有限元法程序设计及应用

南京航空航天大学 能源与动力学院

前 言

有限元法作为最为常用的数值计算方法之一，在工程结构设计中得到了广泛的应用。随着计算机技术和软件技术的发展，有限元法显示出越来越强大的生命力。经过几十年的发展，有限元法的理论日趋成熟，也涌现出了许多优秀的教材，但是涉及程序实现的教材数量较少。作为一种数值计算方法，只有与计算机程序结合在一起才更加有工程价值，因此理解并熟练掌握相关的程序设计是极为重要的。基于这个出发点，本教材在有限元基本理论的基础上，将计算机程序设计与有限元理论结合，系统阐述桁架结构、平面问题、材料非线性问题、动力学问题的有限元计算流程和编程技巧。最后，通过一个实际结构优化的算例，详细描述了结构优化设计的有限元实现方法及其在发动机涡轮锥盘设计中的应用。

本书共九个章节，第一章通过桁架结构系统介绍了有限元法的基本概念和计算流程；第二章介绍了加权余量法与变分方法，主要包含微分方程的等效积分形式、加权余量法、变分法与里兹法以及弹性力学变分原理；第三章以常应变三角形单元为基础，重点介绍了有限元法的基本原理和数值方法；第四章介绍了轴对称问题的有限元求解；第五章介绍了平面等参单元的基本概念，进而联系实际问题，叙述平面四节点等参元的编程实现；第六章详细叙述了三维等参单元与高阶单元；第七章详细叙述了非线性有限元问题，既包括弹塑性问题，又包括有限变形问题，介绍了相关 usermat 子程序的使用方法；第八章介绍了动力学问题的有限元法，包括其基本方程、方程解法、弹性杆振动的程序实现、模态分析以及叶片模态分析的程序实现；第九章叙述了结构优化的有限元实现，以遗传算法为例，介绍了优化的程序实现。本书可作为本科生有限元法的入门教材，也可以作为研究生学习有限元程序设计的参考书。

由于水平限制，本书肯定存在不足和不妥之处，热忱地希望读者和同行专家批评和指正。

版权说明

目录

第一章 绪论

1.1 引言

结构或者材料在受到外载荷作用时，会发生变形，材料内部也会产生应力。当应力达到一定数值后，结构或者材料将产生破坏或者损伤。判断结构是否发生破坏的理论称为强度理论，常用的破坏理论有：最大拉应力理论（第一强度理论），最大伸长线应变理论（第二强度理论），最大剪应力理论（第三强度理论）和形状改变比能理论（第四强度理论）。

从强度理论的表达式可以看出，判断结构是否破坏，主要基于结构的应力或者应变的大小和应力分布情况。因此应变和应力的大小及分布情况是进行结构的强度分析时的重要参数。

力学类的许多课程均讲述如何获得结构的位移、应变和应力的大小及分布情况。例如：理论力学主要讨论刚体的力平衡以及刚体之间的相互运动，材料力学主要研究杆和梁等简单结构的变形问题，弹性力学则重点研究弹性体在外载荷作用下的变形以及应力应变场的求解。然而，弹性力学也仅能给出几何结构和外载荷极为简单的一小类问题的应力应变场的解析解。为了解决复杂结构在外载荷作用下的变形和应力应变场问题，人们在工程实践中提出采用有限元法进行数值求解。

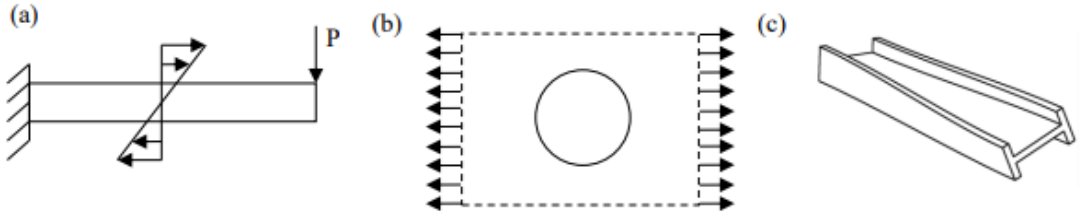


图 1.1 （a）悬臂梁的弯曲问题；（b）无限大带孔板的拉伸问题；（c）真实梁的扭转问题

对于复杂的结构，我们可以将其分割为一系列小的简单形状区域。通过简单形状区域之间的变形协调建立方程组，求解方程组即可获得每个简单区域内的应力分布。有限元法数值求解就是基于上述基本思想进行的。

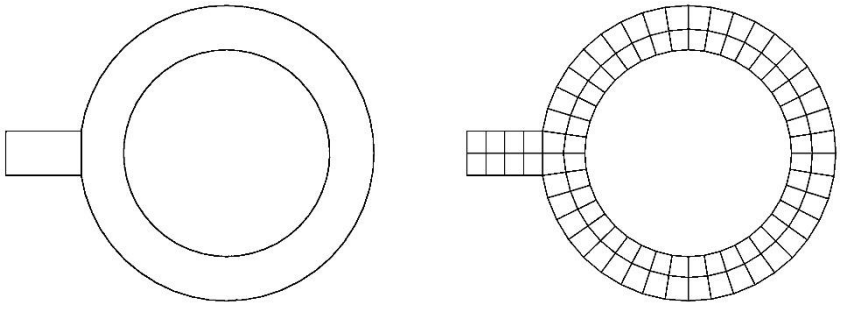


图 1.2 （a）复杂结构； （b）复杂结构离散成简单形状结构的集合

如图 1.3（a）所示，单元、节点和自由度是有限元法中三个基本要素。其中单元是分割连续体的

小区域，有线、面或实体等种类。结点是连接单元的空间点，具有一定的自由度。自由度是描述物理场响应特性的参量，随单元类型变化。

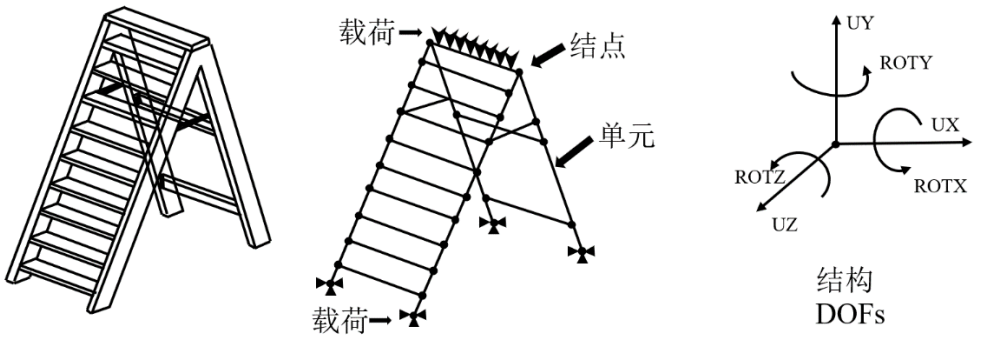


图 1.3（a） 单元、结点和自由度

需要注意的是，相同外形的单元因针对的问题不同，其自由度也可能不同。 例如三维杆单元，每个结点的未知量有三个 Ux，Uy 和 Uz，因此自由度是三。当处理由梁构成的桁架结构时，单元每个结点的未知量变成了三个位移（Ux，Uy 和 Uz）和三个转动（Rotx，Roty 和 Rotz），一共有六个自由度。类似的例子还有二维实体单元和三维四边形壳单元，三维实体结构单元和三维实体热单元等。

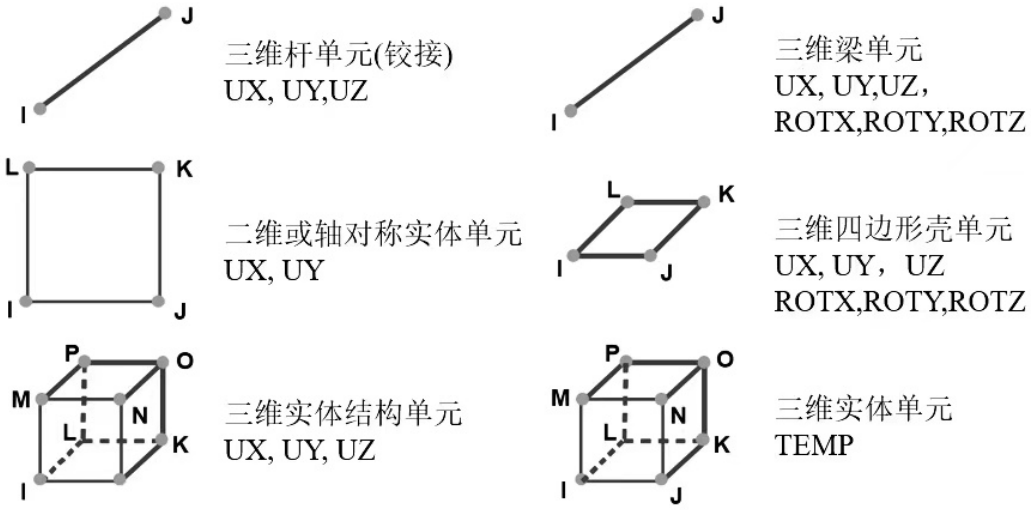


图 1.3（b） 相同外形但自由度数不同的单元

1.2 以杆单元为例介绍有限元法的计算流程

1.2.1 一般流程

本节以简单的桁架结构为例来介绍有限元的基本思想和计算过程。

图 1.4 所示为一桁架结构，左端两节点约束 x 和 y 方向位移，右端受到沿 y 轴反方向、大小为 F 的力。杆为圆形截面，截面积为 A，弹性模量为 E，刚度为 k。我们通过如下步骤进行桁架结构的变

形计算：

（1）假设杆的位移模式；

（2）列出每个杆的刚度方程；

（3）根据节点的力平衡建立总体平衡方程；

（4）求解线性方程组，获得节点位移；

（5）根据位移计算出每根杆的应力。

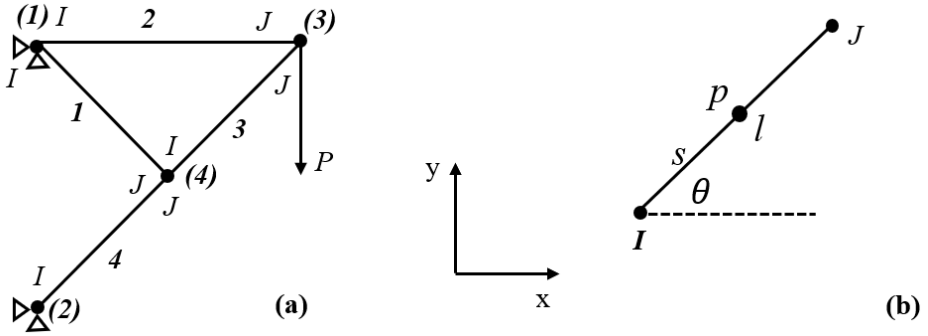


图 1.4 （a）桁架结构；（b）杆的位移函数

第一步，假设杆的位移模式

如图 1.4（b）所示，假设杆的长度为 *l*，两个端点分别用 *I* 和 *J* 来表示。杆上任意一点 *p* 的位置用该点到 I 点的距离 *s* 来表示。如果用 *u* 和 *v* 分别表示 p 点沿 x 方向和 y 方向的位移。那么

$$\begin{aligned} u &= u_I\left(1-\frac{s}{l}\right)+u_J\frac{s}{l} \\ v &= v_I\left(1-\frac{s}{l}\right)+v_J\frac{s}{l} \end{aligned} \tag{1.1}$$

其中 u_I、u_J、v_I、v_J 分别是节点 I 和 J 的 x 方向和 y 方向位移。

第二步，列出每个杆的刚度方程

由于杆只能发生沿着杆轴向的变形，所以杆上任意一点沿轴向的位移为：

$$u_a = u\cos(\theta)+v\sin(\theta) \tag{1.2}$$

轴向的应变

$$\varepsilon_a = \frac{\partial u_a}{\partial s} = \frac{\partial u}{\partial s}\cos\theta+\frac{\partial v}{\partial s}\sin\theta \tag{1.3}$$

将(1.1)代入(1.3)后得：

$$\varepsilon_a = \frac{\partial u_a}{\partial s} = \frac{(u_J-u_I)}{l}\cos\theta+\frac{(v_J-v_I)}{l}\sin\theta \tag{1.4}$$

杆内应力等于

$$\sigma_a = E\cdot\varepsilon_a = E\frac{(u_J-u_I)}{l}\cos\theta + E\frac{(v_J-v_I)}{l}\sin\theta \tag{1.5}$$

J 节点处受到的外力与坐标轴同向，符号为正，I 节点处收到的外力与坐标轴反向，符号为负：

$$F_{Ix} = -EA\frac{(u_J-u_I)}{l}\cos\theta\cos\theta - EA\frac{(v_J-v_I)}{l}\sin\theta\cos\theta \tag{1.6a}$$

$$F_{Iy} = -EA\frac{(u_J-u_I)}{l}\cos\theta\sin\theta - EA\frac{(v_J-v_I)}{l}\sin\theta\sin\theta \tag{1.6b}$$

$$F_{Jx} = EA\frac{(u_J-u_I)}{l}\cos\theta\cos\theta + EA\frac{(v_J-v_I)}{l}\sin\theta\cos\theta \tag{1.6c}$$

$$F_{Jy} = EA\frac{(u_J-u_I)}{l}\cos\theta\sin\theta + EA\frac{(v_J-v_I)}{l}\sin\theta\sin\theta \tag{1.6d}$$

从方程可以看出，方程(1.6)建立了节点位移和节点所受外力之间的关系。我们可以将式(1.6)改写为矩阵形式：

$$\begin{bmatrix} F_{Ix} \\ F_{Iy} \\ F_{Jx} \\ F_{Jy} \end{bmatrix} = \frac{EA}{l}\begin{bmatrix} \cos^2\theta & \sin\theta\cos\theta & -\cos^2\theta & -\sin\theta\cos\theta \\ \sin\theta\cos\theta & \sin^2\theta & -\cos\theta\sin\theta & -\sin^2\theta \\ -\cos^2\theta & -\sin\theta\cos\theta & \cos^2\theta & \sin\theta\cos\theta \\ -\sin\theta\cos\theta & -\sin^2\theta & \sin\theta\cos\theta & \sin^2\theta \end{bmatrix}\cdot\begin{bmatrix} u_I \\ v_I \\ u_J \\ v_J \end{bmatrix} \tag{1.7}$$

方程(1.7)也可以写成如下形式：

$$\mathbf{F}^e = \mathbf{K}^e\cdot\mathbf{u}^e \tag{1.8}$$

其中上标 e 表示“单元”的意思，每个向量和矩阵的含义如下：

$$\mathbf{F}^e = \begin{bmatrix} F_{Ix} \\ F_{Iy} \\ F_{Jx} \\ F_{Jy} \end{bmatrix},\quad \mathbf{K}^e = \frac{EA}{l}\begin{bmatrix} \cos^2\theta & \sin\theta\cos\theta & -\cos^2\theta & -\sin\theta\cos\theta \\ \sin\theta\cos\theta & \sin^2\theta & -\cos\theta\sin\theta & -\sin^2\theta \\ -\cos^2\theta & -\sin\theta\cos\theta & \cos^2\theta & \sin\theta\cos\theta \\ -\sin\theta\cos\theta & -\sin^2\theta & \sin\theta\cos\theta & \sin^2\theta \end{bmatrix},\quad \mathbf{u}^e = \begin{bmatrix} u_I \\ v_I \\ u_J \\ v_J \end{bmatrix} \tag{1.9}$$

方程(1.9)就称为杆单元的单元刚度方程，$\mathbf{K}^e$ 是单元刚度矩阵，$\mathbf{u}^e$ 是单元节点位移向量，$\mathbf{F}^e$ 是单元载荷向量。

需要注意的是，$\mathbf{K}^e$ 是不可逆矩阵。因为，如果$\mathbf{K}^e$ 可逆，根据方程(1.9)，就可以用 $\mathbf{F}^e$ 表示 $\mathbf{u}^e$。意味着任意一组节点力就可以确定一组节点位移。我们知道，由于存在刚体位移，即使这组力满足平衡条件，节点的位移也不能完全确定。因此$\mathbf{K}^e$ 是不可逆的。

为了方便表示和编写程序，$\mathbf{K}^e$ 矩阵还可以表示为如下形式：

$$\mathbf{K}^e = \begin{bmatrix} k_{11}^e & k_{12}^e & k_{13}^e & k_{14}^e \\ k_{21}^e & k_{22}^e & k_{23}^e & k_{24}^e \\ k_{31}^e & k_{32}^e & k_{33}^e & k_{34}^e \\ k_{41}^e & k_{42}^e & k_{43}^e & k_{44}^e \end{bmatrix} \tag{1.10}$$

第三步，根据节点的力平衡建立总体平衡方程

首先要介绍一下节点的两套编号。第一套编号是全局编号，即是图 1.4（a）所示的（1）（2）（3）

（4），每个数字确定唯一点。另一套编号是局部编号，需要单元编号和局部编号来确定一个节点。表 1.1 列出了每个杆单元局部节点 I、J 对应的全局编号。

表 1.1 每个杆单元局部节点 I、J 对应的全局编号

	I	J
1	（1）	（4）
2	（1）	（3）
3	（4）	（3）
4	（2）	（4）

下面针对每个节点建立力平衡方程。

节点（1）的平衡方程：

用 $\mathbf{F}_{(1)}=\begin{bmatrix}F_{(1)x} & F_{(1)y}\end{bmatrix}^T$ 节点（1）的外力。因为节点（1）x 和 y 方向分别受到约束反力 $\boldsymbol{R}_{(1)x}$ 和 $\boldsymbol{R}_{(1)y}$ 作用，因此 $\mathbf{F}_{(1)}=\begin{bmatrix}R_{(1)x} & R_{(1)y}\end{bmatrix}^T$。又因为节点（1）还受到 1 号和 2 号单元的作用，因此 $\mathbf{F}_{(1)}$ 应该等于 1 号单元对节点（1）产生的外力 $\mathbf{F}_{(1)}^1$ 和 2 号单元对节点（1） $\mathbf{F}_{(1)}^2$ 之和。

$$\mathbf{F}_{(1)}=\begin{bmatrix}R_{(1)x} & R_{(1)y}\end{bmatrix}^T=\mathbf{F}_{(1)}^1+\mathbf{F}_{(1)}^2=\mathbf{F}_I^1+\mathbf{F}_I^2 \tag{1.11}$$

将方程(1.7)代入(1.11)后得：

$$\begin{aligned}R_{(1)x}&=F_{Ix}^1+F_{Ix}^2\\&=k_{11}^1u_I^1+k_{12}^1v_I^1+k_{13}^1u_J^1+k_{14}^1v_J^1+k_{11}^2u_I^2+k_{12}^2v_I^2+k_{13}^2u_J^2+k_{14}^2v_J^2\end{aligned} \tag{1.12a}$$

$$\begin{aligned}R_{(1)y}&=F_{Iy}^1+F_{Iy}^2\\&=k_{21}^1u_I^1+k_{22}^1v_I^1+k_{23}^1u_J^1+k_{24}^1v_J^1+k_{21}^2u_I^2+k_{22}^2v_I^2+k_{23}^2u_J^2+k_{24}^2v_J^2\end{aligned} \tag{1.12b}$$

将方程(1.12)中的位移修改为总体节点编号后得：

$$\begin{aligned}R_{(1)x}&=F_{Ix}^1+F_{Ix}^2\\&=k_{11}^1u_{(1)}+k_{12}^1v_{(1)}+k_{13}^1u_{(4)}+k_{14}^1v_{(4)}+k_{11}^2u_{(1)}+k_{12}^2v_{(1)}+k_{13}^2u_{(3)}+k_{14}^2v_{(3)}\end{aligned} \tag{1.13a}$$

$$\begin{aligned}R_{(1)y}&=F_{Iy}^1+F_{Iy}^2\\&=k_{21}^1u_{(1)}+k_{22}^1v_{(1)}+k_{23}^1u_{(4)}+k_{24}^1v_{(4)}+k_{21}^2u_{(1)}+k_{22}^2v_{(1)}+k_{23}^2u_{(3)}+k_{24}^2v_{(3)}\end{aligned} \tag{1.13b}$$

合并同类项后得：

$$R_{(1)x} = F_{Ix}^1 + F_{Ix}^2$$
$$= \left(k_{11}^1 + k_{11}^2\right)u_{(1)} + \left(k_{12}^1 + k_{12}^2\right)v_{(1)} + k_{13}^2 u_{(3)} + k_{14}^2 v_{(3)} + k_{13}^1 u_{(4)} + k_{14}^1 v_{(4)} \tag{1.14a}$$

$$R_{(1)y} = F_{Iy}^1 + F_{Iy}^2$$
$$= \left(k_{21}^1 + k_{21}^2\right)u_{(1)} + \left(k_{22}^1 + k_{22}^2\right)v_{(1)} + k_{23}^2 u_{(3)} + k_{24}^2 v_{(3)} + k_{23}^1 u_{(4)} + k_{24}^1 v_{(4)} \tag{1.14b}$$

节点（2）的平衡：

$$\mathbf{F}_{(2)} = \begin{bmatrix} R_{(2)x} & R_{(2)y} \end{bmatrix}^T = \mathbf{F}_{(2)}^4 = \mathbf{F}_I^4 \tag{1.15}$$

将方程(1.7)代入(1.15)后得：

$$R_{(2)x} = F_{Ix}^4 = k_{11}^4 u_I^4 + k_{12}^4 v_I^4 + k_{13}^4 u_J^4 + k_{14}^4 v_J^4 \tag{1.16a}$$

$$R_{(2)y} = F_{Iy}^4 = k_{21}^4 u_I^4 + k_{22}^4 v_I^4 + k_{23}^4 u_J^4 + k_{24}^4 v_J^4 \tag{1.16b}$$

将方程(1.16)中的位移修改为总体节点编号后得：

$$R_{(2)x} = F_{Ix}^4 = k_{11}^4 u_{(2)} + k_{12}^4 v_{(2)} + k_{13}^4 u_{(4)} + k_{14}^4 v_{(4)} \tag{1.17a}$$

$$R_{(2)y} = F_{Iy}^4 = k_{21}^4 u_{(2)} + k_{22}^4 v_{(2)} + k_{23}^4 u_{(4)} + k_{24}^4 v_{(4)} \tag{1.17b}$$

节点（3）的平衡：

$$\mathbf{F}_{(3)} = \begin{bmatrix} 0 & -P \end{bmatrix}^T = \mathbf{F}_{(3)}^2 + \mathbf{F}_{(3)}^3 = \mathbf{F}_J^2 + \mathbf{F}_J^3 \tag{1.18}$$

将方程(1.7)代入(1.18)后得：

$$0 = F_{Jx}^2 + F_{Jx}^3$$
$$= k_{31}^2 u_I^2 + k_{32}^2 v_I^2 + k_{33}^2 u_J^2 + k_{34}^2 v_J^2 + k_{31}^3 u_I^3 + k_{32}^3 v_I^3 + k_{33}^3 u_J^3 + k_{34}^3 v_J^3 \tag{1.19a}$$

$$-P = F_{Jy}^2 + F_{Jy}^3$$
$$= k_{41}^2 u_I^2 + k_{42}^2 v_I^2 + k_{43}^2 u_J^2 + k_{44}^2 v_J^2 + k_{41}^3 u_I^3 + k_{42}^3 v_I^3 + k_{43}^3 u_J^3 + k_{44}^3 v_J^3 \tag{1.19b}$$

将方程(1.19)中的位移修改为总体节点编号后得：

$$0 = F_{Jx}^2 + F_{Jx}^3$$
$$= k_{31}^2 u_{(1)} + k_{32}^2 v_{(1)} + k_{33}^2 u_{(3)} + k_{34}^2 v_{(3)} + k_{31}^3 u_{(4)} + k_{32}^3 v_{(4)} + k_{33}^3 u_{(3)} + k_{34}^3 v_{(3)} \tag{1.20a}$$

$$-P = F_{Jy}^2 + F_{Jy}^3$$
$$= k_{41}^2 u_{(1)} + k_{42}^2 v_{(1)} + k_{43}^2 u_{(3)} + k_{44}^2 v_{(3)} + k_{41}^3 u_{(4)} + k_{42}^3 v_{(4)} + k_{43}^3 u_{(3)} + k_{44}^3 v_{(3)} \tag{1.20b}$$

合并同类项后得：

$$0 = F_{Jx}^2 + F_{Jx}^3$$
$$= k_{31}^2 u_{(1)} + k_{32}^2 v_{(1)} + \left(k_{33}^2 + k_{33}^3\right)u_{(3)} + \left(k_{34}^2 + k_{34}^3\right)v_{(3)} + k_{31}^3 u_{(4)} + k_{32}^3 v_{(4)} \tag{1.21a}$$

$$-P = F_{Jy}^2 + F_{Jy}^3$$
$$= k_{41}^2 u_{(1)} + k_{42}^2 v_{(1)} + \left(k_{43}^2 + k_{43}^3\right)u_{(3)} + \left(k_{44}^2 + k_{44}^3\right)v_{(3)} + k_{41}^3 u_{(4)} + k_{42}^3 v_{(4)} \tag{1.21b}$$

节点（4）的平衡方程：

$$\mathbf{F}_{(4)}=\begin{bmatrix}0 & 0\end{bmatrix}^T=\mathbf{F}_{(4)}^1+\mathbf{F}_{(4)}^2+\mathbf{F}_{(4)}^3=\mathbf{F}_J^1+\mathbf{F}_I^3+\mathbf{F}_J^4 \tag{1.22}$$

将方程(1.6)代入(1.20)后得：

$$\begin{aligned}0&=F_{Jx}^1+F_{Ix}^3+F_{Jx}^4\\&=k_{31}^1u_I^1+k_{32}^1v_I^1+k_{33}^1u_J^1+k_{34}^1v_J^1+k_{11}^3u_I^3+k_{12}^3v_I^3+k_{13}^3u_J^3+k_{14}^3v_J^3+k_{31}^4u_I^4+k_{32}^4v_I^4+k_{33}^4u_J^4+k_{34}^4v_J^4\end{aligned} \tag{1.23a}$$

$$\begin{aligned}0&=F_{Jy}^1+F_{Iy}^3+F_{Jy}^4\\&=k_{41}^1u_I^1+k_{42}^1v_I^1+k_{43}^1u_J^1+k_{44}^1v_J^1+k_{21}^3u_I^3+k_{22}^3v_I^3+k_{23}^3u_J^3+k_{24}^3v_J^3+k_{41}^4u_I^4+k_{42}^4v_I^4+k_{43}^4u_J^4+k_{44}^4v_J^4\end{aligned} \tag{1.23b}$$

将方程(1.21)中的位移修改为总体节点编号后得：

$$\begin{aligned}0&=F_{Jx}^1+F_{Ix}^3+F_{Jx}^4\\&=k_{31}^1u_{(1)}+k_{32}^1v_{(1)}+k_{33}^1u_{(4)}+k_{34}^1v_{(4)}+k_{11}^3u_{(4)}+k_{12}^3v_{(4)}+k_{13}^3u_{(3)}+k_{14}^3v_{(3)}+k_{31}^4u_{(2)}+k_{32}^4v_{(2)}+k_{33}^4u_{(4)}+k_{34}^4v_{(4)}\end{aligned} \tag{1.24a}$$

$$\begin{aligned}0&=F_{Jy}^1+F_{Iy}^3+F_{Jy}^4\\&=k_{41}^1u_{(1)}+k_{42}^1v_{(1)}+k_{43}^1u_{(4)}+k_{44}^1v_{(4)}+k_{21}^3u_{(4)}+k_{22}^3v_{(4)}+k_{23}^3u_{(3)}+k_{24}^3v_{(3)}+k_{41}^4u_{(2)}+k_{42}^4v_{(2)}+k_{43}^4u_{(4)}+k_{44}^4v_{(4)}\end{aligned} \tag{1.24b}$$

合并同类项后得：

$$\begin{aligned}0&=F_{Jx}^1+F_{Ix}^3+F_{Jx}^4\\&=k_{31}^1u_{(1)}+k_{32}^1v_{(1)}+k_{31}^4u_{(2)}+k_{32}^4v_{(2)}+k_{13}^3u_{(3)}+k_{14}^3v_{(3)}+\left(k_{33}^1+k_{11}^3+k_{33}^4\right)u_{(4)}+\left(k_{34}^1+k_{12}^3+k_{34}^4\right)v_{(4)}\end{aligned} \tag{1.25a}$$

$$\begin{aligned}0&=F_{Jy}^1+F_{Iy}^3+F_{Jy}^4\\&=k_{41}^1u_{(1)}+k_{42}^1v_{(1)}+k_{41}^4u_{(2)}+k_{42}^4v_{(2)}+k_{23}^3u_{(3)}+k_{24}^3v_{(3)}+\left(k_{43}^1+k_{21}^3+k_{43}^4\right)u_{(4)}+\left(k_{44}^1+k_{22}^3+k_{44}^4\right)v_{(4)}\end{aligned} \tag{1.25b}$$

这样，公式(1.14)(1.17)(1.21)和(1.25)建立了整个结构的节点位移和节点力之间的关系。可以将其写成矩阵形式：

$$\begin{bmatrix}R_{(1)x}\\R_{(1)y}\\R_{(2)x}\\R_{(2)y}\\0\\-P\\0\\0\end{bmatrix}=\begin{bmatrix}k_{11}^1+k_{11}^2 & k_{12}^1+k_{12}^2 & 0 & 0 & k_{13}^2 & k_{14}^2 & k_{13}^1 & k_{14}^1\\k_{21}^1+k_{21}^2 & k_{22}^1+k_{22}^2 & 0 & 0 & k_{23}^2 & k_{24}^2 & k_{23}^1 & k_{24}^1\\0 & 0 & k_{11}^4 & k_{12}^4 & 0 & 0 & k_{13}^4 & k_{14}^4\\0 & 0 & k_{21}^4 & k_{22}^4 & 0 & 0 & k_{23}^4 & k_{24}^4\\k_{31}^2 & k_{32}^2 & & & k_{33}^2+k_{33}^3 & k_{34}^2+k_{34}^3 & k_{31}^3 & k_{32}^3\\k_{41}^2 & k_{42}^2 & & & k_{43}^2+k_{43}^3 & k_{44}^2+k_{44}^3 & k_{41}^3 & k_{42}^3\\k_{31}^1 & k_{32}^1 & k_{31}^4 & k_{32}^4 & k_{13}^3 & k_{14}^3 & k_{33}^1+k_{11}^3+k_{33}^4 & k_{34}^1+k_{12}^3+k_{34}^4\\k_{41}^1 & k_{42}^1 & k_{41}^4 & k_{42}^4 & k_{23}^3 & k_{24}^3 & k_{43}^1+k_{21}^3+k_{43}^4 & k_{44}^1+k_{22}^3+k_{44}^4\end{bmatrix}\cdot\begin{bmatrix}u_{(1)}\\v_{(1)}\\u_{(2)}\\v_{(2)}\\u_{(3)}\\v_{(3)}\\u_{(4)}\\v_{(4)}\end{bmatrix} \tag{1.26}$$

第四步：求解线性方程组，获得节点位移

方程(1.26)也是不能直接求解，因为系数矩阵不可逆。原因和单元刚度矩阵不可逆相同，因此要求解方程(1.26)必须加入约束条件，消除系统的刚体位移后方能求解。

根据图 1.4 所示，由于存在约束，$u_{(1)}$ $v_{(1)}$ $u_{(2)}$ $v_{(2)}$ 等于零。因此方程(1.26)可以写成如下形式：

$$\begin{bmatrix} R_{(1)x} \\ R_{(1)y} \\ R_{(2)x} \\ R_{(2)y} \\ 0 \\ -P \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} k_{11}^1+k_{11}^2 & k_{12}^1+k_{12}^2 & 0 & 0 & k_{13}^2 & k_{14}^2 & k_{13}^1 & k_{14}^1 \\ k_{21}^1+k_{21}^2 & k_{22}^1+k_{22}^2 & 0 & 0 & k_{23}^2 & k_{24}^2 & k_{23}^1 & k_{24}^1 \\ 0 & 0 & k_{11}^4 & k_{12}^4 & 0 & 0 & k_{13}^4 & k_{14}^4 \\ 0 & 0 & k_{21}^4 & k_{22}^4 & 0 & 0 & k_{23}^4 & k_{24}^4 \\ k_{31}^2 & k_{32}^2 & 0 & 0 & k_{33}^2+k_{33}^3 & k_{34}^2+k_{34}^3 & k_{31}^3 & k_{32}^3 \\ k_{41}^2 & k_{42}^2 & 0 & 0 & k_{43}^2+k_{43}^3 & k_{44}^2+k_{44}^3 & k_{41}^3 & k_{42}^3 \\ k_{31}^1 & k_{32}^1 & k_{31}^4 & k_{32}^4 & k_{13}^3 & k_{14}^3 & k_{33}^1+k_{11}^3+k_{33}^4 & k_{34}^1+k_{12}^3+k_{34}^4 \\ k_{41}^1 & k_{42}^1 & k_{41}^4 & k_{42}^4 & k_{23}^3 & k_{24}^3 & k_{43}^1+k_{21}^3+k_{43}^4 & k_{44}^1+k_{22}^3+k_{44}^4 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ u_{(3)} \\ v_{(3)} \\ u_{(4)} \\ v_{(4)} \end{bmatrix} \tag{1.27}$$

上述方程组的未知量将为 4 个，理论上只需要 4 个方程即可求解。上述方程组中，由于 $R_{(1)x}$，$R_{(1)y}$，$R_{(2)x}$，$R_{(2)y}$ 未知，因此我们并不需要这几个方程参与求解。将上述方程化简为如下方程组：

$$\begin{bmatrix} 0 \\ -P \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} k_{33}^2+k_{33}^3 & k_{34}^2+k_{34}^3 & k_{31}^3 & k_{32}^3 \\ k_{43}^2+k_{43}^3 & k_{44}^2+k_{44}^3 & k_{41}^3 & k_{42}^3 \\ k_{13}^3 & k_{14}^3 & k_{33}^1+k_{11}^3+k_{33}^4 & k_{34}^1+k_{12}^3+k_{34}^4 \\ k_{23}^3 & k_{24}^3 & k_{43}^1+k_{21}^3+k_{43}^4 & k_{44}^1+k_{22}^3+k_{44}^4 \end{bmatrix} \cdot \begin{bmatrix} u_{(3)} \\ v_{(3)} \\ u_{(4)} \\ v_{(4)} \end{bmatrix} \tag{1.28}$$

对方程(1.28)求解即可获得节点位移 $u_{(3)}$ $v_{(3)}$ $u_{(4)}$ $v_{(4)}$。

第五步：根据节点位移计算出每个杆的应力

计算出节点位移 $u_{(3)}$ $v_{(3)}$ $u_{(4)}$ $v_{(4)}$ 后，由于 $u_{(1)}$，$v_{(1)}$，$u_{(2)}$，$v_{(2)}$ 已知，因此所有节点位移分量均已知。可以应用方程(1.4)计算出每根杆的轴向应变，最后根据本构方程 $\sigma = E \cdot \varepsilon$ 计算出所有杆的应力。

1.2.2 直接叠加法合成总体刚度矩阵

上述建立方程组的方法物理意义十分明确，但是不利于程序设计。如果我们将单元刚度矩阵的脚标转换成结点的总体编号，那么可以采用直接叠加法来合成总体刚度矩阵（简称“总刚”），最后施加位移约束即可进行求解。下式是一个杆单元的单元刚度矩阵。注意到每个元素 k_{ij}^e 代表单元 e 中第 j 个自由度的位移对第 i 个自由度结点载荷的作用。这实际上采用的是局部结点编号。

$$\begin{array}{c} \\ F_{Ix} \\ F_{Iy} \\ F_{Jx} \\ F_{Jy} \end{array} \begin{array}{c} \begin{array}{cccc} u_I & v_I & u_J & v_J \end{array} \\ \begin{bmatrix} k_{11}^e & k_{12}^e & k_{13}^e & k_{14}^e \\ k_{21}^e & k_{22}^e & k_{23}^e & k_{24}^e \\ k_{31}^e & k_{32}^e & k_{33}^e & k_{34}^e \\ k_{41}^e & k_{42}^e & k_{43}^e & k_{44}^e \end{bmatrix} \end{array} \tag{1.29}$$

我们将图 1.4 中所有自由度进行统一编号。按照先结点后方向的方式排序。即按照 $u_1, v_1, u_2, v_2, u_3, v_3, u_4, v_4$ 的顺序编号。第一个单元 I，J 两个结点的全局编号分别是 1 和 4。则第一个单元的四个自由度 u_I, v_I, u_J, v_J 的全局编号分别是 1，2，7，8。标注编号后的单元刚度矩阵是

$$\begin{array}{c} \\ F_1 \\ F_2 \\ F_7 \\ F_8 \end{array}\begin{array}{c} \begin{array}{cccc} u_1 & v_2 & u_7 & v_8 \end{array} \\ \begin{bmatrix} k_{11}^1 & k_{12}^1 & k_{13}^1 & k_{14}^1 \\ k_{21}^1 & k_{22}^1 & k_{23}^1 & k_{24}^1 \\ k_{31}^1 & k_{32}^1 & k_{33}^1 & k_{34}^1 \\ k_{41}^1 & k_{42}^1 & k_{43}^1 & k_{44}^1 \end{bmatrix} \end{array} \tag{1.30}$$

合成总刚时可能涉及到不同单元刚度矩阵的叠加，为了区分哪个单元的元素，我们保留表示单元编号的右上标。第二个单元的 I，J 两个结点的全局编号分别是 1 和 3。则第二个单元的四个自由度 u_I, v_I, u_J, v_J 的全局编号分别是 1，2，5，6。标注编号后的单元刚度矩阵是

$$\begin{array}{c} \\ F_1 \\ F_2 \\ F_5 \\ F_6 \end{array}\begin{array}{c} \begin{array}{cccc} u_1 & v_2 & u_5 & v_6 \end{array} \\ \begin{bmatrix} k_{11}^2 & k_{12}^2 & k_{13}^2 & k_{14}^2 \\ k_{21}^2 & k_{22}^2 & k_{23}^2 & k_{24}^2 \\ k_{31}^2 & k_{32}^2 & k_{33}^2 & k_{34}^2 \\ k_{41}^2 & k_{42}^2 & k_{43}^2 & k_{44}^2 \end{bmatrix} \end{array} \tag{1.31}$$

单元三和单元四的 I，J 两个结点的全局编号分别是 4，3 和 2，4。所以二者的单元刚度矩阵为：

$$\begin{array}{c} \\ F_7 \\ F_8 \\ F_5 \\ F_6 \end{array}\begin{array}{c} \begin{array}{cccc} u_7 & v_8 & u_5 & v_6 \end{array} \\ \begin{bmatrix} k_{11}^3 & k_{12}^3 & k_{13}^3 & k_{14}^3 \\ k_{21}^3 & k_{22}^3 & k_{23}^3 & k_{24}^3 \\ k_{31}^3 & k_{32}^3 & k_{33}^3 & k_{34}^3 \\ k_{41}^3 & k_{42}^3 & k_{43}^3 & k_{44}^3 \end{bmatrix} \end{array} \text{ 和 } \begin{array}{c} \\ F_3 \\ F_4 \\ F_7 \\ F_8 \end{array}\begin{array}{c} \begin{array}{cccc} u_3 & v_4 & u_7 & v_8 \end{array} \\ \begin{bmatrix} k_{11}^4 & k_{12}^4 & k_{13}^4 & k_{14}^4 \\ k_{21}^4 & k_{22}^4 & k_{23}^4 & k_{24}^4 \\ k_{31}^4 & k_{32}^4 & k_{33}^4 & k_{34}^4 \\ k_{41}^4 & k_{42}^4 & k_{43}^4 & k_{44}^4 \end{bmatrix} \end{array} \tag{1.32}$$

总体刚度矩阵元素 k_{ij} 表示第 j 个自由度位移对第 i 个自由度结点力总的作用。如果 j 对 i 的作用不只通过一个单元实现，则需要将所有相关单元对应元素叠加起来。下面采用叠加法构造总体刚度矩阵（本文简称总刚）。首先对总刚清零：

$$\begin{array}{c} \\ F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \\ F_8 \end{array}\begin{array}{c} \begin{array}{cccccccc} u_1 & u_2 & u_3 & u_4 & u_5 & u_6 & u_7 & u_8 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \tag{1.33}$$

然后遍历每个单元的单元刚度矩阵，将每一个元素根据其脚标填到总刚对应的位置。单元 e 的单元刚度矩阵元素 k_{ij}^e 填到总刚的第 i 行第 j 列。例如单元 1 的元素填到总刚后得：

$$
\begin{array}{c}
\begin{array}{cccc} u_1 & v_2 & u_7 & v_8 \end{array} \\
\begin{array}{c} F_1 \\ F_2 \\ F_7 \\ F_8 \end{array}
\begin{bmatrix} k_{11}^1 & k_{12}^1 & k_{13}^1 & k_{14}^1 \\ k_{21}^1 & k_{22}^1 & k_{23}^1 & k_{24}^1 \\ k_{31}^1 & k_{32}^1 & k_{33}^1 & k_{34}^1 \\ k_{41}^1 & k_{42}^1 & k_{43}^1 & k_{44}^1 \end{bmatrix}
\end{array}
\Rightarrow
\begin{array}{c}
\begin{array}{cccccccc} u_1 & u_2 & u_3 & u_4 & u_5 & u_6 & u_7 & u_8 \end{array} \\
\begin{array}{c} F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \\ F_8 \end{array}
\begin{bmatrix} k_{11}^1 & k_{12}^1 & 0 & 0 & 0 & 0 & k_{13}^1 & k_{14}^1 \\ k_{21}^1 & k_{22}^1 & 0 & 0 & 0 & 0 & k_{23}^1 & k_{24}^1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ k_{31}^1 & k_{32}^1 & 0 & 0 & 0 & 0 & k_{33}^1 & k_{34}^1 \\ k_{41}^1 & k_{42}^1 & 0 & 0 & 0 & 0 & k_{43}^1 & k_{44}^1 \end{bmatrix}
\end{array}
\tag{1.34}
$$

叠加单元 2 的元素后得：

$$
\begin{array}{c}
\begin{array}{cccc} u_1 & v_2 & u_5 & v_6 \end{array} \\
\begin{array}{c} F_1 \\ F_2 \\ F_5 \\ F_6 \end{array}
\begin{bmatrix} k_{11}^2 & k_{12}^2 & k_{13}^2 & k_{14}^2 \\ k_{21}^2 & k_{22}^2 & k_{23}^2 & k_{24}^2 \\ k_{31}^2 & k_{32}^2 & k_{33}^2 & k_{34}^2 \\ k_{41}^2 & k_{42}^2 & k_{43}^2 & k_{44}^2 \end{bmatrix}
\end{array}
\Rightarrow
\begin{array}{c}
\begin{array}{cccccccc} u_1 & u_2 & u_3 & u_4 & u_5 & u_6 & u_7 & u_8 \end{array} \\
\begin{array}{c} F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \\ F_8 \end{array}
\begin{bmatrix} k_{11}^1+k_{11}^2 & k_{12}^1+k_{12}^2 & 0 & 0 & k_{13}^2 & k_{14}^2 & k_{13}^1 & k_{14}^1 \\ k_{21}^1+k_{21}^2 & k_{22}^1+k_{22}^2 & 0 & 0 & k_{23}^2 & k_{24}^2 & k_{23}^1 & k_{24}^1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ k_{31}^2 & k_{32}^2 & 0 & 0 & k_{33}^2 & k_{34}^2 & 0 & 0 \\ k_{41}^2 & k_{42}^2 & 0 & 0 & k_{43}^2 & k_{44}^2 & 0 & 0 \\ k_{31}^1 & k_{32}^1 & 0 & 0 & 0 & 0 & k_{33}^1 & k_{34}^1 \\ k_{41}^1 & k_{42}^1 & 0 & 0 & 0 & 0 & k_{43}^1 & k_{44}^1 \end{bmatrix}
\end{array}
\tag{1.35}
$$

叠加单元 3 的元素后得：

$$
\begin{array}{c}
\begin{array}{cccc} u_7 & v_8 & u_5 & v_6 \end{array} \\
\begin{array}{c} F_7 \\ F_8 \\ F_5 \\ F_6 \end{array}
\begin{bmatrix} k_{11}^3 & k_{12}^3 & k_{13}^3 & k_{14}^3 \\ k_{21}^3 & k_{22}^3 & k_{23}^3 & k_{24}^3 \\ k_{31}^3 & k_{32}^3 & k_{33}^3 & k_{34}^3 \\ k_{41}^3 & k_{42}^3 & k_{43}^3 & k_{44}^3 \end{bmatrix}
\end{array}
\Rightarrow
$$

$$
\begin{array}{c}
\begin{array}{cccccccc} u_1 & u_2 & u_3 & u_4 & u_5 & u_6 & u_7 & u_8 \end{array} \\
\begin{array}{c} F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \\ F_8 \end{array}
\begin{bmatrix} k_{11}^1+k_{11}^2 & k_{12}^1+k_{12}^2 & 0 & 0 & k_{13}^2 & k_{14}^2 & k_{13}^1 & k_{14}^1 \\ k_{21}^1+k_{21}^2 & k_{22}^1+k_{22}^2 & 0 & 0 & k_{23}^2 & k_{24}^2 & k_{23}^1 & k_{24}^1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ k_{31}^2 & k_{32}^2 & 0 & 0 & k_{33}^2+k_{33}^3 & k_{34}^2+k_{34}^3 & k_{31}^3 & k_{32}^3 \\ k_{41}^2 & k_{42}^2 & 0 & 0 & k_{43}^2+k_{43}^3 & k_{44}^2+k_{44}^3 & k_{41}^3 & k_{42}^3 \\ k_{31}^1 & k_{32}^1 & 0 & 0 & k_{13}^3 & k_{14}^3 & k_{33}^1+k_{11}^3 & k_{34}^1+k_{12}^3 \\ k_{41}^1 & k_{42}^1 & 0 & 0 & k_{23}^3 & k_{24}^3 & k_{43}^1+k_{21}^3 & k_{44}^1+k_{22}^3 \end{bmatrix}
\end{array}
\tag{1.36}
$$

叠加单元 4 的元素后得：

$$
\begin{array}{c} \\ F_3 \\ F_4 \\ F_7 \\ F_8 \end{array}
\begin{array}{c}
\begin{array}{cccc} u_3 & v_4 & u_7 & v_8 \end{array} \\
\begin{bmatrix} k_{11}^4 & k_{12}^4 & k_{13}^4 & k_{14}^4 \\ k_{21}^4 & k_{22}^4 & k_{23}^4 & k_{24}^4 \\ k_{31}^4 & k_{32}^4 & k_{33}^4 & k_{34}^4 \\ k_{41}^4 & k_{42}^4 & k_{43}^4 & k_{44}^4 \end{bmatrix}
\end{array} \Rightarrow
$$

$$
\begin{array}{c}
\begin{array}{cccccccc} u_1 & u_2 & u_3 & u_4 & u_5 & u_6 & u_7 & u_8 \end{array} \\
\begin{bmatrix}
k_{11}^1+k_{11}^2 & k_{12}^1+k_{12}^2 & 0 & 0 & k_{13}^2 & k_{14}^2 & k_{13}^1 & k_{14}^1 \\
k_{21}^1+k_{21}^2 & k_{22}^1+k_{22}^2 & 0 & 0 & k_{23}^2 & k_{24}^2 & k_{23}^1 & k_{24}^1 \\
0 & 0 & k_{11}^4 & k_{12}^4 & 0 & 0 & k_{13}^4 & k_{14}^4 \\
0 & 0 & k_{21}^4 & k_{22}^4 & 0 & 0 & k_{23}^4 & k_{24}^4 \\
k_{31}^2 & k_{32}^2 & 0 & 0 & k_{33}^2+k_{33}^3 & k_{34}^2+k_{34}^3 & k_{31}^3 & k_{32}^3 \\
k_{41}^2 & k_{42}^2 & 0 & 0 & k_{43}^2+k_{43}^3 & k_{44}^2+k_{44}^3 & k_{41}^3 & k_{42}^3 \\
k_{31}^1 & k_{32}^1 & k_{31}^4 & k_{32}^4 & k_{13}^3 & k_{14}^3 & k_{33}^1+k_{11}^3+k_{33}^4 & k_{34}^1+k_{12}^3+k_{34}^4 \\
k_{41}^1 & k_{42}^1 & k_{41}^4 & k_{42}^4 & k_{23}^3 & k_{24}^3 & k_{43}^1+k_{21}^3+k_{43}^4 & k_{44}^1+k_{22}^3++k_{44}^4
\end{bmatrix}
\end{array}
\begin{array}{c} \\ F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \\ F_8 \end{array}
$$

(1.37)

将上式与(1.26)对比发现二者相同，说明计算结果是正确的。

1.3 杆单元有限元法的 C 语言实现

1.3.1 数据准备

有限元法计算结构的位移、变形和应力，需要涉及大量的数据。其中包括：节点总数、节点坐标、单元总数、单元节点的总体编号、材料类型总数、每种材料的弹性模量 E 和泊松比 μ，节点位移约束总数、节点位移约束、节点载荷总数、节点载荷。

假设图 1.4（a）所示的 1、3、4 杆单元长度均为 1m，2 号杆单元长度是 1.414m，杆半径均为 0.03m，则截面积 A 为 0.00283m^2。弹性模量 E=200GPa。

我们以国际单位制（N，m，s，kg）将上述所有数据写入一个 txt 文件中。该文件命名为 Data.txt。文件具体内容如下：

材料总数

1

材料弹性模量泊松比以及杆的面积

200e9　　0.3　0.00283

节点总数

4

节点坐标（这里以（1）点为坐标原点，假设 1 号杆的长度为 1）

0 0

0 -1.414

1.414 0

0.7071 -0.7071

单元总数

4

单元节点的总体编号

1 1 4

1 1 3

1 4 3

1 2 4

位移约束总数

4

具体的位移约束（第一个数字表示节点编号，后面 x，y 表示约束方向，最后一个数字表示位移约束大小）

1 x 0

1 y 0

2 x 0

2 y 0

载荷总数

1

具体载荷（第一个数字表示节点编号，后面 x，y 表示载荷方向，最后一个数字表示载荷大小）

3 y -1e5

1.3.2 数据读取和输出验证

首先定义四个结构体用来存储所有的数据。其中Data_bound结构体用于存储位移和力边界条件，Data_Link 结构体用于存储所有的数据，Data_Link_UID 结构体用于存储节点位移的编号，Data_Link_KgFg 结构体用于存储总体刚度矩阵和总体载荷向量。

```
struct Data_bound
{
    int id;//节点 id，从 0 开始
    char dir;//方向，‘x',’ y'
    double v;//具体数值
};
struct Data_Link
{
    int Nn, Ne, Nm, Nb, Nf;//分别是节点总数、单元总数、材料总数、位移边界条件总数、载荷条件总数
    double *pMat, *pNodxy;//分别存储材料常数，节点坐标，边界条件，载荷条件
    int *pEle;//存储单元节点编号
    Data_bound *p_displs,*p_force;//位移边界条件，力边界条件
} m_data_link;//这是一个全局变量
struct Data_Link_UID
{
    int ID;//在约化位移向量里的位置
    double v;//已知位移的具体数值，未知位移的具体数值。
};
struct Data_Link_KgFg
{
    double *pKg;//存储总体刚度矩阵
    double *pFg;//存储总体载荷向量
    Data_Link_UID *p_uid;//存储的是一个向量
    int Nu;//约化后的未知位移总数
};
```

读取数据的函数如下：

```
int FEM_Link_Read(char *filename, Data_Link *p)
{
    int Nn, Ne, Nm, Nb, Nf;//分别是节点总数、单元总数、材料总数、位移边界条件总数、载荷
```

条件总数

```
double *pMat, *pNodxy;//分别存储材料常数，节点坐标，边界条件，载荷条件
int *pEle;//存储单元节点编号
int i;//用于循环
Data_bound *p_displs,*p_force;//位移边界条件，力边界条件
FILE *pf=NULL;//指向文件的指针
char temp[200];//用于临时存储数据
////打开数据文件
pf=fopen(filename,"r");
if(pf==NULL)
{
    printf("Read Data file error!\n");
    return 0;
}
fscanf(pf,"%s",temp);    fscanf(pf,"%d",&Nm);
pMat=(double *)malloc(3*Nm*sizeof(double));
if(pMat==NULL)
{
    printf("pMat malloc error!\n");
    return 0;
}
fscanf(pf,"%s",temp);
for(i=0;i<Nm;i++)
{
    fscanf(pf,"%lf %lf %lf",&pMat[3*i],&pMat[3*i+1],&pMat[3*i+2]);
}
//节点
fscanf(pf,"%s",temp);    fscanf(pf,"%d",&Nn);
pNodxy=(double *)malloc(2*Nn*sizeof(double));
if(pNodxy==NULL)
```

```
{
    printf("pNodxy malloc error!\n");
    return 0;
}
fscanf(pf,"%s",temp);
for(i=0;i<Nn;i++)
{
    fscanf(pf,"%lf %lf",&pNodxy[2*i],&pNodxy[2*i+1]);
}
///////单元
fscanf(pf,"%s",temp);    fscanf(pf,"%d",&Ne);
pEle=(int *)malloc(3*Ne*sizeof(int));
if(pEle==NULL)
{
    printf("pEle malloc error!\n");
    return 0;
}
fscanf(pf,"%s",temp);
for(i=0;i<Ne;i++)
{
    fscanf(pf,"%d %d %d",&pEle[3*i],&pEle[3*i+1],&pEle[3*i+2]);
    pEle[3*i]--;//存储材料类型编号，从 0 开始
    pEle[3*i+1]--;//保证节点编号从 0 开始，有利于后面编程
    pEle[3*i+2]--;//保证节点编号从 0 开始，有利于后面编程
}
///////位移边界条件
fscanf(pf,"%s",temp);    fscanf(pf,"%d",&Nb);
p_displs=(Data_bound *)malloc(Nb*sizeof(Data_bound));
if(p_displs==NULL)
{
```

```
        printf("p_displs malloc error!\n");
        return 0;
    }
    fscanf(pf,"%s",temp);
    for(i=0;i<Nb;i++)
    {
        fscanf(pf,"%d",&p_displs[i].id);        p_displs[i].id--;
        fscanf(pf,"%s",&p_displs[i].dir);       fscanf(pf,"%lf",&p_displs[i].v);
    }
    ///////力边界条件
    fscanf(pf,"%s",temp);   fscanf(pf,"%d",&Nf);
    p_force=(Data_bound *)malloc(Nf*sizeof(Data_bound));
    if(p_force==NULL)
    {
        printf("p_force malloc error!\n");
        return 0;
    }
    fscanf(pf,"%s",temp);
    for(i=0;i<Nf;i++)
    {
        fscanf(pf,"%d",&p_force[i].id);         p_force[i].id--;
        fscanf(pf,"%s",&p_force[i].dir);        fscanf(pf,"%lf",&p_force[i].v);
    }
    //////赋值
    p->Nb=Nb;   p->Ne=Ne;   p->Nf=Nf;   p->Nm=Nm;  p->Nn=Nn;   p->pEle=pEle;
    p->pMat=pMat;   p->pNodxy=pNodxy;   p->p_displs=p_displs;   p->p_force=p_force;
    return 1;
}
```

为了验证读取数据的正确性，采用如下函数将读取的数据输出到屏幕上。

```
int FEM_Link_Print(Data_Link m_p)
```

```
{
    int Nn, Ne, Nm, Nb, Nf;//分别是节点总数、单元总数、材料总数、位移边界条件总数、载荷条件总数
    double *pMat, *pNodxy;//分别存储材料常数，节点坐标，边界条件，载荷条件
    int *pEle;//存储单元节点编号
    int i;//用于循环
    Data_bound *p_displs,*p_force;//位移边界条件，力边界条件
    ////赋值
    Nb=m_p.Nb;  Ne=m_p.Ne;  Nf=m_p.Nf;  Nm=m_p.Nm; Nn=m_p.Nn;  pEle=m_p.pEle;
    pMat=m_p.pMat;  pNodxy=m_p.pNodxy;  p_displs=m_p.p_displs;  p_force=m_p.p_force;
    ///
    printf("材料总数\n");    printf("%d\n",Nm);
    if(pMat==NULL)
    {
        printf("pMat malloc error in Output!\n");
        return 0;
    }
    printf("材料弹性模量泊松比以及杆的面积\n");
    for(i=0;i<Nm;i++)
    {
        printf("%e %e %e\n",pMat[3*i],pMat[3*i+1],pMat[3*i+2]);
    }
    //节点
    printf("节点总数\n");    printf("%d\n",Nn);
    if(pNodxy==NULL)
    {
        printf("pNodxy malloc error in Output!\n");
        return 0;
    }
    printf("节点坐标（这里以（1）点为坐标原点，假设 1 号杆的长度为 1）\n");
```

```
for(i=0;i<Nn;i++)
{
    printf("%e %e\n",pNodxy[2*i],pNodxy[2*i+1]);
}
///////单元
printf("单元总数\n");    printf("%d\n",Ne);
if(pEle==NULL)
{
    printf("pEle malloc error in Output!\n");
    return 0;
}
printf("单元节点的总体编号\n");
for(i=0;i<Ne;i++)
{
    printf("%d %d %d\n",pEle[3*i]+1,pEle[3*i+1]+1,pEle[3*i+2]+1);
}
///////位移边界条件
printf("位移约束总数\n");    printf("%d\n",Nb);
if(p_displs==NULL)
{
    printf("p_displs malloc error in Output!\n");
    return 0;
}
printf("具体的位移约束（第一个数字表示节点编号，后面 x，y 表示约束方向，最后一个数字表示位移约束大小）\n");
for(i=0;i<Nb;i++)
{
    printf("%d %c %e\n",p_displs[i].id,p_displs[i].dir,p_displs[i].v);
}
///////力边界条件
```

```
    printf("载荷总数\n");    printf("%d\n",Nf);
    if(p_force==NULL)
    {
        printf("p_force malloc error in Output!\n");
        return 0;
    }
    printf("具体载荷（第一个数字表示节点编号，后面 x，y 表示载荷方向，最后一个数字表示载荷大小）\n");
    for(i=0;i<Nf;i++)
    {
        printf("%d %c %e\n",p_force[i].id,p_force[i].dir,p_force[i].v);
    }
    return 1;
}
```

1.3.3 计算单元刚度矩阵并合成总体刚度矩阵

计算单元刚度矩阵的目的是为了计算出总体刚度矩阵。为了说明总体刚度矩阵每一个元素的含义，我们把方程(1.27)再次写在下面。

$$\begin{bmatrix} R_{(1)x} \\ 0 \\ R_{(2)x} \\ R_{(2)y} \\ 0 \\ -P \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} k_{11}^1+k_{11}^2 & k_{12}^1+k_{12}^2 & 0 & 0 & k_{13}^2 & k_{14}^2 & k_{13}^1 & k_{14}^1 \\ k_{21}^1+k_{21}^2 & k_{22}^1+k_{22}^2 & 0 & 0 & k_{23}^2 & k_{24}^2 & k_{23}^1 & k_{24}^1 \\ 0 & 0 & k_{11}^4 & k_{12}^4 & 0 & 0 & k_{13}^4 & k_{14}^4 \\ 0 & 0 & k_{21}^4 & k_{22}^4 & 0 & 0 & k_{23}^4 & k_{24}^4 \\ k_{31}^2 & k_{32}^2 & 0 & 0 & k_{33}^2+k_{33}^3 & k_{34}^2+k_{34}^3 & k_{31}^3 & k_{32}^3 \\ k_{41}^2 & k_{42}^2 & 0 & 0 & k_{43}^2+k_{43}^3 & k_{44}^2+k_{44}^3 & k_{41}^3 & k_{42}^3 \\ k_{31}^1 & k_{32}^1 & k_{31}^4 & k_{32}^4 & k_{13}^3 & k_{14}^3 & k_{33}^1+k_{11}^3+k_{33}^4 & k_{34}^1+k_{12}^3+k_{34}^4 \\ k_{41}^1 & k_{42}^1 & k_{41}^4 & k_{42}^4 & k_{23}^3 & k_{24}^3 & k_{43}^1+k_{21}^3+k_{43}^4 & k_{44}^1+k_{22}^3+k_{44}^4 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ v_{(1)} \\ 0 \\ 0 \\ u_{(3)} \\ v_{(3)} \\ u_{(4)} \\ v_{(4)} \end{bmatrix} \tag{1.38}$$

仿照方程(1.8)，我们可以将方程(1.27)写成如下的矩阵形式：

$$\mathbf{F} = \mathbf{K} \cdot \mathbf{u} \tag{1.39}$$

其中 $\mathbf{F} = \begin{bmatrix} F_1 & F_2 & F_3 & F_4 & F_5 & F_6 & F_7 & F_8 \end{bmatrix}^T$，这里 F_i（i=1-8）与方程(1.27)中左边的元素对应。同样的 $\mathbf{u} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 & u_5 & u_6 & u_7 & u_8 \end{bmatrix}^T$，这里 u_i（i=1-8）与方程(1.27)中右边位移向量的元素对应。总体刚度矩阵 K 的元素定义为 k_{ij}，i,j=1-8。k_{ij} 中第一个脚标 i 表示 F 中第 i 个元素，j 表示 $\mathbf{u}$ 向量中第 j 个元素。因此 k_{ij} 可以理解为是第 j 个位移元素对 F 中第 i 个元素的作用系数。如果 i 和 j 之

间没有单元直接连接，则 k_{ij} =0。如果 i 和 j 之间通过多个单元直接连接，则 k_{ij} 应等于所有这些单元的单元刚度矩阵对应元素之和。我们可以根据 k_{ij} 的这个含义来构造总体刚度矩阵。

根据图 1.4，u_1 为节点 1 的 x 方向位移，F_1 为节点 1 的 x 方向作用力。由于节点 1 连接了两个单元，因此 u_1 通过单元 1 和 2 作用于 F_1。单元 1 中反映 u_1 对 F_1 作用的单元刚度系数为 k_{11}^1，单元 2 中反映 u_1 对 F_1 作用的单元刚度系数为 k_{11}^2，因此 $k_{11}=k_{11}^1+k_{11}^2$。

同理，u_2 通过单元 1 和 2 作用于 F_1。单元 1 中反映 u_2 对 F_1 作用的单元刚度系数为 k_{12}^1，单元 2 中反映 u_2 对 F_1 作用的单元刚度系数为 k_{12}^2，因此 $k_{12}=k_{12}^1+k_{12}^2$。

节点（2）与节点（1）之间没有单元直接连接，因此 u_3，u_4 对 F_1 的作用均为 0，$k_{13}=k_{14}=0$。

节点（3）与节点（1）之间通过 2 号单元直接连接，因此 u_5，u_6 对 F_1 的作用系数分别等于 k_{13}^2 和 k_{14}^2，也就是 $k_{15}=k_{13}^2$，$k_{16}=k_{14}^2$。

节点（4）与节点（1）之间通过 1 号单元直接连接，因此 u_7，u_8 对 F_1 的作用系数分别等于 k_{13}^1 和 k_{14}^1，也就是 $k_{17}=k_{13}^1$，$k_{18}=k_{14}^1$。

K 中第二行反映的是各节点对 F_2 的作用。F_2 是节点 1 沿 y 方向的作用力，u_1，u_2 通过单元 1 和单元 2 作用于 F_2。因此 $k_{21}=k_{21}^1+k_{21}^2$，$k_{22}=k_{22}^1+k_{22}^2$。

节点（2）与节点（1）之间没有单元直接连接，因此 u_3，u_4 对 F_2 的作用均为 0，$k_{23}=k_{24}=0$。

节点（3）与节点（1）之间通过 2 号单元直接连接，因此 u_5，u_6 对 F_2 的作用系数分别等于 k_{23}^2 和 k_{24}^2，也就是 $k_{25}=k_{23}^2$，$k_{26}=k_{24}^2$。

节点（4）与节点（1）之间通过 1 号单元直接连接，因此 u_7，u_8 对 F_2 的作用系数分别等于 k_{23}^1 和 k_{24}^1，也就是 $k_{27}=k_{23}^1$，$k_{28}=k_{24}^1$。

同理，可以写出 **K** 中所有元素的表达式。

总体刚度矩阵 **K** 中元素的这种性质可以用来构造 **K** 矩阵。

首先根据节点数和维数申请 **K** 的内存空间，对 **K** 清零。然后遍历所有单元，将每个单元刚度矩阵的元素 k_{ij}^e 叠加到 **K** 中对应位置上。当所有单元遍历完，刚度矩阵也完成赋值。

```
int FEM_Link_Kg(Data_Link m_p,Data_Link_KgFg *p_KF)
{
```

```
int Nn, Ne, Nm, Nb, Nf;//分别是节点总数、单元总数、材料总数、位移边界条件总数、载荷条件总数
double *pMat, *pNodxy;//分别存储材料常数，节点坐标，边界条件，载荷条件
int *pEle;//存储单元节点编号
int i,j,n;//用于循环
Data_bound *p_displs,*p_force;//位移边界条件，力边界条件
double *pKg=NULL,*pFg=NULL;//用于存储总体刚度矩阵和总体载荷向量
double l,csxt,sinxt;//分别用于存储杆单元的长度，cos（xt）和 sin（xt）
double x1,y1,x2,y2;//分别用于存储杆单元的两端点坐标。
double E,mu,A;//Young's modulus ,Poison's ratio and cross area of beam
int id[2],idm,id1,id2;//单元节点 I 和 J 的总体编号，以及单元的材料编号。均从 0 开始
double Ke[4][4];//用于储存单元刚度矩阵。
////赋值
Nb=m_p.Nb;  Ne=m_p.Ne;  Nf=m_p.Nf;  Nm=m_p.Nm; Nn=m_p.Nn;  pEle=m_p.pEle;
pMat=m_p.pMat;  pNodxy=m_p.pNodxy;  p_displs=m_p.p_displs;  p_force=m_p.p_force;
////给总刚 pKg 和总体载荷向量 pFg 赋值
pKg=(double *)malloc(2*Nn*2*Nn*sizeof(double));
pFg=(double *)malloc(2*Nn*sizeof(double));
if(pKg==NULL||pFg==NULL)
{
    printf("FEM_Link_Kg error!Not enough memory for pKg or pFg!\n");
    return 0;
}
////清零
for(i=0;i<2*Nn*2*Nn;i++)
{
    pKg[i]=0;
}
for(i=0;i<2*Nn;i++)
{
```

```
    pFg[i]=0;
}
//开始遍历单元，同时装配总体刚度矩阵。
for(n=0;n<Ne;n++)
{
    //先计算单元 n 的刚度矩阵
    id[0]=pEle[3*n+1];      id[1]=pEle[3*n+2];      x1=pNodxy[2*id[0]];
    y1=pNodxy[2*id[0]+1];   x2=pNodxy[2*id[1]];     y2=pNodxy[2*id[1]+1];
    idm=pEle[3*n];
    //
    E=pMat[3*idm];          mu=pMat[3*idm+1];       A=pMat[3*idm+2];
    l=sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
    if(l<1e-6)
    {
        printf("l is small than 1e-6, may be equal to 0! error\n");
        return 0;
    }
    csxt=(x2-x1)/l;         sinxt=(y2-y1)/l;
    //计算总体刚度矩阵
    Ke[0][0]=E*A/l*csxt*csxt;       Ke[0][1]=E*A/l*sinxt*csxt;
    Ke[0][2]=-E*A/l*csxt*csxt;      Ke[0][3]=-E*A/l*sinxt*csxt;
    //
    Ke[1][0]=E*A/l*sinxt*csxt;      Ke[1][1]=E*A/l*sinxt*sinxt;
    Ke[1][2]=-E*A/l*sinxt*csxt;     Ke[1][3]=-E*A/l*sinxt*sinxt;
    //
    Ke[2][0]=-E*A/l*csxt*csxt;      Ke[2][1]=-E*A/l*sinxt*csxt;
    Ke[2][2]=E*A/l*csxt*csxt;       Ke[2][3]=E*A/l*sinxt*csxt;
    //
    Ke[3][0]=-E*A/l*sinxt*csxt;     Ke[3][1]=-E*A/l*sinxt*sinxt;
    Ke[3][2]=E*A/l*sinxt*csxt;      Ke[3][3]=E*A/l*sinxt*sinxt;
```

```
        //开始装配总体刚度矩阵 pKg 按列排，即先排第一列，排满后再排第二列
        for(i=0;i<2;i++)
        {
            for(j=0;j<2;j++)
            {   id1=id[i];id2=id[j];
                pKg[2*id1+2*Nn*(2*id2)]=pKg[2*id1+2*Nn*(2*id2)]+Ke[2*i][2*j];
                pKg[2*id1+1+2*Nn*(2*id2)]=pKg[2*id1+1+2*Nn*(2*id2)]+Ke[2*i+1][2*j];
                pKg[2*id1+2*Nn*(2*id2+1)]=pKg[2*id1+2*Nn*(2*id2+1)]+Ke[2*i][2*j+1];
            pKg[2*id1+1+2*Nn*(2*id2+1)]=pKg[2*id1+1+2*Nn*(2*id2+1)]+Ke[2*i+1][2*j+1];
            }
        }
    }
    //开始计算总体载荷矩阵
    for(i=0;i<Nf;i++)
    {
        id1=p_force[i].id;
        if(p_force[i].dir=='x')
        {
            pFg[2*id1]=p_force[i].v;
        }
        else
        {
            pFg[2*id1+1]=p_force[i].v;
        }
    }
    p_KF->pFg=pFg;   p_KF->pKg=pKg;
    return 1;
}
```

1.3.4 考虑位移约束的总体刚度矩阵

前文提到，由于存在刚体位移，刚度矩阵是不可逆的，必须要加入位移约束条件才能使得总体刚度矩阵可逆，也就是由方程(1.27)变到(1.28)后方程才是可以求解的。施加位移约束条件的方法有很多种，最常用的有“划线法”、“对角元置一法”和“乘大数法”。后两种方法属于近似方法，但很方便实用。第一种方法实际上就是方程(1.27)变到(1.28)的直接体现。后两种方法将在后续章节介绍，本章先介绍“划线法”。

如果某个位移 u_j 已知，我们可以先将 $\boldsymbol{k}_{ij}\cdot\boldsymbol{u}_j$，叠加到 $\boldsymbol{F}_i$ 上。假设一共有 N_b 个位移已知，并且编号为 $i_1,i_2,\cdots i_\alpha\cdots i_{Nb}$。那么经过上述步骤后，方程(1.39)变为如下方程形式：

$$\begin{bmatrix} F_1-\sum_{\alpha=1}^{Nb}k_{1i_\alpha}u_{i_\alpha} \\ F_2-\sum_{\alpha=1}^{Nb}k_{1i_\alpha}u_{i_\alpha} \\ F_3-\sum_{\alpha=1}^{Nb}k_{1i_\alpha}u_{i_\alpha} \\ F_4-\sum_{\alpha=1}^{Nb}k_{1i_\alpha}u_{i_\alpha} \\ F_5-\sum_{\alpha=1}^{Nb}k_{1i_\alpha}u_{i_\alpha} \\ F_6-\sum_{\alpha=1}^{Nb}k_{1i_\alpha}u_{i_\alpha} \\ F_7-\sum_{\alpha=1}^{Nb}k_{1i_\alpha}u_{i_\alpha} \\ F_8-\sum_{\alpha=1}^{Nb}k_{1i_\alpha}u_{i_\alpha} \end{bmatrix}_{8\times1} = \begin{bmatrix} k_{11} & \cdots & k_{1j} & \cdots & k_{18} \\ k_{21} & \cdots & k_{2j} & \cdots & k_{28} \\ k_{31} & \cdots & k_{3j} & \cdots & k_{38} \\ k_{41} & \cdots & k_{4i} & \cdots & k_{48} \\ k_{51} & \cdots & k_{5i} & \cdots & k_{58} \\ k_{61} & \cdots & k_{6i} & \cdots & k_{68} \\ k_{71} & \cdots & k_{7i} & \cdots & k_{78} \\ k_{81} & \cdots & k_{8i} & \cdots & k_{88} \end{bmatrix}_{8\times(8-Nb)} \cdot \begin{bmatrix} u_1 \\ \vdots \\ u_j \\ \vdots \\ u_8 \end{bmatrix}_{(8-Nb)\times1} \tag{1.40}$$

此时刚度矩阵成为 8×（8-Nb）的非方阵，不能求逆运算。我们注意到，如果某个位移 u_j 已知那么 F_j 必然是约束反力。由于约束反力需要将所有位移求解出来后才能计算得到。因此我们可以将与约束反力有关的方程从(1.40)移除，即

$$\begin{bmatrix} F_1-\sum_{\alpha=1}^{Nb}k_{1i_\alpha}u_{i_\alpha} \\ \vdots \\ F_i-\sum_{\alpha=1}^{Nb}k_{ii_\alpha}u_{i_\alpha} \\ \vdots \\ F_8-\sum_{\alpha=1}^{Nb}k_{8i_\alpha}u_{i_\alpha} \end{bmatrix}_{(8-Nb)\times1} = \begin{bmatrix} k_{11} & \cdots & k_{1j} & \cdots & k_{18} \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ k_{i1} & \cdots & k_{ij} & \cdots & k_{i8} \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ k_{81} & \cdots & k_{8j} & \cdots & k_{88} \end{bmatrix}_{(8-Nb)\times(8-Nb)} \cdot \begin{bmatrix} u_1 \\ \vdots \\ u_j \\ \vdots \\ u_8 \end{bmatrix}_{(8-Nb)\times1} \tag{1.41}$$

这样方程(1.41)就能够求解了。由方程(1.41)计算出所有位移后，将位移代入(1.28)后即可计算出所有的反力。需要注意的是方程(1.40)和(1.41)右边的刚度矩阵以及位移向量，如果正好约束的是第 1 个位移或者第 8 个位移，则相应的行和列也要划去。

本部分进行程序设计时，目的就是要构造方程(1.41)中的刚度矩阵和载荷向量。构造之前需要创建一个结构体数组 p_ub，结构体包含一个整数和一个浮点数。整数变量为 ID，存储删除已知位移变量后，该变量在方程(1.41)位移向量中的位置。如果该位移就是被删除的位移，那么 ID=-1。浮点数变量名为 value，存储已知位移变量的位移值，如果该位移是未知位移，则 value=0。

对于任意一个单元刚度矩阵的元素 k_{mn}^{e}，先查到其在未划去已知位移的总体刚度矩阵中的位置 k_{ij}，然后查到第 i 个位移的 ID，即 m_ub[i].ID 和第 j 个位移的 ID，即 m_ub[j].ID。如果 m_ub[i].ID<0，那么这个位移对应的方程不进入方程(1.41)，因此不做任何处理。如果 m_ub[i].ID>0，且 m_ub[j].ID<0，则计算出 $k_{ij}u_j$ 叠加到方程(1.41)左边力向量的 m_ub[i].ID 位置上。如果 m_ub[i].ID>0，且 m_ub[j].ID>0，则将 k_{ij} 叠加到方程(1.41)中刚度矩阵的第 m_ub[i].ID 行和第 m_ub[j].ID 列位置。

```
int FEM_Link_KgYH(Data_Link m_p,Data_Link_KgFg *p_KF)//计算约化后的刚度矩阵和载荷向量。
{
    int Nn, Ne, Nm, Nb, Nf;//分别是节点总数、单元总数、材料总数、位移边界条件总数、载荷条件总数
    double *pMat, *pNodxy;//分别存储材料常数，节点坐标，边界条件，载荷条件
    int *pEle;//存储单元节点编号
    int i,j,n;//用于循环
    Data_bound *p_displs,*p_force;//位移边界条件，力边界条件
    double *pKg=NULL,*pFg=NULL;//用于存储总体刚度矩阵和总体载荷向量
    double l,csxt,sinxt;//分别用于存储杆单元的长度，cos（xt）和 sin（xt）
    double x1,y1,x2,y2;//分别用于存储杆单元的两端点坐标。
    double E,mu,A;//Young's modulus ,Poison's ratio and cross area of beam
    int id[2],idm,idu[4],id1;//单元节点 I 和 J 的总体编号，以及单元的材料编号。均从 0 开始
    double Ke[4][4],idv[4];//用于储存单元刚度矩阵。
    Data_Link_UID *p_uid=NULL;//约化位移 id
    int Nu;//约化后未知位移的总数
    ////赋值
    Nb=m_p.Nb;  Ne=m_p.Ne;  Nf=m_p.Nf;  Nm=m_p.Nm; Nn=m_p.Nn;  pEle=m_p.pEle;
    pMat=m_p.pMat;  pNodxy=m_p.pNodxy;  p_displs=m_p.p_displs;  p_force=m_p.p_force;
```

```
////先构造 p_uid,大小 2×Nn
p_uid=(Data_Link_UID *)malloc(2*Nn*sizeof(Data_Link_UID));
if(p_uid==NULL)
{
    printf("FEM_Link_KgYH error:No enough memory for p_uid!\n");
    return 0;
}
///初始化 p_uid
for(i=0;i<2*Nn;i++)
{
    p_uid[i].ID=0;      p_uid[i].v=0;
}
//
for(i=0;i<Nb;i++)
{
    if(p_displs[i].dir=='x')
    {
        p_uid[2*p_displs[i].id].ID=-1;              p_uid[2*p_displs[i].id].v=p_displs[i].v;
    }
    else
    {
        p_uid[2*p_displs[i].id+1].ID=-1;            p_uid[2*p_displs[i].id+1].v=p_displs[i].v;
    }
}
//计算 Nu 的大小，注意不区分 x 和 y
Nu=0;
for(i=0;i<2*Nn;i++)
{
    if(p_uid[i].ID==0)
    {
```

```
            p_uid[i].ID=Nu;
            Nu++;
        }
    }
    ////给总刚 pKg 和总体载荷向量 pFg 赋值============================
    pKg=(double *)malloc(Nu*Nu*sizeof(double));
    pFg=(double *)malloc(Nu*sizeof(double));
    if(pKg==NULL||pFg==NULL)
    {
        printf("FEM_Link_Kg error!Not enough memory for pKg or pFg!\n");
        return 0;
    }
    ////清零
    for(i=0;i<Nu*Nu;i++)
    {
        pKg[i]=0;
    }
    for(i=0;i<Nu;i++)
    {
        pFg[i]=0;
    }
    //开始遍历单元，同时装配总体刚度矩阵。
    for(n=0;n<Ne;n++)
    {
        //先计算单元 n 的刚度矩阵
        id[0]=pEle[3*n+1];      id[1]=pEle[3*n+2];      x1=pNodxy[2*id[0]];
        y1=pNodxy[2*id[0]+1];       x2=pNodxy[2*id[1]];         y2=pNodxy[2*id[1]+1];
        idm=pEle[3*n];
        //
        E=pMat[3*idm];          mu=pMat[3*idm+1];           A=pMat[3*idm+2];
```

```
l=sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
if(l<1e-6)
{
    printf("l is small than 1e-6, may be equal to 0! error\n");
    return 0;
}
csxt=(x2-x1)/l;          sinxt=(y2-y1)/l;
//计算总体刚度矩阵
Ke[0][0]=E*A/l*csxt*csxt;        Ke[0][1]=E*A/l*sinxt*csxt;
Ke[0][2]=-E*A/l*csxt*csxt;       Ke[0][3]=-E*A/l*sinxt*csxt;
//
Ke[1][0]=E*A/l*sinxt*csxt;       Ke[1][1]=E*A/l*sinxt*sinxt;
Ke[1][2]=-E*A/l*sinxt*csxt;      Ke[1][3]=-E*A/l*sinxt*sinxt;
//
Ke[2][0]=-E*A/l*csxt*csxt;       Ke[2][1]=-E*A/l*sinxt*csxt;
Ke[2][2]=E*A/l*csxt*csxt;        Ke[2][3]=E*A/l*sinxt*csxt;
//
Ke[3][0]=-E*A/l*sinxt*csxt;      Ke[3][1]=-E*A/l*sinxt*sinxt;
Ke[3][2]=E*A/l*sinxt*csxt;       Ke[3][3]=E*A/l*sinxt*sinxt;
////////////////输出单元刚度矩阵
printf("------------ele %d----------\n",n+1);
for(i=0;i<4;i++)
{
    for(j=0;j<4;j++)
    {
        printf("%lf ",Ke[i][j]);
    }
    printf("\n");
}
//开始装配总体刚度矩阵 pKg 按列排，即先排第一列，排满后再排第二列
```

```
        idu[0]=p_uid[2*id[0]].ID;           idu[1]=p_uid[2*id[0]+1].ID;
        idu[2]=p_uid[2*id[1]].ID;           idu[3]=p_uid[2*id[1]+1].ID;
        idv[0]=p_uid[2*id[0]].v;       idv[1]=p_uid[2*id[0]+1].v;
        idv[2]=p_uid[2*id[1]].v;       idv[3]=p_uid[2*id[1]+1].v;
        for(i=0;i<4;i++)
        {
            if(idu[i]!=-1)
            {
                for(j=0;j<4;j++)
                {
                    if(idu[j]==-1)//已知位移，应该把它叠加到载荷向量上
                    {
                        pFg[idu[i]]=pFg[idu[i]]-Ke[i][j]*idv[j];
                    }
                    else//说明是未知位移，应该叠加刚度矩阵
                    {
                        pKg[idu[i]+Nu*idu[j]]=pKg[idu[i]+Nu*idu[j]]+Ke[i][j];
                    }
                }
            }
        }
    }
    //开始计算总体载荷矩阵
    for(i=0;i<Nf;i++)
    {
        if(p_force[i].dir=='x')
        {
            id1=2*p_force[i].id;
        }
        else
```

```
        {
            id1=2*p_force[i].id+1;
        }
        pFg[p_uid[id1].ID]=pFg[p_uid[id1].ID]+p_force[i].v;
    }
    //赋值
    p_KF->pFg=pFg;   p_KF->pKg=pKg;  p_KF->Nu=Nu;     p_KF->p_uid=p_uid;
    return 1;
}
```

1.3.5 求解及计算结果

首先定义一个输出向量的函数，用于将总体刚度矩阵和载荷向量输出到文件。函数参数 p 指向矩阵的指针，Ni 是矩阵的行数，Nj 是矩阵的列数，filename 是文件名。

```
int MATH_OutMatrixD(double *p,int Ni,int Nj,char *filename)
{   int i,j;
    FILE *pf=NULL;
    if(p==NULL)
    {   printf("OutMatixD error:p==NULL\n");
        return 0;  }
    pf=fopen(filename,"w");
    if(pf==NULL)
    {   printf("OutMatixD error:pf==NULL\n");
        return 0;  }
    for(i=0;i<Ni;i++)
    {   for(j=0;j<Nj;j++)
        {fprintf(pf,"%15.2lf ",p[i+Ni*j]);}
        fprintf(pf,"\n");}
    fclose(pf);
    return 1;
}
```

下面定义两个函数 mx_inver 和 mx_multipG 用于矩阵的求逆运算和乘法运算。

```
//全选主元求逆
int mx_inver(double *mx_a,int N)//mx_a 是指向方阵的指针，N 是方阵的阶数，求逆后存于 mx_a。
{   int i,j,m,k,Maxm,flag;
    int *p_i,*p_j;
    double D,S;
    p_i=(int *)malloc(2*N*sizeof(int));
    p_j=(int *)malloc(2*N*sizeof(int));
    if(p_i==NULL||p_j==NULL)
    {   printf("mx_inver malloc p_i or p_j fail!\n");
        return 1;
    }
    m=0;    flag=0;
    for(k=0;k<N;k++)
    {   //全选主元
        D=mx_a[k+N*k]*mx_a[k+N*k];
        for(i=k;i<N;i++)
        {   for(j=k;j<N;j++)
            {   if(mx_a[i+N*j]*mx_a[i+N*j]>D)
                {   D=mx_a[i+N*j]*mx_a[i+N*j];
                    p_i[2*m]=k;  p_i[2*m+1]=i;     p_j[2*m]=k;  p_j[2*m+1]=j;     flag=1;
                }
            }
        }
        //如果 D 接近 0 则矩阵奇异
        if(D<1e-40)
        {   printf("Matirx is sigular!\n");
            return 2;  }
        if(flag)
        {   //交换
```

```
        for(i=0;i<N;i++)
        {     S=mx_a[i+N*p_j[2*m+1]];
              mx_a[i+N*p_j[2*m+1]]=mx_a[i+N*k];
              mx_a[i+N*k]=S;    }
        for(j=0;j<N;j++)
        {     S=mx_a[p_i[2*m+1]+N*j];   mx_a[p_i[2*m+1]+N*j]=mx_a[k+N*j];
              mx_a[k+N*j]=S;
        }
        m++;       flag=0;
}
//bianhuang
mx_a[k+N*k]=1/mx_a[k+N*k];
//第 k 行
for(j=0;j<k;j++)
{     mx_a[k+N*j]=mx_a[k+N*k]*mx_a[k+N*j];       }
for(j=k+1;j<N;j++)
{     mx_a[k+N*j]=mx_a[k+N*k]*mx_a[k+N*j];       }
//行列 11
for(i=0;i<k;i++)
{     for(j=0;j<k;j++)
      {mx_a[i+N*j]=mx_a[i+N*j]-mx_a[i+N*k]*mx_a[k+N*j];  }
}
//行列 12
for(i=0;i<k;i++)
{     for(j=k+1;j<N;j++)
      {     mx_a[i+N*j]=mx_a[i+N*j]-mx_a[i+N*k]*mx_a[k+N*j];    }
}
//行列 21
for(i=k+1;i<N;i++)
{     for(j=0;j<k;j++)
```

```
            {    mx_a[i+N*j]=mx_a[i+N*j]-mx_a[i+N*k]*mx_a[k+N*j];   }
        }
        //行列 22
        for(i=k+1;i<N;i++)
        {    for(j=k+1;j<N;j++)
            {    mx_a[i+N*j]=mx_a[i+N*j]-mx_a[i+N*k]*mx_a[k+N*j];   }
        }
        //第 k 列
        for(i=0;i<k;i++)
        {    mx_a[i+N*k]=-mx_a[i+N*k]*mx_a[k+N*k];            }
        for(i=k+1;i<N;i++)
        {    mx_a[i+N*k]=-mx_a[i+N*k]*mx_a[k+N*k];            }
    }
    Maxm=m;
    //还原
    for(m=Maxm-1;m>=0;m--)
    {    //行变换
        for(j=0;j<N;j++)
        {    S=mx_a[p_j[2*m]+N*j];      mx_a[p_j[2*m]+N*j]=mx_a[p_j[2*m+1]+N*j];
            mx_a[p_j[2*m+1]+N*j]=S;
        }
        //列交换
        for(i=0;i<N;i++)
        {    S=mx_a[i+N*p_i[2*m]];      mx_a[i+N*p_i[2*m]]=mx_a[i+N*p_i[2*m+1]];
            mx_a[i+N*p_i[2*m+1]]=S;
        }
    }
    return 0;
}
//矩阵乘法
```

int mx_multipG(double * mx_a,double * mx_b,double * mx_c,int Nai,int Naj,int Nbi,int Nbj)//普通乘法实现 mx_a*mx_b=mx_c 的运算。Nai、Naj、Nbi、Nbj 分别是矩阵 mx_a 的行数、列数以及 mx_b 矩阵的行数和列数。

```
{   int i,j,k;
    //如果两相乘矩阵不满足相乘条件，则返回 1
    if(Naj!=Nbi)
        return 1;
    for(i=0;i<Nai;i++)
    {   for(j=0;j<Nbj;j++)
        {   mx_c[i+Nai*j]=0;
            for(k=0;k<Naj;k++)
            {   mx_c[i+Nai*j]=mx_c[i+Nai*j]+mx_a[i+Nai*k]*mx_b[k+Nbi*j];   }
        }
    }
    return 0;
}
```

int FEM_Link_Solve(Data_Link m_data, Data_Link_KgFg *p,char *filename)//将计算的位移输出到 filename 文件中

```
{
    double *p_u=NULL;   int N,i;   FILE *pf=NULL;
    p_u=(double *)malloc(p->Nu*sizeof(double));
    if(p_u==NULL)
    {   printf("FEM_Link_Solve error: p_u==NULL\n");
        return 0;
    }
    mx_inver(p->pKg,p->Nu);   mx_multipG(p->pKg,p->pFg,p_u,p->Nu,p->Nu,p->Nu,1);
    N=0;
    for(i=0;i<2*m_data.Nn;i++)
    {   if(p->p_uid[i].ID>-1)
        {   p->p_uid[i].v=p_u[p->p_uid[i].ID];   }
```

```
    }
    pf=fopen(filename,"w");
    for(i=0;i<m_data.Nn;i++)
    {    fprintf(pf,"%5d %12.5e %12.5e\n",i+1,p->p_uid[2*i].v,p->p_uid[2*i+1].v);    }
    fclose(pf);
    return 1;
}
int main()
{
    Data_Link_KgFg m_KF;
    FEM_Link_Read("Data.txt",&m_data_link);//读取 Data.txt 中的数据，存入 m_data_link 中。
    FEM_Link_Print(m_data_link);//输出 m_data_link 的数据到屏幕，验证读取的正确性。
    FEM_Link_KgYH(m_data_link,&m_KF);//根据读取的数据计算单元刚度矩阵、总体刚度矩阵和总体载荷向量。在合成总体刚度矩阵和总体载荷向量的时候就考虑了位移约束的作用。
    MATH_OutMatrixD(m_KF.pKg,m_KF.Nu,m_KF.Nu,"KgYH.txt");//将考虑约束条件的总体刚度矩阵输出到 KgYH.txt 中。
    MATH_OutMatrixD(m_KF.pFg,m_KF.Nu,1,"FgYH.txt");//将考虑约束条件的总体载荷向量输出到 FgYH.txt 中。
    FEM_Link_Solve(m_data_link,&m_KF,"U.txt");//求解方程，将节点位移解输出到 U.txt 文件中。
    return 0;
}
```

将上述函数写 FEM_Link.cpp 文件，运行后打开 U.txt 得到节点位移。入计算后输出节点位移：

$$\mathbf{u}=1\times10^{-3}\left[0\quad 0\quad 0\quad 0\quad 0.2498\quad -0.9561\quad -0.1766\quad -0.1767\right]^{T}$$

计算结果与 ANSYS 计算结果一致，表明计算结果是正确的。

1.4 弹性力学基本方程

前文 1.2 节和 1.3 节介绍了有限元法的基本概念和计算过程，事实上，可以应用前文的方法来计算桁架的应力和变形。但是工程中大多数结构都属于连续体。对于连续体怎么采用有限元法来进行计算呢。为了解决这一问题，我们先要学习弹性力学的相关知识。

1.4.1 应力状态

应力定义为单位面积受到的力，如图 1.5 及下式所示：

$$\vec{p}=\frac{\vec{F}}{A} \tag{1.42}$$

其中 $\vec{p}$ 是应力，$\vec{F}$ 是面积 A 上的载荷，A 是面积。因为力是一个矢量，面积是一个标量，根据式(1.42)，应力 $\vec{p}$ 也是矢量。

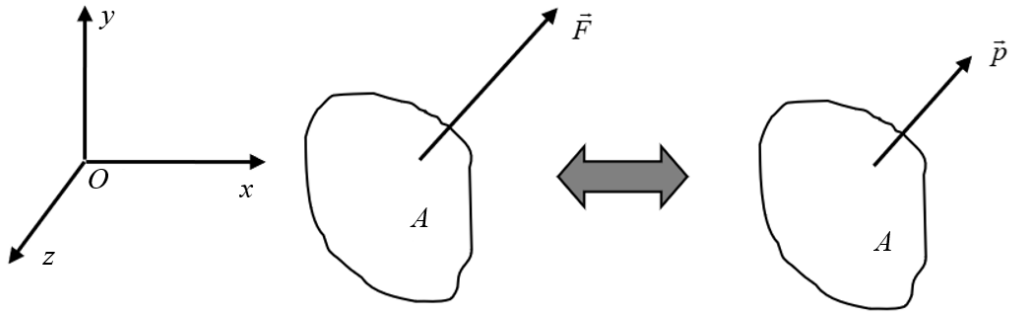


图 1.5 应力的定义

以平面问题为例，对于一个连续体，内部一点的应力不仅与外载荷及连续体的形状有关，还和该点所取的参考平面有关。如图 1.6 所示一个矩形平板，两端受均布拉伸载荷作用。假设均布载荷的大小为 q，根据常识，在平板内任一点 B，并取一个法向为 x 的平面，如图 1.6 所示。在这个平面上沿 x 方向的应力 σ=q，沿 y 方向的应力 τ=0。同样是 B 点，取一个法向与 y 轴反向的平面。在这个平面上沿着 y 方向的应力 σ=0，平行于平面的应力 τ=0。实际上，在同一个点存在无穷多个方向的面，在这些面上的正应力（垂直于面的方向）和剪应力（平行于面的应力）数值都可能不同。

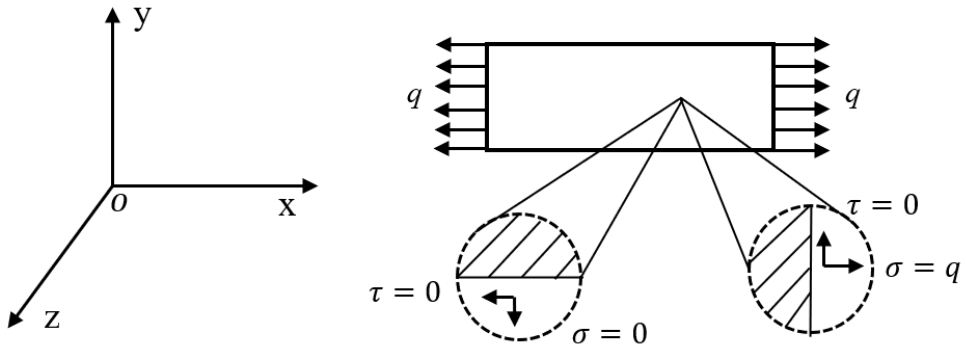


图 1.6 任意一点的应力

人们不禁要问，既然一点处的应力可以有无穷多个数值，那么用什么量来标定这个面所受应力的大小呢？人们发现，不管取的面的法向是什么方向，该面的应力与另外两个面上的应力之间存在着一定的关系。如图 1.7 所示的一点处的微元三角形，假设法向与 x 轴夹角为 θ 的斜面。该斜面的面积为 S，正应力和剪应力分别为 σ 和 τ 。选择另外两个斜面，其法向量分别与 x 轴和 y 轴反向。

定义应力的两个脚标，其中第一个脚标表示该应力方向与脚标所示坐标轴平行，第二个脚标表示该平面的法向与脚标所示坐标轴平行。如果应力所在平面的法向与坐标轴相同，该应力的正方向为坐标轴正方向。如果应力所在平面的法向与坐标轴相反，该应力的正方向为坐标轴反方向。例如σ_{xx}第一个脚标 x 表示该应力方向为 x 方向，第二个脚标表示该应力所在平面的法向与 x 轴平行。图中所示的σ_{xx}落在法向与 x 轴相反的平面上，因此图中σ_{xx}的正方向与 x 轴方向相反。τ_{yx}的第一个脚标表示该应力与 y 轴平行，第二个脚标 x 表示该应力落在法向与 x 轴平行的面上。由于图中该平面的法向与 x 轴相反，因此图中τ_{yx}的正方向为 y 轴的负方向。

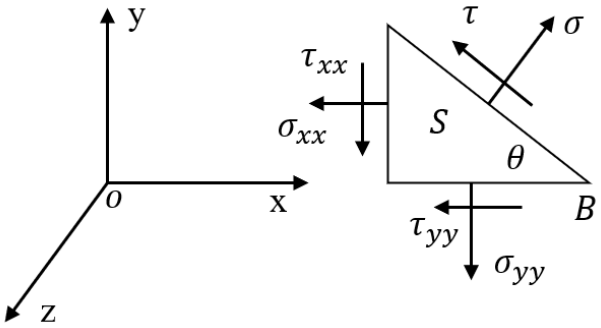


图 1.7 三角形微元体

根据三角形微元体的力平衡方程，可得：

$$\begin{cases}\sigma\cdot S\cdot\cos\theta-\tau\cdot S\cdot\sin\theta=\sigma_{xx}\cdot S\cdot\sin\theta+\tau_{xy}\cdot S\cdot\cos\theta\\ \sigma\cdot S\cdot\sin\theta+\tau\cdot S\cdot\cos\theta=\sigma_{yy}\cdot S\cdot\cos\theta+\tau_{yx}\cdot S\cdot\sin\theta\end{cases}\tag{1.43}$$

通过求解上式，可以将任意截面上的应力写成σ_{xx}、σ_{yy}、τ_{xy}和τ_{yx}的表达式。

$$\begin{cases}\sigma=\left(\sigma_{xx}+\sigma_{yy}\right)\cdot\cos\theta\sin\theta+\tau_{xy}\\ \tau=\sigma_{yy}\cdot\cos\theta\cos\theta-\sigma_{xx}\cdot\sin\theta\sin\theta\end{cases}\tag{1.44}$$

因此可以用σ_{xx}、σ_{yy}、τ_{xy}和τ_{yx}来表征一个点的应力状态。也就是说如果一个点的σ_{xx}、σ_{yy}、τ_{xy}和τ_{yx}已知了，这个点的应力状态就确定了。根据剪引力互等定理，$\tau_{xy}=\tau_{xy}$。因此，对于平面问题，一点的应力状态由σ_{xx}、σ_{yy}和τ_{xy}确定。对于三维问题，一点的应力状态由六个应力分量确定，即σ_{xx}、σ_{yy}、σ_{zz}、τ_{xy}、τ_{xz}、τ_{yz}。

1.4.2 应变状态和几何方程

一点的应变可以由线应变和角应变来表示。假设空间一个矢量$\mathrm{d}\vec{r}$，变形后变为$\mathrm{d}\vec{r}'$，如图 1.8 所示。

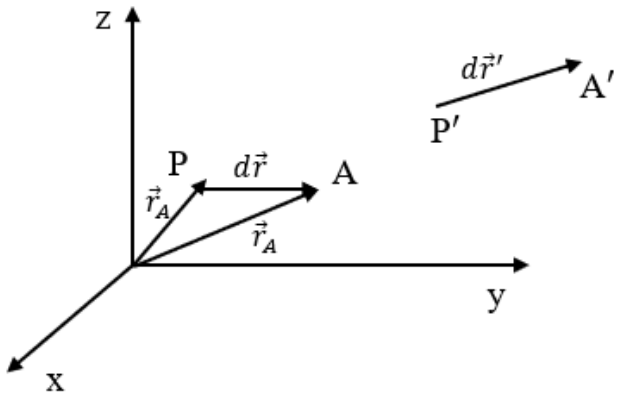


图 1.8 线应变

令 $|\mathrm{d}\vec{r}| = \mathrm{d}r, |\mathrm{d}\vec{r}'| = \mathrm{d}r'$

$$\varepsilon_n = \frac{\mathrm{d}r'}{\mathrm{d}r} - 1 \tag{1.45}$$

因为应变是由位移引起的，因此设法将 $\mathrm{d}r'$ 和 $\mathrm{d}r$ 表示成位移的表达式

$$\mathrm{d}r' = \sqrt{\left(x_{A'} - x_{P'}\right)^2 + \left(y_{A'} - y_{P'}\right)^2 + \left(z_{A'} - z_{P'}\right)^2} \tag{1.46}$$

$$\mathrm{d}r' = \sqrt{\left(x_A + u_A - x_P - u_P\right)^2 + \left(y_A + v_A - y_P - v_P\right)^2 + \left(z_A + w_A - z_P - w_P\right)^2} \tag{1.47}$$

$$\frac{\mathrm{d}r'}{\mathrm{d}r} = \sqrt{1 + 2\frac{\mathrm{d}x\mathrm{d}u}{\mathrm{d}r\mathrm{d}r} + \frac{\mathrm{d}u^2}{\mathrm{d}r^2} + 2\frac{\mathrm{d}y\mathrm{d}v}{\mathrm{d}r\mathrm{d}r} + \frac{\mathrm{d}v^2}{\mathrm{d}r^2} + 2\frac{\mathrm{d}z\mathrm{d}w}{\mathrm{d}r\mathrm{d}r} + \frac{\mathrm{d}w^2}{\mathrm{d}r^2}} \tag{1.48}$$

其中 u、v、w 分别表示参考点沿着 x、y、z 方向的位移。因为变形很小，$\frac{\mathrm{d}r'}{\mathrm{d}r}$ 接近 1，所以 $2\frac{\mathrm{d}x\mathrm{d}u}{\mathrm{d}r\mathrm{d}r} + \frac{\mathrm{d}u^2}{\mathrm{d}r^2} + 2\frac{\mathrm{d}y\mathrm{d}v}{\mathrm{d}r\mathrm{d}r} + \frac{\mathrm{d}v^2}{\mathrm{d}r^2} + 2\frac{\mathrm{d}z\mathrm{d}w}{\mathrm{d}r\mathrm{d}r} + \frac{\mathrm{d}w^2}{\mathrm{d}r^2}$ 是一小量，可以应用近似公式 $\sqrt{1+x} = 1 + \frac{1}{2}x$ 化简

$$\frac{\mathrm{d}r'}{\mathrm{d}r} = \sqrt{1 + 2\frac{\mathrm{d}x\mathrm{d}u}{\mathrm{d}r\mathrm{d}r} + \frac{\mathrm{d}u^2}{\mathrm{d}r^2} + 2\frac{\mathrm{d}y\mathrm{d}v}{\mathrm{d}r\mathrm{d}r} + \frac{\mathrm{d}v^2}{\mathrm{d}r^2} + 2\frac{\mathrm{d}z\mathrm{d}w}{\mathrm{d}r\mathrm{d}r} + \frac{\mathrm{d}w^2}{\mathrm{d}r^2}} \tag{1.49}$$

$$\frac{\mathrm{d}r'}{\mathrm{d}r} = 1 + l_1\frac{\mathrm{d}u}{\mathrm{d}r} + \frac{\mathrm{d}u^2}{2\mathrm{d}r^2} + m_1\frac{\mathrm{d}v}{\mathrm{d}r} + \frac{\mathrm{d}v^2}{2\mathrm{d}r^2} + n_1\frac{\mathrm{d}w}{\mathrm{d}r} + \frac{\mathrm{d}w^2}{2\mathrm{d}r^2} \tag{1.50}$$

忽略二次项

$$\frac{\mathrm{d}r'}{\mathrm{d}r} = 1 + l_1\frac{\mathrm{d}u}{\mathrm{d}r} + m_1\frac{\mathrm{d}v}{\mathrm{d}r} + n_1\frac{\mathrm{d}w}{\mathrm{d}r} \tag{1.51}$$

将上式展开

$$\begin{aligned}
\frac{\mathrm{d}u}{\mathrm{d}r} &= \frac{\partial u}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial r} + \frac{\partial u}{\partial z}\frac{\partial z}{\partial r} = \frac{\partial u}{\partial x}l_1 + \frac{\partial u}{\partial y}m_1 + \frac{\partial u}{\partial z}n_1 \\
\frac{\mathrm{d}v}{\mathrm{d}r} &= \frac{\partial v}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial v}{\partial y}\frac{\partial y}{\partial r} + \frac{\partial v}{\partial z}\frac{\partial z}{\partial r} = \frac{\partial v}{\partial x}l_1 + \frac{\partial v}{\partial y}m_1 + \frac{\partial v}{\partial z}n_1 \\
\frac{\mathrm{d}w}{\mathrm{d}r} &= \frac{\partial w}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial w}{\partial y}\frac{\partial y}{\partial r} + \frac{\partial w}{\partial z}\frac{\partial z}{\partial r} = \frac{\partial w}{\partial x}l_1 + \frac{\partial w}{\partial y}m_1 + \frac{\partial w}{\partial z}n_1
\end{aligned} \tag{1.52}$$

得

$$\frac{\mathrm{d}r'}{\mathrm{d}r}=1+\frac{\partial u}{\partial x}l_1^2+\frac{\partial u}{\partial y}l_1m_1+\frac{\partial u}{\partial z}l_1n_1+\frac{\partial v}{\partial x}m_1l_1+\frac{\partial v}{\partial y}m_1^2+\frac{\partial v}{\partial z}m_1n_1+\frac{\partial w}{\partial x}n_1l_1+\frac{\partial w}{\partial y}n_1m_1+\frac{\partial w}{\partial z}n_1^2 \tag{1.53}$$

最后导出正应变的表达式

$$\varepsilon_n=\frac{\partial u}{\partial x}l_1^2+\frac{\partial v}{\partial y}m_1^2+\frac{\partial w}{\partial z}n_1^2+\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)m_1l_1+\left(\frac{\partial u}{\partial z}+\frac{\partial w}{\partial x}\right)n_1l_1+\left(\frac{\partial v}{\partial z}+\frac{\partial w}{\partial y}\right)n_1m_1 \tag{1.54}$$

角应变可由图 1.9 来计算。假设两个向量 $\vec{r}_1$ 和 $\vec{r}_2$ 的夹角为θ，变形后变为 $\vec{r}_1'$ 和 $\vec{r}_2'$，夹角变为 θ'。

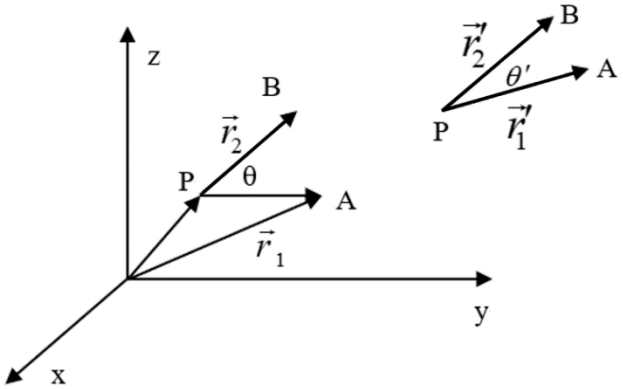


图 1.9 角应变

令矢量 $\overrightarrow{PA}$， $\overrightarrow{PB}$ 的方向矢量为 $[l_1,m_1,n_1]$ 和 $[l_2,m_2,n_2]$。矢量 $\overrightarrow{P'A'}$， $\overrightarrow{P'B'}$ 的方向矢量为 $[l_1',m_1',n_1']$ 和 $[l_2',m_2',n_2']$。因为：

$$\cos\theta'=l_1'l_2'+m_1'm_2'+n_1'n_2' \tag{1.55}$$

$$l_1'=\frac{\mathrm{d}x_1'}{\mathrm{d}r_1'},l_2'=\frac{\mathrm{d}x_2'}{\mathrm{d}r_2'},m_1'=\frac{\mathrm{d}y_1'}{\mathrm{d}r_1'},m_2'=\frac{\mathrm{d}y_2'}{\mathrm{d}r_2'},n_1'=\frac{\mathrm{d}z_1'}{\mathrm{d}r_1'},n_2'=\frac{\mathrm{d}z_2'}{\mathrm{d}r_2'} \tag{1.56}$$

所以

$$\cos\theta'=\frac{\mathrm{d}x_1'\mathrm{d}x_2'+\mathrm{d}y_1'\mathrm{d}y_2'+\mathrm{d}z_1'\mathrm{d}z_2'}{\mathrm{d}r_1'\mathrm{d}r_2'} \tag{1.57}$$

因为：

$$\mathrm{d}x_1'=\mathrm{d}x_1+\mathrm{d}u_1,\mathrm{d}x_2'=\mathrm{d}x_2+\mathrm{d}u_2,\mathrm{d}y_1'=\mathrm{d}y_1+\mathrm{d}v_1,\mathrm{d}y_2'=\mathrm{d}y_2+\mathrm{d}v_2,\mathrm{d}z_1'=\mathrm{d}z_1+\mathrm{d}w_1,\mathrm{d}z_2'=\mathrm{d}z_2+\mathrm{d}w_2 \tag{1.58}$$

所以

$$\mathrm{d}x_1'\mathrm{d}x_2'=(\mathrm{d}x_1+\mathrm{d}u_1)(\mathrm{d}x_2+\mathrm{d}u_2)=\mathrm{d}x_1\mathrm{d}x_2+\mathrm{d}u_1\mathrm{d}x_2+\mathrm{d}x_1\mathrm{d}u_2+\mathrm{d}u_1\mathrm{d}u_2 \tag{1.59}$$

忽略高阶位移增量 $\mathrm{d}u_1\mathrm{d}u_2$，并且 $\mathrm{d}r_1'=(1+\varepsilon_{nA})\mathrm{d}r_1$， $\mathrm{d}r_2'=(1+\varepsilon_{nB})\mathrm{d}r_2$

$$\begin{aligned}\cos\theta'&=\frac{\mathrm{d}x_1'\mathrm{d}x_2'+\mathrm{d}y_1'\mathrm{d}y_2'+\mathrm{d}z_1'\mathrm{d}z_2'}{\mathrm{d}r_1'\mathrm{d}r_2'}\\&=\frac{\cos\theta}{(1+\varepsilon_{nA})(1+\varepsilon_{nB})}+\frac{\mathrm{d}u_1\mathrm{d}x_2+\mathrm{d}x_1\mathrm{d}u_2+\mathrm{d}v_1\mathrm{d}y_2+\mathrm{d}y_1\mathrm{d}v_2+\mathrm{d}w_1\mathrm{d}z_2+\mathrm{d}z_1\mathrm{d}w_2}{(1+\varepsilon_{nA})(1+\varepsilon_{nB})\mathrm{d}r_1\mathrm{d}r_2}\end{aligned} \tag{1.60}$$

因为

$$\frac{\cos\theta}{(1+\varepsilon_{nA})(1+\varepsilon_{nB})}=\frac{(1-\varepsilon_{nA})(1-\varepsilon_{nB})\cos\theta}{(1-\varepsilon_{nA}^2)(1-\varepsilon_{nB}^2)}=\frac{1-\varepsilon_{nA}-\varepsilon_{nB}+\varepsilon_{nA}\varepsilon_{nB}}{1-\varepsilon_{nA}^2-\varepsilon_{nB}^2+\varepsilon_{nA}^2\varepsilon_{nB}^2}\cos\theta \tag{1.61}$$

忽略 2 次以上的应变项

$$\frac{\cos\theta}{(1+\varepsilon_{nA})(1+\varepsilon_{nB})}=(1-\varepsilon_{nA}-\varepsilon_{nB})\cos\theta \tag{1.62}$$

又因为：

$$\frac{\mathrm{d}u_1\mathrm{d}x_2}{\mathrm{d}r_1\mathrm{d}r_2}=\frac{\mathrm{d}u_1}{\mathrm{d}r_1}l_2=\left(\frac{\partial u_1}{\partial x}\frac{\partial x}{\partial r_1}+\frac{\partial u_1}{\partial y}\frac{\partial y}{\partial r_1}+\frac{\partial u_1}{\partial z}\frac{\partial z}{\partial r_1}\right)l_2 \tag{1.63a}$$

$$\frac{\mathrm{d}u_1\mathrm{d}x_2}{\mathrm{d}r_1\mathrm{d}r_2}=\frac{\partial u_1}{\partial x}l_1l_2+\frac{\partial u_1}{\partial y}m_1l_2+\frac{\partial u_1}{\partial z}n_1l_2 \tag{1.63b}$$

$$\frac{\mathrm{d}x_1\mathrm{d}u_2}{\mathrm{d}r_1\mathrm{d}r_2}=\frac{\partial u_2}{\partial x}l_1l_2+\frac{\partial u_2}{\partial y}m_2l_1+\frac{\partial u_2}{\partial z}n_2l_1 \tag{1.63c}$$

$$\frac{\mathrm{d}v_1\mathrm{d}y_2}{\mathrm{d}r_1\mathrm{d}r_2}=\frac{\partial v_1}{\partial x}m_2l_1+\frac{\partial v_1}{\partial y}m_2m_1+\frac{\partial v_1}{\partial z}m_2n_1 \tag{1.63d}$$

$$\frac{\mathrm{d}y_1\mathrm{d}v_2}{\mathrm{d}r_1\mathrm{d}r_2}=\frac{\partial v_2}{\partial x}m_1l_2+\frac{\partial v_2}{\partial y}m_1m_2+\frac{\partial v_2}{\partial z}m_1n_2 \tag{1.63e}$$

$$\frac{\mathrm{d}w_1\mathrm{d}z_2}{\mathrm{d}r_1\mathrm{d}r_2}=\frac{\partial w_1}{\partial x}n_2l_1+\frac{\partial w_1}{\partial y}n_2m_1+\frac{\partial w_1}{\partial z}n_2n_1 \tag{1.63f}$$

$$\frac{\mathrm{d}z_1\mathrm{d}w_2}{\mathrm{d}r_1\mathrm{d}r_2}=\frac{\partial w_2}{\partial x}n_1l_2+\frac{\partial w_2}{\partial y}n_1m_2+\frac{\partial w_2}{\partial z}n_1n_2 \tag{1.63g}$$

因为近似的

$$\frac{\partial u_1}{\partial x}=\frac{\partial u_2}{\partial x}=\frac{\partial u}{\partial x},\quad \frac{\partial v_1}{\partial x}=\frac{\partial v_2}{\partial x}=\frac{\partial v}{\partial x},\quad \frac{\partial w_1}{\partial x}=\frac{\partial w_2}{\partial x}=\frac{\partial w}{\partial x} \tag{1.64a}$$

$$\frac{\partial u_1}{\partial y}=\frac{\partial u_2}{\partial y}=\frac{\partial u}{\partial y},\quad \frac{\partial v_1}{\partial y}=\frac{\partial v_2}{\partial y}=\frac{\partial v}{\partial y},\quad \frac{\partial w_1}{\partial y}=\frac{\partial w_2}{\partial y}=\frac{\partial w}{\partial y} \tag{1.64b}$$

$$\frac{\partial u_1}{\partial z}=\frac{\partial u_2}{\partial z}=\frac{\partial u}{\partial z},\quad \frac{\partial v_1}{\partial z}=\frac{\partial v_2}{\partial z}=\frac{\partial v}{\partial z},\quad \frac{\partial w_1}{\partial z}=\frac{\partial w_2}{\partial z}=\frac{\partial w}{\partial z} \tag{1.64c}$$

所以

$$\cos\theta'=(1-\varepsilon_{nA}-\varepsilon_{nB})\left[\begin{array}{c}\cos\theta+2\left(\dfrac{\partial u}{\partial x}l_1l_2+\dfrac{\partial v}{\partial y}m_1m_2+\dfrac{\partial w}{\partial z}n_1n_2\right)\\+\left(\dfrac{\partial w}{\partial y}+\dfrac{\partial v}{\partial z}\right)(m_1n_2+n_1m_2)\\+\left(\dfrac{\partial u}{\partial z}+\dfrac{\partial w}{\partial x}\right)(n_1l_2+l_1n_2)\\+\left(\dfrac{\partial v}{\partial x}+\dfrac{\partial u}{\partial y}\right)(l_1m_2+m_1l_2)\end{array}\right] \tag{1.65}$$

综合方程(1.54)和(1.65)不难看出，以下六个量决定了空间一点的线应变和角应变：

$$\frac{\partial u}{\partial x},\frac{\partial v}{\partial y},\frac{\partial w}{\partial z},\left(\frac{\partial w}{\partial y}+\frac{\partial v}{\partial z}\right),\ \left(\frac{\partial u}{\partial z}+\frac{\partial w}{\partial x}\right),\ \left(\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}\right)$$

所以，我们定义应变 ε_x 、 ε_y 、 ε_z 为一点的正应变，γ_{xy} 、 γ_{xz} 、 γ_{yz} 为一点的剪切应变。表达式为：

$$\varepsilon_x=\frac{\partial u}{\partial x},\varepsilon_y=\frac{\partial v}{\partial y},\varepsilon_z=\frac{\partial w}{\partial z},\ \gamma_{xy}=\left(\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}\right),\ \ \gamma_{xz}=\left(\frac{\partial u}{\partial z}+\frac{\partial w}{\partial x}\right),\ \ \gamma_{yz}=\left(\frac{\partial w}{\partial y}+\frac{\partial v}{\partial z}\right) \tag{1.66}$$

方程(1.66)描述了位移与应变之间的关系，称为几何方程。对于平面问题，方程(1.66)简化为：

$$\varepsilon_x=\frac{\partial u}{\partial x},\varepsilon_y=\frac{\partial v}{\partial y},\ \gamma_{xy}=\left(\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}\right) \tag{1.67}$$

1.4.3 平衡方程

当物体受力平衡后，物体内部不同点的应力之间需要满足平衡方程。三维问题的平衡方程为：

$$\begin{aligned}&\frac{\partial\sigma_{xx}}{\partial x}+\frac{\partial\tau_{xy}}{\partial y}+\frac{\partial\tau_{xz}}{\partial z}+f_x=0\\&\frac{\partial\tau_{xy}}{\partial x}+\frac{\partial\sigma_{yy}}{\partial y}+\frac{\partial\tau_{yz}}{\partial z}+f_y=0\\&\frac{\partial\tau_{xz}}{\partial x}+\frac{\partial\tau_{yz}}{\partial y}+\frac{\partial\sigma_{zz}}{\partial z}+f_z=0\end{aligned} \tag{1.68}$$

其中 f_x 、 f_y 和 f_z 分别是沿 x、y、z 方向的体积力。

二维问题的平衡方程缩减为：

$$\begin{aligned}&\frac{\partial\sigma_{xx}}{\partial x}+\frac{\partial\tau_{xy}}{\partial y}+f_x=0\\&\frac{\partial\tau_{xy}}{\partial x}+\frac{\partial\sigma_{yy}}{\partial y}+f_y=0\end{aligned} \tag{1.69}$$

1.4.4 弹性物理方程

应力和应变之间还需要满足物理方程。对于各向同性材料，在弹性范围内，应力与应变之间还需要满足虎克定律：

$$\begin{aligned}&\varepsilon_x=\frac{1}{E}\left[\sigma_x-\nu\left(\sigma_y+\sigma_z\right)\right]\quad \gamma_{yz}=\frac{2\left(1+\nu\right)}{E}\tau_{yz}=\frac{1}{G}\tau_{yz}\\&\varepsilon_y=\frac{1}{E}\left[\sigma_y-\nu\left(\sigma_z+\sigma_x\right)\right],\ \gamma_{zx}=\frac{2\left(1+\nu\right)}{E}\tau_{zx}=\frac{1}{G}\tau_{zx}\\&\varepsilon_z=\frac{1}{E}\left[\sigma_z-\nu\left(\sigma_x+\sigma_y\right)\right]\quad \gamma_{xy}=\frac{2\left(1+\nu\right)}{E}\tau_{xy}=\frac{1}{G}\tau_{xy}\end{aligned} \tag{1.70}$$

其中 E、G 和 ν 分别是材料的杨氏模量、剪切模量和泊松比。三者之间满足如下关系式：

$$G=\frac{E}{2\left(1+\nu\right)} \tag{1.71}$$

对于平面应力问题，物理方程缩减为

$$\varepsilon_x=\frac{1}{E}\left(\sigma_x-\nu\sigma_y\right),\varepsilon_y=\frac{1}{E}\left(\sigma_y-\nu\sigma_x\right),\gamma_{xy}=\frac{2\left(1+\nu\right)}{E}\tau_{xy} \tag{1.72}$$

对于平面应变问题，物理方程缩减为：

$$\varepsilon_x=\frac{1-\nu^2}{E}\left(\sigma_x-\frac{\nu}{1-\nu}\sigma_y\right),\varepsilon_y=\frac{1-\nu^2}{E}\left(\sigma_y-\frac{\nu}{1-\nu}\sigma_x\right),\gamma_{xy}=\frac{2\left(1+\nu\right)}{E}\tau_{xy} \tag{1.73}$$

1.4.5 边界条件

平衡方程、几何方程和物理方程构成了弹性力学的三大基本方程。基本方程描述了位移、应变和应力在物体内部的传递规律。为了获得具体的表达式，还需要边界条件。弹性力学中的边界条件主要有位移边界条件和应力边界条件。我们用 Γ_1 表示位移边界条件作用的区域，Γ_2 表示应力边界条件作用的区域。那么应该满足如下边界条件方程：

$$u=\bar{u},\quad v=\bar{v},\quad w=\bar{w}\quad \text{在}\Gamma_1\text{上} \tag{1.74}$$

$$\begin{aligned}&\sigma_x l+\tau_{yx}m+\tau_{zx}n=q_x\\&\tau_{xy}l+\sigma_y m+\tau_{zy}n=q_y\\&\tau_{xz}l+\tau_{yz}m+\sigma_z n=q_z\end{aligned}\quad \text{在}\Gamma_2\text{上} \tag{1.75}$$

对于二维问题，边界条件简化为：

$$u=\bar{u},\quad v=\bar{v}\quad \text{在}\Gamma_1\text{上} \tag{1.76}$$

$$\begin{aligned}&\sigma_x l+\tau_{yx}m=q_x\\&\tau_{xy}l+\sigma_y m=q_y\end{aligned}\quad \text{在}\Gamma_2\text{上} \tag{1.77}$$

1.5 基本方程的张量表示法

弹性力学各类基本方程有许多求和项，为了简化书写复杂度，可以采用张量表示法。例如一个位移向量可以写成：

$$\mathbf{u}=u_1\mathbf{e}_1+u_2\mathbf{e}_2+u_3\mathbf{e}_3 \tag{1.78}$$

其中 $\mathbf{e}_1$、$\mathbf{e}_2$、$\mathbf{e}_3$ 分别是坐标系矢量的单位矢量，u_1、u_2、u_3 分别是位移矢量 $\mathbf{u}$ 在三个坐标上的分量。用求和号，可将(1.78)式写成：

$$\mathbf{u}=\sum_{i=1}^{3}u_i\mathbf{e}_i \tag{1.79}$$

所谓 Einstein 约定求和就是略去求和式中的求和号，例如(1.79)式即可写成

$$\mathbf{u}=u_i\mathbf{e}_i \tag{1.80}$$

在此规则中有相同的下标就表示求和，而不管下标是什么字母，例如(1.80)式也可写成

$$\mathbf{u}=u_i\mathbf{e}_i=u_j\mathbf{e}_j=u_k\mathbf{e}_k \tag{1.81}$$

有时亦称求和的下标为“哑指标”。本书以后如无不同的说明，相同的英文下标总表示 1 至 3 求和。采用这个标记法，几何方程(1.66)可以写为

$$\varepsilon_{ij}=\frac{1}{2}\left(u_{i,j}+u_{j,i}\right) \tag{1.82}$$

其中 $u_{i,j}=\dfrac{\partial u_i}{\partial x_j}$。

需要注意的是，上式表达的剪应变 ε_{12}， ε_{13}， ε_{23} 是(1.66)中对应的剪应变 γ_{12}， γ_{13}， γ_{23} 的二分之一。一般称 γ_{12}， γ_{13}， γ_{23} 为工程剪应变。

平衡方程可以表示为：

$$\sigma_{ij,j}+f_i=0 \tag{1.83}$$

物理方程可以表示为：

$$\sigma_{ij}=D_{ijkl}\varepsilon_{kl} \tag{1.84}$$

边界条件可以表示为：

$$p_i=\sigma_{ij}n_j,\quad u_i=\overline{u}_i \tag{1.85}$$

1.6 矢量与张量基础

1.6.1 什么是矢量

从几何观点来看，矢量（一般用小写字母黑斜体表示）定义为有向线段。三维欧式空间 E^3 中，建立直角坐标系 $\{O;x_1,x_2,x_3\}$，沿坐标 x_i 方向的单位矢量为， $\mathbf{e}_i(i=1,2,3)$ 即其标架为 $\{\mathbf{e}_1,\mathbf{e}_2,\mathbf{e}_3\}$。设从坐标原点 O 至点 A 的矢量为 $\mathbf{a}$，它在所述坐标系中的坐标为 (a_1,a_2,a_3)，那么 $\mathbf{a}$ 可以写成

$$\mathbf{a}=a_1\mathbf{e}_1+a_2\mathbf{e}_2+a_3\mathbf{e}_3 \tag{1.86}$$

设在 E^3 中另有一个坐标系 $\{O;x_1,x_2,x_3\}$，其标架为 $\{\mathbf{e}_1',\mathbf{e}_2',\mathbf{e}_3'\}$，它与 $\{\mathbf{e}_1,\mathbf{e}_2,\mathbf{e}_3\}$ 之间的关系为

$$\begin{cases}\mathbf{e}_1'=C_{11}\mathbf{e}_1+C_{12}\mathbf{e}_2+C_{13}\mathbf{e}_3\\ \mathbf{e}_2'=C_{21}\mathbf{e}_1+C_{22}\mathbf{e}_2+C_{23}\mathbf{e}_3\\ \mathbf{e}_3'=C_{31}\mathbf{e}_1+C_{32}\mathbf{e}_2+C_{33}\mathbf{e}_3\end{cases} \tag{1.87}$$

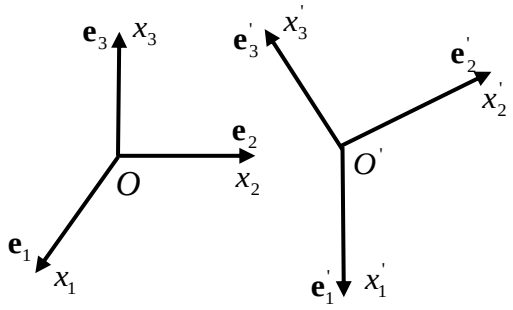


图 1.10 三维欧式空间 E^3 直角坐标系 $\{O;x_1,x_2,x_3\}$ 和坐标系 $\{O';x_1,x_2,x_3\}$

由于单位矢量 $\mathbf{e}_i(i=1,2,3)$ 之间互相正交， $\mathbf{e}_i'(i=1,2,3)$ 之间也互相正交，因此矩阵

$$C=\begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \tag{1.88}$$

是正交矩阵，有 $C^{-1}=C^T$，上标“T”表示转置。从(1.87)式可反解出

$$\begin{cases} \mathbf{e}_1=C_{11}\mathbf{e}_1'+C_{12}\mathbf{e}_2'+C_{13}\mathbf{e}_3' \\ \mathbf{e}_2=C_{21}\mathbf{e}_1'+C_{22}\mathbf{e}_2'+C_{23}\mathbf{e}_3' \\ \mathbf{e}_3=C_{31}\mathbf{e}_1'+C_{32}\mathbf{e}_2'+C_{33}\mathbf{e}_3' \end{cases} \tag{1.89}$$

矢量 $\mathbf{a}$ 在新坐标系 $\{O;x_1',x_2',x_3'\}$ 中的分解记为

$$\mathbf{a}=a_1\mathbf{e}_1'+a_2\mathbf{e}_2'+a_3\mathbf{e}_3' \tag{1.90}$$

将(1.89)式代入(1.86)式，得到

$$\begin{cases} \mathbf{a}_1'=C_{11}\mathbf{a_1}+C_{12}\mathbf{a_2}+C_{13}\mathbf{a_3} \\ \mathbf{a}_2'=C_{21}\mathbf{a_1}+C_{22}\mathbf{a_2}+C_{23}\mathbf{a_3} \\ \mathbf{a}_3'=C_{31}\mathbf{a_1}+C_{32}\mathbf{a_2}+C_{33}\mathbf{a_3} \end{cases} \tag{1.91}$$

公式(1.91)是矢量 $\mathbf{a}$ 的新坐标 $\mathbf{a}_i'(i=1,2,3)$ 和旧坐标 $\mathbf{a}_i(i=1,2,3)$ 之间的关系，它是坐标变换系数的一次齐次式。这个式子应该是有向线段的几何客观性质（如：长度、角度）不随坐标的人为主观选取而变化的一种代数反映。可以说，公式(1.91)表示了矢量在坐标变换下的不变性。

这样，我们就从矢量的几何定义，得到了矢量的代数定义：一个有序数组 (a_1,a_2,a_3)，如果在坐标变换下，关于变换系数 $C_{ij}(i,j=1,2,3)$ 为由(1.91)式所示的一次齐次式，则称之为矢量。

用求和号，可将（1.86）式写成

$$\mathbf{a}=\sum_{i=1}^{3}a_i e_i \tag{1.92}$$

所谓 Einstein 约定求和就是略去求和式中的求和号，例如(1.92)式即可写成

$$\mathbf{a}=a_i\mathbf{e}_i \tag{1.93}$$

在此规则中有相同的下标就表示求和，而不管下标是什么字母，例如(1.93)式也可写成

$$\mathbf{a}=a_i\mathbf{e}_i=a_j\mathbf{e}_j=a_k\mathbf{e}_k \tag{1.94}$$

有时亦称求和的下标为“哑指标”。本书以后如无不同的说明，相同的英文下标总表示 1 至 3 求和。

按约定求和规则，(1.87)与(1.89)式可写式

$$\mathbf{e}_i^{'}=C_{ij}\mathbf{e}_j \tag{1.95}$$

$$\mathbf{e}_i=C_{ij}\mathbf{e}_j^{'} \tag{1.96}$$

将(1.96)式代入(1.93)式，得

$$\mathbf{a}=a_iC_{ij}\mathbf{e}_j^{'}=a_jC_{ij}\mathbf{e}_i^{'}=a_i^{'}\mathbf{e}_i^{'} \tag{1.97}$$

由此就得到了(1.91)式的约定求和写法

$$\mathbf{a}_i^{'}=C_{ij}\mathbf{a}_j\quad(i=1,2,3) \tag{1.98}$$

今引入 Kronecker 记号 δ_{ij}：

$$\delta_{ij}=\begin{cases}1 & i=j\\ 0 & i\neq j\end{cases}\quad(i,j=1,2,3) \tag{1.99}$$

例如 $\delta_{11}=1$，$\delta_{12}=0$，$\cdots$应用 δ_{ij}，单位矢量之间的内积可写成

$$\mathbf{e}_i\cdot\mathbf{e}_j=\delta_{ij} \tag{1.100}$$

矢量 $\mathbf{a}=a_i\mathbf{e}_i$ 与矢量 $\mathbf{b}=b_j\mathbf{e}_j$ 之间的内积可写成

$$\mathbf{a}\cdot\mathbf{b}=a_i\mathbf{e}_i\cdot b_j\mathbf{e}_j=a_ib_j\mathbf{e}_i\cdot\mathbf{e}_j=a_ib_j\delta_{ij}=a_ib_i \tag{1.101}$$

上式中最后一个等号是因为仅当 $i=j$ 时，δ_{ij} 才不等于零，故 δ_{ij} 的作用似乎是将 j 换成了 i，因而也称 δ_{ij} 为“换标记号”。

再引入 Levi-Civita 记号 ε_{ijk}：

$$\varepsilon_{ijk}=\begin{cases}1 & \text{当}i,j,k\text{为偶排列}\\ -1 & \text{当}i,j,k\text{为奇排列}\\ 0 & \text{当}i,j,k\text{中有相同者}\end{cases} \tag{1.102}$$

其中 i,j,k 分别取 1，2，3 中的某一个值。例如:

$\varepsilon_{123}=\varepsilon_{231}=\varepsilon_{312}=1$， $\varepsilon_{132}=\varepsilon_{321}=\varepsilon_{213}=-1$， $\varepsilon_{112}=\varepsilon_{333}=0$， ….

利用 ε_{ijk} ，矢量之间的外积可写为

$$\mathbf{e}_i\times\mathbf{e}_j=\varepsilon_{ijk}\mathbf{e}_k \tag{1.103}$$

$$\mathbf{a}\times\mathbf{b}=a_i\mathbf{e}_i\times b_j\mathbf{e}_j=a_ib_j\varepsilon_{ijk}\mathbf{e}_\mathbf{k} \tag{1.104}$$

$$\varepsilon_{pij}\varepsilon_{pks}=\delta_{ik}\delta_{js}-\delta_{is}\delta_{jk} \tag{1.105}$$

证明：穷举法，先列出 i,j,k,s 所有可能的 81 种取值情况，如下表所示。然后对逐个情形证明，例如，情形 1

$$\varepsilon_{p11}\varepsilon_{p11}=\delta_{11}\delta_{11}-\delta_{11}\delta_{11}=0 \tag{1.106}$$

其他情形略。

矢量加法：

$$\begin{aligned}&\mathbf{a}=a_x\mathbf{e}_\mathbf{x}+a_y\mathbf{e}_\mathbf{y}+a_z\mathbf{e}_\mathbf{z}\\&\mathbf{b}=b_x\mathbf{e}_\mathbf{x}+b_y\mathbf{e}_\mathbf{y}+b_z\mathbf{e}_\mathbf{z}\\&\mathbf{c}=c_x\mathbf{e}_\mathbf{x}+c_y\mathbf{e}_\mathbf{y}+c_z\mathbf{e}_\mathbf{z}\\&\mathbf{c}=\mathbf{a}+\mathbf{b}=(a_x+b_x)\mathbf{e}_x+(a_y+b_y)\mathbf{e}_y+(a_z+b_z)\mathbf{e}_z\end{aligned} \tag{1.107}$$

矢量点乘：

设二维空间内有两个向量 $\vec{a}=(x_1,y_1)$ 和 $\vec{b}=(x_2,y_2)$ ，定义它们的数量积（又叫内积、点积）为以下实数：

$$\vec{a}\cdot\vec{b}=x_1x_2+y_1y_2 \tag{1.108}$$

更一般地， n 维向量的内积定义如下：

$$\mathbf{a}\cdot\mathbf{b}=\sum_{i=1}^{n}a_ib_i=a_1b_1+a_2b_2+\cdots+a_nb_n \tag{1.109}$$

矢量叉乘:

表示方法

两个向量$\mathbf{a}$和$\mathbf{b}$的叉积写作 $\mathbf{a}\times\mathbf{b}$（有时也被写成 $\mathbf{a}\Lambda\mathbf{b}$ ，避免和字母 x 混淆）。

定义

向量积可以被定义为：$\mathbf{a}\times\mathbf{b}=ab\cos\theta$

模长：（在这里θ表示两向量之间的夹角（共起点的前提下）（$0^\circ\le\theta\le180^\circ$），它位于这两个矢量所定义的平面上。）

$$\left|\vec{a}\times\vec{b}\right|=\left|\vec{a}\right|\cdot\left|\vec{b}\right|\cdot\sin\theta \tag{1.110}$$

方向：**a** 向量和b向量的向量积的方向与这两个向量所在平面垂直，且遵守右手定则。（一个简单的确定满足“右手定则”的结果向量的方向的方法是这样的：若坐标系是满足右手定则的，当右手的四指 **a** 以不超过 180 度的转角转向b时，竖起的大拇指指向是 **c** 的方向。）

也可以这样定义（等效）：

向量积$|\mathbf{c}|=|\mathbf{a}\times\mathbf{b}|=|\mathbf{a}||\mathbf{b}|\sin\langle\mathbf{a},\mathbf{b}\rangle$

即 **c** 的长度在数值上等于以 **a**，b，夹角为θ组成的平行四边形的面积。

而 **c** 的方向垂直于a与b所决定的平面，**c** 的指向按右手定则从 **a** 转向b来确定。

*运算结果 **c** 是一个伪向量。这是因为在不同的坐标系中 **c** 可能不同。

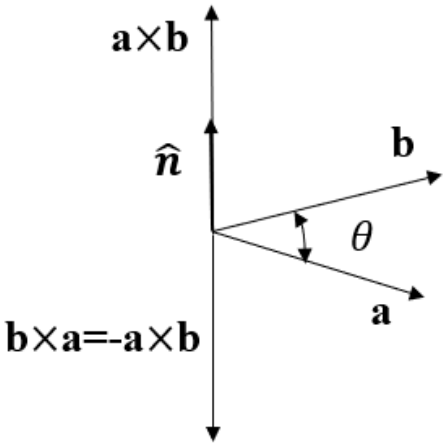


图 1. 11 矢量叉乘示意图

坐标运算

设$\vec{\mathrm{a}}=(\mathrm{a_x},\mathrm{a_y},\mathrm{a_z}),\vec{\mathrm{b}}=(\mathrm{b_x},\mathrm{b_y},\mathrm{b_z})$。分别是 x, y, z 轴方向的单位向量，则：

$$\mathbf{a}\times\mathbf{b}=(a_yb_z-a_zb_y)\mathbf{i}+(a_zb_x-a_xb_z)\mathbf{j}+(a_xb_y-a_yb_x)\mathbf{k} \tag{1.111}$$

为了帮助记忆，利用三阶行列式，写成

$$\det\begin{vmatrix} i & j & k \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix} \tag{1.112}$$

$$\mathbf{a}\times\mathbf{b}=(l,m,n)\times(o,p,q)=(mq-np,no-lq,lp-mo) \tag{1.113}$$

$$\mathbf{a}\times\mathbf{b}=a_i b_j \varepsilon_{ijk}\mathbf{e}_k \tag{1.114}$$

1.6.2 什么是张量

设

$$A=A_{ij}e_i e_j \tag{1.115}$$

其中$e_i e_j$称为并矢基，它们共有 9 个，

$$e_1e_1,e_1e_2,e_1e_3,e_2e_1,e_2e_2,e_2e_3,e_3e_1,e_3e_2,e_3e_3 \tag{1.116}$$

在坐标变换(1.96)下，(1.115)式变为

$$A=A_{ij}C_{ki}C_{sj}e_k^{'}e_s^{'} \tag{1.117}$$

于是

$$A=C_{ki}C_{sj}A_{ks} \tag{1.118}$$

从(1.118)式可引出张量的定义：一个二阶有序数组$A_{ij}\ (i,j=1,2,3)$，在坐标变换下，关于变换系数C_{ij}为二次齐次式，则称$\boldsymbol{A}_{ij}$为张量（用大写黑斜体表示），也记作$\boldsymbol{A}$。$\boldsymbol{A}_{ij}$为其指标记号，$\boldsymbol{A}$为其整体记号。

张量$\boldsymbol{A}$在并矢基$e_i e_j$下的 9 个分量，有矩阵$\mathbf{A}$与之对应，记作

$$\boldsymbol{A}\leftrightarrow\mathbf{A}=\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \tag{1.119}$$

同一个张量$\boldsymbol{A}$在另一组并矢基下$e_i^{'}e_j^{'}$所对应的矩阵为$\mathbf{A}^{'}$

$$\boldsymbol{A}\leftrightarrow\mathbf{A}^{'}=\begin{bmatrix} A_{11}^{'} & A_{12}^{'} & A_{13}^{'} \\ A_{21}^{'} & A_{22}^{'} & A_{32}^{'} \\ A_{31}^{'} & A_{32}^{'} & A_{33}^{'} \end{bmatrix} \tag{1.120}$$

按(1.95)式可知，张量在不同坐标系下所对应的矩阵服从矩阵的合同变换，即

$$\mathbf{A}^{'}=\mathbf{CAC}^{\mathbf{T}} \tag{1.121}$$

其中 **C** 为坐标变换矩阵（1.117）.可逆。

附注 上述张量的定义可以推广为：一个 **r** 阶有序数组 $A_{j1,j2,\cdots,j_r}$，在坐标变换(1.95)下，若服从 C_{ij} 的 r 次齐次式

$$A'_{j1,j2,\cdots,j_r}=C_{i_1j_1}C_{i_2j_2}\cdots C_{i_rj_r}A_{j1,j2,\cdots,j_r} \tag{1.122}$$

则称之为 r 阶张量。按照这种定义，标量可认为是零阶张量，矢量可认为是一阶张量，(1.114)式所述的张量为二阶张量，也可证明 Levi-Civita 记号 ε_{ijk} 为三阶张量。(1.122)式中的下标 i_k 和 j_k $(k=1,2,\cdots,r)$ 的取值范围也可不必限于从 1 到 3，而可从 1 到 n，那么(1.122)式所定义的张量成为 n 维空间中的 r 阶张量。本书所述张量，以后不作说明均为三维二阶张量。

张量的运算

张量 $\boldsymbol{A}=A_{ij}\mathbf{e}_i\mathbf{e}_j$ 与张量 $\boldsymbol{B}=B_{ij}\mathbf{e}_i\mathbf{e}_j$ 的和与差记为 $\boldsymbol{A}\pm\boldsymbol{B}$，

$$\boldsymbol{A}\pm\boldsymbol{B}=(A_{ij}\pm B_{ij})\mathbf{e}_i\mathbf{e}_j \tag{1.123}$$

张量 $\boldsymbol{A}$ 的转置记为 $\boldsymbol{A}^T$，

$$\boldsymbol{A}^T=A_{ij}\mathbf{e}_i\mathbf{e}_j \tag{1.124}$$

不难验证，$\boldsymbol{A}\pm\boldsymbol{B}$ 和 $\boldsymbol{A}^T$ 也是张量。例如

$$(A^T_{ij})'=A'_{ij}=C_{ji}C_{is}A_{ks}=C_{is}C_{jk}A^T_{ij} \tag{1.125}$$

一个张量 $\boldsymbol{A}$ 称为对称张量，如果

$$\boldsymbol{A}^T=\boldsymbol{A} \tag{1.126}$$

与对称张量 $\boldsymbol{A}$ 所对应的矩阵 **A** 为对称矩阵。

一个张量 $\boldsymbol{A}$ 称为反对称张量，如果

$$\boldsymbol{A}^T=-\boldsymbol{A} \tag{1.127}$$

与反对称张量 $\boldsymbol{A}$ 所对应的矩阵 **A** 为反对称矩阵。

我们将反对称矩阵 **A** 记为

$$\mathbf{A}=\begin{bmatrix}0 & -\omega_3 & \omega_2\\ \omega_3 & 0 & -\omega_1\\ -\omega_2 & \omega_1 & 0\end{bmatrix} \tag{1.128}$$

从(1.109)式可以得出，

$$A_{ij} = -\varepsilon_{ijk}\omega_k \tag{1.129}$$

$$\omega_i = -\frac{1}{2}\varepsilon_{ijk}A_{jk} \tag{1.130}$$

不难验证，由(1.130)式所定义的$\boldsymbol{\omega} = \omega_i \mathbf{e}_i$为矢量，它称为相应于反对称张量$\boldsymbol{A}$的轴矢量。

由于$\delta'_{ij} = \mathbf{e}'_i \cdot \mathbf{e}'_j = C_{ik}C_{js}\mathbf{e}_k \cdot \mathbf{e}_s = C_{ik}C_{js}\delta_{ks}$，所以

$$\mathbf{I} = \delta_{ij}\mathbf{e}_i\mathbf{e}_j = \mathbf{e}_i\mathbf{e}_j \tag{1.131}$$

为一张量，称之为单位张量。

张量$\boldsymbol{A}$的迹定义为

$$J(A) = A_{ii} \qquad \text{（求和）} \tag{1.132}$$

张量与矢量的运算

张量$\boldsymbol{A} = A_{ij}\mathbf{e}_i\mathbf{e}_j$与矢量$\mathbf{a} = a_i\mathbf{e}_i$有下列左右两种内积：

$$\boldsymbol{A}\cdot\mathbf{a} = A_{ij}\mathbf{e}_i\mathbf{e}_j \cdot a_k\mathbf{e}_k = A_{ij}a_k\mathbf{e}_i(\mathbf{e}_j \cdot \mathbf{e}_k) = A_{ij}a_j\mathbf{e}_i \tag{1.133}$$

$$\mathbf{a}\cdot\boldsymbol{A} = a_i\mathbf{e}_i \cdot A_{jk}\mathbf{e}_j\mathbf{e}_k = a_iA_{jk}(\mathbf{e}_i \cdot \mathbf{e}_j)\mathbf{e}_k = a_iA_{jk}\mathbf{e}_k \tag{1.134}$$

从上述两式可得左右两种内积之间的关系式

$$\mathbf{a}\cdot\boldsymbol{A} = \boldsymbol{A}^T\cdot\mathbf{a} \tag{1.135}$$

如果$\boldsymbol{A}$称为反对称张量，由（1.129）和（1.133）式，得

$$\boldsymbol{A}\cdot\mathbf{a} = -\varepsilon_{ijk}\omega_k a_j\mathbf{e}_i = \boldsymbol{\omega}\times\mathbf{a} \tag{1.136}$$

张量$\boldsymbol{A} = A_{ij}\mathbf{e}_i\mathbf{e}_j$与矢量$\mathbf{a} = a_i\mathbf{e}_i$有下列左右两种外积：

$$\boldsymbol{A}\times\mathbf{a} = A_{ij}\mathbf{e}_i\mathbf{e}_j \times a_k\mathbf{e}_k = A_{ij}a_k\varepsilon_{jks}\mathbf{e}_i\mathbf{e}_s \tag{1.137}$$

$$\mathbf{a}\times\boldsymbol{A} = a_i\mathbf{e}_i \times A_{jk}\mathbf{e}_j\mathbf{e}_k = a_iA_{jk}\varepsilon_{ijt}\mathbf{e}_t\mathbf{e}_k \tag{1.138}$$

张量$\boldsymbol{A}$与两个矢量$\boldsymbol{a}$和$\boldsymbol{b}$之间有下列4种运算：

（1）$\mathbf{a}\cdot\boldsymbol{A}\cdot\mathbf{b} = a_i\mathbf{e}_i \cdot A_{jk}\mathbf{e}_j\mathbf{e}_k \cdot b_s\mathbf{e}_s = a_iA_{jk}b_s(\mathbf{e}_i \cdot \mathbf{e}_j)(\mathbf{e}_k \cdot \mathbf{e}_s) = a_iA_{jk}b_s$

（2）$\mathbf{a}\cdot\boldsymbol{A}\times\mathbf{b} = a_iA_{jk}b_s\varepsilon_{ksp}\mathbf{e}_p$

（3）$\mathbf{a}\times\boldsymbol{A}\cdot\mathbf{b} = a_iA_{jk}b_s\varepsilon_{ijp}\mathbf{e}_p$

（4） $\mathbf{a}\times\boldsymbol{A}\times\mathbf{b}=a_iA_{jk}b_s\varepsilon_{ijp}\varepsilon_{ksq}\mathbf{e}_p\mathbf{e}_q$

证明 $du=\omega\times dr+\Gamma\cdot dr$

对称张量： $\boldsymbol{\Gamma}=\frac{1}{2}(\mathbf{u}\nabla+\nabla\mathbf{u})$，反对称张量： $\Omega=\frac{1}{2}(\mathbf{u}\nabla-\nabla\mathbf{u})$，轴矢量： $\boldsymbol{\omega}=\frac{1}{2}(\nabla\times\mathbf{u})$

因为 $\mathbf{du}=\boldsymbol{\Omega}\cdot\mathbf{dr}+\boldsymbol{\Gamma}\cdot\mathbf{dr}$，只要证明 $\boldsymbol{\omega}\times\mathbf{dr}=\boldsymbol{\Omega}\cdot\mathbf{dr}$ 即可。我们通过将 $\boldsymbol{\Omega}\cdot\mathbf{dr}$ 和 $\boldsymbol{\omega}\times\mathbf{dr}$ 展开来证明该等式。首先展开 $\boldsymbol{\Omega}\cdot\mathbf{dr}$

$$\boldsymbol{\Omega}\cdot\mathbf{dr}=\frac{1}{2}(\mathbf{u}\nabla-\nabla\mathbf{u})\cdot\mathbf{dr}=\frac{1}{2}\left(u_{i,j}-u_{j,i}\right)\mathbf{e}_i\mathbf{e}_j\cdot\left(dr_k\mathbf{e}_k\right)=\frac{1}{2}\left(u_{i,j}-u_{j,i}\right)dr_j\mathbf{e}_i$$

$$\boldsymbol{\omega}\times\mathbf{dr}=\frac{1}{2}(\nabla\times\mathbf{u})\times\mathbf{dr}=\frac{1}{2}\left(\nabla_iu_j\varepsilon_{ijk}\mathbf{e}_k\right)\times\left(dr_m\mathbf{e}_m\right)=\frac{1}{2}\nabla_iu_j\varepsilon_{ijk}dr_m\varepsilon_{kml}\mathbf{e}_l=\frac{1}{2}\nabla_iu_j\varepsilon_{kij}dr_m\varepsilon_{kml}\mathbf{e}_l$$

$$=\frac{1}{2}\nabla_iu_jdr_m\varepsilon_{kij}\varepsilon_{kml}\mathbf{e}_l=\frac{1}{2}\nabla_iu_jdr_m\varepsilon_{kij}\varepsilon_{kml}\mathbf{e}_l=\frac{1}{2}\nabla_iu_jdr_m\left(\delta_{im}\delta_{jl}-\delta_{il}\delta_{jm}\right)\mathbf{e}_l$$

$$=\frac{1}{2}\left(\nabla_iu_jdr_m\delta_{im}\delta_{jl}-\nabla_iu_jdr_m\delta_{il}\delta_{jm}\right)\mathbf{e}_l=\frac{1}{2}\left(\nabla_iu_jdr_m\delta_{im}\delta_{jl}\mathbf{e}_l-\nabla_iu_jdr_m\delta_{il}\delta_{jm}\mathbf{e}_l\right)=\frac{1}{2}\left(\nabla_iu_jdr_i\mathbf{e}_j-\nabla_iu_jdr_j\mathbf{e}_i\right)$$

$$=\frac{1}{2}\left(\nabla_iu_jdr_i\mathbf{e}_j-\nabla_ju_idr_i\mathbf{e}_j\right)=\frac{1}{2}\left(\nabla_iu_j-\nabla_ju_i\right)dr_i\mathbf{e}_j=\frac{1}{2}\left(u_{i,j}-u_{j,i}\right)dr_j\mathbf{e}_i=\boldsymbol{\Omega}\cdot\mathbf{dr}$$

证明 $du=(u\nabla)\cdot dr=dr\cdot(\nabla u)$ 由泰勒展开式得到

$$\begin{cases}u(x+dx,y+dy,z+dz)=u(x,y,z)+\dfrac{\partial u}{\partial x}dx+\dfrac{\partial u}{\partial y}dy+\dfrac{\partial u}{\partial z}dz\\ v(x+dx,y+dy,z+dz)=v(x,y,z)+\dfrac{\partial v}{\partial x}dx+\dfrac{\partial v}{\partial y}dy+\dfrac{\partial v}{\partial z}dz\\ w(x+dx,y+dy,z+dz)=w(x,y,z)+\dfrac{\partial w}{\partial x}dx+\dfrac{\partial w}{\partial y}dy+\dfrac{\partial w}{\partial z}dz\end{cases}$$

这里， $\mathbf{u}\nabla=u_{i,j}\mathbf{e}_i\mathbf{e}_j$， $\mathbf{dr}=dx_ie_i$

证明

$$\mathbf{du}=\mathbf{u}(x+dx,y+dy,z+dz)-\mathbf{u}(x,y,z)$$

$$=\mathbf{e}_1\left(\frac{\partial u}{\partial x}dx+\frac{\partial u}{\partial y}dy+\frac{\partial u}{\partial z}dz\right)+\mathbf{e}_2\left(\frac{\partial v}{\partial x}dx+\frac{\partial v}{\partial y}dy+\frac{\partial v}{\partial z}dz\right)+\mathbf{e}_3\left(\frac{\partial w}{\partial x}dx+\frac{\partial w}{\partial y}dy+\frac{\partial w}{\partial z}dz\right)$$

$$=\mathbf{e}_1\left(\frac{\partial u}{\partial x}dx\mathbf{e}_1\cdot\mathbf{e}_1+\frac{\partial u}{\partial y}dy\mathbf{e}_2\cdot\mathbf{e}_2+\frac{\partial u}{\partial z}dz\mathbf{e}_3\cdot\mathbf{e}_3\right)+\mathbf{e}_2\left(\frac{\partial v}{\partial x}dx\mathbf{e}_1\cdot\mathbf{e}_1+\frac{\partial v}{\partial y}dy\mathbf{e}_2\cdot\mathbf{e}_2+\frac{\partial v}{\partial z}dz\mathbf{e}_3\cdot\mathbf{e}_3\right)$$

$$+\mathbf{e}_3\left(\frac{\partial w}{\partial x}dx\mathbf{e}_1\cdot\mathbf{e}_1+\frac{\partial w}{\partial y}dy\mathbf{e}_2\cdot\mathbf{e}_2+\frac{\partial w}{\partial z}dz\mathbf{e}_3\cdot\mathbf{e}_3\right)$$

$$=\mathbf{e}_1\left(\frac{\partial u}{\partial x}\mathbf{e}_1+\frac{\partial u}{\partial y}\mathbf{e}_2+\frac{\partial u}{\partial z}\mathbf{e}_3\right)\cdot\left(dx\mathbf{e}_1+dy\mathbf{e}_2+dz\mathbf{e}_3\right)+\mathbf{e}_2\left(\frac{\partial v}{\partial x}\mathbf{e}_1+\frac{\partial v}{\partial y}\mathbf{e}_2+\frac{\partial v}{\partial z}\mathbf{e}_3\right)\cdot\left(dx\mathbf{e}_1+dy\mathbf{e}_2+dz\mathbf{e}_3\right)$$

$$+\mathbf{e}_3\left(\frac{\partial w}{\partial x}\mathbf{e}_1+\frac{\partial w}{\partial y}\mathbf{e}_2+\frac{\partial w}{\partial z}\mathbf{e}_3\right)\cdot\left(dx\mathbf{e}_1+dy\mathbf{e}_2+dz\mathbf{e}_3\right)$$

$$=\left(\mathbf{u}\nabla\right)\cdot\mathbf{dr}$$

再证明

$$\mathbf{du}=\mathbf{dr}\cdot(\nabla\mathbf{u})$$

$$\begin{aligned}
&\mathbf{du}=\mathbf{u}(x+dx,y+dy,z+dz)-\mathbf{u}(x,y,z)\\
&=\left(\frac{\partial u}{\partial x}dx+\frac{\partial u}{\partial y}dy+\frac{\partial u}{\partial z}dz\right)\mathbf{e}_1+\left(\frac{\partial v}{\partial x}dx+\frac{\partial v}{\partial y}dy+\frac{\partial v}{\partial z}dz\right)\mathbf{e}_2+\left(\frac{\partial w}{\partial x}dx+\frac{\partial w}{\partial y}dy+\frac{\partial w}{\partial z}dz\right)\mathbf{e}_3\\
&=\left(\frac{\partial u}{\partial x}dx\mathbf{e}_1\cdot\mathbf{e}_1+\frac{\partial u}{\partial y}dy\mathbf{e}_2\cdot\mathbf{e}_2+\frac{\partial u}{\partial z}dz\mathbf{e}_3\cdot\mathbf{e}_3\right)\mathbf{e}_1+\left(\frac{\partial v}{\partial x}dx\mathbf{e}_1\cdot\mathbf{e}_1+\frac{\partial v}{\partial y}dy\mathbf{e}_2\cdot\mathbf{e}_2+\frac{\partial v}{\partial z}dz\mathbf{e}_3\cdot\mathbf{e}_3\right)\mathbf{e}_2\\
&+\left(\frac{\partial w}{\partial x}dx\mathbf{e}_1\cdot\mathbf{e}_1+\frac{\partial w}{\partial y}dy\mathbf{e}_2\cdot\mathbf{e}_2+\frac{\partial w}{\partial z}dz\mathbf{e}_3\cdot\mathbf{e}_3\right)\mathbf{e}_3\\
&=\left(dx\mathbf{e}_1+dy\mathbf{e}_2+dz\mathbf{e}_3\right)\cdot\left(\frac{\partial u}{\partial x}\mathbf{e}_1+\frac{\partial u}{\partial y}\mathbf{e}_2+\frac{\partial u}{\partial z}\mathbf{e}_3\right)\mathbf{e}_1+\left(dx\mathbf{e}_1+dy\mathbf{e}_2+dz\mathbf{e}_3\right)\cdot\left(\frac{\partial v}{\partial x}\mathbf{e}_1+\frac{\partial v}{\partial y}\mathbf{e}_2+\frac{\partial v}{\partial z}\mathbf{e}_3\right)\mathbf{e}_2\\
&+\left(dx\mathbf{e}_1+dy\mathbf{e}_2+dz\mathbf{e}_3\right)\cdot\left(\frac{\partial w}{\partial x}\mathbf{e}_1+\frac{\partial w}{\partial y}\mathbf{e}_2+\frac{\partial w}{\partial z}\mathbf{e}_3\right)\mathbf{e}_3\\
&=\mathbf{dr}\cdot(\nabla\mathbf{u})
\end{aligned}$$

1.6.3 应变分析

长度的变化如图 1.12 所示，线段 PA 变形后变为 $P\tilde{A}$。假设 PA 矢量为 $\mathbf{dr}$， $P\tilde{A}$ 矢量为 $d\tilde{r}$ 。

$\mathbf{dr}$ 可以表示为

$$\mathbf{dr}=dr\boldsymbol{\xi} \tag{1.139}$$

其中 dr 和 $\boldsymbol{\xi}$ 分别是矢量 $\mathbf{dr}$ 的长度和单位矢量。根据几何关系可得：

$$\mathbf{d\tilde{r}}=\mathbf{dr}+\mathbf{du} \tag{1.140}$$

现在来计算矢量 $\mathbf{d\tilde{r}}$ 的长度 $d\tilde{r}$ ，根据式(1.140)有：

$$(d\tilde{r})^2=\mathbf{d\tilde{r}}\cdot\mathbf{d\tilde{r}}=\mathbf{dr}\cdot\mathbf{dr}+2\mathbf{dr}\cdot\mathbf{du}+\mathbf{du}\cdot\mathbf{du} \tag{1.141}$$

因为

$$\mathbf{du}=\boldsymbol{\omega}\times\mathbf{dr}+\boldsymbol{\Gamma}\cdot\mathbf{dr} \tag{1.142}$$

所以

$$(d\tilde{r})^2=(dr)^2+2\mathbf{dr}\cdot(\boldsymbol{\omega}\times\mathbf{dr}+\boldsymbol{\Gamma}\cdot\mathbf{dr})+\mathbf{du}\cdot\mathbf{du} \tag{1.143}$$

又因为

$$\mathbf{du}=(\mathbf{u}\nabla)\cdot\mathbf{dr}=\mathbf{dr}\cdot(\nabla\mathbf{u}) \tag{1.144}$$

所以

$$(d\tilde{r})^2=(dr)^2+2\mathbf{dr}\cdot(\boldsymbol{\omega}\times\mathbf{dr}+\boldsymbol{\Gamma}\cdot\mathbf{dr})+\mathbf{dr}\cdot(\nabla\mathbf{u})\cdot(\mathbf{u}\nabla)\cdot\mathbf{dr} \tag{1.145}$$

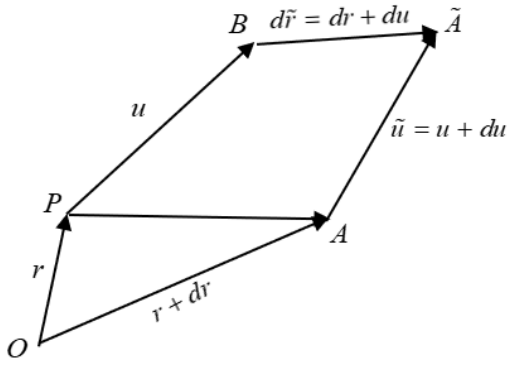


图 1. 12 矢量长度变化示意图

又根据叉乘运算规则，矢量$\boldsymbol{\omega}\times\mathbf{dr}$与矢量$\mathbf{dr}$垂直。所以$\mathbf{dr}\cdot(\boldsymbol{\omega}\times\mathbf{dr})=0$。

于是(1.145)变为

$$(d\tilde{r})^2=(dr)^2+2\mathbf{dr}\cdot\boldsymbol{\Gamma}\cdot\mathbf{dr}+\mathbf{dr}\cdot(\nabla\mathbf{u})\cdot(\mathbf{u}\nabla)\cdot\mathbf{dr} \tag{1.146}$$

又因为$\mathbf{dr}=dr\boldsymbol{\xi}$，所以

$$(d\tilde{r})^2=(dr)^2+2(dr)^2\boldsymbol{\xi}\cdot\boldsymbol{\Gamma}\cdot\boldsymbol{\xi}+(dr)^2\boldsymbol{\xi}\cdot(\nabla\mathbf{u})\cdot(\mathbf{u}\nabla)\cdot\boldsymbol{\xi} \tag{1.147}$$

所以

$$(d\tilde{r})^2=(dr)^2+2(dr)^2\boldsymbol{\xi}\cdot[\boldsymbol{\Gamma}+\frac{1}{2}(\nabla\mathbf{u})\cdot(\mathbf{u})\nabla]\cdot\boldsymbol{\xi} \tag{1.148}$$

令$\mathbf{G}=\boldsymbol{\Gamma}+\frac{1}{2}(\nabla\mathbf{u})\cdot(\mathbf{u}\nabla)$

其中$\mathbf{G}$是 Green 应变张量。$\boldsymbol{\Gamma}$为 Cauchy 应变张量。

考虑小变形，忽略$\mathbf{G}$中有关$u_{i,j}$的二次项部分，所以

$$(d\tilde{r})^2=(dr)^2+2(dr)^2\boldsymbol{\xi}\cdot\boldsymbol{\Gamma}\cdot\boldsymbol{\xi} \tag{1.149}$$

那么dr方向上也就是$\boldsymbol{\xi}$方向上的线应变为：

$$\varepsilon=\frac{d\tilde{r}-dr}{dr}=\sqrt{1+2\boldsymbol{\xi}\cdot\boldsymbol{\Gamma}\cdot\boldsymbol{\xi}}-1\approx\boldsymbol{\xi}\cdot\boldsymbol{\Gamma}\cdot\boldsymbol{\xi}=\xi_i\cdot\gamma_{ij}\cdot\xi_j \tag{1.150}$$

如果$\boldsymbol{\xi}=(1,0,0)$时，由上式可知此时$\varepsilon=\gamma_{11}$。

因此，只要知道某点的应变张量，就可以得到该点任意方向上线元的伸长率。

附注 非线性大变形时，需要考虑 Green 应变张量，它的分量

$$\begin{cases} G_{11}=\dfrac{\partial u}{\partial x}+\dfrac{1}{2}\left[(\dfrac{\partial u}{\partial x})^2+(\dfrac{\partial v}{\partial x})^2+(\dfrac{\partial w}{\partial x})^2\right] \\ G_{22}=\dfrac{\partial v}{\partial y}+\dfrac{1}{2}\left[(\dfrac{\partial u}{\partial y})^2+(\dfrac{\partial v}{\partial y})^2+(\dfrac{\partial w}{\partial y})^2\right] \\ G_{33}=\dfrac{\partial w}{\partial z}+\dfrac{1}{2}\left[(\dfrac{\partial u}{\partial z})^2+(\dfrac{\partial v}{\partial z})^2+(\dfrac{\partial w}{\partial z})^2\right] \\ G_{23}=\dfrac{1}{2}(\dfrac{\partial v}{\partial z}+\dfrac{\partial w}{\partial y})+\dfrac{1}{2}\left[\dfrac{\partial u}{\partial y}\dfrac{\partial u}{\partial z}+\dfrac{\partial v}{\partial y}\dfrac{\partial v}{\partial z}+\dfrac{\partial w}{\partial y}\dfrac{\partial w}{\partial z}\right] \\ G_{31}=\dfrac{1}{2}(\dfrac{\partial w}{\partial x}+\dfrac{\partial u}{\partial z})+\dfrac{1}{2}\left[\dfrac{\partial u}{\partial x}\dfrac{\partial u}{\partial z}+\dfrac{\partial v}{\partial x}\dfrac{\partial v}{\partial z}+\dfrac{\partial w}{\partial x}\dfrac{\partial w}{\partial z}\right] \\ G_{12}=\dfrac{1}{2}(\dfrac{\partial u}{\partial y}+\dfrac{\partial v}{\partial x})+\dfrac{1}{2}\left[\dfrac{\partial u}{\partial x}\dfrac{\partial u}{\partial y}+\dfrac{\partial v}{\partial x}\dfrac{\partial v}{\partial y}+\dfrac{\partial w}{\partial x}\dfrac{\partial w}{\partial y}\right] \end{cases} \tag{1.151}$$

角度的变化

在矢径为r的P点附近有两个点A和B，它们的矢径分别为$\mathbf{r}+\mathbf{dr}$和$\mathbf{r}+\boldsymbol{\delta}\mathbf{r}$，设

$$\mathbf{dr}=(dr)\boldsymbol{\xi}\,,\quad \boldsymbol{\delta}\mathbf{r}=(\delta r)\boldsymbol{\eta} \tag{1.152}$$

其中$\boldsymbol{\xi}$和$\boldsymbol{\eta}$为两个相互垂直的单位矢量。变形后，点P,A,B分别变为$\tilde{P},\tilde{A},\tilde{B}$，它们的位移分别为$\mathbf{u}$、$\mathbf{u}+\mathbf{du}$、$\mathbf{u}+\boldsymbol{\delta}\mathbf{u}$，那么矢量$\tilde{P}\tilde{A}$和$\tilde{P}\tilde{B}$将分别为

$$\mathbf{d\tilde{r}}=\mathbf{dr}+\mathbf{du}\,,\quad \boldsymbol{\delta}\mathbf{\tilde{r}}=\boldsymbol{\delta}\mathbf{r}+\boldsymbol{\delta}\mathbf{u} \tag{1.153}$$

为了考察矢量$\mathbf{d\tilde{r}}$和$\boldsymbol{\delta}\mathbf{\tilde{r}}$之间的夹角，作它们的内积得

$$\mathbf{d\tilde{r}}\cdot\boldsymbol{\delta}\mathbf{\tilde{r}}=\mathbf{dr}\cdot\boldsymbol{\delta}\mathbf{r}+\mathbf{dr}\cdot\boldsymbol{\delta}\mathbf{u}+\mathbf{du}\cdot\boldsymbol{\delta}\mathbf{r}+\mathbf{du}\cdot\mathbf{du} \tag{1.154}$$

将正交条件$\mathbf{d\tilde{r}}\cdot\boldsymbol{\delta}\mathbf{\tilde{r}}=0$，$\mathbf{du}=\mathbf{dr}\cdot(\nabla\mathbf{u})$，$\boldsymbol{\delta}\mathbf{u}=(\mathbf{u}\nabla)\cdot\boldsymbol{\delta}\mathbf{r}$代入上式后得：

$$\mathbf{d\tilde{r}}\cdot\boldsymbol{\delta}\mathbf{\tilde{r}}=\mathbf{dr}\cdot(\boldsymbol{\omega}\times\boldsymbol{\delta}\mathbf{r}+\boldsymbol{\Gamma}\cdot\boldsymbol{\delta}\mathbf{r})+\boldsymbol{\delta}\mathbf{r}\cdot(\boldsymbol{\omega}\times\mathbf{dr}+\boldsymbol{\Gamma}\cdot\mathbf{dr})+\mathbf{dr}\cdot(\nabla\mathbf{u})\cdot(\mathbf{u}\nabla)\cdot\boldsymbol{\delta}\mathbf{r} \tag{1.155}$$

由于$\mathbf{dr}\cdot(\boldsymbol{\omega}\times\boldsymbol{\delta}\mathbf{r})+\boldsymbol{\delta}\mathbf{r}\cdot(\boldsymbol{\omega}\times\mathbf{dr})=0$，所以

$$\mathbf{d\tilde{r}}\cdot\boldsymbol{\delta}\mathbf{\tilde{r}}=\mathbf{dr}\cdot\boldsymbol{\Gamma}\cdot\boldsymbol{\delta}\mathbf{r}+\boldsymbol{\delta}\mathbf{r}\cdot\boldsymbol{\Gamma}\cdot\mathbf{dr}+\mathbf{dr}\cdot(\nabla\mathbf{u})\cdot(\mathbf{u}\nabla)\cdot\boldsymbol{\delta}\mathbf{r} \tag{1.156}$$

$$\mathbf{d\tilde{r}}\cdot\boldsymbol{\delta}\mathbf{\tilde{r}}=2dr\delta r\boldsymbol{\xi}\cdot\mathbf{G}\cdot\boldsymbol{\eta} \tag{1.157}$$

小变形时，有

$$\mathbf{d\tilde{r}}\cdot\boldsymbol{\delta}\mathbf{\tilde{r}}=2dr\delta r\boldsymbol{\xi}\cdot\boldsymbol{\Gamma}\cdot\boldsymbol{\eta} \tag{1.158}$$

设$\mathbf{d\tilde{r}}$和$\boldsymbol{\delta}\mathbf{\tilde{r}}$之间的夹角为$\dfrac{\pi}{2}-2\gamma$；并设$d\tilde{r}=(1+\varepsilon_1)dr$，$\delta\tilde{r}=(1+\varepsilon_2)\delta r$

其中,

$$(1+\varepsilon_1)(1+\varepsilon_2)\sin 2\gamma = 2\xi_i\gamma_{ij}\xi_j \tag{1.159}$$

对于小变形的情形，γ、ε_1 和 ε_2 均很小，略去高阶小量，从上式得：

$$\gamma = \xi_i\gamma_{ij}\xi_j \tag{1.160}$$

上式表明，一旦有了应变张量 γ_{ij}，那么相互垂直的两个方向间的夹角的变化即可算出。

附录 $\mathbf{dr}\cdot(\boldsymbol{\omega}\times\boldsymbol{\delta r})+\boldsymbol{\delta r}\cdot(\boldsymbol{\omega}\times\mathbf{dr})=0$ 的证明。

根据叉乘公式：

$$\boldsymbol{\omega}\times\boldsymbol{\delta r} = \omega_i\delta r_j\varepsilon_{ijk}\mathbf{e}_k \tag{1.161}$$

$$\mathbf{dr}\cdot(\boldsymbol{\omega}\times\boldsymbol{\delta r}) = \mathbf{dr}\cdot(\omega_i\delta r_j\varepsilon_{ijk}\mathbf{e}_k) = \omega_i\delta r_j\varepsilon_{ijk}dr_k \tag{1.162}$$

同理

$$\boldsymbol{\delta r}\cdot(\boldsymbol{\omega}\times\mathbf{dr}) = \omega_i dr_j\varepsilon_{ijk}\delta r_k = \omega_i\delta r_k\varepsilon_{ijk}dr_j = \omega_i\delta r_j\varepsilon_{ikj}dr_k = -\omega_i\delta r_j\varepsilon_{ijk}dr_k = -\mathbf{dr}\cdot(\boldsymbol{\omega}\times\boldsymbol{\delta r}) \tag{1.163}$$

所以

$$\mathbf{dr}\cdot(\boldsymbol{\omega}\times\boldsymbol{\delta r})+\boldsymbol{\delta r}\cdot(\boldsymbol{\omega}\times\mathbf{dr})=0 \tag{1.164}$$

设 $\boldsymbol{\Gamma}$ 在标架 $\{\mathbf{e_1},\mathbf{e_2},\mathbf{e_3}\}$ 下的分量为 γ_{ij}，今有一个新的标架 $\{\mathbf{e}_1',\mathbf{e}_2',\mathbf{e}_3'\}$，基矢量 $\mathbf{e}_i'$ 和 $\mathbf{e}_i$ 的关系为

$$\mathbf{e}_i' = C_{ij}\mathbf{e}_j \tag{1.165}$$

在 $\mathbf{e}_i'$ 方向上的伸长 γ_{11}' 为

$$\gamma_{11}' = C_{1i}\gamma_{ij}C_{1j} = C_{1i}C_{1j}\gamma_{ij} \tag{1.166}$$

按在 $\mathbf{e}_1'$ 和 $\mathbf{e}_2'$ 间的角度变化 γ_{12}' 为

$$\gamma_{12}' = C_{1i}\gamma_{ij}C_{2j} = C_{1i}C_{2j}\gamma_{ij} \tag{1.167}$$

综合(1.166)和(1.167)式，以及关于 γ_{22}'、γ_{33}'、γ_{23}' 和 γ_{31}' 类似的式子，我们可得

$$\gamma_{12}' = C_{1i}\gamma_{ij}C_{2j} = C_{1i}C_{2j}\gamma_{ij} \tag{1.168}$$

另一方面，按前面两节的几何解释，γ_{11}'、γ_{22}'、γ_{33}' 分别为 $\boldsymbol{\Gamma}$ 在新标架下的正应变，

γ_{12}'、γ_{23}'、γ_{31}' 分别为 $\boldsymbol{\Gamma}$ 在新标架下的剪应变，也就是说，γ_{ij}' 为 $\boldsymbol{\Gamma}$ 在新标架下的分量。

从(1.168)式看出，$\boldsymbol{\Gamma}$ 在新旧标架下的分量 γ_{ij}' 与 γ_{ij} 服从关于 C_{ij} 的二次齐次式。

因此，$\boldsymbol{\Gamma}$是一个张量。这样，我们从几何解释给出了$\boldsymbol{\Gamma}$为张量的一种证明。

坐标变换

设应变张量$\boldsymbol{\Gamma}$在新旧坐标系下所对应的矩阵分别为$\boldsymbol{\Gamma}'$和$\boldsymbol{\Gamma}$，那么按(1.168)式，有

$$\boldsymbol{\Gamma}' = \mathbf{C}\boldsymbol{\Gamma}\mathbf{C}^T \tag{1.169}$$

其中矩阵$\mathbf{C}=[C_{ij}]$，上标“T”表示转置。考虑$\mathbf{C}$的两种特殊情形。

情形 1 设绕 z 轴旋转角度为φ（图 1. ），此时变换矩阵$\mathbf{C}$为

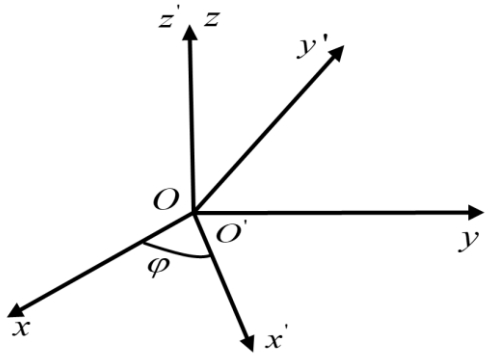


图 1. 13 坐标系$O-xyz$绕 z 轴旋转角度φ后得到新坐标系$O'-x'y'z'$

$$\mathbf{C}=\begin{bmatrix}\cos\varphi & \sin\varphi & 0\\ -\sin\varphi & \cos\varphi & 0\\ 0 & 0 & 1\end{bmatrix} \tag{1.170}$$

将上式代入(1.169)式，得

$$\begin{cases}\gamma_{11}^{'}=\gamma_{11}\cos^2\varphi+\varphi_{22}\sin^2\varphi+\gamma_{12}\sin 2\varphi\\ \gamma_{22}^{'}=\gamma_{11}\sin^2\varphi+\varphi_{22}\cos^2\varphi-\gamma_{12}\sin 2\varphi\\ \gamma_{12}^{'}=(\gamma_{22}-\gamma_{11})\dfrac{\sin 2\varphi}{2}+\gamma_{12}\cos 2\varphi\\ \gamma_{31}^{'}=\gamma_{31}\cos\varphi+\gamma_{23}\sin\varphi\\ \gamma_{23}^{'}=-\gamma_{31}\sin\varphi+\gamma_{23}\cos\varphi\\ \gamma_{33}^{'}=\gamma_{33}\end{cases} \tag{1.171}$$

情形 2 球坐标变换，其标架为$\left\{r^0,\theta^0,\varphi^0\right\}$（图 1. ），此时变换矩阵$\mathbf{C}$为

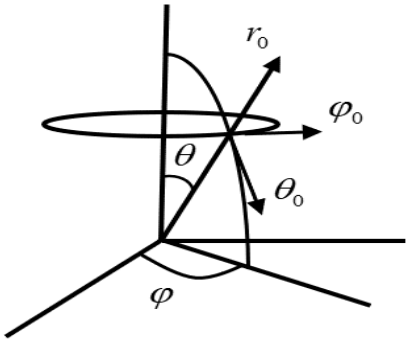


图 1. 14 球坐标下矢量变换示意图

$$\mathbf{C}=\begin{bmatrix}\sin\theta\cos\varphi & \sin\theta\sin\varphi & \cos\theta\\ \cos\theta\cos\varphi & \cos\theta\sin\varphi & -\sin\theta\\ -\sin\varphi & \cos\varphi & 0\end{bmatrix} \tag{1.172}$$

将上式代入(1.172)式，得

$$\begin{cases}\varepsilon_r=\varepsilon_x\sin^2\theta\cos^2\varphi+\varepsilon_y\sin^2\theta\sin^2\varphi+\varepsilon_z\cos^2\theta-\gamma_{xy}\sin^2\theta\sin 2\varphi+\gamma_{yz}\sin 2\theta\sin\varphi+\gamma_{zx}\sin 2\theta\cos\varphi\\ \varepsilon_\theta=\varepsilon_x\cos^2\theta\cos^2\varphi+\varepsilon_y\cos^2\theta\sin^2\varphi+\varepsilon_z\sin^2\theta+\gamma_{xy}\cos^2\theta\sin 2\varphi-\gamma_{yz}\sin 2\theta\sin\varphi-\gamma_{zx}\sin 2\theta\cos\varphi\\ \gamma_{r\theta}=\frac{1}{2}\varepsilon_x\sin 2\theta\cos^2\varphi+\frac{1}{2}\varepsilon_y\sin 2\theta\sin^2\varphi-\frac{1}{2}\varepsilon_z\sin 2\theta+\frac{1}{2}\gamma_{xy}\sin 2\theta\sin 2\varphi+\gamma_{xy}\cos 2\theta\sin\varphi+\gamma_{zx}\cos 2\theta\cos\varphi\\ \gamma_{\theta\varphi}=\frac{1}{2}(\varepsilon_y-\varepsilon_x)\cos\theta\sin 2\varphi+\gamma_{xy}\cos\theta\cos 2\varphi-\gamma_{yz}\sin\theta\cos\varphi+\gamma_{zx}\sin\theta\sin\varphi\\ \gamma_{\varphi r}=\frac{1}{2}(\varepsilon_y-\varepsilon_x)\sin\theta\sin 2\varphi+\gamma_{xy}\sin\theta\cos 2\varphi+\gamma_{yz}\cos\theta\cos\varphi-\gamma_{zx}\cos\theta\sin\varphi\\ \varepsilon_\varphi=\varepsilon_x\sin^2\varphi+\varepsilon_y\cos^2\varphi-\gamma_{zx}\sin 2\varphi\end{cases} \tag{1.173}$$

(1.169)式表示应变张量$\mathbf{\Gamma}$在新旧坐标系下所对应的矩阵$\mathbf{\Gamma}'$和$\mathbf{\Gamma}$构成一个合同变换，从线性代数理论知道，存在标架$\{\boldsymbol{\xi}_1,\boldsymbol{\xi}_2,\boldsymbol{\xi}_3\}$使$\mathbf{\Gamma}'$成对角形，即存在矩阵$\mathbf{C}$，使

$$\mathbf{C}\mathbf{\Gamma}\mathbf{C}^T=\begin{bmatrix}\lambda_1 & 0 & 0\\ 0 & \lambda_2 & 0\\ 0 & 0 & \lambda_3\end{bmatrix} \tag{1.174}$$

通常称$\{\boldsymbol{\xi}_1,\boldsymbol{\xi}_2,\boldsymbol{\xi}_3\}$为应变张量$\mathbf{\Gamma}$的主坐标系，$\xi_i(i=1,2,3)$称为主方向，$\lambda_i(i=1,2,3)$称为主应变。由于$\mathbf{\Gamma}$是实对称矩阵，当然总存在三个互相垂直的主方向。显然，主方向间的剪应变为零。

矩阵$\mathbf{\Gamma}$的本征多项式为

$$\begin{vmatrix}\lambda-\gamma_{11} & -\gamma_{12} & -\gamma_{13}\\ -\gamma_{21} & \lambda-\gamma_{22} & -\gamma_{21}\\ -\gamma_{31} & -\gamma_{32} & \lambda-\gamma_{13}\end{vmatrix}=\lambda^3-I_1\lambda^2+I_2\lambda-I_3 \tag{1.175}$$

既然本征行列式在合同变换下不变，*I_i* (*i*=1,2,3)应为坐标变换下的不变量，其中

$$\begin{aligned}I_1&=\gamma_{11}+\gamma_{22}+\gamma_{33}=\lambda_1+\lambda_2+\lambda_3\\ I_2&=\begin{vmatrix}\gamma_{11} & \gamma_{12}\\ \gamma_{21} & \gamma_{22}\end{vmatrix}+\begin{vmatrix}\gamma_{22} & \gamma_{23}\\ \gamma_{32} & \gamma_{33}\end{vmatrix}+\begin{vmatrix}\gamma_{33} & \gamma_{31}\\ \gamma_{13} & \gamma_{11}\end{vmatrix}=\lambda_1\lambda_2+\lambda_2\lambda_3+\lambda_1\lambda_3\\ I_3&=\begin{vmatrix}\gamma_{11} & \gamma_{12} & \gamma_{13}\\ \gamma_{21} & \gamma_{22} & \gamma_{23}\\ \gamma_{31} & \gamma_{32} & \gamma_{33}\end{vmatrix}=\lambda_1\lambda_2\lambda_3\end{aligned} \tag{1.176}$$

1.6.4 应力分析

应力张量的定义

$$\mathbf{T}=\sigma_{ij}\mathbf{e}_i\mathbf{e}_j \tag{1.177}$$

斜面上的应力，应力张量的含义

$$t_i = n_j \sigma_{ij} (i = 1,2,3) \tag{1.178}$$

应力张量的证明

$$\begin{cases} \sigma_{11}^{'} = (e_1^{'} \cdot T) \cdot e_1^{'} = C_{1k} \sigma_{ks} C_{1s} \\ \sigma_{12}^{'} = (e_1^{'} \cdot T) \cdot e_2^{'} = C_{1k} \sigma_{ks} C_{2s} \end{cases} \tag{1.179}$$

$$T = \sigma_{ij} e_i e_j = \sigma_{ij} C_{ik} \mathbf{e}_k^{'} C_{jl} \mathbf{e}_l^{'} \tag{1.180}$$

所以

$$\sigma_{ij} C_{ik} C_{jl} = \sigma_{kl}^{'} \tag{1.181}$$

(1.182)式表示应力张量 $\mathbf{T}$ 在新旧坐标系下所对应的矩阵 $\mathbf{T}^{'}$ 和 $\mathbf{T}$ 构成一个合同变换，从线性代数理论知道，存在标架 $\{\xi_1, \xi_2, \xi_3\}$ 使 $\mathbf{T}^{'}$ 成对角形，即存在矩阵 $\mathbf{C}$，使

$$\mathbf{CTC}^T = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \tag{1.182}$$

通常称 $\{\xi_1, \xi_2, \xi_3\}$ 为应力张量 $\mathbf{T}$ 的主坐标系，$\xi_i (i = 1,2,3)$ 称为主方向，$\lambda_i (i = 1,2,3)$ 称为主应力。由于 $\mathbf{T}$ 是实对称矩阵，当然总存在三个互相垂直的主方向。显然，主方向斜面的剪应力为零。

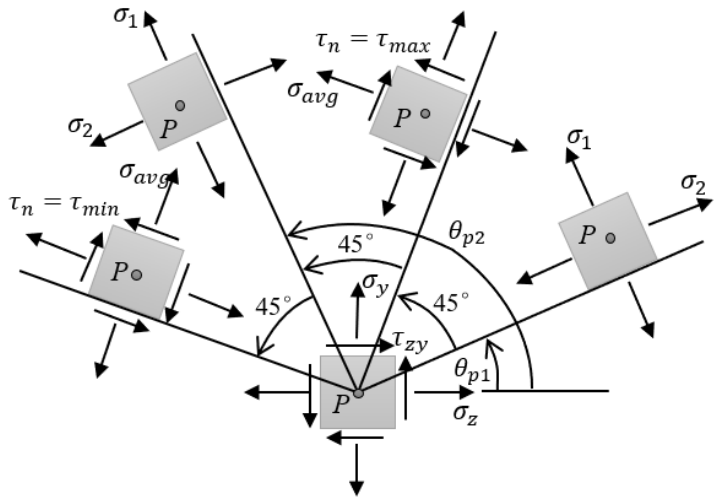


图 1. 15 压力情况

斜面上 P 点的应力 (Stress on any plane)

取点 P 附近斜小平面 AB 与 P 构成直角三角形。 $AB \perp Z$ 轴。斜面沿 X 、 Y 轴方向余弦分别为： l ， m .

$$l=\cos\alpha, m=\sin\alpha \tag{1.183}$$

(1) 推导：

将 P_x 、 P_y 和 $\overline{f}_x$ ， $\overline{f}_y$ 类比可直接得：

$$\begin{pmatrix} p_x \\ p_y \end{pmatrix} = \begin{pmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{pmatrix} \begin{pmatrix} \mathbf{l} \\ \mathbf{m} \end{pmatrix} \tag{1.184}$$

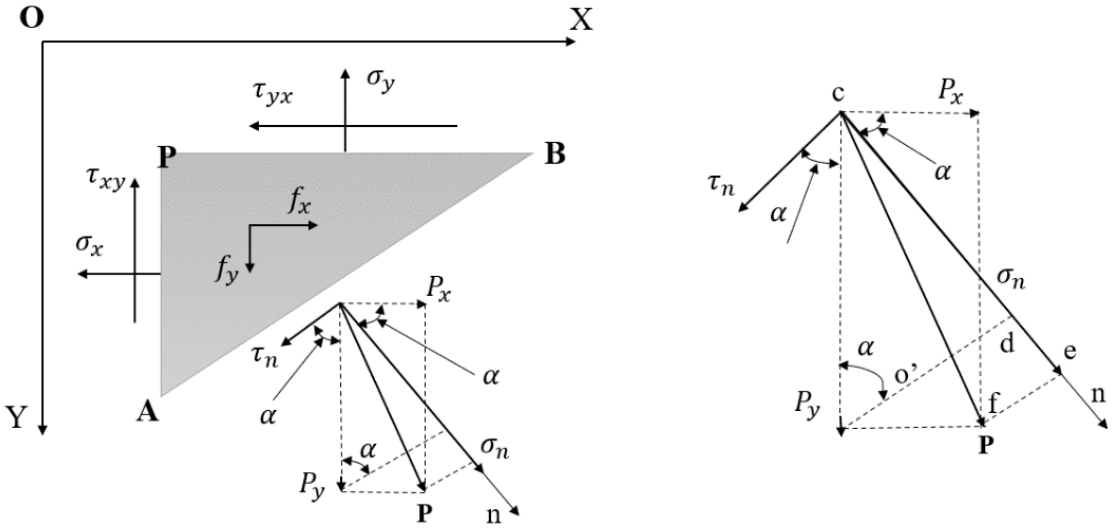


图 1. 16 某一点的应力

2) 将 P_x 、 P_y 向斜面法向投影，可得 σ_n (参看前页) ：

$$(\sigma_n) = (l \quad m)\begin{pmatrix} p_x \\ p_y \end{pmatrix} = (l \quad m)\begin{pmatrix} \sigma_x & \tau_{yx} \\ \tau_{xy} & \sigma_y \end{pmatrix}\begin{pmatrix} l \\ m \end{pmatrix} \tag{1.185}$$

3) 将 P_x 、 P_y 向斜面方向投影，可得 τ_n (参看前页)：

$$(\tau_n) = (-m \quad l)\begin{pmatrix} p_x \\ p_y \end{pmatrix} = (-m \quad l)\begin{pmatrix} \sigma_x & \tau_{yx} \\ \tau_{xy} & \sigma_y \end{pmatrix}\begin{pmatrix} l \\ m \end{pmatrix} \tag{1.186}$$

其中 l, m 为该斜面外法线方向余弦，故有：

$$\sigma_n = l^2\sigma_x + m^2\sigma_y + 2\,l\,m\tau_{xy} \tag{1.187}$$

$$\tau_n = l\,m\,(\sigma_y - \sigma_x) + (l^2 - m^2)\tau_{xy} \tag{1.188}$$

主应力和应力主向 (Principal Stress and Direction)

(1)主应力面(Principal Plane)上 $\tau_n = 0$ ，该面法向为应力主向.

(2)将： $P_x = 1\cdot\sigma, P_y = m\cdot\sigma$ 代入式，整理得齐次方程组(矩阵形式)：

$$\begin{pmatrix} \sigma_x - \sigma & \tau_{xy} \\ \tau_{xy} & \sigma_y - \sigma \end{pmatrix}\begin{pmatrix} \mathbf{l} \\ \mathbf{m} \end{pmatrix} = 0 \tag{1.189}$$

式中 l,m 为斜面外法线方向余弦.

据齐次方程非零解条件有：

$$\sigma^2-(\sigma_x\ +\sigma_y)\sigma+(\sigma_x\cdot\sigma_y-\tau^2_{xy})=0 \tag{1.190}$$

$$\frac{\sigma_1}{\sigma_2}=\frac{\sigma_x+\sigma_y}{2}\pm\sqrt{(\frac{\sigma_x-\sigma_y}{2})^2+\tau^2_{xy}} \tag{1.191}$$

$$\sigma_1+\sigma_2=\sigma_x+\sigma_y \tag{1.192}$$

(3) 应力主向

将式展开成代数方程组：

$$\begin{aligned}-(\sigma-\sigma_x)\,l+\tau_{xy}\,m&=0\\ \tau_{xy}\,l-\ (\sigma-\sigma_y)\,m\quad&=0\end{aligned} \tag{1.193}$$

即：

$$\frac{m}{l}=\frac{\sigma-\sigma_x}{\tau_{xy}}=\frac{\tau_{xy}}{\sigma-\sigma_y}=\tan\alpha \tag{1.194}$$

其中：$\tan\alpha$ 为法线矢量。

分别将 $\sigma=\sigma_1$， $\sigma=\sigma_2$ 代入上式得：

$$\frac{m_1}{l_1}=\frac{\sigma_1-\sigma_x}{\tau_{xy}}=\tan\alpha_1 \tag{1.195}$$

$$\frac{m_2}{l_2}=\frac{\tau_{xy}}{\sigma_2-\sigma_y}=\tan\alpha_2=\frac{-\tau_{xy}}{(\sigma_1-\sigma_x)} \tag{1.196}$$

$(m_1,l_1),(m_2,l_2)$ 分别为两个主方向的方向余弦.所以有

$$\tan\alpha_1\cdot\tan\alpha_2=-1 \tag{1.197}$$

根据二矢量正交条件有：

$$\sigma_1\perp\sigma_2 \tag{1.198}$$

(4) 最大 (小) 主应力

取 x 轴与 y 轴分别与 σ_1、 σ_2 方向同.

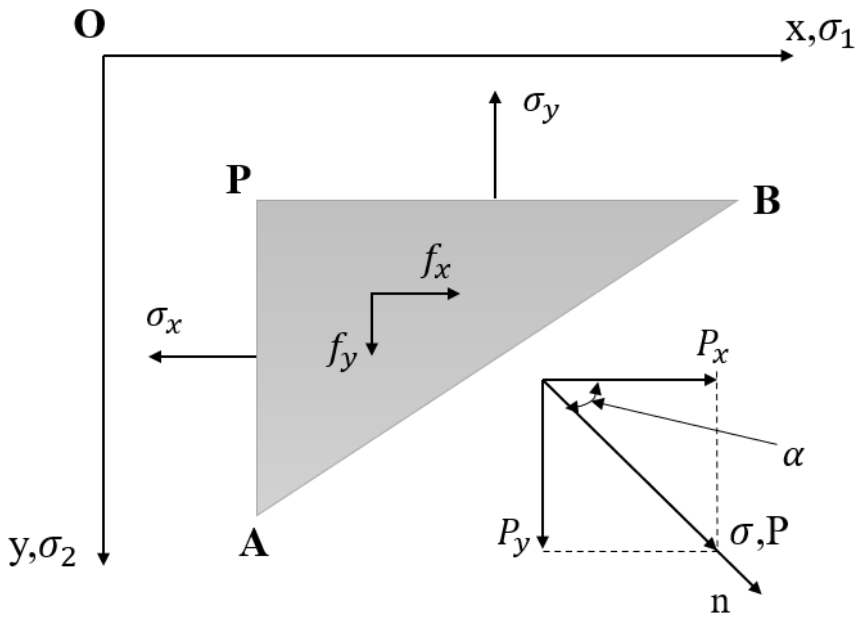


图 1.17 主应力

$$\sigma_n = l^2\sigma_1 + m^2\sigma_2 \tag{1.199}$$

$$\because l^2 + m^2 = 1 \tag{1.200}$$

$$\therefore \sigma_n = l^2(\sigma_1 - \sigma_2) + \sigma_2 \tag{1.201}$$

$$\because 0 \le l \le 1 \tag{1.202}$$

$\therefore \sigma_n$的极值为σ_1、σ_2

$\therefore$两个主应力σ_1、σ_2分别是最大和最小主应力

(5) 最大最小剪应力:

仍按上方式取坐标轴

$$\tau_n = l \cdot m(\sigma_2 - \sigma_1) \tag{1.203}$$

利用 $l^2 + m^2 = 1$ 消去 m 得:

$$\tau_n = \pm l\sqrt{1-l^2}(\sigma_2 - \sigma_1) = \pm\sqrt{l^2 - l^4}(\sigma_2 - \sigma_1) \tag{1.204}$$

$$= \sqrt{\frac{1}{4} - (\frac{1}{2} - l^2)^2}(\sigma_2 - \sigma_1) \tag{1.205}$$

由此可见：当$\frac{1}{2} - l^2 = 0$， τ_n取极值

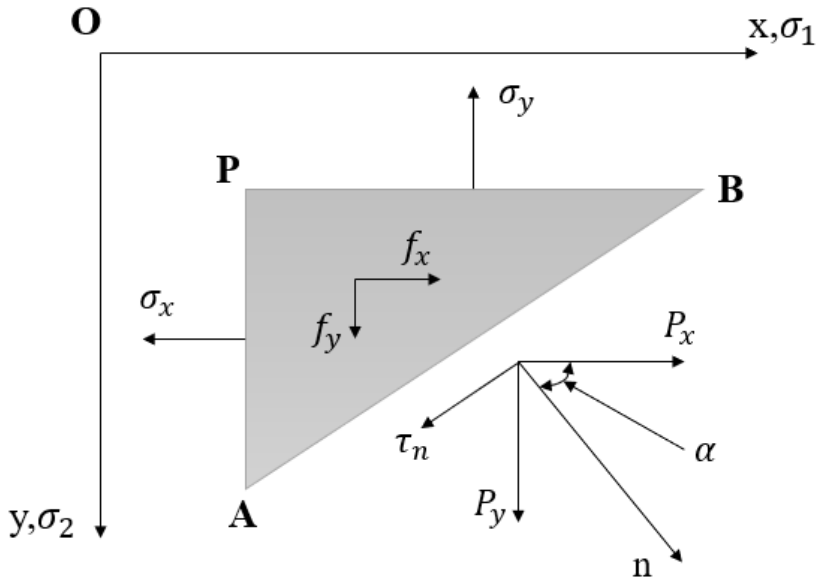


图 1.18 最大剪应力

$l=\pm\sqrt{\frac{1}{2}}$ ，最大和最小剪应力为$\pm\frac{\sigma_1-\sigma_2}{2}$；发生在与主应力面成$45^\circ$斜面上。

1.7 习题

本章介绍了有限元法的基本概念和计算流程，以杆单元为例讲解了如何应用 C 语言实现有限元计算过程，同时介绍了弹性力学的基本方程。请完成以下作业：

（1）推导杆单元的刚度方程；

（2）以本章介绍的方法为基础，编写桁架结构变形计算程序；

（3）推导二维问题中任意斜面的应力计算公式；

（4）推导平衡方程、几何方程和物理方程。

第二章 加权余量法与变分法

2.1 微分方程的等效积分形式

假设空间中一个研究对象Ω如图 2. 1 所示，其边界为Γ。弹性力学中研究的微分方程和边界条件可以表示为：

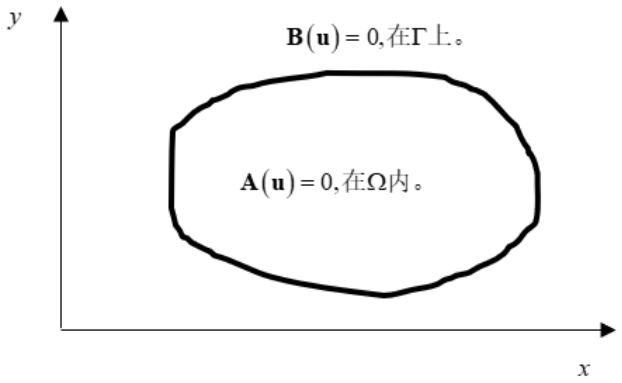


图 2. 1 弹性力学研究对象的空间域和边界域

微分方程 $\mathbf{A}(\mathbf{u})=\begin{pmatrix}A_1(\mathbf{u})\\A_2(\mathbf{u})\\\vdots\end{pmatrix}=0$，在$\Omega$内，边界条件$\mathbf{B}(\mathbf{u})=\begin{pmatrix}B_1(\mathbf{u})\\B_2(\mathbf{u})\\\vdots\end{pmatrix}=0$在$\Gamma$上。其中$\mathbf{A}$，$\mathbf{B}$表示微分算子。微分方程数、边界条件数一般与未知函数$\mathbf{u}$的个数相同。

微分方程的等效积分形式如下：

$$\int_\Omega \mathbf{v}^T\mathbf{A}(\mathbf{u})d\Omega+\int_\Gamma \bar{\mathbf{v}}^T\mathbf{B}(\mathbf{u})d\Gamma=0 \tag{2.1}$$

其中$\mathbf{v}$是任意函数向量，$\mathbf{v}=(v_1,v_2,v_3,\cdots)^T$，$\bar{\mathbf{v}}$是$\mathbf{v}$在边界上的函数。（1）如果$\mathbf{u}$严格满足微分方程和边界条件，则对于任意$\mathbf{v}$，上述积分方程必然成立；（2）如果对于任意的$\mathbf{v}$，$\mathbf{u}$都能让上述积分方程成立，则$\mathbf{u}$必然严格满足微分方程和边界条件。

方程（2.1）能够进行运算的基本条件是方程中的各项积分可积。这对被积函数的连续性提出了要求。等效积分形式对未知函数$\mathbf{u}$连续性要求：

如果微分算子中出现最高次导数为n次，则要求$\mathbf{u}$至少要有$n-1$连续性。即：被积函数不能有无穷大。但是这对函数$\mathbf{u}$的连续性提出了较高的要求。为了降低$\mathbf{u}$的连续性要求，人们提出了方程(2.1)的“弱”形式：

$$\int_\Omega \mathbf{C}^T(\mathbf{v})\mathbf{D}(\mathbf{u})d\Omega+\int_\Gamma \mathbf{E}^T(\mathbf{v})\mathbf{F}(\mathbf{u})d\Gamma=0 \tag{2.2}$$

其中**C**、**D**、**E**、**F**是微分算子。该形式通过对原积分形式使用分部积分、格林公式（二维）、高斯公式（三维），降低**u**的导数阶次，但会增加**v**的导数阶次。下面以二维热传导方程为例说明如何从微分方程和边界条件建立等效积分形式，再进一步推导出其“弱”形式。

二维稳态热传导方程及边界条件如下：

$$A(\phi)=\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)+\frac{\partial}{\partial y}\left(k\frac{\partial\phi}{\partial y}\right)+Q=0\text{，在体积域内}\Omega \tag{2.3}$$

$$B(\phi)=\begin{cases}\phi-\bar{\phi}=0 & (\text{在}\Gamma_\phi\text{上})\\ k\dfrac{\partial\phi}{\partial\mathbf{n}}-\bar{q}=0 & (\text{在}\Gamma_q\text{上})\end{cases} \tag{2.4}$$

其中ϕ表示温度；k是热传导系数；$\bar{\phi}$和$\bar{q}$分别是边界Γ_ϕ和Γ_q上温度和热流值；**n**是边界上的外法线向量；Q是热源密度。如果方程(2.3)在体积域内处处成立，那么在Ω内任意一点有$A(\phi)=\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)+\frac{\partial}{\partial y}\left(k\frac{\partial\phi}{\partial y}\right)+Q=0$。因此方程(2.3)和边界条件(2.4)的等效积分形式为：

$$\int_\Omega v\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)+\frac{\partial}{\partial y}\left(k\frac{\partial\phi}{\partial y}\right)+Q\right]d\Omega+\int_{\Gamma_\phi}\bar{v}\left[\phi-\bar{\phi}\right]d\Gamma+\int_{\Gamma q}\bar{v}\left[k\frac{\partial\phi}{\partial\mathbf{n}}-\bar{q}\right]d\Gamma\equiv 0 \tag{2.5}$$

上述方程对于任意函数v都是成立的。反过来，如果方程(2.5)对于任意v都是成立的，则微分方程(2.3)和(2.4)必然在域内任意一点及边界上都是成立的。因为，如果方程(2.3)和(2.4)在域内某一点不成立，则必然能找到一个v使得积分方程(2.5)不等于零。所以微分方程(2.3)和边界条件(2.4)与积分方程(2.5)是等价的。假设选取的函数ϕ事先满足Γ_ϕ上的边界条件，则方程(2.5)简化为：

$$\int_\Omega v\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)+\frac{\partial}{\partial y}\left(k\frac{\partial\phi}{\partial y}\right)+Q\right]d\Omega+\int_{\Gamma q}\bar{v}\left[k\frac{\partial\phi}{\partial n}-\bar{q}\right]d\Gamma\equiv 0 \tag{2.6}$$

我们注意到方程(2.6)对ϕ的最高阶导数是二阶导数。为了降低方程对ϕ的连续性要求，首先运用分部积分降低ϕ的导数阶次。

$$\int_\Omega v\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)dxdy=\int_\Omega\frac{\partial}{\partial x}\left(v\cdot k\frac{\partial\phi}{\partial x}\right)dxdy-\int_\Omega\frac{\partial v}{\partial x}\cdot k\frac{\partial\phi}{\partial x}dxdy \tag{2.7}$$

注意到积分$\int_\Omega\frac{\partial}{\partial x}\left(v\cdot k\frac{\partial\phi}{\partial x}\right)dxdy$可以应用格林公式进行化简。格林公式如下：

$$\int_\Omega\frac{\partial Q}{\partial x}dxdy=\oint_\Gamma Qdy\text{，}\quad\int_\Omega\frac{\partial P}{\partial y}dxdy=-\oint_\Gamma Pdx \tag{2.8}$$

定义边界Γ的外法线方向为**n**，那么$dx=-n_y d\Gamma$，$dy=n_x d\Gamma$。这样积分(2.7)变为

$$\int_\Omega v\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)dxdy=\oint_\Gamma v\cdot k\frac{\partial\phi}{\partial x}n_x d\Gamma-\int_\Omega\frac{\partial v}{\partial x}\cdot k\frac{\partial\phi}{\partial x}dxdy \tag{2.9}$$

同理

$$\int_\Omega v\frac{\partial}{\partial y}\left(k\frac{\partial\phi}{\partial y}\right)dxdy=\oint_\Gamma v\cdot k\frac{\partial\phi}{\partial y}n_y d\Gamma-\int_\Omega\frac{\partial v}{\partial y}\cdot k\frac{\partial\phi}{\partial y}dxdy \tag{2.10}$$

将(2.9)和(2.10)代入(2.6)后得：

$$\begin{aligned}&\int_\Omega v\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)+\frac{\partial}{\partial y}\left(k\frac{\partial\phi}{\partial y}\right)+Q\right]d\Omega+\int_{\Gamma q}\bar{v}\left[k\frac{\partial\phi}{\partial n}-\bar{q}\right]d\Gamma\\&=\oint_\Gamma v\cdot k\frac{\partial\phi}{\partial x}n_x d\Gamma+\oint_\Gamma v\cdot k\frac{\partial\phi}{\partial y}n_y d\Gamma-\int_\Omega\frac{\partial v}{\partial x}\cdot k\frac{\partial\phi}{\partial x}dxdy-\int_\Omega\frac{\partial v}{\partial y}\cdot k\frac{\partial\phi}{\partial y}dxdy+\int_\Omega vQd\Omega+\int_{\Gamma q}\bar{v}\left[k\frac{\partial\phi}{\partial n}-\bar{q}\right]d\Gamma=0\end{aligned} \tag{2.11}$$

化简后得

$$\int_\Omega\left(\frac{\partial v}{\partial x}\cdot k\frac{\partial\phi}{\partial x}+\frac{\partial v}{\partial y}\cdot k\frac{\partial\phi}{\partial y}\right)dxdy-\int_\Omega vQd\Omega-\oint_\Gamma v\cdot k\left(\frac{\partial\phi}{\partial x}n_x+\frac{\partial\phi}{\partial y}n_y\right)d\Gamma-\int_{\Gamma q}\bar{v}\left[k\frac{\partial\phi}{\partial n}-\bar{q}\right]d\Gamma=0 \tag{2.12}$$

注意到$\frac{\partial\phi}{\partial x}n_x+\frac{\partial\phi}{\partial y}n_y=\frac{\partial\phi}{\partial n}$，且$\oint_\Gamma v\cdot k\left(\frac{\partial\phi}{\partial x}n_x+\frac{\partial\phi}{\partial y}n_y\right)d\Gamma=\oint_{\Gamma_q}v\cdot k\frac{\partial\phi}{\partial n}d\Gamma+\oint_{\Gamma_\phi}v\cdot k\frac{\partial\phi}{\partial n}d\Gamma$，所以(2.12)化简为：

$$\int_\Omega\left(\frac{\partial v}{\partial x}\cdot k\frac{\partial\phi}{\partial x}+\frac{\partial v}{\partial y}\cdot k\frac{\partial\phi}{\partial y}\right)dxdy-\int_\Omega vQd\Omega-\oint_{\Gamma_q}v\cdot k\frac{\partial\phi}{\partial n}d\Gamma-\oint_{\Gamma_\phi}v\cdot k\frac{\partial\phi}{\partial n}d\Gamma-\int_{\Gamma q}\bar{v}\left[k\frac{\partial\phi}{\partial n}-\bar{q}\right]d\Gamma=0 \tag{2.13}$$

进一步，因为$\bar{v}$是任意函数，我们可以令$\bar{v}=-v|_\Gamma$。则方程(2.13)进一步化简为：

$$\int_\Omega\left(\frac{\partial v}{\partial x}\cdot k\frac{\partial\phi}{\partial x}+\frac{\partial v}{\partial y}\cdot k\frac{\partial\phi}{\partial y}\right)dxdy-\int_\Omega vQd\Omega-\oint_{\Gamma_\phi}v\cdot k\frac{\partial\phi}{\partial n}d\Gamma-\int_{\Gamma q}v\bar{q}d\Gamma=0 \tag{2.14}$$

积分方程(2.14)即是等效积分的“弱”形式。

2.2 加权余量法

上一节我们给出了微分方程和边界条件的等效积分形式。这一节我们介绍如何应用等效积分形式获得原微分方程的近似解。假设未知场函数u可以采用如下近似函数来表示：

$$u\approx\bar{u}=\sum_i^n N_i a_i=\mathbf{N}\cdot\mathbf{a} \tag{2.15}$$

其中a_i是待定参数，N_i是试探函数（或基函数、形函数）的已知函数，它取自完全的函数序列，是线性独立的。完全的函数系列的意思是说任一函数都可以用这种序列表示出来。近似解通常选择使之满足强制边界条件和连续性的要求。以三维力学问题为例，位移场的近似解可以表示为：

$$\begin{aligned}u&=\sum_i^n N_i u_i=N_1u_1+N_2u_2+\cdots+N_nu_n\\v&=\sum_i^n N_i v_i=N_1v_1+N_2v_2+\cdots+N_nv_n\\w&=\sum_i^n N_i w_i=N_1w_1+N_2w_2+\cdots+N_nw_n\end{aligned} \tag{2.16}$$

因为是近似解，并不能精确满足微分方程和边界条件，他们将产生残差$\mathbf{R}$及$\bar{\mathbf{R}}$，即：

$$\mathbf{A}(\mathbf{N}\cdot\mathbf{a})=\mathbf{R},\mathbf{B}(\mathbf{N}\cdot\mathbf{a})=\bar{\mathbf{R}} \tag{2.17}$$

将(2.17)代入方程(2.1)中，可以得到如下积分方程：

$$\int_{\Omega}\mathbf{v}^{T}\cdot\mathbf{R}d\Omega+\int_{\Gamma}\bar{\mathbf{v}}^{T}\cdot\bar{\mathbf{R}}d\Gamma=\mathbf{0} \tag{2.18}$$

假设近似函数(2.15)中存在 n 个待定的未知数 a_i ,那么方程(2.18)中也存在 n 个待定的未知数 a_i 。现在的目标就是如何选择权函数 $\mathbf{v}$ 和 $\bar{\mathbf{v}}$ 构造 n 个方程，求解出待定系数 a_i 。按照权函数的类型，一般可以分为如下五种方法：

（1）配点法

配点法选择 $\mathbf{v}_j=\delta\left(\mathbf{x}-\mathbf{x}_j\right)$ 。该函数的意义是，当 $\mathbf{x}\neq\mathbf{x}_j$ 时 $\mathbf{v_j}=\mathbf{0}$ ，但有

$$\int_{\Omega}\mathbf{v}_j d\Omega=\mathbf{I}\quad(j=1，2，\cdots，n) \tag{2.19}$$

这种方法相当于简单地强迫余量在域内 n 个点上等于零。

（2）子域法

在 n 个子域 Ω_j 内， $\mathbf{v}_j=\mathbf{I}$ ，在子域 Ω_j 以外， $\mathbf{v}_j=\mathbf{0}$ 。此方法的实质是强迫余量在 n 个子域 Ω_j 的积分为零。

（3）最小二乘法

当近似解取为 $\tilde{\mathbf{u}}=\sum_{i=1}^{n}\mathbf{N}_i\mathbf{a}_i$ 时，权函数 $\mathbf{v}_j=\frac{\partial}{\partial\mathbf{a}_j}\mathbf{A}(\sum_{i=1}^{n}\mathbf{N}_i\mathbf{a}_i)$

此方法的实质是使得函数

$$\mathbf{I}(\mathbf{a}_i)=\int_{\Omega}\mathbf{A}^{T}(\sum_{i=1}^{n}\mathbf{N}_i\mathbf{a}_i)A(\sum_{i=1}^{n}\mathbf{N}_i\mathbf{a}_i)d\Omega \tag{2.20}$$

取最小值。即要求 $\frac{\partial I}{\partial\mathbf{a}_i}=0$ （ i =1，2，…，n)

（4）力矩法

以一维问题为例，微分方程 $A(u)=0$ ，取近似解 $\tilde{u}$ 并假定已满足边界条件。令

$$\mathbf{v}_j=1,x,x^2,\cdots \tag{2.21}$$

则得到

$$\int_{\Omega}A(\tilde{u})dx=0\ ,\ \int_{\Omega}xA(\tilde{u})dx=0\ ,\ \int_{\Omega}x^2A(\tilde{u})dx=0\ ,\ \cdots \tag{2.22}$$

此方法是强迫余量的各次矩等于零。通常又称此法为积分法。对于二维问题，$\mathbf{v}_j=1,x,y,x^2,xy,y^2\cdots$ 。

（5）伽辽金法

取 $\mathbf{v}_j = \mathbf{N}_j$，在边界上 $\bar{\mathbf{v}}_j = -\mathbf{v}_j = -\mathbf{N}_j$。即简单地利用近似解的试探函数序列作为权函数。代入方程(2.18)可得

$$\int_\Omega \mathbf{N}_j^T \cdot \mathbf{R} d\Omega - \int_\Gamma \mathbf{N}_j^T \cdot \bar{\mathbf{R}} d\Gamma = \mathbf{0} \tag{2.23}$$

由方程(2.17)可知式(2.23)近似等效积分形式为

$$\int_\Omega \mathbf{N}_j^T \cdot \mathbf{A}(\sum_{i=i}^{n} \mathbf{N}_i \mathbf{a}_i) d\Omega - \int_\Gamma \mathbf{N}_j^T \cdot \mathbf{B}(\sum_{i=i}^{n} \mathbf{N}_i \mathbf{a}_i) d\Gamma = \mathbf{0} \tag{2.24}$$

定义近似解 $\tilde{\mathbf{u}}$ 的变分为 $\delta\tilde{\mathbf{u}}$ 为

$$\delta\tilde{\mathbf{u}} = \mathbf{N}_1 \delta\mathbf{a}_1 + \mathbf{N}_2 \delta\mathbf{a}_2 + \cdots + \mathbf{N}_n \delta\mathbf{a}_n \tag{2.25}$$

其中 $\delta\mathbf{a}_i$ 是完全任意的。由此式(2.24)变为

$$\int_\Omega \delta\tilde{\mathbf{u}}^T \cdot \mathbf{A}(\tilde{\mathbf{u}}) d\Omega - \int_\Gamma \delta\tilde{\mathbf{u}}^T \cdot \mathbf{B}(\tilde{\mathbf{u}}) d\Gamma = \mathbf{0} \tag{2.26}$$

对于弱积分形式方程(2.2)则有

$$\int_\Omega \mathbf{C}^T(\delta\tilde{\mathbf{u}}) \square \mathbf{D}(\tilde{\mathbf{u}}) d\Omega + \int_\Gamma \mathbf{E}^T(\delta\tilde{\mathbf{u}}) \square \mathbf{F}(\tilde{\mathbf{u}}) d\Gamma = 0 \tag{2.27}$$

如果算子 $\mathbf{A}$ 是 $2m$ 阶的线性自伴随的，则采用伽辽金法得到的求解方程的系数矩阵是对称的，这是在用加权余量法建立有限元格式时必然采用伽辽金法的主要原因。而且微分方程存在相应的泛函时，伽辽金法与变分法往往得到同样的结果。

下面通过例题来对加权余量法不同函数的应用和结果进行比较，有二阶常微分方程：

$$\frac{d^2u}{dx^2} + u + x = 0 \tag{2.28}$$

边界条件：

$$\begin{cases} \text{当} x = 0 \text{时}，\ u = 0 \\ \text{当} x = 1 \text{时}，\ u = 0 \end{cases} \tag{2.29}$$

取近似解为

$$u = x(1-x)(a_1 + a_2 x + \cdots) \tag{2.30}$$

其中，$a_1, a_2, \cdots$ 为待定参数，试探函数 $N_1 = x(1-x), N_2 = x(1-x)x, \cdots$ 显然，近似解满足边界条件式(2.29)，但不满足方程(2.28)，在域内将产生余量 R。余量的加权积分为零，

$$\int_0^1 v_i R dx = 0 \tag{2.31}$$

近似解可取式(2.30)中一项、两项或 n 项，项数取得越多，计算精度就越高。其中，一项近似解

为：

$$\tilde{u}_1 = a_1 x(1-x) \tag{2.32}$$

代入式(2.28)，余量为

$$R_1(x) = x + a_1(-2 + x - x^2) \tag{2.33}$$

两项近似解：$n=2$

$$\tilde{u}_2 = x(1-x)(a_1 + a_2 x) \tag{2.34}$$

余量为

$$R_2(x) = x + a_1(-2 + x - x^2) + a_2(2 - 6x + x^2 - x^3) \tag{2.35}$$

（1）配点法

一项近似解：取 $x = \frac{1}{2}$ 作为配点，代入式(2.32)得到

$$R_1(\frac{1}{2}) = \frac{1}{2} - \frac{7}{4}a_1 = 0 \quad \Rightarrow \quad a_1 = \frac{2}{7} \tag{2.36}$$

所以求得一项近似解为 $\tilde{u}_1 = \frac{2}{7}x(1-x)$

两项近似解：取 $x_1 = \frac{1}{3}$， $x_2 = \frac{2}{3}$ 作为配点分别代入式(2.36)中：

$$\begin{aligned} R_2(\frac{1}{3}) &= \frac{1}{3} - \frac{9}{16}a_1 + \frac{2}{27}a_2 \\ R_2(\frac{2}{3}) &= \frac{2}{3} - \frac{9}{16}a_1 - \frac{50}{27}a_2 \end{aligned} \tag{2.37}$$

解得 $a_1 = 0.1948, a_2 = 0.1731$，所以两项近似解为 $\tilde{u}_2 = x(1-x)(0.1948 + 0.1731x)$

（2）子域法

一项近似解：子域取全域，即 $v_1 = 1$。当 $0 \le x \le 1$，由式(2.31)得：

$$\int_0^1 R_1(x)dx = \int_0^1 [x + a_1(-2 + x - x^2)]dx = \frac{1}{2} - \frac{11}{6}a_1 = 0 \quad \Rightarrow \quad a_1 = \frac{3}{11} \tag{2.38}$$

求得一项近似解为 $\tilde{u}_1 = \frac{3}{11}x(1-x)$

两项近似解：分别取 $v_1 = 1$，当 $0 \le x \le \frac{1}{2}$ 时（Ω_1）； $v_2 = 1$，当 $\frac{1}{2} < x \le 1$ 时（Ω_2），由式(2.31)得：

$$\begin{aligned} \int_0^{\frac{1}{2}} R_2(x)dx &= \frac{1}{8} - \frac{11}{12}a_1 + \frac{53}{192}a_2 = 0 \\ \int_{\frac{1}{2}}^1 R_2(x)dx &= \frac{3}{8} - \frac{11}{12}a_1 + \frac{229}{192}a_2 = 0 \end{aligned} \tag{2.39}$$

解得 $a_1 = 0.1876, a_2 = 0.1702$，两项近似解为 $\tilde{u}_2 = x(1-x)(0.1876+0.1702x)$

（3）最小二乘法

将余量的二次方 R^2 在域 Ω 中积分

$$I = \int_\Omega R^2 d\Omega \tag{2.40}$$

设近似解的待定系数 a_i，使得余量在全域的积分制 I 达到极小值，因而：

$$\frac{\partial I}{\partial a_i} = 0 \quad (i = 1,2,\cdots,n) \tag{2.41}$$

对式(2.40)求导得：

$$\int_\Omega R \frac{\partial R}{\partial a_i} d\Omega = 0 \tag{2.42}$$

由此得到 n 个方程，用以求解 n 个待定参数 a_i。通过比较式(2.31)和式(2.42)可知，最小二乘法的权函数为

$$v_i = \frac{\partial R}{\partial a_i}, \quad (i = 1,2,\cdots,n) \tag{2.43}$$

一项近似解：

$$R_1(x) = x + a_1(-2 + x - x^2) \tag{2.44}$$

$$\frac{\partial R}{\partial a_i} = -2 + x - x^2 \tag{2.45}$$

代入式(2.42)得：

$$\int_0^1 R_1 \frac{\partial R_1}{\partial a_1} dx = \int_0^1 [x + a_1(-2 + x - x^2)](-2 + x - x^2) dx = 0 \tag{2.46}$$

解得 $a_1 = 0.2723$，一项近似解 $\tilde{u}_1 = 0.2723x(1-x)$

两项近似解：

$$R_2(x) = x + a_1(-2 + x - x^2) + a_2(2 - 6x + x^2 - x^3) \tag{2.47}$$

$$v_1 = \frac{\partial R_2}{\partial a_1} = -2x + x - x^2 \tag{2.48}$$

$$v_2 = \frac{\partial R_2}{\partial a_2} = 2 - 6x + x^2 - x^3 \tag{2.49}$$

代入式(2.42)得：

$$\int_0^1 R_2 \frac{\partial R_2}{\partial a_1} dx = \int_0^1 [x + a_1(-2 + x - x^2) + a_2(2 - 6x + x^2 - x^3)](-2 + x - x^2) dx = 0$$
$$\int_0^1 R_2 \frac{\partial R_2}{\partial a_2} dx = \int_0^1 [x + a_1(-2 + x - x^2) + a_2(2 - 6x + x^2 - x^3)](2 - 6x + x^2 - x^3) dx = 0 \tag{2.50}$$

解得 $a_1 = 0.1875, a_2 = 0.1695$，两项近似解为：$\tilde{u}_2 = x(1-x)(0.1875 + 0.1695x)$

（4）力矩法

一项近似解：取 $v_1 = 1$，由式(2.31)得

$$\int_0^1 1 \cdot R_1(x) dx = \int_0^1 [x + a_1(-2 + x - x^2)] dx = 0 \tag{2.51}$$

解得 $a_1 = \frac{3}{11}$，一项近似解：$\tilde{u}_1 = \frac{3}{11} x(1-x)$，此结果与子域法的结果相同

两项近似解：取 $v_1 = 1, v_2 = x$，代入式(2.31)得

$$\int_0^1 1 \cdot R_1(x) dx = \int_0^1 [x + a_1(-2 + x - x^2) + a_2(2 - 6x + x^2 - x^3)] dx = 0$$
$$\int_0^1 x \cdot R_1(x) dx = \int_0^1 x \cdot [x + a_1(-2 + x - x^2) + a_2(2 - 6x + x^2 - x^3)] dx = 0 \tag{2.52}$$

解得 $a_1 = 0.1880, a_2 = 0.1695$，两项近似解为 $\tilde{u}_2 = x(1-x)(0.1880 + 0.1695x)$。

（5）伽辽金法

取近似函数作为权函数，一项近似解：$\tilde{u}_1 = N_1 a_1 = a_1 x(1-x)$。

取权函数 $v_1 = N_1 = x(1-x)$，由式(2.31)得：

$$\int_0^1 x(1-x) \cdot R_1(x) dx = \int_0^1 x(1-x)[x + a_1(-2 + x - x^2)] dx = 0 \tag{2.53}$$

解得 $a_1 = \frac{5}{18}$，一项近似解：$\tilde{u}_1 = \frac{5}{18} x(1-x)$

两项近似解：$\tilde{u}_2 = N_1 a_1 + N_2 a_2 = a_1 x(1-x) + a_2 x^2(1-x)$。

取权函数 $v_1 = N_1 = x(1-x)$，$v_2 = N_2 = x^2(1-x)$ 代入式(2.31)得到

$$\int_0^1 x(1-x) \cdot R_1(x) dx = \int_0^1 x(1-x)[x + a_1(-2 + x - x^2) + a_2(2 - 6x + x^2 - x^3)] dx = 0$$
$$\int_0^1 x^2(1-x) \cdot R_1(x) dx = \int_0^1 x^2(1-x) \cdot [x + a_1(-2 + x - x^2) + a_2(2 - 6x + x^2 - x^3)] dx = 0 \tag{2.54}$$

解得 $a_1 = 0.1924, a_2 = 0.1707$，两项近似解为 $\tilde{u}_2 = x(1-x)(0.1924 + 0.1707x)$。

这个问题的精确解是 $u = \frac{\sin x}{\sin 1} - x$

用加权余量的几种方法所得到的近似解与精确解的比较如

表 2.1 所示。由表可见，在此类具体问题中所得到的两项近似解已经可以获得较好的近似效果，各种方法所得到的近似解误差均在 3%以内，其中伽辽金法的精度非常高，误差小于 5%。

表 2.1 不同加权余量方法的近似解与精确解结果比较

精确解 $u=\dfrac{\sin x}{\sin 1}-x$		$x=0.25$ $u=0.04401$		$x=0.5$ $u=0.06975$		$x=0.75$ $u=0.06006$	
近似解		值	误差/%	值	误差/%	值	误差/%
一项近似解	$\tilde{u}_1=a_1x(1-x)$						
	1. 配点：$\tilde{u}_1=0.2857x(1-x)$	0.05357	21.7	0.07143	2.4	0.05357	-10.8
	2. 子域：$\tilde{u}_1=0.2727x(1-x)$	0.05114	16.2	0.06818	-2.3	0.05114	-14.9
	3. 最小二乘：$\tilde{u}_1=0.2723x(1-x)$	0.05106	16.0	0.06808	-2.4	0.05106	-15.0
	4. 力矩：$\tilde{u}_1=0.2727x(1-x)$	0.05114	16.2	0.06818	-2.3	0.05114	-14.9
	5. 伽辽金：$\tilde{u}_1=0.2778x(1-x)$	0.05208	18.3	0.06944	-0.4	0.05208	-13.3
两项近似解	$\tilde{u}_2=a_1x(1-x)+a_2x^2(1-x)$						
	1. 配点：$\tilde{u}_2=x(1-x)(0.1948+0.1731x)$	0.04464	1.4	0.07034	0.8	0.06087	1.3
	2. 子域：$\tilde{u}_2=x(1-x)(0.1876+0.1702x)$	0.04315	-2.0	0.06818	-2.3	0.05911	-1.6
	3. 最小二乘：$\tilde{u}_2=x(1-x)(0.1875+0.1695x)$	0.04310	-2.1	0.06806	-2.4	0.05899	-1.8
	4. 力矩：$\tilde{u}_2=x(1-x)(0.1880+0.1695x)$	0.04320	-1.8	0.06819	-2.2	0.05909	-1.6
	5. 伽辽金：$\tilde{u}_2=x(1-x)(0.1924+0.1707x)$	0.04408	0.2	0.06944	-0.4	0.06008	0.03

显然如果增加近似解的项数，就会进一步提高精度。

下面开始讨论加权余量法在一维热传导问题中的应用。如果热传导系数取 1，则微分方程为

$$\mathrm{A}(\phi)=\frac{d^2\phi}{dx^2}+Q(x)=0 \quad (0\le x\le L) \tag{2.55}$$

其中

$$Q(x)=\begin{cases}1 & \text{当}0\le x\le L/2\\ 0 & \text{当}L/2<x\le L\end{cases} \tag{2.56}$$

边界条件是在 $x=0$ 和 $x=L$ 时，$\phi=0$。

取傅里叶级数作为近似解，即

$$\phi \approx \tilde{\phi} = \sum_{i=1}^{n} a_i \sin\frac{i\pi x}{L} \tag{2.57}$$

其中 a_i 为待定参数，试探函数 $N_i = \sin\frac{i\pi x}{L}$ 。近似解满足给定的边界条件，因此在边界上不产生余量，并因为此近似解在域内具有任意解的连续性，因此可直接用近似积分形式进行加权余量各种方法的计算。对方程(2.53)可以简单地表示为

$$\int_0^L v_j[\frac{d^2}{dx^2}(\sum_{i=1}^{n} N_i a_i) + Q]dx = 0 \tag{2.58}$$

现在分别利用配点法、子域法和伽辽金法进行求解，近似解分别取一项解($n=1$)及两项解($n=2$)。采用配点法时，一项解配点取 $x=\frac{L}{2}$， $Q(\frac{L}{2})$ 取两边的平均值 $\frac{1}{2}$。两项解得配点取 $x=\frac{L}{4}$ 及 $x=\frac{3L}{4}$ 。子域法中子域取两个半域 $0 \le x \le \frac{L}{2}$ 及 $\frac{L}{2} < x \le L$ 。求解过程留给读者作为练习。

值得指出的是：采用伽辽金法时，因为权函数 $v_j = N_j$ 是连续的，并在两端有 $N_i = 0$ 。可以对式(2.58)进行分部积分，得到近似积分弱形式

$$\int_0^L [\frac{dv_j}{dx}\frac{d}{dx}(\sum N_i a_i) - v_j Q]dx = 0 \qquad (j = 1,2,\cdots,n) \tag{2.59}$$

上式可改写成

$$\mathbf{Ka} - \mathbf{P} = \mathbf{0} \tag{2.60}$$

其中 $\mathbf{P} = \begin{bmatrix} P_1 \\ P_2 \\ \cdots \\ P_n \end{bmatrix}$， $\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \cdots \\ a_n \end{bmatrix}$， $K_{ij} = \int_0^L \frac{dv_j}{dx}\frac{dN_j}{dx}dx$， $P_i = \int_0^L v_j Q dx$ 。

可以看到当取 $v_j = N_i$ 时，将有 $K_{ij} = K_{ji}$，也就是说采用伽辽金法求解待定系数 a_i 的代数方程组的系数矩阵 $\mathbf{K}$ 是对称的。当方程阶数很高时，这种对称性将给计算带来很大方便。

对于二维热传导问题应用伽辽金法求解，首先求得二维稳态热传导问题的等效积分“弱”形式（2.14）.采用加权余量法求解时，近似解取 $\tilde{\phi} = \sum_{i=1}^{n} N_i a_i$ ，并设 $\tilde{\phi}$ 已事先满足 Γ_ϕ 边界上的边界条件。任意函数取 $\delta\tilde{\phi}$ ，并在 Γ_ϕ 边界上 $\delta\tilde{\phi} = 0$ 。在此情况下，从式(2.14)式可以得到如下方程

$$\mathbf{Ka} = \mathbf{p} \tag{2.61}$$

其中矩阵 $\mathbf{K}$ 和向量 $\mathbf{p}$ 的元素如下表示：

$$K_{ij} = K_{ji} = \int_\Omega \nabla^T N_j k \nabla N_i d\Omega = \int_\Omega k(\frac{\partial N_j}{\partial x}\frac{\partial N_i}{\partial x} + \frac{\partial N_j}{\partial y}\frac{\partial N_i}{\partial y})d\Omega \qquad (i,j = 1,2,\cdots,n) \tag{2.62}$$

$$p_i = \int_\Omega N_i Q d\Omega + \int_{\Gamma_q} N_i \bar{q} d\Gamma \qquad (i = 1,2,\cdots,n) \tag{2.63}$$

其中 $\mathbf{K}$ 称为热传导矩阵， $\mathbf{p}$ 称为热载荷向量。从上式可以看到 $\mathbf{K}$ 也是对称矩阵。

加权余量法可以广泛用于各类方程；对权函数进行不同的选择，就可以产生不同的加权余量法；通过采用等效积分的“弱”形式，对近似函数连续性的要求可以获得有效的降低。如果近似函数取自完全的函数系列，并满足连续性要求，在试探函数的项数不断增加的情况下，近似解可趋近于精确解。但解的收敛性仍未有严格的理论证明，同时近似解一般也不具有明确的上、下界性质。下节将会通过讨论变分原理和利兹方法来从理论上解决上述两方面的问题。

2.3 变分法与里兹法

如果微分方程具有线性和自伴随的性质，则不仅可以建立它的等效积分形式，并利用加权余量法求其近似解,还可以建立与之相等效的变分原理，并进而得到基于它的另一种近似求解方法，即里兹方法。

2.3.1 线性、自伴随微分方程变分原理的建立

1.线性、自伴随微分算子

若有微分方程

$$L(u)+b=0 \quad (\text{在}\Omega\text{域内}) \tag{2.64}$$

其中微分算子 L 具有如下性质：

$$L(\alpha u_1+\beta u_2)=\alpha L(u_1)+\beta L(u_2) \tag{2.65}$$

则称 L 为线性算子，方程(2.64)为线性微分方程。其中 α 和 β 是两个常数。

现定义 $L(u)$ 和任意函数的内积为

$$\int_\Omega L(u)v\,d\Omega \tag{2.66}$$

有也可表示为 $\langle L(u),v\rangle$。对上式进行分部积分直至 u 的导数消失。这样就可以得到转化后的内积并伴随有边界项。结果可表示如下：

$$\int_\Omega L(u)v\,d\Omega=\int_\Omega uL^*(v)\,d\Omega+b.t.(u,v) \tag{2.67}$$

上式右端 $b.t.(u,v)$ 表示在 Ω 的边界 Γ 上由 u 和 v 及其导数组成的积分项。算子 L^* 称为 L 的伴随算子。若 $L^*=L$ 则称算子是自伴随的。称原方程(2.66)为线性、自伴随的微分方程。

现证明算子 $L(\ )=-\dfrac{d^2(\ )}{dx^2}$ 是自伴随的。

构造内积，并进行分部积分

$$\begin{aligned}\int_{x_1}^{x_2} L(u)v\,dx &= \int_{x_1}^{x_2}(-\frac{d^2u}{dx^2})v\,dx = \int_{x_1}^{x_2}\frac{du}{dx}\frac{dv}{dx} - (\frac{du}{dx}v)\Big|_{x_1}^{x_2} \\ &= \int_{x_1}^{x_2}(-\frac{d^2v}{dx^2})u\,dx + (u\frac{dv}{dx})\Big|_{x_1}^{x_2} - (\frac{du}{dx}v)\Big|_{x_1}^{x_2}\end{aligned} \tag{2.68}$$

从上式可以看到 $L = L^*$,因此 L 是自伴随算子。

2.泛函的构造

原问题的微分方程和边界条件表达如下

$$\begin{aligned}&\mathbf{A}(\mathbf{u}) \equiv \mathbf{L}(\mathbf{u}) + \mathbf{f} = \mathbf{0} \quad &(\text{在}\Omega\text{内}) \\ &\mathbf{B}(\mathbf{u}) = \mathbf{0} \quad &(\text{在}\Gamma\text{上})\end{aligned} \tag{2.69}$$

和以上微分方程及边界条件相等效的伽辽金提法可表示如下

$$\int_\Omega \delta\mathbf{u}^T[\mathbf{L}(\mathbf{u}) + \mathbf{f}]d\Omega - \int_\Gamma \delta\mathbf{u}^T\mathbf{B}(\mathbf{u})d\Gamma = 0 \tag{2.70}$$

利用算子是线性、自伴随的，可以导出以下关系式

$$\begin{aligned}\int_\Omega \delta\mathbf{u}^T\mathbf{L}(\mathbf{u})d\Omega &= \int_\Omega [\frac{1}{2}\delta\mathbf{u}^T\mathbf{L}(\mathbf{u}) + \frac{1}{2}\delta\mathbf{u}^T\mathbf{L}(\mathbf{u})]d\Omega \\ &= \int_\Omega [\frac{1}{2}\delta\mathbf{u}^T\mathbf{L}(\mathbf{u}) + \frac{1}{2}\mathbf{u}^T\mathbf{L}(\delta\mathbf{u})]d\Omega + b.t.(\delta\mathbf{u},\mathbf{u}) \\ &= \int_\Omega [\frac{1}{2}\delta\mathbf{u}^T\mathbf{L}(\mathbf{u}) + \frac{1}{2}\mathbf{u}^T\delta\mathbf{L}(\mathbf{u})]d\Omega + b.t.(\delta\mathbf{u},\mathbf{u}) \\ &= \delta\int_\Omega \frac{1}{2}\mathbf{u}^T\mathbf{L}(\mathbf{u})d\Omega + b.t.(\delta\mathbf{u},\mathbf{u})\end{aligned} \tag{2.71}$$

将上式代人(2.70)式,就可得到原问题的变分原理

$$\delta\prod(u) = 0 \tag{2.72}$$

其中 $\prod(u) = \int_\Omega [\frac{1}{2}\mathbf{u}^T\mathbf{L}(\mathbf{u}) + \mathbf{u}^T\mathbf{f}]d\Omega + b.t.(\mathbf{u})$ 是原问题的泛函，因为此泛函中 $\mathbf{u}$ (包括 $\mathbf{u}$ 的导数)的最高次为二次，所以称为二次泛函。上式右端 $b.t.(\mathbf{u})$ 是由(2.69)式中的 $b.t.(\delta\mathbf{u},\mathbf{u})$ 项和(2.68)式中的边界积分项两部分组成。如果场函数 $\mathbf{u}$ 及其变分 $\delta\mathbf{u}$ 满足一定条件，则两部分合成后，能够形成一个全变分(即变分号提到边界积分项之外)，从而得到泛函的变分。这在后面将具体讨论。

由以上讨论可见，原问题的微分方程和边界条件的等效积分的伽辽金提法等效于它的变分原理，即原问题的微分方程和边界条件等效于泛函的变分等于零，亦即泛函取驻值。反之，如果泛函取驻值则等效于满足问题的微分方程和边界条件。而泛函可以通过原问题的等效积分的伽辽金法得到，并称这样得到的变分原理为自然变分原理。

3.泛函的极值性

如果线性自伴随算子 L 是偶数 $(2m)$ 阶的:在利用伽辽金法构造问题的泛函时，假设近似函数 $\tilde{\mathbf{u}}$

事先满足强制边界条件，对应于自然边界条件的任意函数 $\mathbf{v}$ 按一定的方法选取，就会得到泛函的变分。局时所构造的二次泛函不仅取驻值，而且是极值。现在开始阐述和讨论这个条件。

对于 $2m$ 阶微分方程，含 $0 \sim m-1$ 阶导数的边界条件称为强制边界条件，近似函数应事先满足。含 $m \sim 2m-1$ 阶导数的边界条件称为自然边界条件，近似函数不必事先满足。在伽辽金法中对应于此类边界条件，从含 $2m-1$ 阶导数的边界条件开始，任意函数 v 依次取 $-\delta\tilde{\mathbf{u}}$，$-\delta\dfrac{\partial\tilde{\mathbf{u}}}{\partial n}$，$-\delta\dfrac{\partial^2\tilde{\mathbf{u}}}{\partial n^2}$，…在此情况下，对原问题的伽辽金法进行 m 次分部积分后，通常得到如下形式的变分原理，即

$$\delta\prod(\mathbf{u})=0 \tag{2.73}$$

其中

$$\prod(u)=\int_\Omega[(-1)^m\mathbf{C}^T(\mathbf{u})\mathbf{C}(\mathbf{u})+\mathbf{u}^T\mathbf{f}]d\Omega+b.t.(\mathbf{u}) \tag{2.74}$$

$\mathbf{C}(\mathbf{u})$ 是 m 阶的线性算子，$b.t.(\mathbf{u})$ 是在自然边界上的积分项。

从上式可见，此时泛函中包括两部分，一是完全平方项 $\mathbf{C}^T(\mathbf{u})\mathbf{C}(\mathbf{u})$，另一是 $\mathbf{u}$ 的线性项，所以该二次泛函具有极值性，现还可进一步验证。

设近似场函数 $\tilde{\mathbf{u}}=\mathbf{u}+\delta\mathbf{u}$，其中 $\mathbf{u}$ 表示问题的真正解，$\delta\mathbf{u}$ 是它的变分。将此近似函数代入式(2.74)，就得到：

$$\prod(\tilde{\mathbf{u}})=\prod(\mathbf{u}+\delta\mathbf{u})=\prod(\mathbf{u})+\delta\prod(\mathbf{u})+\frac{1}{2}\delta^2\prod(\mathbf{u}) \tag{2.75}$$

其中，$\prod(\mathbf{u})$ 是真正解的泛函，$\delta\prod(\mathbf{u})$ 是原问题微分方程和边界条件的等效积分伽辽金法的弱形式，应有

$$\begin{aligned}&\delta\prod(\mathbf{u})=0\\&\frac{1}{2}\delta^2\prod(\mathbf{u})=\frac{1}{2}\int_\Omega(-1)^m\mathbf{C}^T(\delta\mathbf{u})\mathbf{C}(\delta\mathbf{u})d\Omega\end{aligned} \tag{2.76}$$

除非 $\delta\mathbf{u}=0$，即 $\tilde{\mathbf{u}}=\mathbf{u}$，亦即近似函数取问题的真正解，恒有 $\delta^2\prod(\mathbf{u})>0$（$m$ 为偶数)或恒有 $\delta^2\prod(\mathbf{u})<0$（$m$ 为奇数)。因此真正解使泛函取极值。

泛函的极值性对判断解的近似性质很有意义，利用它可以估计出解的上下界。

下面对二维热传导问题建立它的泛函并研究它的极值性，微分方程和边界条件已在式(2.3)和(2.4)给出。

问题的伽辽金法在 ϕ 已事先满足 Γ_ϕ 上强制条件情况下，可以表示如下

$$\int_\Omega\partial\phi(k\frac{\partial^2\phi}{\partial x^2}+k\frac{\partial^2\phi}{\partial y^2}+Q)d\Omega-\int_{\Gamma_q}\delta\phi(k\frac{\partial\phi}{\partial n}-\bar{q})d\Gamma=0 \tag{2.77}$$

经分部积分，得到它的弱形式，并注意到在 Γ_ϕ 上 $\partial\phi=0$，则有

$$\int_\Omega(-k\frac{\partial\delta\phi}{\partial x}\frac{\partial\phi}{\partial x}-k\frac{\partial\delta\phi}{\partial y}\frac{\partial\phi}{\partial y}+\partial\phi Q)d\Omega+\int_{\Gamma_q}\delta\phi\bar{q}d\Gamma=0 \tag{2.78}$$

从上式可以得到二维热传导问题的变分原理

$$\delta\prod(\phi)=0 \tag{2.79}$$

其中 $\prod(\phi)$ 是问题的泛函

$$\prod(\phi)=\int_\Omega[\frac{1}{2}k(\frac{\partial\phi}{\partial x})^2+\frac{1}{2}k(\frac{\partial\phi}{\partial y})^2-\phi Q]d\Omega-\int_{\Gamma_q}\phi\bar{q}d\Gamma \tag{2.80}$$

如以 $\tilde{\phi}=\phi+\delta\phi$ 代入上式，则得到

$$\prod(\tilde{\phi})=\prod(\phi)+\delta\prod(\phi)+\frac{1}{2}\delta^2\prod(\phi) \tag{2.81}$$

其中，$\prod(\tilde{\phi})$ 和 $\prod(\phi)$ 即是式(2.78)和式(2.80)表示的原问题精确解的泛函和它的变分(亦即伽辽金法的弱形式)，分别等于一确定的值和零。而二次变分项

$$\delta^2\prod(\tilde{\phi})=\int_\Omega[k(\frac{\partial\delta\phi^2}{\partial x})^2+k(\frac{\partial\delta\phi^2}{\partial y})^2]d\Omega\geq0 \tag{2.82}$$

上式只有在 $\delta\phi\equiv0$ 时，$\delta^2\prod(\tilde{\phi})$。因此原问题的精确解使泛函取极值(从式(2.78)得到的泛函的二次项前为负号，实际应用时，改为正号使泛函如式(2.78)).

再次指出，对于 $2m$ 阶线性、自伴随微分方程，通过伽辽金弱形式建立的变分原理，只有在近似场函数事先满足强制边界条件的情况下，才可能使泛函具有极值性。否则，只能使泛函取驻值而不是极值。表现在泛函的二次变分不恒大(或小)于等于零。

2.3.2 里兹方法

对于线性、自伴随微分方程在得到与它相等效的变分原理以后，可以用来建立求近似解的标准过程——里兹方法。具体步骤是：未知函数的近似解仍由一族带有待定参数的试探函数来近似表示，即

$$\mathbf{u}\approx\tilde{\mathbf{u}}=\sum_{i=1}^{n}\mathbf{N}_i\mathbf{a}_i=\mathbf{Na} \tag{2.83}$$

其中 $\mathbf{a}$ 是待定参数；$\mathbf{N}$ 是取自完全系列的已知函数。将式(2.82)代入问题的泛函 $\prod$,得到用试探函数和待定参数表示的泛函表达式。泛函的变分为零相当于将泛函对所包含的待定参数进行全微分，并令所得的方程等于零，即

$$\delta\prod=\frac{\partial\prod}{\partial\mathbf{a}_1}\partial\mathbf{a}_1+\frac{\partial\prod}{\partial\mathbf{a}_2}\partial\mathbf{a}_2+\cdots\frac{\partial\prod}{\partial\mathbf{a}_n}\partial\mathbf{a}_n=0 \tag{2.84}$$

由于$\partial\mathbf{a}_1$，$\partial\mathbf{a}_2$，…是任意的，满足上式时必然有$\frac{\partial\prod}{\partial\mathbf{a}_1}$，$\frac{\partial\prod}{\partial\mathbf{a}_2}$，…都等于零。因此可以得到一组方程为

$$\frac{\partial\prod}{\partial a}=\begin{Bmatrix}\frac{\partial\prod}{\partial\mathbf{a}_1}\\ \frac{\partial\prod}{\partial\mathbf{a}_2}\\ \cdots\\ \frac{\partial\prod}{\partial\mathbf{a}_n}\end{Bmatrix}=0 \tag{2.85}$$

这是与待定参数$\mathbf{a}$的个数相等的方程组,用以求解$\mathbf{a}$。这种求近似解的经典方法叫做里兹法。

如果在泛函$\prod$中$\mathbf{u}$和它的导数的最高方次是二次,则称泛函$\prod$为二次泛函。二次泛函运用于大量的工程和物理问题中，因此应该给予特别的注意。对于二次泛函，式(2.85)退化为一组线性方程。

$$\frac{\partial\prod}{\partial a}\equiv\mathbf{K}\mathbf{a}-\mathbf{P}=\mathbf{0} \tag{2.86}$$

很容易证明矩阵 K 是对称的。考虑向量$\frac{\partial\prod}{\partial a}$的变分可以得到：

$$\delta(\frac{\partial\prod}{\partial\mathbf{a}})=\begin{bmatrix}\frac{\partial}{\partial\mathbf{a}_1}(\frac{\partial\prod}{\partial\mathbf{a}_1})\delta\mathbf{a}_1+\frac{\partial}{\partial\mathbf{a}_2}(\frac{\partial\prod}{\partial\mathbf{a}_1})\delta\mathbf{a}_2+\cdots\\ \cdots\\ \frac{\partial}{\partial\mathbf{a}_1}(\frac{\partial\prod}{\partial\mathbf{a}_n})\delta\mathbf{a}_1+\frac{\partial}{\partial\mathbf{a}_2}(\frac{\partial\prod}{\partial\mathbf{a}_n})\delta\mathbf{a}_2+\cdots\end{bmatrix}\equiv\mathbf{K}_A\delta\mathbf{a} \tag{2.87}$$

很容易看出矩阵$\mathbf{K}_A$的子矩阵为：

$$\begin{aligned}\mathbf{K}_{Aij}&=\frac{\partial^2\prod}{\partial a_i\partial a_j}\\ \mathbf{K}_{Aji}&=\frac{\partial^2\prod}{\partial a_j\partial a_i}\end{aligned} \tag{2.88}$$

因此有

$$\mathbf{K}_{Aij}=\mathbf{K}_{Aji}^T \tag{2.89}$$

这就证明了矩阵$\mathbf{K}_A$是对称矩阵。

对于二次泛函，由式(2.86)可以得到：

$$\delta(\frac{\partial\prod}{\partial\mathbf{a}})=\mathbf{K}\delta\mathbf{a} \tag{2.90}$$

与式(2.87)比较就得到：

$$\mathbf{K}=\mathbf{K}_A \tag{2.91}$$

因此$\mathbf{K}$矩阵亦是对称矩阵。

由变分得到求解方程系数矩阵的对称性是一个极为重要的特性，它将给有限元法计算提供极大的便捷。

对于二次泛函，根据式(2.86)可以将近似泛函表示为

$$\prod = \frac{1}{2}\mathbf{a}^T\mathbf{K}\mathbf{a} - \mathbf{a}^T\mathbf{P} \tag{2.92}$$

上式的正确性用简单求导就可以证明。取上式泛函的变分

$$\delta\prod = \frac{1}{2}\delta\mathbf{a}^T\mathbf{K}\mathbf{a} + \frac{1}{2}\mathbf{a}^T\mathbf{K}\delta\mathbf{a} - \delta\mathbf{a}^T\mathbf{P} \tag{2.93}$$

由于矩阵$\mathbf{K}$的对称性,就有：

$$\delta\mathbf{a}^T\mathbf{K}\mathbf{a} = \mathbf{a}^T\mathbf{K}\delta\mathbf{a} \tag{2.94}$$

因此

$$\delta\prod = \delta\mathbf{a}^T(\mathbf{K}\mathbf{a} - \mathbf{P}) = \mathbf{0} \tag{2.95}$$

因为$\delta\mathbf{a}$是任意的，这样就得到式(2.96):

$$(\mathbf{K}\mathbf{a} - \mathbf{P}) = \mathbf{0} \tag{2.96}$$

里兹法的实质是从一族假定解中寻求满足泛函变分的“最好的”解。显然，近似解的精度与试探函数的选择有关。如果知道所求解的一般性质，那么可以通过选择反映此性质的试探函数来改进近似解，提高近似解的精度。若精确解恰巧包含在试探函数族中,则里兹法将得到精确解。

下面用里兹法求解微分方程(2.29)并做简单讨论：

$$\frac{d^2u}{dx^2} + u + x = 0 \tag{2.97}$$

边界条件：

$$\begin{cases} \text{当}x = 0\text{时}，\ u = 0 \\ \text{当}x = 1\text{时}，\ u = 0 \end{cases} \tag{2.98}$$

此为强制边界条件。

由于算子是线性自伴随的,可以立即利用变分原理，得到泛函为

$$\prod = \int_0^1 [-\frac{1}{2}(\frac{du}{dx})^2 + \frac{1}{2}u^2 + ux]dx \tag{2.99}$$

具体过程由读者自己完成。

选取两种试探函数形式，用里兹法求解。

（1）选取满足边界条件的一项多项式近似解（与加权余量法中选取的一项解相同）

$$\tilde{u} = a_1 x(1-x) \tag{2.100}$$

则有

$$\frac{d\tilde{u}}{dx}=a_1(1-2x) \tag{2.101}$$

代入式(2.99)得到用待定参数a_1表示的泛函为

$$\prod=\int_0^1[-\frac{1}{2}a_1^2(1-2x)^2+\frac{1}{2}a_1^2x^2(1-x)^2+a_1x^2(1-x)]dx=-\frac{3}{20}a_1^2+\frac{1}{12}a_1 \tag{2.102}$$

由泛函变分为零

$$\frac{\partial\prod}{\partial a_1}=0\Rightarrow a_1=\frac{5}{18} \tag{2.103}$$

所以近似解为

$$\tilde{u}=\frac{5}{18}x(1-x) \tag{2.104}$$

与伽辽金法求解的结果相同。证实了：当问题存在自然变分原理时，里兹法和伽辽金法所得的结果是相同的。

（2）取近似解为

$$\tilde{u}=a_1\sin x+a_2x \tag{2.105}$$

满足$x=0$时，$\tilde{u}=0$；但还要求$x=1$时，$\tilde{u}=0$，则应有

$$a_1\sin 1+a_2=0 \tag{2.106}$$

近似解为

$$\tilde{u}=a_1(\sin x-x\sin 1) \tag{2.107}$$

代入式（2.99）得到

$$\begin{aligned}\prod&=\int_0^1[-\frac{1}{2}a_1^2(\cos^2x-2\sin 1\cos x+\sin^2 1)\\&+\frac{1}{2}a_1^2(\sin^2x-2\sin 1\cdot x\cdot\sin x+x^2\sin^2 1)+a_1x(\sin x-x\sin 1)]dx\\&=-\frac{1}{2}a_1^2\sin 1(\frac{2}{3}\sin 1-\cos 1)+a_1(\frac{2}{3}\sin 1-\cos 1)\end{aligned} \tag{2.108}$$

$$\frac{\partial\prod}{\partial a_1}=(\frac{2}{3}\sin 1-\cos 1)(-a_1\sin 1+1)=0\Rightarrow a_1=\frac{1}{\sin 1} \tag{2.109}$$

近似解为

$$\tilde{u}=\frac{\sin x}{\sin 1}-x \tag{2.110}$$

由于所选用的试探函数族正好包含了问题的精确解，因此现在的里兹解就是精确解，即$\tilde{u}=u$。

一般地说，采用里兹法求解，当试探函数族的范围扩大以及待定参数的数目增多时，近似解的精度将会得到提高。

现在简要地介绍一下有关里兹法收敛性在理论上的结论。为便于讨论，假设未知场函数只是标量场ϕ,此时泛函$\prod(\phi)$有如下形式

$$\prod(\phi)=\int_{\Omega}F(\phi,\frac{\partial\phi}{\partial x},\frac{\partial\phi}{\partial y},\cdots)d\Omega+\int_{\Gamma}E(\phi,\frac{\partial\phi}{\partial x},\frac{\partial\phi}{\partial y},\cdots)d\Gamma \tag{2.111}$$

近似函数为

$$\phi\approx\tilde{\phi}=\sum_{i=1}^{n}N_i a_i \tag{2.112}$$

当 n 趋于无穷时，近似解ϕ收敛于微分方程精确解的条件如下。

(1)试探函数$N_1,N_2,\cdots N_n$应取自完备函数系列。满足此要求的试探函数称为是完备的。

(2)试探函数$N_1,N_2,\cdots N_n$应满足C_{m-1}连续性要求，即式(2.111)表示的泛函$\prod(\phi)$中场函数最高的微分阶数是m时，试探函数的$0\sim m-1$阶导数应是连续的，以保证泛函中的积分存在。满足此要求的试探函数称为是协调的。

若试探函数满足上述完备性和连续性的要求，则当$n\to\infty$时，$\tilde{\phi}\to\phi$，并且$\prod(\phi)$单调地收敛于$\prod(\phi)$,即泛函具有极值性。

由于里兹法以变分原理为基础，其收敛性有严格的理论基础；得到的求解方程的系数矩阵是对称的；而且在场函数事先满足强制边界条件(此条件通常不难实现)情况下，解通常具有明确的上、下界等性质。长期以来，近似解法在物理和力学的微分方程中占有很重要的位置，得到了很广泛的应用。但是由于它是在全求解域中定义试探函数，在实际应用中会遇到两方面的困难，即

(1)在求解域比较复杂的时候，选取满足边界条件的试探函数，通常会产生很难克服的困难。

(2)为了提高近似解的精度，需要增加待定参数，即增加试探函数的项数，这样就会增加求解的繁杂性。并且由于试探函数定义于全域，因此不可能根据问题的要求在求解域的不同部位对试探函数提出不同精度的要求，通常由于局部精度的要求使整个问题的求解增加很多的困难。

而同样是建立于变分原理基础上的有限元法，虽然在本质上和里兹法是类似的，但由于近似函数在子域(单元上)定义，因此可以克服上述两方面的困难；并和现代计算机技术相结合，成为对物理、力学以及其他广泛科学技术和工程领域实际问题进行分析和求解的有效工具，并得到愈来愈广泛的应用。

2.4 弹性力学的变分原理

2.4.1 虚功原理

变形体的虚功原理可以叙述如下：变形体中任意满足平衡的力系在任意满足协调条件的变形状态上作的虚功等于零，即体系外力的虚功与内力的虚功之和等于零。

虚功原理是虚位移原理和虚应力原理的总称。它们都可以被认为是与某些控制方程相等效的积分“弱”形式。虚位移原理是平衡方程和力的边界条件的等效积分“弱”形式；虚应力原理则是几何方程和位移边界条件的等效积分“弱”形式。

为了方便，我们使用张量符号推演，并将给出结果的矩阵表达形式。

1．虚位移原理

首先考虑平衡方程

$$\sigma_{ij,j}+\bar{f}_i=0 \quad （在V 内） \quad (i,j=1,2,3) \tag{2.113}$$

以及力的边界条件

$$\sigma_{ij}n_j+\bar{T}_i=0 \quad （在 S_\sigma 内） \quad (i,j=1,2,3) \tag{2.114}$$

可以利用式(2.1)建立与它们等效的积分形式，现在平衡方程相当于 $\mathbf{A(u)=0}$；力的边界条件相当于 $\mathbf{B(u)=0}$，权函数可不失一般地分别取真实位移的变分 δu_i,及其边界值(取负值),这样就可以得到与式(2.1)相当的等效积分

$$\int_V \delta u_i(\sigma_{ij,j}+\bar{f}_i)dV-\int_{S_\sigma}\delta u_i(\sigma_{ij}-\bar{T}_i)dS=0 \tag{2.115}$$

δu_i 是真实位移的变分，就意味着它是连续可导的，同时在给定位移的边界 S_u,上 $\delta u_i=0$。对上式体积分中的第 1 项进行分部积分.并注意到应力张量是对称张量，则可以得到

$$\begin{aligned}&\int_V \delta u_i\sigma_{ij,j}dV=\int_V(\delta u_i\sigma_{ij})_{,j}dV-\int_V\frac{1}{2}(\delta u_{i,j}+\delta u_{j,i})\sigma_{ij}dV\\&=-\int_V\frac{1}{2}(\delta u_{i,j}+\delta u_{j,i})\sigma_{ij}dV+\int_{S_\sigma}\delta u_i\sigma_{ij}n_jdS\end{aligned} \tag{2.116}$$

通过几何方程可见，式中 $\frac{1}{2}(\delta u_{i,j}+\delta u_{j,i})$ 表示的正是应变的变分，即虚应变 $\delta\varepsilon_{ij}$。以此表示代入，并将上式代回式(2.115)，就得到它经分部积分后的“弱”形式

$$\int_V(-\delta\varepsilon_{ij}\sigma_{ij}+\delta u_i\bar{f}_i)dV+\int_{S_\sigma}\delta u_i\bar{T}_idS=0 \tag{2.117}$$

上式体积分中的第一项是变形体内的应力在虚应变上所作之功，即内力的虚功；体积分中的第二项及面积分分别是体积力和面积力在虚位移上所做之功，即外力的虚功。外力的虚功和内力的虚

功的总和为零，这就是虚功原理。现在的虚功是外力和内力分别在虚位移和与之相对应的虚应变上所作之功，所以得到的是虚功原理中的虚位移原理。它是平衡方程和力的边界条件的等效积分“弱”形式。它的矩阵形式是

$$\int_V (-\delta\boldsymbol{\varepsilon}^T\boldsymbol{\sigma}+\delta\mathbf{u}\bar{\mathbf{f}})dV+\int_{S_\sigma}\delta\mathbf{u}\bar{\mathbf{T}}dS=0 \tag{2.118}$$

虚位移原理的力学意义是:如果力系(包括内力 $\boldsymbol{\sigma}$ 和外力 $\bar{\mathbf{f}}$ 及 $\bar{\mathbf{T}}$)是平衡的(即在内部满足平衡方程 $\sigma_{ij,j}+\bar{f}_i=0$，在给定外力边界 S_σ 上满足 $\sigma_{ij}n_j=\bar{T}_i$)，则它们在虚位移(在给定位移边界 S_u 上满足 $\delta u_i=0$)和虚应变(与虚位移相对应,即它们之间服从几何方程 $\delta\varepsilon_{ij}=\frac{1}{2}(\delta u_{i,j}+\delta u_{j,i})$)上所作之功的总和为零。反之，如果力系在虚位移(及虚应变)上所作之功的和等于零，则它们一定是满足平衡的。所以虚位移原理表述了力系平衡的必要而充分的条件。

应该指出，作为平衡方程和力边界条件的等效积分“弱”形式——虚位移原理的建立是以选择在内部连续可导(因而可以通过几何关系，将其导数表示为应变)和在 S_u 上满足位移边界条件的任意函数为条件的。如果任意函数不是连续可导的，尽管平衡方程和力边界条件的等效积分形式仍可建立，但不能通过分部积分建立其等效积分的“弱”形式。再如任意函数在 S_u 上不满足位移边界条件(现在的情况，即 S_u 上 $\delta u_i\neq 0$)，则总虚功应包括 S_u。上约束反力在 δu_i 上所作的虚功。

还应指出，在导出虚位移原理的过程中，未涉及物理方程(应力-应变关系)，所以虚位移原理不仅可以用于线弹性问题，而且可以用于非线性弹性及弹塑性等非线性问题。

2.虚应力原理

现在考虑几何方程式 $\varepsilon_{ij}=\frac{1}{2}(u_{i,j}+u_{j,i})$ 和位移边界条件式 $u_i=\bar{u}_i$ $(i=1,2,3)$ 在 S_u 上。它们分别相当于 $\mathbf{A}(\mathbf{u})=\mathbf{0}$ 和 $\mathbf{B}(\mathbf{u})=\mathbf{0}$。权函数可以分别取真实应力的变分 $\delta\sigma_{ij}$ 及其相应的边界值，δT_i，$\delta T_i=\delta\sigma_{ij}n_j$ 在边界 S_σ 上有 $\delta T_i=0$。这样构成与式(2.1)相当的等效积分是

$$\int_V\delta\sigma_{ij}[\varepsilon_{ij}-\frac{1}{2}(u_{i,j}+u_{j,i})]dV+\int_{S_u}\delta T_i(u_i-\bar{u}_i)dS=0 \tag{2.119}$$

对上式进行分部积分后可得

$$\int_V(\delta\sigma_{ij}\varepsilon_{ij}+u_i\delta\sigma_{ij,j})dV-\int_S\delta\sigma_{ij}n_ju_idS+\int_{S_u}\delta T_i(u_i-\bar{u}_i)dS=0 \tag{2.120}$$

由于 $\delta\sigma_{ij}$ 是真实应力的变分，它应满足平衡方程，即 $\delta\sigma_{ij,j}=0$ 并考虑到边界上 $\delta\sigma_{ij}n_j=\delta T_i$，且在给定力边界 S_σ 上 $\delta T_i=0$，所以上式可简化为

$$\int_V \delta\sigma_{ij}\varepsilon_{ij}dV - \int_{S_u} \delta T_i \bar{u}_i dS = 0 \tag{2.121}$$

上式第一项代表虚应力在应变上所作的虚功(相差一负号),第二项代表虚边界约束反力在给定位移上所作的虚功。为和前述内力和给定外力在虚应变和虚位移上所作的虚功相区别，这两项虚功，从力学意义上更准确地说应称之为余虚功。因此式(2.121)称之为余虚功原理，或虚应力原理。它的矩阵表达式形式是

$$\int_V \delta\boldsymbol{\sigma}^T\boldsymbol{\varepsilon}dV - \int_{S_u} \delta\mathbf{T}^T\bar{\mathbf{u}}dS = 0 \tag{2.122}$$

虚应力原理的力学意义是：如果位移是协调的(即在内部连续可导，因此满足几何方程，并在给定位移的边界S_u上等于给定位移)，则虚应力(在内部满足平衡方程，在给定外力边界S_σ上满足力的边界条件)和虚边界约束反力在它们上面所作之功的总和为零。反之，如果上述虚力系在它们上面所作之功的和为零，则它们一定是满足协调的。所以，虚应力原理表述了位移协调的必要而充分的条件。

和虚位移原理类似，虚应力原理的建立是以选择虚应力(在内部和力边界条件上分别满足平衡方程和力边界条件)作为等效积分形式的任意函数为条件的。否则作为几何方程和位移边界条件的等效积分形式在形式上应和现在导出的虚应力原理有所不同。

和虚位移原理相同，在导出虚应力原理过程中，同样未涉及物理方程。因此，虚应力原理同样可以应用于线弹性以及非线性弹性和弹塑性等不同的力学问题。但是应指出，无论是虚位移原理和虚应力原理，它们所依赖的几何方程和平衡方程都是基于小变形理论的，所以它们不能直接应用于基于大变形理论的力学问题。

2.4.2 线弹性力学的变分原理

弹性力学变分原理包括基于自然变分原理的最小位能原理和最小余能原理，以及基于约束变分原理的胡海昌-鹫津久广义变分原理和 Hellinger-Reissner 混合变分原理等。本章只讨论最小位能原理和最小余能原理。

1.最小位能原理

最小位能原理的建立可以从上节已建立的虚位移原理出发。后者的表达式是

$$\int_V (-\delta\varepsilon_{ij}\sigma_{ij} + \delta u_i \bar{f}_i)dV + \int_{S_\sigma} \delta u_i \bar{T}_i dS = 0 \tag{2.123}$$

其中的应力张量σ_{ij},如利用弹性力学的物理方程式$\sigma_{ij} = D_{ijkl}\varepsilon_{kl}$代入，则可得到

$$\int_V (-\delta\varepsilon_{ij}D_{ijkl}\varepsilon_{kl} + \delta u_i \bar{f}_i)dV + \int_{S_\sigma} \delta u_i \bar{T}_i dS = 0 \tag{2.124}$$

因为 D_{ijkl} 是对称张量，并利用单位体积能应变则有

$$(\delta\varepsilon_{ij})D_{ijkl}\varepsilon_{kl}=\delta(\frac{1}{2}D_{ijkl}\varepsilon_{ij}\varepsilon_{kl})=\delta U(\varepsilon_{mn}) \tag{2.125}$$

由此可见式(2.124)中体积分的第一项就是单位体积应变能的变分。

在线弹性力学中，假定体积力 $\bar{f}_i$ 和边界上面力 $\bar{T}_i$ 的大小和方向都是不变的，即可从位势函数中 $\phi(u_i)$ 和 $\varphi(u_i)$ 导出，则有

$$-\delta\phi(u_i)=\bar{f}_i\delta u_i \qquad -\delta\varphi(u_i)=\bar{T}_i\delta u_i \tag{2.126}$$

将式(2.125)和式(2.126)代人式(2.124)，就得到

$$\delta\prod_p=0 \tag{2.127}$$

其中

$$\begin{aligned}\prod_p&=\prod_p(u_i)=\int_V[U(\varepsilon_{ij})+\phi(u_i)]dV+\int_{S_\sigma}\psi(u_i)dS\\&=\int_V(\frac{1}{2}D_{ijkl}\varepsilon_{ij}\varepsilon_{kl}-\bar{f}_iu_i)dV-\int_{S_\sigma}\bar{T}_iu_idS\end{aligned} \tag{2.128}$$

其中，$\prod_p$ 是系统的总位能，它是弹性体变形位能和外力位能之和。式(2.127)表明：在所有区域内连续可导的(注：连续可导意指 $U(\varepsilon_{ij})$ 中的 ε_{ij} 能够通过几何方程式用 u_i 的导数表示)并在边界上满足给定位移条件可能位移中，真实位移使系统的总位能取驻值。还可以进一步证明在所有可能位移中，真实位移使系统总位能取最小值，因此式(2.127)所表述的称为最小位能原理。

证明最小位能原理是很方便的,以 u_i 表示真实位移，u_i^* 表示可能位移并令

$$u_i^*=u_i+\delta u \tag{2.129}$$

将它们分别代人总位能表达式(2.128)，则有

$$\begin{aligned}\prod_p(u_i)&=\int_V(U(\varepsilon_{ij})-\bar{f}_iu_i)dV-\int_{S_\sigma}\bar{T}_iu_idS\\\prod_p(u_i^*)&=\int_V(U(\varepsilon_{ij})-\bar{f}_iu_i^*)dV-\int_{S_\sigma}\bar{T}_iu_i^*dS\\&=\prod_p(u_i)+\delta\prod_p+\frac{1}{2}\delta^2\prod_p\end{aligned} \tag{2.130}$$

其中 $\delta\prod_p$，和 $\delta^2\prod_p$，分别是总位能的一阶和二阶变分。它们的具体表达式如下：

$$\begin{aligned}\delta\prod_p&=\int_V[\delta U(\varepsilon_{ij})-\bar{f}_i\delta u_i]dV-\int_{S_\sigma}\bar{T}_i\delta u_idS\\\frac{1}{2}\delta^2\prod_p&=\int_V U(\delta\varepsilon_{ij})dV=\int_V\frac{1}{2}D_{ijkl}(\delta\varepsilon_{ij})(\delta\varepsilon_{kl})dV\end{aligned} \tag{2.131}$$

由于u_i是真实位移，根据式(2.127)知道，$\prod_p$的一阶变分$\delta\prod_p$应为0。二阶变分$\delta^2\prod_p$式中只出现应变能函数。由于应变能是正定的，除非$\delta u_i \equiv 0$，则恒有

$$\delta^2\prod_p > 0 \tag{2.132}$$

因此有

$$\prod_p(u_i^*) \geq \prod_p(u_i) \tag{2.133}$$

上述等号只有当$\delta u_i \equiv 0$时，即可能位移就是真实位移时才成立。当$\delta u_i \neq 0$,即可能位移不是真实位移时，系统总位能总是大于取真实位移时系统的总位能。这就证明了最小位能原理。

2.最小余能原理:

最小余能原理的推导步骤和最小位能原理的推导类似，只是现在是从虚应力原理出发,作为几何方程和位移边界条件的等效积分“弱”形式的虚应力原理在 2.4.1 节中已经得到,表达如下

$$\int_V \delta\sigma_{ij}\varepsilon_{ij}dV - \int_{S_u}\delta T_i\bar{u}_i dS = 0 \tag{2.134}$$

将线弹性物理方程式$\varepsilon_{ij} = C_{ijkl}\sigma_{kl}$代入上式，即可得到

$$\int_V \delta\sigma_{ij}C_{ijkl}\sigma_{kl}dV - \int_{S_u}\delta T_i\bar{u}_i dS = 0 \tag{2.135}$$

同样C_{ijkl}也是对称张量，并已知余能表达式$V(\sigma_{mn}) = \frac{1}{2}C_{ijkl}\sigma_{ij}\sigma_{kl}$，所以上式体积分内被积函数就是余能的变分。这是因为

$$\delta\sigma_{ij}C_{ijkl}\sigma_{kl} = \delta(\frac{1}{2}C_{ijkl}\sigma_{ij}\sigma_{kl}) = \delta V(\sigma_{mn}) \tag{2.136}$$

而式(2.135)面积分内被积函数，在给定位移$\bar{u}_i$保持不变情况下是外力的余能。这样一来，式(2.135)可以表示为

$$\delta\prod_c = 0 \tag{2.137}$$

其中

$$\begin{aligned}\delta\prod_c &= \delta\prod_c(\sigma_{ij}) = \int_V V(\sigma_{mn})dV - \int_{S_u}T_i\bar{u}_i dS \\ &= \int_V \frac{1}{2}C_{ijkl}\sigma_{ij}\sigma_{kl}dV - \int_{S_u}T_i\bar{u}_i dS\end{aligned} \tag{2.138}$$

它是弹性体余能和外力余能的总和，即系统的总余能。式(2.137)表明，在所有在弹性体内满足平衡方程，在边界上：满足力的边界条件的可能应力小，真实的应力使系统的总余能取驻值。还可以用于明真实位移使系统总位能取最小值类同的步骤，证明在所有可能的应力中，真实应力使系统总余能取最小值，因此式(2.137)表述的是最小余能原理。

3.弹性力学变分原理的能量上、下界

由于最小位能原理和最小余能原理都是极值原理，它们可以给出能量的上界或下界，这对估计近似解的特性是有重要意义的。

将(2.129)和式(2.138)相加得到

$$\begin{aligned}&\Pi_p(u_i)+\Pi_c(\sigma_{ij})\\&=\int_V(\frac{1}{2}D_{ijkl}\varepsilon_{ij}\varepsilon_{kl}+\frac{1}{2}C_{ijkl}\sigma_{ij}\sigma_{kl})dV-\int_V\overline{f}_iu_idV-\int_{S_\sigma}\overline{T}_iu_idS-\int_{S_u}T_i\overline{u}_idS\\&=\int_V\sigma_{ij}\varepsilon_{ij}dV-\int_V\overline{f}_iu_idV-\int_{S_\sigma}\overline{T}_iu_idS-\int_{S_u}T_i\overline{u}_idS=0\end{aligned}\tag{2.139}$$

式中第一项体积分等于应变能的 2 倍，后三项积分(不包括负号)之和是外力功的 2 倍。根据能量平衡，应变能应等于外力功，因此得到弹性系统的总位能与总余能之和为零。现在用Π_p，Π_c表示取精确解时系统的总位能和总余能；Π_p^*，Π_c^*表示取近似解时系统的总位能和总余能，假定在几何边界S_u上给定位移$\overline{u}_i=0$，可以推得

$$\begin{aligned}\Pi_c&=\int_V\frac{1}{2}C_{ijkl}\sigma_{ij}\sigma_{kl}dV=\int_VV(\sigma_{ij})dV\\\Pi_p&=\int_V\frac{1}{2}D_{ijkl}\varepsilon_{ij}\varepsilon_{kl}dV-\int_V\overline{f}_iu_idV-\int_{S_\sigma}\overline{T}_iu_idS\end{aligned}\tag{2.140}$$

上式后两项积分(不包括负号)此时是外力功的 2 倍，因此总位能数值上等于弹性体系的总应变能，取负号，即

$$\Pi_p=-\int_V\frac{1}{2}D_{ijkl}\varepsilon_{ij}\varepsilon_{kl}dV=-\int_VU(\varepsilon_{ij})dV\tag{2.141}$$

由最小位能原理知道

$$\Pi_p^*\geq\Pi_p\tag{2.142}$$

则有

$$\int_VU(\varepsilon_{ij}^*)dV\leq\int_VU(\varepsilon_{ij})dV\tag{2.143}$$

由最小余能原理

$$\Pi_c^*\geq\Pi_c\tag{2.144}$$

则有

$$\int_VV(\varepsilon_{ij}^*)dV\leq\int_VV(\varepsilon_{ij})dV\tag{2.145}$$

上两式中ε_{ij}^*，σ_{ij}^*分别为取近似解时的应变场和应力场函数。由式(2.143)及(2.145)可见，利用最小位能原理求得位移近似解的弹性变形能是精确解变形能的下界，即近似的位移场在总体上偏小，

也就是说结构的计算模型显得偏于刚硬；而利用最小余能原理得到的应力近似解的弹性余能是精确解余能的上界，即近似的应力解在总体上偏大，结构的计算模型偏于柔软。当分别利用这两个极值原理求解同一问题时，我们将获得这个问题的上界和下界，可以较准确地估计所得近似解的误差，这对于工程计算具有实际意义。

第三章 平面三节点三角形单元

如图 3.1 所示的二维弹性体，假设在边界Γ_1上受到约束，同时在边界Γ_2上受到力的作用，那么这个物体将发生变形。变形后物体内任意一点的 x 方向和 y 方向位移用 u 和 v 来表示，应力和应变分量用σ_{xx}、σ_{yy}、τ_{xy}、ε_{xx}、ε_{yy}和γ_{xy}来表示。它们都是坐标的函数。

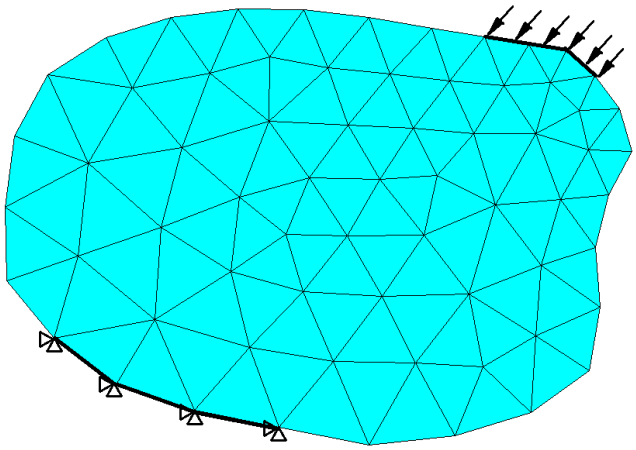


图 3.1 受力后发生变形的弹性体

根据几何方程，任意一点的应变可以表示为位移的函数。再根据物理方程，可以用应变来表示应力。只要知道了位移就可以求出所有的应变和应力，因此我们将位移作为基本未知量。由于很难通过求解基本方程得到位移的解析解，我们用分片函数来逼近精确解。

3.1 位移函数

将连续体离散为有限个三角形单元，如图 3.1 所示。每个单元由三个节点构成，如图 3.2 所示。三个节点按逆时针方向排序为 i、j、m。三个节点的 x 方向和 y 方向的位移分别定义为u_i、u_j、u_m、v_i、v_j、v_m。

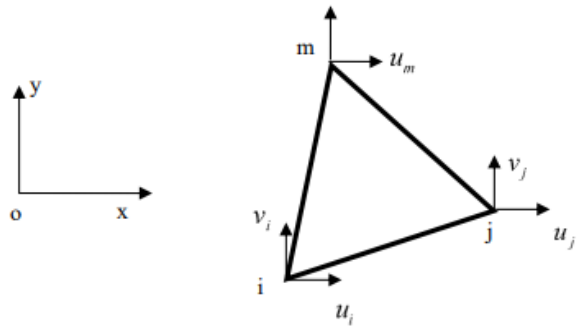


图 3.2 三角形单元

如果能够将单元内的位移表示成节点位移的函数，那么只要节点位移确定了，单元内的位移也就确定。为了达到这个目的，我们首先将位移表示成多项式的形式。

$$\begin{cases} u(x,y)=\alpha_1+\alpha_2 x+\alpha_3 y \\ v(x,y)=\alpha_4+\alpha_5 x+\alpha_6 y \end{cases} \tag{3.1}$$

其中α_1～α_6是待定系数。只要将α_1～α_6表示成节点位移的形式，就可以实现“将单元内的位移表示成节点位移的函数”的目的。因此，将α_1～α_6当作未知数，然后把节点位移和节点坐标代入方程(3.1)后得

$$\begin{cases} u_i=\alpha_1+\alpha_2 x_i+\alpha_3 y_i \\ v_i=\alpha_4+\alpha_5 x_i+\alpha_6 y_i \end{cases} \qquad (i,j,m) \tag{3.2}$$

方程(3.2)中x_i和y_i是节点的坐标，脚标 i,j,m 可轮换，可形成六个方程。这个方程组可以采用克莱姆法则来求解。假设有一个线性方程组，如(3.3)所示。

$$\begin{cases} a_{11}x_1+a_{12}x_2+\cdots+a_{1n}x_n=b_1 \\ a_{21}x_1+a_{22}x_2+\cdots+a_{2n}x_n=b_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ a_{n1}x_1+a_{n2}x_2+\cdots+a_{nn}x_n=b_n \end{cases} \tag{3.3}$$

其中a_{ij}是系数，x_i是未知数。根据克莱姆法则，当系数行列式$|A|$不等于零的时候，方程(3.3)有唯一解。

$$x_i=\frac{|A_i|}{|A|} \tag{3.4}$$

其中A_i是将A中第 j 列元素替换为右端项而得到的行列式。

因此，我们可以将α_1～α_6表示成如下形式，

$$\alpha_1=\frac{\begin{vmatrix}u_i & x_i & y_i\\ u_j & x_j & y_j\\ u_m & x_m & y_m\end{vmatrix}}{2\Delta}\quad \alpha_2=\frac{\begin{vmatrix}1 & u_i & y_i\\ 1 & u_j & y_j\\ 1 & u_m & y_m\end{vmatrix}}{2\Delta}\quad \alpha_3=\frac{\begin{vmatrix}1 & x_i & u_i\\ 1 & x_j & u_j\\ 1 & x_m & u_m\end{vmatrix}}{2\Delta}\quad \alpha_4=\frac{\begin{vmatrix}v_i & x_i & y_i\\ v_j & x_j & y_j\\ v_m & x_m & y_m\end{vmatrix}}{2\Delta}\quad \alpha_5=\frac{\begin{vmatrix}1 & v_i & y_i\\ 1 & v_j & y_j\\ 1 & v_m & y_m\end{vmatrix}}{2\Delta}\quad \alpha_6=\frac{\begin{vmatrix}1 & x_i & v_i\\ 1 & x_j & v_j\\ 1 & x_m & v_m\end{vmatrix}}{2\Delta}$$

(3.5)

其中

$$2\Delta=\begin{vmatrix}1 & x_i & y_i\\ 1 & x_j & y_j\\ 1 & x_m & y_m\end{vmatrix}$$

为了描述方便，引入系数

$$\begin{cases}a_i=x_jy_m-x_my_j\\ a_j=x_my_i-x_iy_m\\ a_m=x_iy_j-x_jy_i\end{cases}\quad\begin{cases}b_i=y_j-y_m\\ b_j=y_m-y_i\\ b_m=y_i-y_j\end{cases}\quad\begin{cases}c_i=-x_j+x_m\\ c_j=-x_m+x_i\\ c_m=-x_i+x_j\end{cases}\tag{3.6}$$

采用循环指标来表示，可以将 a，b，c 表示为：

$$\begin{cases}a_i=x_jy_m-x_my_j\\ b_i=y_j-y_m\\ c_i=-x_j+x_m\end{cases}\qquad(i,\ j,\ m)\tag{3.7}$$

根据(3.4)和(3.6)式，可以将$\alpha_1\sim\alpha_6$写成如下表达式

$$\begin{aligned}&\alpha_1=\frac{1}{2\Delta}\sum a_iu_i,\quad \alpha_2=\frac{1}{2\Delta}\sum b_iu_i\\ &\alpha_3=\frac{1}{2\Delta}\sum c_iu_i,\quad \alpha_4=\frac{1}{2\Delta}\sum a_iv_i\\ &\alpha_5=\frac{1}{2\Delta}\sum b_iv_i,\quad \alpha_6=\frac{1}{2\Delta}\sum c_iv_i\end{aligned}\tag{3.8}$$

这样，我们就完成了将单元内的位移表示为节点位移和坐标的函数。

需要注意的是，这一节我们采用多项式来构造位移函数。事实上，我们还可以采用其他形式的函数来构造位移函数。但是不论采用什么函数，都应该满足如下条件：

（1）完备性-要求位移函数应该包含常应变项和刚体位移项

如果在势能泛函中出现的位移函数的最高阶导数是 m 阶，则选取的位移函数至少是 m 阶完全多项式。

（2）协调性-相邻单元公共边界保持位移连续

如果在势能泛函中所出现的位移函数的最高阶导数是 m 阶，则位移函数在单元交界面上必须具有直至（m-1）阶连续导数，即 C_{m-1} 连续性。

根据有限元理论，弹性势能泛函中出现的位移函数的最高阶导数是 1 阶，所以选择的位移函至少是 1 阶完全多项式。对比方程(3.1)我们知道，本节的位移函数满足完备性要求。我们来证明一下该位移模式包含了常应变项和刚体位移项目。

根据几何方程以及方程(3.1)，三角形单元内部的应变如下：

$$\varepsilon_{xx}=\frac{\partial u}{\partial x}=\alpha_2,\varepsilon_{yy}=\frac{\partial v}{\partial y}=\alpha_6,\gamma_{xy}=\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}=\alpha_3+\alpha_5 \tag{3.9}$$

可见各应变均为常数项，满足“位移函数应该包含常应变项”的要求。我们再来看看位移函数是否包含刚体位移项目。

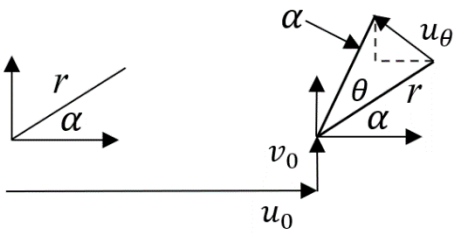


图 3.3 线段的刚体位移

如图 3.3 所示，一个线段 *r* 发生刚体位移，产生平动（u_0， v_0）和转动 θ。那么 r 点的 x 方向位移 $u=u_0-u_\theta\sin\alpha=u_0-r\theta\sin\alpha=u_0-y\theta$。

同理 r 点的 y 方向位移 $v=v_0+u_\theta\cos\alpha=v_0+r\theta\cos\alpha=v_0+x\theta$，所以

$$\begin{aligned}u&=u_0-y\cdot\theta\\v&=v_0+x\cdot\theta\end{aligned} \tag{3.10}$$

这就是刚体位移的位移模式。当应变为零时，三节点三角形单元的位移模式如下：

$$\begin{aligned}u&=\alpha_1-\frac{\alpha_5-\alpha_3}{2}y=u_0-\theta_0 y\\v&=\alpha_4+\frac{\alpha_5-\alpha_3}{2}x=v_0+\theta_0 x\end{aligned} \tag{3.11}$$

对比(3.10)和(3.11)可知，该位移模式包含刚体位移，因此本节的位移函数满足完备性要求。我们将在下节证明该位移函数也满足协调性要求。

3.2 形函数

将方程(3.8)代入(3.2)后，即可将位移表达为节点位移的函数。另一方面，我们发现，位移函数中节点位移的系数具有如下表达式：

$$N_i=\frac{1}{2\Delta}(a_i+b_i x+c_i y) \tag{3.12}$$

因此我们可以将位移函数写成如下表达式：

$$\begin{cases}u=N_i u_i+N_j u_j+N_m u_m\\v=N_i v_i+N_j v_j+N_m v_m\end{cases} \tag{3.13}$$

式中 *Ni*、*Nj*、*Nm* 称为单元的形状函数，简称形函数或插值函数。

为了便于计算和表达，我们将将方程(3.13)写成矩阵形式：

$$\{f\}=\begin{Bmatrix}u\\v\end{Bmatrix}=\begin{bmatrix}N_i & 0 & N_j & 0 & N_m & 0\\0 & N_i & 0 & N_j & 0 & N_m\end{bmatrix}\begin{Bmatrix}u_i\\v_i\\u_j\\v_j\\u_m\\v_m\end{Bmatrix} \tag{3.14}$$

简写为

$$\{f\}=[N]\{\delta\}^e \tag{3.15}$$

其中矩阵$[N]$反映了单元的位移形态，是坐标的函数，称为形函数矩阵。

需要注意的是，*Ni*、*Nj*、*Nm* 三个形函数具有如下性质：

（1）形函数 *Ni* 在结点 i 处的值为 1，而在其他两个结点（j，m）处的值为零。其他两个型函数也有类似的性质，即：

$$\begin{aligned}&N_i(x_i,y_i)=1\text{ 而 }N_i(x_j,y_j)=N_i(x_m,y_m)=0\\&N_j(x_j,y_j)=1\text{ 而 }N_j(x_i,y_i)=N_j(x_m,y_m)=0\\&N_m(x_m,y_m)=1\text{ 而 }N_m(x_i,y_i)=N_m(x_j,y_j)=0\end{aligned} \tag{3.16}$$

（2）在单元任一点处，三个形函数之和等于 1。

$$\begin{aligned}&N_i(x,y)+N_j(x,y)+N_m(x,y)=\\&\frac{1}{2\Delta}\left(a_i+b_ix+c_iy+a_j+b_jx+c_jy+a_m+b_mx+c_my\right)=\\&\frac{1}{2\Delta}\left[\left(a_i+a_j+a_m\right)+\left(b_i+b_j+b_m\right)x+\left(c_i+c_j+c_m\right)y\right]=\\&\frac{1}{2\Delta}\left(2\Delta+0+0\right)=1\end{aligned} \tag{3.17}$$

现在我们证明位移函数协调性。如图 3.4 所示两个相邻的单元 1 和 2。1 号单元的结点为 i、j、m，2 号单元的结点为 j、k、m。

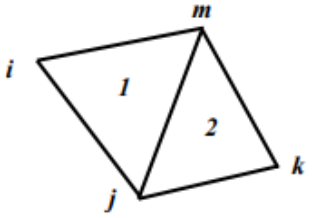


图 3.4 一对相邻单元

根据方程(3.12)，1 号单元和 2 号单元的位移分别表示为：

$$\begin{cases} u_1 = N_i u_i + N_j u_j + N_m u_m \\ v_1 = N_i v_i + N_j v_j + N_m v_m \end{cases} \tag{3.18}$$

$$\begin{cases} u_2 = N_k u_k + N_j u_j + N_m u_m \\ v_2 = N_k v_k + N_j v_j + N_m v_m \end{cases} \tag{3.19}$$

我们只要证明在边界 j-m 上，$u_1 = u_2$ 且 $v_1 = v_2$，那么位移函数就是 C_0 连续的。根据（3.16）所示的形函数性质，在边界 j-m 上 N_i=0，并且 N_k=0。因此

$$\begin{aligned} u_1 &= N_j u_j + N_m u_m = u_2 \\ v_1 &= N_j v_j + N_m v_m = v_2 \end{aligned} \tag{3.20}$$

由此可见位移函数在边界上是 C_0 连续的。

3.3 单元刚度矩阵

参照杆单元有限元法的计算步骤，建立单元刚度方程是有限元极为重要的一个环节。单元刚度矩阵定义了单元节点位移和节点力之间的关系。三角形单元刚度方程具有如下形式：

$$[K]^e \{\delta\}^e = \{F\}^e \tag{3.21}$$

其中：$\{\delta\}^e$ 和 $\{F\}^e$ 分别是单元节点力及节点位移向量，$[K]^e$ 是单元刚度矩阵。

$$\{F\}^e = \begin{bmatrix} F_i & F_j & F_m \end{bmatrix}^T = \begin{bmatrix} F_{ix} & F_{iy} & F_{jx} & F_{jy} & F_{mx} & F_{my} \end{bmatrix}^T \tag{3.22}$$

$$\{\delta\}^e = \begin{bmatrix} \delta_i & \delta_j & \delta_m \end{bmatrix}^T = \begin{bmatrix} u_i & v_i & u_j & v_j & u_m & v_m \end{bmatrix}^T \tag{3.23}$$

根据弹性力学知识，几何方程为：

$$\{\varepsilon\} = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = \begin{Bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial u}{\partial y} + \dfrac{\partial v}{\partial x} \end{Bmatrix} \tag{3.24}$$

将插值函数形式的常应变三角形单元位移函数(3.13)代入(3.24)后得：

$$\{\varepsilon\} = \frac{1}{2\Delta} \begin{bmatrix} b_i u_i + b_j u_j + b_m u_m \\ c_i v_i + c_j v_j + c_m v_m \\ c_i u_i + c_j u_j + c_m u_m + b_i v_i + b_j v_j + b_m v_m \end{bmatrix} \tag{3.25}$$

将节点位移提取出来后得：

$$\{\varepsilon\}=\frac{1}{2\Delta}\begin{bmatrix} b_i & 0 & b_j & 0 & b_m & 0 \\ 0 & c_i & 0 & c_j & 0 & c_m \\ c_i & b_i & c_j & b_j & c_m & b_m \end{bmatrix}\begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \\ u_m \\ v_m \end{Bmatrix} \tag{3.26}$$

令：$[B]=\begin{bmatrix} B_i & B_j & B_m \end{bmatrix}$ 且 $[B_i]=\frac{1}{2\Delta}\begin{bmatrix} b_i & 0 \\ 0 & c_i \\ c_i & b_i \end{bmatrix}$，$(i,j,m)$，方程(3.26)可简化为：

$$\{\varepsilon\}=[B]\{\delta\}^e \tag{3.27}$$

$[B]$矩阵称为单元的几何矩阵，其物理意义反映了单元内任意一点的应变与单元结点之间的关系。对于一个给定的单元，几何形状确定，结点的坐标也是一定的，系数b_i、c_i也随之确定，Δ也为常数，所以几何矩阵是常量矩阵，表明 3 节点三角形单元是一种常应变单元。

根据方程(1.72)和(1.73)，对于平面问题，不论是平面应力问题还是平面应变问题，都可以将应力应变关系写成如下矩阵形式：

$$\{\sigma\}=[D]\{\varepsilon\} \tag{3.28}$$

其中$\{\sigma\}=\begin{bmatrix}\sigma_{xx},\sigma_{yy},\tau_{xy}\end{bmatrix}^T$，$\{\varepsilon\}=\begin{bmatrix}\varepsilon_{xx},\varepsilon_{yy},\gamma_{xy}\end{bmatrix}^T$。

对于平面应力问题

$$[D]=\frac{E}{1-\mu^2}\begin{bmatrix} 1 & \mu & 0 \\ \mu & 1 & 0 \\ 0 & 0 & \dfrac{1-\mu}{2} \end{bmatrix} \tag{3.29}$$

对于平面应变问题

$$[D]=\frac{E(1-\mu)}{(1+\mu)(1-2\mu)}\begin{bmatrix} 1 & \dfrac{\mu}{1-\mu} & 0 \\ \dfrac{\mu}{1-\mu} & 1 & 0 \\ 0 & 0 & \dfrac{1-2\mu}{2(1-\mu)} \end{bmatrix}=\frac{E_1}{1-\mu_1^2}\begin{bmatrix} \mu_1 & 1 & 0 \\ 0 & 0 & \dfrac{1-\mu_1}{2} \end{bmatrix} \tag{3.30}$$

其中 $E_1=\dfrac{E}{1-\mu^2}$　　$\mu_1=\dfrac{\mu}{1-\mu}$。

将方程(3.27)代入(3.28)得

$$\{\sigma\}=[D][B]\{\delta\}^e \tag{3.31}$$

定义

$$[S]=[D][B] \tag{3.32}$$

则

$$\{\sigma\}=[S]\{\delta\}^e \tag{3.33}$$

其中[S]称为单元的应力矩阵。[S]矩阵反映了单元中任一点的应力与结点位移之间的关系。对于3 结点三角形单元，[D]，[B]为常量矩阵，故[S]也为常量矩阵。这种常应变单元，也是一种常应力单元。

建立刚度方程的常用方法有（1）直接刚度法、（2）虚位移原理或最小势能原理（适用于位移型有限元）、（3）余虚功原理或最小余能原理（力型）有限元、（4）变分法（非结构问题）。本文采用虚位移原理建立单元刚度矩阵。

设某单元发生一虚位移，则该单元各结点上的虚位移为$\{\delta^*\}^e$，相应地单元内任一点处的虚应变为$\{\varepsilon^*\}$。根据应变与结点位移间的关系有：

$$\{\varepsilon^*\}=[B]\{\delta^*\}^e \tag{3.34}$$

这时单元体在结点力作用下处于平衡状态。根据虚位移原理，当虚位移发生时，结点力在虚位移上所做的虚功等于单元的虚应变能，即：

$$\{\delta^*\}^{eT}\{F\}^e=\int_{V^e}\{\varepsilon^*\}^T\{\sigma\}dV \tag{3.35}$$

式中 V^e 为单元的体积，上式称为单元的虚功方程。

将单元应力与结点位移之间的关系和虚应变和结点虚位移之间的关系代入，得到

$$\{\delta^*\}^{eT}\{F\}^e=\int_{V^e}\left([B]\{\delta^*\}^e\right)^T\left([D][B]\{\delta\}^e\right)dV \tag{3.36}$$

由于结点位移及结点虚位移均为常量，提到积分号外，有

$$\{\delta^*\}^{eT}\{F\}^e=\{\delta^*\}^{eT}\int_{V^e}[B]^T[D][B]dV\{\delta\}^e \tag{3.37}$$

由于结点虚位移的任意性，进一步可得：

$$\{F\}^e = \int_{V^e} [B]^T [D][B] dV \{\delta\}^e \tag{3.38}$$

令：$$[K]^e = \int_{V^e} [B]^T [D][B] dV \tag{3.39}$$

则上式可写为$$\{F\}^e = [K]^e \{\delta\}^e \tag{3.40}$$

这就求得了我们所需要形式的方程，称为单元刚度方程。式中$[K]^e$称为单元刚度矩阵，反映了结点力与结点位移之间的关系。

单元刚度矩阵具有如下性质：

（1）单元刚度矩阵是对称矩阵

（2）单元刚度矩阵的主对角元素恒为正值

（3）单元刚度矩阵为奇异阵

（4）单元刚度仅与单元的几何特性（几何矩阵[B]）及材料特性（弹性矩阵[D]）有关，而与单元的受力状况无关。

上述四条性质，与杆系的单元刚度性质相同。

3.4 等效节点载荷

根据方程(3.31)，方程左边的载荷向量实际上是节点上受到外力，为集中载荷。但是我们通常会遇到其他类型的载荷，如面载荷、体积载荷、以及热膨胀产生的热应力。这些载荷R_{my}在有限元方程中怎么反应呢。本节就要介绍等效节点载荷的计算。

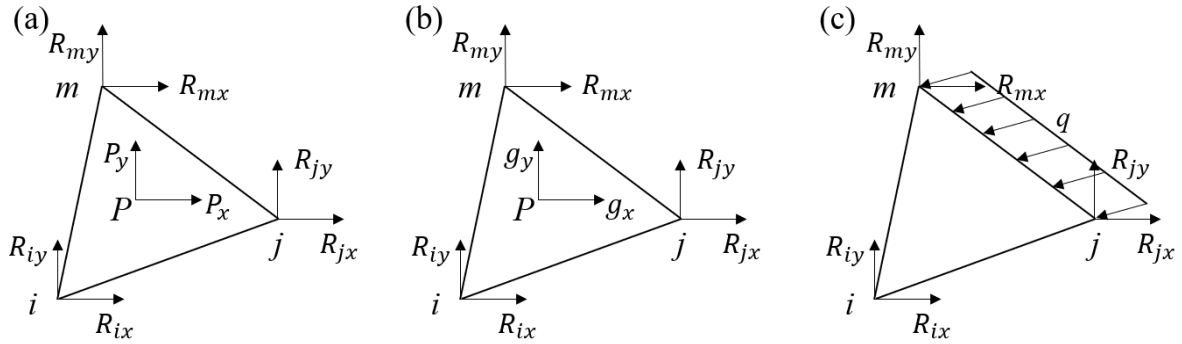


图 3.5 三角形单元的等效节点载荷（a）集中力；（b）体积力；（c）表面力

如图 3.5（a）所示，在三角形单元内任意一点作用一个集中载荷 P，其 x 和 y 方向分量分别是P_x和P_y。与 P 等效的结点载荷为$\mathbf{R} = \left[R_{ix}, R_{iy}, R_{jx}, R_{jy}, R_{mx}, R_{my}\right]^T$。

假设在三个结点处发生了虚位移$\left\{\delta^*\right\}^e = \left[u_i^*, v_i^*, u_j^*, v_j^*, u_m^*, v_m^*\right]^T$。由结点虚位移产生的，在 P 点的虚位移可以用位移函数(3.13)计算：

$$\left\{f^*\right\}=\begin{Bmatrix}\delta u\\ \delta v\end{Bmatrix}=\begin{bmatrix}N_i & 0 & N_j & 0 & N_m & 0\\ 0 & N_i & 0 & N_j & 0 & N_m\end{bmatrix}\begin{Bmatrix}u_i^*\\ v_i^*\\ u_j^*\\ v_j^*\\ u_m^*\\ v_m^*\end{Bmatrix} \tag{3.41}$$

根据能量相等原则，原集中力与单元的等效结点载荷在相应的虚位移上所作的虚功相等。有

$$\left\{\delta^*\right\}^{eT}\left\{R\right\}^e=\left\{f^*\right\}^T\left\{P\right\} \tag{3.42}$$

将方程(3.41)代入(3.42)后得

$$\left[u_i^*,v_i^*,u_j^*,v_j^*,u_m^*,v_m^*\right]\cdot\begin{bmatrix}R_{ix}\\ R_{iy}\\ R_{jx}\\ R_{jy}\\ R_{mx}\\ R_{my}\end{bmatrix}=\left[u_i^*,v_i^*,u_j^*,v_j^*,u_m^*,v_m^*\right]\cdot\begin{bmatrix}N_i & 0\\ 0 & N_i\\ N_j & 0\\ 0 & N_j\\ N_m & 0\\ 0 & N_m\end{bmatrix}\cdot\begin{bmatrix}P_x\\ P_y\end{bmatrix} \tag{3.43}$$

因为上述方程在任意虚位移下都必须成立，因此方程左右两边虚位移的系数必须相等，所以有

$$\begin{bmatrix}R_{ix}\\ R_{iy}\\ R_{jx}\\ R_{jy}\\ R_{mx}\\ R_{my}\end{bmatrix}=\begin{bmatrix}N_i & 0\\ 0 & N_i\\ N_j & 0\\ 0 & N_j\\ N_m & 0\\ 0 & N_m\end{bmatrix}\cdot\begin{bmatrix}P_x\\ P_y\end{bmatrix} \tag{3.44}$$

方程(3.44)建立了单元内任意集中载荷与节点力之间的等效关系式。

体积力、表面力的等效结点载荷也可以通过虚功相等计算出来，步骤与集中载荷相同。留给读者自己推导，本书只给出最终结论。假设$\{g\}=\left[g_x,g_y\right]^T$是三角形单元内单位体积受到的体积力。其等效节点载荷可由(3.45)计算：

$$\left\{R\right\}^e=t\iint_\Delta\left[N\right]^T\cdot\left\{g\right\}dxdy \tag{3.45}$$

其中 t 表示三角形单元的厚度。

如图 3.5（c）所示，三角形单元的 jm 边受到面载荷$\{q\}=\left[q_x,q_y\right]^T$的作用。那么其等效节点载荷可由(3.46)计算：

$$\left\{R\right\}^e=t\int_{l_{jm}}\left[N\right]^T\cdot\left\{q\right\}ds \tag{3.46}$$

计算其他边界上分布载荷的等效节点载荷时，只要将(3.46)式中的积分区域换成 l_{ij},l_{mi} 即可。

在高温环境下，温度的改变通常会引起巨大的热应力。因此由于温度改变而引起的载荷是不容

忽视的。设弹性体的初始温度为 T_1 ，受热后温度升至 T_2，则弹性体的温度改变为 $T=T_2-T_1$ 。在平面问题中，T_1 和 T_2 均为 x、y 的函数。因此，温度改变 T 亦为 x、y 的函数。

若弹性体内各点不受任何约束，由于温度改变 T，将发生正应变 αT ，其中 α 为线膨胀系数。对于各向同性体，温度改变产生的正应变在各个方向均相同，不产生剪应变。因此，在平面应力状态下，$\varepsilon_{x0}=\varepsilon_{y0}=\alpha T$，而 $\gamma_{xy0}=0$。即由于温度改变产生的初应变列阵为

$$\{\varepsilon_0\}=\begin{bmatrix}\alpha T & \alpha T & 0\end{bmatrix}^T=\alpha T\begin{bmatrix}1 & 1 & 0\end{bmatrix}^T \tag{3.47}$$

对三角形常应变单元，如果令 3 个节点处的温度改变分别为 T_i、T_j、T_m，则单元的温度改变 T 可由 3 个节点处的温度改变插值求出，即

$$T=N_iT_i+N_jT_j+N_mT_m \tag{3.48}$$

当考虑温度变化时，平面问题的物理方程为

$$\{\sigma\}=[D](\{\varepsilon\}-\{\varepsilon_0\})=[D][B]\{\delta\}^e-[D]\{\varepsilon_0\} \tag{3.49}$$

其中 $\{\varepsilon\}$ 为单元中任一点的总应变，$\{\varepsilon_0\}$ 为该点的热应变。根据虚功方程，

$$\{\delta^*\}^{eT}\{F\}^e=\int_{V^e}\{\varepsilon^*\}^T\{\sigma\}dV \tag{3.50}$$

$$得\{\delta^*\}^{eT}\{F\}^e=\int_{V^e}\{\varepsilon^*\}^T\left([D][B]\{\delta\}^e-[D]\{\varepsilon_0\}\right)dV \tag{3.51}$$

将方程(3.34)代入上式后得

$$\{\delta^*\}^{eT}\{F\}^e=\int_{V^e}\{\delta^*\}^{eT}[B]^T\left([D][B]\{\delta\}^e-[D]\{\varepsilon_0\}\right)dV \tag{3.52}$$

由于上式在任一节点位移下都必须成立，因此系数必须相等：

$$\{F\}^e=[K]^e\{\delta\}^e-\int_{V^e}[B]^T[D]\{\varepsilon_0\}dV \tag{3.53}$$

$$令\{R_t\}^e=\int_{V^e}[B]^T[D]\{\varepsilon_0\}dV \tag{3.54}$$

上述方程即为考虑了温度改变的单元刚度方程，式中 $\{R_t\}^e$ 是单元温度改变的等效节点载荷。

3.5 导出有限元方程

有限元法本质上是求解弹性力学基本方程的一种近似方法。根据最小位能原理，在弹性范围内，平面问题中的弹性力学基本方程及边界条件与如下方程是等价的。

$$\Pi=\int_\Omega\frac{1}{2}\{\varepsilon\}^T\cdot[D]\cdot\{\varepsilon\}tdxdy-\int_\Omega\{f\}^T\cdot\{g\}tdxdy-\int_{\Gamma 2}\{f\}^T\cdot\{q\}tds \tag{3.55}$$

其中，Ω 是研究对象所占区域，Γ_2 是研究对象的力边界，t 是二维体厚度；$\{g\}$ 是作用在二维体内

的体积力；$\{q\}$ 是作用在 Γ_2 上的面积力。如图 3.1 所示，当结构离散化后，根据积分的性质，方程（3.55）中的积分可以表示为各个单元内积分之和：

$$\Pi=\sum_{e=1}^{N_e}\int_{\Omega^e}\frac{1}{2}\{\varepsilon\}^T\cdot[D]\cdot\{\varepsilon\}tdxdy-\sum_{e=1}^{N_e}\int_{\Omega^e}\{f\}^T\cdot\{g\}tdxdy-\sum_{e=1}^{N_e}\int_{\Gamma_2^e}\{f\}^T\cdot\{q\}tds \tag{3.56}$$

其中 Ω^e 是第 e 个单元所占区域，Γ_2^e 是第 e 个单元的力边界。将(3.15)和(3.27)代入(3.56)后消去 $\{\varepsilon\}$ 和 $\{f\}$ 后得：

$$\begin{aligned}\Pi=&\sum_{e=1}^{N_e}\{\delta\}^{eT}\cdot\int_{\Omega^e}\frac{1}{2}[B]^T\cdot[D]\cdot[B]tdxdy\cdot\{\delta\}^e\\&-\sum_{e=1}^{N_e}\{\delta\}^{eT}\cdot\int_{\Omega^e}[N]^T\cdot\{g\}tdxdy-\sum_{e=1}^{N_e}\{\delta\}^{eT}\int_{\Gamma_2^e}[N]^T\cdot\{q\}tds\end{aligned} \tag{3.57}$$

为了便于计算，我们将所有的节点位移排列成一个向量：

$$\{\delta\}^T=\left[u_1,v_1,u_2,v_2\cdots u_i,v_i,\cdots u_{Ne},v_{Ne}\right]^T \tag{3.58}$$

单元节点位移 $\{\delta\}^{eT}$ 实际上是从 $\{\delta\}^T$ 抽取六个元素构成的一个向量。我们可以通过抽取矩阵 $[G]^e_{6\times 2Ne}$ 建立 $\{\delta\}^T$ 与 $\{\delta\}^{eT}$ 之间的关系：

$$\{\delta\}^e=[G]^e_{6\times 2Ne}\cdot\{\delta\} \tag{3.59}$$

将方程(3.59)代入(3.57)消去 $\{\delta\}^e$ 后得：

$$\begin{aligned}\Pi=&\sum_{e=1}^{N_e}\{\delta\}^T\cdot[G]^{eT}\cdot\int_{\Omega^e}\frac{1}{2}[B]^T\cdot[D]\cdot[B]tdxdy\cdot[G]^e\cdot\{\delta\}\\&-\sum_{e=1}^{N_e}\{\delta\}^T\cdot[G]^{eT}\cdot\int_{\Omega^e}[N]^T\cdot\{g\}tdxdy-\sum_{e=1}^{N_e}\{\delta\}^T\cdot[G]^{eT}\cdot\int_{\Gamma_2^e}[N]^T\cdot\{q\}tds\end{aligned} \tag{3.60}$$

令

$$\begin{aligned}[K]^e&=\int_{\Omega^e}[B]^T\cdot[D]\cdot[B]tdxdy \quad \left[P_g\right]^e=\int_{\Omega^e}[N]^T\cdot\{g\}tdxdy\\\left[P_q\right]^e&=\int_{\Gamma_2^e}[N]^T\cdot\{q\}tds \qquad [P]^e=\left[P_g\right]^e+\left[P_q\right]^e\end{aligned} \tag{3.61}$$

其中 $[K]^e$ 和 $[P]^e$ 分别是单元刚度矩阵和单元等效节点载荷向量。

进一步地，令

$$[K]=\sum_{e=1}^{Ne}[G]^{eT}\cdot[K]^e\cdot[G]^e \quad [P]=\sum_{e=1}^{Ne}[G]^{eT}\cdot[P]^e \tag{3.62}$$

其中$[K]$是总体刚度矩阵，$[P]$是整体载荷向量。方程表(3.60)示成如下形式：

$$\Pi=\frac{1}{2}\{\delta\}^T\cdot[K]\cdot\{\delta\}-\{\delta\}^T\cdot[P] \tag{3.63}$$

根据最小位能原理，结构平衡时的位移，应该是满足位移边界条件，并且能够使泛函Π取极小值。由据变分原理可知，泛函Π取极值的条件是它的一次变分等于零，即

$$\frac{\partial\Pi}{\partial\{\delta\}}=[K]\cdot\{\delta\}-[P]=0 \tag{3.64}$$

这样，我们就得到了有限元方程：

$$[K]\cdot\{\delta\}-[P]=0 \tag{3.65}$$

3.6 总体刚度矩阵和载荷向量的合成

实际上，有限元计算的主要过程都是围绕方程(3.65)展开的。为了求解方程(3.65)，首先要构造$[K]$和$[P]$。根据方程(3.62)可知，$[K]$和$[P]$分别由单元刚度矩阵和单元载荷向量合成而来。我们可以通过方程(3.62)来计算$[K]$和$[P]$，但这种方法需要构造大量的$[G]^e$矩阵，耗费大量内存。因此比较合理的方法是采用本书中 1.3.3 节方法直接合成$[K]$和$[P]$。即遍历所有单元的刚度矩阵和载荷向量，根据元素的脚标的含义，叠加到总体刚度矩阵和总体载荷向量的相应位置。程序实现可参阅本教材 1.3.3、1.3.4 和 3.2.8 节。

需要注意的是，方程(3.65)有无穷多组解。要获得唯一解必须在方程中施加位移约束。常用的方法有划去法、对角元置一法和乘大数法。对角元置一法和乘大数法在许多有限元教材中都有讲述，本文不再介绍。

3.7 应力计算

对(3.65)式施加边界条件后，求解计算得到所有的节点位移。由结构刚度方程解出节点位移 $\{\delta\}$后，就得到了各单元的节点位移$\{\delta\}^e$。利用单元内任一点应变、应力与节点位移间的关系，就可计算出单元中任一点处的应变与应力。

当不考虑温度影响时，单元中任一点的应变为：

$$\begin{aligned}
\varepsilon_x&=\frac{1}{2\Delta}(b_iu_i+b_ju_j+b_mu_m)\\
\varepsilon_y&=\frac{1}{2\Delta}(c_iv_i+c_jv_j+c_mv_m)\\
\gamma_{xy}&=\frac{1}{2\Delta}(b_iv_i+b_jv_j+b_mv_m+c_iu_i+c_ju_j+c_mu_m)
\end{aligned} \tag{3.66}$$

由弹性力学，平面应力状态下单元中任一点的应力为

$$\sigma_x = \frac{E}{1-\mu^2}(\varepsilon_x + \mu\varepsilon_y)$$
$$\sigma_y = \frac{E}{1-\mu^2}(\varepsilon_y + \mu\varepsilon_x) \tag{3.67}$$
$$\tau_{xy} = \frac{E}{2(1+\mu)}\gamma_{xy}$$

将（3.66）代入（3.67）

$$\sigma_x = \frac{E}{2\Delta(1-\mu^2)}[(b_i u_i + b_j u_j + b_m u_m) + \mu(c_i v_i + c_j v_j + c_m v_m)]$$
$$\sigma_y = \frac{E}{2\Delta(1-\mu^2)}[(c_i v_i + c_j v_j + c_m v_m) + \mu(b_i u_i + b_j u_j + b_m u_m)] \tag{3.68}$$
$$\tau_{xy} = \frac{E}{4\Delta(1+\mu)}(c_i u_i + c_j u_j + c_m u_m + b_i v_i + b_j v_j + b_m v_m)$$

根据上式就可由单元节点位移求出单元中任一点的应力。

3.8 平面三节点三角形单元的 C 语言实现

```
#include "stdio.h"//该函数库包含了文件输入输出函数
#include "stdlib.h"//该函数库包含了动态内存分配函数
#ifndef MatMax//定义了最大材料数目
#define MatMax 100
#endif
struct Data//用于存储数据
{
    int Model;//0~RVE 分析,1~结构分析
    int Method;//0 有限元;1GMC;2 边界元;3 混合法;
    int Nm;//材料类型总数
    int Dime;//维数
    int nNode;//局部单元节点数。
    char ElementName[50];
    double E[MatMax],Po[MatMax];//
};
//
int *p_ele2dt=NULL;//p_ele2dt[3*i+0,1,2]，四节点等参元时 p_ele2dt[4*i+0,1,2，3]
double *p_node2d=NULL;//p_node2d[6*i+0,1,2,3,4,5],x,y,ux,uy,fx,fy
```

int *p_node2dplot=NULL,*p_nodeuvid=NULL;//p_node2d[2*i+0,1],x,y

int N_elemt,N_node,N_nodes;//分别是单元总数，节点总数，自由度总数

double X1,Y1,X2,Y2,l,h,a;//节点坐标范围

double m_E,m_mu;//杨氏模量和泊松比

Data *pdata=NULL;//指向 Data 的指针

bool isread_tri=false;//布尔变量，当等于 false 时，表示还未读取数据，ture 表示已读取

double ux_max,ux_mid,ux_min,uy_max,uy_mid,uy_min,uz_max,uz_mid,uz_min;//最大和最小位移，中间位移。

double *pu=NULL;//存储位移的向量，在 solve 函数中分配空间。

void mx_printf(void * p_mx,char p_name[],int Ni,int Nj,int tpye_mx,char *filename)//用于将向量 p_max 输出到文件 filename 中；p_name[]存储字符串，用于说明矩阵的意义；Ni 和 Nj 表示矩阵 p_mx 的行数和列数；tpye_mx 表示矩阵的类型 1 为整数矩阵，2 为浮点矩阵。

```
{
    int i,j;
    int *p_mxint;
    double *p_mxdouble;
    FILE *pf;
    pf=fopen(filename,"w");
    fprintf(pf,"%s\n",p_name);
    switch(tpye_mx)
    {
    case 1:
        p_mxint=(int *)p_mx;
        for(i=0;i<Ni;i++)
        {
            for(j=0;j<Nj;j++)
            {
                fprintf(pf,"%d ",p_mxint[i+Ni*j]);
            }
            fprintf(pf,"\n");
```

```
            }
        case 2:
            p_mxdouble=(double *)p_mx;
            for(i=0;i<Ni;i++)
            {
                for(j=0;j<Nj;j++)
                {
                    fprintf(pf,"%e ",p_mxdouble[i+Ni*j]);
                }
                fprintf(pf,"\n");
            }
        }
        fclose(pf);
        return;
    }
    int mx_multipG(double * mx_a,double * mx_b,double * mx_c,int Nai,int Naj,int Nbi,int Nbj)//普通乘法，实现 mx_a*mx_b=mx_c，Nai, Naj,Nbi,Nbj 分别是矩阵 mx_a 和 mx_b 的行数和列数
    {
        int i,j,k;
        //如果两相乘矩阵不满足相乘条件，则返回 1
        if(Naj!=Nbi)
            return 1;
        for(i=0;i<Nai;i++)
        {
            for(j=0;j<Nbj;j++)
            {
                mx_c[i+Nai*j]=0;
                for(k=0;k<Naj;k++)
                {
                    mx_c[i+Nai*j]=mx_c[i+Nai*j]+mx_a[i+Nai*k]*mx_b[k+Nbi*j];
```

```
                }
            }
        }
        return 0;
}
int mx_inver(double *mx_a,int N)//对矩阵 mx_a 求逆，求逆后存于 mx_a 中，N 是矩阵的阶数
{
        int i,j,m,k,Maxm,flag;
        int *p_i,*p_j;
        double D,S;
        p_i=(int *)malloc(2*N*sizeof(int));
        p_j=(int *)malloc(2*N*sizeof(int));
        if(p_i==NULL||p_j==NULL)
        {
            printf("mx_inver malloc p_i or p_j fail!\n");
            return 1;
        }
        m=0;
        flag=0;
        for(k=0;k<N;k++)
        {
            //全选主元
            D=mx_a[k+N*k]*mx_a[k+N*k];
            for(i=k;i<N;i++)
            {
                for(j=k;j<N;j++)
                {
                    if(mx_a[i+N*j]*mx_a[i+N*j]>D)
                    {
                        D=mx_a[i+N*j]*mx_a[i+N*j];
```

```
                p_i[2*m]=k;    p_i[2*m+1]=i; p_j[2*m]=k;   p_j[2*m+1]=j;
                flag=1;
            }
        }
    }
    //如果 D 接近 0 则矩阵奇异
    if(D<1e-40)
    {
        printf("Matirx is sigular!\n");
        return 2;
    }
    if(flag)
    {
        //交换
        for(i=0;i<N;i++)
        {
            S=mx_a[i+N*p_j[2*m+1]];   mx_a[i+N*p_j[2*m+1]]=mx_a[i+N*k];
            mx_a[i+N*k]=S;
        }
        ////////////
        ///////
        for(j=0;j<N;j++)
        {
            S=mx_a[p_i[2*m+1]+N*j];   mx_a[p_i[2*m+1]+N*j]=mx_a[k+N*j];
            mx_a[k+N*j]=S;
        }
        ////////////
        m++; flag=0;
    }
    //bianhuang
```

```
mx_a[k+N*k]=1/mx_a[k+N*k];
//第 k 行
for(j=0;j<k;j++)
{
    mx_a[k+N*j]=mx_a[k+N*k]*mx_a[k+N*j];
}
for(j=k+1;j<N;j++)
{
    mx_a[k+N*j]=mx_a[k+N*k]*mx_a[k+N*j];
}
//行列 11
for(i=0;i<k;i++)
{
    for(j=0;j<k;j++)
    {
        mx_a[i+N*j]=mx_a[i+N*j]-mx_a[i+N*k]*mx_a[k+N*j];
    }
}
//行列 12
for(i=0;i<k;i++)
{
    for(j=k+1;j<N;j++)
    {
        mx_a[i+N*j]=mx_a[i+N*j]-mx_a[i+N*k]*mx_a[k+N*j];
    }
}
//行列 21
for(i=k+1;i<N;i++)
{
    for(j=0;j<k;j++)
```

```
            {
                mx_a[i+N*j]=mx_a[i+N*j]-mx_a[i+N*k]*mx_a[k+N*j];
            }
        }
        //行列 22
        for(i=k+1;i<N;i++)
        {
            for(j=k+1;j<N;j++)
            {
                mx_a[i+N*j]=mx_a[i+N*j]-mx_a[i+N*k]*mx_a[k+N*j];
            }
        }
        //第 k 列
        for(i=0;i<k;i++)
        {
            mx_a[i+N*k]=-mx_a[i+N*k]*mx_a[k+N*k];
        }
        for(i=k+1;i<N;i++)
        {
            mx_a[i+N*k]=-mx_a[i+N*k]*mx_a[k+N*k];
        }
    }
    Maxm=m;
    //还原
    for(m=Maxm-1;m>=0;m--)
    {
        //行变换
        for(j=0;j<N;j++)
        {
            S=mx_a[p_j[2*m]+N*j];
```

```
            mx_a[p_j[2*m]+N*j]=mx_a[p_j[2*m+1]+N*j];
            mx_a[p_j[2*m+1]+N*j]=S;
        }
        //列交换
        for(i=0;i<N;i++)
        {
            S=mx_a[i+N*p_i[2*m]];
            mx_a[i+N*p_i[2*m]]=mx_a[i+N*p_i[2*m+1]];
            mx_a[i+N*p_i[2*m+1]]=S;
        }
    }
    return 0;
}
bool FEM_ReadData(char *szFile)//用于从文件 szFile 中读取数据，并将单元编号和节点坐标分别存储到 p_ele2dt、p_node2d 中
{
    double temp;
    int i,j;
    FILE *pf=fopen(szFile,"r");
    fscanf(pf,"%lf",&m_E); fscanf(pf,"%lf",&m_mu);    fscanf(pf,"%lf",&temp);
    N_elemt=(int)temp;    fscanf(pf,"%lf",&temp); N_node=(int)temp;
    if(p_ele2dt!=NULL)
    {
        free(p_ele2dt);
    }
    if(p_node2d!=NULL)
    {
        free(p_node2d);
    }
    if(p_node2dplot!=NULL)
```

```
{
    free(p_node2dplot);
}
p_ele2dt=(int *)malloc(sizeof(int)*3*N_elemt);
p_node2d=(double *)malloc(sizeof(double)*6*N_node);
p_node2dplot=(int *)malloc(sizeof(int)*2*N_node);
//读取单元数据
for(i=0;i<N_elemt;i++)
{
    for(j=0;j<3;j++)
    {
        fscanf(pf,"%lf",&temp);
        p_ele2dt[3*i+j]=(int)temp-1;
    }
}
//读取节点数据
X1=Y1=X2=Y2=0;
bool isread=false;
for(i=0;i<N_node;i++)
{
    for(j=0;j<6;j++)
    {
        fscanf(pf,"%lf",&p_node2d[6*i+j]);
    }
    if(!isread)
    {
        X1=X2=p_node2d[6*i]; Y1=Y2=p_node2d[6*i+1];
        isread=true;
    }
    if(X1>p_node2d[6*i])
```

```
        {
            X1=p_node2d[6*i];
        }
        if(X2<p_node2d[6*i])
        {
            X2=p_node2d[6*i];
        }
        if(Y1>p_node2d[6*i+1])
        {
            Y1=p_node2d[6*i+1];
        }
        if(Y2<p_node2d[6*i+1])
        {
            Y2=p_node2d[6*i+1];
        }
    }
    l=X2-X1;h=Y2-Y1;
    if(l==0||h==0)
    {
        printf("Node location ERROR!\n");
    }
    fclose(pf);
    //输出
    pf=fopen("Databack.txt","w");
    fprintf(pf,"%d %d\n",N_elemt,N_node);
    for(i=0;i<N_elemt;i++)
    {
        for(j=0;j<3;j++)
        {
            fprintf(pf,"%d ",p_ele2dt[3*i+j]);
```

```
            }
            fprintf(pf,"\n");
        }
        //输出节点数据
        for(i=0;i<N_node;i++)
        {
            fprintf(pf,"%lf %lf %e %e %lf %lf\n", p_node2d[6*i+0], p_node2d[6*i+1], p_node2d[6*i+2],
p_node2d[6*i+3], p_node2d[6*i+4], p_node2d[6*i+5]);
        }
        fclose(pf);
        return true;
    }
    bool FEM_ElementMatrix2dt(double *pelematrix,double *pload,int id)//三节点三角形单元刚度矩阵
和单元节点载荷向量，pelematrix 是单元刚度矩阵的指针，外部分配空间，id 是单元编号，从 0 开始
    {
        //B'*D*B
        double B[18],D[9],Bt[18],DB[18];
        int i,j;
        double E,mu;
        double a[3],b[3],c[3],x[3],y[3],dlt;
        //
        E=m_E;  mu=m_mu;
        //清零
        for(i=0;i<18;i++)
        {
            B[i]=Bt[i]=0;
        }
        for(i=0;i<9;i++)
        {
            D[i]=0;
```

```
}
//填写 D 矩阵,按列排
D[0]=D[4]=E/(1-mu*mu); D[1]=D[3]=mu*E/(1-mu*mu); D[8]=0.5*(1-mu)*E/(1-mu*mu);
//填写 B 矩阵
for(i=0;i<3;i++)
{
    x[i]=p_node2d[6*p_ele2dt[3*id+i]];
    y[i]=p_node2d[6*p_ele2dt[3*id+i]+1];
}
a[0]=x[1]*y[2]-x[2]*y[1];    b[0]=y[1]-y[2];    c[0]=-x[1]+x[2];
a[1]=x[2]*y[0]-x[0]*y[2];    b[1]=y[2]-y[0];    c[1]=-x[2]+x[0];
a[2]=x[0]*y[1]-x[1]*y[0];    b[2]=y[0]-y[1];    c[2]=-x[0]+x[1];
dlt=0.5*(x[1]*y[2]+x[0]*y[1]+x[2]*y[0]-y[0]*x[1]-x[0]*y[2]-y[1]*x[2]);
B[0]=b[0];    B[2]=c[0];    B[4]=c[0];    B[5]=b[0];
B[6]=b[1];    B[8]=c[1];    B[10]=c[1];    B[11]=b[1];
B[12]=b[2];    B[14]=c[2];    B[16]=c[2];    B[17]=b[2];
//
for(i=0;i<3;i++)
{
    for(j=0;j<6;j++)
    {
        B[i+3*j]=B[i+3*j]/2/dlt;
        Bt[j+6*i]=B[i+3*j];
    }
}
mx_multipG(D,B,DB,3,3,3,6);
mx_multipG(Bt,DB,pelematrix,6,3,3,6);
for(i=0;i<36;i++)
{
    pelematrix[i]=dlt*pelematrix[i];
```

```
        }
        /////
        for(i=0;i<3;i++)
        {
            pload[2*i]=p_node2d[6*p_ele2dt[3*id+i]+4];//fx
            pload[2*i+1]=p_node2d[6*p_ele2dt[3*id+i]+5];//fy
        }
        return true;
}
double * FEM_Solve2dt()//三节点三角形单元求解函数,返回节点位移向量
{
        double *pgm=NULL,*p=NULL,*pus=NULL;
        ///
        double K[36],L[6];
        int i,j,m,id[6];
        //
        //先搜索没有约束的位移，计算自由度总数，形成新的索引编号
        N_nodes=0;
        for(i=0;i<N_node;i++)
        {
            if(p_node2d[6*i+2]!=0)
            {
                N_nodes++;
            }
            if(p_node2d[6*i+3]!=0)
            {
                N_nodes++;
            }
        }
        if(p_nodeuvid!=NULL)
```

```
        free(p_nodeuvid);
    p_nodeuvid=(int *)malloc(sizeof(int)*2*N_node);
    N_nodes=0;
    for(i=0;i<N_node;i++)
    {
        if(p_node2d[6*i+2]!=0)
        {
            p_nodeuvid[2*i]=N_nodes;
            N_nodes++;
        }
        else
        {
            p_nodeuvid[2*i]=-1;
        }
        if(p_node2d[6*i+3]!=0)
        {
            p_nodeuvid[2*i+1]=N_nodes;
            N_nodes++;
        }
        else
        {
            p_nodeuvid[2*i+1]=-1;
        }
    }
    pgm=(double *)malloc(sizeof(double)*N_nodes*N_nodes);
    p=(double *)malloc(sizeof(double)*N_nodes);
    if(pgm==NULL)
        return NULL;
    if(p==NULL)
        return NULL;
```

```
////
for(i=0;i<N_nodes*N_nodes;i++)
{
    pgm[i]=0;
}

///
for(m=0;m<N_elemt;m++)
{
    //
    FEM_ElementMatrix2dt(K,L,m);
    //
    for(i=0;i<3;i++)
    {
        id[2*i]=p_nodeuvid[2*p_ele2dt[3*m+i]];
        id[2*i+1]=p_nodeuvid[2*p_ele2dt[3*m+i]+1];
    }

    ////
    for(i=0;i<6;i++)
    {
        for(j=0;j<6;j++)
        {
            if(id[i]>=0&&id[j]>=0)
            {
                pgm[id[i]+N_nodes*id[j]]=pgm[id[i]+N_nodes*id[j]]+K[i+6*j];
            }
        }
        if(id[i]>=0)
        {
```

```
                p[id[i]]=L[i];
            }
        }

}
//
pus=(double *)malloc(sizeof(double)*N_nodes);
pu=(double *)malloc(sizeof(double)*2*N_node);
mx_inver(pgm,N_nodes);
mx_multipG(pgm,p,pus,N_nodes,N_nodes,N_nodes,1);
N_nodes=0;
for(i=0;i<N_node;i++)
{
    if(p_node2d[6*i+2]!=0)
    {
        pu[2*i]=pus[N_nodes];
        N_nodes++;
    }
    else
    {
        pu[2*i]=0;
    }
    if(p_node2d[6*i+3]!=0)
    {
        pu[2*i+1]=pus[N_nodes];
        N_nodes++;
    }
    else
    {
        pu[2*i+1]=0;
```

```
        }
    }
    free(pgm);      free(p);    free(pus);
    /////////计算 u_max,u_mid,u_min//////////////
    ux_max=ux_mid=ux_min=pu[0];
    uy_max=uy_mid=uy_min=pu[1];
    for(i=0;i<N_node;i++)
    {
        if(pu[2*i]>ux_max)
        {
            ux_max=pu[2*i];
        }
        if(pu[2*i]<ux_min)
        {
            ux_min=pu[2*i];
        }
        if(pu[2*i+1]>uy_max)
        {
            uy_max=pu[2*i+1];
        }
        if(pu[2*i+1]<uy_min)
        {
            uy_min=pu[2*i+1];
        }
    }
    ux_mid=0.5*(ux_max+ux_min);
    uy_mid=0.5*(uy_max+uy_min);
    ////////////////
    mx_printf(pu,"u",2*N_node,1,2,"u.txt");
    printf("Calculating Complete!\n");
```

```
    return pu;
}
int main()
{
    FEM_ReadData("MyFEMdata.fem");
    FEM_Solve2dt();
    return 1;
}
```

数据文件 MyFEMdata.fem 的内容如下：

```
        200e3           0.25
                68.             48.
                46.             37.             11.
                46.             36.             37.
                47.             36.             14.
                47.             35.             36.
                48.             28.             24.
                48.             27.             28.
                45.             44.             27.
                45.              3.             44.
                27.             48.             25.
                48.             15.             25.
                24.             15.             48.
                35.             47.             16.
                47.             12.             16.
                14.             12.             47.
                36.             46.             13.
                46.              2.             13.
                11.              2.             46.
                26.             45.             27.
                45.              1.              3.
```

26.	1.	45.
44.	43.	28.
43.	42.	29.
42.	41.	31.
41.	40.	32.
40.	39.	32.
39.	38.	33.
38.	37.	34.
36.	35.	37.
35.	34.	37.
34.	33.	38.
33.	32.	39.
32.	31.	41.
31.	30.	42.
30.	29.	42.
29.	28.	43.
28.	27.	44.
4.	43.	44.
5.	42.	43.
6.	41.	42.
7.	40.	41.
8.	39.	40.
9.	38.	39.
10.	37.	38.
17.	34.	35.
18.	33.	34.
19.	32.	33.
20.	31.	32.
21.	30.	31.
22.	29.	30.

23.	28.	29.
3.	4.	44.
4.	5.	43.
5.	6.	42.
6.	7.	41.
7.	8.	40.
8.	9.	39.
9.	10.	38.
10.	11.	37.
13.	14.	36.
16.	17.	35.
17.	18.	34.
18.	19.	33.
19.	20.	32.
20.	21.	31.
21.	22.	30.
22.	23.	29.
23.	24.	28.
25.	26.	27.

0.0	0. 0	0. 0	0. 0	0. 0	0. 0
100. 0	0. 0	0.12677E+31	0.12677E+31	100. 0	0. 0
10. 0	0. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
20. 0	0. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
30. 0	0. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
40. 0	0. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
50. 0	0. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
60. 0	0. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
70. 0	0. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
80. 0	0. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
90. 0	0. 0	0.12677E+31	0.12677E+31	0. 0	0. 0

100. 0	30. 0	0.12677E+31	0.12677E+31	100. 0	0. 0
100. 0	10. 0	0.12677E+31	0.12677E+31	100. 0	0. 0
100. 0	20. 0	0.12677E+31	0.12677E+31	100. 0	0. 0
0. 0	30. 0	0. 0E+00	0. 0E+00	0. 0	0. 0
90. 0	30. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
80. 0	30. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
70. 0	30. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
60. 0	30. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
50. 0	30. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
40. 0	30. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
30. 0	30. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
20. 0	30. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
10. 0	30. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
0. 0	20. 0	0. 0E+00	0. 0E+00	0. 0	0. 0
0. 0	10. 0	0. 0E+00	0. 0E+00	0. 0	0. 0
6.6666666667	15. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
15.2380952381	19.8086077197	0.12677E+31	0.12677E+31	0. 0	0. 0
26.7063492063	19.7448102929	0.12677E+31	0.12677E+31	0. 0	0. 0
35.0274632619	22.1941597928	0.12677E+31	0.12677E+31	0. 0	0. 0
43.3333333333	20. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
54.8030021606	17.8389205841	0.12677E+31	0.12677E+31	0. 0	0. 0
66.5103156648	19.4139480068	0.12677E+31	0.12677E+31	0. 0	0. 0
76.9378306878	19.6891783965	0.12677E+31	0.12677E+31	0. 0	0. 0
86.4960947342	21.3187874171	0.12677E+31	0.12677E+31	0. 0	0. 0
93.3333333333	15. 0	0.12677E+31	0.12677E+31	0. 0	0. 0
85.0387377173	9.9555969138	0.12677E+31	0.12677E+31	0. 0	0. 0
73.6456699647	9.7307230160	0.12677E+31	0.12677E+31	0. 0	0. 0
63.3871357185	9.3256147786	0.12677E+31	0.12677E+31	0. 0	0. 0
54.7667783457	7.2003291532	0.12677E+31	0.12677E+31	0. 0	0. 0
46.3334579349	9.3775397347	0.12677E+31	0.12677E+31	0. 0	0. 0

35.0106119198	11.6033028487	0.12677E+31	0.12677E+31	0. 0	0. 0
23.3974661054	9.9070984169	0.12677E+31	0.12677E+31	0. 0	0. 0
13.4297402679	8.2858696400	0.12677E+31	0.12677E+31	0. 0	0. 0
6.0192813869	6.6571739280	0.12677E+31	0.12677E+31	0. 0	0. 0
93.6744142101	6.9911193828	0.12677E+31	0.12677E+31	0. 0	0. 0
93.6666666667	23.2679491924	0.12677E+31	0.12677E+31	0. 0	0. 0
6.3809523810	22.9617215439	0.12677E+31	0.12677E+31	0. 0	0. 0

3.9 小结与习题

本章介绍了平面三节点三角形单元的基本概念和计算流程，介绍了位移函数、形函数、单元刚度矩阵的概念。采用虚位移原理推导了等效节点载荷的计算公式。基于应变能最小原理推导了有限元方程。最后给出了 C 语言实现的有限元计算过程。请完成以下作业：

（1）熟悉三节点三角形单元的位移函数表达式；

（2）推导三节点三角形单元位移函数待定系数的表达式；

（3）推导三节点三角形单元位移函数的形函数表达式；

（4）推导三节点三角形单元的应变与节点位移、应力与节点位移的表达式；

（5）采用虚位移原理推导三节点三角形单元的单元刚度矩阵表达式；

（6）采用虚位移原理推导集中载荷、体积力、表面力和热膨胀的等效节点载荷；

（7）采用能量泛函的变分建立有限元方程；

（8）编写三角形单元的有限元计算程序。

第四章 轴对称的有限元法

工程实际中遇到的如旋转机械中的盘、轴、机匣和承力环等，它们都有一个对称轴，整个物体是通过轴的一个平面上某个图形绕此轴旋转而形成的回转体，称之为轴对称体。

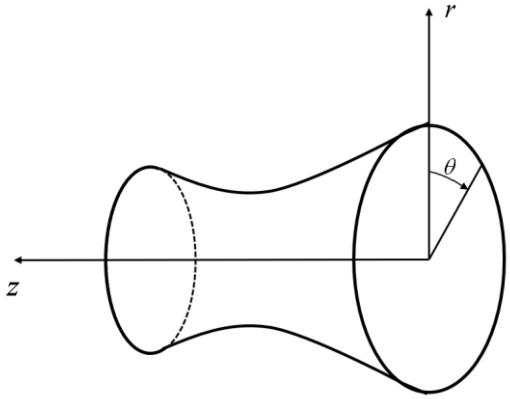


图 4.1 轴对称体模型

若轴对称体的载荷和约束都是轴对称的，则产生的位移、应变和应力必然是轴对称的，如高转速工作的轮盘等。这种结构的应力分析问题称为轴对称问题，本章详细介绍这一类问题。

在轴对称问题中，通常采用圆柱坐标(r,θ,z)，对称轴为 z 轴，沿半径方向为 r 轴，正向如图 4.1 所示。以 z 轴为正向的右螺旋转动方向表示 θ 的正向。

空间轴对称问题虽然属于三维问题，但由于几何形状是轴对称的，在轴对称载荷作用下，所产生的位移、应变和应力与 θ 无关，只是 r 和 z 的函数。因此，轴对称问题是准二维问题，可以按平面问题处理，但与平面问题又有不同之处。

4.1 轴对称问题的基本方程

（1）应力分量

轴对称问题的应力分量为

$$\{\sigma\}=\begin{Bmatrix}\sigma_{\mathrm{r}}\\ \sigma_{\mathrm{Z}}\\ \sigma_{\theta}\\ \tau_{\mathrm{rZ}}\end{Bmatrix} \tag{4.1}$$

（2）几何方程

对应于应力分量的应变分量为

$$\{\varepsilon\}=\begin{Bmatrix}\varepsilon_r\\ \varepsilon_z\\ \varepsilon_\theta\\ \gamma_{rz}\end{Bmatrix} \tag{4.2}$$

如果用 u、v 分别表示 r、z 方向的位移分量，那么应变分量与位移分量之间的关系，即几何方程为

$$\{\varepsilon\}=\begin{Bmatrix}\varepsilon_r\\ \varepsilon_z\\ \varepsilon_\theta\\ \gamma_{rz}\end{Bmatrix}=\begin{Bmatrix}\dfrac{\partial u}{\partial r}\\ \dfrac{\partial v}{\partial z}\\ \dfrac{u}{r}\\ \dfrac{\partial u}{\partial z}+\dfrac{\partial v}{\partial r}\end{Bmatrix} \tag{4.3}$$

上式中周向（θ方向）应变ε_θ是由径向位移u而引起的，这是与平面问题的重要区别之一。

(3)物理方程

对于各向同性材料，不考虑温度变化时的轴对称问题的物理方程为

$$\begin{cases}\sigma_r=\dfrac{E(1-\mu)}{(1+\mu)(1-2\mu)}\left[\varepsilon_r+\dfrac{\mu}{1-\mu}\varepsilon_Z+\dfrac{\mu}{1-\mu}\varepsilon_\theta\right]\\ \sigma_Z=\dfrac{E(1-\mu)}{(1+\mu)(1-2\mu)}\left[\dfrac{\mu}{1-\mu}\varepsilon_r+\varepsilon_Z+\dfrac{\mu}{1-\mu}\varepsilon_\theta\right]\\ \sigma_\theta=\dfrac{E(1-\mu)}{(1+\mu)(1-2\mu)}\left[\dfrac{\mu}{1-\mu}\varepsilon_r+\dfrac{\mu}{1-\mu}\varepsilon_Z+\varepsilon_\theta\right]\\ \tau_{rZ}=\dfrac{E}{2(1+\mu)}\gamma_{rZ}=\dfrac{E(1-\mu)}{(1+\mu)(1-2\mu)}\cdot\dfrac{(1-2\mu)}{2(1-\mu)}\gamma_{rZ}\end{cases} \tag{4.4}$$

写成矩阵形式为

$$\begin{Bmatrix}\sigma_r\\ \sigma_z\\ \sigma_\theta\\ \tau_{rZ}\end{Bmatrix}=\frac{E(1-\mu)}{(1+\mu)(1-2\mu)}\begin{bmatrix}1&\Delta_1&\Delta_1&0\\ \Delta_1&1&\Delta_1&0\\ \Delta_1&\Delta_1&1&0\\ o&o&o&\Delta_2\end{bmatrix}\begin{Bmatrix}\varepsilon_r\\ \varepsilon_z\\ \varepsilon_\theta\\ \gamma_{rZ}\end{Bmatrix} \tag{4.5}$$

又可以简写成

$$\{\sigma\}=[D]\{\varepsilon\} \tag{4.6}$$

其中弹性矩阵为

$$[D]=\frac{E(1-\mu)}{(1+\mu)(1-2\mu)}\begin{bmatrix}1 & \Delta_1 & \Delta_1 & 0\\ \Delta_1 & 1 & \Delta_1 & 0\\ \Delta_1 & \Delta_1 & 1 & 0\\ 0 & 0 & 0 & \Delta_2\end{bmatrix} \tag{4.7}$$

式中

$$\Delta_1=\frac{\mu}{(1-\mu)},\Delta_2=\frac{1-2\mu}{2(1-\mu)} \tag{4.8}$$

从上式可以看出，弹性矩阵 $[D]$ 只与材料的弹性模量 E 和泊松比 μ 有关。

(4)初应变

若轴对称体还受到温度变化引起的热负荷的作用，则要考虑由温度变化引起的初应变。对于各向同性体，令温度变化为T，线膨胀系数为α，则初应变为

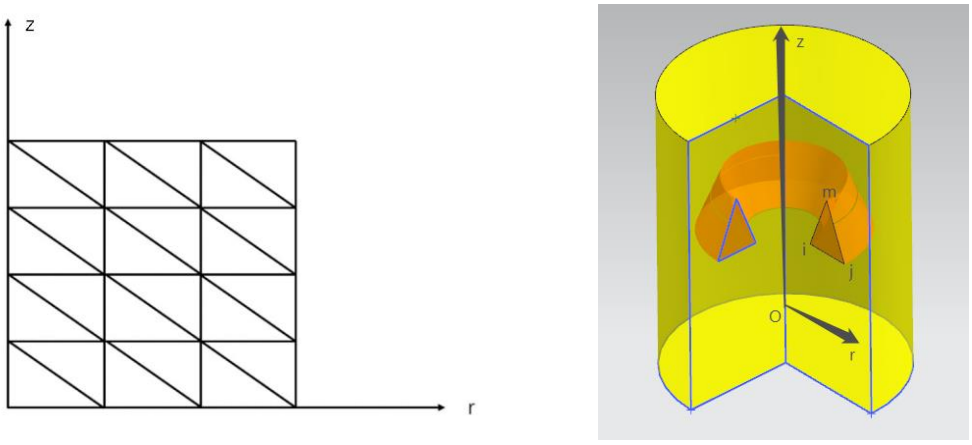
$$\{\varepsilon_0\}=\begin{bmatrix}\varepsilon_{r_0} & \varepsilon_{z_0} & \varepsilon_{\theta_0} & \gamma_{rz_0}\end{bmatrix}^T=\alpha T\begin{bmatrix}1 & 1 & 1 & 0\end{bmatrix}^T \tag{4.9}$$

于是应力与应变之间的关系为

$$\{\sigma\}=[D](\{\varepsilon\}-\varepsilon_0) \tag{4.10}$$

4.2 对称体的离散化

由于轴对称问题的位移和应力仅与坐标 r、z 有关，因此结构离散化只在子午面（ rz 面）内进行。同平面问题一样，把 rz 面求解域划分成若干个互不重叠的三角形单元（如图a），对单元划分的要求与平面问题三角形单元一样。



(a) (b)

图 4.2 轴对称单元

不过要注意的是由于轴对称问题的结构是旋转体，在 rz 面上划分的三角形单元，实际上代表一个 “三棱环体”单元（如图 b），称为三角形环单元。每一个三角形环单元都有三条棱边，这三条棱边是 3 个圆周，它们与 rz 面的交点是三角形的 3 个顶点 i,j,m，3 条棱边称为结点圆，i,j,m 称

为结点。单元节点是圆环形的铰链，三角形单元之间用这些铰链相互连接和传力。

4.3 位移函数

在轴对称问题的应力分析中，由于旋转体承受轴对称载荷，其变形是轴对称的，所以周向位移 $w = 0$，r,z 方向的位移分量 u,v 只与坐标位置有关。

因此，按照平面问题，取线性位移函数，即

$$\begin{Bmatrix} u = \alpha_1 + \alpha_2 r + \alpha_3 z \\ v = \alpha_4 + \alpha_5 r + \alpha_6 z \end{Bmatrix} \tag{4.11}$$

与平面问题类似，可以求出式中各待定系数 $\alpha_1 \sim \alpha_6$，即

$$\begin{aligned} \alpha_1 &= \frac{1}{2\Delta}(a_i u_i + a_j u_j + a_m u_m) \\ \alpha_2 &= \frac{1}{2\Delta}(b_i u_i + b_j u_j + b_m u_m) \\ \alpha_3 &= \frac{1}{2\Delta}(c_i u_i + c_j u_j + c_m u_m) \end{aligned} \tag{4.12}$$

$$\begin{aligned} \alpha_4 &= \frac{1}{2\Delta}(a_i v_i + a_j v_j + a_m v_m) \\ \alpha_5 &= \frac{1}{2\Delta}(b_i v_i + b_j v_j + b_m v_m) \\ \alpha_6 &= \frac{1}{2\Delta}(c_i v_i + c_j v_j + c_m v_m) \end{aligned} \tag{4.13}$$

式中Δ表示三角形环单元的横截面积，即

$$\Delta = \frac{1}{2}\begin{vmatrix} 1 & r_i & z_i \\ 1 & r_j & z_j \\ 1 & r_m & z_m \end{vmatrix} \tag{4.14}$$

这里应注意：节点编号 i,j,m 的排列顺序和平面问题一样，仍按逆时针转向排列。系数为

$$\begin{cases} a_i = r_j Z_m - r_m Z_j \\ b_i = Z_j - z_m \\ c_i = -\left(r_j - r_m\right) \end{cases} \tag{4.15}$$

利用三角形环单元的 3 个节点i,j,m 的位移值，可以得到与平面问题类似的位移函数表达式

$$\begin{cases} u = N_i u_i + N_j u_j + N_m u_m \\ v = N_i v_i + N_j v_j + N_m v_m \end{cases} \tag{4.16}$$

上式写成矩阵形式为

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} N_i & 0 & N_j & 0 & N_m & 0 \\ 0 & N_i & 0 & N_j & 0 & N_m \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \\ u_m \\ v_m \end{Bmatrix} \tag{4.17}$$

或

$$\{f\} = [N]\{\delta\}^e \tag{4.18}$$

式中 $\{f\}$为单元中任意一点$P(r,z)$的位移列阵

$$\{f\} = \begin{Bmatrix} u \\ v \end{Bmatrix} \tag{4.19}$$

其中$\{\delta\}^e$表示单元 3 个节点（圆）的位移值

$$\{\delta\}^e = [\delta_i \quad \delta_j \quad \delta_m]^T = [u_i \quad v_i \quad u_j \quad v_j \quad u_m \quad v_m]^T \tag{4.20}$$

其中$[N]$为形矩阵函数

$$[N] = \begin{bmatrix} IN_i & IN_j & IN_m \end{bmatrix} = \begin{bmatrix} N_i & 0 & N_j & 0 & N_m & 0 \\ 0 & N_i & 0 & N_j & 0 & N_m \end{bmatrix} \tag{4.21}$$

式中[I]是二阶单位矩阵。$[N]$是单元节点坐标和单元内任意一点$P\ (r,z)$坐标的函数，称为形状函数矩阵，$Ni(i,j,m)$为形函数，即

$$N_i = \frac{1}{2\Delta}\left(a_i + b_i r + c_i z\right) \tag{4.22}$$

4.4 单元的应变和应力

(1)单元的应变

利用式(4.12)和(4.13)，则式(4.3)可表示为

$$\varepsilon_r = \frac{1}{2\Delta}\left(b_i u_i + b_j u_j + b_m u_m\right) \varepsilon_z = \frac{1}{2\Delta}\left(c_i v_i + c_j v_j + c_m v_m\right)$$

$$\varepsilon_\theta = \frac{1}{2\Delta}\left[\left(\frac{a_i}{r} + b_i + \frac{c_i z}{r}\right)u_i + \left(\frac{a_j}{r} + b_j + \frac{c_j Z}{r}\right)u_j + \left(\frac{a_m}{r} + b_m + \frac{c_m Z}{r}\right)u_m\right] \tag{4.23}$$

$$\gamma_{rz} = \frac{1}{2\Delta}\left(c_i u_i + b_i v_i + c_j u_j + b_j v_j + c_m u_m + b_m v_m\right)$$

写成矩阵形式

$$\begin{Bmatrix}\varepsilon_r\\ \varepsilon_z\\ \varepsilon_\theta\\ r_{rz}\end{Bmatrix}=\frac{1}{2\Delta}\begin{bmatrix}b_i & 0 & b_j & 0 & b_m & 0\\ 0 & c_i & 0 & c_j & 0 & c_m\\ A_i & 0 & A_j & 0 & A_m & 0\\ c_i & b_i & c_i & b_i & c_m & b_m\end{bmatrix}\begin{Bmatrix}u_i\\ v_i\\ u_j\\ v_j\\ u_m\\ v_m\end{Bmatrix} \tag{4.24}$$

简写成

$$\{\varepsilon\}=[B]\{\delta\}^e=\begin{bmatrix}B_i & B_j & B_m\end{bmatrix}\{\delta\}^e \tag{4.25}$$

式中

$$\begin{bmatrix}B_i\end{bmatrix}=\frac{1}{2\Delta}\begin{bmatrix}b_i & 0\\ 0 & c_i\\ A_i & 0\\ c_i & b_i\end{bmatrix}\quad (i,j,m)$$
$$A_i=\frac{\alpha_i}{r}+b_i+\frac{c_i z}{r}\quad (i,j,m) \tag{4.26}$$

从上面应变表达式可以看到，单元中的应变分量 ε_r 、ε_z 和γ_{rz} 为常量，周向应变 ε_θ 不是常量，不仅与节点坐标有关，而且还与单元内各点的位置 (r,z) 有关，同时矩阵 B 中包含了$1/r$ 项，给计算带来了麻烦.

(2)初应变

在轴对称问题中，物体的温度$T = T(r,z)$，与坐标θ无关。单元内的初应变一般说来是不均匀的，但当单元的尺寸很小时，可以用一个平均温度值来表示。

若单元的 3 个结点的温度改变值分别为T_i 、T_j 和 T_m ，则单元的平均温度改变值为

$$\bar{T}=\frac{1}{3}\left(T_i+T_j+T_m\right) \tag{4.27}$$

于是单元的初应变为

$$\{\varepsilon_0\}=\alpha\bar{T}\begin{Bmatrix}1\\ 1\\ 1\\ 0\end{Bmatrix} \tag{4.28}$$

(3)单元的应力

将式(4.16)代入式 (4.5) ，得到单元内的应力

$$\{\sigma\}=[D][B]\{\delta\}^e=[S]\{\delta\}^e=\begin{bmatrix}S_i & S_j & S_m\end{bmatrix}\{\delta\}^e \tag{4.29}$$

式中[S] 称为应力矩阵，应力子矩阵为

$$\begin{bmatrix}S_i\end{bmatrix}=[D]\begin{bmatrix}B_i\end{bmatrix}=\frac{E(1-\mu)}{2\Delta(1+\mu)(1-2\mu)}\begin{bmatrix}b_i+\Delta_1 A_i & \Delta_1 c_i\\ \Delta_1\left(b_i+A_i\right) & c_i\\ \Delta_1 b_i+A_i & \Delta_1 c_i\\ \Delta_2 c_i & \Delta_2 b_i\end{bmatrix}(i,j,m) \tag{4.30}$$

式中

$$\Delta_1=\frac{\mu}{1-\mu},\quad \Delta_2=\frac{1-2\mu}{2(1-\mu)} \tag{4.31}$$

当考虑初应变时，单元应力为

$$\{\sigma\}=[D]\left(\{\varepsilon\}-\{\varepsilon_0\}\right) \tag{4.32}$$

从式(4.21)和(4.22)可见，单元中只有剪应力τ_{rz}为常量，其它应力分量都不是常量。

4.5 单元刚度矩阵

应用普遍公式 (2.76) ，可以写出

$$\begin{aligned}[K]^e&=\int_{V^e}[B]^T[D][B]dV=\int_{V^e}[B]^T[D][B]rd\theta drdz\\&=2\pi\int_{\Delta}[B]^T[D][B]rdrdz\end{aligned} \tag{4.33}$$

式中

$$[B]=\begin{bmatrix}B_i & B_j & B_m\end{bmatrix},[B]^T=\begin{Bmatrix}B_i^T\\B_j^T\\B_m^T\end{Bmatrix} \tag{4.34}$$

于是

$$[K]^e=2\pi\int_{\Delta}\begin{Bmatrix}B_i^T\\B_j^T\\B_m^T\end{Bmatrix}[D]\begin{bmatrix}B_i & B_j & B_m\end{bmatrix}rdrdz \tag{4.35}$$

上式又可以写成

$$[K]^e=\begin{bmatrix}K_{ii} & K_{ij} & K_{im}\\K_{ji} & K_{jj} & K_{jm}\\K_{mi} & K_{mj} & K_{mm}\end{bmatrix} \tag{4.36}$$

式中子矩阵$[K_{st}]$为 2×2 阶矩阵，即

$$K_{St}=2\pi\iint_{\Delta}[B_S]^T[D][B_t]rdrdz\quad(s,t=i,j,m) \tag{4.37}$$

与平面问题不同的是矩阵[B]内含有元素

$$A_i=\frac{a_i}{r}+b_i+\frac{c_iz}{r} \tag{4.38}$$

为了便于积分，简化计算，同时为了消除在对称轴上 r = 0 时所引起的麻烦，引入三角形形心坐标

$$\begin{cases}\overline{r}=\dfrac{r_i+r_j+r_m}{3}\\ \overline{Z}=\dfrac{z_i+z_j+z_m}{3}\end{cases} \tag{4.39}$$

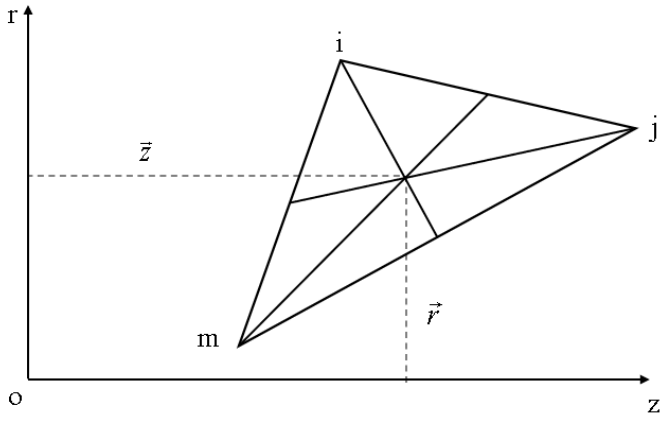


图 4.3 三角形形心坐标

将矩阵$[B_s]$（$[B_t]$）分成与坐标 r , z 无关的常值部分$[\overline{B_s}]$($[\overline{B_t}]$)和与坐标 r, z 有关的变值部分$[B_s']$($[B_t']$)，即

$$\begin{aligned}[B_S]&=[\overline{B}_S]+[B_S']\\ [B_t]&=[\overline{B}_t]+[B_t']\end{aligned} \tag{4.40}$$

以矩阵$[B_s]$为例，将其展开

$$[B_S]=\frac{1}{2\Delta}\begin{bmatrix}b_S&0\\0&c_S\\A_S&0\\c_S&b_S\end{bmatrix}=\frac{1}{2\Delta}\begin{bmatrix}b_S&0\\0&c_S\\\overline{A}_S&0\\c_S&b_S\end{bmatrix}+\frac{1}{2\Delta}\begin{bmatrix}0&0\\0&0\\A_S'&0\\0&0\end{bmatrix}=[\overline{B}_S]+[B_S'] \tag{4.41}$$

式中

$$\bar{A}_S=\frac{a_S}{\bar{r}}+b_S+\frac{c_S\bar{Z}}{\bar{r}}$$
$$A_S^{'}=-\frac{a_S}{\bar{r}}-\frac{c_S\bar{Z}}{\bar{r}}+\frac{a_S}{r}+\frac{c_SZ}{r} \tag{4.42}$$

常值部分$[\bar{B}_s]$是对三角形形心坐标$\bar{r}$，$\bar{z}$求出的，对于给定的三角形单元，是个常量矩阵，与坐标 r,z 无关。

单元刚度矩阵的子矩阵为

$$\left[K_{st}\right]=2\pi\int_{\Delta}\left[B_S\right]^T[D]\left[B_t\right]rdrdz=2\pi\int_{\Delta}\left(\left[\bar{B}_S\right]^T+\left[B_S^{'}\right]^T\right)[D]\left(\left[\bar{B}_t\right]+\left[B_t^{'}\right]\right)rdrdz=$$
$$2\pi\int_{\Delta}\left[\bar{B}_S\right]^T[D]\left[\bar{B}_t\right]rdrdz+2\pi\int_{\Delta}\left[B_S^{'}\right]^T[D]\left[B_t^{'}\right]rdrdz+2\pi\int_{\Delta}\left[\bar{B}_s\right]^T[D]\left[B_t^{'}\right]rdrdz+ \tag{4.43}$$
$$2\pi\int_{\Delta}\left[B_S^{'}\right]^T[D]\left[\bar{B}_t\right]rdrdz\quad(s,t=i,j,m)$$

可以证明，变值部分$[B_S']$对三角形面积的积分等于零。于是，单元刚度子矩阵简化为

$$\left[K_{st}\right]=\left[\bar{K}_{st}\right]+\left[K_{st}^{'}\right]=2\pi\int_{\Delta}\left[\bar{B}_s\right]^T[D]\left[\bar{B}_t\right]rdrdz+2\pi\int_{\Delta}\left[B_s^{'}\right]^T[D]\left[B_t^{'}\right]rdrdz \tag{4.44}$$

其中$[B_S']$积分等于零，单元刚度矩阵子矩阵的常值部分可以进一步表示为

$$\left[\bar{K}_{st}\right]=2\pi\int_{\Delta}\left[\bar{B}_s\right]^T[D]\left[\bar{B}_t\right]rdrdz=2\pi\left[\bar{B}_S\right]^T[D]\left[\bar{B}_t\right]\int_{\Delta}rdrdz=2\pi\left[\bar{B}_s\right]^T[D]\left[\bar{B}_t\right]\bar{r}\Delta=$$
$$\frac{\pi\bar{r}\Delta_3}{2\Delta}\begin{bmatrix}b_s\left(b_t+\Delta_1\bar{A}_t\right)+\bar{A}_s\left(\Delta_1b_t+\bar{A}_t\right)+\Delta_2c_sc_t & \Delta_1c_t\left(b_s+\bar{A}_s\right)+\Delta_2c_sb_t\\ \Delta_1c_s\left(b_t+\bar{A}_t\right)+\Delta_2b_sc_t & c_sc_t+\Delta_2b_sb_t\end{bmatrix}(s,t=i,j,m) \tag{4.45}$$

式中

$$\Delta_1=\frac{\mu}{1-\mu},\Delta_2=\frac{1-2\mu}{2(1-\mu)},\Delta_3=\frac{E(1-\mu)}{(1+\mu)(1-2\mu)} \tag{4.46}$$

单元刚度矩阵子矩阵的变值部分可以进一步表示为

$$\left[K_{st}^{'}\right]=2\pi\int_{\Delta}\left[B_S^{'}\right]^T[D]\left[B_t^{'}\right]rdrdz=\frac{\pi\bar{r}\Delta_3}{2\Delta}\Omega_{St}\begin{bmatrix}1&0\\0&0\end{bmatrix}(s,t=i,j,m) \tag{4.47}$$

式中

$$\Delta_3=\frac{E(1-\mu)}{(1+\mu)(1-2\mu)}\quad\Omega_{St}=\frac{1}{\bar{r}}\left[a_Sa_t\left(I_1-\frac{1}{\bar{r}}\right)+\left(a_Sc_t+a_tc_S\right)\left(I_2-\frac{\bar{z}}{\bar{r}}\right)+c_Sc_t\left(I_3-\frac{\bar{z}^2}{\bar{r}}\right)\right] \tag{4.48}$$
$$(s,t=i,j,m)$$

式中

$$\begin{cases} I_1 = \dfrac{1}{\Delta}\displaystyle\int_{\Delta r}\frac{1}{r}drdz \\ I_2 = \dfrac{1}{\Delta}\displaystyle\int_{\Delta}\frac{z}{r}drdz \\ I_3 = \dfrac{1}{\Delta}\displaystyle\int_{\Delta}\frac{z^2}{r}drdz \end{cases} \tag{4.49}$$

式(4.31)中的 3 个积分可以利用二维高斯积分公式进行数值计算，即

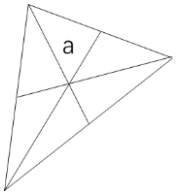
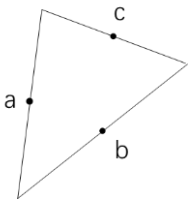
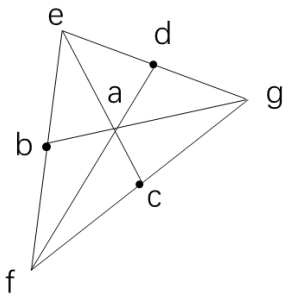
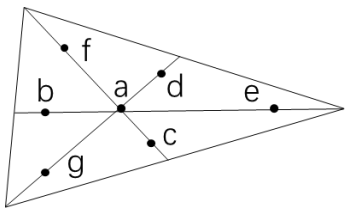
$$\int_{-1}^{1}\int_{-1}^{1} f(\xi,\eta)d\xi d\eta = \sum_{n}^{i=1}\sum_{n}^{j=1} W_i W_j f\left(\xi_i,\eta_j\right) \tag{4.50}$$

这一部分内容，将在第五章中具体介绍。在笛卡尔坐标系中计算上式积分比较麻烦。为了方便起见，可以在面积坐标系中计算，即

$$\int_{\Delta} f(r,z)drdz = \int_{\Delta} F\left(L_i,L_j,L_m\right)2\Delta dL_i dL_j = \Delta\sum_{n}^{k=1} 2H_k F\left(L_{ik},L_{jk},L_{mk}\right) \tag{4.51}$$

式中 n 为积分点数，$2H_k$为加权系数，L_{ik}，L_{jk}, L_{mk}为点 k 的 3 个面积坐标值，可参见表 4.1。利用圆柱坐标与面积坐标的关系，将上述三个积分转化为面积坐标表达式，然后进行数值积分。

表 4.1 三角形面积上数值积分表

图形	n	积分点的面积坐标	加权系数 2H
	1	$a\left(\frac{1}{3},\frac{1}{3},\frac{1}{3}\right)$	1
	3	$a\left(\frac{1}{2},\frac{1}{2},0\right)$ $b\left(0,\frac{1}{2},\frac{1}{2}\right)$ $c\left(\frac{1}{2},0,\frac{1}{2}\right)$	$\frac{1}{3}$
	7	$a\left(\frac{1}{3},\frac{1}{3},\frac{1}{3}\right)$ $\begin{cases} b\left(\frac{1}{2},\frac{1}{2},0\right) \\ c\left(0,\frac{1}{2},\frac{1}{2}\right) \\ d\left(\frac{1}{2},0,\frac{1}{2}\right) \end{cases}$ $e(1,0,0)$ $f(0,1,0)$ $g(0,0,1)$	$\frac{27}{60}$ $\frac{8}{60}$ $\frac{3}{60}$
	7	$a\left(\frac{1}{3},\frac{1}{3},\frac{1}{3}\right)$ 其中：	0.225 0.13239415 0.12593918

		$\begin{cases} b(\alpha_2,\beta_2,\beta_2) \\ c(\beta_2,\alpha_2,\beta_2) \\ d(\beta_2,\beta_2,\alpha_2) \end{cases}$ $\begin{cases} e(\alpha_3,\beta_3,\beta_3) \\ f(\beta_3,\alpha_3,\beta_3) \\ g(\beta_3,\beta_3,\alpha_3) \end{cases}$ $\alpha_2 = 0.05961587$ $\beta_2 = 0.47019206$ $\alpha_3 = 0.79742699$ $\beta_3 = 0.10128651$	

$$
\begin{aligned}
I_1 &= \sum_{n}^{k=1} 2H_k \frac{1}{L_{ik}r_i + L_{jk}r_j + L_{mk}r_m} \\
I_2 &= \sum_{n}^{k=1} 2H_k \frac{L_{ik}Z_i + L_{jk}Z_j + L_{mk}Z_m}{L_{ik}r_i + L_{jk}r_j + L_{mk}r_m} \\
I_3 &= \sum_{n}^{k=1} 2H_k \frac{\left(L_{ik}z_i + L_{jk}z_j + L_{mk}z_m\right)^2}{L_{ik}r_i + L_{jk}r_j + L_{mk}r_m}
\end{aligned}
\tag{4.52}
$$

4.6 结构刚度矩阵

求出单元刚度矩阵后，按照 2.5 节的方法，通过考查各节点（圆）静力平衡，得出结构的总体载荷列阵$\{R\}$与各节点位移列阵$\{\delta\}$之间关系，即

$$
[K]\{\delta\} = \{R\} \tag{4.53}
$$

式中$[K]$为结构刚度矩阵，式(4.32)称为结构刚度方程。结构刚度矩阵可由各单元刚度矩阵叠加得到。

4.7 等效节点载荷

轴对称问题在结构离散化过程中，若轴对称载荷不是作用在结点位置，则可以按照能量相等的原则，转化为等效结点载荷。这些等效结点载荷仍然是轴对称载荷，是分布在结点圆上的线载荷。

等效结点载荷的计算公式仍然是第二章的普遍公式(3.77)至(3.80)。下面介绍几种用于轴对称载荷的等效结点载荷计算公式。

(1)集中力的等效节点载荷

在三角形环单元边界上作用有集中力

$$P=\{P\}=\begin{Bmatrix}P_r\\R_z\end{Bmatrix} \tag{4.54}$$

即轴对称分布的线载荷（分布在一个圆周上的轴对称载荷）。按类似常应变三角形单元的方法，移置到 3 个节点上的等效结点载荷列阵为

$$\{R\}^e=\begin{Bmatrix}R_i\\R_j\\R_m\end{Bmatrix}=\begin{Bmatrix}R_{ir}\\R_{iz}\\R_{jr}\\R_{jz}\\R_{mr}\\R_{mz}\end{Bmatrix}=2\pi r[N]^T\{P\} \tag{4.55}$$

式中 r 为集中力作用点的半径。值得注意的是等效结点载荷也是结点圆周上的分布力。

(2)体积力的等效结点载荷

设单元作用有体积力

$$g=\{g\}=\begin{Bmatrix}g_r\\g_z\end{Bmatrix} \tag{4.56}$$

按类似常应变三角形单元的方法，其等效结点载荷列阵为

$$\{R\}^e=\begin{Bmatrix}R_i\\R_j\\R_m\end{Bmatrix}=\begin{Bmatrix}R_{ir}\\R_{iz}\\R_{jr}\\R_{jz}\\R_{mr}\\R_{mz}\end{Bmatrix}=2\pi\iint_{\Delta}[N]^T\{g\}rdrdz \tag{4.57}$$

如果轴对称载荷为自重，材料密度为ρ， 旋转对称轴 z 垂直地面，此时重力只有 z 方向的分量，则体积力

$$\{g\}=\begin{Bmatrix}g_r\\g_z\end{Bmatrix}=\begin{Bmatrix}0\\-\rho\end{Bmatrix} \tag{4.58}$$

等效结点载荷列阵为

$$\{R\}^e=\begin{Bmatrix}R_i\\R_j\\R_m\end{Bmatrix}=2\pi\iint_{\Delta}[N]^T\begin{Bmatrix}0\\-\rho\end{Bmatrix}rdrdz \tag{4.59}$$

对于结点i，则有

$$\{R_i\}^e=\begin{Bmatrix}R_{ir}\\R_{iz}\end{Bmatrix}=2\pi\int_{\Delta}N_i\begin{Bmatrix}0\\-\rho\end{Bmatrix}rdrdz(i,j,m) \tag{4.60}$$

利用面积坐标，可以计算得到结点i的等效结点载荷为

$$\begin{gathered}\{R_i\}^e=\begin{Bmatrix}R_{ir}\\R_{iz}\end{Bmatrix}=\begin{Bmatrix}0\\-\dfrac{\pi}{6}\Delta\left(3\overline{r}+r_i\right)\end{Bmatrix}(i,j,m)\\\{g\}=\begin{Bmatrix}g_r\\g_z\end{Bmatrix}=\begin{Bmatrix}\rho\omega^2 r\\0\end{Bmatrix}\end{gathered} \tag{4.61}$$

等效结点载荷为

$$\{R\}^e=\begin{Bmatrix}R_i\\R_j\\R_m\end{Bmatrix}=2\pi\int_{\Delta}[N]^T\left\{\rho\omega^2 r\right\}rdrdz \tag{4.62}$$

对于结点 i，并利用面积积分公式，得

$$\{R_i\}^e=\begin{Bmatrix}R_{ir}\\R_{iz}\end{Bmatrix}=\begin{Bmatrix}\dfrac{\pi\rho\omega^2\Delta}{15}\left(9\overline{r}^2+2r_i^2-r_jr_m\right)\end{Bmatrix}(i,j,m) \tag{4.63}$$

(3)表面力的等效结点载荷

设三角形环单元的 $i-m$环形表面上作用有均布载荷 q（图 4.4），则表面力

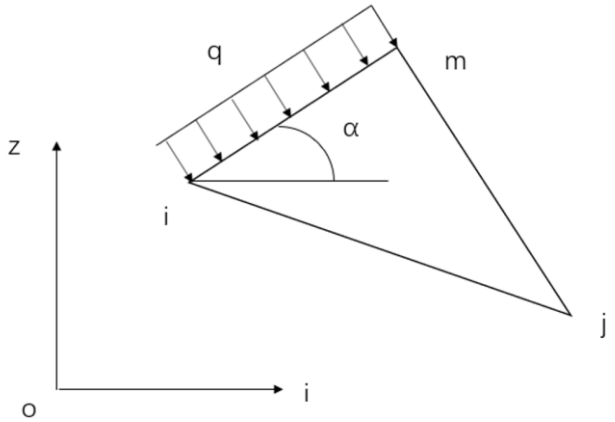


图 4.4 单元表面力

$$\{q\}=\begin{Bmatrix}q_r\\q_z\end{Bmatrix}=\begin{Bmatrix}q\sin\alpha\\-q\cos\alpha\end{Bmatrix}=\begin{Bmatrix}q\dfrac{z_m-z_i}{l_{im}}\\q\dfrac{r_i-r_m}{l_{im}}\end{Bmatrix} \tag{4.64}$$

式中，l_{im}为im边的边长。根据式(3.37),等效节点载荷公式为

$$\{R\}^e=\begin{Bmatrix}R_i\\R_j\\R_m\end{Bmatrix}=2\pi\int_{l_{im}}[N]^T\{q\}rds \tag{4.65}$$

对于结点i有

$$\{R_i\}^e=\begin{Bmatrix}R_{ir}\\R_{iz}\end{Bmatrix}=2\pi\int_{l_{im}}N_i\begin{Bmatrix}q\dfrac{z_m-z_i}{l_{im}}\\q\dfrac{r_i-r_m}{l_{im}}\end{Bmatrix}rds \tag{4.66}$$

式中积分

$$\int_{l_{im}}N_i rds=\int_{l_{im}}L_i\left(L_i r_i+L_j r_j+L_m r_m\right)ds \tag{4.67}$$

注意到沿边界 $i-m$ 积分时$L_j=0$，则上式为

$$\int_{l_{im}}N_i rds=\int_{l_{im}}L_i\left(L_i r_i+L_m r_m\right)ds \tag{4.68}$$

代入式 (4.38) 得

$$\{R_i\}^e=\begin{Bmatrix}R_{ir}\\R_{iz}\end{Bmatrix}=\frac{\pi q}{3}\left(2r_i+r_m\right)\begin{Bmatrix}z_m-z_i\\r_i-r_m\end{Bmatrix} \tag{4.69}$$

同理可得结点m的等效结点载荷为

$$\{R_m\}^e=\begin{Bmatrix}R_{mr}\\R_{mz}\end{Bmatrix}=\frac{\pi q}{3}\left(r_i+2r_m\right)\begin{Bmatrix}z_m-z_i\\r_i-r_m\end{Bmatrix} \tag{4.70}$$

因为沿 $i-m$ 边，$L_j=0$，所以

$$\{R_j\}^e=\begin{Bmatrix}R_{jr}\\R_{jz}\end{Bmatrix}=\begin{Bmatrix}0\\0\end{Bmatrix} \tag{4.71}$$

(4)温度改变的等效结点载荷

由初始热应变引起的等效结点载荷为

$$\{R_t\}^e=\begin{Bmatrix}R_{ti}\\R_{tj}\\R_{tm}\end{Bmatrix}=\int_{V^e}[B]^T[D]\{\varepsilon_0\}dV=2\pi\int_{\Delta}[B]^T[D]\{\varepsilon_0\}rdrdz \tag{4.72}$$

对于结点 i 有

$$\{R_{ti}\}^e=\begin{Bmatrix}R_{tir}\\R_{tiz}\end{Bmatrix}=2\pi\iint_{\Delta}[B_i]^T[D]\{\varepsilon_0\}rdrdz$$
$$=\frac{E\pi\alpha T}{1-2\mu}\begin{Bmatrix}b_i\bar{r}+\frac{2}{3}\Delta\\c_i\bar{r}\end{Bmatrix}\quad(i,j,m) \tag{4.73}$$

4.8 应力计算

引入几何边界条件，求解以结点位移列阵{δ}为未知数的矩阵方程

$$[K]\{\delta\}=\{R\} \tag{4.74}$$

可以得到各结点的位移值。由下式求出单元内任意一点的应力

$$\{\sigma\}=[D][B]\{\delta\}^e=[S]\{\delta\}^e=\begin{bmatrix}S_i & S_j & S_m\end{bmatrix}\begin{Bmatrix}\delta_i\\\delta_j\\\delta_m\end{Bmatrix}$$
$$=[S_i][\delta_i]+[S_j][\delta_j]+[S_m][\delta_m]=\{\sigma_i\}+\{\sigma_j\}+\{\sigma_m\} \tag{4.75}$$

将[S_i]和{δ_i}代入上式，得到

$$\{\sigma_i\}=[S_i]\{\delta_i\}=\frac{E(1-\mu)}{2\Delta(1+\mu)(1-2\mu)}\begin{bmatrix}b_i+\Delta_1A_i & \Delta_1c_i\\\Delta_1(b_i+A_i) & c_i\\\Delta_1b_i+A_i & \Delta_1c_i\\\Delta_2c_i & \Delta_2b_i\end{bmatrix}\begin{Bmatrix}u_i\\v_i\end{Bmatrix}$$
$$=\frac{E(1-\mu)}{2\Delta(1+\mu)(1-2\mu)}\begin{Bmatrix}b_iu_i+\Delta_1A_iu_i+\Delta_1c_iv_i\\\Delta_1b_6u_i+\Delta_1A_iu_i+c_iv_i\\\Delta_1b_iu_i+A_iu_i+\Delta_1c_iv_i\\\Delta_2c_iu_i+\Delta_2b_iv_i\end{Bmatrix}(i,j,m) \tag{4.76}$$

将{σ_i}、{σ_j}和{σ_m}叠加，即可得到所要求的应力列阵

$$\{\sigma\}=\{\sigma_i\}+\{\sigma_j\}+\{\sigma_m\} \tag{4.77}$$

为了便于计算，常用单元形心位置（$\bar{r}$，$\bar{z}$）的应力作为单元的平均应力，这时将形心坐标$\bar{r}$，$\bar{z}$替换式(4.42)中的r，z即可。

若计算热应力，在计算等效结点载荷时，附加计算一个由温度变化引起的等效结点载荷，由考虑温度变化的结构刚度方程求出结点位移后，再计算单元内任意一点的应变，其中应扣除初始热应变，应力列阵由下式计算，即

$$\{\sigma\}=[D](\{\varepsilon\}-\{\varepsilon_0\}) \tag{4.78}$$

第五章 平面等参单元

平面三节点三角形单元是最为简单的二维平面单元，它的应变在单元内部是常数。实际构件中的应变通常是连续分布的，因此采用三角形单元计算时，如果单元的密度较小时，计算结果与真实值的差别较大，需要通过提高网格密度来提高计算精度。

人们通常采用提高单元位移函数阶次的方式来提高计算精度。本章我们将学习最常用的一类单元——等参元，并介绍构造此类单元的一般方法。

5.1 位移函数

如图 5. 1 所示结构，我们用平面四节点单元来离散结构。取出一个单元来作为研究对象，建立曲线局部坐标系ζ-o-η。单元的四个节点用 1、2、3、4 来表示。ζ 和η的变化范围均为-1 到 1。单元内任意一点 P 的 x 方向和 y 方向位移分别为u和v。它们是局部坐标ζ 和η的函数。

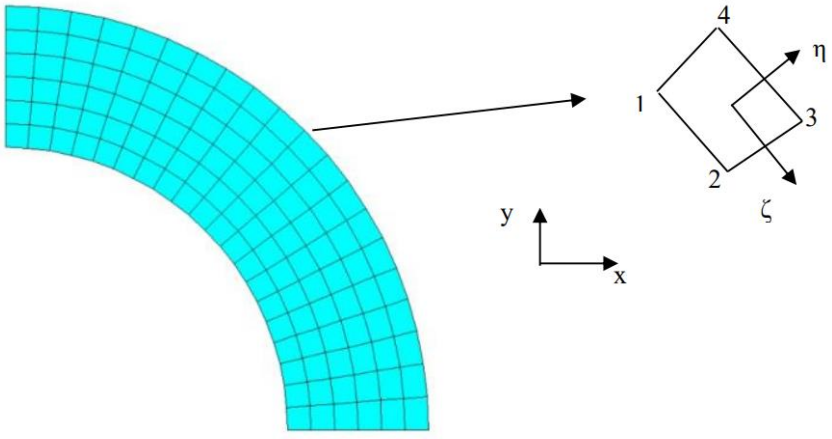


图 5. 1 采用平面四节点单元离散的结构

有限元法中通常将位移表示为形函数与节点位移的线性组合。图 5. 1 所示单元含有四个节点，因此位移函数可表示为如下形式：

$$\begin{aligned} u(\varsigma,\eta) &= N_1(\varsigma,\eta)u_1 + N_2(\varsigma,\eta)u_2 + N_3(\varsigma,\eta)u_3 + N_4(\varsigma,\eta)u_4 \\ v(\varsigma,\eta) &= N_1(\varsigma,\eta)v_1 + N_2(\varsigma,\eta)v_2 + N_3(\varsigma,\eta)v_3 + N_4(\varsigma,\eta)v_4 \end{aligned} \tag{5.1}$$

根据 3.1 节所述，位移函数需要满足以下两个条件：

（1）完备性-要求位移函数应该包含常应变项和刚体位移项

如果在势能泛函中出现的位移函数的最高阶导数是 m 阶，则选取的位移函数至少是 m 阶完全多项式。

（2）协调性-相邻单元公共边界保持位移连续

如果在势能泛函中所出现的位移函数的最高阶导数是 m 阶，则位移函数在单元交界面上必须具有直至（$m-1$）阶连续导数，即 C_{m-1} 连续性。

5.2 形函数

为了保证位移函数满足上述两个条件，形函数一般需要具备如下两个性质：

（1）形函数 N_i 在结点 i 处的值为 1，而在其他三个结点处的值为零。

（2）在单元任一点处，四个形函数之和等于 1。

其中性质（1）是为了保证位移函数的协调性，性质（2）为了保证位移函数的收敛性。我们先根据性质（1）构造形函数。

以 N_1 为例，参照图 5. 1，N_1 在除 1 结点以外的结点处等于零，那么 N_1 应该包含过结点 3 和 4 的直线方程以及过结点 2 和 3 的直线方程。于是初步构造 N_1 等于：

$$N_1 = (\zeta - 1)(\eta - 1) \tag{5.2}$$

该函数能够保证 N_1 在 2、3、4 点为零。我们将 1 点的坐标代入方程(5.2)后发现 $N_1 = 4$，为了保证 N_1 等于 1，我们将方程(5.2)除以 4。于是得到最终的 1 结点形函数：

$$N_1 = \frac{1}{4}(\zeta - 1)(\eta - 1) \tag{5.3}$$

采用相同的方法构造 2、3、4 点的形函数，最终得到统一表达式：

$$N_i(\zeta,\eta) = \frac{1}{4}(1+\zeta_i\zeta)(1+\eta_i\eta) \qquad (i = 1,2,3,4) \tag{5.4}$$

其中 (ζ_i,η_i) 是 i 节点的局部坐标。读者可以将四个形函数加起来，发现满足性质（2），因此该形函数是可行的。

四结点等参元的位移函数表达式如下：

$$u(\zeta,\eta) = \sum_{i=1}^{4} N_i(\zeta,\eta)u_i \quad v(\zeta,\eta) = \sum_{i=1}^{4} N_i(\zeta,\eta)v_i \tag{5.5}$$

为了提高位移函数的阶次，还可以在图 5. 1 所示单元上添加中间节点。这样单元的结点数变为 8。如图 5. 2 所示。

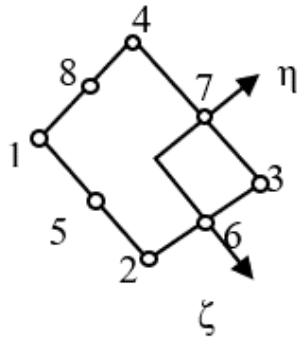


图 5.2 平面八节点等参元

位移函数具有如下形式：

$$u(\zeta,\eta)=\sum_{i=1}^{8}N_i(\zeta,\eta)u_i \quad v(\zeta,\eta)=\sum_{i=1}^{8}N_i(\zeta,\eta)v_i \tag{5.6}$$

仍然采用四结点单元的“划线法”来构造 8 结点单元的形函数，以 1 结点为例。因为要保证在除了 1 节点外所有的结点 $N_1=0$，所以 N_1 的表达式中应该包含 2-3 结点、3-4 结点以及 5-8 结点构成的直线方程。初步构造 N_1 的表达式如下：

$$N_1=(\zeta-1)(\eta-1)(\zeta+\eta+1) \tag{5.7}$$

为了保证 N_1 在结点 1 处的数值等于 1，将 1 结点的坐标代入方程(5.7)后得 $N_1=-4$。因此将 N_1 除以−4 后得到最终的 N_1：

$$N_1=\frac{1}{4}(1-\zeta)(1-\eta)(-\zeta-\eta-1) \tag{5.8}$$

结点 2、3、4 也用类似的方法可以得到。于是可得结点 1-4 的形函数表达式：

$$N_i(\zeta,\eta)=\frac{1}{4}(1+\zeta_i\zeta)(1+\eta_i\eta)(\zeta_i\zeta+\eta_i\eta-1) \quad （i=1，2，3，4） \tag{5.9}$$

结点 5 的形函数要求在除 5 结点以外的结点处等于 0，因此 N_5 应该包含过结点 1-4、2-3 和 3-4 的直线方程，初步得 N_5 的表达式：

$$N_5=(\zeta-1)(\zeta+1)(\eta-1) \tag{5.10}$$

在结点 5 处 $N_5=2$，为了保证 N_5 在结点 5 处等于 1，于是将 N_5 除以 2，最终得 N_5 的表达式：

$$N_5=\frac{1}{2}(1-\zeta)(\zeta+1)(1-\eta) \tag{5.11}$$

用类似的方法还可以得到 N_6、N_7 和 N_8 的表达式：

$$N_i=\frac{1}{2}(1-\zeta^2)(1+\eta_i\eta) \quad (i=5,7) \quad N_i=\frac{1}{2}(1-\eta^2)(1+\zeta_i\zeta) \quad (i=6,8) \tag{5.12}$$

5.3 单元刚度矩阵

有了位移函数(5.5)和(5.6)我们就可以通过几何方程得到应变的表达式：

$$\varepsilon_x=\frac{\partial u}{\partial x}=\sum_{i=1}^{m}\frac{\partial N_i}{\partial x}u_i \quad \varepsilon_y=\frac{\partial v}{\partial y}=\sum_{i=1}^{m}\frac{\partial N_i}{\partial y}v_i \quad \gamma_{xy}=\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}=\sum_{i=1}^{m}\frac{\partial N_i}{\partial y}u_i+\sum_{i=1}^{m}\frac{\partial N_i}{\partial x}v_i \tag{5.13}$$

其中 m 是单元内结点总数，对于四节点单元 $m=4$ ，对于八结点单元 $m=8$ 。

从方程(5.13)可见，要计算应变需要计算形函数对坐标的偏导数。然而所有形函数均表达为局部坐标的形式，还需要建立局部坐标和全局坐标 x 、y 之间的关系才能进行计算。受到位移函数的启发，我们只要将方程(5.5)或者(5.6)的节点位移换成节点坐标，即可建立局部坐标和全局笛卡尔坐标之间的关系，即：

$$x(\zeta,\eta)=\sum_{i=1}^{m}N_i(\zeta,\eta)x_i \quad y(\zeta,\eta)=\sum_{i=1}^{m}N_i(\zeta,\eta)y_i \tag{5.14}$$

其中 x_i 和 y_i 是结点 i 的笛卡尔坐标。

于是：

$$\frac{\partial N_i}{\partial \zeta}=\frac{\partial N_i}{\partial x}\frac{\partial x}{\partial \zeta}+\frac{\partial N_i}{\partial y}\frac{\partial y}{\partial \zeta}\ , \quad \frac{\partial N_i}{\partial \eta}=\frac{\partial N_i}{\partial x}\frac{\partial x}{\partial \eta}+\frac{\partial N_i}{\partial y}\frac{\partial y}{\partial \eta} \tag{5.15}$$

写成矩阵形式：

$$\begin{bmatrix}\dfrac{\partial N_i}{\partial \zeta}\\ \dfrac{\partial N_i}{\partial \eta}\end{bmatrix}=\begin{bmatrix}\dfrac{\partial x}{\partial \zeta} & \dfrac{\partial y}{\partial \zeta}\\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta}\end{bmatrix}\cdot\begin{bmatrix}\dfrac{\partial N_i}{\partial x}\\ \dfrac{\partial N_i}{\partial y}\end{bmatrix} \tag{5.16}$$

求逆后：

$$\begin{bmatrix}\dfrac{\partial N_i}{\partial x}\\ \dfrac{\partial N_i}{\partial y}\end{bmatrix}=\begin{bmatrix}\dfrac{\partial x}{\partial \zeta} & \dfrac{\partial y}{\partial \zeta}\\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta}\end{bmatrix}^{-1}\cdot\begin{bmatrix}\dfrac{\partial N_i}{\partial \zeta}\\ \dfrac{\partial N_i}{\partial \eta}\end{bmatrix} \tag{5.17}$$

形如方程(5.14)的变换称为等参变换，这样的单元也称为等参元。如果单元的坐标插值的阶次大于位移的阶次，那么该单元称为超参元。相反，如果单元坐标插值的阶次低于位移的阶次，称之为次参元。定义：

$$[J]=\begin{bmatrix}\dfrac{\partial x}{\partial \zeta} & \dfrac{\partial y}{\partial \zeta}\\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta}\end{bmatrix} \tag{5.18}$$

式中 $[J]$ 为坐标变换矩阵或者雅可比矩阵。其坐标变换矩阵的逆矩阵为：

$$[J]^{-1}=\frac{1}{|J|}\begin{bmatrix}\dfrac{\partial y}{\partial \eta} & -\dfrac{\partial y}{\partial \zeta}\\ -\dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \zeta}\end{bmatrix} \tag{5.19}$$

其中

$$|J|=\frac{\partial x}{\partial \zeta}\frac{\partial y}{\partial \eta}-\frac{\partial y}{\partial \zeta}\frac{\partial x}{\partial \eta} \tag{5.20}$$

称作变换行列式或雅可比行列式。将方程(5.19)代入(5.17)后得：

$$\begin{aligned}\frac{\partial N_i}{\partial x}&=\frac{1}{|J|}\left(\frac{\partial y}{\partial \eta}\frac{\partial N_i}{\partial \zeta}-\frac{\partial y}{\partial \zeta}\frac{\partial N_i}{\partial \eta}\right)\\ \frac{\partial N_i}{\partial y}&=\frac{1}{|J|}\left(-\frac{\partial x}{\partial \eta}\frac{\partial N_i}{\partial \zeta}+\frac{\partial x}{\partial \zeta}\frac{\partial N_i}{\partial \eta}\right)\end{aligned} \tag{5.21}$$

利用上式，可以把任一形函数 $N_i(\zeta,\eta)$ 对x、y求导的问题转化为对 ζ 、η 求导的问题。为了计算单元刚度矩阵及等效结点载荷，还要把总体坐标下的微元面积dA 转换到局部坐标上去。

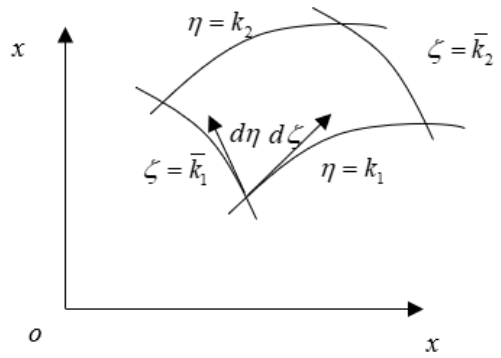


图 5.3 曲线坐标上的微元面积

如图 5.3 所示，设 ζ 和 η 是平面中的曲线坐标。$d\zeta$ 是与曲线 $\eta=k_1$ 相切的矢量，$d\eta$ 是与曲线 $\zeta=\bar{k}_1$ 相切的矢量。其中 k_1 、 $\bar{k}_1$ 均为常量。于是：

$$\begin{cases}\mathbf{d\zeta}=\mathrm{i}\dfrac{\partial x}{\partial \zeta}d\zeta+\mathrm{j}\dfrac{\partial y}{\partial \zeta}d\zeta\\ \mathbf{d\eta}=\mathrm{i}\dfrac{\partial x}{\partial \eta}d\eta+\mathrm{j}\dfrac{\partial y}{\partial \eta}d\eta\end{cases} \tag{5.22}$$

令

$$\mathbf{C}=\mathbf{d\zeta}\times\mathbf{d\eta} \tag{5.23}$$

$$\mathbf{C}=\mathbf{d\zeta}\times\mathbf{d\eta}=\begin{vmatrix}\mathrm{i} & \mathrm{j} & \mathrm{k}\\ \dfrac{\partial x}{\partial \zeta}d\zeta & \dfrac{\partial y}{\partial \zeta}d\zeta & 0\\ \dfrac{\partial x}{\partial \eta}d\eta & \dfrac{\partial y}{\partial \eta}d\eta & 0\end{vmatrix}=\mathrm{k}\begin{vmatrix}\dfrac{\partial x}{\partial \zeta} & \dfrac{\partial y}{\partial \zeta}\\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta}\end{vmatrix}d\zeta\,d\eta \tag{5.24}$$

由矢量运算可知，以 **dζ** 和 **dη** 为边的平行四边形的微面积等于向量 **C** 的模。所以

$$dA=|\mathbf{C}|=\begin{vmatrix}\dfrac{\partial x}{\partial \zeta} & \dfrac{\partial y}{\partial \zeta}\\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta}\end{vmatrix}d\zeta d\eta=|J|d\zeta d\eta \tag{5.25}$$

由上面讨论可知，为了求得雅可比变换矩阵的逆矩阵$[J]^{-1}$及单元刚度矩阵积分式中的微元面积 dA，要求变换矩阵的行列式在整个单元上不等于零，即：

$$|J|\neq 0 \tag{5.26}$$

这就是确保等参变换（总体坐标与局部坐标一一对应）的必要条件。

为了能确保进行等参变换，在总体坐标下所划分的斜四边形单元必须是凸四边形，而不能有一内角大于或等于 180 度的四边形。或者说任意两条边沿伸时不能在单元上出现交点。

应变分量的计算公式为：

$$\{\varepsilon\}=\begin{Bmatrix}\dfrac{\partial u}{\partial x}\\ \dfrac{\partial v}{\partial y}\\ \dfrac{\partial u}{\partial y}+\dfrac{\partial v}{\partial x}\end{Bmatrix}=[B]\{\delta\}^e=\begin{bmatrix}B_1 & B_2 & \cdots & B_m\end{bmatrix}\{\delta\}^e \tag{5.27}$$

其中 m 为节点数。式中$\{\delta\}^e=\begin{bmatrix}\delta_1 & \delta_2 & \cdots & \delta_m\end{bmatrix}^T$，$\{\delta_i\}=\begin{bmatrix}u_i & v_i\end{bmatrix}^T\ \ (i=1,2,\cdots,m)$

$$[B_i]=\begin{bmatrix}\dfrac{\partial N_i}{\partial x} & 0\\ 0 & \dfrac{\partial N_i}{\partial y}\\ \dfrac{\partial N_i}{\partial y} & \dfrac{\partial N_i}{\partial x}\end{bmatrix}\quad (i=1,2,\cdots,m) \tag{5.28}$$

根据 3.3 节，单元刚度矩阵可由虚功原理得到：

$$[K]^e=\iint_{\Delta}[B]^T[D][B]t\mathrm{d}x\mathrm{d}y=\iint_{\Delta}[B]^T[D][B]t\mathrm{d}A \tag{5.29}$$

将(5.24)代入(5.28)得：

$$[K]^e=\int_{-1}^{1}\int_{-1}^{1}[B]^T[D][B]t|J|\mathrm{d}\zeta\,\mathrm{d}\eta \tag{5.30}$$

式中 t 为单元厚度，$[K]^e$ 也可用分块形式写出

$$[K]^e=\begin{bmatrix} K_{11} & K_{12} & \cdots & K_{18} \\ K_{21} & K_{22} & \cdots & K_{28} \\ \cdots & \cdots & \cdots & \cdots \\ K_{81} & K_{82} & \cdots & K_{88} \end{bmatrix} \tag{5.31}$$

$$\left[K_{ij}\right]=\int_{-1}^{1}\int_{-1}^{1}[B_i]^T[D]\left[B_j\right]t|J|d\zeta d\eta \quad (i,j=1,2,\cdots,8) \tag{5.32}$$

此式积分比较复杂，很难用解析法计算，一般采用数值积分来计算。本文将在本章后续章节介绍如何采用高斯积分法计算(5.31)。

5.4 等效节点载荷

这里以平面四节点等参元为例介绍等效节点载荷的计算方法。等参单元的等效节点载荷 R_{1x} R_{1y} R_{2x} R_{2y} R_{3x} R_{3y} R_{4x} R_{4y} 计算流程与三角形单元相同，即通过虚功相等原理将各类型载荷等效为结点载荷。

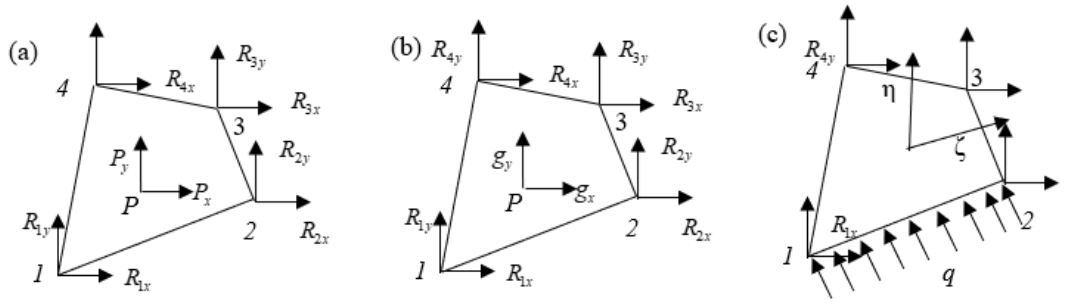


图 5.4 等参单元的等效节点载荷（a）集中力；（b）体积力；（c）表面力

如图 5. 4（a）所示，在单元内任意一点作用一个集中载荷 P，其 x 和 y 方向分量分别是 P_x 和 P_y。与 P 等效的结点载荷为 $\{R\}^e=\left[R_{1x},R_{1y},R_{2x},R_{2y},R_{3x},R_{3y},R_{4x},R_{4y}\right]^T$。

假设在节点处发生了虚位移 $\left\{\delta^*\right\}^e=\left[u_1^*,v_1^*,u_2^*,v_2^*,u_3^*,v_3^*,u_4^*,v_4^*\right]^T$。由结点虚位移产生的，在 P 点的虚位移可以用位移函数(5.5)计算：

$$\left\{f^*\right\}=\begin{Bmatrix}\delta u\\ \delta v\end{Bmatrix}=\begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 \end{bmatrix}\begin{Bmatrix} u_1^* \\ v_1^* \\ u_2^* \\ v_2^* \\ u_3^* \\ v_3^* \\ u_4^* \\ v_4^* \end{Bmatrix} \tag{5.33}$$

根据能量相等原则，原集中力与单元的等效结点载荷在相应的虚位移上所作的虚功相等，有

$$\left\{\delta^*\right\}^{eT}\{R\}^e=\left\{f^*\right\}^T\{P\} \tag{5.34}$$

将方程(5.32)代入(5.33)后得

$$\left[u_1^*,v_1^*,u_2^*,v_2^*,u_3^*,v_3^*,u_4^*,v_4^*\right]\cdot\begin{bmatrix}R_{1x}\\R_{1y}\\R_{2x}\\R_{2y}\\R_{3x}\\R_{3y}\\R_{4x}\\R_{4y}\end{bmatrix}=\left[u_1^*,v_1^*,u_2^*,v_2^*,u_3^*,v_3^*,u_4^*,v_4^*\right]\cdot\begin{bmatrix}N_1&0\\0&N_1\\N_2&0\\0&N_2\\N_3&0\\0&N_3\\N_4&0\\0&N_4\end{bmatrix}\cdot\begin{bmatrix}P_x\\P_y\end{bmatrix}\tag{5.35}$$

因为上述方程在任意虚位移下都必须成立，因此方程左右两边虚位移的系数必须相等，所以有

$$\begin{bmatrix}R_{1x}\\R_{1y}\\R_{2x}\\R_{2y}\\R_{3x}\\R_{3y}\\R_{4x}\\R_{4y}\end{bmatrix}=\begin{bmatrix}N_1&0\\0&N_1\\N_2&0\\0&N_2\\N_3&0\\0&N_3\\N_4&0\\0&N_4\end{bmatrix}\cdot\begin{bmatrix}P_x\\P_y\end{bmatrix}\tag{5.36}$$

方程(5.35)建立了单元内任意集中载荷与节点力之间的等效关系式。

体积力、表面力的等效结点载荷也可以通过虚功相等计算出来，步骤与集中载荷相同。留给读者自己推导，本书只给出最终结论。假设$\{g\}=\left[g_x,g_y\right]^T$是单元内单位体积受到的体积力。其等效节点载荷可由(5.36)计算：

$$\{R\}^e=t\iint_{\Delta}[N]^T\cdot\{g\}\,dxdy\tag{5.37}$$

其中t表示三角形单元的厚度。

如图 5. 4（c）所示，单元的 1-2 边受到面载荷$\{q\}=\left[q_x,q_y\right]^T$的作用。那么其等效节点载荷可由(5.37)计算：

$$\{R\}^e=t\int_{l_{12}}[N]^T\cdot\{q\}\,ds\tag{5.38}$$

计算其他边界上分布载荷的等效节点载荷时，只要将(5.)式中的积分区域换成l_{23}、l_{34}、l_{41}即可。

热应力等效载荷的计算：在平面应力状态下，$\varepsilon_{x0}=\varepsilon_{y0}=\alpha T$，而$\gamma_{xy0}=0$。即由于温度改变产生的初应变列阵为

$$\{\varepsilon_0\}=\left[\alpha T\quad\alpha T\quad 0\right]^T=\alpha T\left[1\quad 1\quad 0\right]^T\tag{5.39}$$

对单元，如果令结点处的温度改变分别为 T_1 、 T_2 、 T_3 、 T_4 ，则单元的温度改变 T 可由 4 个结点处的温度改变插值求出，即

$$T = N_1T_1 + N_2T_2 + N_3T_3 + N_4T_4 \tag{5.40}$$

当考虑温度变化时，平面问题的物理方程为

$$\{\sigma\} = [D](\{\varepsilon\} - \{\varepsilon_0\}) = [D][B]\{\delta\}^e - [D]\{\varepsilon_0\} \tag{5.41}$$

其中 $\{\varepsilon\}$ 为单元中任一点的总应变，$\{\varepsilon_0\}$ 为该点的热应变。根据虚功方程：

$$\{\delta^*\}^{eT}\{F\}^e = \int_{V^e}\{\varepsilon^*\}^T\{\sigma\}dV \tag{5.42}$$

得

$$\{\delta^*\}^{eT}\{F\}^e = \int_{V^e}\{\varepsilon^*\}^T\left([D][B]\{\delta\}^e - [D]\{\varepsilon_0\}\right)dV \tag{5.43}$$

将方程(5.5)代入上式后得

$$\{\delta^*\}^{eT}\{F\}^e = \int_{V^e}\{\delta^*\}^{eT}[B]^T\left([D][B]\{\delta\}^e - [D]\{\varepsilon_0\}\right)dV \tag{5.44}$$

由于上述方程在任意虚位移下都成立，因此系数相等

$$\{F\}^e = [K]^e\{\delta\}^e - \int_{V^e}[B]^T[D]\{\varepsilon_0\}dV \tag{5.45}$$

其中

$$[K]^e = \int_{V^e}[B]^T[D][B]dV \tag{5.46}$$

令

$$\{R_t\}^e = \int_{V^e}[B]^T[D]\{\varepsilon_0\}dV \tag{5.47}$$

上述方程即为考虑了温度改变的单元刚度方程，式中 $\{R_t\}^e$ 单元温度改变的等效节点载荷。

5.5 高斯积分

根据前文所述，计算单元刚度矩阵和等效节点载荷，需要计算如下积分：

$$\left[K_{ij}\right] = \int_{-1}^{1}\int_{-1}^{1}[B_i]^T[D]\left[B_j\right]t|J|d\zeta d\eta \quad (i,j=1,2,\cdots,8) \tag{5.48}$$

$$\{R\}^e = t\iint_{\Delta}[N]^T\cdot\{g\}\,dxdy = t\int_{-1}^{1}\int_{-1}^{1}[N]^T\cdot\{g\}|J|d\zeta d\eta \tag{5.49}$$

$$\{R\}^e = t\int_{l_{12}}[N]^T\cdot\{q\}\,ds = t\int_{-1}^{1}[N]^T\cdot\{q\}\Big|_{\eta=-1}d\zeta \tag{5.50}$$

$$\{R_t\}^e = \int_{V^e}[B]^T[D]\{\varepsilon_0\}dV = t\int_{-1}^{1}\int_{-1}^{1}[B]^T[D]\{\varepsilon_0\}|J|d\zeta d\eta \tag{5.51}$$

对比三角形单元，增加位移函数阶次后上述积分难以给出解析表达式，需要用数值积分方法计算。从方程(5.31)、(5.36)、(5.37)和(5.45)可以看出，只要计算 $\int_{-1}^{1} f(\zeta,\eta)d\zeta$ 和二重积分 $\int_{-1}^{1}\int_{-1}^{1} f(\zeta,\eta)d\zeta d\eta$ 即可。

我们采用高斯积分计算两类积分，高斯积分的理论可参考数值计算方法的教科书。本文只介绍如何使用。根据高斯积分法，定积分 $\int_{-1}^{1} f(\zeta,\eta)d\zeta$ 可用下式近似计算：

$$\int_{-1}^{1} f(\zeta)\mathrm{d}\zeta = \sum_{k=1}^{n} f(\zeta_k)H_k \tag{5.52}$$

其中 n 等于积分点的数量，数值 n 越大，积分精度越高，但是计算量也越大。ζ_k 是积分点坐标，H_k 是对应的权系数。当 n 取不同的数值时，与 n 对应的 ζ_i 和 H_i 值如表 5. 1 所示。比如当 $n=3$ 时，

$$\begin{aligned}\int_{-1}^{1} f(\zeta)\mathrm{d}\zeta &= \sum_{k=1}^{3} f(\zeta_k)H_k = f(\zeta_1)H_1 + f(\zeta_2)H_2 + f(\zeta_3)H_3 \\ &= \frac{5}{9}f(\zeta_1) + \frac{8}{9}f(\zeta_2) + \frac{5}{9}f(\zeta_3)\end{aligned} \tag{5.53}$$

表 5. 1 高斯求积公式积分点坐标与加权系数

点数 n	坐标 ζ_i			加权系数 H_i		
1	0.0			2.0		
2	±0.57735	02691	89626	1.00000	00000	00000
3	±0.77459	66692	41483	0.55555	55555	55555
	1.00000	00000	00000	0.88888	88888	88888
4	±0.86113	63115	94053	0.34785	48451	37454
	±0.33998	10435	84856	0.65214	51548	62546
5	±0.90617	98459	38664	0.23692	68850	56189
	±0.53846	93101	05683	0.47862	86704	99366
	0.00000	00000	00000	0.56888	88888	88889
6	±0.93246	95142	03152	0.17132	44923	79179
	±0.66120	93864	66256	0.36076	15730	48139
	±0.23861	91860	83197	0.46791	39345	72691

二重积分 $\int_{-1}^{1}\int_{-1}^{1} f(\zeta,\eta)d\zeta d\eta$ 可以转换为二次积分。把 η 当作常量，先对 ζ 进行积分，即

$$\int_{-1}^{1}\int_{-1}^{1} f(\zeta,\eta)\mathrm{d}\zeta\,\mathrm{d}\eta=\int_{-1}^{1}\left(\int_{-1}^{1} f(\zeta,\eta)\mathrm{d}\zeta\right)\mathrm{d}\eta\approx\int_{-1}^{1}\left(\sum_{k=1}^{n} f(\zeta_k,\eta)H_k\right)\mathrm{d}\eta=\sum_{k=1}^{n}H_k\int_{-1}^{1} f(\zeta_k,\eta)\mathrm{d}\eta \quad (5.54)$$

然后再对η进行近似积分，得到

$$\int_{-1}^{1}\int_{-1}^{1} f(\zeta,\eta)\mathrm{d}\zeta\,\mathrm{d}\eta\approx\sum_{k=1}^{n}\sum_{j=1}^{n} f(\zeta_k,\eta_j)H_kH_j \quad (5.55)$$

这就是二维高斯积分公式。其中ζ_k或η_j均为高斯积分点，而H_k和H_j为相应的求积加权系数，单元内的积分点数为n^2个。

采用高斯积分法计算方程(5.31)时，首先将积分内的矩阵写成ζ和η的函数

$$\left[\tilde{K}_{ij}(\zeta,\eta)\right]=[B_i]^T[D][B_j]t|J| \quad (5.56)$$

然后采用高斯积分近似计算，假设$n=2$，查表 5．1，只取小数点后 5 位得：

$$\begin{aligned}&\int_{-1}^{1}\int_{-1}^{1}\left[\tilde{K}_{ij}(\zeta,\eta)\right]\mathrm{d}\zeta\,\mathrm{d}\eta\approx\sum_{k=1}^{2}\sum_{l=1}^{2}\left[\tilde{K}_{ij}(\zeta_k,\eta_l)\right]H_kH_l\\&=\left[\tilde{K}_{ij}(-0.57735,-0.57735)\right]+\left[\tilde{K}_{ij}(-0.57735,0.57735)\right]\\&+\left[\tilde{K}_{ij}(0.57735,-0.57735)\right]+\left[\tilde{K}_{ij}(0.57735,0.57735)\right]\end{aligned} \quad (5.57)$$

方程（5.36）（5.37）（5.46）的积分过程类似。

5.6 导出有限元方程

本章前几小节介绍了单元刚度矩阵的计算和单元载荷向量的计算，此时还需要导出有限元方程。方程的导出方法与第三章类似，需要用到势能泛函。根据最小位能原理，在弹性范围内，平面问题中的弹性力学基本方程及边界条件与如下能量泛函的极值条件是等价的。

$$\Pi=\int_{\Omega}\frac{1}{2}\{\varepsilon\}^T\cdot[D]\cdot\{\varepsilon\}\,t\mathrm{d}x\mathrm{d}y-\int_{\Omega}\{f\}^T\cdot\{g\}\,t\mathrm{d}x\mathrm{d}y-\int_{\Gamma 2}\{f\}^T\cdot\{q\}\,t\mathrm{d}s \quad (5.58)$$

其中，Ω是研究对象所占区域，Γ_2是研究对象的力边界，t是二维体厚度；$\{g\}$是作用在二维体内的体积力；$\{q\}$是作用在Γ_2上的面积力。当结构离散化后，根据积分的性质，方程(5.52)中的积分可以表示为各个单元内积分之和：

$$\Pi=\sum_{e=1}^{N_e}\int_{\Omega^e}\frac{1}{2}\{\varepsilon\}^T\cdot[D]\cdot\{\varepsilon\}\,t\mathrm{d}x\mathrm{d}y-\sum_{e=1}^{N_e}\int_{\Omega^e}\{f\}^T\cdot\{g\}\,t\mathrm{d}x\mathrm{d}y-\sum_{e=1}^{N_e}\int_{\Gamma_2^e}\{f\}^T\cdot\{q\}\,t\mathrm{d}s \quad (5.59)$$

其中Ω^e是第e个单元所占区域，Γ_2^e是第e个单元的力边界。以平面四节点等参元为例，将(5.32)和(5.26)代入(5.52)后消去$\{\varepsilon\}$和$\{f\}$后得：

$$\Pi=\sum_{e=1}^{N_e}\{\delta\}^{eT}\cdot\int_{\Omega^e}\frac{1}{2}[B]^T\cdot[D]\cdot[B]tdxdy\cdot\{\delta\}^e$$
$$-\sum_{e=1}^{N_e}\{\delta\}^{eT}\cdot\int_{\Omega^e}[N]^T\cdot\{g\}tdxdy-\sum_{e=1}^{N_e}\{\delta\}^{eT}\int_{\Gamma_2^e}[N]^T\cdot\{q\}tds \tag{5.60}$$

为了便于计算，我们将所有的节点位移排列成一个向量：

$$\{\delta\}^T=[u_1,v_1,u_2,v_2\cdots u_i,v_i,\cdots u_{Ne},v_{Ne}]^T \tag{5.61}$$

单元节点位移$\{\delta\}^{eT}$实际上是从$\{\delta\}^T$抽取八个元素构成的一个向量。我们可以通过抽取矩阵$[G]^e_{8\times 2Ne}$建立$\{\delta\}^T$与$\{\delta\}^{eT}$之间的关系：

$$\{\delta\}^e=[G]^e_{8\times 2Ne}\cdot\{\delta\} \tag{5.62}$$

将方程(5.56)代入(5.54)消去$\{\delta\}^e$后得：

$$\Pi=\sum_{e=1}^{N_e}\{\delta\}^T\cdot[G]^{eT}\cdot\int_{\Omega^e}\frac{1}{2}[B]^T\cdot[D]\cdot[B]tdxdy\cdot[G]^e\cdot\{\delta\}$$
$$-\sum_{e=1}^{N_e}\{\delta\}^T\cdot[G]^{eT}\cdot\int_{\Omega^e}[N]^T\cdot\{g\}tdxdy-\sum_{e=1}^{N_e}\{\delta\}^T\cdot[G]^{eT}\cdot\int_{\Gamma_2^e}[N]^T\cdot\{q\}tds \tag{5.63}$$

令

$$[K]^e=\int_{\Omega^e}[B]^T\cdot[D]\cdot[B]tdxdy\quad[P_g]^e=\int_{\Omega^e}[N]^T\cdot\{g\}tdxdy$$
$$[P_q]^e=\int_{\Gamma_2^e}[N]^T\cdot\{q\}tds\qquad[P]^e=[P_g]^e+[P_q]^e \tag{5.64}$$

其中$[K]^e$和$[P]^e$分别是单元刚度矩阵和单元等效节点载荷向量。

进一步地，令

$$[K]=\sum_{e=1}^{Ne}[G]^{eT}\cdot[K]^e\cdot[G]^e\quad[P]=\sum_{e=1}^{Ne}[G]^{eT}\cdot[P]^e \tag{5.65}$$

其中$[K]$是总体刚度矩阵，$[P]$是整体载荷向量。方程(5.57)表示成如下形式：

$$\Pi=\frac{1}{2}\{\delta\}^T\cdot[K]\cdot\{\delta\}-\{\delta\}^T\cdot[P] \tag{5.66}$$

根据最小位能原理，结构平衡时的位移，应该是满足位移边界条件，并且能够使泛函Π取极小值。由变分原理可知，泛函Π取极值的条件是它的一次变分等于零，即

$$\frac{\partial\Pi}{\partial\{\delta\}}=[K]\cdot\{\delta\}-[P]=0 \tag{5.67}$$

这样，我们就得到了有限元方程：

$$[K]\cdot\{\delta\}-[P]=0 \tag{5.68}$$

对比本节和 3.5 节，我们可以看出，两个过程基本相同。所以不论什么单元，单元刚度矩阵和单元载荷向量都可以写成(5.58)的形式，总体刚度矩阵和总体载荷向量都可以写成(5.59)的形式。

5.7 总体刚度矩阵和载荷向量的合成

根据等式(5.59)可以看出，单元刚度矩阵$[K]^e$和单元载荷向量$[P]^e$都通过$[G]^e$映射到总体刚度矩阵。由等式(5.56)可知，$[G]^{eT}$的元素具有如下性质：

$$g_{ij}^{eT}=\begin{cases}1 & \text{单元}e\text{的第}j\text{个自由度就是总体位移向量的第}i\text{个自由度}\\ 0 & \text{其他}\end{cases} \tag{5.69}$$

其中g_{ij}^{eT}表示矩阵$[G]^{eT}$的第i行和j列个元素。因此$[G]^{eT}$与$[P]^e$相乘的过程就是根据局部结点位移投射到总体位移矢量的过程。我们在计算$[P]$时没有必要构造$[G]^{eT}$，只要计算出$[P]^e$，然后查出单元 e 的第 j 个自由度，对应的总体位移向量中的第i个自由度，然后将P_j^e直接叠加到P_i上，即

$$P_i=P_i+P_j^e \tag{5.70}$$

当然要使上述运算正确，P_i的初始值等于零。

计算总体刚度矩阵时，$[K]^e$要经过两次乘法运算然后再叠加，为了展示中间计算过程，我们令

$$[\tilde{K}]=[G]^{eT}\cdot[K]^e\cdot[G]^e \tag{5.71}$$

表示成张量乘积

$$\tilde{K}_{ij}=\sum_{m=1}^{8}\sum_{l=1}^{8}g_{il}^{eT}\cdot K_{lm}^e\cdot g_{mj}^e \tag{5.72}$$

g_{il}^{eT}的取值及含义见(5.63)，g_{mj}^e的取值如下：

$$g_{mj}^e=\begin{cases}1 & \text{单元}e\text{的第}m\text{个自由度就是总体位移向量的第}j\text{个自由度}\\ 0 & \text{其他}\end{cases} \tag{5.73}$$

计算的时候，不需要构造$[G]^e$矩阵，只要先查到单元 e 的第l个自由度和第 m 个自由度分别对应的总体位移向量里的第i个自由度和第 j 个自由度。然后将K_{lm}^e叠加到$\tilde{K}_{ij}$即可。最后将所有的$\tilde{K}_{ij}$叠加到K_{ij}上即可。程序设计的时候，这一过程实际上也没有必要构造$\tilde{K}_{ij}$，只要构造一个$[K]$矩阵，然

后对每一个元素清零，之后按照下式计算：

$$K_{ij} = K_{ij} + K^{e}_{lm} \tag{5.74}$$

其中含义是，找到第 e 个单元第l个自由度和第 m 个自由度对应的全局位移向量中的第i个自由度和第j个自由度，然后叠加到K_{ij}即可。

5.8 应力计算

对方程(5.62)施加位移边界条件后，求解，即可获得所有结点位移。然后应用(5.40)计算出单元内的应力。需要注意的是，不同的三角形单元，四节点等参元的应力在单元内部不是一个恒定值，是局部坐标 ζ 和 η 的函数。

此外，应变矩阵 **B** 是插值函数 N 对坐标进行求导后得到的矩阵。求导一次，插值多项式的次数就降低一次。所以通过导数运算得到的应变ε和应力σ的精度较位移u降低了，即计算的应力和应变与真实解误差增加。具体表现在：

（1）单元内部的应力一般不能满足平衡方程；

（2）应力在单元的边界上一般不连续；

（3）在力的边界 Γ_2 上一般也不能满足里的边界条件。

为了得到较为合理的应力结果，一般需要对应力进行处理。最简单的处理应力结果的方法是取相邻单元或者围绕结点的各单元应力的平均值。

最简单的方法是取相邻单元应力的平均值，这种方法最常用于 3 结点三角形单元中。这种简单而又实用的单元得到的应力解在单元内是常数。可以将其看作是单元内应力的平均值，或是单元形心处的应力。由于应力近似解总是在精确解上下振荡，可以取相邻单元应力的平均值作为此两个单元合成的较大四边形单元形心处的应力。这样处理常常能取得比较好的结果。

取平均应力可以采用算术平均，即

$$\bar{\sigma} = \frac{1}{N}\sum_{i=1}^{N}\sigma_i \tag{5.75}$$

其中 $\bar{\sigma}$ 是某节点的应力，N 是围绕该节点的单元总数，σ_i 是该单元的应力。也可以采用精确一些的面积加权平均，即

$$\bar{\sigma} = \frac{1}{N}\sum_{i=1}^{N}S_i\sigma_i \tag{5.76}$$

其中 S_i 是单元 i 的面积。对于四结点等参元，可以用单元内的平均应力代替等式(5.70)中的 σ_i

即可。

5.9 平面四节点等参元的 C 语言实现

数据文件：data02.txt 或者 data0201.txt

```
#include"stdio.h"
#include"stdlib.h"
#include"math.h"
#include"matlab.h"
double E,mu;
int num_elem,num_node;
int *elem;
double *node;
double *u;
//读取并校核文件
void readfile_quar()
{
    int i,j;
    char filename[32];
    //读取文件
    FILE *fp;
    printf("请输入要打开的文件名\n 提示：数据文件名为：data02.txt\tdata0201.txt\n");
    scanf("%s",filename);
    fp=fopen(filename,"r");
    if(fp==NULL)
    {
        printf("当前目录下无此文件\n");
        system("pause");
        exit(1);
    }
```

```
else
{
    printf("打开成功\n");
}
fscanf(fp,"%lf%lf",&E,&mu);
fscanf(fp,"%d%d",&num_elem,&num_node);
if(elem!=NULL)
{
    free(elem);
}
if(node!=NULL)
{
    free(node);
}
elem=(int *)malloc(sizeof(int)*4*num_elem);
node=(double *)malloc(sizeof(double)*6*num_node);
for(i=0;i<num_elem;i++)
{
    for(j=0;j<4;j++)
    {
        fscanf(fp,"%d",&elem[4*i+j]);
    }
}
for(i=0;i<num_node;i++)
{
    for(j=0;j<6;j++)
    {
        fscanf(fp,"%lf",&node[6*i+j]);
    }
}
```

```
    fclose(fp);
    //输出到文件
    FILE *fc;
    fc=fopen("check02.txt","w");
    fprintf(fc,"%lf    %lf\n",E,mu);
    fprintf(fc,"%d    %d\n",num_elem,num_node);
    for(i=0;i<num_elem;i++)
    {
        for(j=0;j<4;j++)
        {
            fprintf(fc,"%d  ",elem[4*i+j]);
        }
        fprintf(fc,"\n");
    }
    for(i=0;i<num_node;i++)
    {
        for(j=0;j<6;j++)
        {
            fprintf(fc,"%lf  ",node[6*i+j]);
        }
        fprintf(fc,"\n");
    }
    fclose(fc);
}
//计算单元刚度矩阵，单元载荷矩阵
void elem_severity_quar(double (*Ke)[8],double *load,int elem_id)
{
    int m,n,i,j,k;
    double J[4]={-0.8611363116,-0.3399810436,0.3399810436,0.8611363116};
    double H[4]={0.3478548451,0.6521451549,0.6521451549,0.3478548451};
```

```
double *Ke_temp;
Ke_temp=(double *)malloc(sizeof(double)*8*8);
//计算 Ke
for(i=0;i<8;i++)
{
    for(j=0;j<8;j++)
    {
        Ke[i][j]=0;
    }
}
for(m=0;m<4;m++)
{
    for(n=0;n<4;n++)
    {
        double D[3][3]={0},B[3][8]={0},Bt[8][3]={0},DB[3][8]={0},BDB[8][8]={0};
        double x1,x2,x3,x4,y1,y2,y3,y4,X,Y,Jh;
        X=J[m];Y=J[n];
        D[0][0]=E/(1-mu*mu);D[0][1]=E*mu/(1-mu*mu);
        D[1][0]=D[0][1];D[1][1]=D[0][0];
        D[2][2]=E*((1-mu)/2)/(1-mu*mu);
        x1=node[6*(elem[4*elem_id]-1)];
        x2=node[6*(elem[4*elem_id+1]-1)];
        x3=node[6*(elem[4*elem_id+2]-1)];
        x4=node[6*(elem[4*elem_id+3]-1)];
        y1=node[6*(elem[4*elem_id]-1)+1];
        y2=node[6*(elem[4*elem_id+1]-1)+1];
        y3=node[6*(elem[4*elem_id+2]-1)+1];
        y4=node[6*(elem[4*elem_id+3]-1)+1];
        Jh=(0.25*((-1+Y)*x1+(1-Y)*x2+(1+Y)*x3+(-1-Y)*x4))*(0.25*((-1+X)*y1+(-1-
X)*y2+(1+X)*y3+(1-X)*y4))-(0.25*((-1+Y)*y1+(1-Y)*y2+(1+Y)*y3+(-1-Y)*y4))*(0.25*((-1+X)*x1+(-
```

```
1-X)*x2+(1+X)*x3+(1-X)*x4));
                B[0][0]=(0.25*((-1+X)*y1+(-1-X)*y2+(1+X)*y3+(1-X)*y4)*0.25*(Y-1)-0.25*((-
1+Y)*y1+(1-Y)*y2+(1+Y)*y3+(-1-Y)*y4)*0.25*(X-1))/Jh;
                B[0][2]=(0.25*((-1+X)*y1+(-1-X)*y2+(1+X)*y3+(1-X)*y4)*0.25*(-Y+1)-0.25*((-
1+Y)*y1+(1-Y)*y2+(1+Y)*y3+(-1-Y)*y4)*0.25*(-X-1))/Jh;
                B[0][4]=(0.25*((-1+X)*y1+(-1-X)*y2+(1+X)*y3+(1-X)*y4)*0.25*(Y+1)-0.25*((-
1+Y)*y1+(1-Y)*y2+(1+Y)*y3+(-1-Y)*y4)*0.25*(X+1))/Jh;
                B[0][6]=(0.25*((-1+X)*y1+(-1-X)*y2+(1+X)*y3+(1-X)*y4)*0.25*(-Y-1)-0.25*((-
1+Y)*y1+(1-Y)*y2+(1+Y)*y3+(-1-Y)*y4)*0.25*(-X+1))/Jh;
                B[1][1]=(-0.25*((-1+X)*x1+(-1-X)*x2+(1+X)*x3+(1-X)*x4)*0.25*(Y-1)+0.25*((-
1+Y)*x1+(1-Y)*x2+(1+Y)*x3+(-1-Y)*x4)*0.25*(X-1))/Jh;
                B[1][3]=(-0.25*((-1+X)*x1+(-1-X)*x2+(1+X)*x3+(1-X)*x4)*0.25*(-Y+1)+0.25*((-
1+Y)*x1+(1-Y)*x2+(1+Y)*x3+(-1-Y)*x4)*0.25*(-X-1))/Jh;
                B[1][5]=(-0.25*((-1+X)*x1+(-1-X)*x2+(1+X)*x3+(1-X)*x4)*0.25*(Y+1)+0.25*((-
1+Y)*x1+(1-Y)*x2+(1+Y)*x3+(-1-Y)*x4)*0.25*(X+1))/Jh;
                B[1][7]=(-0.25*((-1+X)*x1+(-1-X)*x2+(1+X)*x3+(1-X)*x4)*0.25*(-Y-1)+0.25*((-
1+Y)*x1+(1-Y)*x2+(1+Y)*x3+(-1-Y)*x4)*0.25*(-X+1))/Jh;

B[2][0]=B[1][1];B[2][1]=B[0][0];B[2][2]=B[1][3];B[2][3]=B[0][2];B[2][4]=B[1][5];B[2][5]=B[0][4];B[2
[6]=B[1][7];B[2][7]=B[0][6];
                for(i=0;i<3;i++)
                {
                    for(j=0;j<8;j++)
                    {
                        Bt[j][i]=B[i][j];
                    }
                }
                //计算 DB
                for(i=0;i<3;i++)
                {
```

```
        for(j=0;j<8;j++)
        {
            for(k=0;k<3;k++)
            {
                DB[i][j]=DB[i][j]+D[i][k]*B[k][j];
            }
        }
    }
    //计算 BDB
    for(i=0;i<8;i++)
    {
        for(j=0;j<8;j++)
        {
            for(k=0;k<3;k++)
            {
                BDB[i][j]=BDB[i][j]+Bt[i][k]*DB[k][j];
            }
         Ke_temp[8*i+j]=BDB[i][j];
        }
     }
    for(i=0;i<8;i++)
     {
          for(j=0;j<8;j++)
          {
              Ke_temp[8*i+j]*=Jh*H[m]*H[n];
              Ke[i][j]+=Ke_temp[8*i+j];
          }
     }

}
```

```
        }
        free(Ke_temp);
        //单元载荷矩阵
        for(i=0;i<4;i++)
        {
            load[2*i]=node[6*(elem[4*elem_id+i]-1)+4];
            load[2*i+1]=node[6*(elem[4*elem_id+i]-1)+5];
        }
    }
    //求解总体刚度矩阵，总体载荷矩阵
    void solve_quar()
    {
        double Ke[8][8];double *load;double *L;double *K,*KN;int id[8];double temp,tt=0;int t;int c;
        int i,j,m;
        int *uvid;int nodes=0;
        //重新定义节点约束
        uvid=(int *)malloc(2*sizeof(int)*num_node);
        load=(double *)malloc(8*sizeof(double));
        for(i=0;i<num_node;i++)
        {
            if(node[6*i+2]!=0)
            {
                uvid[2*i]=nodes;
                nodes++;
            }
            else
            {
                uvid[2*i]=-1;
            }
            if(node[6*i+3]!=0)
```

```
        {
            uvid[2*i+1]=nodes;
            nodes++;
        }
        else
        {
            uvid[2*i+1]=-1;
        }
}

K=(double *)malloc(sizeof(double)*nodes*nodes);
L=(double *)malloc(sizeof(double)*nodes);
for(i=0;i<nodes*nodes;i++)
{
    K[i]=0;
}
for(i=0;i<nodes;i++)
{
    L[i]=0;
}
for(m=0;m<num_elem;m++)
{
    for(i=0;i<4;i++)
    {
        id[2*i]=uvid[2*(elem[4*m+i]-1)];
        id[2*i+1]=uvid[2*(elem[4*m+i]-1)+1];
        //printf("%d    %d\n",id[2*i],id[2*i+1]);
    }

    elem_severity_quar(Ke,load,m);
```

```
        for(i=0;i<8;i++)
        {
            for(j=0;j<8;j++)
            {
                if(id[i]>=0&&id[j]>=0)
                {
                    K[id[i]*nodes+id[j]]+=Ke[i][j];
                }
                if(id[i]>=0)
                {
                    L[id[i]]=load[i];
                }
            }
        }
    }
    //验证总体刚度矩阵
    printf("划去与约束位移对应的行与列后，总体刚度矩阵为：\n");
    for(i=1;i<=nodes*nodes;i++)
    {
        printf("%lf\t",K[i-1]);
        if(i%nodes==0)
        {
            printf("\n");
        }
    }
    //验证总体载荷矩阵
    printf("划去与约束位移对应的行与列后，总体载荷矩阵为:\n");
    for(i=0;i<nodes;i++)
    {
```

```
        printf("%lf\n",L[i]);
    }
    printf("请选择求解方程的方法：\n1：高斯消去法\n2：求逆矩阵法\n");
    scanf("%d",&c);
    if(c==1)
    {
        //高斯消去法求解方程
        KN=(double *)malloc(sizeof(double)*nodes*nodes);
        for(m=0;m<nodes;m++)
        {
        //全选主元
        temp=K[m*nodes+m];
        t=m;
        for(i=m;i<nodes;i++)
        {
            if(fabs(K[i*nodes+m])>fabs(temp))
            {
                temp=K[i*nodes+m];
                t=i;
            }
        }
        if(t!=m)
        {
            for(j=m;j<nodes;j++)
            {
                tt=K[m*nodes+j];K[m*nodes+j]=K[t*nodes+j];K[t*nodes+j]=tt;
            }
                tt=L[m];L[m]=L[t];L[t]=tt;
        }
        L[m]=L[m]/K[m*nodes+m];
```

```
        KN[m*nodes+m]=K[m*nodes+m];
        for(j=m;j<nodes;j++)
        {
            K[m*nodes+j]=K[m*nodes+j]/KN[m*nodes+m];
        }
        for(i=m+1;i<nodes;i++)
        {
            KN[i*nodes+m]=K[i*nodes+m]/K[m*nodes+m];

            for(j=m;j<nodes;j++)
            {
                K[i*nodes+j]=K[i*nodes+j]-KN[i*nodes+m]*K[m*nodes+j];

            }
            L[i]=L[i]-KN[i*nodes+m]*L[m];
        }
    }
     //回代
    u=(double *)malloc(sizeof(double)*nodes);
    u[nodes-1]=L[nodes-1];
    for(i=nodes-2;i>=0;i--)
     {
        temp=0;
        for(j=i+1;j<nodes;j++)
        {
            temp=temp+K[i*nodes+j]*u[j];
        }
        u[i]=L[i]-temp;
     }
}
```

```
        else
        {
            u=(double *)malloc(sizeof(double)*nodes);
            for(i=0;i<nodes;i++)
            {
                u[i]=0;
            }
            mx_inver(K,nodes);
            for(i=0;i<nodes;i++)
            {
                for(m=0;m<nodes;m++)
                {
                    u[i]=u[i]+K[i*nodes+m]*L[m];
                }
            }
        }
        printf("节点位移为：\n");
        for(i=0;i<nodes;i++)
        {
            printf("%lf\n",u[i]);
        }
}
//输出节点位移
void writefile_quar()
{
        int i,j=0;
        double temp=0;
        FILE *fw;
        fw=fopen("uxuy02.txt","w");
        fprintf(fw,"节点位移为：\n");
```

```
        for(i=0;i<num_node;i++)
        {
            if(node[6*i+2]!=0)
            {
                fprintf(fw,"%lf\n",u[j]);
                j++;
            }
            else
            {
                fprintf(fw,"%lf\n",temp);
            }
            if(node[6*i+3]!=0)
            {
                fprintf(fw,"%lf\n",u[j]);
                j++;
            }
            else
            {
                fprintf(fw,"%lf\n",temp);
            }
        }
        printf("节点位移已成功输出到 uxuy02.txt\n");
    }
    //主程序
    void main()
    {
        readfile_quar();
        solve_quar();
        writefile_quar();
        return;
```

}

5.10 小结与习题

本章介绍了平面等参单元的基本概念和计算流程。请完成以下作业：

（1）推导平面四结点和八结点等参元位移函数表达式（形函数形式）；

（2）推导平面四结点和八结点等参元节点位移与单元应变、单元应力的关系式；

（3）应用能量泛函最小原理推导有限元方程；

（4）应用虚功相等原理推导单元刚度矩阵表达式；

（5）采用虚功相等原理推导集中载荷、体积力、表面力和热膨胀的等效节点载荷；

（6）编写平面四节点等参元的有限元计算程序。

第六章 三维等参单元与高阶单元

6.1 拉格朗日插值函数

可以用拉格朗日插值函数写出单元内任一点的位移表达式。在一维情况下，拉格朗日插值函数的一般形式为：

$$L_i^n(x)=\frac{(x-x_0)(x-x_1)...(x-x_{i-1})(x-x_{i+1})...(x-x_n)}{(x_i-x_0)(x_i-x_1)...(x_i-x_{i-1})(x_i-x_{i+1})...(x_i-x_n)}=\prod_{\substack{m=0\\m\neq i}}^{n}\frac{x-x_m}{x_i-x_m} \tag{6.1}$$

式中 $x_0,x_1,\ldots,x_n$ 为 n+1 个节点的 x 坐标值。这是一个 n 次多项式，由 n 个因子组成。当 $x=x_i$ 时，分子分母相等，多项式值为 1。当 $x=x_m(m\neq i)$时，多项式值为零。利用这种插值函数，只要 $\varphi(x)$在 $x_0,x_1,\ldots,x_n$ 处的值 $\varphi_0,\varphi_1,\ldots,\varphi_n$ 已知，就能用下列 n 次多项式近似地表示 $\varphi(x)$，即

$$\varphi(x)=\sum_{i=0}^{n}L_i^n(x)\varphi_i \tag{6.2}$$

显而易见，$L_i^n(x)$ 具有如下性质：

$$L_i^n(x_k)=\begin{cases}0 & k\neq i\\ 1 & k=i\end{cases} \tag{6.3}$$

这与形函数的定义是一致的。还可以将拉格朗日插值函数用于二维或三维情况。二维问题的位移函数可写成：

$$\varphi(x,y)=\sum_{i=0}^{n}\sum_{j=0}^{m}L_i^n(x)L_j^m(y)\varphi_{ij} \tag{6.4}$$

式中，n 和 m 分别为在 x 和 y 方向的分段数。二维拉格朗日插值函数可写成：

$$L_{ij}^{nm}(x,y)=L_i^n(x)L_j^m(y) \tag{6.5}$$

6.2 九节点正方形单元的形函数

第五章我们学习了平面四节点单元的位移函数。该形函数也可以直接采用拉格朗日插值函数直接构造：

$$u(\xi,\eta)=\sum_{i=0}^{1}\sum_{j=0}^{1}L_i^1(\xi)L_j^1(\eta)u_{ij} \tag{6.6}$$

其中 u_{ij} 表示单元四个角节点的位移。

因为

$$L_0^1(\xi)=\frac{1}{2}(1-\xi),L_1^1(\xi)=\frac{1}{2}(1+\xi)\text{，}\quad L_0^1(\eta)=\frac{1}{2}(1-\eta),L_1^1(\eta)=\frac{1}{2}(1+\eta) \tag{6.7}$$

展开后得：

$$\begin{aligned}u(\xi,\eta)&=\frac{1}{4}(1-\xi)(1-\eta)u_{00}+\frac{1}{4}(1-\xi)(1+\eta)u_{01}\\&+\frac{1}{4}(1+\xi)(1-\eta)u_{10}+\frac{1}{4}(1+\xi)(1+\eta)u_{11}\end{aligned} \tag{6.8}$$

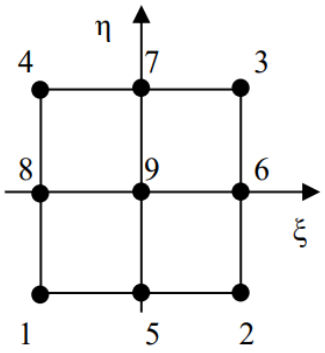


图 6.1 九节点单元

显然，如果提高拉格朗日插值函数的阶次就能得到高阶单元。例如图 6.1 所示九结点单元，令式(6.4)的 m 和 n 都等于 2，得：

$$u(\xi,\eta)=\sum_{i=0}^{2}\sum_{j=0}^{2}L_i^2(\xi)L_j^2(\eta)u_{ij} \tag{6.9}$$

其中

$$\begin{cases}L_0^2(\xi)=\frac{1}{2}\xi(\xi-1)\\L_1^2(\xi)=(1-\xi^2)\\L_2^2(\xi)=\frac{1}{2}\xi(1+\xi)\end{cases}\quad\begin{cases}L_0^2(\eta)=\frac{1}{2}\eta(\eta-1)\\L_1^2(\eta)=(1-\eta^2)\\L_2^2(\eta)=\frac{1}{2}\eta(1+\eta)\end{cases}$$

位移函数也可以写成形函数形式：

$$u=\sum_{i=1}^{9}N_iu_i \tag{6.10}$$

其中，

$N_1(\xi,\eta)=L_0^2(\xi)L_0^2(\eta)$， $N_2(\xi,\eta)=L_2^2(\xi)L_0^2(\eta)$， $N_3(\xi,\eta)=L_2^2(\xi)L_2^2(\eta)$， $N_4(\xi,\eta)=L_0^2(\xi)L_2^2(\eta)$

$N_5(\xi,\eta)=L_1^2(\xi)L_0^2(\eta)$， $N_6(\xi,\eta)=L_2^2(\xi)L_1^2(\eta)$， $N_7(\xi,\eta)=L_1^2(\xi)L_2^2(\eta)$， $N_8(\xi,\eta)=L_0^2(\xi)L_1^2(\eta)$，

$N_9(\xi,\eta)=L_1^2(\xi)L_1^2(\eta)$

根据收敛性要求，该单元得位移函数需要满足四个条件：（1）包含刚体位移；（2）包含常应变项目；（3）单元内连续；（4）单元之间连续。前三条请读者自己证明。本文证明单元之间的连续性。

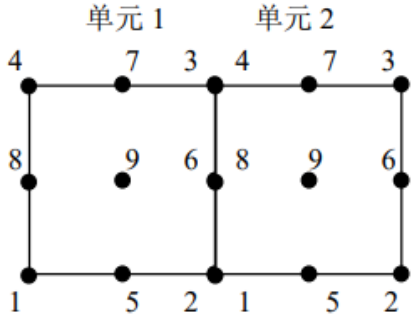


图 6.2 两个相邻九节点单元

假设有两个相邻单元，分别是单元 1 和单元 2。两个单元的位移分别是 $\boldsymbol{u}^1$ 和 $\boldsymbol{u}^2$ 。则 $u^1=\sum_{i=1}^{9}N_i^1u_i^1$，$u^2=\sum_{i=1}^{9}N_i^2u_i^2$ 其中右上标整数表示单元编号。在公共边上，对于单元 1，$L_0^2(\xi)=L_1^2(\xi)=0$。所以 $N_1^1=N_4^1=N_5^1=N_7^1=N_8^1=N_9^1=0$，则 $u^1=N_2^1u_2^1+N_3^1u_3^1+N_6^1u_6^1$。同理：$u^2=N_1^2u_1^2+N_4^2u_4^2+N_8^2u_8^2$。

在公共边上：$u_2^1=u_1^2$， $u_3^1=u_4^2$， $u_6^1=u_8^2$。所以 $u^1-u^2=\left(N_2^1-N_1^2\right)u_2^1+\left(N_3^1-N_4^2\right)u_3^1+\left(N_6^1-N_8^2\right)u_6^1$，$N_2^1=L_2^2(\xi)L_0^2(\eta)$，公共边上 $L_2^2(\xi)=1$，所以 $N_2^1=L_0^2(\eta)$， $N_1^2=L_0^2(\xi)L_0^2(\eta)$，公共边上 $L_0^2(\xi)=1$，所以 $N_1^2=L_0^2(\eta)$，即 $\left(N_2^1-N_1^2\right)=0$。

同理可证 $\left(N_3^1-N_4^2\right)=0$，$\left(N_6^1-N_8^2\right)=0$。所以公共边上 $u^1-u^2\equiv0$。这就证明了九节点正方形单元位移在公共边上具有连续性。

6.3 八节点正方形单元的位移函数

九节点单元提高了单元的阶次，但是其含有单元中间节点。工程上更多应用无中间节点的单元。一种直接的思路是直接从九节点单元中去掉中间结点形成八节点单元。

$$u(\xi,\eta)=\sum_{i=0}^{8}N_i(\xi,\eta)u_i \tag{6.11}$$

式中形函数表达式与式(6.4)相同。但是该位移函数不满足收敛性要求。因为在 $(0,0)$ 处 u 恒等于零。

说明该表达式不包含刚体位移，需要进行修正。

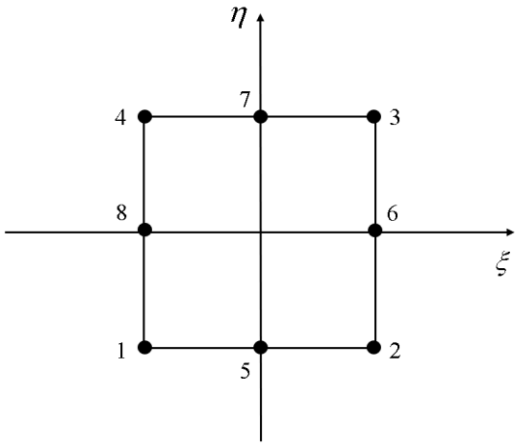


图 6.3 八节点单元

注意到该类型单元包含四个角节点和四个边中节点。我们首先写出中间节点的形函数：

$$N_5(\xi,\eta)=L_1^2(\xi)L_0^1(\eta)=\frac{1}{2}(1-\xi^2)(1-\eta) \tag{6.12}$$

对于$\xi_i=0$（$i=5,7$）的边中节点，其形函数的表达式为

$$N_i(\xi,\eta)=\frac{1}{2}(1-\xi^2)(1+\eta_i\eta)\qquad(i=5,7) \tag{6.13}$$

同理对于$\eta_i=0$（$i=6,8$）的边中节点的形函数为

$$N_i(\xi,\eta)=\frac{1}{2}(1+\xi_i\xi)(1-\eta^2)\qquad(i=6,8) \tag{6.14}$$

以上推导了边中节点的形函数，角节点的形函数构成要复杂一些。若把角节点（例如结点 1）的形函数 N_c 写成 $N_c(\xi,\eta)=L_0^2(\xi)L_0^1(\eta)$，则在$\xi$=-1 边上节点 8 处 N_c 的值不等于零，而等于 0.5。因此，需要对 N_c 进行修正，使角节点的形函数在除结点 1 外的所有其它节点的值都等于零。修正的办法是：

$$\begin{aligned}N_1(\xi,\eta)&=N_c(\xi,\eta)-\frac{1}{2}N_8(\xi,\eta)=L_0^2(\xi)L_0^1(\eta)-\frac{1}{2}L_0^1(\xi)L_1^1(\eta)\\&=\frac{1}{4}(1-\xi)(1-\eta)(-\xi-\eta-1)\end{aligned} \tag{6.15}$$

对所有角节点上的形函数，写成统一表达式

$$N_i(\xi,\eta)=\frac{1}{4}(1+\xi_i\xi)(1+\eta_i\eta)(\xi_i\xi+\eta_i\eta-1)\quad(i=1,2,3,4) \tag{6.16}$$

式中ξ_i, η_i 表示该节点的局部坐标值。

6.4 正六面体单元的形函数

正六面体单元是由平面单元引伸而来，形函数的求法与平面单元基本相同，只是稍微复杂而已。

正六面体单元是研究任意六面体和曲六面体单元的基础。通过坐标变换，可以把正六面体转换成任意六面体或曲六面体。

我们首先研究八节点六面体单元的位移函数。可以直接由拉格朗日插值函数构造：

$$u(\xi,\eta,\varsigma)=\sum_{i=0}^{1}\sum_{j=0}^{1}\sum_{k=0}^{1}L_i^1(\xi)L_j^1(\eta)L_k^1(\varsigma)u_{ijk} \tag{6.17}$$

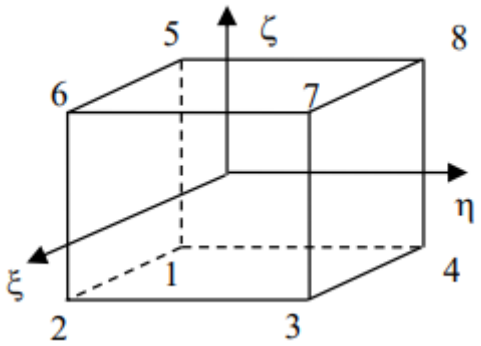


图 6.4 八节点正六面体单元

展开后得：

$$u(\xi,\eta,\varsigma)=N_i(\xi,\eta,\varsigma)u_i \qquad (i=1,2,\ldots,8) \tag{6.18}$$

$$N_i(\xi,\eta,\zeta)=\frac{1}{8}(1+\xi_i\xi)(1+\eta_i\eta)(1+\zeta_i\zeta) \quad (i=1,2,\ldots,8) \tag{6.19}$$

式中ξ_i, η_i, ζ_i表示该结点的局部坐标值。

6.5 构造形函数的画线（面）法

拉格朗日插值函数可以直接构造 4 节点、9 节点平面四边形单元和 8 节点正六面体单元，但是不能直接构造 8 节点平面四边形单元和 20 节点正六面体单元，具有一定的局限性。下面介绍作者提出的划线（面）法，可以方便的直接构造上述所有类型单元。

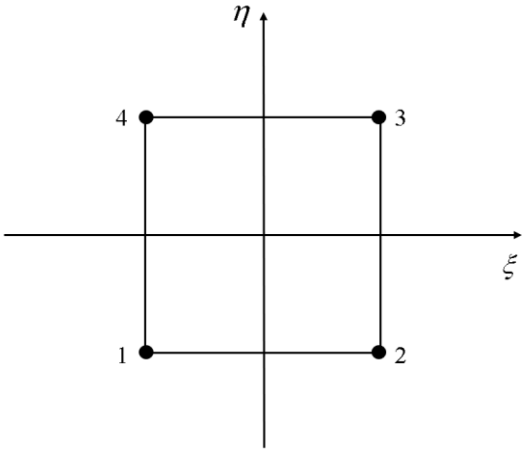


图 6.5 四节点单元

首先构造 4 节点平面四边形单元的形函数。根据形函数的特性，N_i 在节点 i 等于 1，在其他节点等于 0。那么 N_1 的表达式包含图 6.5 中 2-3 和 3-4 两条直线的方程，得：

$$N_1 = A(\xi - 1)(\eta - 1) \tag{6.20}$$

又因为 N_i 在节点 *1* 等于 1，所以 $N_1(-1,-1) = A(-1-1)(-1-1) = 1$，得 A = 1/4。采用相同的方法可以直接写出其他节点的形函数。

从上面的例子可以看出划线法的流程，划出直线过除自己点以外的所有点。将所有直线方程相乘作为初始的形函数。代入自己所在点的坐标，调整系数使得形函数在该点标上取值为 1。需要注意的是直线的交点必须是单元节点，否则会引入新的等于零的点，为不合理划线。

构造 8 节点平面四边形单元的形函数。见图 6.3，首先构造 N_1。该函数需要在除节点 1 以外的节点取值为零。所以先划 2-3、3-4 两条线，但是还差节点 5 和节点 8。有的读者会选择划 5-7 和 6-8 两条线来覆盖节点 5、8。但是直线 5-7 和直线 6-8 的交点（单元中心点）不是单元节点。因此该划线方案不可行。可以引入斜线 5-8 构造 N_1。这样 N_1 包含 2-3、3-4 和 5-8 三条线，其初始方程为：

$$N_1 = A(\xi - 1)(\eta - 1)(\xi + \eta + 1) \tag{6.21}$$

代入节点 1 的坐标得：$N_1 = A(-1-1)(-1-1)(-1-1+1) = 1$，得 $A = 1/4$。

构造 N_5 时，划线 1-4、3-4、2-3。此时 $N_5 = A(\xi + 1)(\xi - 1)(\eta - 1)$，代入节点 5 的坐标得 $N_5 = A(0+1)(0-1)(-1-1) = 1$，得 $A = 1/2$。其他节点形函数的构造方法类似。

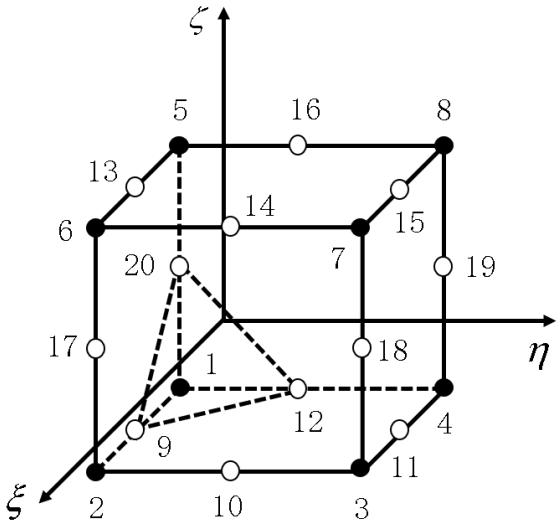


图 6.6 20 节点正六面体单元

对于 20 节点正六面体单元，也可以通过划面法构造形函数。首先构造 N_1，该函数需要在除节点 1 以外的节点取值为零。构造四个面，2-3-7-6、3-4-8-7、5-6-7-8、1-12-20。四个面的方程分别是 $\xi-1=0$、$\eta-1=0$、$\zeta-1=0$、$\xi+\eta+\zeta+2=0$。N_1的初始表达式为：

$$N_1 = A(\xi-1)(\eta-1)(\zeta-1)(\xi+\eta+\zeta+2) \tag{6.22}$$

代入节点 1 的坐标得：$N_1 = A(-1-1)(-1-1)(-1-1)(-1-1-1+2)=1$，得 $A=1/8$。

6.6 六节点三角形单元

三角形单元天然具有很好的几何适应性，如果增加三角形单元位移模式多项式的阶数，就能成为高阶高精度的单元。考虑图 6.7 所示 6 节点三角形单元，单元每边中点设一个节点，则单元有 12 个自由度，因此位移模式恰好取完全二次多项式：

$$\begin{aligned} u &= \beta_1 + \beta_2 x + \beta_3 y + \beta_4 xy + \beta_5 x^2 + \beta_6 y^2 \\ v &= \beta_7 + \beta_8 x + \beta_9 y + \beta_{10} xy + \beta_{11} x^2 + \beta_{12} y^2 \end{aligned} \tag{6.23}$$

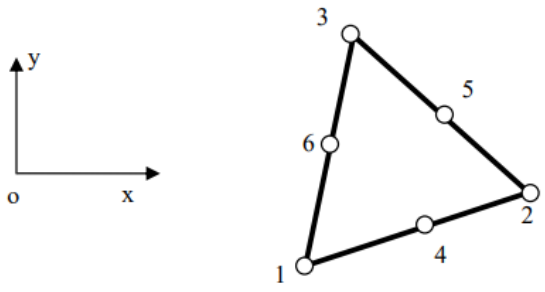


图 6.7 六节点三角形单元

显然单元满足完备性要求。由于该位移模式决定了单元边界上位移呈二次抛物线分布，相邻单元公共边界上有三个公共节点，正好能够保证相邻单元在边界上位移的连续性，因而是协调元，单元满足收敛条件。该单元应变、应力随坐标完全呈线性变化，属于高精度单元。进行广义坐标代换后位移模式仍可写成标准形式：

$$u=\sum_{i=1}^{6}N_i u_i\ ,\ \ v=\sum_{i=1}^{6}N_i v_i \tag{6.24}$$

但是，采取如前面 3 节点单元建立形函数的办法过于复杂，下面介绍用三角形单元的面积坐标描述单元位移模式和形函数的方法。单元内任一点的面积坐标满足关系:

$$L_i+L_j+L_m=1 \tag{6.25}$$

即 3 个面积坐标只有 2 个面积坐标是独立的。面积坐标与直角坐标之间有确定的变换关系，因此，对三角形单元的描述完全可以用面积坐标进行。不难导出下列变换关系：

$$L_i=\frac{1}{2A}\left(a_i+b_i x+c_i y\right)\qquad (i,j,m) \tag{6.26}$$

显然，面积坐标与 3 节点三角形单元的形函数完全相同。矩阵形式：

$$\begin{bmatrix}L_i\\L_j\\L_m\end{bmatrix}=\frac{1}{2A}\begin{bmatrix}a_i & b_i & c_i\\a_j & b_j & c_j\\a_m & b_m & c_m\end{bmatrix}\begin{bmatrix}1\\x\\y\end{bmatrix} \tag{6.27}$$

不难导出下列变换关系：

$$x=x_i L_i+x_j L_j+x_m L_m\ ,\ \ y=y_i L_i+y_j L_j+y_m L_m \tag{6.28}$$

矩阵形式：

$$\begin{bmatrix}1\\x\\y\end{bmatrix}=\begin{bmatrix}1 & 1 & 1\\x_i & x_j & x_m\\y_i & y_j & y_m\end{bmatrix}\begin{bmatrix}L_i\\L_j\\L_m\end{bmatrix} \tag{6.28}$$

利用上面变换式，三角形单元上的任何多项式函数可以方便地在两种坐标之间转换。多面积坐标的各种形式幂函数在三角形上的积分有很简便的计算公式。根据形函数性质直接构造出用面积坐标表示的形函数如下：

$$N_i=\left(2L_i-1\right)L_i\qquad \left(i=1,2,3\right) \tag{6.29}$$

即 $N_4=4L_1L_2$ ， $N_5=4L_2L_3$ ， $N_6=4L_3L_1$

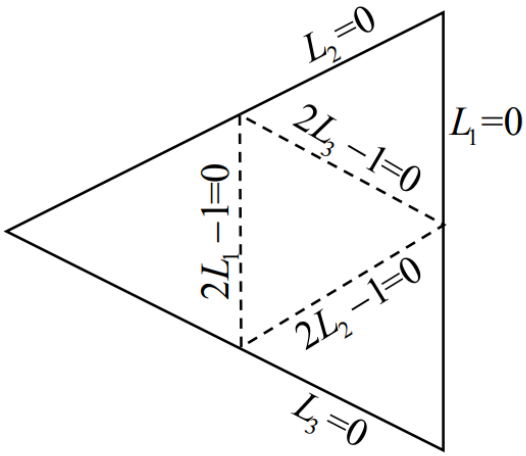


图 6.8 六节点三角形单元的面积坐标

不难验证，上述 6 个形函数满足形函数的 2 个主要性质：

$$N_i\left(P_j\right)=\delta_{ij}\ ,\ \ \sum N_i=1\ ,\ \ N_i=\left(2L_i-1\right)L_i \qquad \left(i=1,2,3\right) \tag{6.30}$$

采用面积坐标后，单元刚度矩阵和等效节点力的计算都比较方便。6 节点三角形单元列式推导原理与其他单元相同。

6.7 三维等参变换

为了将局部（自然）坐标中几何形状规则的单元转换成总体（笛卡尔）坐标中几何形状扭曲的单元，以满足对一般形状求解域进行离散化的需要，必须建立一个坐标变换，即

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}=f\left(\begin{bmatrix} \xi \\ \eta \\ \zeta \end{bmatrix}\right)\text{或}f\left(\begin{bmatrix} L_1 \\ L_2 \\ L_3 \\ L_4 \end{bmatrix}\right) \tag{6.31}$$

图 6.1 和图 6.2 所示就是这种变换的某些例子。

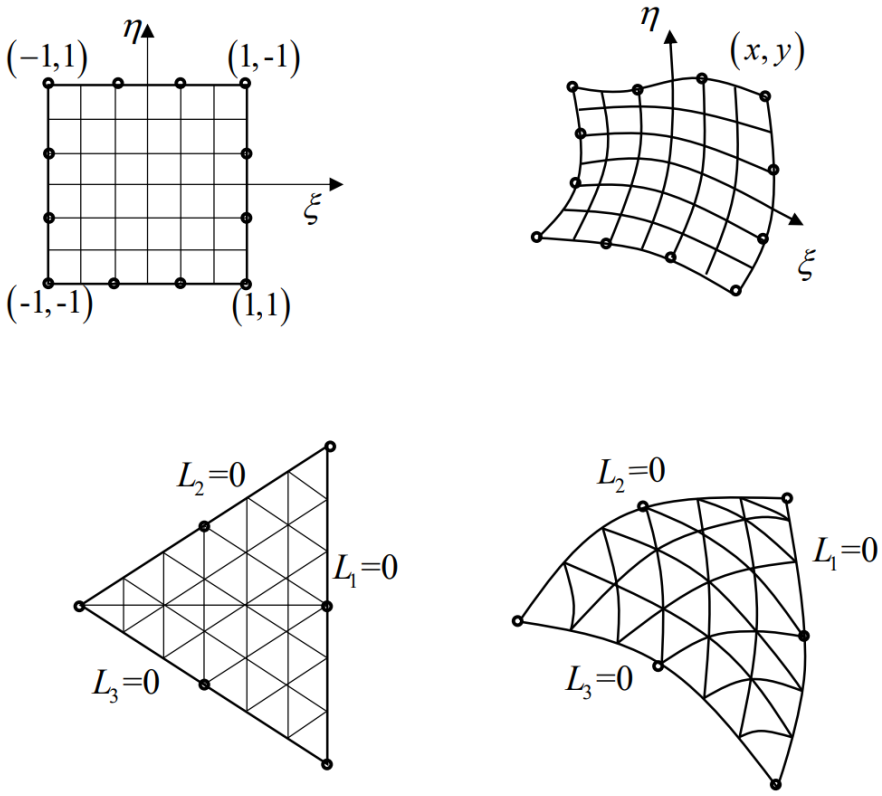


图 6.1 某些二维单元的转变

为建立前面所述的变换，最方便的方法是将(6.2.1)式也表示成插值函数的形式，即

$$x=\sum_{i=1}^{m}N_i^{'}x_i \qquad y=\sum_{i=1}^{m}N_i^{'}y_i \qquad z=\sum_{i=1}^{m}N_i^{'}z_i \tag{6.32}$$

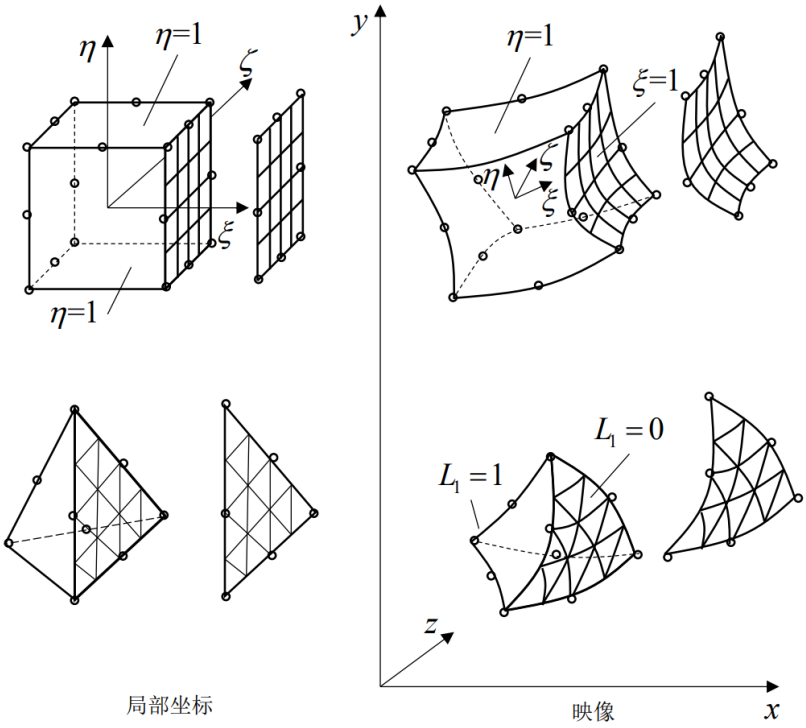


图 6.2 某些三维单元的变换

其中 m 是用以进行坐标变换的单元节点数，x_i，y_i，z_i是这些节点在总体（笛卡尔）坐标内的坐标值，N_i称为形状函数，实际上它也是用局部（自然）坐标表示的插值函数。

通过上式所建立起的两个坐标系之间的变换，从而使得自然坐标内的形状规则的单元变换成总体笛卡尔坐标内的形状扭曲的单元。今后称前者为母单元，后者为子单元。

还可以看到坐标变换关系式(6.2.2)和函数的插值表示式：$\phi=\sum_{i=1}^{n}N_i x_i$，则称这种变换为等参变换。如果坐标变换结点数多于函数插值的节点数，即 m>n 则称为超参变换。反之，m<n，则称为次（亚）参变换。

在有限元分析中，为建立求解方程，需要进行各个单元体积内和面积内的积分，它们的一般形式可表示为

$$\int_{V_e} G dV=\iiint_{V_e} G(x,y,z)dxdydz \tag{6.33}$$

$$\int_{S_e} g dS=\iint_{S_e} g(x,y,z)dS \tag{6.34}$$

而 G 和 g 中还常包含着场函数对于总体坐标 x,y,z 的导数。由于在目前的情况下，场函数是用自然坐标来表述的，又因为在自然坐标内的积分限是规格化的，所以希望能在自然坐标内按规格化

的数值积分方法开展上述积分。因此需要建立两个坐标系内导数、体积微元、面积微元之间的变换关系。

6.8 导数之间的变换

按照通常的偏微分规则，函数 N_i 对 ξ 的偏导数可表示成

$$\frac{\partial N_i}{\partial \xi}=\frac{\partial N_i}{\partial x}\frac{\partial x}{\partial \xi}+\frac{\partial N_i}{\partial y}\frac{\partial y}{\partial \xi}+\frac{\partial N_i}{\partial z}\frac{\partial z}{\partial \xi} \tag{6.35}$$

对于其他两个坐标 (η,ζ)，可得到类似的表达式。将它们集合成矩阵形式，则有：

$$\begin{bmatrix}\frac{\partial N_i}{\partial \xi}\\ \frac{\partial N_i}{\partial \eta}\\ \frac{\partial N_i}{\partial \zeta}\end{bmatrix}=\begin{bmatrix}\frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi}\\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta}\\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta}\end{bmatrix}\begin{bmatrix}\frac{\partial N_i}{\partial x}\\ \frac{\partial N_i}{\partial y}\\ \frac{\partial N_i}{\partial z}\end{bmatrix}=J\begin{bmatrix}\frac{\partial N_i}{\partial x}\\ \frac{\partial N_i}{\partial y}\\ \frac{\partial N_i}{\partial z}\end{bmatrix} \tag{6.36}$$

式中 J 为雅可比矩阵，可记作 $\partial(x,y,z)/\partial(\xi,\eta,\zeta)$。利用（6.2.2）式，$J$ 可以显式地表示为自然坐标的函数，即：

$$\begin{aligned}J\equiv\frac{\partial(x,y,z)}{\partial(\xi,\eta,\zeta)}&=\begin{bmatrix}\sum_{i=1}^{m}\frac{\partial N_i'}{\partial \xi}x_i & \sum_{i=1}^{m}\frac{\partial N_i'}{\partial \xi}y_i & \sum_{i=1}^{m}\frac{\partial N_i'}{\partial \xi}z_i\\ \sum_{i=1}^{m}\frac{\partial N_i'}{\partial \eta}x_i & \sum_{i=1}^{m}\frac{\partial N_i'}{\partial \eta}y_i & \sum_{i=1}^{m}\frac{\partial N_i'}{\partial \eta}z_i\\ \sum_{i=1}^{m}\frac{\partial N_i'}{\partial \zeta}x_i & \sum_{i=1}^{m}\frac{\partial N_i'}{\partial \zeta}y_i & \sum_{i=1}^{m}\frac{\partial N_i'}{\partial \zeta}z_i\end{bmatrix}\\ &=\begin{bmatrix}\frac{\partial N_1'}{\partial \xi} & \frac{\partial N_2'}{\partial \xi} & \cdots & \frac{\partial N_m'}{\partial \xi}\\ \frac{\partial N_1'}{\partial \eta} & \frac{\partial N_2'}{\partial \eta} & \cdots & \frac{\partial N_m'}{\partial \eta}\\ \frac{\partial N_1'}{\partial \zeta} & \frac{\partial N_2'}{\partial \zeta} & \cdots & \frac{\partial N_m'}{\partial \zeta}\end{bmatrix}\begin{bmatrix}x_1 & y_1 & z_1\\ x_2 & y_2 & z_2\\ \vdots & \vdots & \vdots\\ x_m & y_m & z_m\end{bmatrix}\end{aligned} \tag{6.37}$$

这样一来，N_i 对于 x, y, z 的偏导数可用自然坐标显式地表示为：

$$\begin{bmatrix} \dfrac{\partial N_i}{\partial x} \\ \dfrac{\partial N_i}{\partial y} \\ \dfrac{\partial N_i}{\partial z} \end{bmatrix} = J^{-1} \begin{bmatrix} \dfrac{\partial N_i}{\partial \xi} \\ \dfrac{\partial N_i}{\partial \eta} \\ \dfrac{\partial N_i}{\partial \zeta} \end{bmatrix} \tag{6.38}$$

其中 J^{-1} 是 J 的逆矩阵，它可按下式计算得到

$$J^{-1} = \frac{1}{|J|} J^{*} \tag{6.39}$$

$|J|$ 是 J 的行列式，称为雅可比行列式。J^{-1} 是 J 的伴随矩阵，它的元素 J_{ij}^{*} 是 J 的元素 J_{ji} 的代数余子式。

6.9 体积微元、面积微元的变换

从图 6.2 可以看到 $d\xi$ 、$d\eta$ 、$d\zeta$ 在笛卡尔坐标系内形成的体积微元是：

$$dV = d\xi \cdot (d\eta \times d\zeta) \tag{6.40}$$

其中，

$$\begin{aligned} d\xi &= \frac{\partial x}{\partial \xi} d\xi i + \frac{\partial y}{\partial \xi} d\xi j + \frac{\partial z}{\partial \xi} d\xi k \\ d\eta &= \frac{\partial x}{\partial \eta} d\eta i + \frac{\partial y}{\partial \eta} d\eta j + \frac{\partial z}{\partial \eta} d\eta k \\ d\zeta &= \frac{\partial x}{\partial \zeta} d\zeta i + \frac{\partial y}{\partial \zeta} d\zeta j + \frac{\partial z}{\partial \zeta} d\zeta k \end{aligned} \tag{6.41}$$

式中 i, j和k 是笛卡尔坐标 x, y和z 方向的单位向量。将(6.2.10)式代入(6.2.9)式，得到：

$$dV = \begin{vmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} & \dfrac{\partial z}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} & \dfrac{\partial z}{\partial \eta} \\ \dfrac{\partial x}{\partial \zeta} & \dfrac{\partial y}{\partial \zeta} & \dfrac{\partial z}{\partial \zeta} \end{vmatrix} d\xi d\eta d\zeta = |J| d\xi d\eta d\zeta \tag{6.42}$$

关于面积微元，例如在$\xi=$常数(c)的面上有：

$$dA=\left|d\eta\times d\zeta\right|_{\xi=1}$$

$$=\left[\left(\frac{\partial y}{\partial\eta}\frac{\partial z}{\partial\zeta}-\frac{\partial y}{\partial\zeta}\frac{\partial z}{\partial\eta}\right)^2+\left(\frac{\partial z}{\partial\eta}\frac{\partial x}{\partial\zeta}-\frac{\partial z}{\partial\zeta}\frac{\partial x}{\partial\eta}\right)^2+\left(\frac{\partial x}{\partial\eta}\frac{\partial y}{\partial\zeta}-\frac{\partial z}{\partial\zeta}\frac{\partial y}{\partial\eta}\right)^2\right]^{1/2}d\eta d\zeta$$

$$=Ad\eta d\zeta \tag{6.43}$$

其他面上的dA可以通过轮换ξ,η,ζ得到。

在有了以上几种坐标变换关系式以后，积分(6.2.3)和(6.2.4)式最终可以变换到自然坐标系的规则化区域内进行，它们可分别表示为：

$$\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1}G^*(\xi,\eta,\zeta)d\xi d\eta d\zeta \tag{6.44}$$

$$\int_{-1}^{1}\int_{-1}^{1}g^*(c,\eta,\zeta)d\eta d\zeta \text{ 等} \tag{6.45}$$

$$\left(\xi=\pm1\text{的面上，}\ \mathrm{c}=\pm1\right)$$

其中，

$$G^*(\xi,\eta,\zeta)=G(x(\xi,\eta,\zeta),y(\xi,\eta,\zeta),z(\xi,\eta,\zeta))\,|J|$$

$$g^*(c,\eta,\zeta)=g(x(c,\eta,\zeta),y(c,\eta,\zeta),z(c,\eta,\zeta))A \tag{6.46}$$

对于二维情况，以上各式将相应蜕化，这时雅可比矩阵是：

$$J=\frac{\partial(x,y,)}{\partial(\xi,\eta,)}=\begin{bmatrix}\sum_{i=1}^{m}\frac{\partial N_i^{'}}{\partial\xi}x_i & \sum_{i=1}^{m}\frac{\partial N_i^{'}}{\partial\xi}y_i\\ \sum_{i=1}^{m}\frac{\partial N_i^{'}}{\partial\eta}x_i & \sum_{i=1}^{m}\frac{\partial N_i^{'}}{\partial\eta}y_i\end{bmatrix}$$

$$=\begin{bmatrix}\frac{\partial N_1^{'}}{\partial\xi} & \frac{\partial N_2^{'}}{\partial\xi} & \cdots & \frac{\partial N_m^{'}}{\partial\xi}\\ \frac{\partial N_1^{'}}{\partial\eta} & \frac{\partial N_2^{'}}{\partial\eta} & \cdots & \frac{\partial N_m^{'}}{\partial\eta}\end{bmatrix}\begin{bmatrix}x_1 & y_1\\ x_2 & y_2\\ \vdots & \vdots\\ x_m & y_m\end{bmatrix} \tag{6.47}$$

两个坐标之间的偏导数关系是：

$$\begin{bmatrix}\frac{\partial N_i}{\partial x}\\ \frac{\partial N_i}{\partial y}\end{bmatrix}=J^{-1}\begin{bmatrix}\frac{\partial N_i}{\partial\xi}\\ \frac{\partial N_i}{\partial\eta}\end{bmatrix} \tag{6.48}$$

$d\xi$ 和 $d\eta$ 在笛卡尔坐标内形成的面积微元是：

$$dA = |J| d\xi d\eta \tag{6.49}$$

在 $\xi = c$ 的曲线上，$d\eta$ 在笛卡尔坐标内的线段微元的长度是：

$$ds = \left[\left(\frac{\partial x}{\partial \eta} \right)^2 + \left(\frac{\partial y}{\partial \eta} \right)^2 \right]^{1/2} d\eta = s d\eta \tag{6.50}$$

6.10 自然坐标为面积（或体积）坐标时的变换公式

以上关于 J, dV, dA, ds 等的公式原则上对于任何坐标和笛卡尔坐标之间的变换都是适用的。但是当自然坐标是面积或体积坐标时要注意两点：

（1）面积或体积坐标都不是完全独立的，分别存在关系式 $L_1 + L_2 + L_3 = 1$ 和 $L_1 + L_2 + L_3 + L_4 = 1$。因此可以重新定义新的自然坐标，例如对于三维情况，可令 L_1, L_2, L_3 为相当于 ξ, η, ζ 的独立变量，即令

$$L_1 = \xi \qquad L_2 = \eta \qquad L_3 = \zeta \tag{6.51}$$

并有：

$$L_4 = 1 - L_1 - L_2 - L_3 = 1 - \xi - \eta - \zeta \tag{6.52}$$

这样一来，(6.2.5)~(6.2.12)式形式上都保持不变，N_i 也保持它的原来形式，只是它对 ξ, η, ζ 的导数应作如下替换，即：

$$\frac{\partial N_i}{\partial \xi} = \frac{\partial N_i}{\partial L_1}\frac{\partial L_1}{\partial \xi} + \frac{\partial N_i}{\partial L_2}\frac{\partial L_2}{\partial \xi} + \frac{\partial N_i}{\partial L_3}\frac{\partial L_3}{\partial \xi} + \frac{\partial N_i}{\partial L_4}\frac{\partial L_4}{\partial \xi} = \frac{\partial N_i}{\partial L_1} - \frac{\partial N_i}{\partial L_4}$$

$$\frac{\partial N_i}{\partial \eta} = \frac{\partial N_i}{\partial L_2} - \frac{\partial N_i}{\partial L_4}$$

$$\frac{\partial N_i}{\partial \zeta} = \frac{\partial N_i}{\partial L_3} - \frac{\partial N_i}{\partial L_4} \tag{6.53}$$

对于二维情况，则因为可令：

$$L_1 = \xi \qquad L_2 = \eta \qquad L_3 = 1 - \xi - \eta \tag{6.54}$$

所以有：

$$\frac{\partial N_i}{\partial \xi}=\frac{\partial N_i}{\partial L_1}-\frac{\partial N_i}{\partial L_3}\qquad \frac{\partial N_i}{\partial \eta}=\frac{\partial N_i}{\partial L_2}-\frac{\partial N_i}{\partial L_{3.}} \tag{6.55}$$

（2）(6.2.13)(6.2.14)等式的积分限应根据体积坐标和面积坐标的特点，作必要的改变，这样一来，上述各式将成为：

$$\int_0^1\int_0^{1-I_3}\int_0^{1-L_2-L_3}G^*(\xi,\eta,\zeta)dL_1dL_2dL_3 \tag{6.56}$$

$$\int_0^1\int_0^{1-L_3}g^*\left(0,L_2,L_3\right)dL_2dL_3 \tag{6.57}$$

等式(6.2.24)适用于 $L_1=0$ 的表面。类似地可以得到用于 $L_2=0$， $L_3=0$ 和 $L_4=0$ 表面的表达式。应注意的是，由于 L_4 可以不以显式出现，对于 $L_4=0$ 面上的积分，可以表示成：

$$\int_0^1\int_0^{1-L_3}g^*\left(1-L_2-L_3,L_2,L_3\right)dL_2dL_3 \tag{6.58}$$

6.11 等参变换的条件

从微积分学知识已知，两个坐标之间一对一变换的条件是雅可比行列式 $|J|$ 不得为零，等参变换作为一种坐标变换也必须服从此条件。这点从上节各个关系式中的意义也可清楚的看出。首先从(6.2.11)和(6.2.17)式可见，如 $|J|=0$ 则表明笛卡尔坐标中体积微元（或面积微元）为零，即在自然坐标中的体积微元 $d\xi d\eta d\zeta$ （或面积微元 $d\xi d\eta$ ）对应笛卡尔坐标中的一个点，这种变换显然不是一一对应的。另外，因为 $|J|=0$， J^{-1} 将不成立，所以两个坐标之间偏导数的变换(6.2.7)和(6.2.16)式就不可能实现。

现在着重研究在有限元分析的实际中如何防止出现 $|J|=0$ 的情况。为简单起见，先讨论二维的情况，从(6.2.17)式已知 $dA=|J|d\xi d\eta$ ，另一方面笛卡尔坐标中的面积微元可直接表示为

$$\mathrm{d}A=|d\xi\times d\eta|=|d\xi||d\eta|\sin\left(d\xi,d\eta\right) \tag{6.59}$$

$|d\xi\times d\eta|$ 表示 $d\xi\times d\eta$ 的模；$|d\xi|,|d\eta|$ 表示 $d\xi,d\eta$ 的长度。

所以从上式和(6.2.17)式可得：

$$|J|=\frac{|d\xi||d\eta|\sin\left(d\xi,d\eta\right)}{d\xi d\eta} \tag{6.60}$$

从上式可见，只要以下三种情况之一成立，即：

$$|d\xi|=0 \text{ 或 } |d\eta|=0 \text{ 或 } \sin(d\xi,d\eta)=0 \tag{6.61}$$

就将出现$|J|=0$的情况，因此在笛卡尔坐标内划分单元时，要注意防止以上所列举情况的发生。图 6.3(a)所示单元是正常情况，而图 6.3(b)~(d)都属于应该防止出现的不正常情况。图 6.3(b)所示单元节点 3,4 退化为一个节点，在该点$|d\xi|=0$，图 6.3(c)所示单元节点 2,3 退化为一个节点，在该点$|d\eta|=0$，图 6.3(d)所示单元在节点 1,2,3，$\sin(d\xi,d\eta)>0$，而在节点 4，$\sin(d\xi,d\eta)<0$。因为$\sin(d\xi,d\eta)$在单元内连续变化，所以单元内肯定存在$\sin(d\xi,d\eta)=0$，即$d\xi$和$d\eta$共线的情况。这种情况是因为单元过分歪曲导致的。

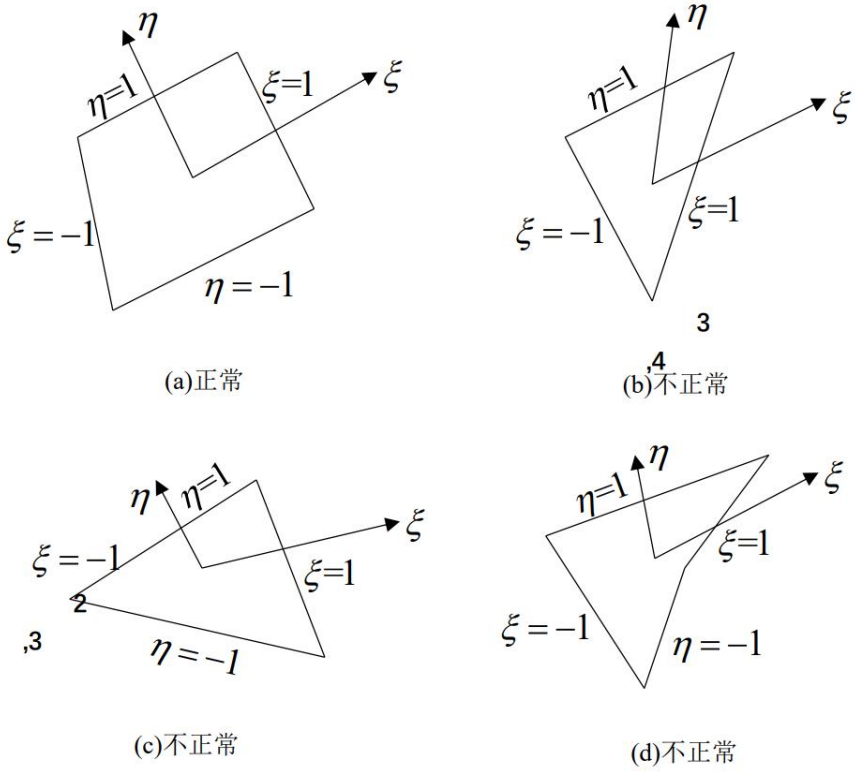


图 6.3 单元划分的正常与不正常情况

上面的讨论可以推广运用到三维的情况，即为保证变换的一一对应，应该防止因为任意的二个结点退化为一个结点而使得，$|d\eta|$，$|d\zeta|$中的任一个为零。同时还应该防止因单元过分歪曲而导致的$d\xi$，$d\eta$，$d\zeta$中的任何二个出现共线的情况。

需要指出，某些文献中建议，从统一的四边形单元的表达形式出发，利用图 6.3(b)，(c)所示 2 个结点退化为 1 个节点的方法，将四边形单元退化为三角形单元，从而不必另行推导后者的表达格式。并用类似的方法，将三维六面体单元退化为五面体单元或四面体单元。如上所述，在这些退化单元

的某些角点$|J|=0$。但是在实际分析中仍可应用，是因为数值执行中单元矩阵是利用数值积分方法计算形成的（见 6.5 节）。而$|d\xi|$数值积分点通常在单元内部。因此可以避免某些角点$|J|=0$的问题。应予指出的是，退化单元由于形态不好，而精度较差。同时，为得到一种形状的退化单元可以采用不同的退化方案。例如图 6.4 所示，(a)是 9 节点四边形单元，图 6.4(b)，(c)是采用不同退化方案得到的同样形状的 6 节点三角形单元。如果刚度矩阵采用 2×2 的高斯积分（见 6.5 节），则图(b), (c)所示退化单元中高斯积分点的位置是不同的，因此最后形成的刚度矩阵也有差别，从而影响到解的唯一性。由以上讨论可见，在一般情况下，应该尽量避免采用上述退化单元。

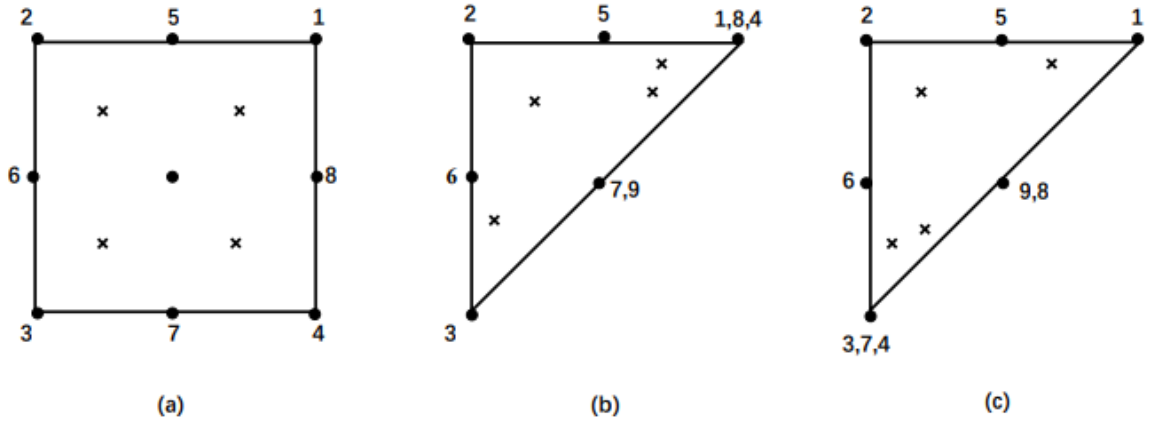


图 6.4 四边形单元退化为三角形单元（⊔节点；×高斯积分点）

6.12 等参元的收敛性

2.4 节讨论了有限元分析中解的收敛性条件，即单元必须是协调的和完备的。现在来讨论等参元是否满足此条件。

为研究单元集合体的协调性，需要考虑单元之间的公共边（或面）。为了保证协调，相 邻单元在这些公共边(或面)上应有完全相同的结点，同时每一单元沿这些边（或面）的坐标和未知函数应采用相同的插值函数加以确定。显然，只要适当划分网格和选择单元，等参元是完全能满足协调性条件的。图 6.5(a)所示正是这种情况，而图 6.5(b)所示是不满足协调性条件的。

关于单元的完备性，对于C_0型单元，要求插值函数中包含完全的线性项（即一次完全 多项式）。这样的单元可以表现函数及其一次导数为常数的情况，显然，本章讨论的所有单元在自然坐标中是满足这个要求的。现在要研究的是经过等参变换后，在笛卡尔坐标中这个要求是否仍然满足

现考查一个三维等参元，坐标和函数的插值表达式是

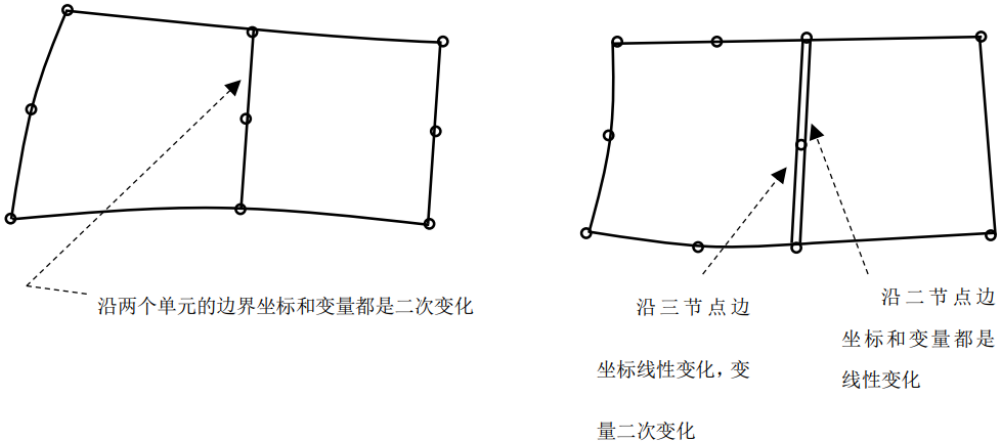


图 6.5 单元交界面上变量协调和不协调情况

$$x=\sum_{i=1}^{n}N_i x_i \qquad y=\sum_{i=1}^{n}N_i y_i \qquad z=\sum_{i=1}^{n}N_i z_i \tag{6.62}$$

$$\phi=\sum_{i=1}^{n}N_i\phi_i \tag{6.63}$$

现将线性变化场函数：

$$\phi=a+bx+cy+dz \tag{6.64}$$

在单元各个节点的数值赋予各个节点参数，即有：

$$\phi_i=a+bx_i+cy_i+dz_i \quad (i=1,2,\cdots,n) \tag{6.65}$$

将上式代入(6.3.5)式并利用(6.3.4)式，就得到单元内的函数表达式为：

$$\phi=a\sum_{i=1}^{n}N_i+bx+cy+dz \tag{6.66}$$

从上式可以看到，如果插值函数满足条件：

$$\sum_{i=1}^{n}N_i=1 \tag{6.67}$$

则(6.3.8)式和(6.3.6)式完全一致，说明在单元内确实得到了和原来给予各个节点的线性变化相应的场函数，即单元能够表示线性变化的场函数，亦即满足了完备性的要求

我们知道在构造插值函数时，条件(6.3.9)是确实满足了的。由此还可进一步看到等参元的好处，在母单元内只要满足条件(6.3.9)，则子单元可以满足更严格的完备性要求。

如果单元不是等参的，即坐标插值表示式(6. 2. 2)式中的节点数 m 和插值函数 N_i' 各自 不等于函数插值表示式 $\phi=\sum_{i=1}^{n}N_i\phi_i$ 中的节点数 n 和插值函数 N_i，这时可分为两种情况：

(1) 超参元，即 $m>n$，单元完备性要求通常是不满足的。

(2) 次参元，即 $m<n$，这时从构造变节点单元插值函数的一般方法可以推知存在下列关系式，即

$$N_i'=\sum_{n}^{i=1}C_{ij}N_i \qquad x_j=\sum_{m}^{i=1}C_{ij}x_i' \tag{6.68}$$

其中 C_{ij} 是常系数。利用上式和(6.3.7)、(6.2.2)式可以得到单元内的函数表达式为

$$\phi=a\sum_{i=1}^{n}N_i+b\sum_{i=1}^{m}N_i'x_i+c\sum_{i=1}^{m}N_i'y_i+d\sum_{i=1}^{m}N_i'z_i$$

$$=a\sum_{i=1}^{n}N_i+bx+cy+dz \tag{6.69}$$

这样就得到和(6.3.8)式同样的结果，也就是说只要 $\sum_{i=1}^{n}N_i=1$ 条件得到满足，则次参元满足完备性要求，而 $\sum_{i=1}^{n}N_i=1$ 在构造插值函数时已得到保证。

6.13 等参元用于分析弹性力学问题的一般格式

等参元通常也以位移作为基本未知量，因此在 2. 3 节中用最小位能原理变分得到的有限元一般格式对等参元同样适用。差别在于等参元的插值函数是用自然坐标给出的，等参元的一切计算都是在自然坐标系中形状规则的母单元内进行。因此只要用 6.2.2 节中有关的转换公式对 2. 3 节中的一般格式作一定的修正即可得到等参元的一般格式。

系统方程仍是 $Ka=P$,其中 $K=\sum_{e}G^TK^eG;P=\sum_{e}G^TP^e$；只需要作两方面的修改就可以计算单元矩阵:积分变量（取自然坐标）及积分限。下面举一个三维单元的例子，讨论单元矩阵计算公 式，采用两种不同的自然坐标系。

（1）**母单元为 ξ,η,ζ 坐标系中的立方体单元系列**。可以是 8 节点的一次单元，20 节点的二次单元等。自然坐标有：

$$-1\le\xi\le1 \qquad -1\le\eta\le1 \qquad -1\le\zeta\le1$$

单元矩阵计算时，可将(2.3.7)及(2.3.9)式中的被积函数 N,B 等表示成自然坐标的函数；同时 dV 和 dS 用(6. 2. 11)及(6.2. 12)式代入并确定积分的上、下限即可得到：

$$K^{\gamma}=\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1}B^{T}DB\,|J|\,d\xi d\eta d\zeta \tag{6.70}$$

$$P_{j}^{e}=\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1}N^{T}f\,|J|\,d\xi d\eta d\zeta \tag{6.71}$$

$$P_{s}^{e}=\int_{-1}^{1}\int_{-1}^{1}N^{T}TAd\eta d\xi \tag{6.72}$$

$$P_{\sigma_0}^{e}=-\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1}B^{T}\sigma_{0}\left|J\right|d\xi d\eta d\zeta \tag{6.73}$$

$$P_{\varepsilon_0}^{e}=-\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1}B^{T}D\varepsilon_{0}\left|J\right|d\xi d\eta d\zeta \tag{6.74}$$

其中，

$$\left|J\right|=\begin{vmatrix}\dfrac{\partial x}{\partial\xi} & \dfrac{\partial y}{\partial\xi} & \dfrac{\partial z}{\partial\xi}\\ \dfrac{\partial x}{\partial\eta} & \dfrac{\partial y}{\partial\eta} & \dfrac{\partial z}{\partial\eta}\\ \dfrac{\partial x}{\partial\zeta} & \dfrac{\partial y}{\partial\zeta} & \dfrac{\partial z}{\partial\zeta}\end{vmatrix} \tag{6.75}$$

$$A=\left[\left(\frac{\partial y}{\partial\eta}\frac{\partial z}{\partial\zeta}-\frac{\partial y}{\partial\zeta}\frac{\partial z}{\partial\eta}\right)^{2}+\left(\frac{\partial z}{\partial\eta}\frac{\partial x}{\partial\zeta}-\frac{\partial z}{\partial\zeta}\frac{\partial x}{\partial\eta}\right)^{2}+\left(\frac{\partial x}{\partial\eta}\frac{\partial y}{\partial\zeta}-\frac{\partial z}{\partial\zeta}\frac{\partial y}{\partial\eta}\right)^{2}\right]^{1/2} \tag{6.76}$$

在求作用于 $\eta=1$ 或 $\zeta=1$ 面上的面载荷引起的等效节点载荷时，只需将(6.4.3)式中的积分变量作相应变化，并将(6.4.7)式的 A 作坐标轮换即可。

（2）母单元为四面锥的单元系列。如一次 4 节点单元、二次 10 节点单元等。自然坐标取体积坐标 L_1,L_2,L_3,L_4, 因为它们不完全独立。如令 L_1,L_2,L_3 为相当于 ξ,η,ζ 的独立变量，则有：

$$L_4=1-L_1-L_2-L_3 \tag{6.77}$$

对于(6.4.1)~ (6.4.5)式可以改写成如下形式：

$$K^{\gamma}=\int_{0}^{1}\int_{0}^{1-L_3}\int_{0}^{1-L_2-L_3}B^{T}DB\,|J|\,dL_1dL_2dL_3 \tag{6.78}$$

$$P_{j}^{e}=\int_{0}^{1}\int_{0}^{1-L_3}\int_{0}^{1-L_2-L_3}N^{T}f\,|J|\,dL_1dL_2dL_3 \tag{6.79}$$

$$P_{s}^{e}=\int_{0}^{1}\int_{0}^{1-L_3}N^{T}TAdL_2dL_3\quad\left(T\text{作用在}L_1\text{=0的面}\right) \tag{6.80}$$

在上述计算中,计算应变矩阵B需要用到雅可比矩阵的逆矩阵，将插值函数对总体坐标的求导转化为对自然坐标求导。

对于二维问题只要将以上两组公式退化即可以得到母单元为正方形系列以及三角形系列的二维等参元的相应公式。

对于以上各积分式表示的单元矩阵和向量，只有对于少数规则形状的单元，积分可以解析地积出。对于三维单元、矩阵和向量可以解析积分的是棱边为直线的四面体单元、平行六面体单元以及上下底面为全等且平行的三角形组成的五面体单元。对于二维单元，可解析积分的是周边为直线的三角形单元和平行四边形单元。因为这些在棱边或周边上无边内节点或有等距分布边内节点的情况下，雅可比矩阵是常数矩阵，当然相应的雅可比行列式$|J|$和A等尺度转换参数也是常数。这里给出面（体）积坐标的幂函数的常用积分公式。

1. 在棱（周）边（例如ij边）上的积分公式：

$$\int_l L_i^a L_j^b dl = \frac{a!b!}{(a+b+1)!} l \tag{6.81}$$

其中l为边界长度，空间两点(x_i, y_i, z_i)，(x_j, y_j, z_j)之间的直线长度为：

$$l = \left[(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2 \right]^{1/2} \tag{6.82}$$

2. 在三角形（例如ijk）全面积上的积分公式

$$\int_A L_i^a L_j^b L_k^c dA = \frac{a!b!c!}{(a+b+c+2)!} 2A \tag{6.83}$$

其中A为三角形面积，边长为r, s, t的三角形面积为：

$$A = \frac{1}{4}\left[(r+s+t)(s+t-r)(t+r-s)(r+s-t) \right]^{1/2} \tag{6.84}$$

3. 在四面体（$ijkm$）全体积上的积分公式

$$\int_V L_i^a L_j^b L_k^c L_m^d dV = \frac{a!b!c!d!}{(a+b+c+d+3)!} 6V \tag{6.85}$$

其中V为四面体的体积，如4顶点为i, j, k, m（$i \to j \to k$右螺旋指向m）的四面体体积为

$$V=\frac{1}{6}\begin{vmatrix}1 & x_i & y_i & z_i\\ 1 & x_j & y_j & z_j\\ 1 & x_k & y_k & z_k\\ 1 & x_m & y_m & z_m\end{vmatrix} \tag{6.86}$$

利用上述积分公式很容易求有关的积分，例如

$$\iint_A L_i dxdy=\frac{1!0!0!}{(1+0+0+2)!}2A=\frac{A}{3}\qquad (i,j,m) \tag{6.87}$$

$$\iint_A L_i^2 dxdy=\frac{2!0!0!}{(2+0+0+2)!}2A=\frac{A}{6}\qquad (i,j,m) \tag{6.88}$$

$$\iint_A L_i L_j dxdy=\frac{1!1!0!}{(1+1+0+2)!}2A=\frac{A}{12}\quad (i,j,m) \tag{6.89}$$

这些积分运算是十分简便的。有了这些公式后，进行例如 2. 2. 4 节和 2. 5. 3 节中等效结点载荷等的计算将毫无困难。

由于子单元形状复杂多变，因此在通常情况下，J 及$|J|$都比较复杂，在单元矩阵的计算中，尽管采用了自然坐标后，积分限规格化了，但是除了上述少数较简单的情况外，通常都不能进行显式积分，面需求助于数值积分。计算单元特性矩阵一般采用高斯数值积分，例如单元刚度矩阵，它的数值积分形式可以表示为：

$$K^{\tau}\approx K^{\tau}=\sum_{n_g}^{i=1}H_i B_i^T D B_i |J_i| \tag{6.90}$$

其中 H_i 是权系数，n_g 是高斯积分点的点数，$B_i,|J_i|$ 等是 B，$|J|$ 等在高斯积分点 (ξ_i,η_i,ζ_i) 的取值。

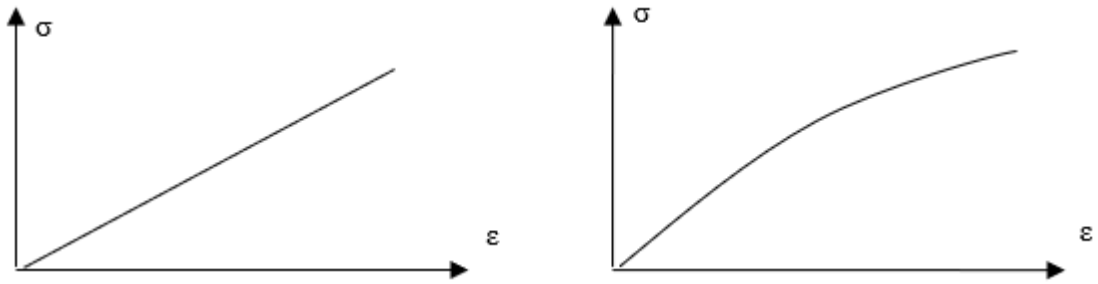
第七章 材料损伤非线性有限元

7.1 材料的非线性力学行为和增量法

弹性力学基本方程有三类：平衡方程、几何方程和物理方程，这三类方程需加上边界条件才能进行求解。而有限元法本质上是求解弹性力学基本方程和边界条件的近似方法，相较于弹性力学的三类基本方程，其除了平衡方程外，对应的非线性问题有三类：材料非线性、几何非线性和边界条件非线性。本章将介绍材料非线性问题的有限元计算方法。

材料非线性问题是最为常见的一类非线性问题，主要研究的问题包括如何模拟材料的非线性应力应变关系，以及如何计算具有非线性本构特性的结构变形问题。目前获得非线性本构关系的方法有实验方法和微观力学方法。二者相辅相成，实验方法能够获得材料本构响应的客观描述，但是难以测定极端环境下（例如极高温度、极高压力）和复杂载荷条件下（例如热力电磁耦合环境）的力学行为。通过微观力学来描述材料的力学行为越来越成为力学以及材料学研究领域的重要方法，由于微观力学已经超出了本教材的范围，本文仅介绍经典损伤力学的基本概念和方法，起到一个抛砖引玉的作用。更详细和深入的介绍请参阅损伤力学以及连续介质力学的教材。

本教材的 1.4.4 节介绍了弹性本构关系，该关系表明材料的应力 σ 和应变 ε 的关系是线性关系。对于最简单一维的情况，可以用应力应变曲线来表示，如图 7.1 所示。



（a）线性应力应变关系　　（b）非线性应力应变关系

图 7.1 应力应变曲线

对于图 7.1（b）所示的非线性应力应变曲线，一般采用增量形式方程来描述：

$$\mathrm{d}\sigma = f(\varepsilon)\mathrm{d}\varepsilon \tag{7.1}$$

其中 $\mathrm{d}\sigma$ 和 $\mathrm{d}\varepsilon$ 表示应力和应变增量。如果 $f(\varepsilon)$ 是一个常数，那么方程（7.1）演变为虎克定律。增量形式的本构方程可以采用差分法进行计算，用 $\Delta\varepsilon$ 和 $\Delta\sigma$ 代替 $\mathrm{d}\varepsilon$ 和 $\mathrm{d}\sigma$，然后给定初始条件，即可计算出所有应变下的应力，计算流程如下

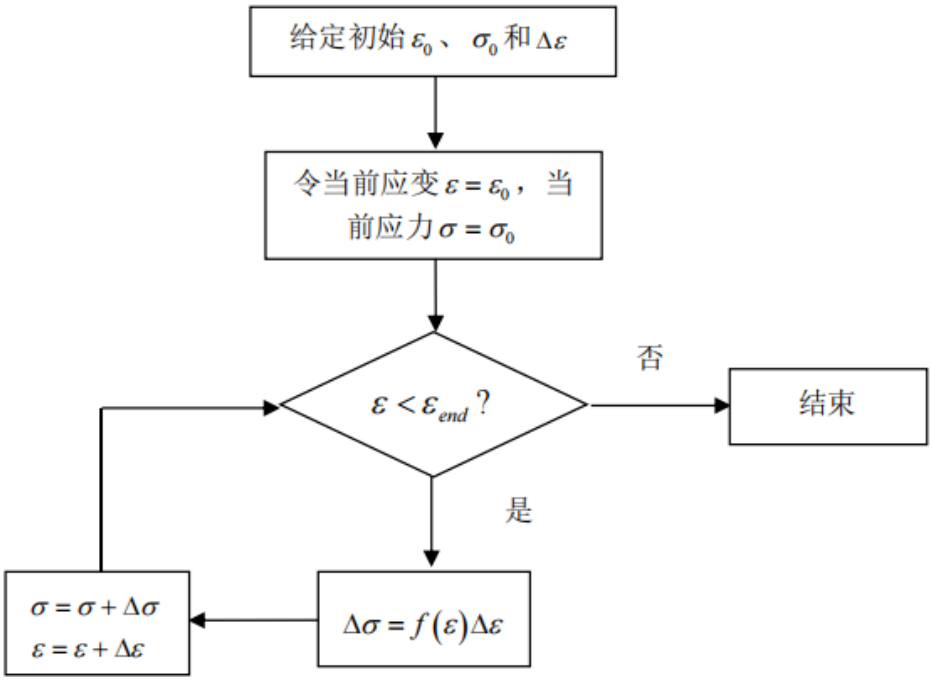


图 7.2 增量本构方程计算流程

我们以方程(7.2)为例，说明如何实现增量计算。

$$f(\varepsilon)=\begin{cases} E & \varepsilon<\varepsilon_0 \\ E\dfrac{(\varepsilon-\varepsilon_1)}{(\varepsilon_0-\varepsilon_1)}+E^t\dfrac{(\varepsilon-\varepsilon_0)}{(\varepsilon_1-\varepsilon_0)} & \varepsilon_0\le\varepsilon<\varepsilon_1 \\ E^t & \varepsilon_1\le\varepsilon \end{cases} \tag{7.2}$$

程序如下：

```
#include "stdio.h"
#include "stdlib.h"
double Func_Ep(double dep,double ep,double E,double Et,double ep0,double ep1)
{
    double ds;
    if(ep<ep0)
    {
        ds=E*dep;
    }
    else if(ep>=ep0&&ep<ep1)
    {
```

```
        ds=(E*(ep-ep1)/(ep0-ep1)+Et*(ep-ep0)/(ep1-ep0))*dep;
    }
    else
    {
        ds=Et*dep;
    }
    return ds;
}
void main()
{
    double    dep=0.0001,ep0=0.05,ep1=0.1,E=200e3,Et=20e3,ds;
    int n=3000,i;
    double *ep=NULL,*ss=NULL;
    FILE *pf=NULL;
    ///////
    ep=(double *)malloc(sizeof(double)*(n+1));
    ss=(double *)malloc(sizeof(double)*(n+1));
    ep[0]=0;
    ss[0]=0;
    for(i=0;i<n;i++)
    {
        ds=Func_Ep(dep,ep[i],E,Et,ep0,ep1);
        ss[i+1]=ss[i]+ds;
        ep[i+1]=ep[i]+dep;
    }
    pf=fopen("Result.txt","w");
    for(i=0;i<n+1;i++)
    {
        fprintf(pf,"%20e %20e\n",ep[i],ss[i]);
    }
```

```
    fclose(pf);
    return;
}
```

计算结果如图 7.3 所示。

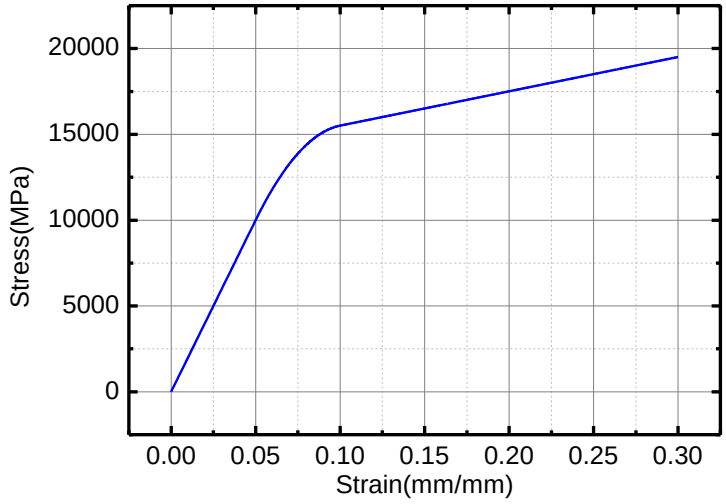


图 7.3 增量本构方程计算结果

7.2 增量基本方程

由 7.1 可知，对于材料非线性问题，本构方程一般表示成增量的形式。前文所述的弹性力学方程以及有限元方程是以位移 u 、应变 ε 和应力 σ 为基本未知量的。为了求解材料非线性问题，还需要建立以 Δu 、 $\Delta\varepsilon$ 和 $\Delta\sigma$ 为基本未知量的方程。

假设研究对象的区域为 Ω，位移边界为 Γ_1、力边界为 Γ_2。则 t 时刻在位移边界 Γ_1，位移满足位移边界条件：

$$ {}^{t}u = {}^{t}\bar{u},\quad {}^{t}v = {}^{t}\bar{v},\quad {}^{t}w = {}^{t}\bar{w} \tag{7.3} $$

同样 t 时刻在力边界 Γ_2，力满足应力边界条件：

$$ \begin{aligned} {}^{t}\sigma_x l + {}^{t}\tau_{yx} m + {}^{t}\tau_{zx} n &= {}^{t}q_x \\ {}^{t}\tau_{xy} l + {}^{t}\sigma_y m + {}^{t}\tau_{zy} n &= {}^{t}q_y \\ {}^{t}\tau_{xz} l + {}^{t}\tau_{yz} m + {}^{t}\sigma_z n &= {}^{t}q_z \end{aligned} \tag{7.4} $$

区域 Ω 内的体积力为 ${}^{t}f_x$、${}^{t}f_y$ 和 ${}^{t}f_z$，假设此刻区域 Ω 内的应力满足平衡方程：

$$\begin{aligned}
\frac{\partial {}^{t}\sigma_{xx}}{\partial x}+\frac{\partial {}^{t}\tau_{xy}}{\partial y}+\frac{\partial {}^{t}\tau_{xz}}{\partial z}+{}^{t}f_{x}=0\\
\frac{\partial {}^{t}\tau_{xy}}{\partial x}+\frac{\partial {}^{t}\sigma_{yy}}{\partial y}+\frac{\partial {}^{t}\tau_{yz}}{\partial z}+{}^{t}f_{y}=0\\
\frac{\partial {}^{t}\tau_{xz}}{\partial x}+\frac{\partial {}^{t}\tau_{yz}}{\partial y}+\frac{\partial {}^{t}\sigma_{zz}}{\partial z}+{}^{t}f_{z}=0
\end{aligned} \tag{7.5}$$

在$t+\Delta t$时刻，Γ_1上的位移边界条件有一增量，即：

$$^{t+\Delta t}\bar{u}={}^{t}\bar{u}+\Delta\bar{u},\quad {}^{t+\Delta t}\bar{v}={}^{t}\bar{v}+\Delta\bar{v},\quad {}^{t+\Delta t}\bar{w}={}^{t}\bar{w}+\Delta\bar{w} \tag{7.6}$$

Γ_2上的应力和Ω内的体积力也产生一增量：

$$\begin{aligned}
&{}^{t+\Delta t}q_x={}^{t}q_x+\Delta q_x,\ {}^{t+\Delta t}q_y={}^{t}q_y+\Delta q_y,\ {}^{t+\Delta t}q_z={}^{t}q_z+\Delta q_z\\
&{}^{t+\Delta t}f_x={}^{t}f_x+\Delta f_x,\ {}^{t+\Delta t}f_y={}^{t}f_y+\Delta f_y,\ {}^{t+\Delta t}f_z={}^{t}f_z+\Delta f_z
\end{aligned} \tag{7.7}$$

由上述载荷产生Ω内的应力、应变和位移也产生一个增量：

$$\begin{aligned}
&{}^{t+\Delta t}\sigma_{xx}={}^{t}\sigma_{xx}+\Delta\sigma_{xx},\ {}^{t+\Delta t}\sigma_{yy}={}^{t}\sigma_{yy}+\Delta\sigma_{yy},\ {}^{t+\Delta t}\sigma_{zz}={}^{t}\sigma_{zz}+\Delta\sigma_{zz}\\
&{}^{t+\Delta t}\tau_{xy}={}^{t}\tau_{xy}+\Delta\tau_{xy},\ {}^{t+\Delta t}\tau_{yz}={}^{t}\tau_{yz}+\Delta\tau_{yz},\ {}^{t+\Delta t}\tau_{xz}={}^{t}\tau_{xz}+\Delta\tau_{xz}\\
&{}^{t+\Delta t}\varepsilon_{xx}={}^{t}\varepsilon_{xx}+\Delta\varepsilon_{xx},\ {}^{t+\Delta t}\varepsilon_{yy}={}^{t}\varepsilon_{yy}+\Delta\varepsilon_{yy},\ {}^{t+\Delta t}\varepsilon_{zz}={}^{t}\varepsilon_{zz}+\Delta\varepsilon_{zz}\\
&{}^{t+\Delta t}\varepsilon_{xy}={}^{t}\varepsilon_{xy}+\Delta\varepsilon_{xy},\ {}^{t+\Delta t}\varepsilon_{yz}={}^{t}\varepsilon_{yz}+\Delta\gamma_{yz},\ {}^{t+\Delta t}\varepsilon_{xz}={}^{t}\varepsilon_{xz}+\Delta\varepsilon_{xz}\\
&{}^{t+\Delta t}u={}^{t}u+\Delta u,\ {}^{t+\Delta t}v={}^{t}v+\Delta v,\ {}^{t+\Delta t}w={}^{t}w+\Delta w
\end{aligned} \tag{7.8}$$

由等式(7.8)定义的变量仍然要满足平衡方程、几何方程、物理方程和边界条件：

$$\begin{aligned}
\frac{\partial\left({}^{t}\sigma_{xx}+\Delta\sigma_{xx}\right)}{\partial x}+\frac{\partial\left({}^{t}\tau_{xy}+\Delta\tau_{xy}\right)}{\partial y}+\frac{\partial\left({}^{t}\tau_{xz}+\Delta\tau_{xz}\right)}{\partial z}+{}^{t}f_{x}+\Delta f_{x}=0\\
\frac{\partial\left({}^{t}\tau_{xy}+\Delta\tau_{xy}\right)}{\partial x}+\frac{\partial\left({}^{t}\sigma_{yy}+\Delta\sigma_{yy}\right)}{\partial y}+\frac{\partial\left({}^{t}\tau_{yz}+\Delta\tau_{yz}\right)}{\partial z}+{}^{t}f_{y}+\Delta f_{y}=0\\
\frac{\partial\left({}^{t}\tau_{xz}+\Delta\tau_{xz}\right)}{\partial x}+\frac{\partial\left({}^{t}\tau_{yz}+\Delta\tau_{yz}\right)}{\partial y}+\frac{\partial\left({}^{t}\sigma_{zz}+\Delta\sigma_{zz}\right)}{\partial z}+{}^{t}f_{z}+\Delta f_{z}=0
\end{aligned} \tag{7.9}$$

$$\begin{aligned}
&{}^{t+\Delta t}\varepsilon_{xx}={}^{t}\varepsilon_{xx}+\Delta\varepsilon_{xx}=\frac{\partial {}^{t}u}{\partial x}+\frac{\partial\Delta u}{\partial x},\ {}^{t+\Delta t}\varepsilon_{yy}={}^{t}\varepsilon_{yy}+\Delta\varepsilon_{yy}=\frac{\partial {}^{t}v}{\partial y}+\frac{\partial\Delta v}{\partial y},\ {}^{t+\Delta t}\varepsilon_{zz}={}^{t}\varepsilon_{zz}+\Delta\varepsilon_{zz}=\frac{\partial {}^{t}w}{\partial z}+\frac{\partial\Delta w}{\partial z}\\
&{}^{t+\Delta t}\varepsilon_{xy}={}^{t}\varepsilon_{xy}+\Delta\varepsilon_{xy}=\frac{1}{2}\left(\frac{\partial {}^{t}u}{\partial y}+\frac{\partial {}^{t}v}{\partial x}\right)+\frac{1}{2}\left(\frac{\partial\Delta u}{\partial y}+\frac{\partial\Delta v}{\partial x}\right)\\
&{}^{t+\Delta t}\varepsilon_{yz}={}^{t}\varepsilon_{yz}+\Delta\varepsilon_{yz}=\frac{1}{2}\left(\frac{\partial {}^{t}v}{\partial z}+\frac{\partial {}^{t}w}{\partial y}\right)+\frac{1}{2}\left(\frac{\partial\Delta v}{\partial z}+\frac{\partial\Delta w}{\partial y}\right)\\
&{}^{t+\Delta t}\varepsilon_{xz}={}^{t}\varepsilon_{xz}+\Delta\varepsilon_{xz}=\frac{1}{2}\left(\frac{\partial {}^{t}u}{\partial z}+\frac{\partial {}^{t}w}{\partial x}\right)+\frac{1}{2}\left(\frac{\partial\Delta u}{\partial z}+\frac{\partial\Delta w}{\partial x}\right)
\end{aligned} \tag{7.10}$$

$$
\begin{aligned}
\Delta\sigma_{xx} &= {}^{t}D_{xx,xx}\Delta\varepsilon_{xx} + {}^{t}D_{xx,yy}\Delta\varepsilon_{yy} + {}^{t}D_{xx,zz}\Delta\varepsilon_{zz} + {}^{t}D_{xx,xy}\Delta\varepsilon_{xy} + {}^{t}D_{xx,yz}\Delta\varepsilon_{yz} + {}^{t}D_{xx,xz}\Delta\varepsilon_{xz}\\
\Delta\sigma_{yy} &= {}^{t}D_{yy,xx}\Delta\varepsilon_{xx} + {}^{t}D_{yy,yy}\Delta\varepsilon_{yy} + {}^{t}D_{yy,zz}\Delta\varepsilon_{zz} + {}^{t}D_{yy,xy}\Delta\varepsilon_{xy} + {}^{t}D_{yy,yz}\Delta\varepsilon_{yz} + {}^{t}D_{yy,xz}\Delta\varepsilon_{xz}\\
\Delta\sigma_{zz} &= {}^{t}D_{zz,xx}\Delta\varepsilon_{xx} + {}^{t}D_{zz,yy}\Delta\varepsilon_{yy} + {}^{t}D_{zz,zz}\Delta\varepsilon_{zz} + {}^{t}D_{zz,xy}\Delta\varepsilon_{xy} + {}^{t}D_{zz,yz}\Delta\varepsilon_{yz} + {}^{t}D_{zz,xz}\Delta\varepsilon_{xz}\\
\Delta\tau_{xy} &= {}^{t}D_{xy,xx}\Delta\varepsilon_{xx} + {}^{t}D_{xy,yy}\Delta\varepsilon_{yy} + {}^{t}D_{xy,zz}\Delta\varepsilon_{zz} + {}^{t}D_{xy,xy}\Delta\varepsilon_{xy} + {}^{t}D_{xy,yz}\Delta\varepsilon_{yz} + {}^{t}D_{xy,xz}\Delta\varepsilon_{xz}\\
\Delta\tau_{yz} &= {}^{t}D_{yz,xx}\Delta\varepsilon_{xx} + {}^{t}D_{yz,yy}\Delta\varepsilon_{yy} + {}^{t}D_{yz,zz}\Delta\varepsilon_{zz} + {}^{t}D_{yz,xy}\Delta\varepsilon_{xy} + {}^{t}D_{yz,yz}\Delta\varepsilon_{yz} + {}^{t}D_{yz,xz}\Delta\varepsilon_{xz}\\
\Delta\tau_{xz} &= {}^{t}D_{xz,xx}\Delta\varepsilon_{xx} + {}^{t}D_{xz,yy}\Delta\varepsilon_{yy} + {}^{t}D_{xz,zz}\Delta\varepsilon_{zz} + {}^{t}D_{xz,xy}\Delta\gamma_{xy} + {}^{t}D_{xz,yz}\Delta\gamma_{yz} + {}^{t}D_{xz,xz}\Delta\gamma_{xz}
\end{aligned}
\tag{7.11}
$$

边界条件

$$
{}^{t}u+\Delta u={}^{t}\bar{u}+\Delta\bar{u},\quad {}^{t}v+\Delta v={}^{t}\bar{v}+\Delta\bar{v},\quad {}^{t}w+\Delta w={}^{t}\bar{w}+\Delta\bar{w}\quad \text{在}\Gamma_1\text{上} \tag{7.12}
$$

和应力边界条件：

$$
\begin{aligned}
&{}^{t}\sigma_{xx}l+{}^{t}\tau_{yx}m+{}^{t}\tau_{zx}n+\Delta\sigma_{xx}l+\Delta\tau_{yx}m+\Delta\tau_{zx}n={}^{t}q_x+\Delta q_x\\
&{}^{t}\tau_{xy}l+{}^{t}\sigma_{yy}m+{}^{t}\tau_{zy}n+\Delta\tau_{xy}l+\Delta\sigma_{yy}m+\Delta\tau_{zy}n={}^{t}q_y+\Delta q_y\quad \text{在}\Gamma_2\text{上}\\
&{}^{t}\tau_{xz}l+{}^{t}\tau_{yz}m+{}^{t}\sigma_{zz}n+\Delta\tau_{xz}l+\Delta\tau_{yz}m+\Delta\sigma_{zz}n={}^{t}q_z+\Delta q_z
\end{aligned}
\tag{7.13}
$$

因为 t 时刻的应力、应变和位移满足平衡方程、几何方程、物理方程和边界条件，因此方程(7.9)、(7.10)、(7.12)和(7.13)缩减为：

$$
\begin{aligned}
&\frac{\partial(\Delta\sigma_{xx})}{\partial x}+\frac{\partial(\Delta\tau_{xy})}{\partial y}+\frac{\partial(\Delta\tau_{xz})}{\partial z}+\Delta f_x=0\\
&\frac{\partial(\Delta\tau_{xy})}{\partial x}+\frac{\partial(\Delta\sigma_{yy})}{\partial y}+\frac{\partial(\Delta\tau_{yz})}{\partial z}+\Delta f_y=0\\
&\frac{\partial(\Delta\tau_{xz})}{\partial x}+\frac{\partial(\Delta\tau_{yz})}{\partial y}+\frac{\partial(\Delta\sigma_{zz})}{\partial z}+\Delta f_z=0
\end{aligned}
\tag{7.14}
$$

$$
\begin{aligned}
&\Delta\varepsilon_{xx}=\frac{\partial\Delta u}{\partial x},\Delta\varepsilon_{yy}=\frac{\partial\Delta v}{\partial y},\Delta\varepsilon_{zz}=\frac{\partial\Delta w}{\partial z}\\
&\Delta\varepsilon_{xy}=\frac{1}{2}\left(\frac{\partial\Delta u}{\partial y}+\frac{\partial\Delta v}{\partial x}\right),\Delta\varepsilon_{yz}=\frac{1}{2}\left(\frac{\partial\Delta v}{\partial z}+\frac{\partial\Delta w}{\partial y}\right),\Delta\varepsilon_{xz}=\frac{1}{2}\left(\frac{\partial\Delta u}{\partial z}+\frac{\partial\Delta w}{\partial x}\right)
\end{aligned}
\tag{7.15}
$$

$$
\Delta u=\Delta\bar{u},\ \Delta v=\Delta\bar{v},\ \Delta w=\Delta\bar{w}\quad \text{在}\Gamma_1\text{上} \tag{7.16}
$$

$$
\begin{aligned}
&\Delta\sigma_{xx}l+\Delta\tau_{yx}m+\Delta\tau_{zx}n=\Delta q_x\\
&\Delta\tau_{xy}l+\Delta\sigma_{yy}m+\Delta\tau_{zy}n=\Delta q_y\quad \text{在}\Gamma_2\text{上}\\
&\Delta\tau_{xz}l+\Delta\tau_{yz}m+\Delta\sigma_{zz}n=\Delta q_z
\end{aligned}
\tag{7.17}
$$

由此可见，增量型方程只不过将原方程中的位移、应力和应变修改为增量形式。但是需要注意的是方程(7.8)是方程(7.1)的线性化的结果。因为 $\Delta\sigma$ 应通过对非线性关系积分得到，即：

$$
\Delta\sigma=\int_t^{t+\Delta t}\mathrm{d}\sigma=\int_t^{t+\Delta t}f(\varepsilon)\mathrm{d}\varepsilon \tag{7.18}
$$

式中，$f(\varepsilon)$ 是 ε 的函数。如果直接用 $\Delta\sigma=f(\varepsilon)\Delta\varepsilon$ 代替(7.18)则相当于最简单的欧拉方法。当然，也可以采用其他的数值积分方法从方程（7.18）中导出 $\Delta\sigma$ 和 $\Delta\varepsilon$ 的关系。

7.3 增量虚位移原理

首先采用增量形式的虚位移原理。如果在 $t+\Delta t$ 时刻的应力 ${}^{t}\sigma+\Delta\sigma$ 和体积载荷及边界载荷满足平衡方程，则此力系在满足几何协调条件的虚位移 $\delta(\Delta u_i)$ 上的虚功等于零，即：

$$\int_{\Omega}\left({}^{t}\sigma_{ij}+\Delta\sigma_{ij}\right)\delta\left(\Delta\varepsilon_{ij}\right)dV-\int_{\Omega}\left({}^{t}f_i+\Delta f_i\right)\delta\left(\Delta u_i\right)dV-\int_{\Gamma 2}\left({}^{t}q_i+\Delta q_i\right)\delta\left(\Delta u_i\right)ds=0 \tag{7.19}$$

上式中

$$\int_{\Omega}\left({}^{t}\sigma_{ij}+\Delta\sigma_{ij}\right)\delta\left(\Delta\varepsilon_{ij}\right)dV=\sum_{i=x,y,z}\sum_{j=x,y,z}\int_{\Omega}\left({}^{t}\sigma_{ij}+\Delta\sigma_{ij}\right)\delta\left(\Delta\varepsilon_{ij}\right)dV$$

$$\int_{\Omega}\left({}^{t}f_i+\Delta f_i\right)\delta\left(\Delta u_i\right)dV=\sum_{i=x,y,z}\int_{\Omega}\left({}^{t}f_i+\Delta f_i\right)\delta\left(\Delta u_i\right)dV$$

$$\int_{\Gamma 2}\left({}^{t}q_i+\Delta q_i\right)\delta\left(\Delta u_i\right)ds=\sum_{i=x,y,z}\int_{\Gamma 2}\left({}^{t}q_i+\Delta q_i\right)\delta\left(\Delta u_i\right)ds$$

将方程（7.11）代入上式后得：

$$\begin{aligned}&\int_{\Omega}{}^{t}D_{ijkl}\Delta\varepsilon_{kl}\delta\left(\Delta\varepsilon_{ij}\right)dV-\int_{\Omega}\Delta f_i\delta\left(\Delta u_i\right)dV-\int_{\Gamma 2}\Delta q_i\delta\left(\Delta u_i\right)ds\\&=-\int_{\Omega}{}^{t}\sigma_{ij}\delta\left(\Delta\varepsilon_{ij}\right)dV+\int_{\Omega}{}^{t}f_i\delta\left(\Delta u_i\right)dV+\int_{\Gamma 2}{}^{t}q_i\delta\left(\Delta u_i\right)ds\end{aligned} \tag{7.20}$$

如果 t 时刻的应力严格满足平衡方程，上式的右端等于零。通常由于数值计算结果不可能严格满足平衡方程，因此通常将右端项保留作为矫正项，可以理解为不平衡力势能（相差一负号）的变分。这就是增量形式的虚位移原理。

单元内的位移增量可以通过型函数表达，即：

$$\Delta\mathbf{u}=\mathbf{N}\cdot\Delta\mathbf{a}^{e} \tag{7.21}$$

其中 $\Delta\mathbf{a}^{e}$ 是单元 e 的节点位移向量。

单元内应变与节点位移的关系为：

$$\Delta\boldsymbol{\varepsilon}=\mathbf{B}\cdot\Delta\mathbf{a}^{e} \tag{7.22}$$

将上式代入方程（7.20）后得：

$$\begin{aligned}&\sum_{e}\int_{\Omega^e}\delta\left(\Delta\mathbf{a}^{e}\right)^{T}\cdot\mathbf{B}^{T}\cdot{}^{t}\mathbf{D}\cdot\mathbf{B}\cdot\left(\Delta\mathbf{a}^{e}\right)dV-\sum_{e}\int_{\Omega^e}\delta\left(\Delta\mathbf{a}^{e}\right)^{T}\cdot\mathbf{N}^{T}\cdot\Delta\mathbf{f}dV-\sum_{e}\int_{\Gamma_2^e}\delta\left(\Delta\mathbf{a}^{e}\right)^{T}\cdot\mathbf{N}^{T}\cdot\Delta\mathbf{q}ds\\&=-\sum_{e}\int_{\Omega^e}\delta\left(\Delta\mathbf{a}^{e}\right)^{T}\cdot\mathbf{B}^{T}\cdot{}^{t}\boldsymbol{\sigma}dV+\sum_{e}\int_{\Omega^e}\delta\left(\Delta\mathbf{a}^{e}\right)^{T}\cdot\mathbf{N}^{T}\cdot{}^{t}\mathbf{f}dV+\sum_{e}\int_{\Gamma_2^e}\delta\left(\Delta\mathbf{a}^{e}\right)^{T}\cdot\mathbf{N}^{T}\cdot{}^{t}\mathbf{q}ds\end{aligned} \tag{7.23}$$

我们可以通过抽取矩阵 $\mathbf{G}^{e}$ 建立 $\Delta\mathbf{a}^{e}$ 与 $\Delta\mathbf{a}$ 之间的关系：

$$\Delta\mathbf{a}^{e}=\mathbf{G}^{e}\cdot\Delta\mathbf{a} \tag{7.24}$$

将方程(7.24)代入(7.23)消去 $\Delta\mathbf{a}^{e}$ 后得：

$$\begin{aligned}&\delta(\Delta\mathbf{a})^T\cdot\left(\sum_e\int_{\Omega^e}\mathbf{G}^{eT}\cdot\mathbf{B}^T\cdot{}^t\mathbf{D}\cdot\mathbf{B}\cdot\mathbf{G}^e dV\right)\cdot\Delta\mathbf{a}-\delta(\Delta\mathbf{a})^T\left(\sum_e\int_{\Omega^e}\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot\Delta\mathbf{f}dV\right)-\delta(\Delta\mathbf{a})^T\left(\sum_e\int_{\Gamma_2^e}\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot\Delta\mathbf{q}ds\right)\\&=-\delta(\Delta\mathbf{a})^T\left(\sum_e\int_{\Omega^e}\mathbf{G}^{eT}\cdot\mathbf{B}^T\cdot{}^t\boldsymbol{\sigma}dV\right)+\delta(\Delta\mathbf{a})^T\left(\sum_e\int_{\Omega^e}\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot{}^t\mathbf{f}dV\right)+\delta(\Delta\mathbf{a})^T\sum_e\int_{\Gamma_2^e}\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot{}^t\mathbf{q}ds\end{aligned}\tag{7.25}$$

由于变分 $\delta(\Delta\mathbf{a})^T$ 的任意性，上式可化简为：

$$^t\mathbf{K}\cdot\Delta\mathbf{a}=\Delta\mathbf{Q}\tag{7.26}$$

其中 $^t\mathbf{K}$ 、$\Delta\mathbf{a}$ 和 $\Delta\mathbf{Q}$ 分别是系统的刚度矩阵，增量位移向量和不平衡力向量。它们分别由单元的各个对应量集成，即：

$$\begin{aligned}&^t\mathbf{K}=\sum_e\int_{\Omega^e}\mathbf{G}^{eT}\cdot\mathbf{B}^T\cdot{}^t\mathbf{D}\cdot\mathbf{B}\cdot\mathbf{G}^e dV\\&\Delta\mathbf{Q}=\sum_e\int_{\Omega^e}\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot\Delta\mathbf{f}dV+\sum_e\int_{\Gamma_2^e}\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot\Delta\mathbf{q}ds\\&-\sum_e\int_{\Omega^e}\mathbf{G}^{eT}\cdot\mathbf{B}^T\cdot{}^t\boldsymbol{\sigma}dV+\sum_e\int_{\Omega^e}\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot{}^t\mathbf{f}dV+\sum_e\int_{\Gamma_2^e}\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot{}^t\mathbf{q}ds\end{aligned}\tag{7.27}$$

并且

$$\begin{aligned}&^t\mathbf{K}^e=\int_{\Omega^e}\mathbf{B}^T\cdot{}^t\mathbf{D}\cdot\mathbf{B}dV\\&^{t+\Delta t}\mathbf{Q}^e=\int_{\Omega^e}\mathbf{N}^T\cdot{}^{t+\Delta t}\mathbf{f}dV+\int_{\Gamma_2^e}\mathbf{N}^T\cdot{}^{t+\Delta t}\mathbf{q}ds\\&^t\mathbf{Q}^e=\int_{\Omega^e}\mathbf{B}^T\cdot{}^t\boldsymbol{\sigma}dV\end{aligned}\tag{7.28}$$

上式中，$^{t+\Delta t}\mathbf{Q}^e$ 和 $^t\mathbf{Q}^e$ 分别代表 $t+\Delta t$ 时刻的外载荷向量和 t 时刻的内力向量，所以 $\Delta\mathbf{Q}$ 称为不平衡力向量。如果 $^t\mathbf{Q}^e$ 满足平衡的要求，则 $\Delta\mathbf{Q}^e$ 表示载荷增量向量。表示成现在的形式是为了进行平衡矫正，以避免解的漂移。

7.4 程序实现

根据等式(7.28)可知，材料非线性有限元计算过程中，对于不同的材料类型，计算过程中仅 $^t\mathbf{D}$ 不同而已，其他项都与材料本身的特性无关。因此我们可以设想出这样一个通用程序，该程序实现了不平衡力向量、合成总刚、施加位移约束、求解线性方程组等功能，而将 $^t\mathbf{D}$ 的计算作为一个开放函数发布。这样，使用者只要构造不同的 $^t\mathbf{D}$ 表达式即可实现不同非线性材料的计算。ANSYS 的 usermat 子程序就是基于这样的思想设计的。下面我们来介绍一下 usermat 子程序的使用方法。

根据（7.11），$^t\mathbf{D}$ 的定义是应变增量和应力增量之间的关系。需要注意的是，$^t\mathbf{D}$ 通常是当前应变和应力的函数，由于单元内应变和应力不一定是常数，所以 $^t\mathbf{D}$ 通常也是坐标的函数。

7.4.1 usermat 子程序

ANSYS 的用户子程序包括本构模型开发子程序、单元开发子程序、载荷定义子程序、计算干预子程序等，其中材料非线性有限元计算所用的子程序主要就是本构模型开发子程序，该类型子程序

主要有 usermat、UserHyper、usercreep、usercr、user_tbelastic、userpl、usersw、userck、UserVisLaw、userfric、userfc 以及支持函数 eigen。其中，usermat 是使用范围最广的子程序，单元每一积分点在每一载荷子步的每次迭代都会调用 usermat。

usermat 实现本构模型开发主要包含两步：（1）给出一致切线刚度张量，即 ANSYS 中所称的雅克比矩阵（ $Jacobian = \frac{\partial \Delta \sigma}{\partial \Delta \varepsilon}$ ），这个雅可比矩阵就是 $^t\mathbf{D}$ ；（2）由给定的应变增量（由 ANSYS 主程序传入 usermat）计算出应力增量，实现应力更新。ANSYS 分别将应力应变张量及一致切线刚度张量存储为向量和矩阵形式，具体如下：

（1）应力、应变张量存储顺序：

杆单元应力状态：11

梁单元应力状态：11、13、12

二维平面应力/平面应变及轴对称应力状态：11、22、33、12

三维应力状态：11、22、33、12、23、13

（2）一致切线刚度张量存储顺序：

a、杆单元应力状态：

1111

b、梁单元应力状态：

1111	1122	1112
22211	2222	2212
1211	1222	1212

c、二维平面应力/平面应变状态及轴对称状态：

1111	1122	1133	1112
2211	2222	2233	2212
3311	3322	3333	3312
1211	1222	1233	1212

d、三维应力状态：

1111	1122	1133	1112	1123	1113
2211	2222	2233	2212	2223	2213
3311	3322	3333	3312	3323	3313
1211	1222	1233	1212	1223	1213

2311	2322	2333	2312	2323	2313
1311	1322	1333	1312	1323	1313

7.4.2 usermat 编译连接

想要利用子程序进行非线性计算首先需要实现子程序的编译连接，以生产用户定制版的 ANSYS，实现 ANSYS 主程序与子程序直接的相互连接和调用。

用户子程序以 FORTRAN 语言形式给出，所以实现编译连接前首先需要安装 FORTRAN 编译器，同时，不同版本 ANSYS 需要不同版本的编译器。本书采用较为成熟的 ANSYS 9.0 与 Compaq Visual FORTRAN 6.6 编译器配合使用。另外还需安装 Compaq Visual FORTRAN 6.6 的升级包，该升级包包含编译连接需要的函数库文件，否则编译会报错。

安装完成后，分别在 ANSYS 和 FORTRAN 安装目录中找到如图 7.4 所示文件，并且将所有文件拷贝至同一文件夹中。注意，该文件夹文件路径最好不要出现中文字符，否则编译容易报错。

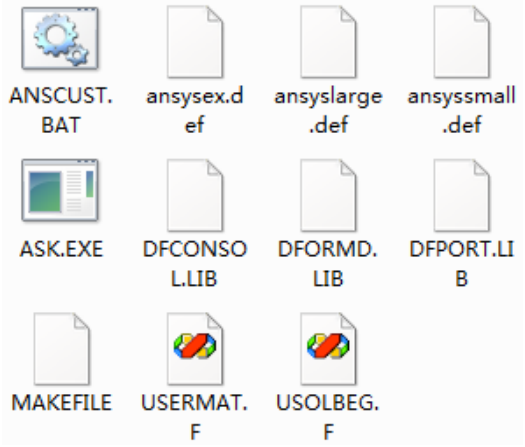


图 7.4 编译 usermat 所需文件

双击运行 ANSCUST.BAT 文件，弹出图 7.5 所示窗口，按照提示按任意键，再输入 Y 并回车，此时开始进入编译连接，连接成功后窗口会自动消失，同时上述文件夹中会新出现一批文件，包括 ANSYS.exe，此时编译连接完成。打开 ANSYS.exe 文件，弹出图 7.6 所示 ANSYS 输出窗口，按照提示输入/show,on 并回车，再输入/menu,on 并回车，此时即弹出 ANSYS 主程序窗口。

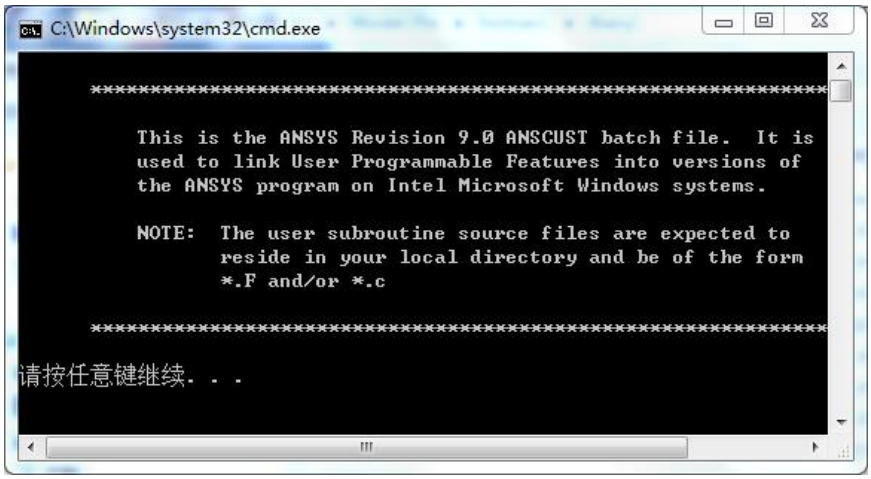


图 7.5 编译连接窗口

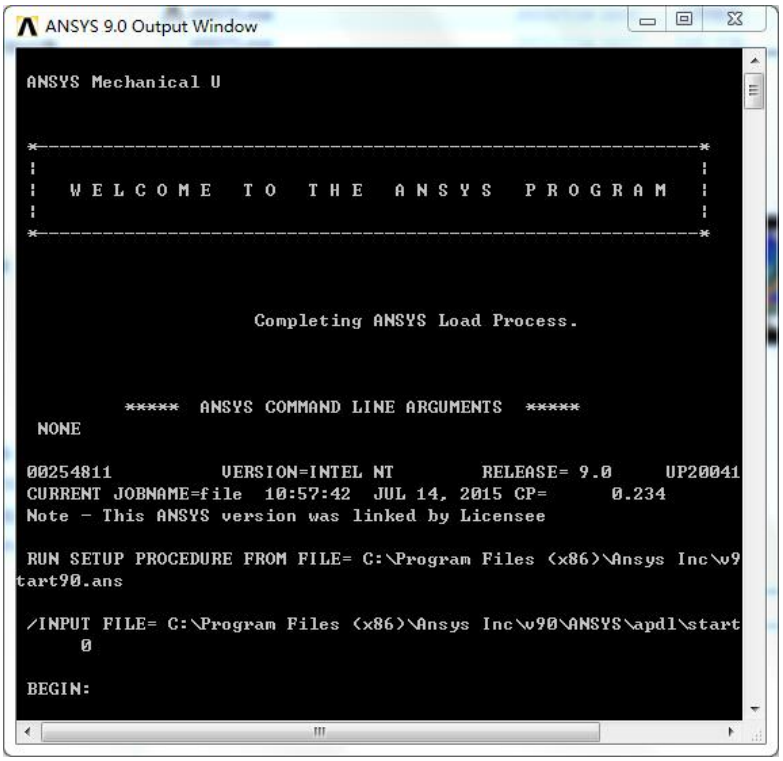


图 7.6 输出窗口

7.4.3 usermat 应用实例

现举例说明 usermat 子程序具体使用过程。假设某种材料为各向同性材料，弹性模量为 210GPa，泊松比为 0.3，拉伸损伤门槛应力为 100MPa。

对于三维应力状态，‘**D** 矩阵可以用拉梅系数 λ 和剪切模量 G 表示。

$$
{}^{t}\mathbf{D}=\begin{bmatrix}
\lambda+2G & \lambda & \lambda & 0 & 0 & 0\\
\lambda & \lambda+2G & \lambda & 0 & 0 & 0\\
\lambda & \lambda & \lambda+2G & 0 & 0 & 0\\
0 & 0 & 0 & G & 0 & 0\\
0 & 0 & 0 & 0 & G & 0\\
0 & 0 & 0 & 0 & 0 & G
\end{bmatrix} \tag{7.29}
$$

当拉伸应力达到门槛应力后，${}^{t}\mathbf{D}$ 对角元前三项折减为 $0.8(\lambda+2G)$，即：

$$
{}^{t}\mathbf{D}=\begin{bmatrix}
0.8(\lambda+2G) & \lambda & \lambda & 0 & 0 & 0\\
\lambda & 0.8(\lambda+2G) & \lambda & 0 & 0 & 0\\
\lambda & \lambda & 0.8(\lambda+2G) & 0 & 0 & 0\\
0 & 0 & 0 & G & 0 & 0\\
0 & 0 & 0 & 0 & G & 0\\
0 & 0 & 0 & 0 & 0 & G
\end{bmatrix} \tag{7.30}
$$

usermat 子程序具体内容如下：

```
*deck,usermat    parallel    user                                    gal
      subroutine usermat(
     &                    matId, elemId,kDomIntPt, kLayer, kSectPt,
     &                    ldstep,isubst,keycut,
     &                    nDirect,nShear,ncomp,nStatev,nProp,
     &                    Time,dTime,Temp,dTemp,
     &                    stress,statev,dsdePl,sedEl,sedPl,epseq,
     &                    Strain,dStrain, epsPl, prop, coords,
     &                    rotateM, defGrad_t, defGrad,
     &                    tsstif, epsZZ,
     &                    var1, var2, var3, var4, var5,
     &                    var6, var7, var8)
c**********************************************************************
#include "impcom.inc"
c
      INTEGER
     &                    matId, elemId,
     &                    kDomIntPt, kLayer, kSectPt,
```

```
     &                    ldstep,isubst,keycut,
     &                    nDirect,nShear,ncomp,nStatev,nProp
      DOUBLE PRECISION
     &                    Time,    dTime,    Temp,    dTemp,
     &                    sedEl,   sedPl,    epseq,   epsZZ
      DOUBLE PRECISION
     &                    stress   (ncomp   ), statev   (nStatev),
     &                    dsdePl   (ncomp,ncomp),
     &                    Strain   (ncomp   ), dStrain (ncomp   ),
     &                    epsPl    (ncomp   ), prop     (nProp   ),
     &                    coords   (3),        rotateM (3,3),
     &                    defGrad (3,3),       defGrad_t(3,3),
     &                    tsstif   (2)
      DOUBLE PRECISION var1, var2, var3, var4, var5,
     &                    var6, var7, var8
c-----------------------------------------------用户自定义-------------------------------------------------------
c
c-----------------------------------------------用户自定义变量---------------------------------------------------
c----------- EMOD(弹性模量)，NU（泊松比），SO（门槛应力）-------------------------------------------
-
c----------- EG（剪切模量），ELAM（拉梅系数）---------------------------------------------------------------
      DOUBLE PRECISION   EMOD,NU,SO,EG,ELAM
      INTEGER            k1,k2

c-----------------------------------------------材料参数赋值-----------------------------------------------------
      EMOD=prop(1)
      NU=prop(2)
      SO=prop(3)
      EG=EMOD/(2*(1+NU))
      ELAM=EMOD*NU/((1+NU)*(1-2*NU))
```

```
c----------------------------------------------------计算一致切线算子----------------------------------------------
      DO k1=1,3
        If (stress(k1).LE.EPO) THEN                    -门槛值判断
          dsdepl(k1,k1)=2*EG+ELAM                      -主应力方向
        ELSE
          dsdepl(k1,k1)=0.8*(2*EG+ELAM)
        END IF
      END DO

      DO k1=4,6
        dsdepl(k1,k1)=EG                               -剪切方向
      END DO
        dsdepl(1,2)=ELAM
        dsdepl(1,3)=ELAM
        dsdepl(2,1)=ELAM
        dsdepl(2,3)=ELAM
        dsdepl(3,1)=ELAM
        dsdepl(3,2)=ELAM
c----------------------------------------------------更新应力----------------------------------------------------
        DO k1=1,ncomp
        DO k2=1,ncomp
        stress(k2)=stress(k2)+dsdepl(k2,k1)*dStrain(k1)
        END DO
        END DO

        RETURN
        END
```

完成 usermat 程序编写后，按照上一小节方法进行子程序的编译连接，弹出 ANSYS 主程序窗口后开始建模分析。

按照图 7.7 所示尺寸建立有限元模型，模型一端施加约束，另外一端施加拉伸载荷。

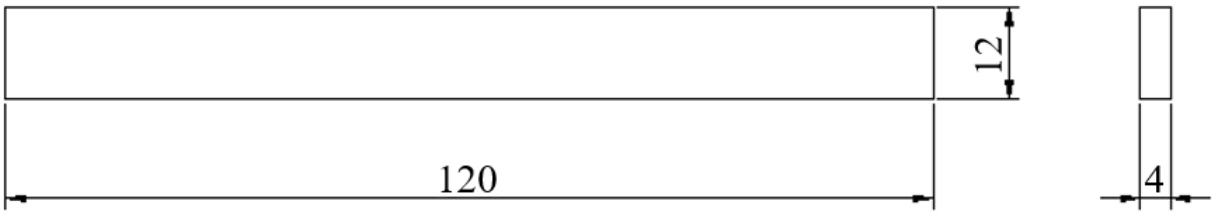


图 7.7 有限元模型尺寸

命令流如下：

```
finish
/clear
/PREP7
et,1,solid185                          ！定义单元类型
tb,user,1,1,3                          ！自定义材料类型，一个温度点，三个材料常数
tbdata,1,210e9,0.3,1e8                 ！给出弹性模量、泊松比及应力门槛值
K,1,0,6,0,                             ！建立几何模型
K,2,0,-6,0,
K,3,120,-6,0,
K,4,120,6,0,
K,4,120,6,0,
LSTR,        1,        2
LSTR,        2,        3
LSTR,        3,        4
LSTR,        4,        1
FLST,2,4,4
FITEM,2,4
FITEM,2,2
FITEM,2,1
FITEM,2,3
AL,P51X
VOFFST,1,4, ,
LESIZE,1,,,6                           ！划分网格密度
```

```
LESIZE,2,,,30
LESIZE,9,,,4
VSWEEP,ALL                          ！扫略划分网格
FINISH
/SOL                                ！进入求解器
FLST,2,1,5,ORDE,1
FITEM,2,4
DA,P51X,ALL,0                       ！约束一端面
FLST,2,1,5,ORDE,1
FITEM,2,6
DA,P51X,UX,0.15                     ！另一端面施加拉伸载荷（此处为位移载荷）
time,1
pred,on                             ！打开线性预测
nsubst,40,100,10                    ！定义载荷子步数
outres,all,all                      ！输出所有子步结果
solve
```

提取模型某一节点结果获得其应力应变曲线，如图 7.8 所示。由图可知，当应力小于 100MPa 时，材料应力应变曲线斜率保持不变，当应力值达到 100MPa 后，曲线斜率发生折减。本算例验证了 usermat 子程序用于材料非线性有限元计算的可行性。

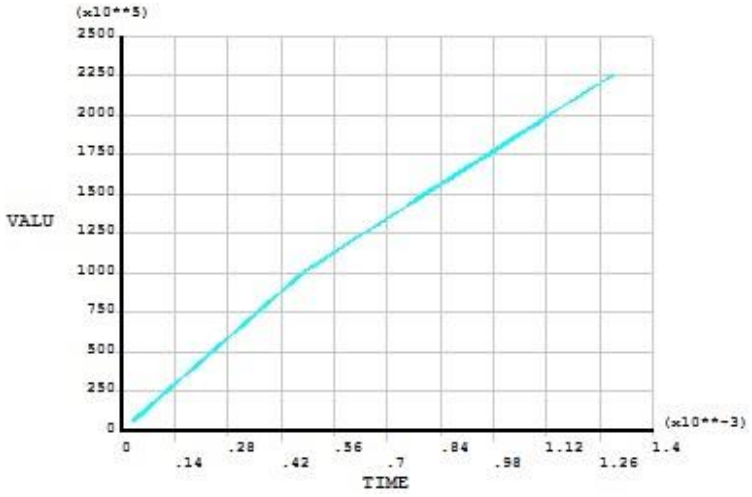


图 7.8 由 usermat 子程序计算出来的应力应变曲线

7.5 材料弹塑性本构关系

7.5.1 材料弹塑性行为的描述

由前文可知，具有非线性本构关系的材料被称为非线性材料。同时，根据是否存在某一临界应力值，使得当超过该临界应力值并卸去载荷以后，存在不可恢复的永久变形，又可将非线性材料分为非线性弹性材料和非线性弹塑性材料。

如图 7.9 所示，以材料的单向受力情况为例，对于非线性弹塑性材料，当卸载后，材料存在着不可恢复的永久变形，这是区别于非线性弹性材料的重要特征。弹塑性材料中存在的不可恢复的永久变形被称为塑性变形。由于塑性变形的存在，应力和应变之间不再存在一一对应的关系，而与加载历史有关。

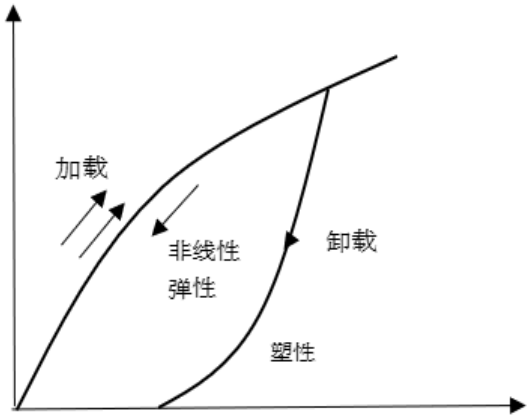


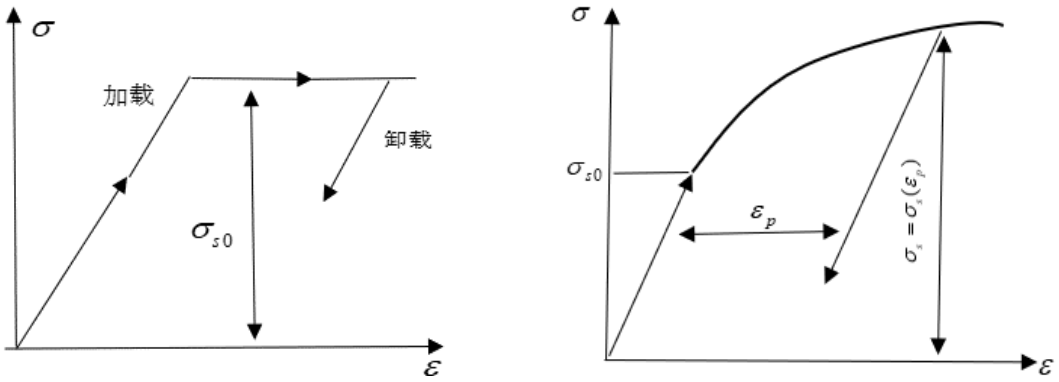
图 7.9 非线性弹性和塑性

由于在工程应用中的金属材料大多为非线性弹塑性材料，故本节主要研究非线性弹塑性材料，并详细介绍材料的弹塑性本构关系。

1. 单向加载

对于大多数弹塑性材料来说，存在一个比较明显的极限（屈服）应力σ_{s0}。当应力低于σ_{s0}时，材料保持为弹性状态，而当应力到达σ_{s0}以后，材料开始进入弹塑性状态。

当材料进入塑性状态后，如果应力不再增加，而变形可以继续增加，即变形处于不定的流动状态，如图 7.10(a)所示，则称该材料为理想弹塑性材料；反之如果应力不再增加时，变形也无法继续增加，如图 7.10 (b)所示，则称该材料为应变硬化弹塑性材料。



(a) 理想弹塑性　　(b) 应变硬化弹塑性

图 7.10 弹塑性加载曲线

2. 反向加载

对于应变硬化弹塑性材料，当单向加载进入塑性，而后在$\sigma_s=\sigma_{r1}$时反向加载至一定载荷时，材料将进入新的塑性状态。反向加载的屈服应力记为σ_{s1}。如图 7. 11 所示，如果$\sigma_{s1}=-\sigma_{r1}$，则称为材料为各向同性硬化的；如果$\sigma_{s1}=\sigma_{r1}-2\sigma_{s0}$，则称材料为运动硬化的；如果介于两者之间，即$\sigma_{r1}-2\sigma_{s0}<\sigma_{s1}<-\sigma_{r1}$，则称材料为混合硬化的。

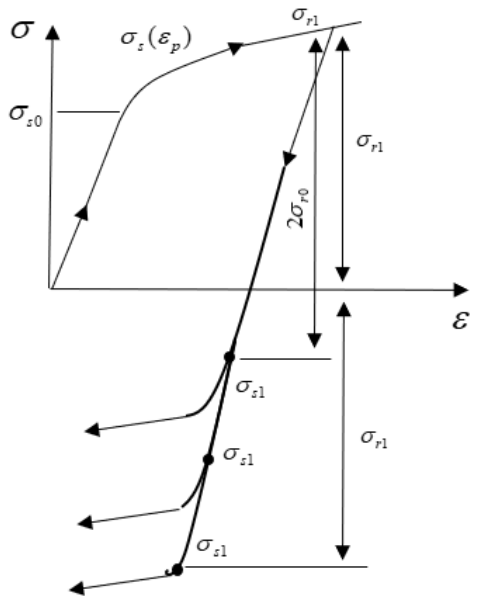


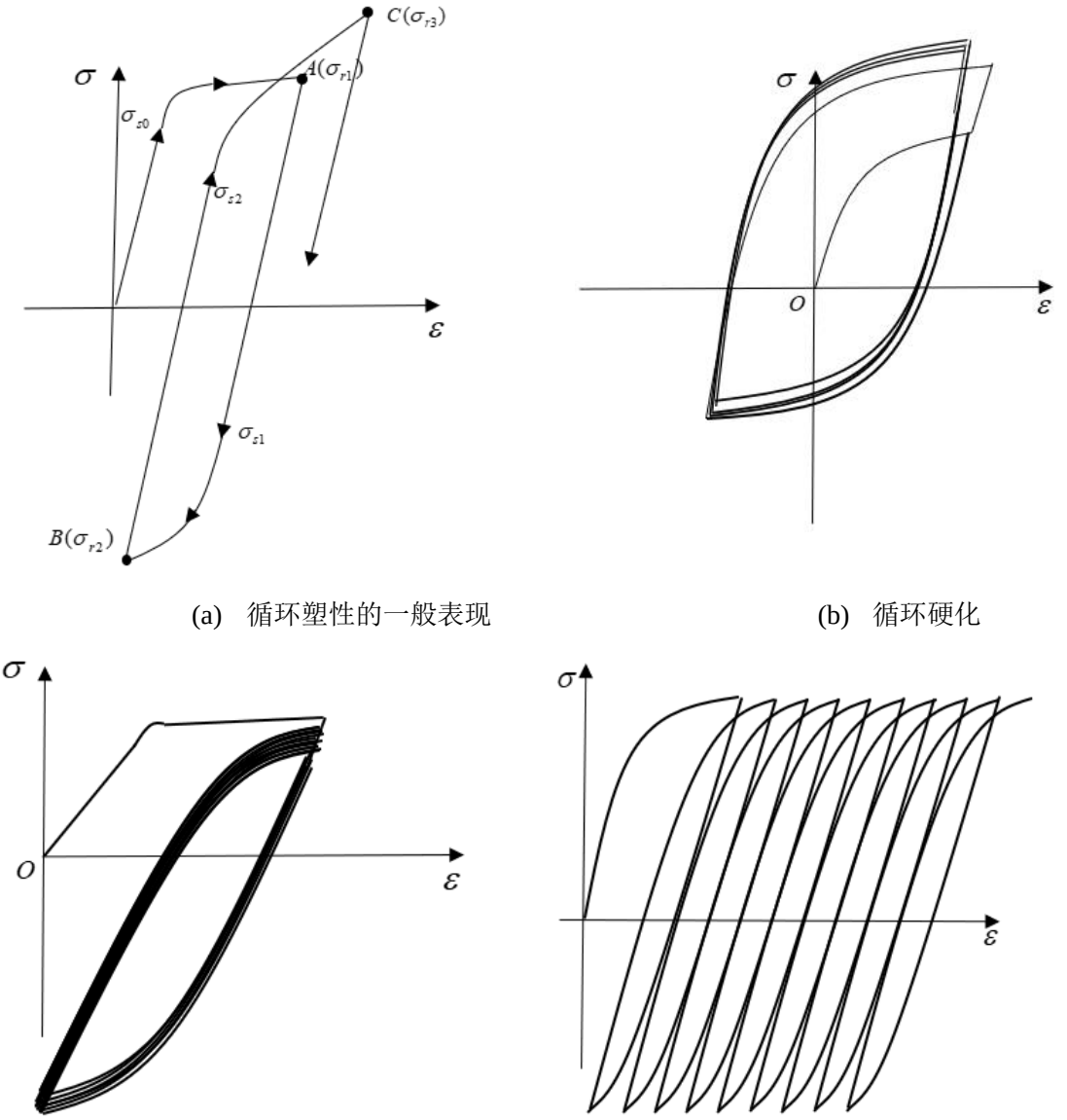
图 7.11 各种硬化塑性的特征

3. 循环加载

循环加载是在反向加载后，载荷再次反向，即回到正向，又一次到达新的屈服点和进入新的塑性变形，如此反复循环。循环加载时，材料有两种可能的响应特征，即“循环硬化”和“循环软化”。

在对称等幅应变控制的循环加载条件下，材料通常呈现出循环硬（软）化特征，如图 7.12(b)所

示。在非对称等幅应变控制的循环加载条件下，材料呈现循环松弛特征，如图 7.12(c)所示。在非对称等幅应力控制的循环加载条件下，材料呈现出循环蠕变的特征，又称为棘轮效应，如图 7.12 (d)所示。



(a) 循环塑性的一般表现　　(b) 循环硬化

(c) 循环松弛　　(d) 循环蠕变（棘轮现象）

图 7.12 材料循环塑性的特征行为

7.5.2 塑性力学的基本法则

1. 初始屈服准则

在简单的拉伸试验中，可直接比较应力与屈服极限σ_{s0}的大小，来判断材料处于弹性阶段还是塑性阶段。然而，在复杂应力状态下，一点的应力状态是由六个应力分量所确定的，所以无法通过单

一的屈服极限来判断材料是否进入塑性状态。这就需要引入适用于复杂应力状态的屈服准则。

对于各向同性材料，初始屈服准则可以表示为：

$$F^0\left(\sigma_{ij},k_0\right)=0 \tag{7.31}$$

其中 σ_{ij} 表示应力张量分量，k_0 是给定的材料参数。对于金属材料，通常采用的屈服准则有：

(1) Tresca 屈服准则

1864 年法国工程师 Tresca 发现在变形的金属表面有很细的痕纹，且这些痕纹的方向很接近于最大剪应力的方向。因此，他认为金属的塑性变形是由于剪切应力引起金属中晶体滑移而形成的。Tresca 提出：当最大剪切应力 $\tau_{\max}$ 达到某一极限值 τ_{s0} 时，材料便进入塑性状态。在三维应力空间中，当 $\sigma_1>\sigma_2>\sigma_3$ 时，由于 $\tau_{\max}=\dfrac{\sigma_1-\sigma_3}{2}$ 且 $\tau_{s0}=\dfrac{\sigma_{s0}}{2}$，这个条件可写为如下形式：

$$\sigma_1-\sigma_3=\sigma_{s0} \tag{7.32}$$

如果不知道主应力的大小和次序，则应写为：

$$F^0\left(\sigma_{ij},\sigma_{s0}\right)=\left[\left(\sigma_1-\sigma_2\right)^2-\sigma_{s0}^2\right]\left[\left(\sigma_2-\sigma_3\right)^2-\sigma_{s0}^2\right]\left[\left(\sigma_3-\sigma_1\right)^2-\sigma_{s0}^2\right]=0 \tag{7.33}$$

(2) Mises 屈服准则

1913 年 Mises 提出了如下的屈服条件：

$$F^0\left(\sigma_{ij},k_0\right)=f_0-k_0=0 \tag{7.34}$$

其中

$$f_0=\frac{1}{6}\left[\left(\sigma_1-\sigma_2\right)^2+\left(\sigma_2-\sigma_3\right)^2+\left(\sigma_3-\sigma_1\right)^2\right]$$

$$k_0=\frac{1}{3}\sigma_{s0}^2$$

f_0 还可表示为：

$$f_0=\frac{1}{6}\left[\left(\sigma_x-\sigma_y\right)^2+\left(\sigma_y-\sigma_z\right)^2+\left(\sigma_z-\sigma_x\right)^2+6\left(\tau_{xy}^2+\tau_{yz}^2+\tau_{zx}^2\right)\right]$$

2. 流动准则

流动准则用来规定材料进入塑性阶段后的塑性应变增量在各个方向上的分量，以及应力分量和应力增量之间的关系。Mises 流动准则假设塑性应变增量可从塑性势导出，即：

$$d\varepsilon_{ij}^{p}=d\lambda\frac{\partial Q}{\partial\sigma_{ij}} \tag{7.35}$$

其中，$d\varepsilon_{ij}^{p}$ 是塑性应变增量的分量；$d\lambda$ 是正的待定有限量，它的具体数值与材料硬化准则有关；Q 是塑性势函数，一般说它是应力状态和塑性应变的函数。对于稳定的$(d\sigma\cdot d\varepsilon>0)$应变硬化材料，$Q$ 通常取和后继屈服函数 F 相同的形式，称之为和屈服函数相关联的塑性势。对于关联塑性情况，流动法则表示为：

$$d\varepsilon_{ij}^{p}=d\lambda\frac{\partial F}{\partial\sigma_{ij}}=d\lambda\frac{\partial f}{\partial\sigma_{ij}} \tag{7.36}$$

3. 硬化准则

硬化法则是用来规定材料进入塑性变形后的后继屈服函数在应力空间中变化的规则。

（1） 各向同性硬化法则

此法则规定，当材料进入塑性变形后，加载曲面的形状、中心及其在应力空间中的方位均保持不变，而作形状相似的均匀膨胀。

如采用 Mises 屈服准则，则各向同性硬化的后继屈服函数可表示为：

$$F\left(\sigma_{ij},k\right)=f-k=0 \tag{7.37}$$

其中，f 与 Mises 初始屈服准则中的 f_0 相同，所不同的是 k 发生了变化。即：

$$f=f_0$$

$$k=\frac{1}{3}\sigma_s^2\left(\bar{\varepsilon}_p\right)$$

式中 σ_s 是实际的弹塑性应力，它是等效塑性应变 $\bar{\varepsilon}_p$ 的函数。

$$\bar{\varepsilon}_p=\int d\bar{\varepsilon}_p=\int\left(\frac{2}{3}d\bar{\varepsilon}_{ij}^{p}d\bar{\varepsilon}_{ij}^{p}\right)^{1/2} \tag{7.38}$$

定义

$$E_p=\frac{d\sigma_s}{d\bar{\varepsilon}_p}$$

为材料的塑性模量，又称之为硬化系数。它和弹性模量及切向模量 E_t 的关系为：

$$E_p = \frac{EE_t}{E - E_t} \tag{7.39}$$

（2） 运动强化法则

此法则规定，当材料进入塑性变形后，加载曲面的形状、大小及其在应力空间中的方位均保持不变，而作刚体移动。

如采用 Mises 屈服准则，则运动强化后继屈服函数可表示为：

$$F\left(\sigma_{ij}, \sigma_{ij}^{'}, k\right) = f - k = 0 \tag{7.40}$$

其中 $\sigma_{ij}^{'}$ 为初始屈服面中心在应力空间中的刚体位移量，k 与 Mises 初始屈服准则中的 k_0 相同，所不同的是 f 发生了变化。即：

$$f\left(\sigma_{ij}\right) = f_0\left(\sigma_{ij} - \sigma_{ij}^{'}\right)$$

$$k = k_0$$

7.5.3 弹塑性增量的应力应变关系

在三维应力空间中，各向同性材料的弹性本构关系表现为：

$$\boldsymbol{\sigma} = \boldsymbol{D}^e \boldsymbol{\varepsilon} \tag{7.41}$$

当材料进入塑性阶段后，材料的弹塑性本构关系将发生变化。下面给出弹塑性增量本构关系的推导过程及具体表达式。

定义应力应变的增量关系式为：

$$d\sigma_{ij} = D_{ijkl}^{ep} d\varepsilon_{kl} \tag{7.42}$$

其中 $D_{ijkl}^{ep} = D_{ijkl}^{e} - D_{ijkl}^{p}$。$D_{ijkl}^{p}$ 为塑性矩阵，它的一般表达式为：

$$\boldsymbol{D}^p = \frac{\boldsymbol{D}^e \left(\frac{\partial f}{\partial \boldsymbol{\sigma}}\right)\left(\frac{\partial f}{\partial \boldsymbol{\sigma}}\right)^T \boldsymbol{D}^e}{\left(\frac{\partial f}{\partial \boldsymbol{\sigma}}\right)^T \boldsymbol{D}^e \left(\frac{\partial f}{\partial \boldsymbol{\sigma}}\right) + \frac{4}{9}\sigma_s^2 E_p} \tag{7.43}$$

对于各向同性硬化材料，屈服函数中

$$f = \frac{1}{6}\left[\left(\sigma_x - \sigma_y\right)^2 + \left(\sigma_y - \sigma_z\right)^2 + \left(\sigma_z - \sigma_x\right)^2 + 6\left(\tau_{xy}^2 + \tau_{yz}^2 + \tau_{zx}^2\right)\right] \tag{7.44}$$

可改写为：

$$f=\frac{1}{2}\left(s_x^2+s_y^2+s_z^2+2\tau_{xy}^2+2\tau_{yz}^2+2\tau_{zx}^2\right) \tag{7.45}$$

其中

$$s_i=\sigma_i-\frac{1}{3}\left(\sigma_x+\sigma_y+\sigma_z\right)\qquad\left(i=x,y,z\right)$$

可以得到：

$$\frac{\partial f}{\partial \boldsymbol{\sigma}}=\begin{bmatrix}s_x & s_y & s_z & 2\tau_{xy} & 2\tau_{yz} & 2\tau_{xz}\end{bmatrix}^{\mathrm{T}} \tag{7.46}$$

将 $\boldsymbol{D}^e$ 和式(7.46)代入式(7.43)，可得：

$$\boldsymbol{D}^p=\frac{9G^2\boldsymbol{S}\boldsymbol{S}^T}{\sigma_s^2\left(3G+E_p\right)} \tag{7.47}$$

其中

$$\boldsymbol{S}=\begin{bmatrix}s_x & s_y & s_z & \tau_{xy} & \tau_{yz} & \tau_{xz}\end{bmatrix}^{\mathrm{T}}$$

至此，即可得到各向同性材料的弹塑性增量本构关系的具体表达式。

7.6 材料非线性有限元程序设计

7.6.1 材料非线性有限元程序算法

本节介绍材料非线性有限元程序设计的算法步骤，为以 C 语言为编程语言编写材料非线性有限元程序打下理论基础。具体如下：

(1) 将载荷等分为 I 个增量步。

(2) 初始状态下，$\boldsymbol{u}_0=\boldsymbol{0}$，$\boldsymbol{\varepsilon}_0=\boldsymbol{0}$，$\boldsymbol{\sigma}_0=\boldsymbol{0}$。

(3) 赋值当前增量步 i=1，位移增量比较值 $\Delta\boldsymbol{u}'=\boldsymbol{0}$。

(4) 按照当前本构关系和增量有限元法，计算位移增量 $\Delta\boldsymbol{u}$。

(5) 根据位移增量 $\Delta\boldsymbol{u}$，计算应变增量 $\Delta\boldsymbol{\varepsilon}_i$。

$$\Delta\boldsymbol{\varepsilon}_i=\boldsymbol{B}\Delta\boldsymbol{u} \tag{7.48}$$

(6) 按照弹性本构关系，计算应力增量 $\Delta\boldsymbol{\sigma}_i^e$。

$$\Delta\boldsymbol{\sigma}_i^e=\boldsymbol{D}^e\Delta\boldsymbol{\varepsilon}_i \tag{7.49}$$

(7) 计算第 i 增量步的试探应力 $\boldsymbol{\sigma}_i$。

$$\boldsymbol{\sigma}_i = \boldsymbol{\sigma}_{i-1} + \Delta\boldsymbol{\sigma}_i^e \tag{7.50}$$

(8) 计算位移增量差值的二范数，若 $\left\|\Delta\boldsymbol{u} - \Delta\boldsymbol{u}^{'}\right\| \le \mathrm{c}$（c 为设定的较小的正常数），则进一步比较当前增量步与总步数的关系，若 $i < I$，则 $i = i+1$，$\Delta\boldsymbol{u}^{'} = \boldsymbol{0}$，并返回步骤（4），否则计算结束；若 $\left\|\Delta\boldsymbol{u} - \Delta\boldsymbol{u}^{'}\right\| > \mathrm{c}$，则赋值 $\Delta\boldsymbol{u}^{'} = \Delta\boldsymbol{u}$，并进入下一步。

(9) 根据试探应力 $\boldsymbol{\sigma}_i$，计算屈服函数 F_i^0 的值。若采用 Mises 屈服准则，则为：

$$\frac{1}{6}\left[\left(\sigma_x - \sigma_y\right)^2 + \left(\sigma_y - \sigma_z\right)^2 + \left(\sigma_z - \sigma_x\right)^2 + 6\left(\tau_{xy}^2 + \tau_{yz}^2 + \tau_{zx}^2\right)\right] - \frac{1}{3}\sigma_{s0}^2 = 0 \tag{7.51}$$

(10)如果 $F_i^0 \le 0$，则单元处于弹性阶段，单元的应力增量完全由弹性部分构成，第(5)步中计算的试探应力即为第 i 增量步下的真实应力；如果 $F_i^0 > 0$，则还需要判断第 i-1 增量步下的屈服函数 F_{i-1}^0，如果 $F_{i-1}^0 \le 0$，则单元的应力增量由弹性部分和塑性部分构成，弹性部分的应力增量为 $r\Delta\boldsymbol{\sigma}_i^e$。其中 r 为弹性部分比例因子（0<r<1），可根据屈服函数 $F_i^0 = F\left(\sigma_{i-1} + r\Delta\sigma^e, \kappa_{i-1}\right) = 0$ 计算得到。如果 $F_{i-1}^0 > 0$，单元的应力增量完全有由塑性部分构成，弹性部分比例因子 r=0。

(11)采用加权平均的方法，计算第 i 增量步的本构关系矩阵 $\overline{\boldsymbol{D}}^{ep}$，并返回步骤（4）。

$$\overline{\boldsymbol{D}}^{ep} = r\boldsymbol{D}^e + \left(1-r\right)\boldsymbol{D}^{ep} \tag{7.52}$$

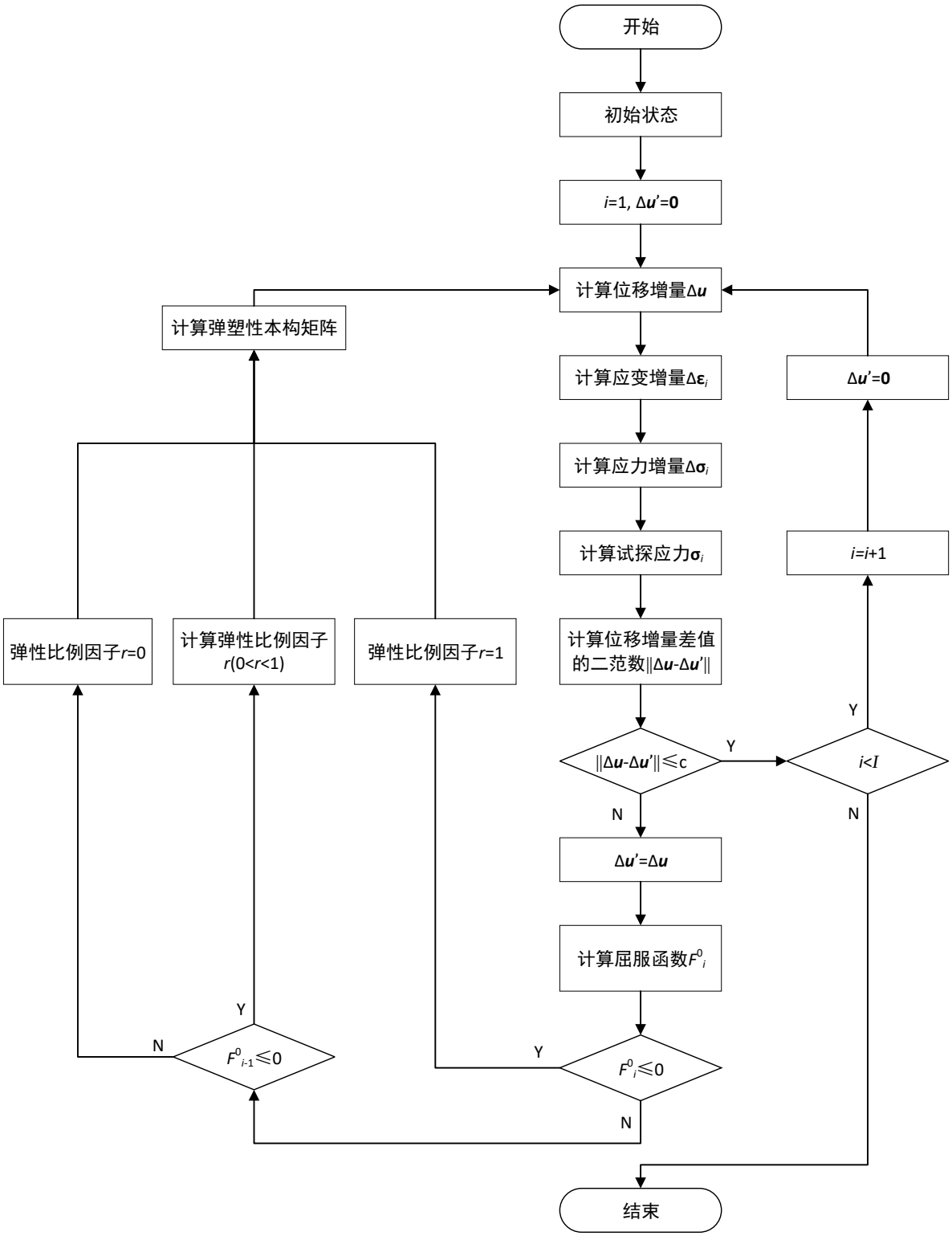


图 7.13 材料非线性有限元程序算法流程图

7.6.2 材料非线性有限元程序的 C 语言实现

程序见附录。

7.7 典型发动机零部件弹塑性变形分析

现以扭转平板为例，进行结构件的弹塑性变形分析。图 7.14 和图 7.15 分别给出了在 ANSYS 中建立的扭转平板的几何模型和有限元模型。其中长度为 30cm，宽度为 10cm，厚度为 4cm；长度方向分段数为 10，宽度方向分段数为 10，厚度方向分段数为 4。在扭转板的根部平面节点施加全约束，在其顶部平面节点施加力载荷。单节点受力 150N，所有顶部平面节点共受力 8250N。

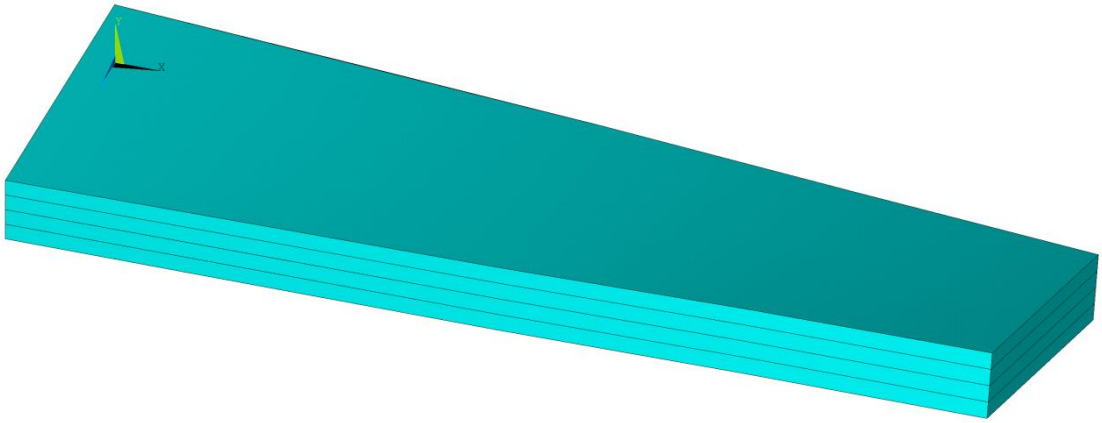


图 7.14 扭转平板的几何模型

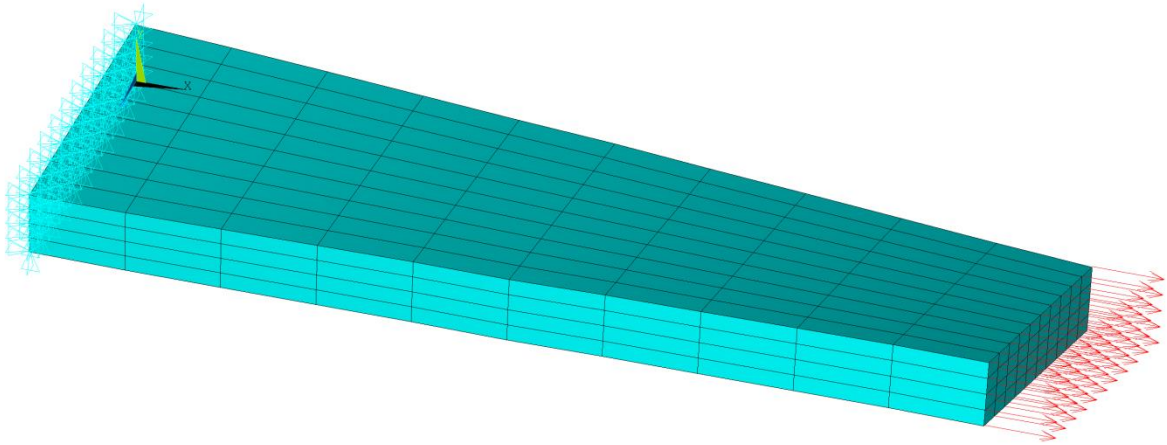


图 7.15 扭转平板的有限元网格模型

通过将扭转平板的有限元信息（网格、边界条件等）按特定的格式输出到文本文件中，随后调用材料非线性有限元程序进行有限元网格信息读取及计算，实现了对扭转平板的弹塑性变形分析。程序中设置的材料参数如表 7.1 所示，载荷步数为 50 步。

表 7.1 扭转平板的材料非线性力学参数

参数	弹性模量 (GPa)	泊松比	屈服应力 (MPa)	切向模量 (GPa)
数值	210	0.3	200	30

通过将材料非线性有限元程序的计算结果，包括沿 x 轴方向的单元应力重新输入到 ANSYS 中，并显示相应的云图，如图 7.16 所示，实现了对扭转平板的弹塑性计算结果的后处理。

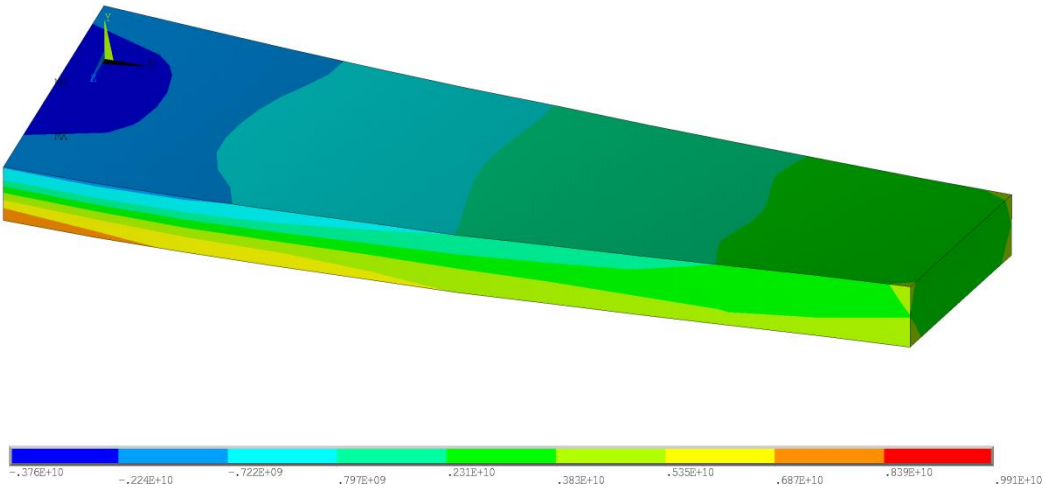


图 7.16 扭转平板的应力云图

第八章 动力学问题的有限元法

8.1 基本方程

前文介绍的都属于静力学问题。静力学与动力学问题的本质区别就是静力学问题中不考虑自身加速度的作用。而对于动力学问题，加速度的影响不可忽略。根据运动学原理，连续体Ω内部任意一点应该满足如下：

① 平衡方程：

$$\sigma_{ij,j} + f_i - \rho\ddot{u}_i - \mu\dot{u}_i = 0 \quad (i,j = 1, 2, 3) \tag{8.1}$$

其中 $\sigma_{ij,j} = \dfrac{\partial \sigma_{ij}}{\partial x_j}$，$f_i$ 是沿方向的体积力密度，ρ 是质量密度，$\ddot{u}_i$ 是沿 i 方向的加速度，$\dot{u}_i$ 是沿 i 方向的速度，μ 是阻尼系数。

② 几何方程：

$$\varepsilon_{ij} = \frac{1}{2}\left(u_{i,j} + u_{j,i}\right) \quad (i,j = 1, 2, 3) \tag{8.2}$$

其中 $u_{i,j} = \dfrac{\partial u_i}{\partial x_j}$。

③ 物理方程：

$$\sigma_{ij} = D_{ijkl}\varepsilon_{kl} \quad (i,j,k,l = 1, 2, 3) \tag{8.3}$$

其中 D_{ijkl} 是弹性参数。

④ 边界条件：

$$u_i = \bar{u}_i \text{（在}\Gamma_1\text{边界上）}，\ \sigma_{ij}n_j = q_i \text{（在}\Gamma_2\text{边界上）} \tag{8.4}$$

⑤ 初始条件：

$$u_i\left(x,y,z,t=0\right) = u_i\left(x,y,z\right)，\ \dot{u}_i\left(x,y,z,t=0\right) = \dot{u}_i\left(x,y,z\right) \tag{8.5}$$

我们的目标是求解任意一个时刻 t 的平衡方程(8.1)并让它满足边界条件(8.4)。根据加权余量法的原理，得知同时满足平衡方程(8.1)和边界条件(8.4)与满足积分式（8.6）是等价的。理由如下，如果应力严格满足方程(8.1)和边界条件(8.4)，那么积分号内部全为0，因此方程左右两边恒等。如果应力在任意虚位移 δu_i 下都满足(8.6)式，那么必然要求 $\sigma_{ij,j} + f_i - \rho\ddot{u}_i - \mu\dot{u}_i = 0$，且 $\sigma_{ij}n_j - q_i = 0$，即

满足满足方程(8.1)和边界条件(8.4)。

$$\int_\Omega \delta u_i \left(\sigma_{ij,j} + f_i - \rho \ddot{u}_i - \mu \dot{u}_i\right) dv - \int_{\Gamma_2} \delta u_i \left(\sigma_{ij} n_j - q_i\right) ds = 0 \tag{8.6}$$

其中 δu_i 是微元的任意虚位移。积分式第一项采用分部积分进行计算得：

$$\int_\Omega \delta u_i \sigma_{ij,j} dv = \int_\Omega \left(\delta u_i \sigma_{ij}\right)_{,j} dv - \int_\Omega \delta u_{i,j} \sigma_{ij} dv \tag{8.7}$$

再用高斯定律可以将 $\int_\Omega \left(\delta u_i \sigma_{ij}\right)_{,j} dv$ 转化为面积分的形式，即：

$$\int_\Omega \left(\delta u_i \sigma_{ij}\right)_{,j} dv = \int_\Gamma \delta u_i \sigma_{ij} n_j ds = \int_{\Gamma_2} \delta u_i \sigma_{ij} n_j ds \tag{8.8}$$

其中 Γ 是区域 Ω 的边界，$\Gamma = \Gamma_1 + \Gamma_2$。

因为在 Γ_1 上位移是确定的，所以 δu_i =0，$\int_\Gamma \delta u_i \sigma_{ij} n_j ds = \int_{\Gamma_2} \delta u_i \sigma_{ij} n_j ds$。将(8.8)代入(8.7)后得：

$$\int_\Omega \delta u_i f_i dv + \int_{\Gamma_2} \delta u_i q_i ds = \int_\Omega \left(\delta u_{i,j} \sigma_{ij} + \delta u_i \rho \ddot{u}_i + \delta u_i \mu \dot{u}_i\right) dv \tag{8.9}$$

其中 $\delta u_{i,j} \sigma_{ij} = \delta \varepsilon_{ij} \sigma_{ij}$。如果材料是线弹性的，则满足物理（本构）方程（8.3），$\delta \varepsilon_{ij} \sigma_{ij} = \delta \varepsilon_{ij} D_{ijkl} \varepsilon_{kl}$。

$$\int_\Omega \delta u_i f_i dv + \int_{\Gamma_2} \delta u_i q_i ds = \int_\Omega \left(\delta \varepsilon_{ij} D_{ijkl} \varepsilon_{kl} + \delta u_i \rho \ddot{u}_i + \delta u_i \mu \dot{u}_i\right) dv \tag{8.10}$$

接下来，我们采用有限元法求解方程(8.10)。首先将研究区域 Ω 进行离散化得：

$$\sum_e \int_{\Omega^e} \delta u_i f_i dv + \sum_e \int_{\Gamma_2^e} \delta u_i q_i ds = \sum_e \int_{\Omega^e} \left(\delta \varepsilon_{ij} D_{ijkl} \varepsilon_{kl} + \delta u_i \rho \ddot{u}_i + \delta u_i \mu \dot{u}_i\right) dv \tag{8.10}$$

采用形函数插值的方法，可以建立节点位移 $\mathbf{a}^e$ 和和单元内位移 $\mathbf{u}$ 之间的关系：

$$\mathbf{u} = \mathbf{N} \cdot \mathbf{a}^e, \quad \dot{\mathbf{u}} = \mathbf{N} \cdot \dot{\mathbf{a}}^e, \quad \ddot{\mathbf{u}} = \mathbf{N} \cdot \ddot{\mathbf{a}}^e \tag{8.11}$$

单元内应变与节点位移之间的关系为：

$$\boldsymbol{\varepsilon} = \mathbf{B} \cdot \mathbf{a}^e \tag{8.12}$$

将(8.11)和(8.12)代入(8.10)后得：

$$\begin{aligned} &\sum_e \int_{\Omega^e} \delta \mathbf{a}^{eT} \cdot \mathbf{N}^T \cdot \mathbf{f} dv + \sum_e \int_{\Gamma_2^e} \delta \mathbf{a}^{eT} \cdot \mathbf{N}^T \cdot \mathbf{q} ds \\ &= \sum_e \int_{\Omega^e} \left(\delta \mathbf{a}^{eT} \cdot \mathbf{B}^T \cdot \mathbf{D} \cdot \mathbf{B} \cdot \mathbf{a}^e + \delta \mathbf{a}^{eT} \cdot \mathbf{N}^T \cdot \mathbf{N} \cdot \ddot{\mathbf{a}}^e \rho + \delta \mathbf{a}^{eT} \cdot \mathbf{N}^T \cdot \mathbf{N} \cdot \dot{\mathbf{a}}^e \mu\right) dv \end{aligned} \tag{8.13}$$

我们可以通过抽取矩阵 $\mathbf{G}^e$ 建立 $\mathbf{a}^e$ 与 $\mathbf{a}$ 之间的关系：

$$\mathbf{a}^e = \mathbf{G}^e \cdot \mathbf{a}, \dot{\mathbf{a}}^e = \mathbf{G}^e \cdot \dot{\mathbf{a}}, \ddot{\mathbf{a}}^e = \mathbf{G}^e \cdot \ddot{\mathbf{a}} \tag{8.14}$$

将(8.14)代入(8.13)后得：

$$
\begin{aligned}
&\delta\mathbf{a}^T\cdot\left(\sum_e\int_{\Omega^e}\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot\mathbf{f}dv\right)+\delta\mathbf{a}^T\cdot\left(\sum_e\int_{\Gamma_2^e}\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot\mathbf{q}ds\right)\\
&=\delta\mathbf{a}^T\cdot\sum_e\int_{\Omega^e}\left(\mathbf{G}^{eT}\cdot\mathbf{B}^T\cdot\mathbf{D}\cdot\mathbf{B}\cdot\mathbf{G}^e\right)dv\cdot\mathbf{a}\\
&+\delta\mathbf{a}^T\cdot\sum_e\int_{\Omega^e}\left(\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot\mathbf{N}\cdot\mathbf{G}^e\rho\right)dv\cdot\ddot{\mathbf{a}}\\
&+\delta\mathbf{a}^T\cdot\sum_e\int_{\Omega^e}\left(\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot\mathbf{N}\cdot\mathbf{G}^e\mu\right)dv\cdot\dot{\mathbf{a}}
\end{aligned}
\tag{8.15}
$$

由于在任意 $\delta\mathbf{a}^T$ 下方程(8.15)都是成立的，因此

$$
\begin{aligned}
&\left(\sum_e\int_{\Omega^e}\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot\mathbf{f}dv\right)+\left(\sum_e\int_{\Gamma_2^e}\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot\mathbf{q}ds\right)\\
&=\sum_e\int_{\Omega^e}\left(\mathbf{G}^{eT}\cdot\mathbf{B}^T\cdot\mathbf{D}\cdot\mathbf{B}\cdot\mathbf{G}^e\right)dv\cdot\mathbf{a}+\sum_e\int_{\Omega^e}\left(\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot\mathbf{N}\cdot\mathbf{G}^e\rho\right)dv\cdot\ddot{\mathbf{a}}\\
&+\sum_e\int_{\Omega^e}\left(\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot\mathbf{N}\cdot\mathbf{G}^e\mu\right)dv\cdot\dot{\mathbf{a}}
\end{aligned}
\tag{8.16}
$$

上式简化为：

$$
\mathbf{M}\cdot\ddot{\mathbf{a}}+\mathbf{C}\cdot\dot{\mathbf{a}}+\mathbf{K}\cdot\mathbf{a}=\mathbf{Q}
\tag{8.17}
$$

其中

$$
\mathbf{M}=\sum_e\int_{\Omega^e}\left(\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot\mathbf{N}\cdot\mathbf{G}^e\rho\right)dv
\tag{8.18a}
$$

$$
\mathbf{C}=\sum_e\int_{\Omega^e}\left(\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot\mathbf{N}\cdot\mathbf{G}^e\mu\right)dv
\tag{8.18b}
$$

$$
\mathbf{K}=\sum_e\int_{\Omega^e}\left(\mathbf{G}^{eT}\cdot\mathbf{B}^T\cdot\mathbf{D}\cdot\mathbf{B}\cdot\mathbf{G}^e\right)dv
\tag{8.18c}
$$

$$
\mathbf{Q}=\left(\sum_e\int_{\Omega^e}\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot\mathbf{f}dv\right)+\left(\sum_e\int_{\Gamma_2^e}\mathbf{G}^{eT}\cdot\mathbf{N}^T\cdot\mathbf{q}ds\right)
\tag{8.18d}
$$

$$
\begin{aligned}
&\mathbf{M}^e=\int_{\Omega^e}\left(\mathbf{N}^T\cdot\mathbf{N}\rho\right)dv,\mathbf{C}^e=\int_{\Omega^e}\left(\mathbf{N}^T\cdot\mathbf{N}\cdot\mu\right)dv\\
&\mathbf{K}^e=\int_{\Omega^e}\left(\mathbf{B}^T\cdot\mathbf{D}\cdot\mathbf{B}\right)dv,\mathbf{Q}^e=\int_{\Omega^e}\mathbf{N}^T\cdot\mathbf{f}dv+\int_{\Gamma_2^e}\mathbf{N}^T\cdot\mathbf{q}ds
\end{aligned}
\tag{8.19}
$$

方程(8.17)就是动力学的有限元方程。

8.2 有限元方程的解法

方程(8.17)是二阶常微分方程组，理论上求解该方程组可以采用各类有限差分方法。本文介绍最常用的中心差分法。加速度和速度可以表示为位移的表达式：

$$\ddot{\mathbf{a}}_t = \frac{1}{\Delta t^2}\left(\mathbf{a}_{t-\Delta t} - 2\mathbf{a}_t + \mathbf{a}_{t+\Delta t}\right), \dot{\mathbf{a}}_t = \frac{1}{2\Delta t}\left(-\mathbf{a}_{t-\Delta t} + \mathbf{a}_{t+\Delta t}\right) \tag{8.20}$$

假设 t 时刻满足(8.17)的运动方程，则：

$$\mathbf{M}\cdot\ddot{\mathbf{a}}_t + \mathbf{C}\cdot\dot{\mathbf{a}}_t + \mathbf{K}\cdot\mathbf{a}_t = \mathbf{Q}_t \tag{8.21}$$

将(8.20)代入方程(8.21)后得：

$$\mathbf{M}\cdot\frac{1}{\Delta t^2}\left(\mathbf{a}_{t-\Delta t} - 2\mathbf{a}_t + \mathbf{a}_{t+\Delta t}\right) + \mathbf{C}\cdot\frac{1}{2\Delta t}\left(-\mathbf{a}_{t-\Delta t} + \mathbf{a}_{t+\Delta t}\right) + \mathbf{K}\cdot\mathbf{a}_t = \mathbf{Q}_t \tag{8.22}$$

合并同类项后得：

$$\left(\mathbf{M}\cdot\frac{1}{\Delta t^2} + \mathbf{C}\cdot\frac{1}{2\Delta t}\right)\mathbf{a}_{t+\Delta t} = \mathbf{Q}_t - \left(\mathbf{K} - \frac{2\mathbf{M}}{\Delta t^2}\right)\cdot\mathbf{a}_t - \left(\frac{\mathbf{M}}{\Delta t^2} - \frac{\mathbf{C}}{2\Delta t}\right)\cdot\mathbf{a}_{t-\Delta t} \tag{8.23}$$

由此可见，已知$\mathbf{a}_t$和$\mathbf{a}_{t-\Delta t}$即可根据（8.23）计算出$\mathbf{a}_{t+\Delta t}$。在 t=0 时刻，由于并不知道$\mathbf{a}_{t-\Delta t}$的数值，因此需要一个起步过程。由于在 $t=0$ 时刻，我们知道$\mathbf{a}_0$、$\dot{\mathbf{a}}_0$和$\ddot{\mathbf{a}}_0$，那么根据(8.20)可知：

$$\mathbf{a}_{-\Delta t} = \mathbf{a}_0 - \Delta t\dot{\mathbf{a}}_0 + \frac{\Delta t^2}{2}\ddot{\mathbf{a}}_0$$

（8.24）其中$\mathbf{a}_0$和$\dot{\mathbf{a}}_0$可从给定的初始条件得到，而$\ddot{\mathbf{a}}_0$则可以利用 t=0 时的运动方程(8.21)得到：

$$\ddot{\mathbf{a}}_0 = \mathbf{M}^{-1}\cdot\left(\mathbf{Q}_0 - \mathbf{C}\cdot\dot{\mathbf{a}}_0 - \mathbf{K}\cdot\mathbf{a}_0\right) \tag{8.25}$$

此刻，可将利用中心差分法逐步求解运动方程的算法步骤归结如下：

（1）根据研究对象的几何和物理特性以及单元划分的情况建立刚度矩阵 **K**、质量 **M** 和阻尼矩阵 **C**；

（2）给定初始位移$\mathbf{a}_0$和初始速度$\dot{\mathbf{a}}_0$；

（3）根据方程(8.25)计算$\ddot{\mathbf{a}}_0$；

（4）选择计算时间步长Δt，$\Delta t < \Delta t_{ct}$，并计算积分常数$c_0 = \frac{1}{\Delta t^2}$，$c_1 = \frac{1}{2\Delta t}$，$c_2 = 2c_0$，$c_3 = 1/c_2$。

（5）采用(8.24)计算$\mathbf{a}_{-\Delta t}$；

（6）由(8.23)计算出Δt时刻的的位移$\mathbf{a}_{\Delta t}$；

（7）不断由(8.23)计算出各个时刻的位移$\mathbf{a}_{\Delta t}, \mathbf{a}_{2\Delta t}, \mathbf{a}_{3\Delta t}\cdots$。

8.3 弹性杆振动的程序实现

本节我们以弹性杆的轴向振动为例来说明动力学有限元的计算过程。如图 8.1 所示弹性杆，左端固支，右端受到轴向力作用。我们要计算杆的端点在不同时刻的位移，采用有限元法求解。

图 8.1 弹性杆的轴向振动问题

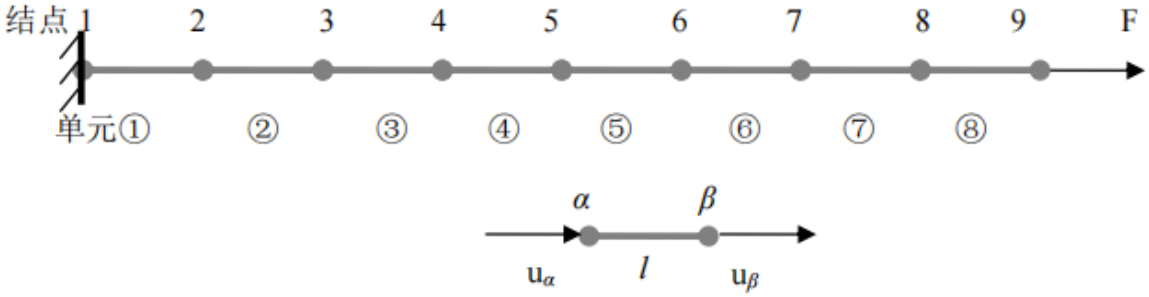


图 8.2 弹性杆的有限元模型

如图 8.2 所示，首先将弹性杆离散成一系列两结点杆单元。一共包含 n 个两节点杆单元。单元内位移可以用形函数矩阵表示，如等式（8.26）所示。

$$u=\mathbf{N}^e\cdot\mathbf{a}^e=\begin{bmatrix}1-\dfrac{x}{l} & \dfrac{x}{l}\end{bmatrix}\cdot\begin{bmatrix}u_\alpha^e\\ u_\beta^e\end{bmatrix},\mathbf{N}^e=\begin{bmatrix}1-\dfrac{x}{l} & \dfrac{x}{l}\end{bmatrix},\mathbf{a}^e=\begin{bmatrix}u_\alpha^e\\ u_\beta^e\end{bmatrix} \tag{8.26}$$

式中 x 表示单元内一点到结点 α 的距离，l 是杆单元的长度，$\mathbf{a}^e$ 是单元 e 的结点位移。

单元内应变可以用几何矩阵与结点位移向量的乘积来计算：

$$\varepsilon=\mathbf{B}^e\cdot\mathbf{a}^e=\begin{bmatrix}-\dfrac{1}{l} & \dfrac{1}{l}\end{bmatrix}\cdot\begin{bmatrix}u_\alpha^e\\ u_\beta^e\end{bmatrix},\mathbf{B}^e=\begin{bmatrix}-\dfrac{1}{l} & \dfrac{1}{l}\end{bmatrix},\mathbf{a}^e=\begin{bmatrix}u_\alpha^e\\ u_\beta^e\end{bmatrix} \tag{8.27}$$

刚度矩阵：

$$\mathbf{K}^e=\int_0^l\mathbf{B}^TEA\mathbf{B}dx=\frac{EA}{l}\begin{bmatrix}1 & -1\\ -1 & 1\end{bmatrix}=\begin{bmatrix}K_{11}^e & K_{12}^e\\ K_{21}^e & K_{22}^e\end{bmatrix} \tag{8.28}$$

质量矩阵：

$$\mathbf{M}^e=\int_0^l\mathbf{N}^T\rho A\mathbf{N}dx=\frac{\rho Al}{6}\begin{bmatrix}2 & 1\\ 1 & 2\end{bmatrix}=\begin{bmatrix}M_{11}^e & M_{12}^e\\ M_{21}^e & M_{22}^e\end{bmatrix} \tag{8.29}$$

忽略阻尼的影响，$\mathbf{C}$为零矩阵。单元长度相同，则每个单元的形函数、几何矩阵、刚度矩阵和质量矩阵都相同，我们统一用$\mathbf{K}^e$和$\mathbf{M}^e$来表示每个单元的单元刚度矩阵和单元质量矩阵。

对一端固支，一端受简谐力的杆，其边界条件为：

$$\begin{cases} u(0,t)=0 \\ -EA\dfrac{\partial u(L,t)}{\partial x}=F(t) \end{cases} \tag{8.30}$$

令单元载荷向量$\mathbf{Q}^e=\begin{bmatrix} q_\alpha^e \\ q_\beta^e \end{bmatrix}$，则$\mathbf{Q}^1=\mathbf{Q}^2=\cdots=\mathbf{Q}^n=\begin{bmatrix} 0 \\ 0 \end{bmatrix},\mathbf{Q}^{n+1}=\begin{bmatrix} 0 \\ F(t) \end{bmatrix}$。

将单元质量矩阵和单元刚度矩阵合成，得总体质量矩阵，总体刚度矩阵，总体载荷矩阵：

$$\mathbf{M}=\begin{bmatrix} M_{11}^e & M_{12}^e & & & & & \\ M_{21}^e & M_{22}^e+M_{11}^e & M_{12}^e & & & & \\ & M_{21}^e & M_{22}^e+M_{11}^e & \ddots & & & \\ & & & \ddots & \ddots & & \\ & & & & \ddots & & \\ & & & & & M_{22}^e+M_{11}^e & M_{12}^e \\ & & & & & M_{21}^e & M_{22}^e \end{bmatrix}_{n+1,n+1} \tag{8.31}$$

$$\mathbf{K}=\begin{bmatrix} K_{11}^e & K_{12}^e & & & & & \\ K_{21}^e & K_{22}^e+K_{11}^e & K_{12}^e & & & & \\ & K_{21}^e & K_{22}^e+K_{11}^e & \ddots & & & \\ & & & \ddots & \ddots & & \\ & & & & \ddots & & \\ & & & & & K_{22}^e+K_{11}^e & K_{12}^e \\ & & & & & K_{21}^e & K_{22}^e \end{bmatrix}_{n+1,n+1} \tag{8.32}$$

$$\mathbf{Q}=\begin{bmatrix} 0 & 0 & \cdots & 0 & F(t) \end{bmatrix}^T \tag{8.33}$$

$$\mathbf{a}=\begin{bmatrix} u_1 & u_2 & \cdots & u_n & u_{n+1} \end{bmatrix}^T \tag{8.34}$$

因为 1 号节点固支，所以令$u_1=0$。可将$\mathbf{M}$和$\mathbf{K}$的第一行和第一列划去，分别形成$\hat{\mathbf{M}}$和$\hat{\mathbf{K}}$。然后将$\mathbf{Q}$和$\mathbf{a}$的第一个元素除去，分别形成$\hat{\mathbf{Q}}$和$\hat{\mathbf{a}}$。

$$\hat{\mathbf{M}}=\begin{bmatrix} M_{22}^e+M_{11}^e & M_{12}^e & & & & \\ M_{21}^e & M_{22}^e+M_{11}^e & \ddots & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & & \\ & & & & M_{22}^e+M_{11}^e & M_{12}^e \\ & & & & M_{21}^e & M_{22}^e \end{bmatrix}_{n,n} \tag{8.35}$$

$$\hat{\mathbf{K}}=\begin{bmatrix} K_{22}^e+K_{11}^e & K_{12}^e & & & & \\ K_{21}^e & K_{22}^e+K_{11}^e & \ddots & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & & \\ & & & & K_{22}^e+K_{11}^e & K_{12}^e \\ & & & & K_{21}^e & K_{22}^e \end{bmatrix}_{n,n} \tag{8.36}$$

$$\hat{\mathbf{Q}}=\begin{bmatrix}0 & \cdots & 0 & F(t)\end{bmatrix}^T \tag{8.37}$$

$$\hat{\mathbf{a}}=\begin{bmatrix}u_2 & \cdots & u_n & u_{n+1}\end{bmatrix}^T \tag{8.38}$$

其中，$\hat{\mathbf{Q}}$ 和 $\hat{\mathbf{a}}$ 均为 $n\times 1$ 的列向量。

采用中心差分法计算，根据等式(8.23)，可得：

$$\hat{\mathbf{a}}_{t+\Delta t}=\hat{\mathbf{M}}^{-1}\hat{\mathbf{Q}}_t\Delta t^2-\left(\hat{\mathbf{M}}^{-1}\hat{\mathbf{K}}\Delta t^2-2\mathbf{I}\right)\cdot\hat{\mathbf{a}}_t-\hat{\mathbf{a}}_{t-\Delta t} \tag{8.39}$$

起步阶段，$\hat{\mathbf{a}}_0$，$\dot{\hat{\mathbf{a}}}_0$ 已知，则有：

$$\mathbf{a}_{-\Delta t}=\mathbf{a}_0-\Delta t\dot{\mathbf{a}}_0+\frac{\Delta t^2}{2}\ddot{\mathbf{a}}_0 \tag{8.40}$$

其中，

$$\ddot{\hat{\mathbf{a}}}_0=\mathbf{M}^{-1}\cdot\left(\mathbf{Q}_0-\mathbf{C}\cdot\dot{\hat{\mathbf{a}}}_0-\mathbf{K}\cdot\hat{\mathbf{a}}_0\right) \tag{8.41}$$

下面给出了计算程序。弹性杆的长度为 0.1 m，横截面积 $A=1\times10^{-4}$ mm^2，$E=70$ GPa，杆在长度方向划分为 8 个杆单元，时间步长 $\Delta t=1\times10^{-7}$ s。外载荷 $F(t)=a\sin(2\pi ft)$，其中 $a=1$ N, $f=1.2729\times10^4$ Hz。

由杆振动的解析解可知其固有频率：$\omega_n = \dfrac{n\pi}{2l}\sqrt{\dfrac{E}{\rho}}$，$n$ = 1, 2, 3, … n。其第一阶固有频率为1.2729×10^4 Hz，C++程序计算得到第八结点的位移响应如图 8.3 所示。其中左图是除了 1 号结点外其余节点的位移—时间响应，2 号结点到 9 号节点的振幅依次增大。从图中可以看出，由于外载荷频率等于固有频率，因此位移响应随时间逐步增大，表现了能量逐步累积的过程，并不是如解析解给出的振幅直接为无限大的。右图是不同时刻杆的位移响应，其分布形态与第一阶振型相似。

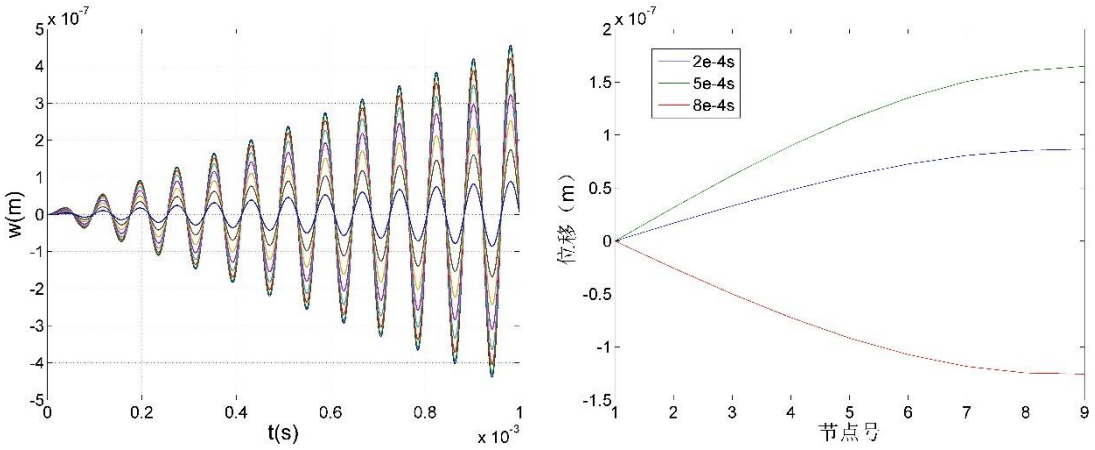


图 8.3 杆的振动响应曲线

8.4 弹性杆振动的 C++程序

其中 int mx_inver(double *mx_a,int N)函数的具体代码见 1.3.5 节。

```
#include <stdio.h>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cmath>
#include <cstdlib>

using namespace std;

#define pi 3.141592653589793
double fp(double tn);
int mx_inver(double *mx_a,int N);
```

```
//外载幅值，频率
double a=1,f=12729;
int main()
{
    FILE*file=NULL;
    file=fopen("w.txt","w");
    if(NULL==file)
    {
        cout<<"创建文件失败";
        return-1;//要返回错误代码
    }
    fclose(file);
    const int n=8;
    double ru=2700,A=1e-4,L=0.1,E=70e9;
    double l=L/n,t=0;
    double step=1e-7;//步长
    int total=10000;
    double M[n*n]={0};//总体质量
    double K[n][n]={0};//总体刚度
    double MK[n][n]={0};//中间量 M 逆*K
    double m[2][2]={0};//单元质量
    double k[2][2]={0};//单元刚度
    double *w=(double*)malloc(n*total*sizeof(double));//所有节点位移
    double F[n]={0};//载荷矩阵

    m[0][0]=ru*A*l/6*2;    m[0][1]=ru*A*l/6;  m[1][0]=m[0][1];   m[1][1]=m[0][0];

    k[0][0]=E*A/l;      k[0][1]=-E*A/l;     k[1][0]=k[0][1];     k[1][1]=k[0][0];

    int i,j,kk;
```

```
for (i=0;i<n-1;i++)
{
    K[i][i]+=k[0][0];        K[i][i+1]=k[0][1];        K[i+1][i]=k[1][0];
    K[i+1][i+1]+=k[1][1];

    M[i*n+i]+=m[0][0];        M[i*n+i+1]=m[0][1];        M[(i+1)*n+i]=m[1][0];
    M[(i+1)*n+i+1]+=m[1][1];

}
K[0][0]+=k[0][0];  M[0]+=m[0][0];
//M 求逆，结果仍在 M
mx_inver(M,n);
for(i=0;i<n;i++)
    for(j=0;j<n;j++)
        for(kk=0;kk<n;kk++)
            MK[i][j]+=M[i*n+kk]*K[kk][j];

for(i=0;i<n*total;i++)
    w[i]=0;
int jishu=0;
while(jishu<total)
{
    jishu++;        t=step*jishu;        F[n-1]=fp(t);
    if(jishu==1)
    {
        for(i=0;i<n;i++)
        {
            w[i*total+jishu]=0;
            for(j=0;j<n;j++)
            {
```

```
                        w[i*total+jishu]+=M[i*n+j]*F[j]*step*step-MK[i][j]*w[j*total+(jishu-
1)]*step*step;
                    }
                    w[i*total+jishu]+=2*w[i*total+jishu-1];
                }
            }
            else
            {
                for(i=0;i<n;i++)
                {
                    w[i*total+jishu]=0;
                    for(j=0;j<n;j++)
                    {
                        w[i*total+jishu]+=M[i*n+j]*F[j]*step*step-MK[i][j]*w[j*total+(jishu-
1)]*step*step;
                    }
                    w[i*total+jishu]+=2*w[i*total+jishu-1]-w[i*total+jishu-2];
                }
            }

        }//while
        fstream write;
        //位移
        write.open("w.txt",ios::out | ios::in    );
        if(!write)
        {
            cout<<"打开 w 文件失败";
            exit(0);
        }
        for(i=0;i<total;++i)
```

```
        {
            write.seekp(0L,ios::end);
            write<<i<<' '<<setprecision(10)<<w[(n-1)*total+i]<<'\n';
        }
        write.close();
        return 1;
    }
    //外载
    double fp(double tn)
    {
        return a*sin(2*pi*f*tn);
    }
```

8.5 有限元模态分析

连续体有限元形式的运动控制方程如式(8.17)所示。求解该式除了采用8.2节介绍的直接积分法，还可使用振型叠加法。振型叠加法可获得比直接积分法高的计算效率。该法首先进行模态计算，若忽略阻尼的影响，系统的自由振动方程可表示为：

$$\mathbf{M}\cdot\ddot{\mathbf{a}}+\mathbf{K}\cdot\mathbf{a}=\mathbf{0} \tag{8.42}$$

由振动理论相关知识可知，连续体自由振动时作固有频率下的周期运动，即：

$$\mathbf{a}=\boldsymbol{\phi}\sin\omega t \tag{8.43}$$

将式(8.43)代入式(8.42)中，可得到一个广义特征值问题，即：

$$\left(\mathbf{K}-\omega^2\mathbf{M}\right)\boldsymbol{\phi}=\mathbf{0} \tag{8.44}$$

求解上式即可得到各阶固有频率ω_i和各阶固有振型$\boldsymbol{\phi}_i$。振型叠加法基于模态分析结果，将位移向量从以节点位移为基向量的空间转换到以固有振型为基向量的空间，使得二阶常微分方程组成功解耦成独立的单自由度控制方程。振型叠加法首先将计算出的固有振型正则化，即使得：

$$\boldsymbol{\phi}_i^T\mathbf{M}\boldsymbol{\phi}_j=\begin{cases}1 & (i=j)\\ 0 & (i\neq j)\end{cases} \tag{8.45}$$

同时满足：

$$\boldsymbol{\phi}_i^T \mathbf{K} \boldsymbol{\phi}_j = \begin{cases} \omega_i^2 & (i=j) \\ 0 & (i \neq j) \end{cases} \tag{8.46}$$

为使阻尼矩阵能被解耦计算，常采用 Rayleigh 阻尼模型进行简化，即 $\mathbf{C}=\alpha\mathbf{M}+\beta\mathbf{K}$，其中的 α 和 β 是常系数。正则化的固有振型使得：

$$\boldsymbol{\phi}_i^T \mathbf{C} \boldsymbol{\phi}_j = \boldsymbol{\phi}_i^T \left(\alpha\mathbf{M}+\beta\mathbf{K}\right)\boldsymbol{\phi}_j = \begin{cases} 2\omega_i\zeta_i & (i=j) \\ 0 & (i \neq j) \end{cases} \tag{8.47}$$

其中，ω_i、$\boldsymbol{\phi}_i$、ζ_i 分别是第 i 阶的固有频率、固有振型向量、振型阻尼比。

进一步，以固有振型矩阵引入坐标变换：

$$\mathbf{a} = \boldsymbol{\Phi} \cdot \mathbf{x} \tag{8.48}$$

其中，$\boldsymbol{\Phi}$ 是由各阶振型向量组成的矩阵，$\mathbf{x}$ 是由各阶广义位移值组成的向量，即

$$\boldsymbol{\Phi} = \left[\boldsymbol{\phi}_1 \ \ \boldsymbol{\phi}_2 \ \ \cdots \ \ \boldsymbol{\phi}_n\right] \tag{8.49}$$

$$\mathbf{x} = \left\{\mathrm{x}_1 \ \ \mathrm{x}_2 \ \ \cdots \ \ \mathrm{x}_n\right\}^T \tag{8.50}$$

将式(8.48)代入控制方程（8.17）中，并在两端左乘 $\boldsymbol{\Phi}^{\mathrm{T}}$，结合式(8.45)式(8.47)，可得模态坐标下的运动方程

$$\ddot{\mathrm{x}}_i + 2\zeta_i\omega_i\dot{\mathrm{x}}_i + \omega_i^2\mathrm{x}_i = \boldsymbol{\phi}_i^T\mathbf{Q} \quad (i=1,2,\cdots,n) \tag{8.51}$$

以上就使控制方程得以解耦，将式(8.51)的各解回代式(8.48)即可得到总响应。以上是振型叠加法的求解步骤，是线性系统动力学响应的重要求解方法。

接下来我们以某航空发动机涡轮叶片为例来进行模态分析。如图 8.4 所示，首先将涡轮叶片划分成一系列的二十节点六面体单元。

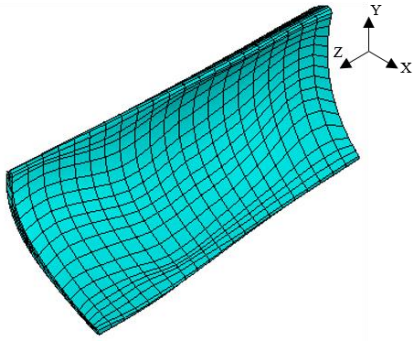


图 8.4 航空发动机涡轮叶片有限元模型图

对于二十节点六面体单元，单元示意图如图 8.5 所示，其中 ξ, η, γ 为单元的局部坐标系。

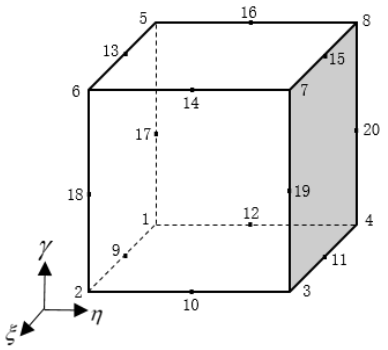


图 8.5 二十节点六面体单元示意图

则式(8.11)中的形函数可以表示为：

$$N_i(\xi,\eta,\gamma)=\frac{1}{4}(1-\xi^2)(1+\eta_i\eta)(1+\gamma_i\gamma)\quad(i=9,11,13,15)$$

$$N_i(\xi,\eta,\gamma)=\frac{1}{4}(1-\eta^2)(1+\xi_i\xi)(1+\gamma_i\gamma)\quad(i=10,12,14,16)$$

$$N_i(\xi,\eta,\gamma)=\frac{1}{4}(1-\gamma^2)(1+\eta_i\eta)(1+\xi_i\xi)\quad(i=17,18,19,20)$$

$$N_i(\xi,\eta,\gamma)=\frac{1}{8}(1+\xi_i\xi)(1+\eta_i\eta)(1+\gamma_i\gamma)(\xi_i\xi+\eta_i\eta+\gamma_i\gamma-2)\quad(i=1\sim 8)\tag{8.52}$$

形函数确定后，单元位移、单元坐标均可通过形函数用节点位移、节点坐标插值获得。式(8.12)中的单元几何矩阵 **B** 形式为 **B**={$\mathbf{B}_1\ \mathbf{B}_2\ \cdots\ \mathbf{B}_{20}$}，其中，

$$\mathbf{B}_i=\begin{bmatrix}\dfrac{\partial \mathrm{N}_i}{\partial x} & 0 & 0 & \dfrac{\partial \mathrm{N}_i}{\partial y} & 0 & \dfrac{\partial \mathrm{N}_i}{\partial z}\\ 0 & \dfrac{\partial \mathrm{N}_i}{\partial y} & 0 & \dfrac{\partial \mathrm{N}_i}{\partial x} & \dfrac{\partial \mathrm{N}_i}{\partial z} & 0\\ 0 & 0 & \dfrac{\partial \mathrm{N}_i}{\partial z} & 0 & \dfrac{\partial \mathrm{N}_i}{\partial y} & \dfrac{\partial \mathrm{N}_i}{\partial x}\end{bmatrix}^T\quad\left(i=1,2,\cdots,20\right)\tag{8.53}$$

求解上式时，需建立局部坐标系和总体坐标系之间的关系，即：

$$\begin{Bmatrix}\dfrac{\partial \mathrm{N}_i}{\partial \xi}\\ \dfrac{\partial \mathrm{N}_i}{\partial \eta}\\ \dfrac{\partial \mathrm{N}_i}{\partial \gamma}\end{Bmatrix}=\begin{bmatrix}\displaystyle\sum_{i=1}^{20}\frac{\partial \mathrm{N}_i}{\partial \xi}x_i & \displaystyle\sum_{i=1}^{20}\frac{\partial \mathrm{N}_i}{\partial \xi}y_i & \displaystyle\sum_{i=1}^{20}\frac{\partial \mathrm{N}_i}{\partial \xi}z_i\\ \displaystyle\sum_{i=1}^{20}\frac{\partial \mathrm{N}_i}{\partial \eta}x_i & \displaystyle\sum_{i=1}^{20}\frac{\partial \mathrm{N}_i}{\partial \eta}y_i & \displaystyle\sum_{i=1}^{20}\frac{\partial \mathrm{N}_i}{\partial \eta}z_i\\ \displaystyle\sum_{i=1}^{20}\frac{\partial \mathrm{N}_i}{\partial \gamma}x_i & \displaystyle\sum_{i=1}^{20}\frac{\partial \mathrm{N}_i}{\partial \gamma}y_i & \displaystyle\sum_{i=1}^{20}\frac{\partial \mathrm{N}_i}{\partial \gamma}z_i\end{bmatrix}\begin{Bmatrix}\dfrac{\partial \mathrm{N}_i}{\partial x}\\ \dfrac{\partial \mathrm{N}_i}{\partial y}\\ \dfrac{\partial \mathrm{N}_i}{\partial z}\end{Bmatrix}=\mathbf{J}\begin{Bmatrix}\dfrac{\partial \mathrm{N}_i}{\partial x}\\ \dfrac{\partial \mathrm{N}_i}{\partial y}\\ \dfrac{\partial \mathrm{N}_i}{\partial z}\end{Bmatrix}\tag{8.54}$$

上式中 **J** 是雅克比矩阵，将计算得到的雅克比矩阵代入上式即可确定单元几何矩阵。通过雅克比矩阵同样可以将总体坐标系下的微元体积 *dv* 转换到局部坐标系中，即：

$$dv=dxdydz=\left|\mathbf{J}\right|d\xi d\eta d\gamma\tag{8.55}$$

其中，|**J**|是雅克比矩阵的行列式。将上式代入式(8.19)中，并结合高斯积分公式即可确定有限元程序中的单元质量矩阵 $\mathbf{M}^e$ 和单元刚度矩阵 $\mathbf{K}^e$。进一步，通过抽取矩阵组装总体质量矩阵 **M** 和总体刚度矩阵 **K**。求解叶片固有频率，只需将 **M** 和 **K** 矩阵代入式(8.44)求解广义特征值问题即可。

下面给出了计算程序，该叶片密度是 7850 kg/m^3，E=200 GPa，有限元模型总计包含 1134 个单元，6041 个节点，叶片根部施加固支约束。设置求解前六阶，C++程序将结果输出到 txt 文件中，结果如图 8.6 所示。

阶数	频率(Hz)
第1阶	1586.9154
第2阶	2589.8132
第3阶	4834.21
第4阶	5311.0819
第5阶	6901.8572
第6阶	7444.6239

图 8.6 C++程序的输出结果

进一步采用 Workbench 软件进行模态分析，并与 C++程序的计算结果进行了对比，如图 8.7 所示。

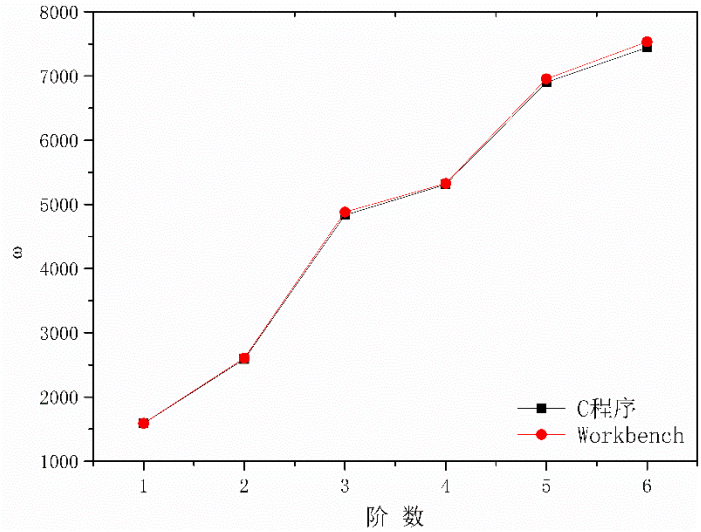


图 8.7 C++程序和 Workbench 计算结果对比图

由上图可知，C++程序与 Workbench 软件计算结果接近，误差在 5%以内。

8.6 叶片模态分析的 C++程序

```
#define EIGEN_USE_MKL_ALL
#define EIGEN_VECTORIZE_SSE4_2
#include <iostream>
#include <fstream>
#include <cassert>
#include <string>
#include <stdlib.h>
#include <Eigen/Dense>
#include <Eigen/Sparse>
#include <time.h>
#include <math.h>
#include <vector>
#include <iomanip>
#include "Eigen/SPQRSupport"
using namespace std;
using namespace Eigen;
char fileName[30];
double ru = 7850;//材料密度
double E = 2e11;//弹性模量
double v = 0.3;//泊松比
int num_elem = 1134;//单元数
int num_node = 6041;//节点数
int nodes = 0;///自由度
MatrixXi elem = MatrixXi::Zero(num_elem, 20);
MatrixXd node = MatrixXd::Zero(num_node, 9);
MatrixXi uvid = MatrixXi::Zero(num_node, 3);
//读取 elem 信息
void readfile_elem()
```

```
{
    ifstream infile;
    int i,j;
    infile.open("elem.txt");
    assert(infile.is_open());
    while (!infile.eof())
    {
        for (i = 0; i < elem.rows(); i++)
        {
            for (j = 0; j < elem.cols()/2; j++)
            {
                infile >> elem(i,j);
            }
        }
        for (i = 0; i < elem.rows(); i++)
        {
            for (j = elem.cols() / 2; j < elem.cols(); j++)
            {
                infile >> elem(i, j);
            }
        }
    }
    infile.close();
}
//读取 node 信息
void readfile_node()
{
    ifstream infile;
    int i, j;
    infile.open("node.txt");
```

```
    assert(infile.is_open());
    while (!infile.eof())
    {
        for (i = 0; i < node.rows(); i++)
        {
            for (j = 0; j < node.cols(); j++)
            {
                infile >> node(i, j);
            }
        }
    }
    infile.close();
    for (i = 0; i < num_node; i++)
    {
        for (j = 0; j < 3; j++)
        {
            node(i, j) /= 1000;///将单位由 mm 变成 m
        }
    }
}
//自由度衰减
void dof_reduce()
{
    for (int i = 0; i<node.rows(); i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (node(i, 3 + j) != 0)
            {
                uvid(i, j) = nodes;
```

```
                nodes++;
            }
            else
            {
                uvid(i, j) = -1;
            }
        }
    }
}
//求质量矩阵和刚度矩阵
void solve_MK(SparseMatrix < double >& M_sparse, SparseMatrix < double >& K_sparse)
{
    int i, j, m, n, o;
    int elment_id;
    double J[2] = { -0.577350269189626, 0.577350269189626};
    double H[2] = { 1, 1 };
    double X, Y, Z;
    double N_element[20] = { 0 };
    MatrixXd N = MatrixXd::Zero(3, 60);
    VectorXd position_x = VectorXd::Zero(20);////每个单元 20 个结点的 x 坐标
    VectorXd position_y = VectorXd::Zero(20);////每个单元 20 个结点的 y 坐标
    VectorXd position_z = VectorXd::Zero(20);////每个单元 20 个结点的 z 坐标
    VectorXd N_x = VectorXd::Zero(20);/////N 对 x 局部坐标求导
    VectorXd N_y = VectorXd::Zero(20);/////N 对 y 局部坐标求导
    VectorXd N_z = VectorXd::Zero(20);/////N 对 z 局部坐标求导
    Vector3d N_global = Vector3d::Zero();
    MatrixXd Jacques = MatrixXd::Zero(3, 3);
    double Jacques_det = 0;
    double sum_element = 0;
    double sum_trace = 0;
```

```
MatrixXd Me = MatrixXd::Zero(60, 60);
MatrixXd M = MatrixXd::Zero(nodes, nodes);
MatrixXd D = MatrixXd::Zero(6, 6);
MatrixXd Be0 = MatrixXd::Zero(6, 60);
MatrixXd Ke0 = MatrixXd::Zero(60, 60);
MatrixXd K0 = MatrixXd::Zero(nodes, nodes);
int id[60];
///弹性矩阵 D
double lmta = E*v / (1 + v) / (1 - 2 * v);
double G = E / 2 / (1 + v);
D(0, 0) = D(1, 1) = D(2, 2) = lmta + 2 * G;
D(0, 1) = D(1, 0) = D(0, 2) = D(2, 0) = D(1, 2) = D(2, 1) = lmta;
D(3, 3) = D(4, 4) = D(5, 5) = G;
////形成单元质量矩阵、单元刚度矩阵
for (elment_id = 0; elment_id < num_elem; elment_id++)
{
    Me.fill(0);
    Ke0.fill(0);
    ///单元 20 节点坐标向量
    for (i = 0; i < 20; i++)
    {
        position_x(i) = node(elem(elment_id, i) - 1, 0);
        position_y(i) = node(elem(elment_id, i) - 1, 1);
        position_z(i) = node(elem(elment_id, i) - 1, 2);
    }
    ///高斯积分
    for (m = 0; m < 2; m++)
    {
        for (n = 0; n < 2; n++)
        {
```

```
for (o = 0; o < 2; o++)
{
    X = J[m]; Y = J[n]; Z = J[o];
    /////求 N 矩阵
    N_element[0] = (1 - X)*(1 - Y)*(1 - Z)*(-X - Y - Z - 2) / 8;///N1
    N_element[1] = (1 + X)*(1 - Y)*(1 - Z)*(X - Y - Z - 2) / 8;///N2
    N_element[2] = (1 + X)*(1 + Y)*(1 - Z)*(X + Y - Z - 2) / 8;///N3
    N_element[3] = (1 - X)*(1 + Y)*(1 - Z)*(-X + Y - Z - 2) / 8;///N4
    N_element[4] = (1 - X)*(1 - Y)*(1 + Z)*(-X - Y + Z - 2) / 8;
    N_element[5] = (1 + X)*(1 - Y)*(1 + Z)*(X - Y + Z - 2) / 8;
    N_element[6] = (1 + X)*(1 + Y)*(1 + Z)*(X + Y + Z - 2) / 8;
    N_element[7] = (1 - X)*(1 + Y)*(1 + Z)*(-X + Y + Z - 2) / 8;
    N_element[8] = (1 - X*X)*(1 - Y)*(1 - Z) / 4;
    N_element[9] = (1 + X)*(1 - Y*Y)*(1 - Z) / 4;
    N_element[10] = (1 - X*X)*(1 + Y)*(1 - Z) / 4;
    N_element[11] = (1 - X)*(1 - Y*Y)*(1 - Z) / 4;
    N_element[12] = (1 - X*X)*(1 - Y)*(1 + Z) / 4;
    N_element[13] = (1 + X)*(1 - Y*Y)*(1 + Z) / 4;
    N_element[14] = (1 - X*X)*(1 + Y)*(1 + Z) / 4;
    N_element[15] = (1 - X)*(1 - Y*Y)*(1 + Z) / 4;
    N_element[16] = (1 - X)*(1 - Y)*(1 - Z*Z) / 4;
    N_element[17] = (1 + X)*(1 - Y)*(1 - Z*Z) / 4;
    N_element[18] = (1 + X)*(1 + Y)*(1 - Z*Z) / 4;
    N_element[19] = (1 - X)*(1 + Y)*(1 - Z*Z) / 4;
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 20; j++)
        {
            N(i, 3 * j + i) = N_element[j];
        }
```

```
}
//N 对局部坐标 X 求导
N_x(0) = (1 - Y)*(1 - Z)*(2 * X + Y + Z + 1) / 8;
N_x(1) = (1 - Y)*(1 - Z)*(2 * X - Y - Z - 1) / 8;
N_x(2) = (1 + Y)*(1 - Z)*(2 * X + Y - Z - 1) / 8;
N_x(3) = (1 + Y)*(1 - Z)*(2 * X - Y + Z + 1) / 8;
N_x(4) = (1 - Y)*(1 + Z)*(2 * X + Y - Z + 1) / 8;
N_x(5) = (1 - Y)*(1 + Z)*(2 * X - Y + Z - 1) / 8;
N_x(6) = (1 + Y)*(1 + Z)*(2 * X + Y + Z - 1) / 8;
N_x(7) = (1 + Y)*(1 + Z)*(2 * X - Y - Z + 1) / 8;
N_x(8) = -X*(1 - Y)*(1 - Z) / 2;
N_x(9) = (1 - Y*Y)*(1 - Z) / 4;
N_x(10) = -X*(1 + Y)*(1 - Z) / 2;
N_x(11) = -(1 - Y*Y)*(1 - Z) / 4;
N_x(12) = -X*(1 - Y)*(1 + Z) / 2;
N_x(13) = (1 - Y*Y)*(1 + Z) / 4;
N_x(14) = -X*(1 + Y)*(1 + Z) / 2;
N_x(15) = -(1 - Y*Y)*(1 + Z) / 4;
N_x(16) = -(1 - Y)*(1 - Z*Z) / 4;
N_x(17) = (1 - Y)*(1 - Z*Z) / 4;
N_x(18) = (1 + Y)*(1 - Z*Z) / 4;
N_x(19) = -(1 + Y)*(1 - Z*Z) / 4;
//N 对局部坐标 Y 求导
N_y(0) = (1 - X)*(1 - Z)*(X + 2 * Y + Z + 1) / 8;
N_y(1) = (1 + X)*(1 - Z)*(-X + 2 * Y + Z + 1) / 8;
N_y(2) = (1 + X)*(1 - Z)*(X + 2 * Y - Z - 1) / 8;
N_y(3) = (1 - X)*(1 - Z)*(-X + 2 * Y - Z - 1) / 8;
N_y(4) = (1 - X)*(1 + Z)*(X + 2 * Y - Z + 1) / 8;
N_y(5) = (1 + X)*(1 + Z)*(-X + 2 * Y - Z + 1) / 8;
N_y(6) = (1 + X)*(1 + Z)*(X + 2 * Y + Z - 1) / 8;
```

```
N_y(7) = (1 - X)*(1 + Z)*(-X + 2 * Y + Z - 1) / 8;
N_y(8) = -(1 - X*X)*(1 - Z) / 4;
N_y(9) = -2 * Y*(1 + X)*(1 - Z) / 4;
N_y(10) = (1 - X*X)*(1 - Z) / 4;
N_y(11) = -2 * Y*(1 - X)*(1 - Z) / 4;
N_y(12) = -(1 - X*X)*(1 + Z) / 4;
N_y(13) = -2 * Y*(1 + X)*(1 + Z) / 4;
N_y(14) = (1 - X*X)*(1 + Z) / 4;
N_y(15) = -2 * Y*(1 - X)*(1 + Z) / 4;
N_y(16) = -(1 - X)*(1 - Z*Z) / 4;
N_y(17) = -(1 + X)*(1 - Z*Z) / 4;
N_y(18) = (1 + X)*(1 - Z*Z) / 4;
N_y(19) = (1 - X)*(1 - Z*Z) / 4;
//N 对局部坐标 Z 求导
N_z(0) = (1 - X)*(1 - Y)*(X + Y + 2 * Z + 1) / 8;
N_z(1) = (1 + X)*(1 - Y)*(-X + Y + 2 * Z + 1) / 8;
N_z(2) = (1 + X)*(1 + Y)*(-X - Y + 2 * Z + 1) / 8;
N_z(3) = (1 - X)*(1 + Y)*(X - Y + 2 * Z + 1) / 8;
N_z(4) = (1 - X)*(1 - Y)*(-X - Y + 2 * Z - 1) / 8;
N_z(5) = (1 + X)*(1 - Y)*(X - Y + 2 * Z - 1) / 8;
N_z(6) = (1 + X)*(1 + Y)*(X + Y + 2 * Z - 1) / 8;
N_z(7) = (1 - X)*(1 + Y)*(-X + Y + 2 * Z - 1) / 8;
N_z(8) = -(1 - X*X)*(1 - Y) / 4;
N_z(9) = -(1 + X)*(1 - Y*Y) / 4;
N_z(10) = -(1 - X*X)*(1 + Y) / 4;
N_z(11) = -(1 - X)*(1 - Y*Y) / 4;
N_z(12) = (1 - X*X)*(1 - Y) / 4;
N_z(13) = (1 + X)*(1 - Y*Y) / 4;
N_z(14) = (1 - X*X)*(1 + Y) / 4;
N_z(15) = (1 - X)*(1 - Y*Y) / 4;
```

```
                    N_z(16) = -2 * Z*(1 - X)*(1 - Y) / 4;
                    N_z(17) = -2 * Z*(1 + X)*(1 - Y) / 4;
                    N_z(18) = -2 * Z*(1 + X)*(1 + Y) / 4;
                    N_z(19) = -2 * Z*(1 - X)*(1 + Y) / 4;
                    /////求雅克比矩阵
                    Jacques(0, 0) = N_x.dot(position_x); Jacques(1, 0) = N_y.dot(position_x);
Jacques(2, 0) = N_z.dot(position_x);
                    Jacques(0, 1) = N_x.dot(position_y); Jacques(1, 1) = N_y.dot(position_y);
Jacques(2, 1) = N_z.dot(position_y);
                    Jacques(0, 2) = N_x.dot(position_z); Jacques(1, 2) = N_y.dot(position_z);
Jacques(2, 2) = N_z.dot(position_z);
                    //////求雅克比矩阵的行列式
                    Jacques_det = Jacques.determinant();
                    //////求单元 B 矩阵
                    for (i = 0; i < 20; i++)
                    {
                        N_global = Jacques.inverse()*(Vector3d() << N_x(i), N_y(i),
N_z(i)).finished();
                        Be0(5, 3 * i + 2) = Be0(3, 3 * i + 1) = Be0(0, 3 * i) = N_global(0);
                        Be0(4, 3 * i + 2) = Be0(3, 3 * i) = Be0(1, 3 * i + 1) = N_global(1);
                        Be0(5, 3 * i) = Be0(4, 3 * i + 1) = Be0(2, 3 * i + 2) = N_global(2);
                    }
                    Me+= N.transpose() * N * ru * Jacques_det * H[m] * H[n] * H[o];///求单
元质量矩阵
                    Ke0 += Be0.transpose() * D * Be0 * Jacques_det * H[m] * H[n] * H[o];///
求单元刚度矩阵
                }
            }
        }
        sum_element = Me.sum();
```

```
sum_trace = Me.trace();
for (i = 0; i < 60; i++)
{
    for (j = 0; j < 60; j++)
    {
        if (i != j)
        {
            Me(i, j) = 0;
        }
        else
        {
            Me(i, j) = sum_element / sum_trace*Me(i, j);///集中质量矩阵
        }
    }
}
for (i = 0; i < 20; i++)
{
    for (j = 0; j < 3; j++)
    {
        id[3 * i + j] = uvid(elem(elment_id, i) - 1, j);
    }
}
for (i = 0; i < 60; i++)
{

    for (j = 0; j < 60; j++)
    {
        if (id[i] >= 0 && id[j] >= 0)
        {
            M(id[i], id[j]) += Me(i, j);///合成总体质量矩阵
```

```
                    K0(id[i], id[j]) += Ke0(i, j);///合成总体刚度矩阵
                }
            }
        }
    }
    M_sparse = M.sparseView();///稠密矩阵转成稀疏矩阵
    K_sparse = K0.sparseView();
}
//计算模态
void Eig(SparseMatrix < double >& K_sparse, SparseMatrix < double >& M_sparse,int& num_solve)
{
    int i, j;
    ofstream outfile;
    cout << "开始计算特征值" << endl;
    K_sparse.makeCompressed();
    MatrixXd X = MatrixXd::Random(nodes, 30);
    MatrixXd Y = MatrixXd::Zero(nodes, 30);
    Y = M_sparse*X;
    MatrixXd Xnew = MatrixXd::Zero(nodes, 30);
    MatrixXd Mnew = MatrixXd::Zero(30, 30);
    MatrixXd Knew = MatrixXd::Zero(30, 30);
    MatrixXd Knewni = MatrixXd::Zero(30, 30);
    MatrixXd DL = MatrixXd::Zero(30, 30);
    MatrixXd D = MatrixXd::Zero(30, 30);
    MatrixXd V = MatrixXd::Zero(30, 30);
    VectorXd Eigenvalue_new = VectorXd::Zero(30);
    VectorXd Eigenvalue_old = VectorXd::Zero(30);
    VectorXd Eigenvalue_precision = VectorXd::Zero(30);
    MatrixXd Eigenvector_initial = MatrixXd::Zero(30, 30);
    MatrixXd Eigenvector_later = MatrixXd::Zero(nodes, 30);
```

```
double judgment;
SimplicialLDLT<SparseMatrix<double>>      solver;
solver.compute(K_sparse);
while (1)///求特征值
{
    Xnew = solver.solve(Y);
    Mnew = Xnew.transpose()*M_sparse*Xnew;
    Knew = Xnew.transpose()*K_sparse*Xnew;
    Knewni = Knew.inverse();
    DL = Knewni*Mnew;
    EigenSolver<MatrixXd> es(DL);
    D = es.pseudoEigenvalueMatrix();
    V = es.pseudoEigenvectors();
    //////对特征值进行排序
    VectorXi ind = VectorXi::LinSpaced(D.rows(), 0, D.rows() - 1);//[0 1 2 3 ... N-1]
    auto rule = [D](int i, int j)->bool
    {
        return D(i, i)>D(j, j);///w 从小到大排序
    };
    std::sort(ind.data(), ind.data() + ind.size(), rule);
    for (i = 0; i<D.rows(); i++)
    {
        Eigenvalue_new(i) = 1 / sqrt(D(ind(i), ind(i)));
        for (j = 0; j < D.rows(); j++)
        {
            Eigenvector_initial(j, i) = V(j, ind(i));
        }
    }
    //设置判断条件
    for (i = 0; i < 30; i++)
```

```
            {
                Eigenvalue_precision(i) = fabs(Eigenvalue_new(i) - Eigenvalue_old(i)) /
Eigenvalue_new(i);
            }
            judgment = Eigenvalue_precision.maxCoeff();
            ///设置下一步的迭代条件
            Y = M_sparse * Xnew * Eigenvector_initial;
            Eigenvalue_old = Eigenvalue_new;
            if (judgment < pow(10, -2))break;
        }
        ///输出特征值
        for (i = 0; i < num_solve; i++)
        {
            sprintf(fileName, "Eigenvalue.txt");
            outfile.open(fileName, ios::app);
            outfile <<"第"<< i + 1 <<"阶"<< "\t" << std::setprecision(8) << Eigenvalue_old(i) / 2 /
3.1415926 << "Hz"<<endl;
            outfile.close();
        }
    }
    //主函数
    int main(int argc, char *argv[])
    {
        mkl_set_dynamic(2);//指定计算使用 CPU 数量
        int num_solve;///求解的阶数
        cout << "请输入求解的阶数：" << endl;
        cin >> num_solve;
        cout << endl<<"计算开始" << endl;
        ///读取数据
        cout << "读取数据" << endl;
```

```
    readfile_elem();
    readfile_node();
    dof_reduce();
    SparseMatrix < double > M_sparse(nodes,nodes);///定义质量矩阵
    SparseMatrix < double > K_sparse(nodes,nodes);///定义刚度矩阵
    solve_MK(M_sparse,    K_sparse);
    Eig(K_sparse, M_sparse, num_solve);
    cout << "计算结束" << endl;
    return nodes;
}
```

第九章 结构优化设计的有限元实现

9.1 结构优化设计的基本概念

在结构设计过程中，设计师通常会问一个问题，在保证功能和可靠性的条件下，关键尺寸取多少才能使该结构的重量最轻，或者用量最省。这就属于结构优化设计的范畴。为了加深理解，我们再来看一个例子。

如图 9.1 所示是一个矩形水池，水池的长度和宽度分别用 l 和 w 来表示，深度为 1m。采用铁皮作为水池围挡。假设铁皮厚度一定，由于资金有限，围成水池的铁皮总长度是一定的。这意味着水池的周长是一定的。如果水池的总周长为 24m，那么问 l 和 w 分别取多少时水池的面积最大。

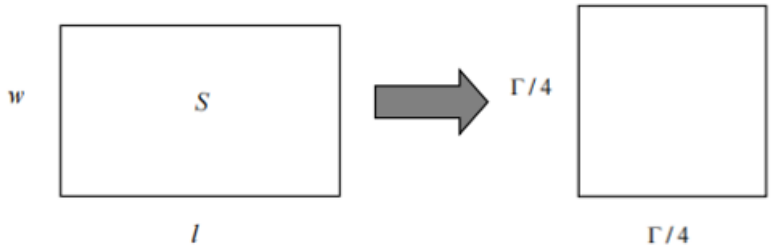


图 9.1 水池的结构拓扑优化设计

假设水池的面积为 S，周长为Γ，根据简单的几何关系可得$S=l\cdot w$，$\Gamma=2(l+w)$。我们利用周长公式，在 S 的表达式中消去 l 后得$S=\frac{\Gamma}{2}w-w^2$。根据极值条件$\frac{\partial S}{\partial w}=\frac{\Gamma}{2}-2w=0$，得$w=\frac{\Gamma}{4}$。由此可见，当水池的长和宽相等时，围成的面积最大。

上述例子尽管简单，但是包含了优化设计的所有要素。首先我们的目标是希望面积最大，因此面积 S 是我们优化的对象，称为目标函数。由于我们要通过调整水池的长和宽来改变面积，因此 l 和 w 是我们要设计的，称为设计变量。另外 l 和 w 并不能够能随便取任意值，还需要满足周长的限制，因此周长方程就称为约束条件。由此我们可以总结出优化设计的三要素：目标函数，设计变量，约束条件。在任何一个优化问题中，上述三要素缺一不可。

我们再来看一个例子。如果图 w 中水池的形状不限于矩形，而是可以用正多边形来设计，那么变长多大的时候 S 最大呢。如图 9.2 所示，假设用正 n 边形来建造水池，那么水池的边长$l=\frac{\Gamma}{n}$，水池的面积$S=\frac{l^2\cdot n}{4\tan\left(\frac{360}{2n}\right)}=\frac{\Gamma^2}{4\cdot n\cdot\tan\left(\frac{360}{2n}\right)}$。我们画出 S 随 n 变化的曲线如图 9.3 所示。可见随着 n 增大 S 也增大。极端情况下，当$n\to\infty$时 S 最大。因此在周长一定的条件下，圆形水池的面积最大。

该算例与图 9.1 所示算例不同之处在于，该算例是以结构的形状作为设计变量，而图 9.1 所示算例是以结构的尺寸作为设计变量。

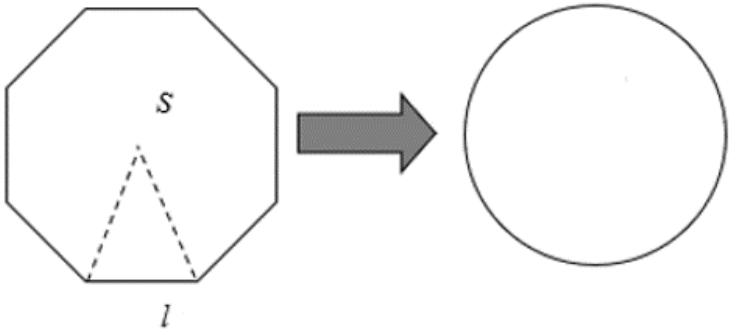


图 9.2 水池的形状优化设计

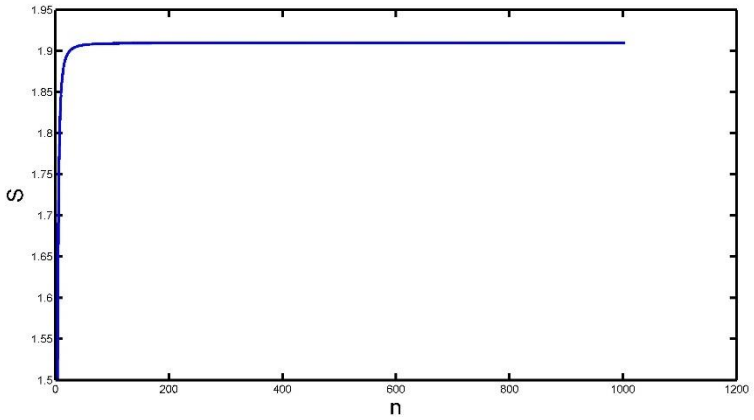


图 9.3 水池面积函数 S

因此，尺寸优化和形状优化是结构优化的两种基本问题。事实上，还有一类更为复杂的优化设计问题，称为拓扑优化。该类型问题通过改变研究对象区域内材料的有无和分布来得到拓扑类型，如图 9.4 所示典型的长悬臂梁问题。其中 L=2H，梁的右端面中点受到竖直方向的集中力作用。

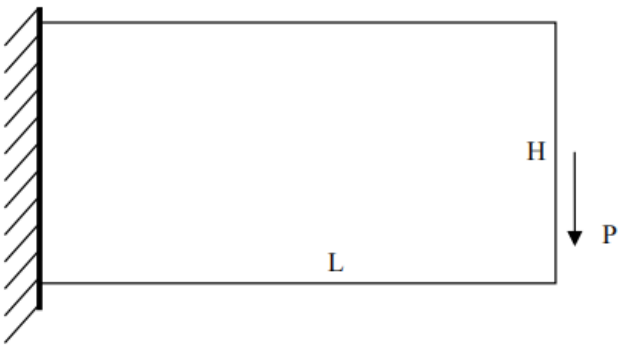


图 9.4 长臂梁的拓扑优化问题

问题是，保持载荷 P 大小不变，如何设计矩形梁内部的拓扑结构使得梁的重量最轻，并且保证作用点的位移不超过规定值。这是一个最经典的拓扑优化问题。研究者采用不同的优化方法得到了

如图所示的集中拓扑结构。可见拓扑结构已经超越了尺寸优化和形状优化，真正实现了结构从无到有的创新设计。

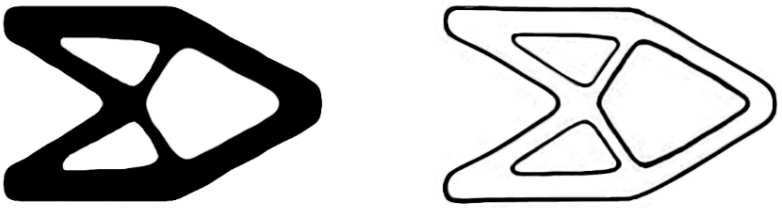


图 9.5 长臂梁的拓扑优化结果

总结起来，结构优化设计的类型主要包括三类：尺寸优化、形状优化和拓扑优化。目标函数、设计变量和约束条件是实现结构优化的三要素。一个结构优化问题可以概括为如下数学问题。

设目标函数 W 和约束条件 g_j（j=1～m）是设计变量 x_i（i=1～n）的函数。结构优化问题表述为寻找一组最佳的设计变量 x_i^*（i=1～n），使得 $W\left(x_1^*, x_2^*, \cdots, x_i^*, \cdots x_n^*\right)$ 取最小值（或最大值），并且设计变量还同时满足约束条件 $g_j\left(x_1^*, x_2^*, \cdots, x_i^*, \cdots x_n^*\right) = 0$（j=1～m）。有时约束条件采用不等式来表示 $g_j\left(x_1^*, x_2^*, \cdots, x_i^*, \cdots x_n^*\right) \le 0$。

获得最佳设计 $\left(x_1^*, x_2^*, \cdots, x_i^*, \cdots x_n^*\right)$ 的过程就是寻优的过程。我们采用优化算法来获得最优解。有关优化算法的详细介绍，读者可以参考各类结构优化的文献，本教材仅介绍最常用的适用范围较广的“遗传算法”。以此为例子介绍如何应用有限元法实现结构的优化设计。

9.2 遗传算法

遗传算法（GA）是由美国学者 John H. Holland 及其学生在 20 世纪 60 年代末至 70 年代初提出的一种智能优化方法。其基本思想来源于自然界中“物竞天择，适者生存”这一生物进化法则。

生物在其延续生存的过程中，逐渐适应其生存环境，使其品质不断得到改良，这种生命现象称为进化。生物的进化是以集团的形式共同进行的，这样的一个集团称为种群，组成种群的单个生物称为个体，每个个体对其生存环境都有不同的适应能力，称为适应度。按照达尔文进化论，那些具有较强适应度的个体具有更高的生存能力，容易存活下来，并有较多的机会产生后代。而那些适应度较低的个体则被淘汰，产生后代的机会较少。通过进化过程，种群的平均适应度越来越高，物种越来越优良。

生物进化的本质体现在染色体的改变上。有性生物在繁殖下一代时，两个同源染色体之间通过

交叉而重组，即两个染色体的某一相同位置处的染色体被切断，其前后两串分别交叉组合而形成两个相同的染色体。另外，在进行复制时，可能以很小的概率产生某些差错，即染色体会产生变异。

遗传算法借鉴了生物进化这一过程，根据问题的目标函数构造一个适值函数，对一个由多个解（每个解对应一个个体）构成的种群进行评估、选择和遗传运算，经过多代繁殖，获得适值最好的个体作为问题的最优解。

9.2.1 构成要素

（1）种群和种群大小

种群是个体的集合，每个个体就是一个染色体，每个染色体对应着问题的一个解。种群中个体的数量称为种群大小或种群规模。种群大小通常是恒定的，并且一般种群规模越大越好。但种群规模的增大将导致运算时间的增大。

（2）编码方案

在遗传算法中，种群的每个个体即染色体对应着最优化问题的一个解，染色体由基因组成。染色体与问题的解之间的对应关系即是染色体的编码方案。正确地对染色体进行编码来表示问题的解是遗传算法的基础工作。

（3）适值函数

适值函数与优化问题的目标函数并不完全一致，但适值函数是根据目标函数进行设计的。

（4）遗传算子

遗传算子包括交叉和变异两种。遗传算子模拟了每一代生物繁殖后代的过程，是遗传算法的精髓。

交叉是对两个染色体进行操作，组合二者的特性产生新的后代。交叉的最简单方式是在双亲的染色体上随机地选择一个断点，将断点的右端相互交换，从而形成两个新的后代。遗传算法的性能在很大程度上取决于采用的交叉运算的性能。双亲的染色体是否进行交叉由交叉率来控制。交叉率定义为各代中进行交叉操作的个体数与种群中总的个体数之比。

变异是在染色体上自发地产生随机的变化。最简单的变异方式是在染色体上替换一个或多个基因。在遗传算法中，变异可以提供初始种群中不含有的基因，或者找到在进化过程中丢失的基因。染色体是否进行变异由变异率来控制。变异率定义为种群中产生变异的个体数与种群中总的个体数之比。

（5）选择策略

选择策略是从当前种群中选择适应值高的个体以生成交配池的过程。使用最多的是正比选择策

略。选择过程体现了生物进化过程中的“物竞天择，适者生存”的思想。

（6）停止准则

一般使用最大迭代次数作为停止准则。

9.2.2 算法流程

遗传算法一般过程的流程图如图 9.6 所示：

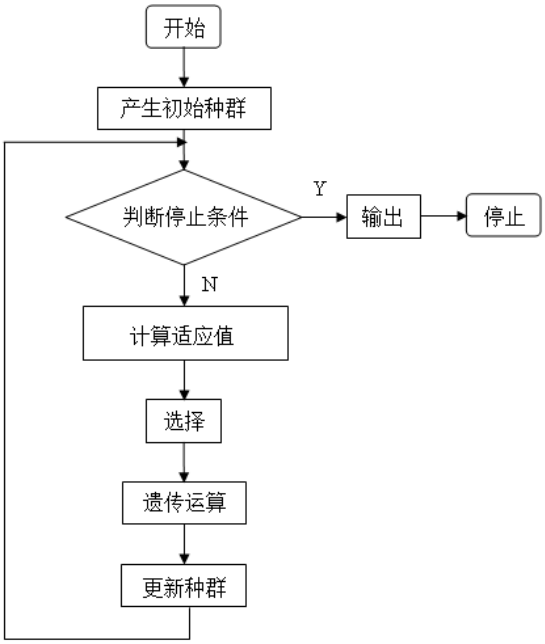


图 9.6 遗传算法流程图

由图 9.6 可以看到遗传算法实现过程中的要素包括：（1）编码方案；（2）初始种群的产生；（3）选择策略；（4）适值函数；（5）遗传运算；（6）停止准则。

（1）编码方案

本文采用二进制编码，染色体长度为 7，即用长度为 7 的 0-1 字符串表示一个染色体，例如：

X=(0110010)

就可以表示一个染色体。二进制编码的优点是便于进行位值计算，且包括的实数范围大。

（2）初始种群的产生

初始种群的产生一般是随机的。本文采用二进制编码方法，染色体长度为 7，种群中个体数为 N。因此，本文产生初始种群的方法是利用计算机产生 N 个长度为 7 的 0-1 随机字符串。这样就产生了初始种群。

（3）选择策略

本文采用的选择策略是正比选择，即每个个体被选中进行遗传运算的概率为该个体的适应值和

种群中所有个体适应值总和比值。对于个体 i, 设其适应值为 F_i，种群大小为 N，则该个体被选择的概率可以表示为：

$$P_i = \frac{F_i}{\sum_{j=1}^{N} F_j} \tag{9.1}$$

（4）适值函数

前文已说明，适值函数是根据目标函数来设计的。本文的优化目标是锥盘体积最小化，但采用的选择策略是正比选择，即个体适应值越大则被选中的概率也越大。因此适值函数与目标函数间应呈负相关。本文设计的适值函数为：

$$F(x) = M - volume(x) \tag{9.2}$$

其中 M 为一很大的正数。

（5）遗传运算

①交叉

本文采用的交叉方案是单切点交叉。从种群中选出两个个体 P_1 和 P_2，随机选择一个切点，将切点两侧分别看作两个子串，将右侧的子串交换，则得到两个新的个体 C_1 和 C_2,如图 9.7 所示：

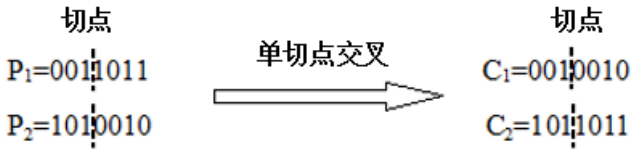


图 9.7 单切点交叉示意图

②变异

变异是在种群中按照变异概率任选若干基因位改变其位值。由于本文采用的是二进制编码方式，变异操作就是将基因位的值进行 0-1 反转。

（6）停止准则

本文采用的停止准则是设定最大迭代次数即种群最大代数。

9.2.3 约束的处理

对于有约束条件的优化问题，遗传算法优化过程中可能会产生不在可行域中的个体。对此，遗传算法有如下几种处理约束的方法：

（1）拒绝策略

拒绝策略是抛弃进化过程中产生的所有不可行解。这种方法简单但效率低。

（2）修复策略

修复策略是在进化过程中获得不可行解后，将其修复为可行解。对于很多组合优化问题，创建修复过程相对容易，但是可能导致失去种群多样性。

（3）惩罚策略

惩罚策略是对约束进行处理的最一般的方式，是通过对不可行解的惩罚来将约束问题转化为无约束问题。任何对于约束的违反都要在目标函数中添加惩罚项，这就要设计惩罚函数。

（4）特殊的编码和遗传策略

也可以使用特殊的编码策略，在编码时就充分考虑约束问题，在编码时产生的都是符合约束的染色体。为了使染色体在遗传操作后仍然保持可行性，也要使用特殊的遗传策略，使遗传后染色体仍然保持可行。

本文对约束的处理采用惩罚策略，定义适值函数为

$$F(x)=[M-volume(x)]P(x) \tag{9.3}$$

与公式（9.2）比较可发现，只是将原适值函数乘以了 P(x)，P(x)称为罚函数

本文中 P(x)定义如下：

$$P(x)=\begin{cases}1 & \sigma_{max}(x)\le[\sigma] \text{ and } U_{max}(x)\le[U] \\ 0.1 & \sigma_{max}(x)\ge[\sigma] \text{ or } U_{max}(x)\ge[U]\end{cases} \tag{9.4}$$

9.3 形状优化的程序实现

9.3.1 计算流程

本节将借助有限元计算软件 ANSYS 以及本教程介绍的遗传算法实现回转盘的形状优化。回转盘是工程机械中一种比较常见的零件类型，如图 9.8（左图）所示是某种回转盘的三维模型。它由图 9.8（右图）的截面围绕旋转轴旋转 360 度形成。

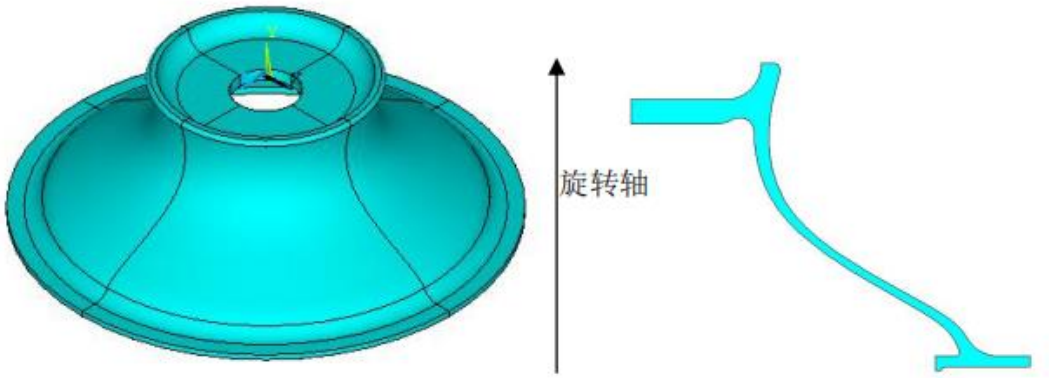


图 9.8 回转盘的三维模型（左图）和截面模型（右图）

回转盘的材料参数如表 9.1~9.4 所示：

表 9.1 回转盘的弹性模量

θ°C	400	500	650
E（GPa）	170.5	166	155

表 9.2 回转盘的泊松比

θ°C	20	300	400	500	600
λ	0.3	0.3	0.31	0.32	0.32

表 9.3 回转盘的线膨胀系数

θ°C	100	200	300	400	500	600
α，10^{-6}/°C	11.8	13.0	13.5	14.1	14.4	14.8

表 9.4 回转盘的屈服应力

θ°C	400	500	650
$\sigma_{0.2}$（MPa）	932.52	875.06	700.26

由于回转盘模型是三维的，为了使网格划分比较规整，先对锥盘的剖面进行网格划分，采用 SOLID95 单元。锥盘实体模型被划分为 7518 个单元，共 23220 个结点。回转盘受到的载荷参数为：（1）轴向力：F=2.8807×10³N；（2）扭矩：T=8.0540×103N·M；（3）温度分布：$T=\frac{600-400}{R_b-R_0}(R-R_0)+400℃$；（4）转速：ω=400rad/s。

其中轴向力以均布力施加在图 9.9 所示端面 A 处。扭矩以节点力的形式施加在图 9.9 左图 B 所指的外缘处。力的方向与圆周相切，如图 9.9 右图所示。

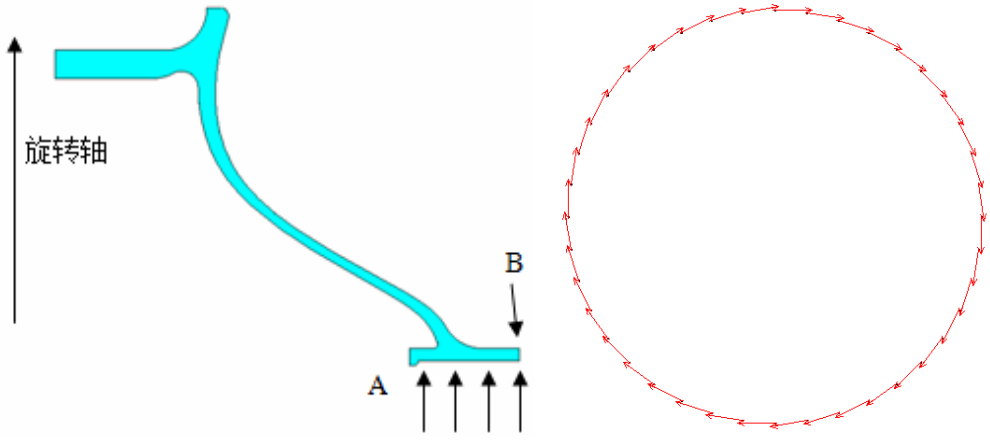


图 9.9 回转盘的受力

回转盘的温度分布采用公式 $T=\frac{600-400}{R_b-R_0}(R-R_0)+400℃$ 计算，其中 R 是回转盘中任意一点到旋转轴的距离，R_0 是回转盘内半径，R_b 是回转盘外半径。由于已知模型内部温度分布，模型内

各结点的温度值即可求出。本文利用 APDL 语言中的循环语句将每个结点赋上其温度值，即可求出锥盘内的热应力分布情况。APDL 是 ansys 的参数化设计语言，每条语句对应 ansys 的一个命令，详细内容见 ansys 帮助文件。为了便于进行受力分析和尽量模拟实际情况，本文采取的约束方法是：建模时将回转盘与一段轴建为一体，在轴的另一端施加固支约束，其二维模型如图 9.10 所示。

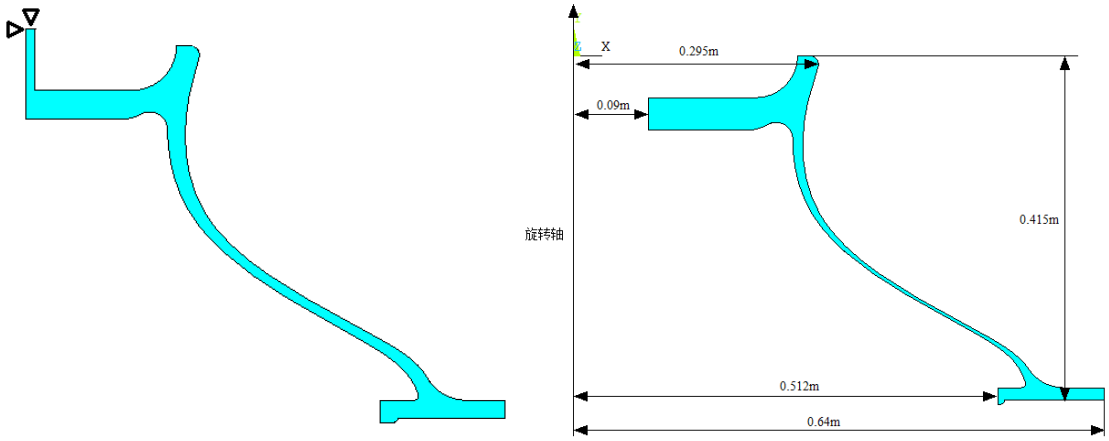


图 9.10 约束和主要尺寸示意图

结合锥盘材料的屈服极限和实际情况，本文对锥盘强度和刚度的要求是：

最大应力：$\sigma_{\max} \le 650MPa$；最大位移：$U_{\max} \le 8.0mm$

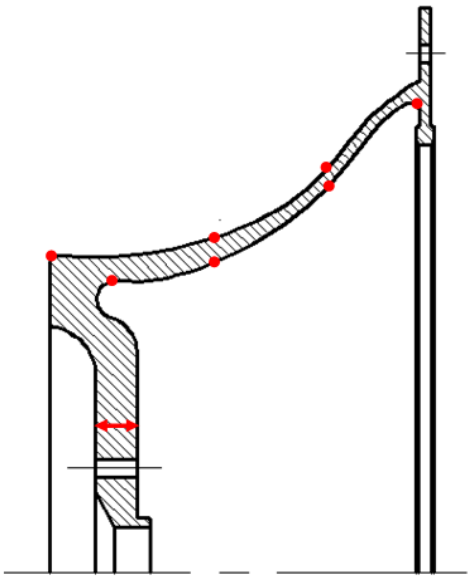


图 9.11 设计变量含义

共有 8 个设计变量，其含义如图 9.11 所示：其中前 7 个设计变量（如图中红点所示）用于表征两个样条曲线的形状，定义为每个点到旋转轴的距离。第 8 个设计变量（如图中箭头所示）用于表征锥盘前端的厚度。样条锥盘的结构优化问题的数学模型为：

$$
\begin{aligned}
&\min \quad f(x)=\text{volume}(x)\\
&\text{s.t.} \quad -\sigma_{\max}(x) \geq -[\sigma]\\
&\qquad -U_{\max}(x) \geq -[U]\\
&\qquad x_k^L \leq x_k \leq x_k^U, \quad k=1,2\cdots\cdots 8\\
&\qquad x_k \in R
\end{aligned}
\tag{9.6}
$$

因为质量最轻即体积最小，所以把锥盘的体积作为目标函数。约束条件是：锥盘的最大应力小于最大允许应力（强度条件），锥盘内部的最大位移小于最大允许位移（刚度条件）以及设计变量的取值要在一定的范围内。

我们采用遗传算法寻找方程(9.6)的最优解，计算流程如图 9.12 所示。在优化算法的每个迭代步中调用 ANSYS 程序实现有限元分析。

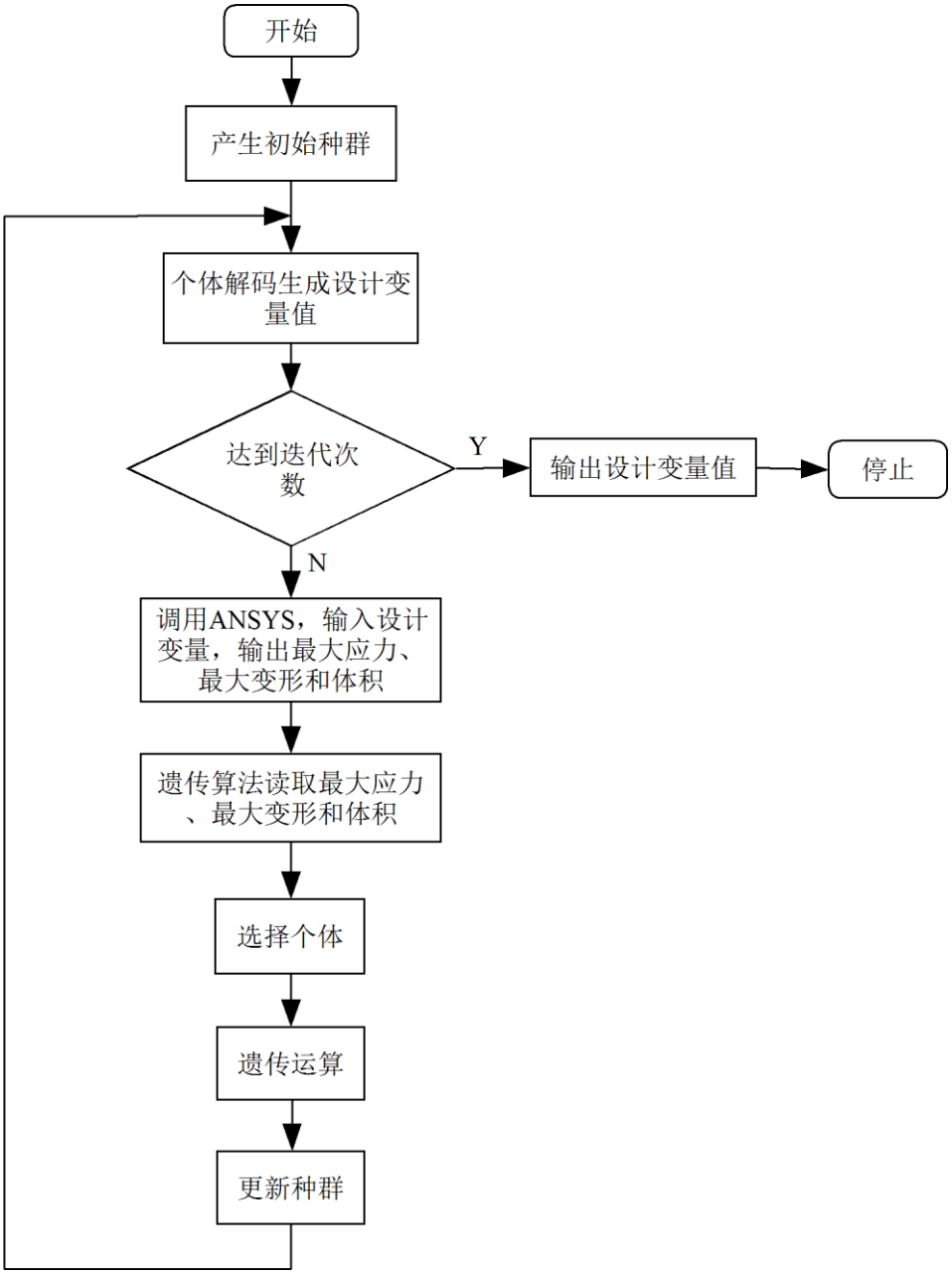


图 9.12 遗传算法调用 ansys 实现回转盘优化的流程图

本文中，遗传算法是用 C 语言编写的，使用函数“system(″ANSYSXXX.exe -b -p ane3fl -i huizhuanpan.txt -o output.out″)”调用函数。函数中的 XXX 表示版本号，如果是 11.0 的版本则 XXX 用 110 代替。在调用该函数之前，C 语言程序将设计变量数值写入 ZHI.txt。运行 system(″ANSYSXXX.exe -b -p ane3fl -i huizhuanpan.txt -o output.out″)命令后，ansys 将自动读取 huizhuanpan.txt 作为输入文件，并且按照 huizhuanpan.txt 中记录的命令进行建模、计算并输出结果。注意，在 huizhuanpan.txt 中 ansys 读取 ZHI.txt 获得设计变量的数值。system 函数中指定的 output.out 文件只记录了 ansys 运行的状态。ansys 运行过程会产生 MAXEQV.txt，用来保存当前结构的最大应力；MAXU.txt，用来保存当前结构的最大变形；VOLUME.txt，用来保存当前结构的体积；然后 C 语

言编写的遗传算法读取 MAXEQV.txt、MAXU.txt 和 VOLUME.txt 文件中的计算结果，然后再根据此结果作为下一步计算的依据。C 语言编写的遗传算法还会产生 currentbestfitness.txt 用来保存每一步迭代的结果（包括设计变量值和种群最大适应值）。另外，C 语言编写的程序还会每隔 5 迭代步将当前最佳设计的设计变量保存在 ZHI2.txt 中，然后使用命令 system(″ANSYS110.exe -b -p ane3fl -i ImageOutput.txt -o output.out″)；产生一幅当前锥盘形状的图片。

C 语言编写的计算程序附后。

目标函数值（锥盘与短轴体积之和，下同）随迭代次数增加的变化情况如表 9.5 和图 9.12 所示。其体积（不含短轴）为：0.02498m3，最大应力为：633.17MPa，最大位移为：7.0mm，满足最大约束条件。最终的几何形状如图 9.13 所示。

表 9.5 遗传算法目标函数值随迭代次数增加的变化情况

代数	第 1 次目标函数值	第 2 次目标函数值	第 3 次目标函数值
1	0.03510	0.04177	0.05455
2	0.03423	0.04083	0.05395
3	0.03423	0.03581	0.04466
4	0.03393	0.03457	0.04442
5	0.03393	0.03457	0.04259
6	0.03321	0.03441	0.04259
7	0.03321	0.03418	0.04259
8	0.03321	0.03418	0.04019
9	0.03321	0.03418	0.04019
10	0.03321	0.03418	0.04019
11	0.03321	0.03418	0.03851
12	0.03321	0.03142	0.03807
13	0.03321	0.03066	0.03807
14	0.03321	0.03066	0.03807
15	0.03281	0.03066	0.03791
16	0.03281	0.03039	0.03791
17	0.03281	0.02934	0.03791

18	0.03281	0.02934	0.03785
19	0.03281	0.02934	0.03581
20	0.03281	0.02934	0.03581
21	0.03281	0.02934	0.03581
22	0.03281	0.02843	0.03479
23	0.03281	0.02843	0.03479
24	0.03240	0.02843	0.03479
25	0.03240	0.02843	0.03479
26	0.03236	0.02843	0.03479
27	0.03236	0.02843	0.03479
28	0.03231	0.02843	0.03303
29	0.03009	0.02843	0.03185
30	0.03009	0.02843	0.03062

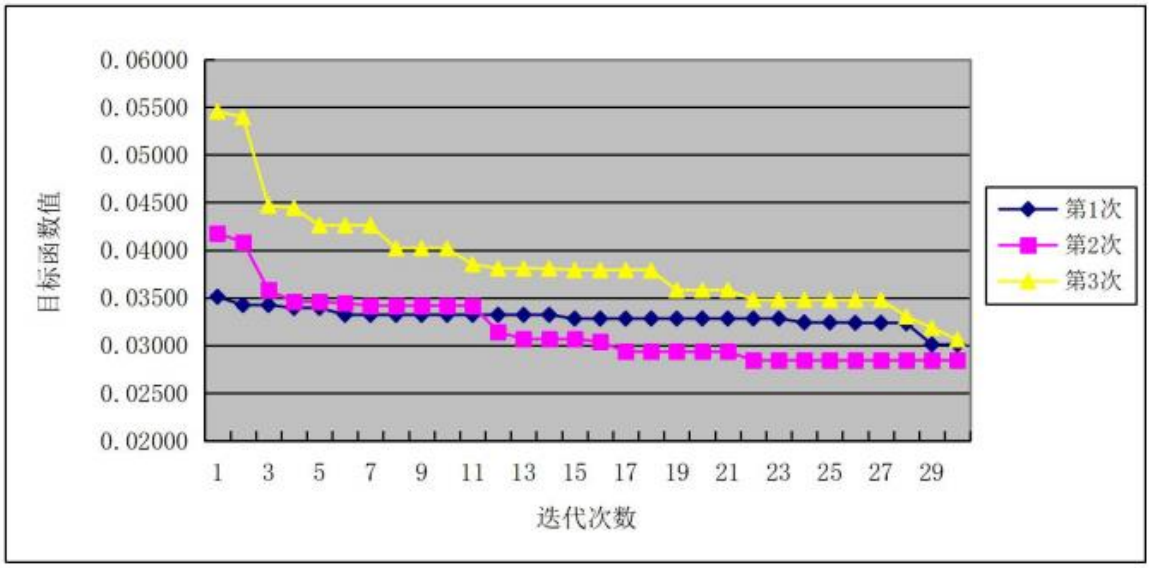


图 9.12 遗传算法目标函数值随迭代次数增加的变化情况

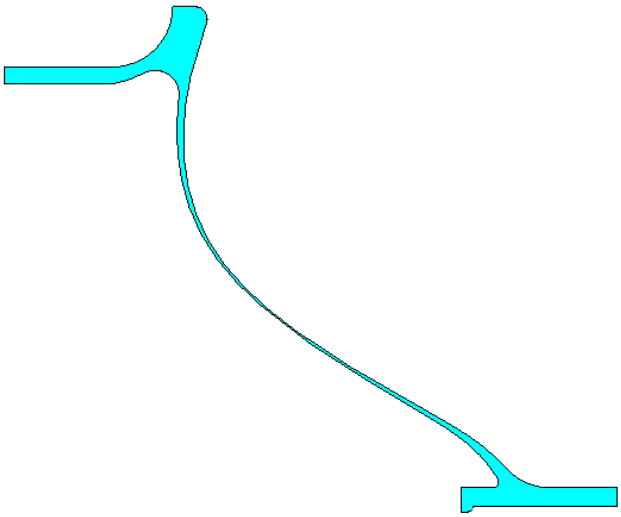


图 9.13 优化后的回转盘二维模型

9.3.2 C 语言代码

```
#include<iostream>
#include<fstream>
#include<stdlib.h>
#include<ctime>
#include<math.h>
#include<process.h>
using namespace std;
#define PopSize 10                  //群体规模
#define NVARS 8                     //变量个数
#define LENGTH 7
#define CHROMLENGTH NVARS*LENGTH
#define MaxGeneration 2000          //运行的最大代数
double Pc=0.5;                      //杂交概率
double Pm=0.15;                     //变异概率
struct individual
{
    char chrom[CHROMLENGTH+1];
    double value;
```

```
        double fitness;
        double yueshu1;
        double yueshu2;
};
int generation;
int best_index;
int worst_index;
struct individual bestindividual;
struct individual worstindividual;
struct individual currentbest;
struct individual population[PopSize];
void GenerateInitialPopulation(void);
void GenerateNextPopulation(int generation);
void EvaluatePopulation(void);
double DecodeChromosome(char*,int,int);
void CalculateObjectValue(void);
void FindBestAndWorstIndividual(void);
void SelectionOperator(void);
void CrossoverOperator(int generation);          //交叉操作
void MutationOperator(int generation);        //变异操作
int random(int n);
void display(void);
void main()
{
     int chushi;
       chushi=0;
       cout<<"Solving..."<<endl;
       time_t time0 = time(0);
     char time1[64];
     strftime( time1, sizeof(time1), "%Y/%m/%d %X",localtime(&time0) );
```

```
  fstream out,out1;
  double temp[NVARS],x[NVARS];
out.open("currentbestfitness.txt",ios::out);
  cout<<time1<<endl;
  out<<time1<<endl;
  generation=0;
GenerateInitialPopulation();
EvaluatePopulation();
  while(generation<MaxGeneration)
  {
       generation++;
     GenerateNextPopulation(generation);
     EvaluatePopulation();
       for(int k=0;k<NVARS;k++)
       {
            temp[k]=DecodeChromosome(currentbest.chrom,k*LENGTH,LENGTH);
       }
       x[0]=(temp[0]+300)/1000;//AA
       x[1]=(temp[1]+300)/1000;//BB
       x[2]=(temp[2]+400)/1000;//CC
       x[3]=0.56-(temp[3]+5)/1000;//EE
       x[4]=x[2]-(temp[4]+5)/1000;//FF
       x[5]=x[1]-(temp[5]+5)/1000;//GG
       x[6]=x[0]-(temp[6]+5)/1000;//HH
       x[7]=temp[7]/3000;//DD

cout<<generation<<','<<x[0]<<','<<x[1]<<','<<x[2]<<','<<x[3]<<','<<x[4]<<','<<x[5]<<','<<x[6]<<','<<x[7
]<<','<<endl<<currentbest.fitness<<endl;

out<<generation<<','<<x[0]<<','<<x[1]<<','<<x[2]<<','<<x[3]<<','<<x[4]<<','<<x[5]<<','<<x[6]<<','<<x[7]
```

```
<<','<<endl<<currentbest.fitness<<endl;
                //
    if(((chushi==0)||(generation%5==1))&&(currentbest.fitness>0.0011))
                //if((generation%5==1)&&(currentbest.fitness>0.002))
                {
                    chushi=1;
                    out1.open("ZHI2.txt",ios::out);
    out1<<"AA="<<x[0]<<endl<<"BB="<<x[1]<<endl<<"CC="<<x[2]<<endl<<"EE="<<x[3]<<endl<<
"FF="<<x[4]<<endl<<"GG="<<x[5]<<endl<<"HH="<<x[6]<<endl<<"DD="<<x[7];
                    out1.close();
                    system("ANSYS110.exe -b -p ane3fl -i ImageOutput.txt -o output.out");
                }
                //
            }
            time0 = time(0);
           strftime( time1, sizeof(time1), "%Y/%m/%d %X",localtime(&time0) );
            cout<<time1<<endl;
            out<<time1<<endl;
            out.close();
            cout<<"Solution done..."<<endl;
    }
    void GenerateInitialPopulation(void)
    {
        int i,j;
        for(i=0;i<PopSize;i++)
        {
            for(j=0;j<CHROMLENGTH;j++)
            {
                population[i].chrom[j]=(rand()%10<5?'0':'1');
            }
```

```
            population[i].chrom[CHROMLENGTH]='\0';
        }
    }
    void GenerateNextPopulation(int generation)
    {
        SelectionOperator();
        CrossoverOperator(generation);
        MutationOperator(generation);
    }
    void EvaluatePopulation(void)
    {
        CalculateObjectValue();
        FindBestAndWorstIndividual();
    }
    double DecodeChromosome(char *string,int point, int length)
    {
        int i;
        double decimal=0,temp;
        char *pointer;
        temp=1;
        for(i=0,pointer=string+point+length-1;i<length;i++,pointer--)
        {
            decimal+=(*pointer-'0')*temp;
            temp*=2;
        }
        return(decimal);
    }
    void CalculateObjectValue(void)
    {
        fstream in,out;
```

```
int i,k;
double temp[NVARS];
double x[NVARS],s,p;
for(i=0;i<PopSize;i++)
{
    for(k=0;k<NVARS;k++)
    {
    temp[k]=DecodeChromosome(population[i].chrom,k*LENGTH,LENGTH);
    }
    x[0]=(temp[0]+300)/1000;//AA
    x[1]=(temp[1]+300)/1000;//BB
    x[2]=(temp[2]+400)/1000;//CC
    x[3]=0.56-(temp[3]+5)/1000;//EE
    x[4]=x[2]-(temp[4]+5)/1000;//FF
    x[5]=x[1]-(temp[5]+5)/1000;//GG
    x[6]=x[0]-(temp[6]+5)/1000;//HH
    x[7]=temp[7]/3000;//DD
    out.open("ZHI.txt",ios::out);
out<<"AA="<<x[0]<<endl<<"BB="<<x[1]<<endl<<"CC="<<x[2]<<endl<<"EE="<<x[3]<<endl<<"
FF="<<x[4]<<endl<<"GG="<<x[5]<<endl<<"HH="<<x[6]<<endl<<"DD="<<x[7];
    out.close();
   out.open("VOLUME.txt",ios::out);
    out<<0.099;
    out.close();
    out.open("MAXEQV.txt",ios::out);
    out<<1.0;
    out.close();
    out.open("MAXU.txt",ios::out);
    out<<0.01;
    out.close();
```

```
            system("ANSYS110.exe -b -p ane3fl -i huizhuanpan.txt -o output.out");
            in.open("VOLUME.txt",ios::_Nocreate);
            in>>s;
            in.close();
            s=1.0-s-0.9;
            in.open("MAXEQV.txt",ios::_Nocreate);
            in>>population[i].yueshu1;
            in.close();
            in.open("MAXU.txt",ios::_Nocreate);
            in>>population[i].yueshu2;
            in.close();
p=((4.0e9>population[i].yueshu1)&&(0.02>population[i].yueshu2))?1:0.1;
            s=s*p;

            population[i].fitness=s;
        }
}
void FindBestAndWorstIndividual(void)
{
    int i;
    bestindividual=population[0];
    worstindividual=population[0];
    for(i=1;i<PopSize;i++)
    {
        if(population[i].fitness>bestindividual.fitness)
        {
            bestindividual=population[i];
            best_index=i;
        }
        else
```

```
            if(population[i].fitness<worstindividual.fitness)
            {
                worstindividual=population[i];
               worst_index=i;
            }
        }
        if (generation==0)
        {
            currentbest=bestindividual;
        }
        else
        {
            if(bestindividual.fitness>currentbest.fitness)
            {
                currentbest=bestindividual;
            }
        }
}
void SelectionOperator(void)
{
        int i,index;
        double p,sum=0.0;
        double cfitness[PopSize];//cumulative fitness value
        struct individual newpopulation[PopSize];
        for(i=0;i<PopSize;i++)
        {
            sum+=population[i].fitness;
        }
        for(i=0;i<PopSize;i++)
        {
```

```
        cfitness[i]=population[i].fitness/sum;
    }
    for(i=1;i<PopSize;i++)
    {
        cfitness[i]=cfitness[i-1]+cfitness[i];
    }
    for(i=0;i<PopSize;i++)
    {
        p=rand()%1000/1000.0;
        index=0;
        while(p>cfitness[index])
        {
            index++;
        }
        newpopulation[i]=population[index];
    }
    for(i=0;i<PopSize;i++)
    {
        population[i]=newpopulation[i];
    }
}
void CrossoverOperator(int generation)
{
    int i,j;
    int index[PopSize];
    int point,temp;
    double p;
    char ch;
    for(i=0;i<PopSize;i++)
    {
```

```
        index[i]=i;
    }
    for(i=0;i<PopSize;i++)
    {
        point=random(PopSize-i);
        temp=index[i];
        index[i]=index[point+i];
        index[point+i]=temp;
    }
    if(generation%2==1)
        for(i=0;i<PopSize-1;i+=2)
        {
            p=rand()%1000/1000.0;
            if(p<Pc)
            {
                point=random(CHROMLENGTH-1)+1;
                for (j=point;j<CHROMLENGTH;j++)
                {
                    ch=population[index[i]].chrom[j];
population[index[i]].chrom[j]=population[index[i+1]].chrom[j];
                    population[index[i+1]].chrom[j]=ch;
                }
            }
        }
    else
        for(i=0;i<PopSize;i++)
        {
            p=rand()%1000/1000.0;
            if(p<Pc)
            {
```

```
                point=random(CHROMLENGTH-1)+1;
                for (j=point;j<CHROMLENGTH;j++)
                {
                    population[i].chrom[j]=currentbest.chrom[j];
                }
            }
        }

}
void MutationOperator(int generation)
{
    int i,j;
    double p;
    if(generation%2==1)
        for(i=0;i<PopSize;i++)
        {
            for(j=0;j<CHROMLENGTH;j++)
            {
                p=rand()%1000/1000.0;
                if(p<Pm)
                {
    population[i].chrom[j]=(population[i].chrom[j]=='0')?'1':'0';
                }
            }
        }
    else
        for(i=0;i<PopSize;i++)
        {
            j=0;
            while(population[i].chrom[j]==currentbest.chrom[j])
```

```
                j++;
            for(;j<CHROMLENGTH;j++)
            {
                p=rand()%1000/1000.0;
                if(p<Pm)
                {
                    population[i].chrom[j]=currentbest.chrom[j];
                }
            }
        }
}
int random(int n)
{
    return rand()%n;
}
void display()
{
        for(int i=0;i<PopSize;i++)
        {
            for(int j=0;j<CHROMLENGTH;j++)
            {
                cout<<population[i].chrom[j]<<" ";
            }
            cout<<endl;
        }
}
```

9.3.3 输入文件：huizhuanpan.txt

```
FINISH
/INPUT,ZHI,TXT
```

```
/PREP7
ET,1,MESH200
KEYOPT,1,1,7
ET,2,21
KEYOPT,2,3,0
R,2,1E-6
ET,3,SOLID95
MP,EX,1,2.0E11
MP,PRXY,1,0.3
MP,DENS,1,8200
MP,ALPX,1,1.40E-5
MP,PRXY,2,0.3

K,1,AA,0
K,2,BB,-0.20
K,3,CC,-0.35
K,4,0.56,-0.40
k,5,0.60,-0.40
K,6,0.64,-0.40
K,7,0.64,-0.415
k,8,0.60,-0.415
K,9,0.52,-0.415
K,10,0.515,-0.42
K,11,EE-0.035,-0.42
K,12,EE-0.035,-0.4
K,13,EE,-0.4
K,14,FF,-0.35
K,15,GG,-0.20
K,16,HH,0
K,17,0.22,-0.05-DD
```

K,18,0.15,-0.05-DD

K,19,0.13,-0.05-DD

K,20,0.11,-0.05-DD

K,21,0.09,-0.05-DD

K,22,0.09,-0.05

K,23,0.13,-0.05

K,24,0.22,-0.05

K,25,0.27,0

K,26,0.22,0.01

K,27,0,0

K,28,0,-0.45

BSPLIN,1,2,3,4

L,4,6

L,6,7

L,7,9

LARC,9,10,12,0.005

L,10,11

L,11,12

L,12,13

BSPLIN,13,14,15,16

L,17,18

L,18,20

L,20,21

L,21,22

L,22,24

LARC,24,25,26,0.05

LARC,16,17,24,0.1

LFILLT,10,16,0.05

```
LCOMB,16,17
LSBL,9,16,,DELETE,KEEP
LSBL,16,17,,DELETE,KEEP
LDELE,18,18,,1
LDELE,9,9,,1
LFILLT,17,19,0.02
L,1,25
LFILLT,1,16,0.01
LFILLT,8,17,0.005
LFILLT,1,2,0.05
NUMMRG,ALL
AL,ALL
L,5,8
L,19,23

MSHAPE,1,2D
ESIZE,0.01
MSHKEY,0
AMESH,ALL

TYPE,3
EXTOPT,ESIZE,5,0
EXTOPT,ACLEAR,0
ALLSEL,ALL
VROTAT,ALL,,,,,,27,28,,

*GET,NMIN,ELEM,,NUM,MIN
*GET,NMAX,ELEM,,NUM,MAX
FVOLU=0
*DO,K,NMIN,NMAX,1
```

```
*GET,EVOLU,ELEM,K,VOLU
FVOLU=FVOLU+EVOLU
*ENDDO

KSEL,S,,,28
TYPE,2
REAL,2
KMESH,ALL
ALLSEL

NSEL,S,LOC,Y,-0.415,-0.415
CSYS,5
NSEL,R,LOC,X,0.60,0.64
CSYS,0
NSEL,A,LOC,Y,-0.45,-0.45
NPLOT
CERIG,NODE(0,-0.45,0),ALL,ALL,,,,
ALLSEL
FINISH
/SOLU

F,NODE(0,-0.45,0),MY,100E3

OMEGA,0,600,0,0

CSYS,5
ALLSEL,ALL
*GET,NMIN,NODE,,NUM,MIN
*GET,NMAX,NODE,,NUM,MAX
*DO,I,NMIN,NMAX,1
```

```
TEMP_I=(500-200)/(0.64-0.09)*(NX(I)-0.09)+200
BF,I,TEMP,TEMP_I
*ENDDO
CSYS,0

SFA,4,,PRES,40E6
SFA,70,,PRES,40E6
SFA,26,,PRES,40E6
SFA,48,,PRES,40E6

SFA,5,,PRES,10E6
SFA,71,,PRES,10E6
SFA,27,,PRES,10E6
SFA,49,,PRES,10E6

NSEL,S,LOC,Y,-0.05,-0.05
CSYS,5
NSEL,R,LOC,X,0.11,0.15
CSYS,0
D,ALL,ALL


ALLSEL
SOLVE
FINISH


/POST1
ALLSEL
NSORT,S,EQV,0,0,ALL
```

```
*GET,MAX_EQV,SORT,0,MAX
ALLSEL
NSORT,U,SUM,0,0,ALL
*GET,MAX_U,SORT,0,MAX
FINISH

*CFOPEN,MAXEQV,TXT
*VWRITE,MAX_EQV
%15.2F
*CFCLOS

*CFOPEN,MAXU,TXT
*VWRITE,MAX_U
%10.5F
*CFCLOS

*CFOPEN,VOLUME,TXT
*VWRITE,FVOLU
%10.5F
*CFCLOS
/EXIT
```

9.3.4 输入文件：ImageOutput.txt

```
FINISH
/INPUT,ZHI2,TXT
/PREP7
ET,1,MESH200
KEYOPT,1,1,7
ET,2,21
KEYOPT,2,3,0
```

R,2,1E-6

ET,3,SOLID95

MP,EX,1,2.0E11

MP,PRXY,1,0.3

MP,DENS,1,8200

MP,ALPX,1,1.40E-5

MP,PRXY,2,0.3

K,1,AA,0

K,2,BB,-0.20

K,3,CC,-0.35

K,4,0.56,-0.40

k,5,0.60,-0.40

K,6,0.64,-0.40

K,7,0.64,-0.415

k,8,0.60,-0.415

K,9,0.52,-0.415

K,10,0.515,-0.42

K,11,EE-0.035,-0.42

K,12,EE-0.035,-0.4

K,13,EE,-0.4

K,14,FF,-0.35

K,15,GG,-0.20

K,16,HH,0

K,17,0.22,-0.05-DD

K,18,0.15,-0.05-DD

K,19,0.13,-0.05-DD

K,20,0.11,-0.05-DD

K,21,0.09,-0.05-DD

K,22,0.09,-0.05

K,23,0.13,-0.05

K,24,0.22,-0.05

K,25,0.27,0

K,26,0.22,0.01

K,27,0,0

K,28,0,-0.45

BSPLIN,1,2,3,4

L,4,6

L,6,7

L,7,9

LARC,9,10,12,0.005

L,10,11

L,11,12

L,12,13

BSPLIN,13,14,15,16

L,17,18

L,18,20

L,20,21

L,21,22

L,22,24

LARC,24,25,26,0.05

LARC,16,17,24,0.1

LFILLT,10,16,0.05

LCOMB,16,17

LSBL,9,16,,DELETE,KEEP

LSBL,16,17,,DELETE,KEEP

LDELE,18,18,,1

LDELE,9,9,,1

```
LFILLT,17,19,0.02
L,1,25
LFILLT,1,16,0.01
LFILLT,8,17,0.005
LFILLT,1,2,0.05
NUMMRG,ALL
AL,ALL
L,5,8
L,19,23

APLOT
/RGB,INDEX,100,100,100,0
/RGB,INDEX,80,80,80,13
/RGB,INDEX,60,60,60,14
/RGB,INDEX,0,0,0,15
/REPLOT

/SHOW,JPEG
APLOT
/SHOW,CLOSE
/EXIT
}
```

致谢

本书由江苏省研究生教育教学改革研究与实践项目“研究生机械强度课程体系研究型教学模式的改革与实践”（项目编号：JGZZ13_011）与南京航空航天大学本科专业建设项目“有限元基础研究性课程”（项目编号：1402ZJ01XX05）资助出版。

参考文献

[1] 刘鸿文，材料力学（第四版），高等教育出版社，2004 年
[2] 王国安，材料力学，机械工业出版社，2015 年
[3] 徐芝伦，弹性力学简明教程，高等教育出版社，2002 年
[4] 王光钦，弹性力学（第三版），清华大学出版社，2015 年
[5] 王勖成，有限单元法，清华大学出版社，2003 年
[6] 洛根，有限元方法基础教程（第五版），电子工业出版社，2014 年
[7] 关玉璞、陈伟、崔海涛，航空航天结构有限元法，哈尔滨工业大学出版社，2009 年
[8] 陈雪峰，有限元方法及其工程案例，科学出版社，2014 年
[9] 钱令希，工程结构优化设计，科学出版社，2011 年
[10] 余建星，工程结构可靠原理及其优化设计，中国建筑工业出版社，2013 年
[11] 龚曙光、谢桂兰、黄云清编著，ANSYS 参数化编程与命令手册，机械工业出版社，2009 年
[12] 宋鹏，ANSYS 有限元分析从入门到精通，机械工业出版社，2015 年
[13] 谭浩强，中国高等院校计算机基础教育课程体系规划教材：C 程序设计（第 4 版），清华大学出版社，2010 年
[14] 吉顺如，c 语言程序设计教程（第三版），机械工业出版社，2015 年

附录 A 材料非线性有限元的 C 语言程序

```
#include"stdio.h"
#include"stdlib.h"
#include"math.h"

double E, mu, cs, Et;//E为杨氏模量, mu为泊松比, cs为材料非线性问题的屈服强度,Et为材料非线性问题的切向模量
int NumElem, NumNode;//NumElem为单元数, NumNode为节点总数
int* Elem;
double* Node;
double* UAll;//UAll为所有节点的位移增量
double DltU[24] = { 0 };//DltU为材料非线性问题中上一载荷步产生的位移增量

//读取并校核文件
void nonl_readfile_3d()
{
    int i, j;
    char filename[32];
    //读取文件
    FILE* fp;
    printf("请输入要打开的文件名\n提示: 数据文件名为: data06.txt\n");
    scanf("%s", filename);
    fp = fopen(filename, "r");
    if (fp == NULL)
    {
        printf("当前目录下无此文件\n");
        system("pause");
        exit(1);
    }
    else
    {
        printf("打开成功\n");
    }
    fscanf(fp, "%lf%lf%lf%lf", &E, &mu, &cs, &Et);
    fscanf(fp, "%d%d", &NumElem, &NumNode);
    Elem = (int*)malloc(sizeof(int) * 8 * NumElem);
    if (Elem == NULL)
    {
        exit(-2);
    }
    Node = (double*)malloc(sizeof(double) * 9 * NumNode);
    if (Node == NULL)
```

```
    {
        exit(-2);
    }
    for (i = 0; i < NumElem; i++)
    {
        for (j = 0; j < 8; j++)
        {
            fscanf(fp, "%d", &Elem[8 * i + j]);
        }
    }
    for (i = 0; i < NumNode; i++)
    {
        for (j = 0; j < 9; j++)
        {
            fscanf(fp, "%lf", &Node[9 * i + j]);
        }
    }
    fclose(fp);
    //输出到文件
    FILE* fc;
    fc = fopen("check06.txt", "w");
    fprintf(fc, "%lf\t%lf\t%lf\t%lf\n", E, mu, cs, Et);
    fprintf(fc, "%d\t%d\n", NumElem, NumNode);
    for (i = 0; i < NumElem; i++)
    {
        for (j = 0; j < 8; j++)
        {
            fprintf(fc, "%d\t", Elem[8 * i + j]);
        }
        fprintf(fc, "\n");
    }
    for (i = 0; i < NumNode; i++)
    {
        for (j = 0; j < 9; j++)
        {
            fprintf(fc, "%lf\t", Node[9 * i + j]);
        }
        fprintf(fc, "\n");
    }
    fclose(fc);
}

//计算不同载荷步下、不同单元的单元刚度矩阵、单元载荷矩阵
void nonl_elem_severity_3d(double(*Ke)[24], double* Load, double* Stress, int ElemId, int
```

```
StepId)
{
    int m, n, p, i, j, k;
    double J[2] = { -0.5773502692,0.5773502692 };
    double H[2] = { 1,1 };
    double Jc[9] = { 0 };//Jc为三维的雅可比矩阵
    //计算Ke
    for (i = 0; i < 24; i++)
    {
        for (j = 0; j < 24; j++)
        {
            Ke[i][j] = 0;
        }
    }
    for (m = 0; m < 2; m++)
    {
        for (n = 0; n < 2; n++)
        {
            for (p = 0; p < 2; p++)
            {
                double De[6][6] = { 0 }, Dp[6][6] = { 0 }, B[6][24] = { 0 }, Bt[24][6] =
{ 0 }, DB[6][24] = { 0 }, BDB[24][24] = { 0 };
                double x[8] = { 0 }, y[8] = { 0 }, z[8] = { 0 }; double X, Y, Z, Jh;
                X = J[m]; Y = J[n]; Z = J[p];
                //计算De
                De[5][5] = De[4][4] = De[3][3] = E / (2 * (1 + mu));
                De[1][0] = De[1][2] = De[2][0] = De[2][1] = De[0][2] = De[0][1] = E * mu /
((1 + mu) * (1 - 2 * mu));
                De[1][1] = De[2][2] = De[0][0] = De[1][0] + 2 * De[3][3];
                for (i = 0; i < 8; i++)
                {
                    x[i] = Node[9 * (Elem[8 * ElemId + i] - 1)];
                }
                for (i = 0; i < 8; i++)
                {
                    y[i] = Node[9 * (Elem[8 * ElemId + i] - 1) + 1];
                }
                for (i = 0; i < 8; i++)
                {
                    z[i] = Node[9 * (Elem[8 * ElemId + i] - 1) + 2];
                }
                double Na1, Na2, Na3, Na4, Na5, Na6, Na7, Na8, Nb1, Nb2, Nb3, Nb4, Nb5, Nb6,
Nb7, Nb8, Nc1, Nc2, Nc3, Nc4, Nc5, Nc6, Nc7, Nc8;
                double xa, xb, xc, ya, yb, yc, za, zb, zc;
```

```
                Na1 = -0.125 * (1 - Y) * (1 - Z); Na2 = -Na1; Na3 = 0.125 * (1 + Y) * (1 -
Z); Na4 = -Na3; Na5 = -0.125 * (1 - Y) * (1 + Z); Na6 = -Na5; Na7 = 0.125 * (1 + Y) * (1 + Z);
Na8 = -Na7;
                Nb1 = -0.125 * (1 - X) * (1 - Z); Nb2 = -0.125 * (1 + X) * (1 - Z); Nb3 = -
Nb2; Nb4 = -Nb1; Nb5 = -0.125 * (1 - X) * (1 + Z); Nb6 = -0.125 * (1 + X) * (1 + Z); Nb7 = -
Nb6; Nb8 = -Nb5;
                Nc1 = -0.125 * (1 - X) * (1 - Y); Nc2 = -0.125 * (1 + X) * (1 - Y); Nc3 = -
0.125 * (1 + X) * (1 + Y); Nc4 = -0.125 * (1 - X) * (1 + Y); Nc5 = -Nc1; Nc6 = -Nc2; Nc7 = -
Nc3; Nc8 = -Nc4;
                xa = Na1 * x[0] + Na2 * x[1] + Na3 * x[2] + Na4 * x[3] + Na5 * x[4] + Na6 *
x[5] + Na7 * x[6] + Na8 * x[7];
                xb = Nb1 * x[0] + Nb2 * x[1] + Nb3 * x[2] + Nb4 * x[3] + Nb5 * x[4] + Nb6 *
x[5] + Nb7 * x[6] + Nb8 * x[7];
                xc = Nc1 * x[0] + Nc2 * x[1] + Nc3 * x[2] + Nc4 * x[3] + Nc5 * x[4] + Nc6 *
x[5] + Nc7 * x[6] + Nc8 * x[7];
                ya = Na1 * y[0] + Na2 * y[1] + Na3 * y[2] + Na4 * y[3] + Na5 * y[4] + Na6 *
y[5] + Na7 * y[6] + Na8 * y[7];
                yb = Nb1 * y[0] + Nb2 * y[1] + Nb3 * y[2] + Nb4 * y[3] + Nb5 * y[4] + Nb6 *
y[5] + Nb7 * y[6] + Nb8 * y[7];
                yc = Nc1 * y[0] + Nc2 * y[1] + Nc3 * y[2] + Nc4 * y[3] + Nc5 * y[4] + Nc6 *
y[5] + Nc7 * y[6] + Nc8 * y[7];
                za = Na1 * z[0] + Na2 * z[1] + Na3 * z[2] + Na4 * z[3] + Na5 * z[4] + Na6 *
z[5] + Na7 * z[6] + Na8 * z[7];
                zb = Nb1 * z[0] + Nb2 * z[1] + Nb3 * z[2] + Nb4 * z[3] + Nb5 * z[4] + Nb6 *
z[5] + Nb7 * z[6] + Nb8 * z[7];
                zc = Nc1 * z[0] + Nc2 * z[1] + Nc3 * z[2] + Nc4 * z[3] + Nc5 * z[4] + Nc6 *
z[5] + Nc7 * z[6] + Nc8 * z[7];
                double Nabc1[3] = { Na1,Nb1,Nc1 }, Nabc2[3] = { Na2,Nb2,Nc2 }, Nabc3[3] =
{ Na3,Nb3,Nc3 }, Nabc4[3] = { Na4,Nb4,Nc4 }, Nabc5[3] = { Na5,Nb5,Nc5 }, Nabc6[3] =
{ Na6,Nb6,Nc6 }, Nabc7[3] = { Na7,Nb7,Nc7 }, Nabc8[3] = { Na8,Nb8,Nc8 };
                double Nxyz1[3] = { 0 }, Nxyz2[3] = { 0 }, Nxyz3[3] = { 0 }, Nxyz4[3] =
{ 0 }, Nxyz5[3] = { 0 }, Nxyz6[3] = { 0 }, Nxyz7[3] = { 0 }, Nxyz8[3] = { 0 };
                Jc[0] = xa; Jc[1] = xb; Jc[2] = xc; Jc[3] = ya; Jc[4] = yb; Jc[5] = yc; Jc[6]
= za; Jc[7] = zb; Jc[8] = zc;
                Jh = Jc[0] * Jc[4] * Jc[8] + Jc[3] * Jc[7] * Jc[2] + Jc[6] * Jc[1] * Jc[5] -
Jc[2] * Jc[4] * Jc[6] - Jc[1] * Jc[3] * Jc[8] - Jc[0] * Jc[5] * Jc[7];//Jh为雅可比矩阵Jc的行列
式
                mx_inver(Jc, 3);
                for (i = 0; i < 3; i++)
                {
                    for (k = 0; k < 3; k++)
                    {
                        Nxyz1[i] += Jc[i + 3 * k] * Nabc1[k];
                        Nxyz2[i] += Jc[i + 3 * k] * Nabc2[k];
```

```
                    Nxyz3[i] += Jc[i + 3 * k] * Nabc3[k];
                    Nxyz4[i] += Jc[i + 3 * k] * Nabc4[k];
                    Nxyz5[i] += Jc[i + 3 * k] * Nabc5[k];
                    Nxyz6[i] += Jc[i + 3 * k] * Nabc6[k];
                    Nxyz7[i] += Jc[i + 3 * k] * Nabc7[k];
                    Nxyz8[i] += Jc[i + 3 * k] * Nabc8[k];
                }
            }
            B[0][0] = Nxyz1[0]; B[1][1] = Nxyz1[1]; B[2][2] = Nxyz1[2]; B[3][1] = B[5][2] = B[0][0]; B[3][0] = B[4][2] = B[1][1]; B[4][1] = B[5][0] = B[2][2];
            B[0][3] = Nxyz2[0]; B[1][4] = Nxyz2[1]; B[2][5] = Nxyz2[2]; B[3][4] = B[5][5] = B[0][3]; B[3][3] = B[4][5] = B[1][4]; B[4][4] = B[5][3] = B[2][5];
            B[0][6] = Nxyz3[0]; B[1][7] = Nxyz3[1]; B[2][8] = Nxyz3[2]; B[3][7] = B[5][8] = B[0][6]; B[3][6] = B[4][8] = B[1][7]; B[4][7] = B[5][6] = B[2][8];
            B[0][9] = Nxyz4[0]; B[1][10] = Nxyz4[1]; B[2][11] = Nxyz4[2]; B[3][10] = B[5][11] = B[0][9]; B[3][9] = B[4][11] = B[1][10]; B[4][10] = B[5][9] = B[2][11];
            B[0][12] = Nxyz5[0]; B[1][13] = Nxyz5[1]; B[2][14] = Nxyz5[2]; B[3][13] = B[5][14] = B[0][12]; B[3][12] = B[4][14] = B[1][13]; B[4][13] = B[5][12] = B[2][14];
            B[0][15] = Nxyz6[0]; B[1][16] = Nxyz6[1]; B[2][17] = Nxyz6[2]; B[3][16] = B[5][17] = B[0][15]; B[3][15] = B[4][17] = B[1][16]; B[4][16] = B[5][15] = B[2][17];
            B[0][18] = Nxyz7[0]; B[1][19] = Nxyz7[1]; B[2][20] = Nxyz7[2]; B[3][19] = B[5][20] = B[0][18]; B[3][18] = B[4][20] = B[1][19]; B[4][19] = B[5][18] = B[2][20];
            B[0][21] = Nxyz8[0]; B[1][22] = Nxyz8[1]; B[2][23] = Nxyz8[2]; B[3][22] = B[5][23] = B[0][21]; B[3][21] = B[4][23] = B[1][22]; B[4][22] = B[5][21] = B[2][23];
            for (i = 0; i < 6; i++)
            {
                for (j = 0; j < 24; j++)
                {
                    Bt[j][i] = B[i][j];
                }
            }
            double DltStrain[6] = { 0 }, DltStress[6] = { 0 }, DltStressp[6] = { 0 };
double F1, F2, r = 1, r1, r2;
            if (StepId == 0)
            {
                for (i = 0; i < 24; i++)
                {
                    DltU[i] = 0;
                }
            }
            else
            {
                //调用上一载荷步下的节点增量位移
                for (i = 0; i < 8; i++)
```

```
                    {
                        for (j = 0; j < 3; j++)
                        {
                            DltU[3 * i + j] = UAll[3 * (Elem[8 * ElemId + i] - 1) + j];
                        }
                    }
                    for (i = 0; i < 6; i++)
                    {
                        for (k = 0; k < 24; k++)
                        {
                            DltStrain[i] = DltStrain[i] + B[i][k] * DltU[k];
                        }
                    }
                    for (i = 0; i < 6; i++)
                    {
                        for (k = 0; k < 6; k++)
                        {
                            DltStress[i] = DltStress[i] + De[i][k] * DltStrain[k];
                        }
                    }
                    if (StepId == 1)
                    {
                        for (i = 0; i < 6; i++)
                        {
                            Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n +
p) + i] += DltStress[i];
                        }
                    }
                    for (i = 0; i < 6; i++)
                    {
                        Stress[ElemId * 2400 + 48 * StepId + 6 * (4 * m + 2 * n + p) + i] =
Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n + p) + i] + DltStress[i];
                    }
                    double sx, sy, sz, sx_1, sy_1, sz_1, sx_r, sy_r, sz_r;
                    sx_1 = (2 * Stress[ElemId * 2400 + 48 * StepId + 6 * (4 * m + 2 * n + p)
+ 0] - Stress[ElemId * 2400 + 48 * StepId + 6 * (4 * m + 2 * n + p) + 1] - Stress[ElemId * 2400
+ 48 * StepId + 6 * (4 * m + 2 * n + p) + 2]) / 3;
                    sy_1 = (2 * Stress[ElemId * 2400 + 48 * StepId + 6 * (4 * m + 2 * n + p)
+ 1] - Stress[ElemId * 2400 + 48 * StepId + 6 * (4 * m + 2 * n + p) + 0] - Stress[ElemId * 2400
+ 48 * StepId + 6 * (4 * m + 2 * n + p) + 2]) / 3;
                    sz_1 = (2 * Stress[ElemId * 2400 + 48 * StepId + 6 * (4 * m + 2 * n + p)
+ 2] - Stress[ElemId * 2400 + 48 * StepId + 6 * (4 * m + 2 * n + p) + 0] - Stress[ElemId * 2400
+ 48 * StepId + 6 * (4 * m + 2 * n + p) + 1]) / 3;
                    sx = (2 * Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n +
```

```
p) + 0] - Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n + p) + 1] -
Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n + p) + 2]) / 3;
                    sy = (2 * Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n +
p) + 1] - Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n + p) + 0] -
Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n + p) + 2]) / 3;
                    sz = (2 * Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n +
p) + 2] - Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n + p) + 0] -
Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n + p) + 1]) / 3;
                    F1 = (sx_1 * sx_1 + sy_1 * sy_1 + sz_1 * sz_1 + 2 * Stress[ElemId * 2400
+ 48 * StepId + 6 * (4 * m + 2 * n + p) + 3] * Stress[ElemId * 2400 + 48 * StepId + 6 * (4 * m
+ 2 * n + p) + 3] + 2 * Stress[ElemId * 2400 + 48 * StepId + 6 * (4 * m + 2 * n + p) + 4] *
Stress[ElemId * 2400 + 48 * StepId + 6 * (4 * m + 2 * n + p) + 4] + 2 * Stress[ElemId * 2400 +
48 * StepId + 6 * (4 * m + 2 * n + p) + 5] * Stress[ElemId * 2400 + 48 * StepId + 6 * (4 * m +
2 * n + p) + 5]) / 2 - cs * cs / 3;//各向同性硬化
                    F2 = (sx * sx + sy * sy + sz * sz + 2 * Stress[ElemId * 2400 + 48 *
(StepId - 1) + 6 * (4 * m + 2 * n + p) + 3] * Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4
* m + 2 * n + p) + 3] + 2 * Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n + p)
+ 4] * Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n + p) + 4] + 2 *
Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n + p) + 5] * Stress[ElemId * 2400
+ 48 * (StepId - 1) + 6 * (4 * m + 2 * n + p) + 5]) / 2 - cs * cs / 3;
                    //求解弹塑性比例因子r
                    if (F1 > 0)
                    {
                        if (F2 < 0)//开始进入弹塑性阶段
                        {
                            r1 = 0; r2 = 1;
                            while ((r2 - r1) > 0.000000001)
                            {
                                r = (r1 + r2) / 2;
                                F1 = ((sx + r * (2 * DltStress[0] - DltStress[1] -
DltStress[2]) / 3) * (sx + r * (2 * DltStress[0] - DltStress[1] - DltStress[2]) / 3) + (sy + r
* (2 * DltStress[1] - DltStress[0] - DltStress[2]) / 3) * (sy + r * (2 * DltStress[1] -
DltStress[0] - DltStress[2]) / 3) + (sz + r * (2 * DltStress[2] - DltStress[0] - DltStress[1])
/ 3) * (sz + r * (2 * DltStress[2] - DltStress[0] - DltStress[1]) / 3) + 2 * (Stress[ElemId *
2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n + p) + 3] + r * DltStress[3]) * (Stress[ElemId *
2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n + p) + 3] + r * DltStress[3]) + 2 *
(Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n + p) + 4] + r * DltStress[4]) *
(Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n + p) + 4] + r * DltStress[4]) +
2 * (Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n + p) + 5] + r *
DltStress[5]) * (Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n + p) + 5] + r *
DltStress[5])) / 2 - cs * cs / 3;
                                if (F1 > 0)
                                {
                                    r2 = r;
```

```
                                }
                                else
                                {
                                    r1 = r;
                                }
                            }
                            sx_r = (2 * (Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 *
m + 2 * n + p) + 0] + r * DltStress[0]) - (Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 *
m + 2 * n + p) + 1] + r * DltStress[1]) - (Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 *
m + 2 * n + p) + 2] + r * DltStress[2])) / 3;
                            sy_r = (2 * (Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 *
m + 2 * n + p) + 1] + r * DltStress[1]) - (Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 *
m + 2 * n + p) + 0] + r * DltStress[0]) - (Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 *
m + 2 * n + p) + 2] + r * DltStress[2])) / 3;
                            sz_r = (2 * (Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 *
m + 2 * n + p) + 2] + r * DltStress[2]) - (Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 *
m + 2 * n + p) + 0] + r * DltStress[0]) - (Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 *
m + 2 * n + p) + 1] + r * DltStress[1])) / 3;
                            //printf("单元%d开始进入弹塑性阶段\n比例因子
r=%lf\n",elem_id+1,r);
                            double G = E / (2 * (1 + mu)); double Ep = E * Et / (E - Et);
                            double S[6] = { sx_r,sy_r,sz_r,Stress[ElemId * 2400 + 48 *
(StepId - 1) + 6 * (4 * m + 2 * n + p) + 3] + r * DltStress[3],Stress[ElemId * 2400 + 48 *
(StepId - 1) + 6 * (4 * m + 2 * n + p) + 4] + r * DltStress[4],Stress[ElemId * 2400 + 48 *
(StepId - 1) + 6 * (4 * m + 2 * n + p) + 5] + r * DltStress[5] };
                            for (i = 0; i < 6; i++)
                            {
                                for (j = 0; j < 6; j++)
                                {
                                    Dp[i][j] = (9 * G * G / (cs * cs * (3 * G + Ep))) *
S[i] * S[j];
                                }
                            }
                        }
                        else //塑性阶段
                        {
                            r = 0;
                            double G = E / (2 * (1 + mu)); double Ep = E * Et / (E - Et);
                            double css = sqrt((sx_1 * sx_1 + sy_1 * sy_1 + sz_1 * sz_1 + 2 *
Stress[ElemId * 2400 + 48 * StepId + 6 * (4 * m + 2 * n + p) + 3] * Stress[ElemId * 2400 + 48 *
StepId + 6 * (4 * m + 2 * n + p) + 3] + 2 * Stress[ElemId * 2400 + 48 * StepId + 6 * (4 * m + 2
* n + p) + 4] * Stress[ElemId * 2400 + 48 * StepId + 6 * (4 * m + 2 * n + p) + 4] + 2 *
Stress[ElemId * 2400 + 48 * StepId + 6 * (4 * m + 2 * n + p) + 5] * Stress[ElemId * 2400 + 48 *
StepId + 6 * (4 * m + 2 * n + p) + 5]) * 3 / 2);
```

```
                            double S[6] = { sx_1,sy_1,sz_1,Stress[ElemId * 2400 + 48 *
StepId + 6 * (4 * m + 2 * n + p) + 3],Stress[ElemId * 2400 + 48 * StepId + 6 * (4 * m + 2 * n +
p) + 4],Stress[ElemId * 2400 + 48 * StepId + 6 * (4 * m + 2 * n + p) + 5] };
                            for (i = 0; i < 6; i++)
                            {
                                for (j = 0; j < 6; j++)
                                {
                                    Dp[i][j] = (9 * G * G / (css * css * (3 * G + Ep))) *
S[i] * S[j];
                                }
                            }
                        }
                    }
                    for (i = 0; i < 6; i++)
                    {
                        for (j = 0; j < 6; j++)
                        {
                            DltStressp[i] = DltStressp[i] + (De[i][j] - Dp[i][j]) * (1 - r)
* DltStrain[j];
                        }
                    }
                    for (i = 0; i < 6; i++)
                    {
                        Stress[ElemId * 2400 + 48 * StepId + 6 * (4 * m + 2 * n + p) + i] =
Stress[ElemId * 2400 + 48 * (StepId - 1) + 6 * (4 * m + 2 * n + p) + i] + r * DltStress[i] +
DltStressp[i];
                    }
                }
                //计算DB
                for (i = 0; i < 6; i++)
                {
                    for (j = 0; j < 24; j++)
                    {
                        for (k = 0; k < 6; k++)
                        {
                            DB[i][j] += (r * De[i][k] + (1 - r) * (De[i][k] - Dp[i][k])) *
B[k][j];
                        }
                    }
                }
                //计算BDB
                for (i = 0; i < 24; i++)
                {
                    for (j = 0; j < 24; j++)
```

```
                    {
                        for (k = 0; k < 6; k++)
                        {
                            BDB[i][j] += Bt[i][k] * DB[k][j];
                        }
                    }
                }
                for (i = 0; i < 24; i++)
                {
                    for (j = 0; j < 24; j++)
                    {
                        Ke[i][j] += BDB[i][j] * Jh * H[m] * H[n] * H[p];
                    }
                }
            }
        }
    }
    //单元载荷矩阵
    for (i = 0; i < 8; i++)
    {
        Load[3 * i] = 1.0 / 50 * Node[9 * (Elem[8 * ElemId + i] - 1) + 6];
        Load[3 * i + 1] = 1.0 / 50 * Node[9 * (Elem[8 * ElemId + i] - 1) + 7];
        Load[3 * i + 2] = 1.0 / 50 * Node[9 * (Elem[8 * ElemId + i] - 1) + 8];
    }
}

void nonl_writefile_t_3d(double* UTotal)
{
    int i, j = 0;
    double temp = 0;
    FILE* fw;
    fw = fopen("uxuyuz06_total.txt", "w");
    for (i = 0; i < NumNode; i++)
    {
        if (Node[9 * i + 3] != 0)
        {
            fprintf(fw, "%lf\t", UTotal[j]);
            j++;
        }
        else
        {
            fprintf(fw, "%lf\t", temp);
        }
        if (Node[9 * i + 4] != 0)
```

```
            {
                fprintf(fw, "%lf\t", UTotal[j]);
                j++;
            }
            else
            {
                fprintf(fw, "%lf\t", temp);
            }
            if (Node[9 * i + 5] != 0)
            {
                fprintf(fw, "%lf\n", UTotal[j]);
                j++;
            }
            else
            {
                fprintf(fw, "%lf\n", temp);
            }
        }
        fclose(fw);
        printf("节点位移已成功输出至data06_total.txt\n");
}

//Gauss消去法求解线性方程组
int gauss(double* K_gauss, int n, double* u, double* L_gauss)//K为横向存储行矩阵
{
    //高斯消去法求解方程
    //K为左方阵,n为K的阶数,u为待求解向量,F为右向量
    int i, j, m, t;
    double tt, temp;
    double* KN;//Gauss消去法需要不变矩阵KN
    KN = (double*)malloc(sizeof(double) * n * n);
    if (KN == NULL)
    {
        exit(-2);
    }
    double* L, * K;
    L = (double*)malloc(sizeof(double) * n);
    K = (double*)malloc(sizeof(double) * n * n);
    if (K == NULL || L == NULL)
    {
        exit(-2);
    }
    for (i = 0; i < n; i++)
    {
```

```
        for (j = 0; j < n; j++)
        {
            K[i * n + j] = K_gauss[i * n + j];
        }
        L[i] = L_gauss[i];
    }
    for (m = 0; m < n; m++)
    {
        //全选主元
        temp = K[m * n + m];
        t = m;
        for (i = m; i < n; i++)
        {
            if (fabs(K[i * n + m]) > fabs(temp))
            {
                temp = K[i * n + m];
                t = i;
            }
        }
        if (t != m)
        {
            for (j = m; j < n; j++)
            {
                tt = K[m * n + j]; K[m * n + j] = K[t * n + j]; K[t * n + j] = tt;
            }
            tt = L[m]; L[m] = L[t]; L[t] = tt;
        }
        L[m] = L[m] / K[m * n + m];
        KN[m * n + m] = K[m * n + m];
        for (j = m; j < n; j++)
        {
            K[m * n + j] = K[m * n + j] / KN[m * n + m];
        }
        for (i = m + 1; i < n; i++)
        {
            KN[i * n + m] = K[i * n + m] / K[m * n + m];

            for (j = m; j < n; j++)
            {
                K[i * n + j] = K[i * n + j] - KN[i * n + m] * K[m * n + j];
            }
            L[i] = L[i] - KN[i * n + m] * L[m];
        }
    }
```

```
    //回代
    u[n - 1] = L[n - 1];
    for (i = n - 2; i >= 0; i--)
    {
        temp = 0;
        for (j = i + 1; j < n; j++)
        {
            temp = temp + K[i * n + j] * u[j];
        }
        u[i] = L[i] - temp;
    }
    free(KN);
    free(K); free(L);
    return 0;
}

//求解不同载荷步下的总体刚度矩阵、总体载荷矩阵，求解节点增量位移
void nonl_solve_3d()
{
    double Ke[24][24]; double* Load; double* L; double* K; int Id[24]; double* Stress; double*
UU, * UTotal, * EU;//UU为本次增量步中上一次的迭代结果位移,EU为迭代位移差
    int i, j, m, n;
    Stress = (double*)malloc(sizeof(double) * 2400 * NumElem);
    if (Stress == NULL)
    {
        exit(-2);
    }
    for (i = 0; i < 2400 * NumElem; i++)
    {
        Stress[i] = 0;
    }
    int* UVId; int Nodes = 0;
    //重新定义节点约束
    UVId = (int*)malloc(3 * sizeof(int) * NumNode);
    if (UVId == NULL)
    {
        exit(-2);
    }
    Load = (double*)malloc(24 * sizeof(double));
    if (Load == NULL)
    {
        exit(-2);
    }
    for (i = 0; i < NumNode; i++)
```

```
{
    if (Node[9 * i + 3] != 0)
    {
        UVId[3 * i] = Nodes;
        Nodes++;
    }
    else
    {
        UVId[3 * i] = -1;
    }
    if (Node[9 * i + 4] != 0)
    {
        UVId[3 * i + 1] = Nodes;
        Nodes++;
    }
    else
    {
        UVId[3 * i + 1] = -1;
    }
    if (Node[9 * i + 5] != 0)
    {
        UVId[3 * i + 2] = Nodes;
        Nodes++;
    }
    else
    {
        UVId[3 * i + 2] = -1;
    }
}
K = (double*)malloc(sizeof(double) * Nodes * Nodes);
if (K == NULL)
{
    exit(-2);
}
L = (double*)malloc(sizeof(double) * Nodes);
if (L == NULL)
{
    exit(-2);
}
double* U;
U = (double*)malloc(sizeof(double) * Nodes);
if (U == NULL)
{
    exit(-2);
```

```
}
for (i = 0; i < Nodes * Nodes; i++)
{
    K[i] = 0;
}
for (i = 0; i < Nodes; i++)
{
    L[i] = 0;
}
UU = (double*)malloc(sizeof(double) * Nodes);
if (UU == NULL)
{
    exit(-2);
}
UAll = (double*)malloc(sizeof(double) * 3* NumNode);
if (UAll == NULL)
{
    exit(-2);
}
for (i = 0; i < Nodes; i++)
{
    UU[i] = 0;
}
EU = (double*)malloc(sizeof(double) * Nodes);
if (EU == NULL)
{
    exit(-2);
}
double;//材料非线性问题中的总位移
UTotal = (double*)malloc(sizeof(double) * Nodes);
if (UTotal == NULL)
{
    exit(-2);
}
for (i = 0; i < Nodes; i++)
{
    UTotal[i] = 0;
}
for (n = 0; n < 50; n++)
{
    while (1 > 0)
    {
        for (i = 0; i < Nodes * Nodes; i++)
        {
```

```
    K[i] = 0;
}
double ERU = 0;//二范数
for (m = 0; m < NumElem; m++)
{
    for (i = 0; i < 8; i++)
    {
        Id[3 * i] = UVId[3 * (Elem[8 * m + i] - 1)];
        Id[3 * i + 1] = UVId[3 * (Elem[8 * m + i] - 1) + 1];
        Id[3 * i + 2] = UVId[3 * (Elem[8 * m + i] - 1) + 2];
    }
    nonl_elem_severity_3d(Ke, Load, Stress, m, n);
    for (i = 0; i < 24; i++)
    {
        for (j = 0; j < 24; j++)
        {
            if (Id[i] >= 0 && Id[j] >= 0)
            {
                K[Id[i] * Nodes + Id[j]] += Ke[i][j];
            }
            if (Id[i] >= 0)
            {
                L[Id[i]] = Load[i];
            }
        }
    }
}
//求解方程
gauss(K, Nodes, U, L);
for (i = 0; i < Nodes; i++)
{
    EU[i] = U[i] - UU[i];
}
j = 0;
for (i = 0; i < NumNode; i++)
{
    if (Node[9 * i + 3] != 0)
    {
        UAll[3 * i + 0] = U[j];
        j++;
    }
    else
    {
        UAll[3 * i + 0] = 0;
```

```
            }
            if (Node[9 * i + 4] != 0)
            {
                UAll[3 * i + 1] = U[j];
                j++;
            }
            else
            {
                UAll[3 * i + 1] = 0;
            }
            if (Node[9 * i + 5] != 0)
            {
                UAll[3 * i + 2] = U[j];
                j++;
            }
            else
            {
                UAll[3 * i + 2] = 0;
            }
        }

        for (i = 0; i < Nodes; i++)
        {
            ERU = ERU + EU[i] * EU[i];
        }
        ERU = sqrt(ERU);
        printf("%.9lf\n",ERU);
        if (ERU < 0.000000001)
        {
            break;
        }
        for (i = 0; i < Nodes; i++)
        {
            UU[i] = U[i];
        }
    }

    for (i = 0; i < Nodes; i++)
    {
        UTotal[i] += U[i];
    }
    printf("第%d/50次加载完成\n", n + 1);
}
```

```
    nonl_writefile_t_3d(UTotal);

    free(Elem);
    free(Node);
    free(UVId);
    free(Load);
    free(K);
    free(L);
    free(Stress);
    free(U);
    free(UU);
    free(UAll);
    free(EU);
    free(UTotal);
}

/*主函数*/
void main()
{
    nonl_readfile_3d();
    nonl_solve_3d()
    system("pause");
}
```