * x[1] - 3  
234         elif bIx == 1:  
235             res = 4 * x[0] - 1  
236         elif bIx == 2:  
237             res = 0  
238         elif bIx == 3:  
239             res = -8 * x[0] - 4 * x[1] + 4  
240         elif bIx == 4:  
241             res = 4 * x[1]  
242         elif bIx == 5:  
243             res = -4 * x[1]  
244     elif dD[0] == 0 and dD[1] == 1:  
245         if bIx == 0:  
246             res = 4 * x[1] + 4 * x[0] - 3  
247         elif bIx == 1:  
248             res = 0  
249         elif bIx == 2:  
250             res = 4 * x[1] - 1  
251         elif bIx == 3:  
252             res = -4 * x[0]  
253         elif bIx == 4:  
254             res = 4 * x[0]  
255         elif bIx == 5:  
256             res = -8 * x[1] - 4 * x[0] + 4  
257     elif dD[0] == 2 and dD[1] == 0:  
258         if bIx == 0:  
259             res = 4  
260         elif bIx == 1:  
261             res = 4  
262         elif bIx == 2:  
263             res = 0  
264         elif bIx == 3:  
265             res = -8  
266         elif bIx == 4:
```

```
265             res = 0
266         elif bIx == 5:
267             res = 0
268         elif dD[0] == 0 and dD[1] == 2:
269             if bIx == 0:
270                 res = 4
271             elif bIx == 1:
272                 res = 0
273             elif bIx == 2:
274                 res = 4
275             elif bIx == 3:
276                 res = 0
277             elif bIx == 4:
278                 res = 0
279             elif bIx == 5:
280                 res = -8
281         elif dD[0] == 1 and dD[1] == 1:
282             if bIx == 0:
283                 res = 4
284             elif bIx == 1:
285                 res = 0
286             elif bIx == 2:
287                 res = 0
288             elif bIx == 3:
289                 res = -4
290             elif bIx == 4:
291                 res = 4
292             elif bIx == 5:
293                 res = -4
294         return res
295     elif bTp == "LinearQua":
296         if dD[0] == 0 and dD[1] == 0:
297             if bIx == 0:
298                 res = 0.25 * (1 - x[0]) * (1 - x[1])
299             elif bIx == 1:
300                 res = 0.25 * (1 + x[0]) * (1 - x[1])
301             elif bIx == 2:
302                 res = 0.25 * (1 + x[0]) * (1 + x[1])
303             elif bIx == 3:
304                 res = 0.25 * (1 - x[0]) * (1 + x[1])
305         elif dD[0] == 1 and dD[1] == 0:
306             if bIx == 0:
307                 res = -0.25 * (1 - x[1])
308             elif bIx == 1:
```



```

360                                     gaussPointLocal[i], ver,
361                                     testdD, bIx[1], bTp[1])
362
363
364     def assembleMtxA(self, func, traldD, testdD, PbTrial, PbTest,
365                           TbTrial, TbTest,
366                           trialBasisNum, testBasisNum,
367                           bTp):
368         tmpMtxA = np.zeros((PbTest.shape[0], PbTrial.shape[0]))
369         for n in range(self.mesh.T.shape[0]):
370             ver = np.zeros((self.mesh.T.shape[1], 2))
371             for i in range(self.mesh.T.shape[1]):
372                 ver[i][0] = self.mesh.P[self.mesh.T[n][i]][0]
373                 ver[i][1] = self.mesh.P[self.mesh.T[n][i]][1]
374
375             for alpha in range(trialBasisNum):
376                 for beta in range(testBasisNum):
377                     tmp = self.gaussInteTrialAndTest(ver, [alpha, beta],
378                                         traldD, testdD, func, bTp)
379
380                     row = TbTest[n][beta]
381                     col = TbTrial[n][alpha]
382                     tmpMtxA[row][col] += tmp
383
384         return tmpMtxA
385
386
387     def assembleAllMtxA(self):
388         print("\n-->> Assembling Matrix <<---")
389         timeStart = time.time()
390
391         mtxA1 = self.assembleMtxA(fun.funC, [1, 0], [1, 0], self.mesh.
392                                   Pb, self.mesh.Pb, self.mesh.
393                                   Tb, self.mesh.Tb, self.
394                                   baseNum, self.baseNum, [self.
395                                   .mesh.bTp, self.mesh.bTp])
396
397         mtxA2 = self.assembleMtxA(fun.funC, [0, 1], [0, 1], self.mesh.
398                                   Pb, self.mesh.Pb, self.mesh.
399                                   Tb, self.mesh.Tb, self.
400                                   baseNum, self.baseNum, [self.
401                                   .mesh.bTp, self.mesh.bTp])
402
403         self mtxA = mtxA1 + mtxA2
404
405

```

```
389     # Mass Matrix
390     self.massMtx = self.assembleMtxA(fun.funM, [0, 0], [0, 0],
391                                         self.mesh.Pb, self.mesh.Pb,
392                                         self.mesh.Tb, self.mesh.Tb,
393                                         self.baseNum, self.baseNum,
394                                         [self.mesh.bTp, self.mesh.
395                                         bTp])
396
397     if self.simSet["theta"] != 0:
398         self.mtxAFinal = self.massMtx / (self.simSet["theta"] *
399                                         self.dt) + self.mtxA
400     else:
401         self.mtxAFinal = self.massMtx / self.dt
402
403     timeEnd = time.time()
404     print("---->> Time = %f s <<---" % (timeEnd - timeStart))
405     if self.simSet['showMtxBeforeBC'] == 'yes':
406         print("---->> Matrix Before BC treatment <<---")
407         print("mtxA.shape = ", self.mtxA.shape)
408         print(self.mtxA)
409         print()
410         print("massMtx.shape = ", self.massMtx.shape)
411         print(self.massMtx)
412         print()
413         print("mtxAFinal.shape = ", self.mtxAFinal.shape)
414         print(self.mtxAFinal)
415
416
417     # 载荷向量组装功能函数
418     def gaussInteTest(self, ver, teBIX, func, bTp, time):
419         res = 0
420         self.gaussWeightAndPointLocal(ver)
421
422         for i in range(self.gPtN):
423             tmp = func(self.gaussPointLocal[i], time)
424             tmpBase = self.localBaseFun(self.gaussPointLocal[i], ver,
425                                         [0, 0], teBIX, bTp)
426             res += self.gaussWeightLocal[i] * tmp * tmpBase
427
428         return res
429
430
431     def assembleVectorB(self, func, time, PbTest, TbTest, testBasisNum,
432                         , bTp):
```

```

425     tmpVecB = np.zeros(PbTest.shape[0])
426     for n in range(TbTest.shape[0]):
427         ver = np.zeros((TbTest.shape[1], 2))
428         for i in range(TbTest.shape[1]):
429             ver[i][0] = PbTest[TbTest[n][i]][0]
430             ver[i][1] = PbTest[TbTest[n][i]][1]
431         for beta in range(testBasisNum):
432             tmp = self.gaussInteTest(ver, beta, func, bTp, time)
433             tmpVecB[TbTest[n][beta]] += tmp
434     return tmpVecB
435
436
437     def assembleAllVectorB(self, xOld, t):
438         vecB1 = self.assembleVectorB(fun.funF, t, self.mesh.Pb, self.
439                                     mesh.Tb, self.baseNum, self.
440                                     mesh.bTp)
441         vecB2 = self.assembleVectorB(fun.funF, t-self.dt, self.mesh.Pb
442                                     , self.mesh.Tb, self.baseNum
443                                     , self.mesh.bTp)
444         self.vecB = self.simSet["theta"] * vecB1 + (1 - self.simSet["
445                                         theta"]) * vecB2
446         if self.simSet["theta"] != 0:
447             self.vecB = self.vecB + self.massMtx.dot(xOld) / (self.
448                                         simSet["theta"] * self.dt)
449         else:
450             self.vecB = self.vecB + (self.mtxAFinal - self.mtxA).dot(
451                                         xOld)
452         if self.simSet["showVecBeforeBC"] == 'yes':
453             print("----> Vector Before BC treatment <<---")
454             print("vecB1.shape = ", vecB1.shape)
455             print(vecB1)
456             print()
457             print("vecB2.shape = ", vecB2.shape)
458             print(vecB2)
459             print()
460             print("self.vecB.shape = ", self.vecB.shape)
461             print(self.vecB)
462             print()
463
464
465     def fieldInit(self):
466         print("\n----> Field Initialization <<---")
467         timeStart = time.time()
468         u0 = np.zeros(self.mesh.Pb.shape[0])

```

```
462     for i in range(self.mesh.Pb.shape[0]):  
463         u0[i] = fun.fieldInit(self.mesh.Pb[i])  
464     timeEnd = time.time()  
465     print("--->> Time = %f s <<---" % (timeEnd - timeStart))  
466     return u0  
467  
468  
469     def boundaryTreatment(self, cTime):  
470         self.treatDirichletBC(cTime)  
471         if self.simSet["showMtxAfterBC"] == 'yes':  
472             print("--->> Matrix After BC treatment <<---")  
473             print(self mtxAFinal)  
474         if self.simSet["showVecAfterBC"] == 'yes':  
475             print("\n--->> Vector After BC treatment <<---")  
476             print(self.vecB)  
477  
478  
479     def treatDirichletBC(self, time):  
480         if self.simSet["theta"] != 0:  
481             for k in range(self.mesh.nodeBC.shape[0]):  
482                 i = self.mesh.nodeBC[k][1]  
483                 if self.mesh.nodeBC[k][0] == -1:  
484                     # print("Treat < Dirichlet > Boundary Node:  
485                     self.mtxAFinal[i, :] = 0  
486                     self.mtxAFinal[i][i] = 1  
487                     self.vecB[i] = self.simSet["theta"] * fun.funG(  
488                         self.mesh.Pb[i], time) + (1-  
489                         self.simSet["theta"]) * fun.  
490                         funG(self.mesh.Pb[i], time-  
491                         self.dt)  
492             else:  
493                 for k in range(self.mesh.nodeBC.shape[0]):  
494                     i = self.mesh.nodeBC[k][1]  
495                     if self.mesh.nodeBC[k][0] == -1:  
496                         # print("Treat < Dirichlet > Boundary Node:  
497                         self.mtxAFinal[i, :] = 0  
498                         self.mtxAFinal[i][i] = 1  
499                         self.vecB[i] = fun.funG(self.mesh.Pb[i], time)  
500  
501  
502     def parabolicSolver(self):  
503         xOld = self.fieldInit()  
504         timeStart = time.time()  
505         print("\n--->> Linear System Solving <<---\n")  
506         for i in range(1, self.simSet["Nt"]+1):
```

```

501     t = self.simSet["simTime"][0] + i * self.dt
502     print("current time step: %d, time = %f" % (i, t))
503     # 载荷向量组装
504     self.assembleAllVectorB(xOld, t)
505
506     # 边界条件处理
507     self.boundaryTreatment(t)
508
509     # 线性系统求解
510     res = np.linalg.solve(self mtxAFinal, self.vecB)
511
512     # Update the result
513     if self.simSet["theta"] != 0:
514         res = (res - xOld) / self.simSet["theta"] + xOld
515
516     xOld = res
517     self.res = res
518
519     timeEnd = time.time()
520     print("--->> Simulation Done !!! <<---")
521     print("--->> Time = %f s <<---" % (timeEnd - timeStart))
522     if self.simSet["printRes"] == 'yes':
523         print("\n--->> Result <<---")
524         print(self.res)
525
526     self.errorCalculation()
527     if self.simSet["printError"] == 'yes':
528         print("\n--->> Error Information <<---")
529         print("maxError = %.4e" % self.maxError)
530         print("Linfinity = %.4e" % self.Linfinity)
531         print("L2 = %.4e" % self.L2)
532         print("H1 = %.4e" % self.H1)
533
534     """
535     误差分析
536     """
537
538     def infinityFeRes(self, x, uhLocal, ver, dD, baseNum, bTp):
539         res = 0
540         for i in range(baseNum):
541             res += uhLocal[i] * self.localBaseFun(x, ver, dD, i, bTp)
542         return res
543
544

```

```

545     def gaussIntegralFeRes(self, uhLocal, ver, function, dD, baseNum,
546                             bTp):
547         res = 0
548         self.gaussWeightAndPointLocal(ver)
549         for i in range(self.gPtN):
550             res += self.gaussWeightLocal[i] * pow(function(self.
551                                         gaussPointLocal[i], self.
552                                         simSet["simTime"][1]) - self.
553                                         .infinityFeRes(self.
554                                         gaussPointLocal[i], uhLocal,
555                                         ver, dD, baseNum, bTp), 2)
556
557         return res
558
559
560
561
562     def maxErrorCompute(self):
563         res = abs(fun.exactSolution(self.mesh.Pb[0], self.simSet["
564                                         simTime"][1]) - self.res[0])
565         for i in range(1, self.mesh.Pb.shape[0]):
566             tmp = abs(fun.exactSolution(self.mesh.Pb[i], self.simSet["
567                                         simTime"][1]) - self.res[i])
568             if tmp > res:
569                 res = tmp
570
571         return res
572
573
574
575     def errorInfinityNorm(self, resVec, dD, func, Pb, Tb, baseNum, bTp
576                           ):
577         res = 0
578         uhLocal = [0 for i in range(baseNum)]
579         for n in range(Tb.shape[0]):
580             ver = np.zeros((Tb.shape[1], 2))
581             for i in range(Tb.shape[1]):
582                 ver[i][0] = Pb[Tb[n][i]][0]
583                 ver[i][1] = Pb[Tb[n][i]][1]
584             self.gaussWeightAndPointLocal(ver)
585             for i in range(baseNum):
586                 uhLocal[i] = resVec[Tb[n][i]]
587             tmp = 0
588             for i in range(self.gPtN):
589                 value = abs(func(self.gaussPointLocal[i], self.simSet[
590                                         "simTime"][1]) - self.
591                                         infinityFeRes(self.
592                                         gaussPointLocal[i], uhLocal,
593                                         ver, dD, baseNum, bTp))
594
595         return res

```


B.5.4 functions.py

```
1 from sympy import *
2
3 def fieldInit(x):
4     return exp(x[0] + x[1])
5
6
7 def funC(x):
8     return 2
9
10
11 def funM(x):
12     return 1
13
14
15 def funF(x, t):
16     return -3 * exp(x[0] + x[1] + t)
17
18
19 def funG(x, t):
20     return exp(x[0] + x[1] + t)
21
22
23 def exactSolution(x, t):
24     return exp(x[0] + x[1] + t)
25
26
27 def uDx(x, t):
28     return exp(x[0] + x[1] + t)
29
30
31 def uDy(x, t):
32     return exp(x[0] + x[1] + t)
```

B.5.5 postProcess.py

```
1 import matplotlib.pyplot as plt
2 import matplotlib.tri as mtri
3 import functions as fun
4 import numpy as np
```

```

5   fontStyle = {'family':'DejaVu Sans', 'style':'italic', 'weight':'
6     'normal', 'size':25}
7
8   def PlotResNoMesh(mesh, system):
9     fig = plt.figure(figsize=(10,5))
10    ax = fig.add_subplot(111)
11    plt.xlabel('x', fontproperties = fontStyle)
12    plt.ylabel('y', fontproperties = fontStyle)
13    plt.xticks(fontproperties=fontStyle)
14    plt.yticks(fontproperties=fontStyle)
15    plt.tick_params("both", which='major', length=5, width=1.5, colors
16                  ='k', direction='in', pad=12)
17    fig.patch.set_facecolor("white")
18    tri = mtri.Triangulation(mesh.P[:, 0], mesh.P[:, 1], mesh.T)
19    if mesh.bTp == 'LinearTri':
20      plot = plt.tricontourf(tri, system.res, cmap= 'jet')
21    else:
22      x = [mesh.x0[0] + i * mesh.dh[0] / 2 for i in range(2 * mesh.
23                                              NE[0] + 1)]
24      y = [mesh.x0[1] + j * mesh.dh[1] / 2 for j in range(2 * mesh.
25                                              NE[1] + 1)]
26      res = []
27      for j in range(2 * mesh.NE[1] + 1):
28        tmp = []
29        for i in range(2 * mesh.NE[0] + 1):
30          tmp.append(system.res[j + i * (2 * mesh.NE[1] + 1)])
31        res.append(tmp)
32      x, y = np.meshgrid(x, y)
33      plot = plt.contourf(x, y, res,12, cmap='jet')
34      xTicks=np.linspace(mesh.Pb[0][0],mesh.Pb[-1][0],5)
35      yTicks=np.linspace(mesh.Pb[0][1],mesh.Pb[-1][1],5)
36      plt.xticks(xTicks)
37      plt.yticks(yTicks)
38      colorBar = fig.colorbar(plot)
39      colorBar.ax.set_title("u", fontproperties=fontStyle)
40      colorBar.ax.tick_params(labelsize=fontStyle['size'])
41      plt.autoscale(enable=True, axis='both', tight=True)
42      plt.tight_layout()
43      plt.show()

44
45
46   def PlotAnalyticalRes(simSet):
47     nCells = [100, 50]

```

```
44     x=np.linspace(simSet["simDomain"][0][0], simSet["simDomain"][1][0]
45                           , nCells[0])
46     y=np.linspace(simSet["simDomain"][0][1], simSet["simDomain"][1][1]
47                           , nCells[1])
48     res = []
49     for j in y:
50         tmp = []
51         for i in x:
52             value = fun.func(simSet["exact"], [i, j], simSet["simTime"][
53                                         1])
54             tmp.append(value)
55         res.append(tmp)
56
57     x,y=np.meshgrid(x,y)
58     fig, ax = plt.subplots(figsize=(10,5))
59     ax.tick_params("both", which='major', length=8, width=1.5, colors=
60                           'k', direction='in', pad=12)
61     plot = plt.contourf(x,y,res,12,cmap='jet')
62     ax.set_xlabel('x', fontproperties=fontStyle)
63     ax.set_ylabel('y', fontproperties=fontStyle)
64     xTicks=np.linspace(simSet["simDomain"][0][0], simSet["simDomain"][
65                           1][0],5)
66     yTicks=np.linspace(simSet["simDomain"][0][1], simSet["simDomain"][
67                           1][1],5)
68     plt.xlim([min(xTicks), max(xTicks)])
69     plt.ylim([min(yTicks), max(yTicks)])
70     plt.xticks(xTicks)
71     plt.yticks(yTicks)
72     plt.setp(ax.get_xticklabels(), fontproperties=fontStyle)
73     plt.setp(ax.get_yticklabels(), fontproperties=fontStyle)
74     colorBar = fig.colorbar(plot)
75     colorBar.ax.set_title("uExact", fontproperties=fontStyle)
76     colorBar.ax.tick_params(direction='in', labelsize=fontStyle['size']
77                               ])
78     plt.tight_layout()
79     plt.show()
```

