湖南大學

HUNAN UNIVERSITY

课 程 论 文

论 文 题 目	基于 Matlab 的二维和三维结构有限元分析
学 生 姓 名	何景林
学 生 学 号	S230200192
课 程 名 称	有限元方法与应用
专 业 班 级	机械 2304 班
学 院 名 称	机械与运载工程学院
指 导 老 师	王琥

2024 年 02 月 04 日

基于 Matlab 的二维和三维结构有限元分析

摘　　要

有限元分析(finite element analysis)，是求取复杂微分方程近似解的一种非常有效的工具，是现代数字化科技的一种重要基础性原理。其广泛应用于航空航天、车辆、土木等多个领域，重要性不言而喻，本文主要基于 Matlab 实现了二维和三维结构的有限元分析。

本文进行的有限元分析工作具体如下：

（1）对平面悬臂梁模型进行有限元分析，采用四节点四边形等参元进行求解域离散，通过 matlab 语言编程求解，并与有限元分析商业软件计算结果对比，其位移的平均相对计算误差为 0.07800。

（2）对三维工字型悬臂梁进行有限元分析，采用三维八节点六面体单元进行求解域离散，分析有限元分析的具体过程和原理，而后通过 matlab 语言编程求解，并与有限元分析商业软件计算结果对比，其最大位移的相对计算误差为 0.00484。从结果中可以看出由于腹板的支撑，工字梁的加载翼缘从角落开始向内弯曲，而中心则没有变形那么多，这意味着该有限元代码可以准确的捕捉结构的物理性质。

关键词：有限元分析；四节点四边形单元；三维八节点六面体单元

目 录

插图索引

附表索引

第 1 章 基于 Matlab 的二维结构有限元分析

1.1. 问题描述

本问题对下图所示平面悬臂梁模型进行有限元分析，采用四节点四边形等参元进行求解域离散，通过 matlab 语言编程求解得出该模型受力后的位移场、应力场和应变场，并与商业软件的求解结果进行比较分析。

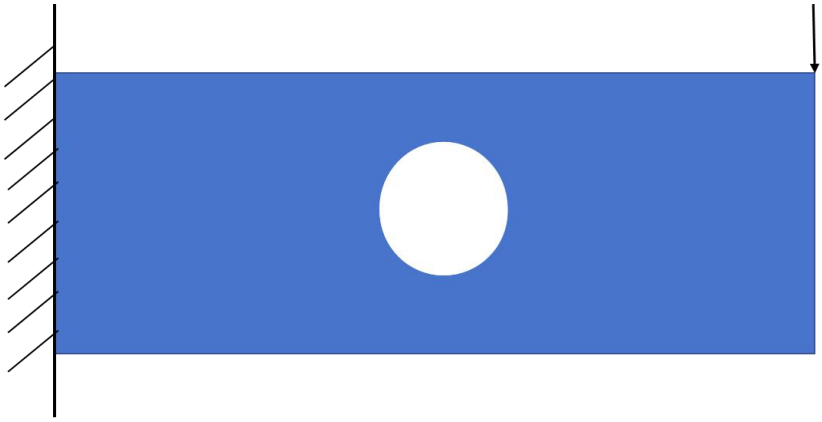

图 1.1 模型

考虑一个左端固定的悬臂梁如图 1.1 所示，其几何参数为：长为 200mm，宽为 100mm，圆孔的直径为 40mm，圆孔位于矩形的形心处；材料参数为：杨氏模量 E=2.1e5Mpa，泊松比 μ=0.3；载荷情况为右上角有一个集中载荷 F=-1000N，求解得出该梁受力变形后的位移、应力和应变。

1.2. 有限元离散单元与方法

1.2.1. 四节点四边形单元

该单元为由 4 节点组成的四边形单，每个节点有 2 个位移(即 2 个自由度)，具体形式如下图 1.2 所示。

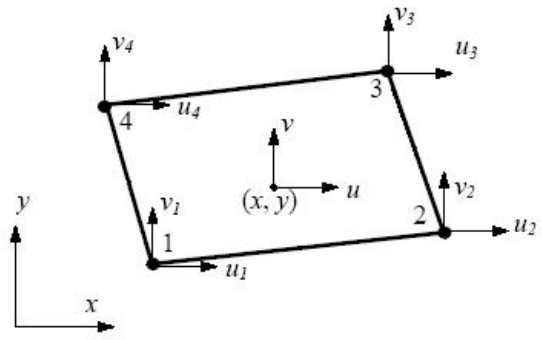

图 1.2 四节点四边形单元

其在内部的映射关系如下图 1.3 所示。

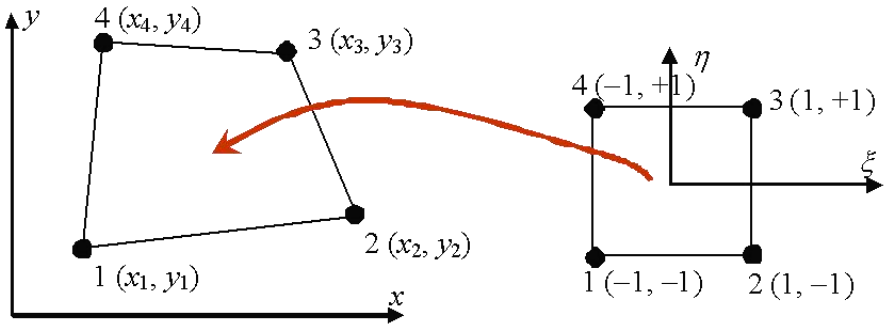

图 1.3 映射关系

1.2.2. 控制方程

形状函数：

$$N_1 = \frac{1}{4}(1-\xi)(1-\eta)$$
$$N_2 = \frac{1}{4}(1+\xi)(1-\eta)$$
$$N_3 = \frac{1}{4}(1+\xi)(1+\eta)$$
$$N_4 = \frac{1}{4}(1-\xi)(1+\eta)$$

雅克比矩阵：

形状函数的导数由以下链式法则公式给出：

$$\begin{cases} \dfrac{\partial N_i}{\partial \xi} = \dfrac{\partial N_i}{\partial x}\dfrac{\partial x}{\partial \xi} + \dfrac{\partial N_i}{\partial y}\dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial N_i}{\partial \eta} = \dfrac{\partial N_i}{\partial x}\dfrac{\partial x}{\partial \eta} + \dfrac{\partial N_i}{\partial y}\dfrac{\partial y}{\partial \eta} \end{cases}$$

用矩阵的表达形式如下：

$$\begin{Bmatrix} \dfrac{\partial N_i}{\partial \xi} \\ \dfrac{\partial N_i}{\partial \eta} \end{Bmatrix} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} \begin{bmatrix} \dfrac{\partial N_i}{\partial x} \\ \dfrac{\partial N_i}{\partial y} \end{bmatrix} = \mathbf{J} \begin{bmatrix} \dfrac{\partial N_i}{\partial x} \\ \dfrac{\partial N_i}{\partial y} \end{bmatrix}$$

$$\mathbf{J}=\begin{bmatrix}\dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi}\\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta}\end{bmatrix}$$

$$\mathbf{J}=\begin{bmatrix}\dfrac{\partial N_1}{\partial \xi} & \dfrac{\partial N_2}{\partial \xi} & \dfrac{\partial N_3}{\partial \xi} & \dfrac{\partial N_4}{\partial \xi}\\ \dfrac{\partial N_1}{\partial \eta} & \dfrac{\partial N_2}{\partial \eta} & \dfrac{\partial N_3}{\partial \eta} & \dfrac{\partial N_4}{\partial \eta}\end{bmatrix}\begin{bmatrix}x_1 & y_1\\ x_2 & y_2\\ x_3 & y_3\\ x_4 & y_4\end{bmatrix}$$

应变矩阵：

$$\mathbf{B}(x,y)=\mathbf{LN}=\begin{bmatrix}\dfrac{\partial}{\partial x} & 0\\ 0 & \dfrac{\partial}{\partial y}\\ \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x}\end{bmatrix}\begin{bmatrix}N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 & 0\\ 0 & N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4\end{bmatrix}$$

本构矩阵

$$\mathbf{D}=\frac{E}{1-\nu^2}\begin{bmatrix}1 & \nu & 0\\ \nu & 1 & 0\\ 0 & 0 & \dfrac{1-\nu}{2}\end{bmatrix}$$

刚度矩阵

$$\mathbf{K}^e=\int_{\Omega^e}\mathbf{B}^{e^T}\mathbf{D}\mathbf{B}^e dV=\int_{-1}^{1}\int_{-1}^{1}\mathbf{B}^{e^T}\mathbf{D}\mathbf{B}^e\left|\mathbf{J}\right|hd\xi d\eta$$

1.2.3. 边界条件

在这一问题中，与悬臂梁加载荷相对的边上的节点所有自由度都设为零。

1.3. Matlab 程序设计与软件验证

Matlab 程序设计

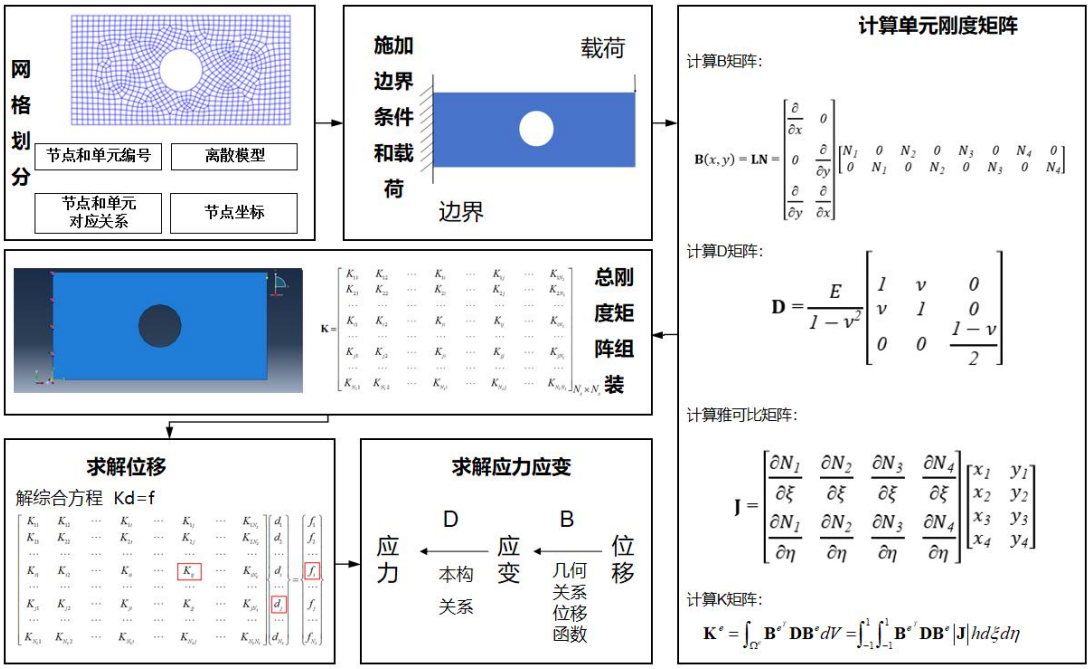

图 1.4 二维 Matlab 程序设计流程

软件验证

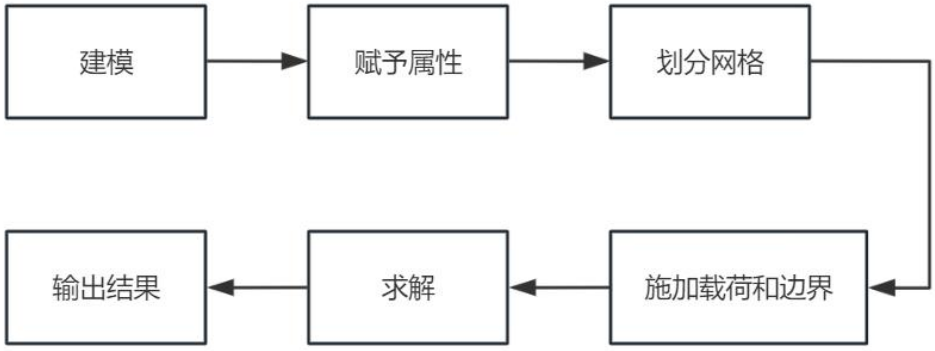

图 1.5 商业软件分析流程

1.4. 结果分析

将程序求解结果与软件计算结果进行对比，下述左图为软件求解结果，右图为程序求解结果。

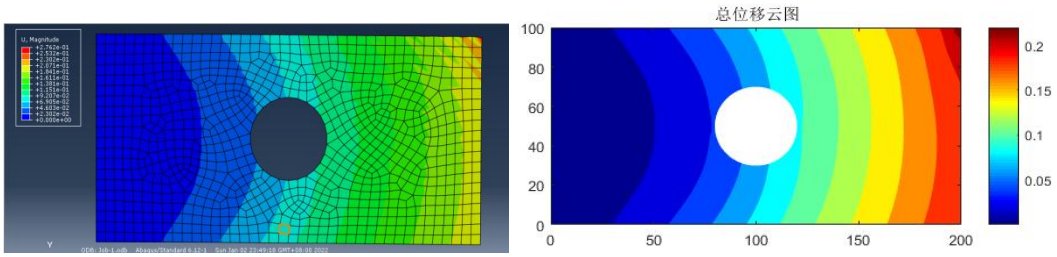

图 1.6 总位移结果对比

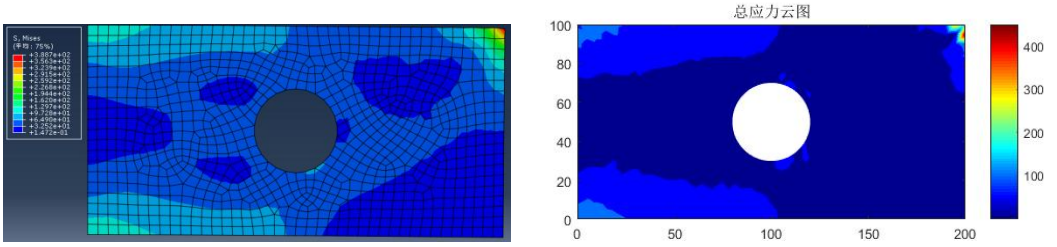

图 1.7 总应力结果对比

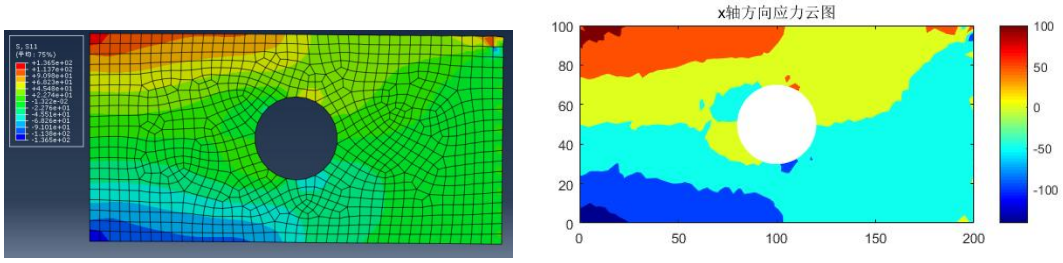

图 1.8 x 轴方向应力对比

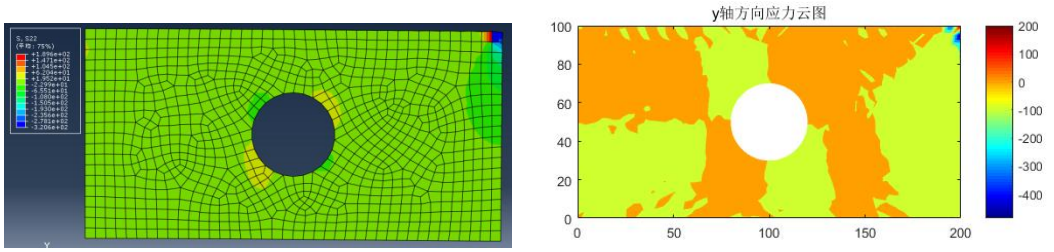

图 1.9 y 轴方向应力对比

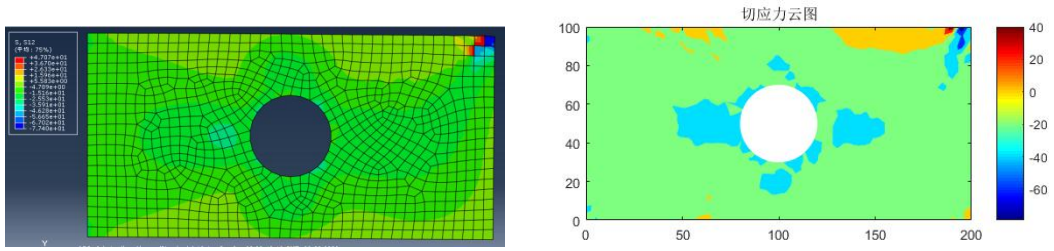

图 1.10 切应力对比

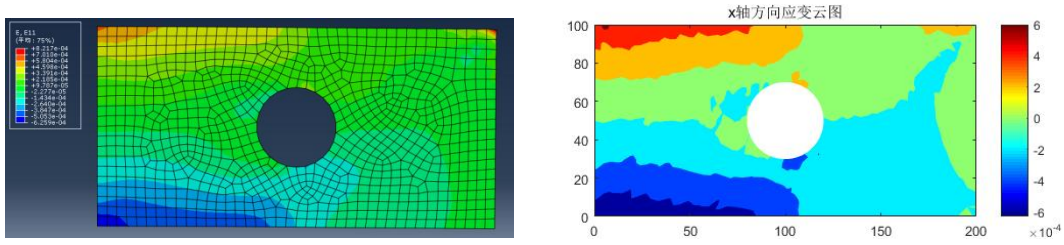

图 1.11 x 轴应变对比

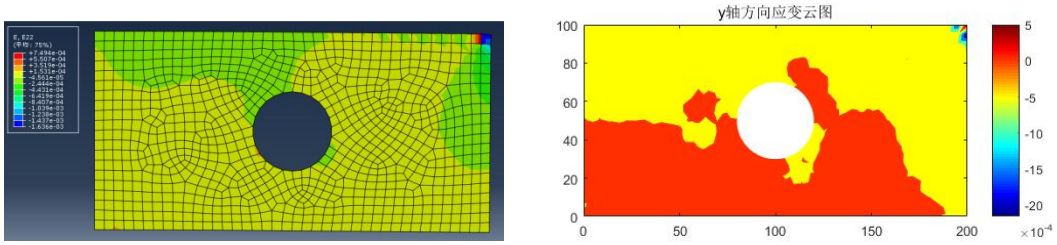

图 1.12 y 轴应变对比

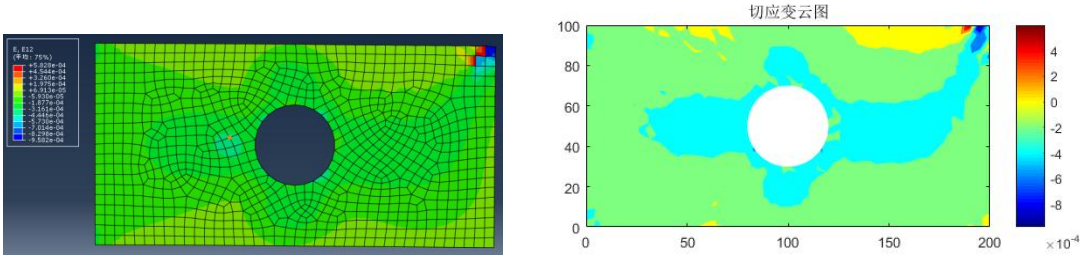

图 1.13 切应变对比

1.5. 误差原文分析

计算出位移的平均相对计算误差为 0.078，应力的平均相对计算误差为 1.1476，应变的平均相对计算误差为 0.6373。位移的求解精度较高，应力和应变的求解误差比较大。

应力和应变的求解误差比较大的原因可能主要受因为 4 节点等参元缺少边中点，不能很好的适应悬臂梁的弯曲变形，加之单元的划分比较少，网格解就存在一定的误差。此外，在应力磨平方面基于 4 个节点上的应力可由高斯点上的应力外推得到的原理可能存在一定的误差。

第 2 章 基于 Matlab 的三维结构有限元分析

2.1. 问题描述

本章节的目的在于通过 Matlab 解决悬臂梁有限元分析问题，涉及工字悬臂梁，采用八节点六面体单元进行求解域离散，载荷被施加在梁的一侧，而相反的面是固定的，工字梁如图所示。

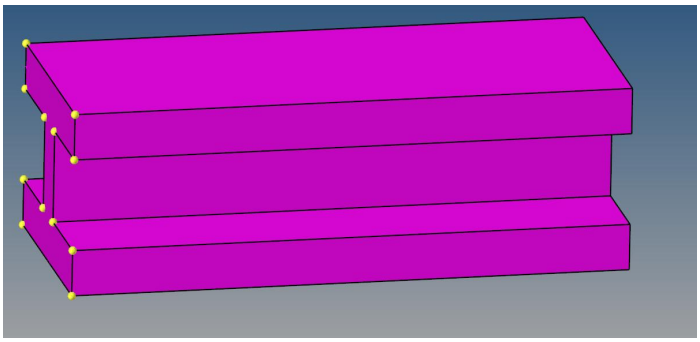

图 2.1 模型

2.2. 有限元离散单元与方法

2.2.1. 8 节点六面体单元

该单元为由 8 节点组成的正六面体单元(hexahedron element)，每个节点有 3 个位移(即 3 个自由度) ，六面体单元的等参坐标或自然坐标称为ξ，η和μ∈(−1；1）；其中，节点编号是非常重要的，它允许我们保证一个正的体积(或者更准确地说是在每个点保证一个正的雅可比行列式)。节点编号可遵循以下规则:

选择一个起始角，编号为 1，其他 3 个角分别为 2、3、4，沿初始面逆时针方向移动；将正对 1、2、3、4 的对角分别编号为 5、6、7、8。

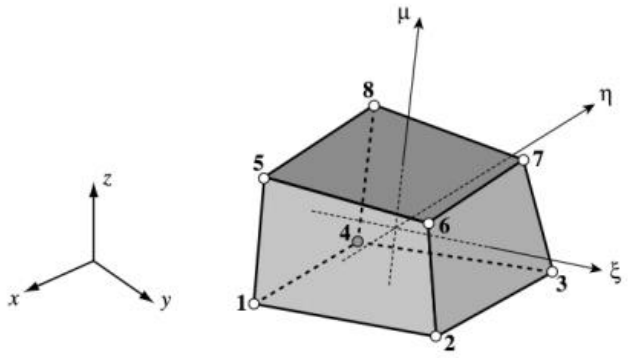

图 2.2 六面体单元

因而，ξ、η和μ的定义可以是：ξ值从 1485 面(中心)的-1 到 2376 面的+1；η值从 1265 面(中心)的-1 变为 3487 面的+1；μ从面 1234(中心)的-1 到面 5678 的+1。

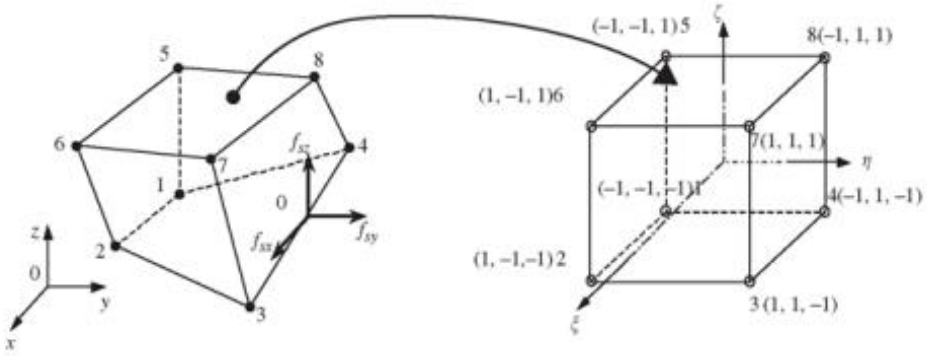

图 2.3 映射关系

2.2.2. 控制方程

形状函数：

$$N_1^e = \frac{1}{8}(1-\xi)(1-\eta)(1-\mu)$$
$$N_2^e = \frac{1}{8}(1+\xi)(1-\eta)(1-\mu)$$
$$N_3^e = \frac{1}{8}(1+\xi)(1+\eta)(1-\mu)$$
$$N_4^e = \frac{1}{8}(1-\xi)(1+\eta)(1-\mu)$$
$$N_5^e = \frac{1}{8}(1-\xi)(1-\eta)(1+\mu)$$
$$N_6^e = \frac{1}{8}(1+\xi)(1-\eta)(1+\mu)$$
$$N_7^e = \frac{1}{8}(1+\xi)(1+\eta)(1+\mu)$$
$$N_8^e = \frac{1}{8}(1-\xi)(1+\eta)(1+\mu)$$

同时，

$$x(\xi,\eta,\mu) = \sum_i N_i(\xi,\eta,\mu)x_i$$

$$y(\xi,\eta,\mu) = \sum_i N_i(\xi,\eta,\mu)y_i$$

$$z(\xi,\eta,\mu) = \sum_i N_i(\xi,\eta,\mu)z_i$$

$$u(\xi,\eta,\mu)=\sum_i N_i(\xi,\eta,\mu)u_i$$
$$v(\xi,\eta,\mu)=\sum_i N_i(\xi,\eta,\mu)v_i$$
$$w(\xi,\eta,\mu)=\sum_i N_i(\xi,\eta,\mu)w_i$$

雅克比矩阵：

形状函数的导数由以下链式法则公式给出：

$$\frac{\partial N_i^e}{\partial x}=\frac{\partial N_i^e}{\partial \xi}\frac{\partial \xi}{\partial x}+\frac{\partial N_i^e}{\partial \eta}\frac{\partial \eta}{\partial x}+\frac{\partial N_i^e}{\partial \mu}\frac{\partial \mu}{\partial x}$$

$$\frac{\partial N_i^e}{\partial y}=\frac{\partial N_i^e}{\partial \xi}\frac{\partial \xi}{\partial y}+\frac{\partial N_i^e}{\partial \eta}\frac{\partial \eta}{\partial y}+\frac{\partial N_i^e}{\partial \mu}\frac{\partial \mu}{\partial y}$$

$$\frac{\partial N_i^e}{\partial z}=\frac{\partial N_i^e}{\partial \xi}\frac{\partial \xi}{\partial z}+\frac{\partial N_i^e}{\partial \eta}\frac{\partial \eta}{\partial z}+\frac{\partial N_i^e}{\partial \mu}\frac{\partial \mu}{\partial z}$$

用矩阵的表达形式如下：

$$\begin{pmatrix}\frac{\partial N_i^e}{\partial x}\\ \frac{\partial N_i^e}{\partial y}\\ \frac{\partial N_i^e}{\partial z}\end{pmatrix}=\begin{pmatrix}\frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} & \frac{\partial \mu}{\partial x}\\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} & \frac{\partial \mu}{\partial y}\\ \frac{\partial \xi}{\partial z} & \frac{\partial \eta}{\partial z} & \frac{\partial \mu}{\partial z}\end{pmatrix}\begin{pmatrix}\frac{\partial N_i^e}{\partial \xi}\\ \frac{\partial N_i^e}{\partial \eta}\\ \frac{\partial N_i^e}{\partial \mu}\end{pmatrix}$$

（ J^{-1}）

$$\Longrightarrow J^{-1}=\frac{\partial(\xi,\eta,\mu)}{\partial(x,y,z)}$$

$$\Longrightarrow J=\frac{\partial(x,y,z)}{\partial(\xi,\eta,\mu)}=\begin{pmatrix}\frac{\partial x}{\partial\xi} & \frac{\partial y}{\partial\xi} & \frac{\partial z}{\partial\xi}\\ \frac{\partial x}{\partial\eta} & \frac{\partial y}{\partial\eta} & \frac{\partial z}{\partial\eta}\\ \frac{\partial x}{\partial\mu} & \frac{\partial y}{\partial\mu} & \frac{\partial z}{\partial\mu}\end{pmatrix}$$

由于，六面体元的等参定义为：

$$x=x_iN_i^e, y=y_iN_i^e, z=z_iN_i^e$$

$$\Longrightarrow J=\begin{pmatrix}x_i\frac{\partial N_i^e}{\partial\xi} & y_i\frac{\partial N_i^e}{\partial\xi} & z_i\frac{\partial N_i^e}{\partial\xi}\\ x_i\frac{\partial N_i^e}{\partial\eta} & y_i\frac{\partial N_i^e}{\partial\eta} & z_i\frac{\partial N_i^e}{\partial\eta}\\ x_i\frac{\partial N_i^e}{\partial\mu} & y_i\frac{\partial N_i^e}{\partial\mu} & z_i\frac{\partial N_i^e}{\partial\mu}\end{pmatrix}$$

应变矩阵

应变矩阵 B 由下式给出：

$$\mathbf{B}=\mathbf{D\Phi}=\begin{pmatrix}\frac{\partial}{\partial x} & 0 & 0\\ 0 & \frac{\partial}{\partial x} & 0\\ 0 & 0 & \frac{\partial}{\partial x}\\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0\\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y}\\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x}\end{pmatrix}\begin{pmatrix}\mathbf{q} & 0 & 0\\ \mathbf{q} & \mathbf{q} & 0\\ \mathbf{q} & \mathbf{q} & 0\\ \mathbf{q_y} & \mathbf{q_x} & 0\\ 0 & \mathbf{q_z} & \mathbf{q_y}\\ \mathbf{q_z} & 0 & \mathbf{q_x}\end{pmatrix}=\begin{pmatrix}\mathbf{q_x} & 0 & 0\\ 0 & \mathbf{q_y} & 0\\ 0 & 0 & \mathbf{q_z}\\ \mathbf{q_y} & \mathbf{q_x} & 0\\ 0 & \mathbf{q_z} & \mathbf{q_y}\\ \mathbf{q_z} & 0 & \mathbf{q_x}\end{pmatrix}$$

其中，

$$\mathbf{q} = [N_1^e \quad \dots \quad N_n^e]$$

本构矩阵

三维材料矩阵 C 与材料泊松比μ和杨氏模量 E 有关；具体的本构矩阵 C 由下式给出：

$$\mathbf{C} = \begin{pmatrix} \lambda+2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda+2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda+2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{pmatrix}$$

其中，

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}$$

$$\mu = \frac{E}{2(1+\nu)}$$

刚度矩阵

单元的刚度矩阵由下式给出：

$$\mathbf{K^e} = \int_{V^e} \mathbf{B^T C B}\, dV^e$$

在二维的情况下，这被一个数值积分公式所取代，这个公式现在包含了一个超越传统高斯正交规则的三重环。假设 C 矩阵是常数，则：

$$\mathbf{K}^e = \sum_{i=1}^{p_1}\sum_{j=1}^{p_2}\sum_{k=1}^{p_3} w_i w_j w_k \mathbf{B_{ijk}^T C B_{ijk}} J_{ijk}$$

其中，B_{ijk} 和 J_{ijk} 是：

$$\mathbf{B}_{ijk} = \mathbf{B}(\xi_i, \eta_j, \mu_k) and J_{ijk} = det\mathbf{J}(\xi_i, \eta_j, \mu_k)$$

而且，p1，p2 和 p3 分别表示ξ，η和μ方向上的高斯点个数，各方向上的高斯点个数一般相同，即对于 8 节点六面体单元 p = p1，p2 和 p3 = 2。

高斯点的坐标如下：

$$\begin{cases} \left(-\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}\right) \\ \left(\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}\right) \\ \left(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}\right) \\ \left(-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}\right) \\ \left(-\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right) \\ \left(\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right) \\ \left(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right) \\ \left(-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right) \end{cases}$$

2.2.3. 边界条件

在这一问题中，与悬臂梁加载边相对的面上的所有自由度都设为零。

2.3. Matlab 程序设计与商业软件验证

Matlab 程序设计

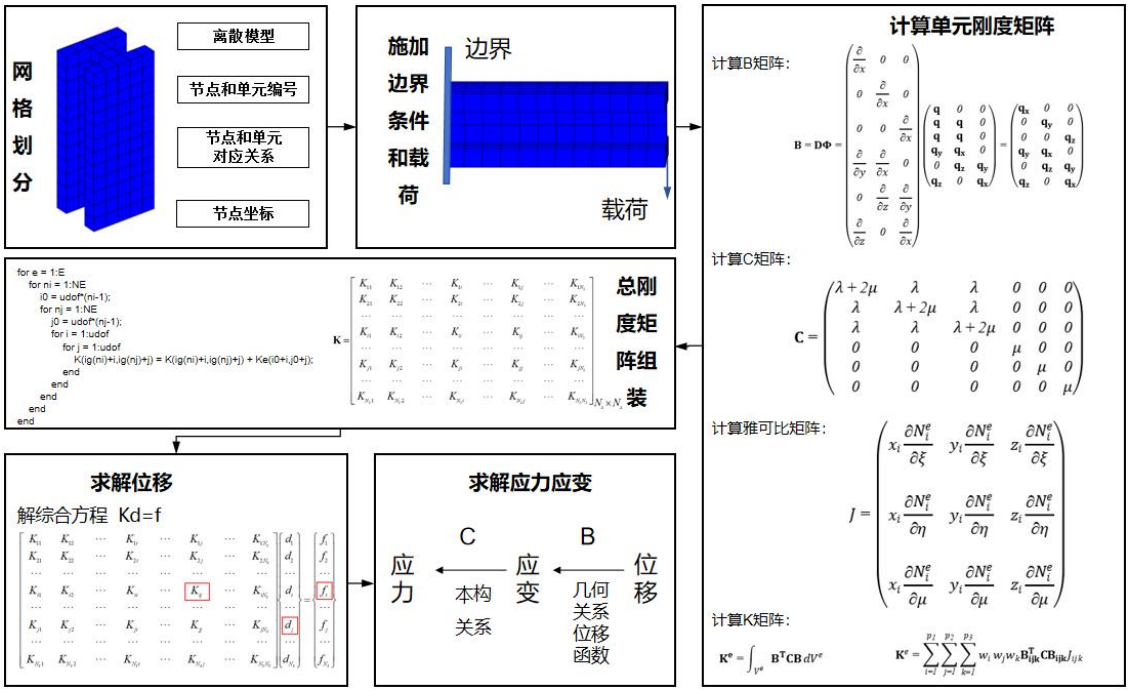

图 2.4 三维 Matlab 程序设计流程

基于前文所述的有限元离散分析方法，基于 matlab 设计了悬臂梁一类问题的分析程序，程序流程如下图所示，具体代码和注释见附录。

软件验证

软件验证流程如下所示：

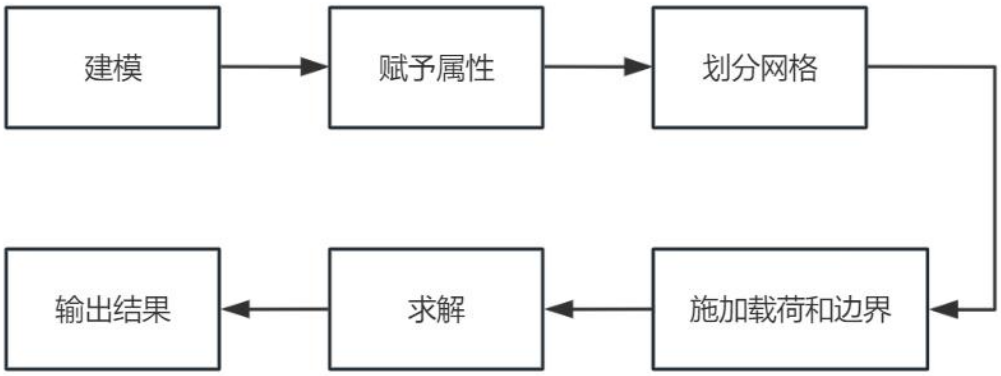

图 2.5 商业软件分析流程

2.4. 结果分析

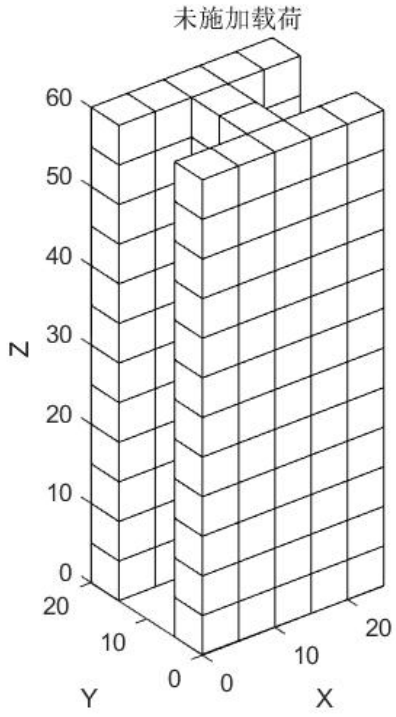

图 2.6 未施加载荷前工字悬臂梁

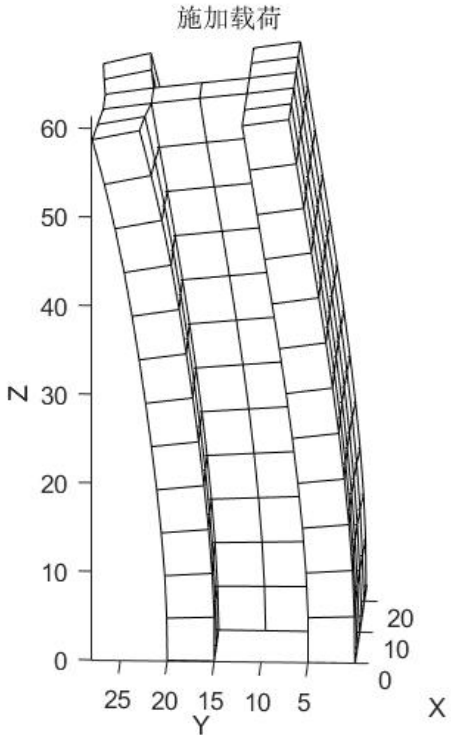

图 2.7 施加载荷后的工字悬臂梁变形

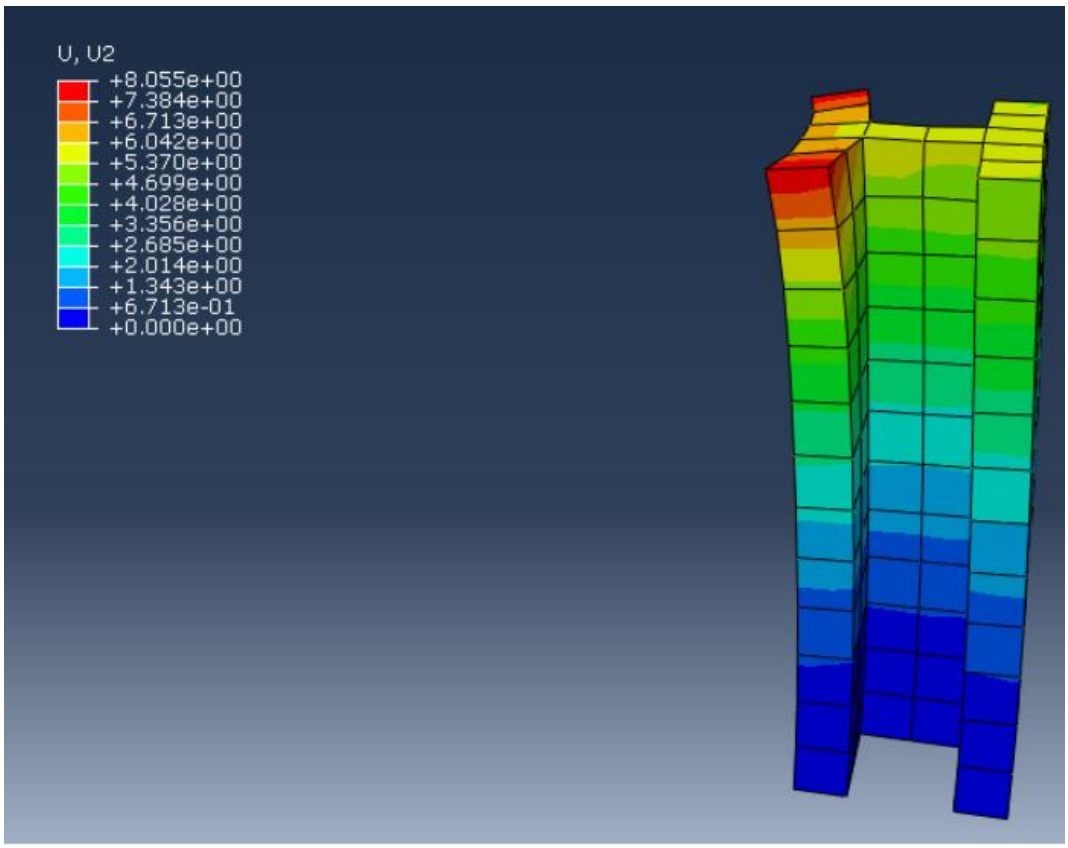

图 2.8 商用软件位移分析结果

表 2-1 位移计算结果分析

	最大位移	方向	相对误差
代码计算结果	8.0164	Y	0.00484
软件仿真结果	8.05540	Y	

从上面的表格和图中可以看出，有限元代码的结果与软件仿真的结果相吻合。从结果中可以看出由于腹板的支撑，工字梁的加载翼缘从角落开始向内弯曲，而中心则没有变形那么多，整个梁是弯曲的，这意味着有限元代码可以准确地捕捉结构的物理性质。

结 论

本文主要基于 Matlab 对二维和三维的悬臂梁进行了有限元分析，并将计算结果与商用有限元软件进行对比验证分析，

1. 对平面悬臂梁模型进行有限元分析，采用四节点四边形等参元进行求解域离散，通 atlab 语言编程求解，并与有限元分析商业软件计算结果对比，其位移的平均相对计算误差为 0.07800。

对三维工字型悬臂梁进行有限元分析，采用三维八节点六面体单元进行求解域离散，分析有限元分析的具体过程和原理，而后通过 matlab 语言编程求解，并与有限元分析商业软件计算结果对比，其最大位移的相对计算误差为 0.00484。从结果中可以看出由于腹板的支撑，工字梁的加载翼缘从角落开始向内弯曲，而中心则没有变形那么多，整个梁是弯曲的，这意味着有限元代码可以准确地捕捉结构的物理性质。

参考文献

[1]Gary F. Dargush. Lecture notes in finite element analysis(mae529). Department of Mechanical and Aerospace Engineering, University at Buffalo, The State University of New York, Fall 2016.

[2]Carlos A. Felippa. Advanced finite element methods (asen 6367), department of aerospace engineering sciences university of colorado at boulder, 2013.

附 录

附录 A 二维悬臂梁分析 Matlab 程序

```
clear
format short
first_time=cputime; % 用于计算程序执行总时长
%% 初始化
% 物理参数
E=2.1e5; %杨氏模量
miu=0.3; %泊松比
% 网格节点信息输入
node=importdata('nodes1.txt'); % 节点坐标信息，表示节点 x 和 y 方向的坐标
ele=importdata('element1.txt'); % 单元节点信息，表示每个单元上的节点号码
nel=length(ele(:,1)); % 单元总个数
nnode=length(node(:,1)); % 节点总个数
nnel=4; % 每单元节点数
ndof=2; % 每节点自由度数
sdof=nnode*ndof; % 总自由度数
edof=nnel*ndof; % 每单元自由度数
% 显示网格划分情况
figure(1)
hold on
axis off
axis equal
xp=[];
yp=[];
for ie=1:nel
for j=1:nnel+1
```

```
j1=mod(j-1,nnel)+1;
xp(j)=node(ele(ie,j1),1);
yp(j)=node(ele(ie,j1),2);
end
plot(xp,yp,'b-')
end
%% 施加边界条件和载荷
fix_node=[26, 28, 12, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141,142,
143, 144, 145, 146];
bcdof=[]; % 受约束的自由度
bcval=[]; % 受约束自由度对应的位移约束值
for i=1:length(fix_node)
bcdof=[bcdof 2*fix_node(i)-1 2*fix_node(i)];
bcval=[bcval 0 0];
end
% 施加载荷
ff(54,1)=-1000; % 给右上角节点施加一个 y 方向的向下的集中力，大小为 1000N
%% 单元刚度矩阵计算与总体刚度矩阵的组装
% 矩阵初始化
ff=zeros(sdof,1); % 系统载荷初始化
k=zeros(edof,edof); % 单元刚度初始化
kk=sparse(sdof,sdof); % 整体刚度初始化
disp=zeros(sdof,1); % 整体位移初始化
index=zeros(edof,1); % 各单元的 8 个自由度对应的整体的自由度编号
B=sparse(3,edof); % 应变矩阵 B
D=E/(1-miu*miu)*[1 miu 0;
miu 1 0;
0 0 (1-miu)/2]; % 平面应力问题的本构矩阵 D
G=2; % 高斯积分积分点的个数，G=1 时为一点高斯积分
```

```
[point,weight]=fun_GaussQuadrature(G);
% fun_GaussQuadrature()函数返回高斯积分的积分点坐标和权值，具体含义在子函数中有解释
for iel=1:nel % 遍历各单元
for i=1:nnel % 遍历每单元各节点
nd(i)=ele(iel,i); % 向量 nd 用来提取每个单元的节点序号
xcoord(i)=node(nd(i),1); % 向量 xcoord 用来提取每个单元的节点序号在物理空间下的 x 坐标
ycoord(i)=node(nd(i),2); % 向量 ycoord 用来提取每个单元的节点序号在物理空间下的 y 坐标
end
% 开始计算刚度矩阵
for ix=1:2 % 遍历 x 方向的各积分点
x=point(ix,1); % 提取 x 方向积分点的坐标值
wx=weight(ix,1); % 提取 x 方向积分点的权重
for iy=1:2 % 遍历 y 方向的各积分点
y=point(iy,2); % 提取 y 方向积分点的坐标值
wy=weight(iy,2); % 提取 y 方向积分点的权重
[shape,dNdr,dNds]=fun_shapeFunction(x,y);% 计算积分点的形函数的值和对自然坐标 r、s 的导
数
Jacobian=fun_Jacobian(nnel,dNdr,dNds,xcoord,ycoord); % 计算雅可比矩阵
detJacobian=det(Jacobian); % 计算雅可比矩阵的行列式
invJacobian=inv(Jacobian); % 计算雅可比矩阵的逆
[dNdx,dNdy]=fun_dNdx_dNdy(nnel,dNdr,dNds,invJacobian);% 计算形函数对物理空间坐标 x 和 y
的导数
% dNdx 和 dNdy 是 4 维的向量，分别表示第 i 个形函数对 x 和 y 的导数
B=fun_B(nnel,dNdx,dNdy); % 组装 B 矩阵
k=k+B'*D*B*wx*wy*detJacobian; % 组装单元刚度矩阵
end
end
index=fun_index(nd,nnel,ndof); % 提取单元的系统自由度编号
kk=fun_assembleKK(kk,k,index); % 组装整体刚度矩阵
```

```
end
%% 零位移边界条件处理：对角线元素改 1 法
[kk,ff]=fun_boundary(kk,ff,bcdof,bcval);
%% 方程求解
[L,U]=lu(kk);
temp=L\ff;
disp=U\temp ; % 得到节点位移
solve_time = cputime-first_time % 输出求解时间
% 变形后的坐标
disp=load('disp.txt');
disp_x=disp(1:2:sdof);
disp_y=disp(2:2:sdof);
dis=[disp_x,disp_y];
node_new=node+dis;
%% 单元应力应变应变计算
const=[1.866 -0.5 0.134 -0.5; % 常数矩阵，用于应力外推，用高斯点的应力值来插值节点的应力
值
-0.5 1.866 -0.5 0.134;
0.134 -0.5 1.866 -0.5;
-0.5 0.134 -0.5 1.866];
stress=zeros(nel,4,3);
strain=zeros(nel,4,3);
for iel=1:nel % 遍历各单元
for i=1:nnel % 遍历每单元各节点
nd(i)=ele(iel,i); % 向量 nd 用来提取每个单元的节点序号
xcoord(i)=node(nd(i),1); % 向量 xcoord 用来提取每个单元的节点序号在物理空间下的 x 坐标
ycoord(i)=node(nd(i),2); % 向量 ycoord 用来提取每个单元的节点序号在物理空间下的 y 坐标
end
pk=0;
```

```
stress_Gauss=zeros(3,4);
strain_Gauss=zeros(3,4);
for ix=1:2 % 遍历 x 方向的各积分点
x=point(ix,1); % 提取 x 方向积分点的坐标值
wx=weight(ix,1); % 提取 x 方向积分点的权重
for iy=1:2 % 遍历 y 方向的各积分点
y=point(iy,2); % 提取 y 方向积分点的坐标值
wy=weight(iy,2); % 提取 y 方向积分点的权重
[shape,dNdr,dNds]=fun_shapeFunction(x,y);% 计算积分点的形函数的值和对自然坐标 r、s 的导
数
Jacobian=fun_Jacobian(nnel,dNdr,dNds,xcoord,ycoord); % 计算雅可比矩阵
detJacobian=det(Jacobian); % 计算雅可比矩阵的行列式
invJacobian=inv(Jacobian); % 计算雅可比矩阵的逆
[dNdx,dNdy]=fun_dNdx_dNdy(nnel,dNdr,dNds,invJacobian);% 计算形函数对物理空间坐标 x 和 y
的导数
% dNdx 和 dNdy 是 4 维的向量，分别表示第 i 个形函数对 x 和 y 的导数
B=fun_B(nnel,dNdx,dNdy); % 组装 B 矩阵
index=fun_index(nd,nnel,ndof); % 单元节点自由度索引
for i=1:edof
disp_ele(i,1)=disp(index(i)); % 单元节点位移
end
strain_ele=B*disp_ele; % 计算单元应变
stress_ele=D*strain_ele; % 计算单元应力
pk=pk+1;
stress_Gauss(:,pk)=stress_ele;
strain_Gauss(:,pk)=strain_ele;
end
end
for i=1:3
```

```
for j=1:4
for n=1:4
stress(iel,j,i)= stress(iel,j,i)+const(j,n)*stress_Gauss(i,n);
strain(iel,j,i)= strain(iel,j,i)+const(j,n)*strain_Gauss(i,n);
end
end
end
end
% 找出每节点相邻的单元
neigh_node = cell(nnode,1);
neigh_node_ind = cell(nnode,1);
indneigh=zeros(1,nnode);
for i=1:nel
for j=1:4
indneigh(ele(i,j))=indneigh(ele(i,j))+1;
neigh_node{ele(i,j)}(indneigh(ele(i,j)))=i;
neigh_node_ind{ele(i,j)}(indneigh(ele(i,j)))=j;
end
end
% 计算节点处的应力应变
stress_node=zeros(3,nnode);
strain_node=zeros(3,nnode);
for inode=1:nnode
numel= indneigh(inode);
for i=1:numel
ind_nel= neigh_node{inode}(i);
ind_nod=neigh_node_ind{inode}(i);
for j=1:3
stress_node(j,inode)=stress_node(j,inode)+stress(ind_nel,ind_nod,j);
```

```
strain_node(j,inode)=strain_node(j,inode)+strain(ind_nel,ind_nod,j);
end
end
stress_node(:,inode)=stress_node(:,inode)/numel; % 节点处的应力为不同相邻单元磨平后应力的平均值
strain_node(:,inode)=strain_node(:,inode)/numel;
end
%% 绘图
fun_picture(node,disp,stress_node,strain_node);
%% 输出
% 计算误差检查
fun_check(disp,stress_node,strain_node);
% 输出所有计算结果到 report_all.txt 文件
%---------------------------------------------------------------
fid_out=fopen('report_all.txt','w');
fprintf(fid_out,'TITLE="程序计算结果"\n');
fprintf(fid_out,'VARIABLES="x" "y" "u" "v" "sigax" "sigmay" "sigmaxy" "epsilonx" "epsilony" "epsilonxy"\n');
fprintf(fid_out,' N= %8d,E=%8d,Element Type = QUADRILATERA\n',nnode,nel);
for i=1:nnode
fprintf(fid_out,'%16.6e%16.6e%16.6e%16.6e%16.6e%16.6e%16.6e%16.6e%16.6e%16.6e\n',node(i,1),node(i,2),disp(2*i-1),disp(2*i),stress_node(1,i),stress_node(2,i),stress_node(3,i),strain_node(1,i),strain_node(2,i),strain_node(3,i));
end
for i=1:nel
fprintf(fid_out,'%8d%8d%8d%8d\n',ele(i,1),ele(i,2),ele(i,3),ele(i,4));
end
%% 子函数
% 提取单元的整体自由度编号
```

```
function index=fun_index(nd,nnel,ndof)
% nd：单元节点号
% nnel：每单元节点数
% ndof：每节点自由度
edof = nnel*ndof;
k=0;
for i=1:nnel
start = (nd(i)-1)*ndof;
for j=1:ndof
k=k+1;
index(k)=start+j;
end
end
end
% 计算形函数对物理坐标 x 和 y 的导数
function [dNdx,dNdy]=fun_dNdx_dNdy(nnel,dNdr,dNds,invJacobian)
for i=1:nnel
dNdx(i)=invJacobian(1,1)*dNdr(i)+invJacobian(1,2)*dNds(i);
dNdy(i)=invJacobian(2,1)*dNdr(i)+invJacobian(2,2)*dNds(i);
end
end
% 返回高斯积分点处的形函数值和形函数对自然坐标的导数值
function [shape,dNdr,dNds]=fun_shapeFunction(r,s)
% r，s 为积分点坐标值
% 计算自然坐标系下每个节点的形函数
shape(1)=0.25*(1-r)*(1-s);
shape(2)=0.25*(1+r)*(1-s);
shape(3)=0.25*(1+r)*(1+s);
shape(4)=0.25*(1-r)*(1+s);
```

```
% 计算形函数对自然坐标 r 的导数
dNdr(1)=-0.25*(1-s);
dNdr(2)=0.25*(1-s);
dNdr(3)=0.25*(1+s);
dNdr(4)=-0.25*(1+s);
% 计算形函数对自然坐标 s 的导数
dNds(1)=-0.25*(1-r);
dNds(2)=-0.25*(1+r);
dNds(3)=0.25*(1+r);
dNds(4)=0.25*(1-r);
end
% 计算雅可比矩阵
function Jacobian=fun_Jacobian(nnel,dNdr,dNds,xcoord,ycoord)
% nnel:单元节点数
% dNdr，dNds：形函数对自然坐标的导数
% xcoord,ycoord：物理空间下单元四个节点的坐标
Jacobian=zeros(2,2);
for i=1:nnel
Jacobian(1,1)=Jacobian(1,1)+dNdr(i)*xcoord(i);
Jacobian(1,2)=Jacobian(1,2)+dNdr(i)*ycoord(i);
Jacobian(2,1)=Jacobian(2,1)+dNds(i)*xcoord(i);
Jacobian(2,2)=Jacobian(2,2)+dNds(i)*ycoord(i);
end
end
% 输出二维高斯积分的求积节点
function [point,weight]=fun_GaussQuadrature(i)
% point 和 weight 矩阵的第一列表示 x 轴积分的节点和权重
% point 和 weight 矩阵的第二列表示 y 轴积分的节点和权重
% 有几个求积节点 point 和 weight 就有几行
```

```
point=zeros(i,2); % i 表示有 i 个求积节点，2 指代 x 轴和 y 轴
weight=zeros(i,2);
if i==1 % 1 点高斯积分
point=[0,0];
weight=[2,2];
elseif i==2 % 2 点高斯积分
point=[-0.577,-0.577;
0.577,0.577];
weight=[1,1
1,1];
end
end
% 组装整体刚度矩阵
function kk=fun_assembleKK(kk,k,index)
edof = length(index);
for i=1:edof
ii=index(i);
for j=1:edof
jj=index(j);
kk(ii,jj)=kk(ii,jj)+k(i,j);
end
end
end
end
% 组装应变矩阵 B
function B=fun_B(nnel,dNdx,dNdy)
for i=1:nnel
i1=(i-1)*2+1;
i2=i1+1;
B(1,i1)=dNdx(i);
```

```
B(2,i2)=dNdy(i);
B(3,i1)=dNdy(i);
B(3,i2)=dNdx(i);
end
end
% 引入边界条件，采用对角线元素改 1 法消去矩阵线性方程组的约束项
function [kk,ff]=fun_boundary(kk,ff,bcdof,bcval)
n=length(bcdof);
sdof=length(kk);
for i=1:n
c=bcdof(i);
for j=1:sdof
kk(c,j)=0;
end
kk(c,c)=1;
ff(c)=bcval(i);
end
end
% 绘制云图
function fun_picture(node,disp,stress_node,strain_node)
x=node(:,1);
y=node(:,2);
% 绘制位移云图
[m,n]=size(disp);
z=[];
for i=1:2:m-1
z=[z;sqrt(disp(i)*disp(i)+disp(i+1)*disp(i+1))];
end
```

```
[X,Y,Z]=griddata(x,y,z,linspace(min(x),max(x),2000)',linspace(min(y),max(y),1000),
'linear'); %插值
%去掉模型外部分的值
for i=1:2000
for j=1:1000
if sqrt((i-1000)^2+(j-500)^2)<200
Z(j,i)=NaN;
end
end
end
%绘制等高线图
figure('name','合位移云图');
contourf(X,Y,Z,'LineStyle','none');
colormap jet
hold on;
colorbar;
axis equal
title('合位移云图');

%绘制应力云图
[m1,n2]=size(stress_node);
z2=[];
for i=1:n2
z2=[z2 sqrt(stress_node(1,i)*stress_node(1,i)+stress_node(2,i)*stress_node(2,i))];
end
[X,Y,Z0]=griddata(x,y,z2,linspace(min(x),max(x),2000)',linspace(min(y),max(y),1000)
,'linear');
%去掉模型外部分的值
for i=1:2000
```

```
for j=1:1000
if sqrt((i-1000)^2+(j-500)^2)<200
Z0(j,i)=NaN;
end
end
end
%绘制等高线图
figure('name','总应力云图');
contourf(X,Y,Z0,'LineStyle','none') ;
colormap jet
hold on;
colorbar;
axis equal;
title('总应力云图');
%%x 轴应力
[X,Y,Z1]=griddata(x,y,stress_node(1,:),linspace(min(x),max(x),2000)',linspace(min(
y),max(y),1000),'linear');
%去掉模型外部分的值
for i=1:2000
for j=1:1000
if sqrt((i-1000)^2+(j-500)^2)<200
Z1(j,i)=NaN;
end
end
end
%绘制等高线图
figure('name','x 轴方向应力云图');
contourf(X,Y,Z1,'LineStyle','none') ;
colormap jet
```

```
hold on;
colorbar;
axis equal;
title('x 轴方向应力云图');
%%y 轴应力
%插值
[X,Y,Z2]=griddata(x,y,stress_node(2,:),linspace(min(x),max(x),2000)',linspace(min(
y),max(y),1000),'linear');
%去掉模型外部分的值
for i=1:2000
for j=1:1000
if sqrt((i-1000)^2+(j-500)^2)<200
Z2(j,i)=NaN;
end
end
end
%绘制等高线图
figure('name','y 轴方向应力云图');
contourf(X,Y,Z2,'LineStyle','none') ;
colormap jet
hold on;
colorbar;
axis equal;
title('y 轴方向应力云图');
%%切应力
[X,Y,Z3]=griddata(x,y,stress_node(3,:),linspace(min(x),max(x),2000)',linspace(min(
y),max(y),1000),'linear');
%去掉模型外部分的值
for i=1:2000
```

```
for j=1:1000
if sqrt((i-1000)^2+(j-500)^2)<200
Z3(j,i)=NaN;
end
end
end
%绘制等高线图
figure('name','切应力云图');
contourf(X,Y,Z3,'LineStyle','none') ;
colormap jet
hold on;
colorbar;
axis equal;
title('切应力云图');
%%绘制应变云图
%%x 轴应变
[X,Y,Z1]=griddata(x,y,strain_node(1,:),linspace(min(x),max(x),2000)',linspace(min(
y),max(y),1000),'linear');
%去掉模型外部分的值
for i=1:2000
for j=1:1000
if sqrt((i-1000)^2+(j-500)^2)<200
Z1(j,i)=NaN;
end
end
end
%绘制等高线图
figure('name','x 轴方向应变云图');
contourf(X,Y,Z1,'LineStyle','none') ;
```

```
colormap jet
hold on;
colorbar;
axis equal;
title('x 轴方向应变云图');
%%y 轴应变
%插值
[X,Y,Z2]=griddata(x,y,strain_node(2,:),linspace(min(x),max(x),2000)',linspace(min(
y),max(y),1000),'linear');
%去掉模型外部分的值
for i=1:2000
for j=1:1000
if sqrt((i-1000)^2+(j-500)^2)<200
Z2(j,i)=NaN;
end
end
end
%绘制等高线图
figure('name','y 轴方向应变云图');
contourf(X,Y,Z2,'LineStyle','none') ;
colormap jet
hold on;
colorbar;
axis equal;
title('y 轴方向应变云图');
%%切应变
[X,Y,Z3]=griddata(x,y,strain_node(3,:),linspace(min(x),max(x),2000)',linspace(min(
y),max(y),1000),'linear');
%去掉模型外部分的值
```

```
for i=1:2000
for j=1:1000
if sqrt((i-1000)^2+(j-500)^2)<200
Z3(j,i)=NaN;
end
end
end
%绘制等高线图
figure('name','切应变云图');
contourf(X,Y,Z3,'LineStyle','none') ;
colormap jet
hold on;
colorbar;
axis equal;
title('切应变云图');
end
% 与软件对比求解误差
function fun_check(disp,stress,strain)
stress_node=stress';
strain_node=strain';
% 导入位移、应力、应变矩阵
u0=load('u_abaqus.txt');
[m,n]=size(u0);
for i=1:m
u(2*i-1, 1)=u0(i, 1);
u(2*i,1)=u0(i,2);
end
s= load('s_abaqus.txt');
E= load('E_abaqus.txt');
```

```
% 计算误差
d_u=disp-u; %位移差值
d_s=stress_node-s; %应力差值
d_E=strain_node-E; %应变差值
for i=1:2*m
eu(i,1)=abs(d_u(i,1)/disp(i,1));
end
for j=1:m
esx(j,1)=abs(d_s(j,1)/stress_node(j,1));
esy(j,1)=abs(d_s(j,2)/stress_node(j,2));
esxy(j,1)=abs(d_s(j,3)/stress_node(j,3));
eEx(j,1)=abs(d_E(j,1)/strain_node(j,1));
eEy(j,1)=abs(d_E(j,2)/strain_node(j,2));
eExy(j,1)=abs(d_E(j,3)/strain_node(j,3));
end
es=[esx;esy;esxy];
eE=[eEx;eEy;eExy];
% 计算误差均值
eu_total=mean(eu);
es_total=mean(es);
eE_total=mean(eE);
end
```

附录 B 三维悬臂梁分析 Matlab 程序

```
clc
clear
close all
%% 初始化
% 读取节点坐标
Nodes = xlsread('Nodes1.xlsx');
```

```
[N,~] = size(Nodes); %节点数
% 读取单元与节点之间的对应关系
Elems = xlsread('Elements1.xlsx');
[E,~] = size(Elems); %单元数
% 每个单元的节点数目
NE = 8;
% 读取材料信息
Mats = load('Materials.txt');%材料属性
ipstrn = 2;
nstrn = 3;
%% 定义力和边界节点
% 边界约束节点
j_dbc=1;
for (i=1:N)
if (Nodes(i,4)==0)
DBC(j_dbc,1)=Nodes(i,1);
DBC(j_dbc,2)=1;
DBC(j_dbc,3)=0;
j_dbc=j_dbc+1;
end
end
[P,~] = size(DBC);
% 载荷施加节点
j_nbc=1;
for (i=1:N)
if (Nodes(i,4)==60 && Nodes(i,3)==20)
right(j_nbc,1)=Nodes(i,1);
j_nbc=j_nbc+1;
end
```

```
end
j_nbc=1;
for i=1:E
for j=1:size(right(:,1))
for k=3:10
if Elems(i,k)==right(j,1)
el_list(j_nbc,1)=Elems(i,1);
el_list(j_nbc,2)=right(j,1);
j_nbc=j_nbc+1;
break
end
end
end
end
NBC(:,1)=unique(el_list(:,1));
j_nbc=1;
for i=1:2:size(el_list(:,1))
for j=3:10
if (el_list(i,2)==Elems(el_list(i,1),j))
NBC(j_nbc,2)=el_list(i,2);
NBC(j_nbc,3)=el_list(i+1,2);
j_nbc=j_nbc+1;
end
end
end
end
NBC(:,4)=2;
NBC(:,5)=1;
[Q,l] = size(NBC);
%% 计算刚度矩阵
```

```
% 总自由度
udof = 3; % 单个节点自由度
NDOF = N*udof;
% 矩阵初始化
K = zeros(NDOF,NDOF); % 刚度矩阵
U = zeros(NDOF,1); % 位移矩阵
F = zeros(NDOF,1); % 力矩阵
% 位移约束惩罚
Klarge = 10^8;
% 设置高斯点位置和权重
NG = 8; % 高斯点个数
[XG,WG] = Gauss_Points(NG);
% 遍历所有单元
for e = 1:E
% 建立单元和节点坐标的联系
Nnums = Elems(e,3:2+NE);
xyz = Nodes(Nnums(:),2:4);
% 提取单元弹性杨氏模量和泊松比
Y = Mats(Elems(e,2),2);
nu = Mats(Elems(e,2),3);
% 构建单元刚度矩阵
[Ke] = Stiff(ipstrn,xyz,Y,nu,udof,NE,NG,XG,WG);
% 组装整体刚度矩阵
ig = udof*(Nnums(:)-1);
for ni = 1:NE
i0 = udof*(ni-1);
for nj = 1:NE
j0 = udof*(nj-1);
for i = 1:udof
```

```
for j = 1:udof
K(ig(ni)+i,ig(nj)+j) = K(ig(ni)+i,ig(nj)+j) + Ke(i0+i,j0+j);
end
end
end
end
end
%% 定义力矢
NES = 2; %受力单元节点个数
NGS = 2; %受力单元高斯点个数
[XGS,WGS] = Gauss_Points_Surf(NGS);%加载节点高斯点坐标和权重
for q = 1:Q
in = zeros(NES);
tval = zeros(NES,1);
fval = zeros(NES,1);
% 确定加载的节点
e = NBC(q,1);%加载单元
in1 = NBC(q,2);%加载单元节点 1
in2 = NBC(q,3);%加载单元节点 2
idof = NBC(q,4);%加载方向
tval(:,1) = NBC(q,5);
for i=1:NGS
xi = XGS(i);
wgt = WGS(i);
NshapeS(1) = (1-xi)/2;
NshapeS(2) = (1+xi)/2;
DNshapeS(1) = -1/2;
DNshapeS(2) = +1/2;
xyS(1,1) = Nodes(in1,2);
```

```
xyS(1,2) = Nodes(in1,3);
xyS(1,3) = Nodes(in1,4);
xyS(2,1) = Nodes(in2,2);
xyS(2,2) = Nodes(in2,3);
xyS(2,3) = Nodes(in2,4);
[detJS] = Jacobian_Surf(NES,xi,xyS,DNshapeS);
fval = fval + wgt*NshapeS'*NshapeS*tval*detJS;
end
iloc1 = udof*(in1-1)+idof;
iloc2 = udof*(in2-1)+idof;
F(iloc1) = F(iloc1) + fval(1);
F(iloc2) = F(iloc2) + fval(2);
end
%% 定义边界条件
for p = 1:P
inode = DBC(p,1);
idof1 =1;
idof2 =2;
idof3 =3;
idiag1 = udof*(inode-1) + idof1;
idiag2 = udof*(inode-1) + idof2;
idiag3 = udof*(inode-1) + idof3;
K(idiag1,idiag1) = Klarge;
K(idiag2,idiag2) = Klarge;
K(idiag3,idiag3) = Klarge;
F(idiag1) = Klarge*DBC(p,3);
F(idiag2) = Klarge*DBC(p,3);
F(idiag3) = Klarge*DBC(p,3);
end
```

```
F=F/sum(F);
%
%F = F*5;
F(956)=0.1;
F(959)=0.1;
F(995)=0.2;
F(998)=0.2;
F(1001)=0.2;
F(1004)=0.2;
%}
%% 求解位移、应变和应力
U = inv(K)*F;
% 初始化位移、应变、应力
nedof = udof*NE;
Disp = zeros(E,nedof);
Eps = zeros(E,nstrn,NG);
Sig = zeros(E,nstrn,NG);
% 计算应变应力
for e = 1:E
Nnums = Elems(e,3:2+NE);
xyz = Nodes(Nnums(:),2:4);
h = Elems(e,3);
% 提取单元弹性杨氏模量和泊松比
Y = Mats(Elems(e,2),2);
nu = Mats(Elems(e,2),3);
% 提取位移
for i=1:NE
inode = Nnums(i);
iglb1 = udof*(inode-1)+1;
```

```
iglb2 = udof*inode;
iloc1 = udof*(i-1)+1;
iloc2 = udof*i;
Disp(e,iloc1) = U(iglb1);
Disp(e,iloc2) = U(iglb2);
end
% 应变、应力
u = Disp(e,:)';
[eps,sig] = Str(ipstrn,xyz,u,h,Y,nu,udof,NE,NG,XG);
Eps(e,:,:) = eps(:,:);
Sig(e,:,:) = sig(:,:);
end
%% 绘图
Plot_mesh(Nodes(:,2:4),Elems(:,3:10));
title('未施加载荷');
xlabel('X ');
ylabel('Y ');
zlabel('Z ');
j=1;
for i=1:3:size(U)
n_disp(j,1)=U(i);
n_disp(j,2)=U(i+1);
n_disp(j,3)=U(i+2);
j=j+1;
end
n_final(:,1)=Nodes(:,2)+n_disp(:,1);
n_final(:,2)=Nodes(:,3)+n_disp(:,2);
n_final(:,3)=Nodes(:,4)+n_disp(:,3);
figure;
```

```
Plot_mesh(n_final,Elems(:,3:10));
title('施加载荷');
xlabel('X ');
ylabel('Y ');
zlabel('Z ');
%% 子函数
% 计算高斯积分点等参坐标和权重
function [XG,WG] = Gauss_Points(NG)
alf = sqrt(1/3);
XG(1,1) = -alf;
XG(2,1) = +alf;
XG(3,1) = +alf;
XG(4,1) = -alf;
XG(5,1) = -alf;
XG(6,1) = +alf;
XG(7,1) = +alf;
XG(8,1) = -alf;
XG(1,2) = -alf;
XG(2,2) = -alf;
XG(3,2) = +alf;
XG(4,2) = +alf;
XG(5,2) = -alf;
XG(6,2) = -alf;
XG(7,2) = +alf;
XG(8,2) = +alf;
XG(1,3) = -alf;
XG(2,3) = -alf;
XG(3,3) = -alf;
XG(4,3) = -alf;
```

```
XG(5,3) = +alf;

XG(6,3) = +alf;

XG(7,3) = +alf;

XG(8,3) = +alf;

for i=1:NG

WG(i) = 1;

end

end

% 计算单元刚度矩阵

function [Ke] = Stiff(ipstrn,xyz,Y,nu,udof,NE,NG,XG,WG)

ndof = NE*udof;

nstrn = 6;

Ke = zeros(ndof,ndof);

for i=1:NG

xi = XG(i,1);

eta = XG(i,2);

mu = XG(i,3);

wgt = WG(i);

% 计算形函数对局部坐标的偏导

DNshape(1,1)=-((eta - 1)*(mu - 1))/8;

DNshape(2,1)= ((eta - 1)*(mu - 1))/8;

DNshape(3,1)=-((eta + 1)*(mu - 1))/8;

DNshape(4,1)= ((eta + 1)*(mu - 1))/8;

DNshape(5,1)= ((eta - 1)*(mu + 1))/8;

DNshape(6,1)=-((eta - 1)*(mu + 1))/8;

DNshape(7,1)= ((eta + 1)*(mu + 1))/8;

DNshape(8,1)=-((eta + 1)*(mu + 1))/8;

DNshape(1,2)=-(xi/8 - 1/8)*(mu - 1);

DNshape(2,2)= (xi/8 + 1/8)*(mu - 1);
```

```
DNshape(3,2)=-(xi/8 + 1/8)*(mu - 1);
DNshape(4,2)= (xi/8 - 1/8)*(mu - 1);
DNshape(5,2)= (xi/8 - 1/8)*(mu + 1);
DNshape(6,2)=-(xi/8 + 1/8)*(mu + 1);
DNshape(7,2)= (xi/8 + 1/8)*(mu + 1);
DNshape(8,2)=-(xi/8 - 1/8)*(mu + 1);
DNshape(1,3)=-(xi/8 - 1/8)*(eta - 1);
DNshape(2,3)= (xi/8 + 1/8)*(eta - 1);
DNshape(3,3)=-(xi/8 + 1/8)*(eta + 1);
DNshape(4,3)= (xi/8 - 1/8)*(eta + 1);
DNshape(5,3)= (xi/8 - 1/8)*(eta - 1);
DNshape(6,3)=-(xi/8 + 1/8)*(eta - 1);
DNshape(7,3)= (xi/8 + 1/8)*(eta + 1);
DNshape(8,3)=-(xi/8 - 1/8)*(eta + 1);
% 计算雅可比矩阵
Jac = zeros(3);
for ii=1:NE
Jac(1,1) = Jac(1,1) + DNshape(ii,1)*xyz(ii,1);
Jac(1,2) = Jac(1,2) + DNshape(ii,1)*xyz(ii,2);
Jac(1,3) = Jac(1,3) + DNshape(ii,1)*xyz(ii,3);
Jac(2,1) = Jac(2,1) + DNshape(ii,2)*xyz(ii,1);
Jac(2,2) = Jac(2,2) + DNshape(ii,2)*xyz(ii,2);
Jac(2,3) = Jac(2,3) + DNshape(ii,2)*xyz(ii,3);
Jac(3,1) = Jac(3,1) + DNshape(ii,3)*xyz(ii,1);
Jac(3,2) = Jac(3,2) + DNshape(ii,3)*xyz(ii,2);
Jac(3,3) = Jac(3,3) + DNshape(ii,3)*xyz(ii,3);
end
detJ = det(Jac);
Jhat = inv(Jac);
```

```
%计算 B 矩阵
B = zeros(nstrn,ndof);
i=1;
for j=1:NE
qx=DNshape(j,1)*Jhat(1,1)+DNshape(j,2)*Jhat(1,2)+DNshape(j,3)*Jhat(1,3);
qy=DNshape(j,1)*Jhat(2,1)+DNshape(j,2)*Jhat(2,2)+DNshape(j,3)*Jhat(2,3);
qz=DNshape(j,1)*Jhat(3,1)+DNshape(j,2)*Jhat(3,2)+DNshape(j,3)*Jhat(3,3);
B(1,i)=qx;
B(1,i+1)=0;
B(1,i+2)=0;
B(2,i)=0;
B(2,i+1)=qy;
B(2,i+2)=0;
B(3,i)=0;
B(3,i+1)=0;
B(3,i+2)=qz;
B(4,i)=qy;
B(4,i+1)=qx;
B(4,i+2)=0;
B(5,i)=0;
B(5,i+1)=qz;
B(5,i+2)=qy;
B(6,i)=qz;
B(6,i+1)=0;
B(6,i+2)=qx;
i=i+3;
end
% 计算三维弹性矩阵 C
lambda=nu*Y/((1+nu)*(1-2*nu));
```

```
c=Y/(2*(1+nu));
C = [lambda+2*nu lambda lambda 0 0 0; lambda lambda+2*nu lambda 0 0 0; lambda lambda
lambda+2*nu 0 0 0; 0 0 0 nu 0 0; 0 0 0 0 nu 0; 0 0 0 0 0 nu];
% 计算单元刚度矩阵
Ke = Ke + wgt*transpose(B)*C*B*detJ;
end
end
% 计算受力处高斯点坐标和权重
function [XGS,WGS] = Gauss_Points_Surf(NGS)
if (NGS == 2)
alf = sqrt(1/3);
XGS(1,1) = -alf;
XGS(2,1) = +alf;
WGS(1) = 1;
WGS(2) = 1;
elseif (NGS == 3)
alf = sqrt(3/5);
XGS(1,1) = -alf;
XGS(2,1) = 0;
XGS(3,1) = +alf;
WGS(1) = 5/9;
WGS(2) = 8/9;
WGS(3) = 5/9;
elseif (NGS == 4)
alf = 0.8611363115940526;
bet = 0.3399810435848563;
XGS(1,1) = -alf;
XGS(2,1) = -bet;
XGS(3,1) = bet;
```

```
XGS(4,1) = alf;
WGS(1)=0.3478548451374538;
WGS(2)=0.6521451548625461;
WGS(3)=0.6521451548625461;
WGS(4)=0.3478548451374538;
end
end
function [detJS] = Jacobian_Surf(NES,~,xyS,DNshapeS)
dxdxi = 0;
dydxi = 0;
dzdxi = 0;
for i=1:NES
dxdxi = dxdxi + DNshapeS(i)*xyS(i,1);
dydxi = dydxi + DNshapeS(i)*xyS(i,2);
dzdxi = dzdxi + DNshapeS(i)*xyS(i,3);
end
detJS = sqrt( dxdxi*dxdxi + dydxi*dydxi + dzdxi*dzdxi);
end
% 计算应力应变
function [eps,sig] = Str(ipstrn,xyz,u,h ,Y,nu,udof,NE,NG,XG)
ndof = NE*udof;
nstrn = 3;
eps = zeros(nstrn,NG);
sig = zeros(nstrn,NG);
for i=1:NG
xi = XG(i,1);
eta = XG(i,2);
mu = XG(i,3);
% 计算形函数对局部坐标的偏导
```

```
DNshape(1,1)=-((eta - 1)*(mu - 1))/8;
DNshape(2,1)= ((eta - 1)*(mu - 1))/8;
DNshape(3,1)=-((eta + 1)*(mu - 1))/8;
DNshape(4,1)= ((eta + 1)*(mu - 1))/8;
DNshape(5,1)= ((eta - 1)*(mu + 1))/8;
DNshape(6,1)=-((eta - 1)*(mu + 1))/8;
DNshape(7,1)= ((eta + 1)*(mu + 1))/8;
DNshape(8,1)=-((eta + 1)*(mu + 1))/8;
DNshape(1,2)=-(xi/8 - 1/8)*(mu - 1);
DNshape(2,2)= (xi/8 + 1/8)*(mu - 1);
DNshape(3,2)=-(xi/8 + 1/8)*(mu - 1);
DNshape(4,2)= (xi/8 - 1/8)*(mu - 1);
DNshape(5,2)= (xi/8 - 1/8)*(mu + 1);
DNshape(6,2)=-(xi/8 + 1/8)*(mu + 1);
DNshape(7,2)= (xi/8 + 1/8)*(mu + 1);
DNshape(8,2)=-(xi/8 - 1/8)*(mu + 1);
DNshape(1,3)=-(xi/8 - 1/8)*(eta - 1);
DNshape(2,3)= (xi/8 + 1/8)*(eta - 1);
DNshape(3,3)=-(xi/8 + 1/8)*(eta + 1);
DNshape(4,3)= (xi/8 - 1/8)*(eta + 1);
DNshape(5,3)= (xi/8 - 1/8)*(eta - 1);
DNshape(6,3)=-(xi/8 + 1/8)*(eta - 1);
DNshape(7,3)= (xi/8 + 1/8)*(eta + 1);
DNshape(8,3)=-(xi/8 - 1/8)*(eta + 1);
% 计算雅可比矩阵
Jac = zeros(3);
for ii=1:NE
Jac(1,1) = Jac(1,1) + DNshape(ii,1)*xyz(ii,1);
Jac(1,2) = Jac(1,2) + DNshape(ii,1)*xyz(ii,2);
```

```
Jac(1,3) = Jac(1,3) + DNshape(ii,1)*xyz(ii,3);
Jac(2,1) = Jac(2,1) + DNshape(ii,2)*xyz(ii,1);
Jac(2,2) = Jac(2,2) + DNshape(ii,2)*xyz(ii,2);
Jac(2,3) = Jac(2,3) + DNshape(ii,2)*xyz(ii,3);
Jac(3,1) = Jac(3,1) + DNshape(ii,3)*xyz(ii,1);
Jac(3,2) = Jac(3,2) + DNshape(ii,3)*xyz(ii,2);
Jac(3,3) = Jac(3,3) + DNshape(ii,3)*xyz(ii,3);
end
detJ = det(Jac);
Jhat = inv(Jac);
B = zeros(nstrn,ndof);
for j=1:NE
jloc1 = 2*(j-1)+1;
jloc2 = jloc1 + 1;
B(1,jloc1) = B(1,jloc1) + Jhat(1,1)*DNshape(j,1) ...
+ Jhat(1,2)*DNshape(j,2);
B(2,jloc2) = B(2,jloc2) + Jhat(2,1)*DNshape(j,1) ...
+ Jhat(2,2)*DNshape(j,2);
B(3,jloc1) = B(3,jloc1) + Jhat(2,1)*DNshape(j,1) ...
+ Jhat(2,2)*DNshape(j,2);
B(3,jloc2) = B(3,jloc2) + Jhat(1,1)*DNshape(j,1) ...
+ Jhat(1,2)*DNshape(j,2);
end
if (ipstrn == 1)
c = Y*(1-nu)/(1-2*nu)/(1+nu);
C = c*[ 1 nu/(1-nu) 0; nu/(1-nu) 1 0; 0 0 (1-2*nu)/(1-nu)/2 ];
else
c = Y/(1-nu)/(1+nu);
C = c*[ 1 nu 0; nu 1 0; 0 0 (1-nu)/2 ];
```

```
end
eps(:,i) = B*u;
sig(:,i) = C*eps(:,i);
end
end
% 绘图函数
function Plot_mesh(node_coord,elements)
n_el = length(elements) ; % 单元数
node_face = [1 2 6 5; 2 3 7 6; 3 4 8 7; 4 1 5 8; 1 2 3 4; 5 6 7 8];
XYZ = cell(1,n_el) ;
for e=1:n_el
nd=elements(e,:);
XYZ{e} = [node_coord(nd,1) node_coord(nd,2) node_coord(nd,3)] ;
end
% Plot
axis equal;
axis tight;
cellfun(@patch,repmat({'Vertices'},1,n_el),XYZ,repmat({'Faces'},1,n_el),repmat({no
de_face},1,n_el),repmat({'FaceColor'},1,n_el),repmat({'w'},1,n_el));
view(3)
end
```