

有限元方法及应用

|  |  |
| --- | --- |
| **题 目：** | **有限元大作业** |
| **班 级：** | **2305班** |
| **专 业：** | **机械工程** |
| **姓 名：** | **汤定进** |
| **学 号：** | **S230200194** |
| **指导教师：** | **王 琥** |
| **时 间：** | **2023.12.25** |

目录

[1. 问题描述 3](#_Toc157792356)

[1.1. 几何结构、边界条件、载荷条件、材料设置 3](#_Toc157792357)

[1.2. 所做工作 4](#_Toc157792358)

[2. Python-Code文件夹介绍 4](#_Toc157792359)

[2.1. main-FEM.py函数 4](#_Toc157792360)

[2.2. data文件夹 5](#_Toc157792361)

[2.3. result文件夹 5](#_Toc157792362)

[2.4. visualize文件夹 5](#_Toc157792363)

[2.5. bridge.inp文件 5](#_Toc157792364)

[2.6. 其他说明 5](#_Toc157792365)

[3. 八结点立方体单元介绍 6](#_Toc157792366)

[3.1. 单元节点 6](#_Toc157792367)

[3.2. 等参单元形函数 6](#_Toc157792368)

[3.3. 刚度矩阵 7](#_Toc157792369)

[3.4. 后处理 7](#_Toc157792370)

[4. 计算结果对比 8](#_Toc157792371)

[4.1. 线性二积分点C3D8单元 8](#_Toc157792372)

[4.2. 代码展示 14](#_Toc157792373)

# 问题描述

## 几何结构、边界条件、载荷条件、材料设置

赵州桥是世界上现存年代久远、跨度最大、保存最完整的单孔坦弧敞肩石拱桥，其建造工艺独特，在世界桥梁史上首创“敞肩拱”结构形式。本作业对赵州桥类似结构进行简化建模，分析桥底约束、桥面受力下的桥体应力、应变云图。

对于图1所示的赵州桥拱形结构，底部长度400mm，顶部长度500mm，最高点高度为100mm，宽50mm。划分图2所示网格细分5.5mm的C3D8立方体八节点单元，得到共计9162个单元，11620个节点。对上表面930个节点施加[0, -10000/930N, 0]集中力；对下表面180个节点施加PINNed约束，限制节点[U1 U2 U3]自由度，如图3、4所示。弹性模量：2.1e11Pa，泊松比0.3，忽略重力影响，分析该结构位移、应力大小及其分布，共计34860个自由度。

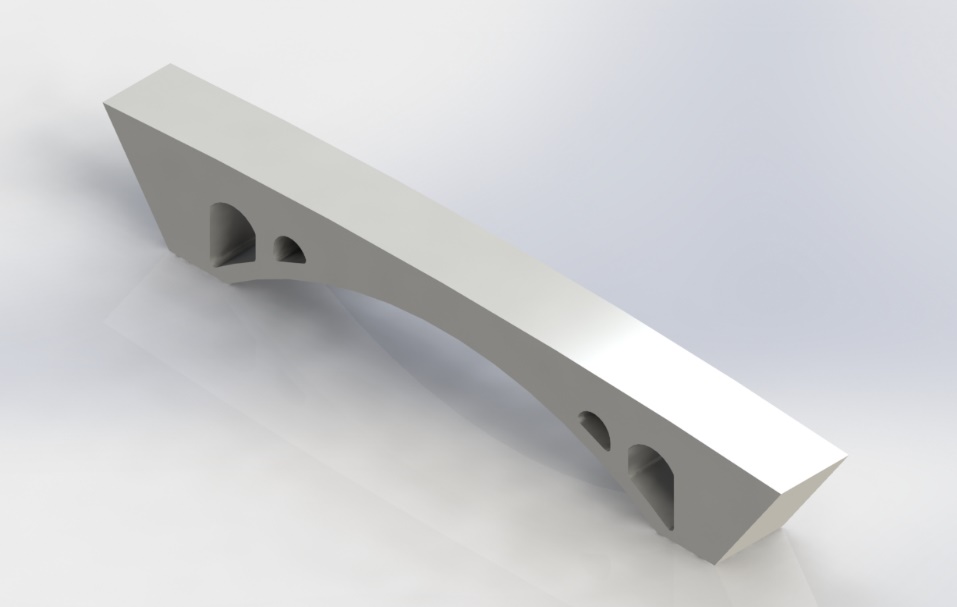


图 1 拱形待分析模型

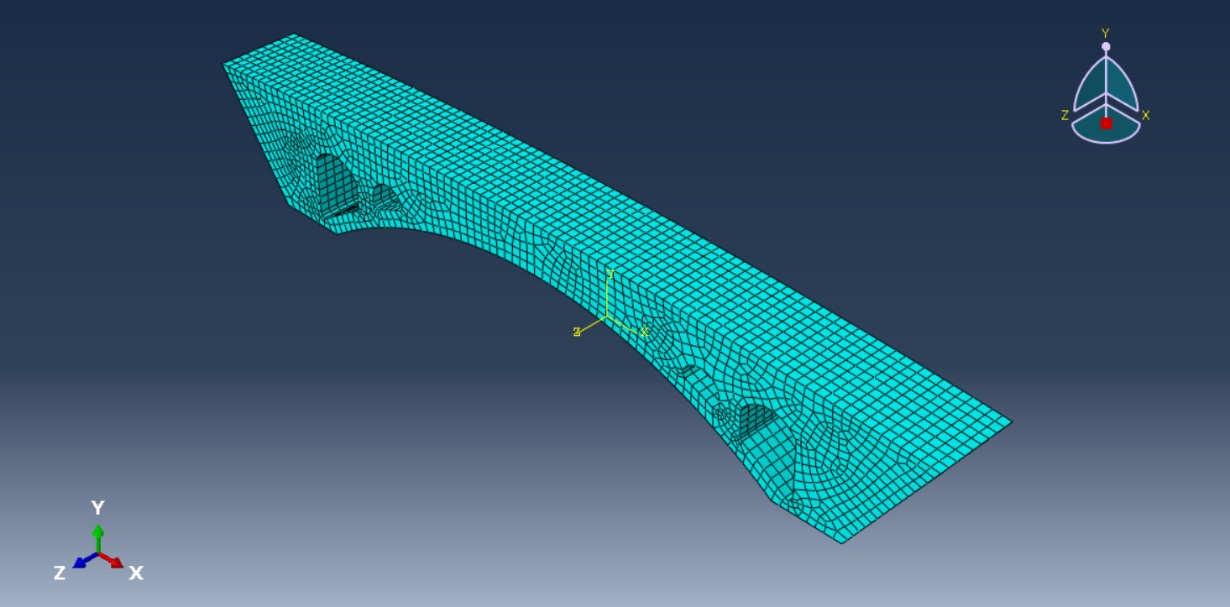


图 2 以5.5网格密度划分网格

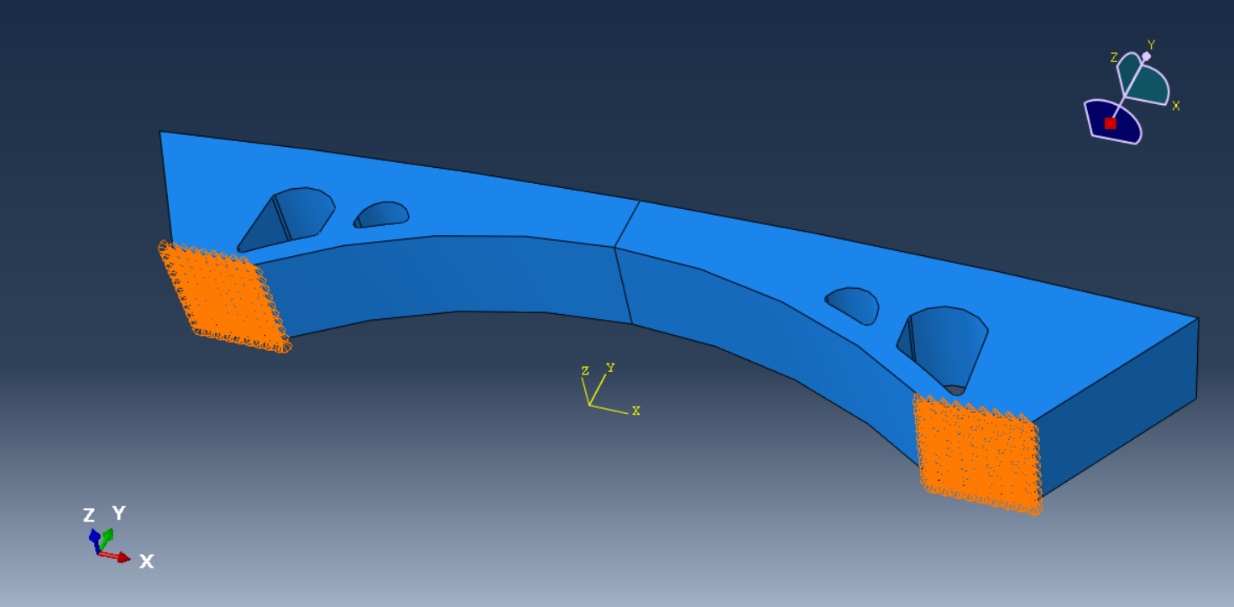


图 3 底部约束

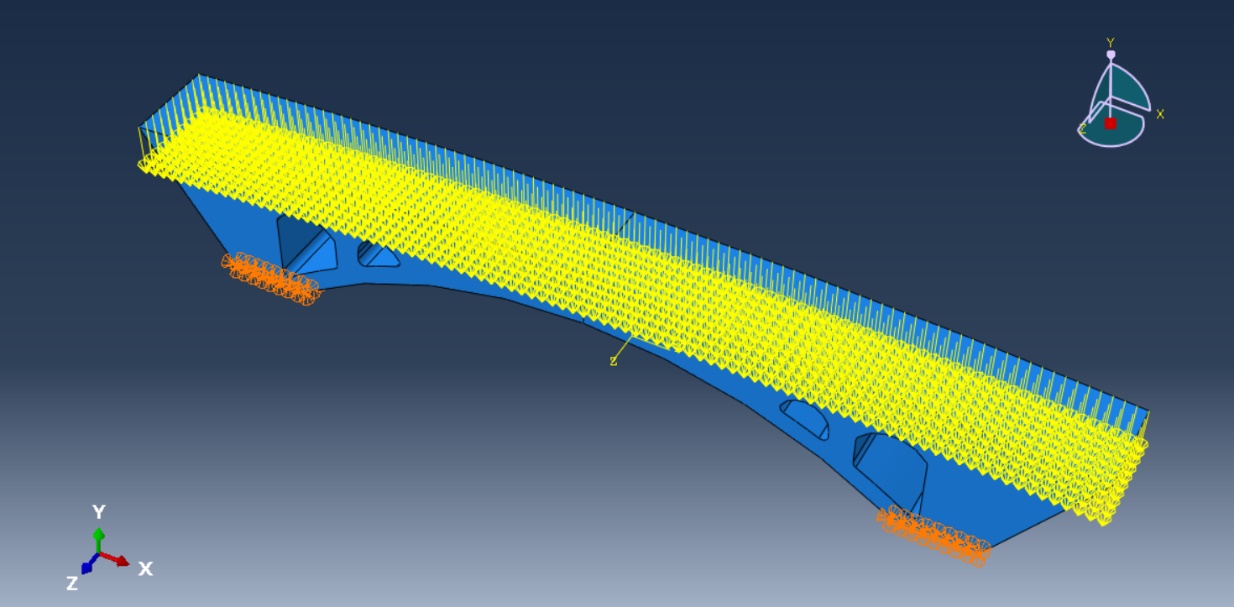


图 4 顶部集中力节点

## 所做工作

1. 利用pycharm软件，使用2个高斯积分点进行有限元分析，完成后处理；
2. 利用pycharm软件，使用线性八节点四面体单元C3D8进行有限元建模；
3. 利用pycharm软件，提取Abaqus作业文件.inp中的节点坐标及单元节点；通过VTK库，生成.vtk文件；
4. 在ParaView中打开.vtk文件，将Abaqus分析结果与pycharm计算结果进行对比。

# Python-Code文件夹介绍

## main-FEM.py函数

1. main\_FEM.py：作业主函数，主程序，读取节点、单元、边界节点、载荷节点，求解位移、应力分量，包含以下所有函数；
2. cal\_k\_matrix：获得六面体单元刚度矩阵K；形函数矩阵B，用于后处理；
3. cal\_b\_matrix：形函数矩阵B，被C3D8\_K.m调用；
4. cal\_d\_matrix：返回二维、三维问题的D矩阵；
5. gauss\_legendre\_1D：返回一维高斯积分点坐标及权重；
6. gauss\_legendre\_3D：返回三维高斯积分点坐标及权重；
7. load\_apply：施加节点力，生成F矩阵；
8. poly\_：后处理中将坐标转化为差值多项式；
9. FEDataModel：非结构化网格vtk类。

## data文件夹

1. Boundary\_Nodes.txt：存储施加PINNed边界条件节点编号；
2. Elements.txt：所有C3D8单元序号及对应八个节点编号；
3. Load\_Nodes.txt：载荷节点编号；
4. Nodes.txt：节点编号及坐标；

## result文件夹

1. Inp+S11.txt：有限元节点，S11应力值，其余类推；
2. Inp+U1.txt：有限元节点，U1方向位移值，其余类推。

## visualize文件夹

1. Inp+S11.vtk：有限元模型S11应力云图vtk文件，其余类推；
2. Inp+U1.vtk：有限元模型U1应力云图vtk文件，其余类推。

## bridge.inp文件

1. 预先在Abaqus软件中按第一节，定义集合、边界、载荷，输出.inp文件。
2. 底部约束节点为Cast集合，顶部施力节点为Load集合。

## 其他说明

对于不同的计算单元，上述输入、输出文件内容、格式大同小异，根据作业内容稍有变更。ParaView打开.vtk文件时，选择的Reader为XML Unstructured Grid Reader。

# 八结点立方体单元介绍

## 单元节点

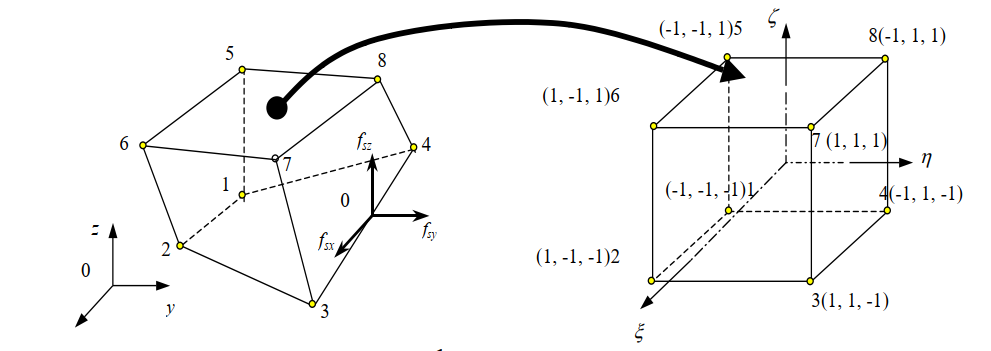


图 5 八结点立方体等参单元

## 等参单元形函数





(*ξi*, *ηi*, *ξi*)为等参单元中点对应的节点自然坐标，(*x*, *y*, *z*)为实体单元中的物理坐标。由此可得：



假设节点位移矩阵：



根据式：



得单元应变矩阵，该矩阵中的偏导数项根据下式求解：







## 刚度矩阵

刚度矩阵表达式为：



对积分区域为[-1 1]的被积函数，使用高斯积分点近似求积。



## 后处理

根据式：可以计算出积分点应力，对节点应力需要进行自然坐标插值，其差值多项式为：



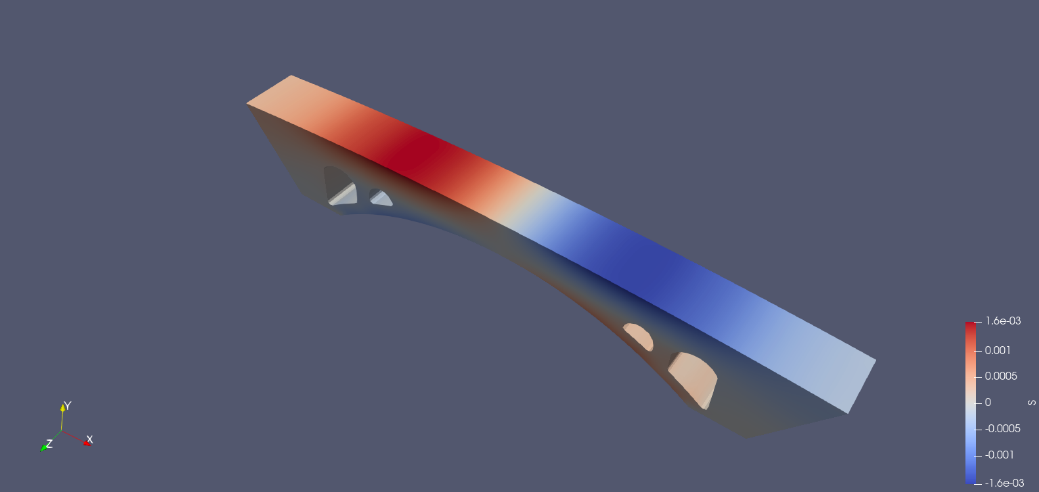
对于多个单元的公共节点应力，进行取平均值处理。

# 计算结果对比

所有图均为：左侧为作业计算结果，右侧为Abaqus同网格计算结果

## 线性二积分点C3D8单元

1. 位移对比



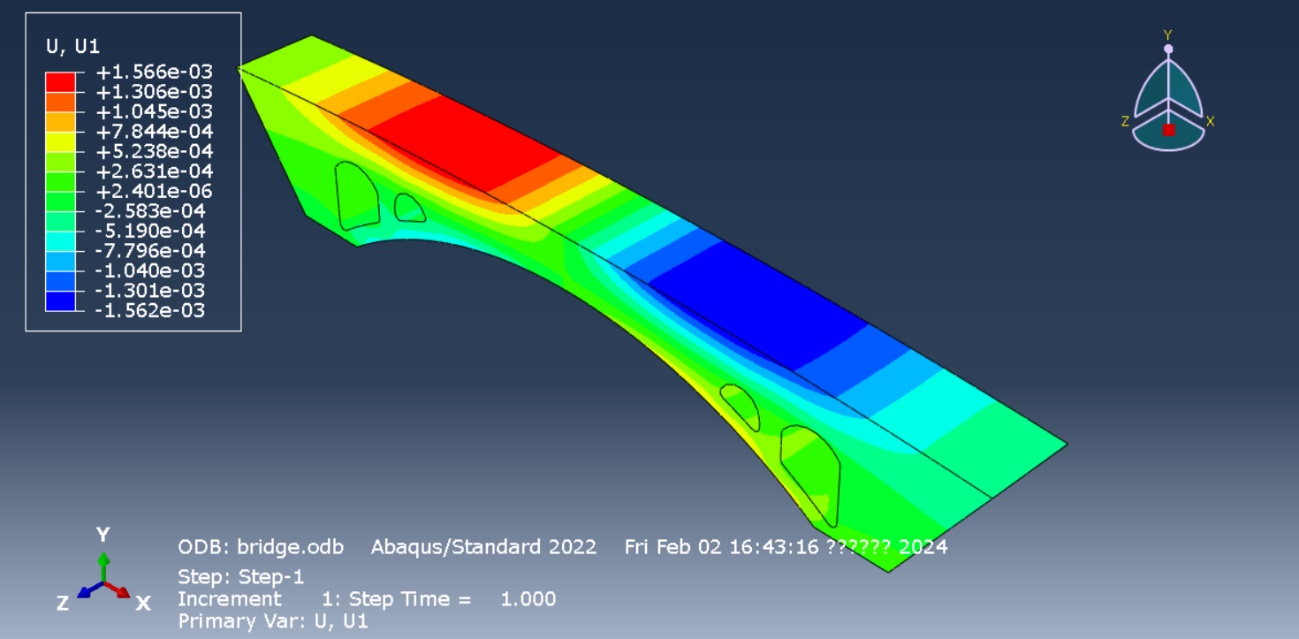
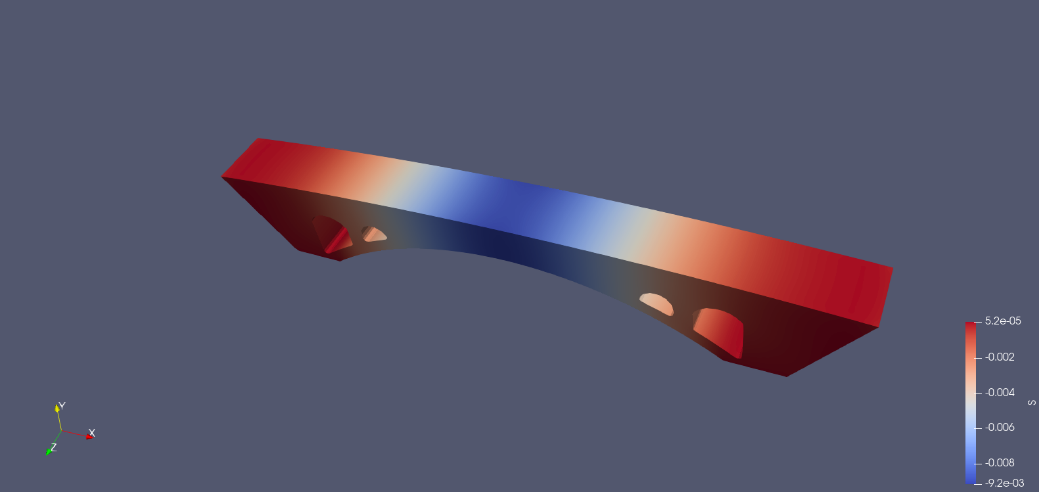


图 6 U1



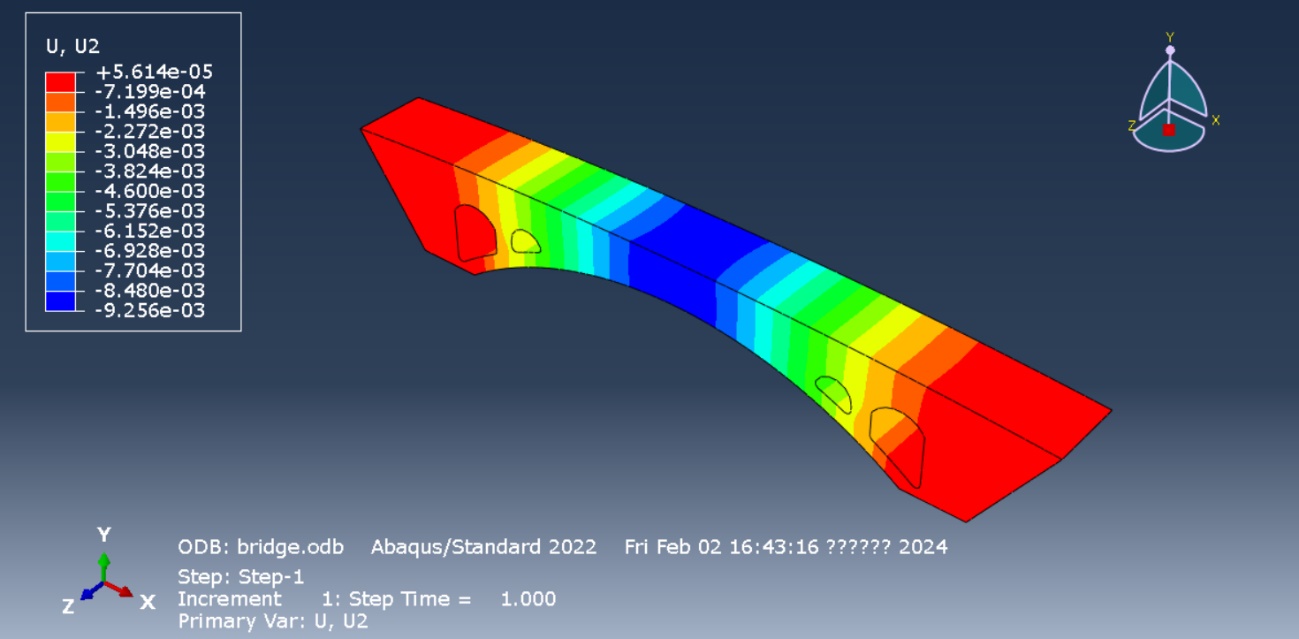
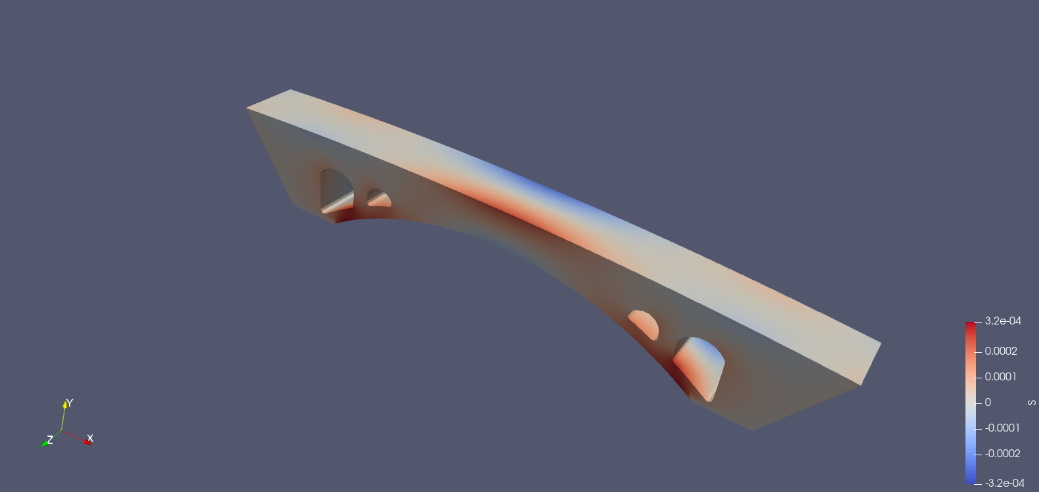


图 7 U2

****

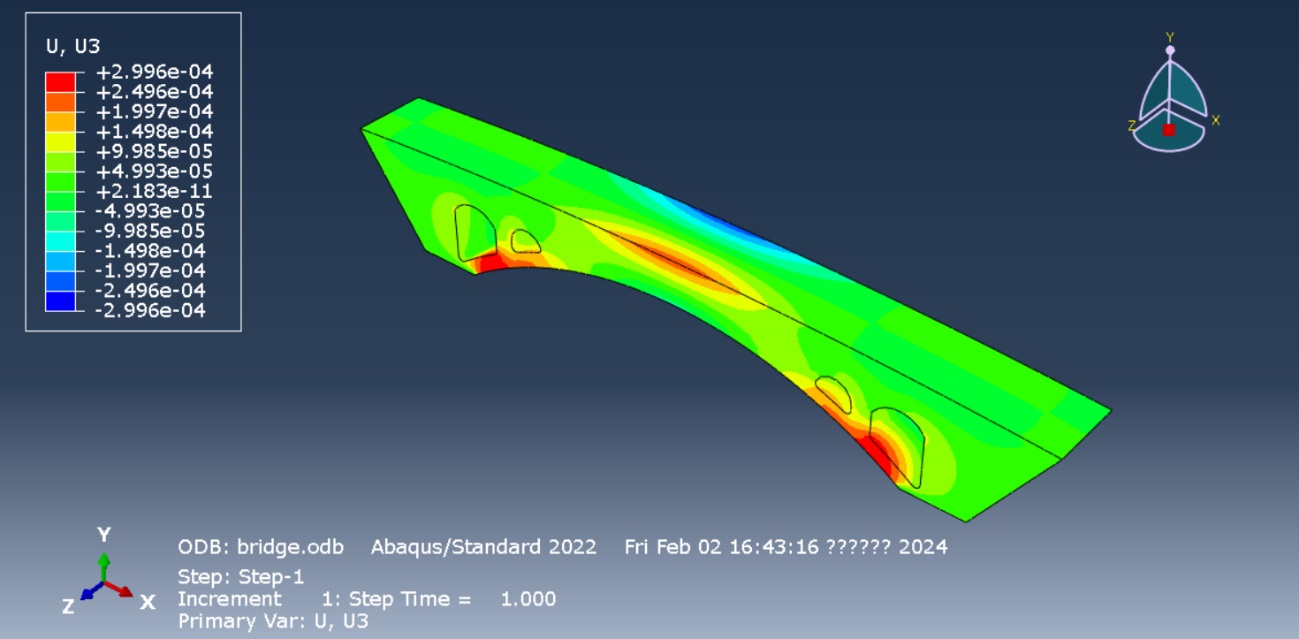
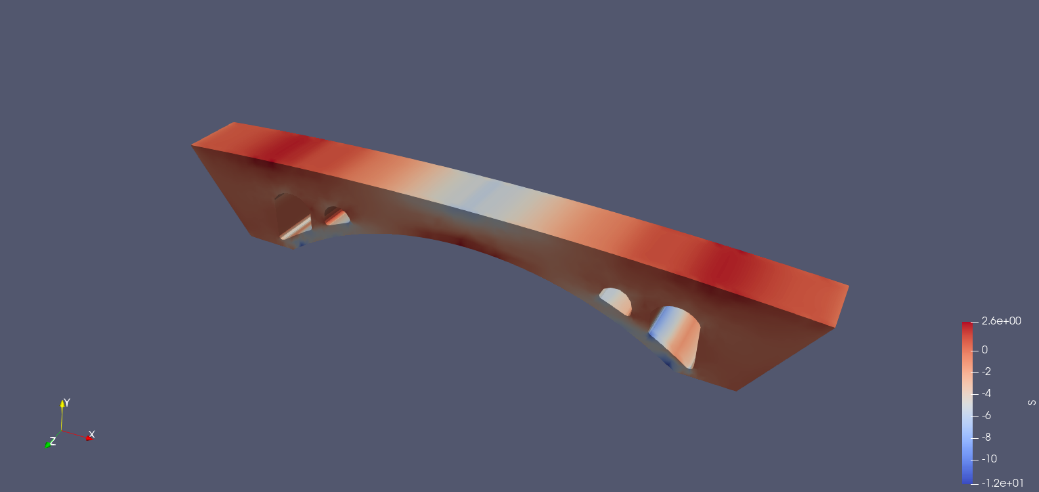
****

图 8 U3

1. 应力对比

****

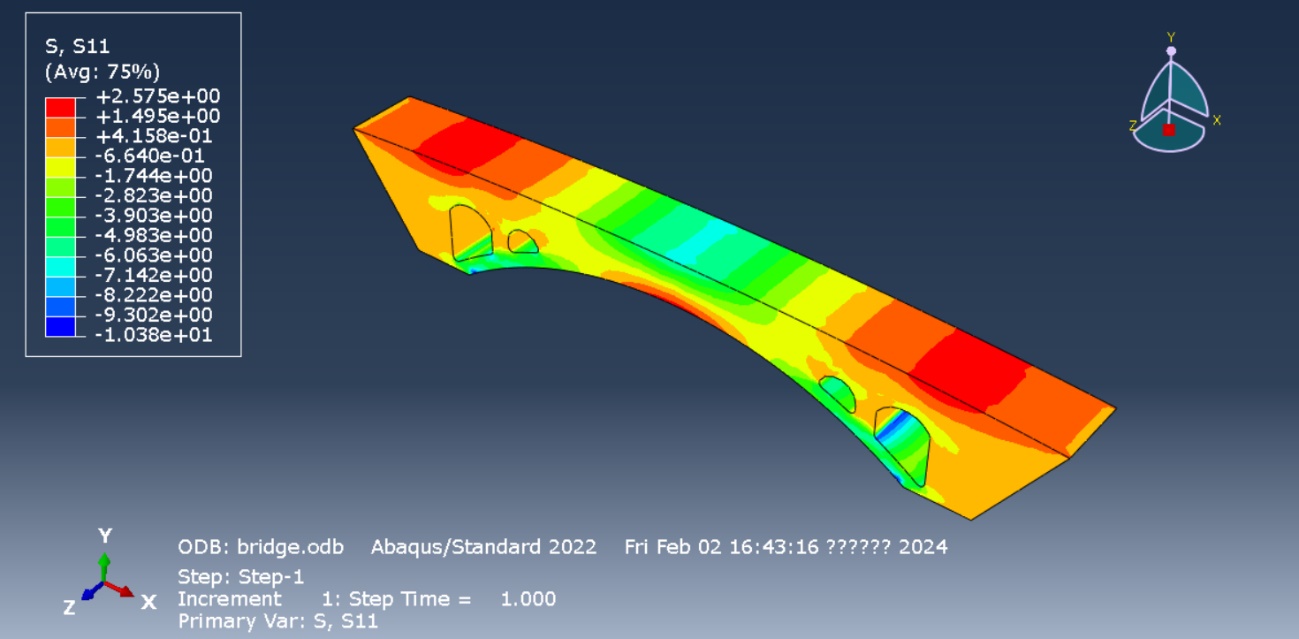
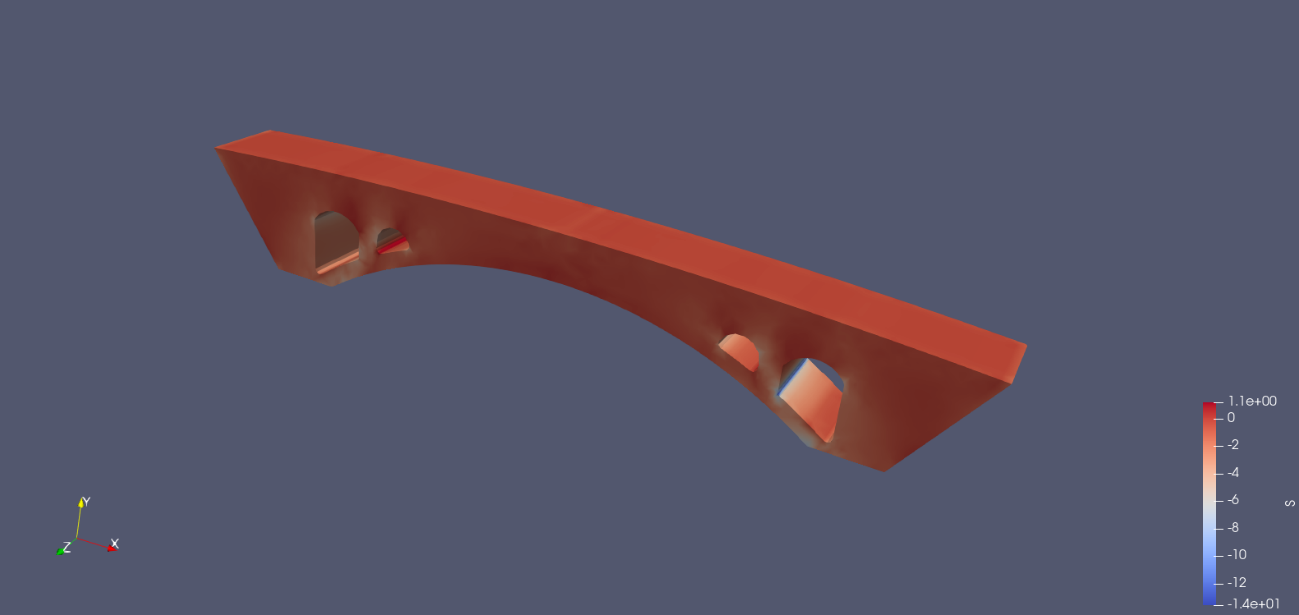
****

图 9 S11

****

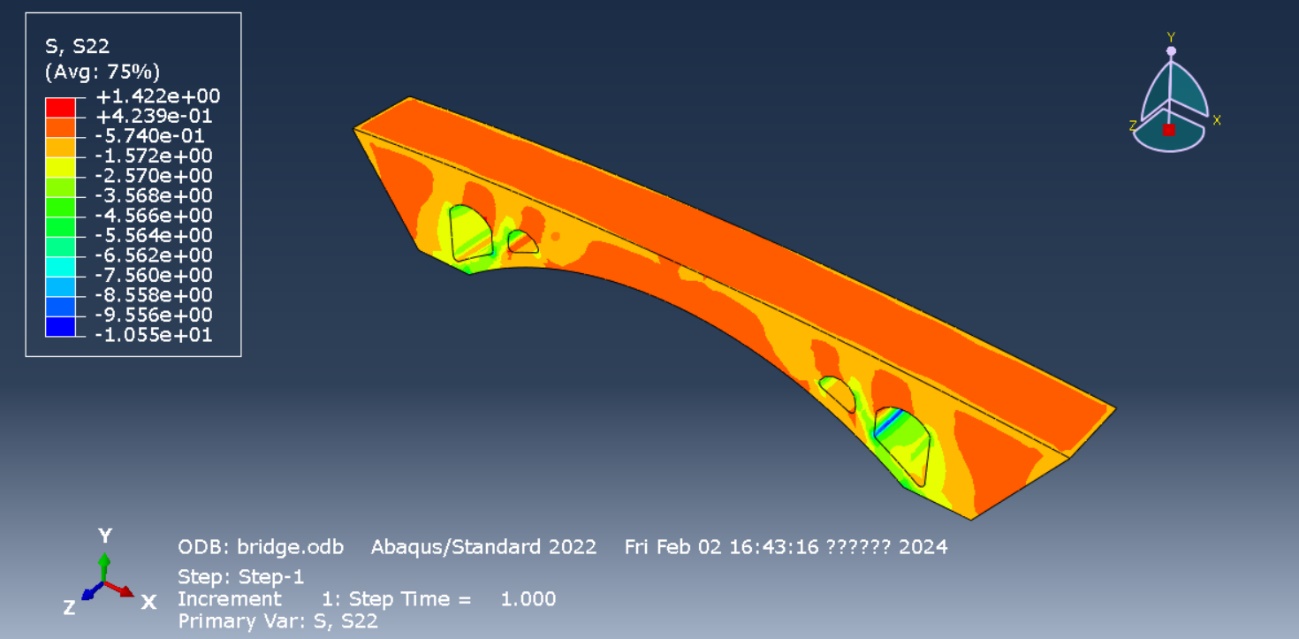
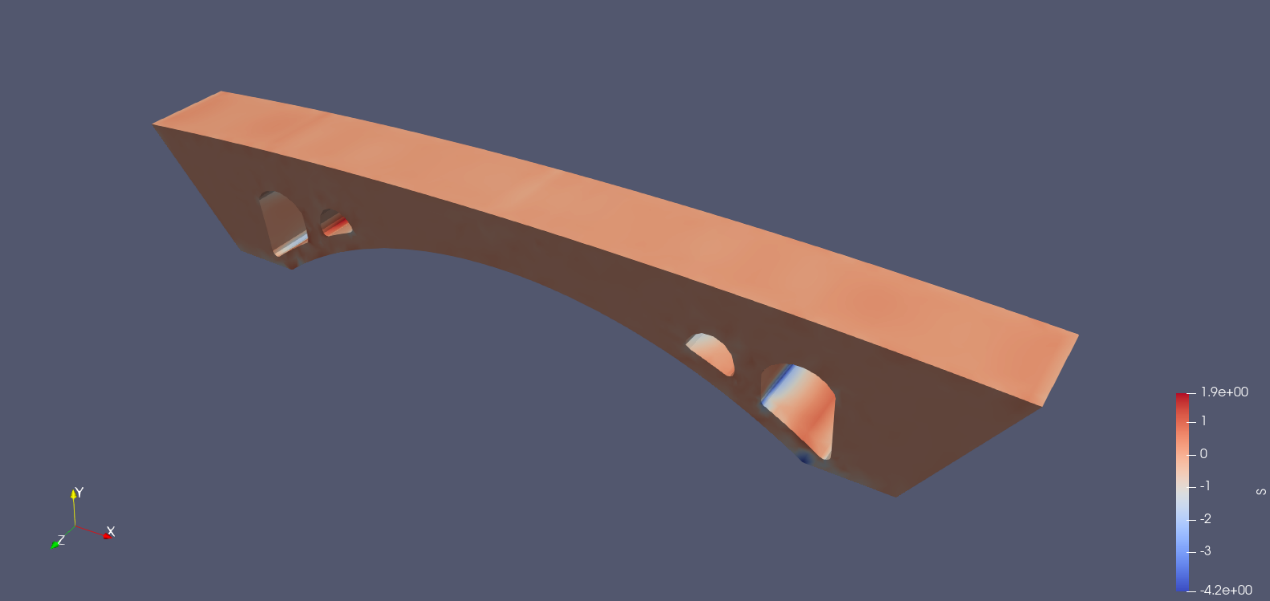
****

图 10 S22

****

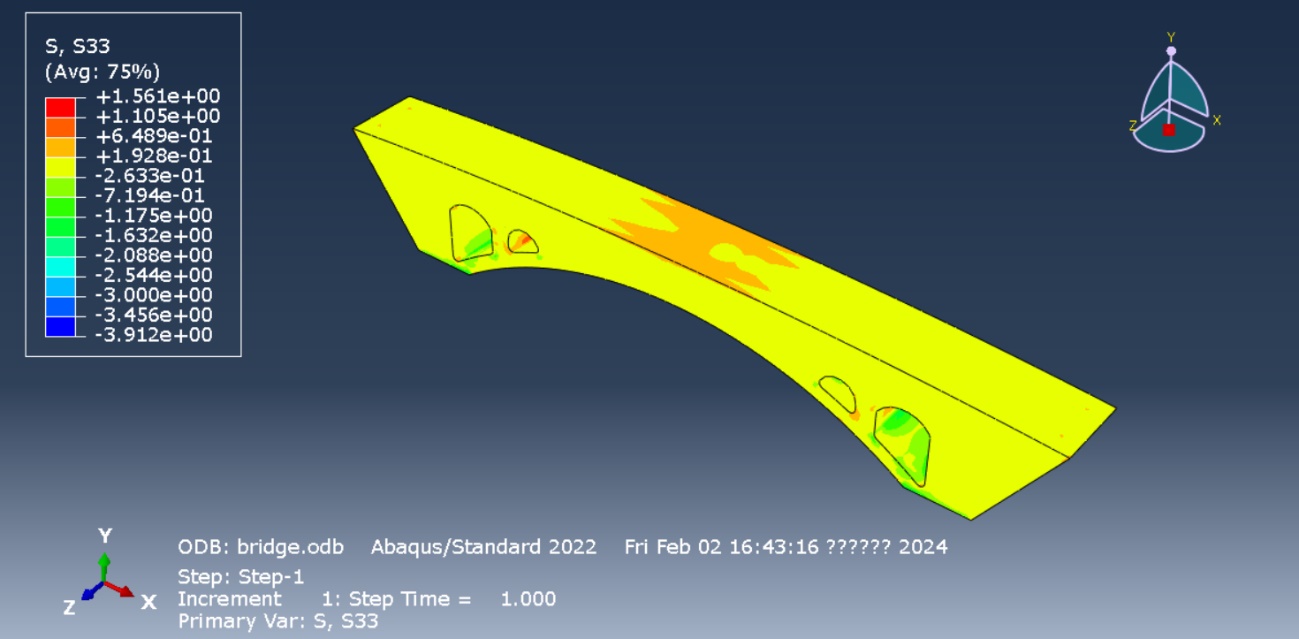
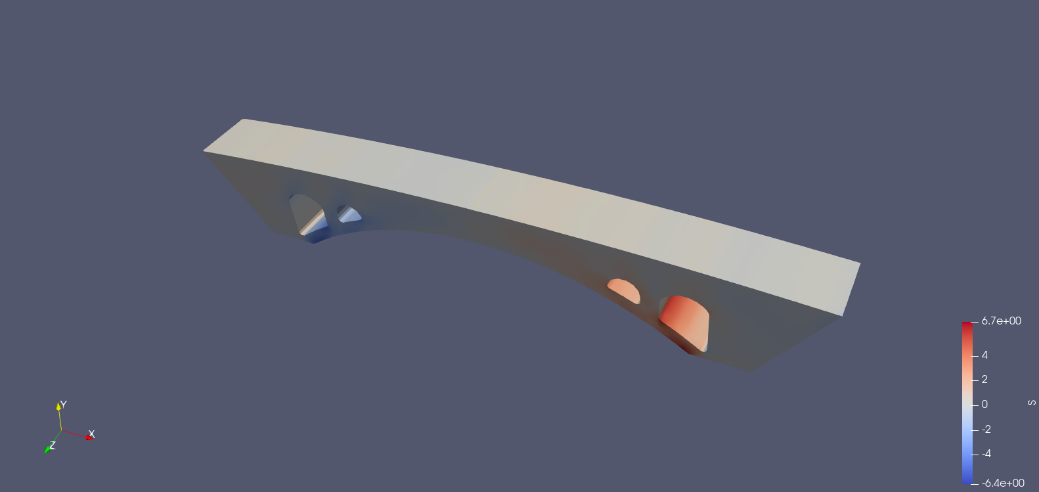
****

图 11 S33

****

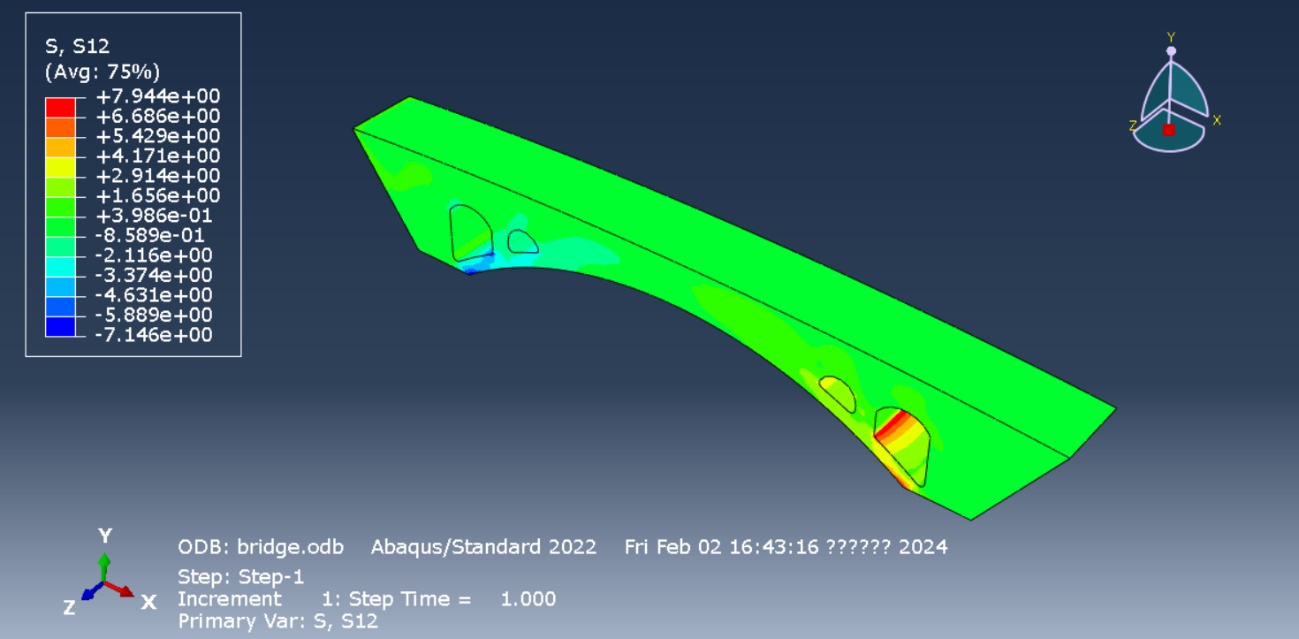
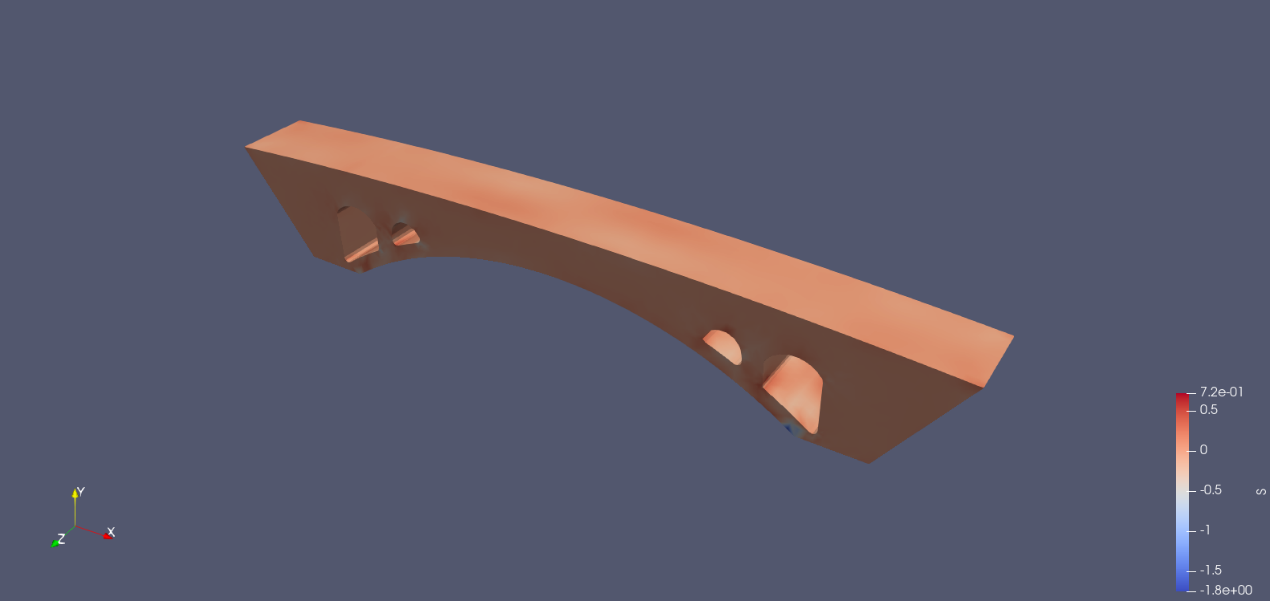
****

图 12 S12

****

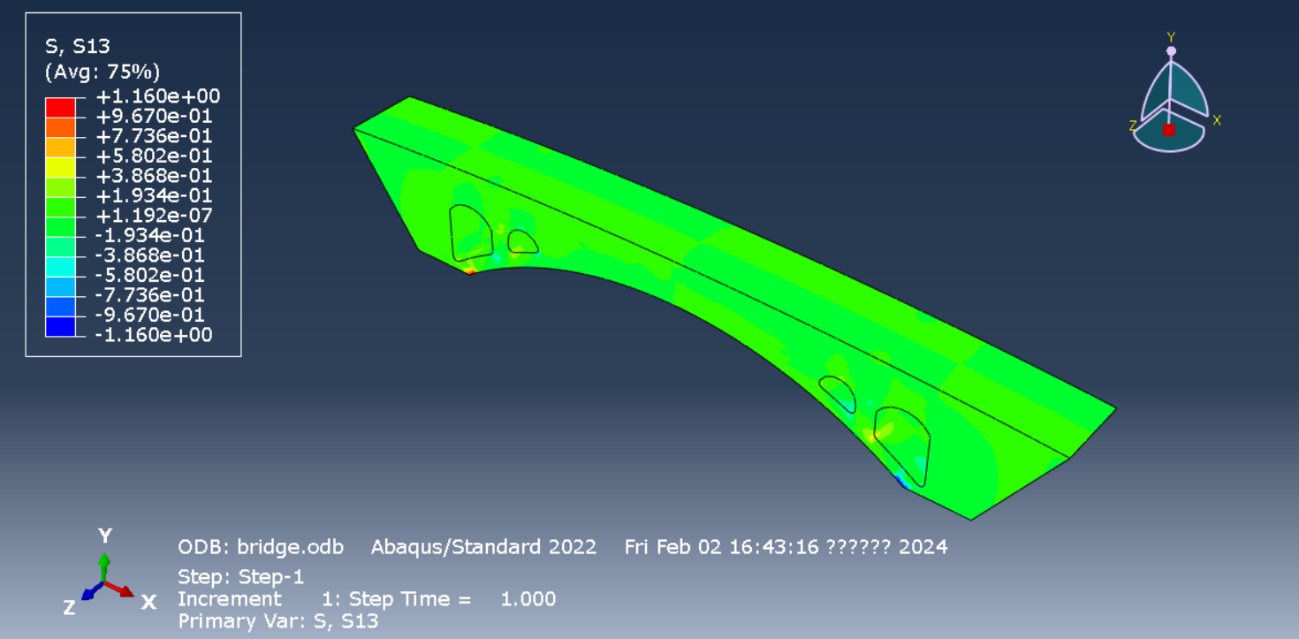
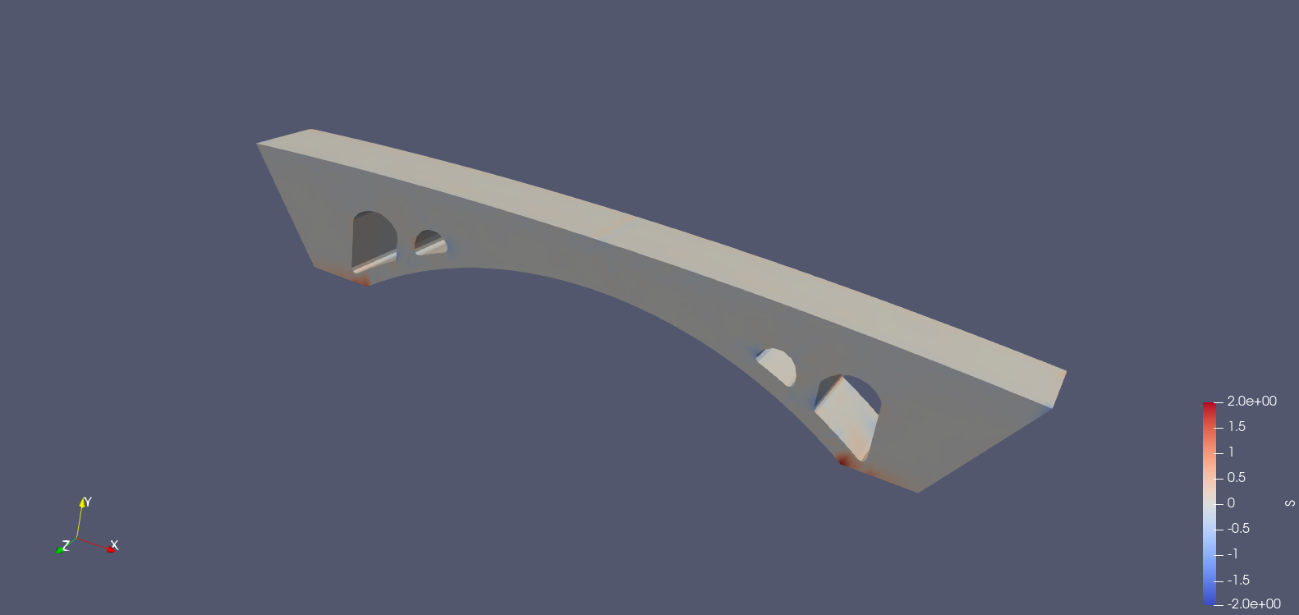
****

图 13 S13

****

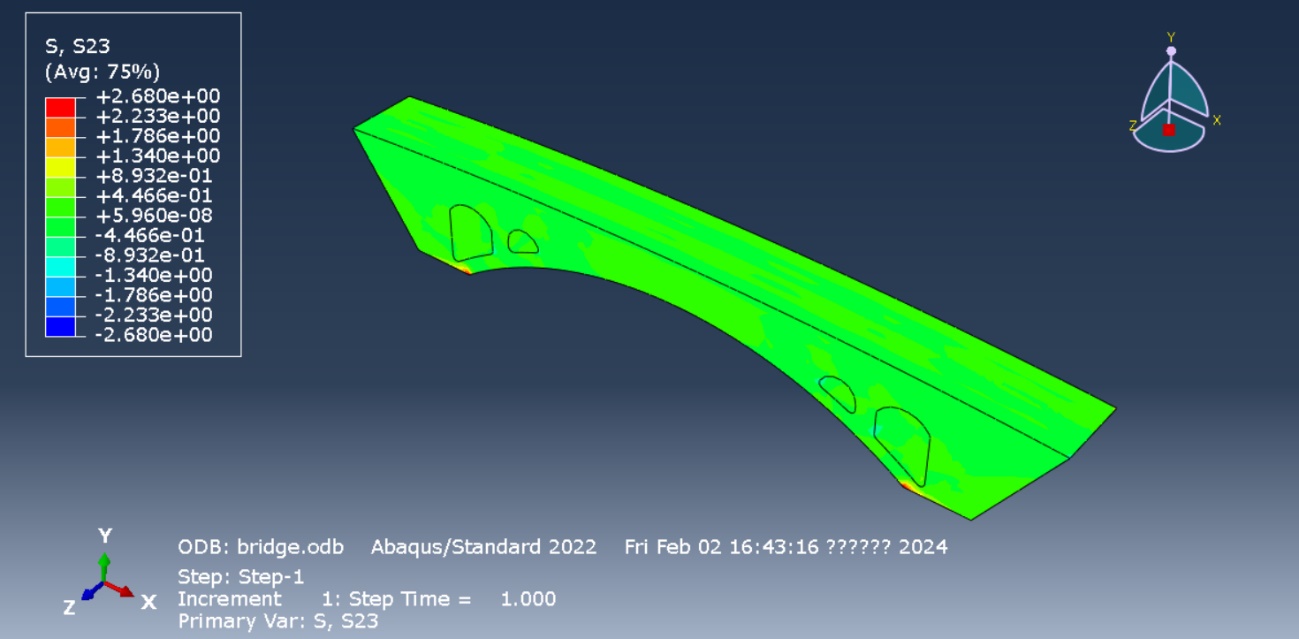
****

图 14 S23

1. 位移误差对比
2. 作业计算值：

U1最大位移：1.556e-3 U1最小位移：-1.553e-3

U2最大位移：5.169e-5 U2最小位移：-9.186e-3

U3最大位移：3.184e-4 U3最小位移：-3.184e-4

Abauqs计算值：

U1最大位移：1.566e-3 U1最小位移：-1.562e-3

U2最大位移：5.614e-5 U2最小位移：-9.256e-3

U3最大位移：2.996e-4 U3最小位移：-2.996e-4

取绝对值后误差：

U1：-0.639% U1：-0.557%

U2：-7.927% U2：-0.756%

U3：6.275% U3：6.275%

计算结果显示，在数量级为e-3时，位移计算相对误差在0.5~0.8%之间；在数量级为e-5次方时，位移计算相对误差在6~7%之间，相对误差扩大一个数量级，体现代码计算方向上的正确性。

## 代码展示

由于使用函数不多，所有代码均放在main-FEM.py文件中。

import numpy as np

import matplotlib.pyplot as plt

import time

import vtkmodules.all as vtk

# %%

inp\_file = input("请输入inp文件名")

# 创建文件

inp\_f = open(inp\_file + ".inp", "r")

nodes\_f = open("data/Nodes.txt", "w+")

elements\_f = open("data/Elements.txt", "w+")

load\_nodes\_f = open('data/Load\_Nodes.txt', "w+")

boundary\_nodes\_f = open('data/Boundary\_Nodes.txt', "w+")

# 提取节点

line\_temp = inp\_f.readline()

while (line\_temp != "\*Node\n"):

line\_temp = inp\_f.readline()

line\_temp = inp\_f.readline()

while (line\_temp != "\*Element, type=C3D8\n"):

nodes\_f.write(line\_temp)

line\_temp = inp\_f.readline()

# 提取单元

while (line\_temp != "\*Element, type=C3D8\n"):

line\_temp = inp\_f.readline()

line\_temp = inp\_f.readline()

while (line\_temp != "\*Nset, nset=Set-6, generate\n"):

elements\_f.write(line\_temp)

line\_temp = inp\_f.readline()

# 提取约束点

while (line\_temp != "\*Nset, nset=Cast\n"):

line\_temp = inp\_f.readline()

line\_temp = inp\_f.readline()

while (line\_temp != "\*Nset, nset=Load\n"):

line\_temp = line\_temp.replace("\n", ",")

boundary\_nodes\_f.write(line\_temp)

line\_temp = inp\_f.readline()

# 提取加载点

while (line\_temp != "\*Nset, nset=Load\n"):

line\_temp = inp\_f.readline()

line\_temp = inp\_f.readline()

while (line\_temp != "\*\* Section: Steel-45\n"):

line\_temp = line\_temp.replace("\n", ",")

load\_nodes\_f.write(line\_temp)

line\_temp = inp\_f.readline()

# 保存文件

inp\_f.close()

nodes\_f.close()

elements\_f.close()

load\_nodes\_f.close()

boundary\_nodes\_f.close()

# %%

# 返回一维高斯勒让德积分点

def gauss\_legendre\_1D(ngl):

"""

函数说明：一维返回高斯积分点

参数说明: ngl (int): 积分点数量，范围从 1 到 5。

返回值: point (numpy.ndarray): 积分点坐标；

weight (numpy.ndarray): 积分点权重。

"""

point = np.zeros(ngl)

weight = np.zeros(ngl)

if ngl == 1:

point[0] = 0

weight[0] = 2

elif ngl == 2:

point[:] = [-0.577350269189626, 0.577350269189626]

weight[:] = 1

elif ngl == 3:

point[0] = [-0.774596669241483, 0, 0.774596669241483]

weight[:] = [0.55555555, 0.88888888, 0.55555555]

elif ngl == 4:

point[0] = [-0.861136311594053, -0.339981043584856, 0.339981043584856, 0.861136311594053]

weight[:] = [0.347854845137454, 0.652145154862546, 0.652145154862546, 0.347854845137454]

elif ngl == 5:

point[0] = [-0.906179845938664, -0.538469310105683, 0, 0.538469310105683, 0.906179845938664]

weight[:] = [0.236926885056189, 0.478628670499366, 0.568888888888889, 0.478628670499366, 0.236926885056189]

return point, weight

# 返回三维高斯勒让德积分点

def gauss\_legendre\_3D(nglx, ngly, nglz):

"""

函数说明：计算三维 Gauss-Legendre 积分点和权重。

参数说明:

nglx (int): x 方向的积分点数量；

ngly (int): y 方向的积分点数量；

nglz (int): z 方向的积分点数量。

返回值: point (numpy.ndarray): 三维积分点坐标。

weight (numpy.ndarray): 三维积分点权重。

"""

if nglx > ngly:

if nglx > nglz:

ngl = nglx

else:

ngl = nglz

else:

if ngly > nglz:

ngl = ngly

else:

ngl = nglz

point = np.zeros((ngl, 3))

weight = np.zeros((ngl, 3))

pointx, weightx = gauss\_legendre\_1D(nglx)

pointy, weighty = gauss\_legendre\_1D(ngly)

pointz, weightz = gauss\_legendre\_1D(nglz)

for intx in range(nglx):

point[intx, 0] = pointx[intx]

weight[intx, 0] = weightx[intx]

for inty in range(ngly):

point[inty, 1] = pointy[inty]

weight[inty, 1] = weighty[inty]

for intz in range(nglz):

point[intz, 2] = pointz[intz]

weight[intz, 2] = weightz[intz]

return point, weight

# 返回b矩阵

def cal\_b\_matrix(x, y, z, eight\_nodes\_coordinates):

"""

函数说明：计算 3D8N 元素在自然坐标(x, y, z)处的 B 矩阵和雅可比行列式。

参数说明:

x (float): 自然坐标 s。

y (float): 自然坐标 t。

z (float): 自然坐标 c。

eight\_nodes\_coordinates (numpy.ndarray): 八个节点的坐标矩阵。

返回值: b (numpy.ndarray): B 矩阵。

detjacob (float): 雅可比行列式。

"""

n\_diff = np.array([[-(1 - y) \* (1 - z) / 8, (1 - y) \* (1 - z) / 8, (1 + y) \* (1 - z) / 8, -(1 + y) \* (1 - z) / 8,

-(1 - y) \* (1 + z) / 8, (1 - y) \* (1 + z) / 8, (1 + y) \* (1 + z) / 8, -(1 + y) \* (1 + z) / 8],

[(1 - x) \* -(1 - z) / 8, (1 + x) \* -(1 - z) / 8, (1 + x) \* (1 - z) / 8, (1 - x) \* (1 - z) / 8,

(1 - x) \* -(1 + z) / 8, (1 + x) \* -(1 + z) / 8, (1 + x) \* (1 + z) / 8, (1 - x) \* (1 + z) / 8],

[(1 - x) \* (1 - y) \* -1 / 8, (1 + x) \* (1 - y) \* -1 / 8, (1 + x) \* (1 + y) \* -1 / 8,

(1 - x) \* (1 + y) \* -1 / 8, (1 - x) \* (1 - y) / 8, (1 + x) \* (1 - y) / 8, (1 + x) \* (1 + y) / 8,

(1 - x) \* (1 + y) / 8]])

n\_J = np.dot(n\_diff, eight\_nodes\_coordinates)

detjacob = np.linalg.det(n\_J)

n\_diff = np.linalg.solve(n\_J, n\_diff)

Bs = np.zeros((6, 3, 8))

for i in range(8):

Bs[:, :, i] = np.array([[n\_diff[0, i], 0, 0],

[0, n\_diff[1, i], 0],

[0, 0, n\_diff[2, i]],

[0, n\_diff[2, i], n\_diff[1, i]],

[n\_diff[2, i], 0, n\_diff[0, i]],

[n\_diff[1, i], n\_diff[0, i], 0]])

b = np.hstack(

(Bs[:, :, 0], Bs[:, :, 1], Bs[:, :, 2], Bs[:, :, 3], Bs[:, :, 4], Bs[:, :, 5], Bs[:, :, 6], Bs[:, :, 7]))

return b, detjacob

# 返回d矩阵

def cal\_d\_matrix(iopt, elastic, poisson):

"""

函数说明：返回D矩阵

参数说明:

iopt (int): 分析类型.

1 - 平面应力

2 - 平面应变

3 - 堆对称分析

4 - 三维问题

elastic (float): 弹性模量

poisson (float): 泊松比

返回值:

D (numpy.ndarray): D矩阵

"""

if iopt == 1: # plane stress

d = elastic / (1 - poisson \* poisson) \* \

np.array([[1, poisson, 0],

[poisson, 1, 0],

[0, 0, (1 - poisson) / 2]])

elif iopt == 2: # plane strain

d = elastic / ((1 + poisson) \* (1 - 2 \* poisson)) \* \

np.array([[1 - poisson, poisson, 0],

[poisson, 1 - poisson, 0],

[0, 0, (1 - 2 \* poisson) / 2]])

elif iopt == 3: # axisymmetry

d = elastic / ((1 + poisson) \* (1 - 2 \* poisson)) \* \

np.array([[1 - poisson, poisson, poisson, 0],

[poisson, 1 - poisson, poisson, 0],

[poisson, poisson, 1 - poisson, 0],

[0, 0, 0, (1 - 2 \* poisson) / 2]])

else: # three-dimensional

d = elastic / ((1 + poisson) \* (1 - 2 \* poisson)) \* \

np.array([[1 - poisson, poisson, poisson, 0, 0, 0],

[poisson, 1 - poisson, poisson, 0, 0, 0],

[poisson, poisson, 1 - poisson, 0, 0, 0],

[0, 0, 0, (1 - 2 \* poisson) / 2, 0, 0],

[0, 0, 0, 0, (1 - 2 \* poisson) / 2, 0],

[0, 0, 0, 0, 0, (1 - 2 \* poisson) / 2]])

return d

# 返回b、k矩阵

def cal\_k\_matrix(D, eight\_nodes\_coordinates, integral\_nodes=[2, 2, 2]):

"""

函数说明：为C3D8单元返回B矩阵、K矩阵.

参数说明:

D: numpy.ndarray 分析使用的D矩阵.

eight\_nodes\_coordinates: numpy.ndarray八个节点坐标矩阵.

integral\_nodes: list, optional高斯积分点，默认值为 [2, 2, 2].

返回值:

k: numpy.ndarray单元刚度矩阵.

B: numpy.ndarray位移应变B矩阵.

"""

nglx, ngly, nglz = integral\_nodes

point3, weight3 = gauss\_legendre\_3D(nglx, ngly, nglz) # Assuming GLI\_PW3 is implemented elsewhere

k = np.zeros((24, 24))

B = np.zeros((6, 24, nglx \* ngly \* nglz))

time = 0

for intx in range(nglx):

x, wtx = point3[intx, 0], weight3[intx, 0]

for inty in range(ngly):

y, wty = point3[inty, 1], weight3[inty, 1]

for intz in range(nglz):

z, wtz = point3[intz, 2], weight3[intz, 2]

b, detjacob = cal\_b\_matrix(x, y, z, eight\_nodes\_coordinates) # Assuming D3N8\_B is implemented elsewhere

time += 1

B[:, :, time - 1] = b

k += np.dot(np.dot(b.T, D), b) \* wtx \* wty \* wtz \* detjacob

return k, B

# 施加边界条件

def load\_apply(Load\_nodes, Nodes\_num, Dof, Total\_Force):

"""

函数说您：将加载条件应用到力矩阵 F 上。

参数说明:

Load\_nodes (numpy.ndarray): 被加载的节点的索引。

Nodes\_num (int): 节点总数。

Dof (int): 每个节点的自由度。

Total\_Force (numpy.ndarray): 应用在加载节点上的总力。

返回值:

F (numpy.ndarray): 应用加载条件后的力矩阵。

"""

F = np.zeros(Dof \* Nodes\_num, dtype=np.float32)

Load\_nodes\_num = Load\_nodes.shape[1]

for i in range(Dof):

F[(Load\_nodes - 1) \* Dof + i] = Total\_Force[i] / Load\_nodes\_num

return F

# 后处理时，单元内插值项

def poly\_(x):

return [1, x[0], x[1], x[2], x[0] \* x[1], x[0] \* x[2], x[1] \* x[2], x[0] \* x[1] \* x[2]]

# %%

# 读取有限元模型数据

#

try:

f\_nodes = open('data/Nodes.txt')

f\_load\_nodes = open('data/Load\_Nodes.txt')

f\_boundary\_nodes = open('data/Boundary\_Nodes.txt')

f\_elements = open('data/Elements.txt')

nodes = np.array([np.array(node.replace(' ', '').replace('\n', '').split(','), dtype=np.float16) for node in

f\_nodes.readlines()])

elements = np.array([np.array(node.replace(' ', '').replace('\n', '').split(','), dtype=np.int16) for node in

f\_elements.readlines()])

boundary\_nodes = np.array(

[np.array(node.strip(',').replace(' ', '').replace('\n', '').split(','), dtype=np.int16) for node in

f\_boundary\_nodes.readlines()])

load\_nodes = np.array(

[np.array(node.strip(',').replace(' ', '').replace('\n', '').split(','), dtype=np.int16) for node in

f\_load\_nodes.readlines()])

except Exception as Err:

print(Err)

# 关闭读写文件

f\_nodes.close()

f\_load\_nodes.close()

f\_boundary\_nodes.close()

f\_elements.close()

# 每个节点三个自由度

dof = 3

# 节点总数

total\_nodes = nodes.shape[0]

# 单元总数

total\_elements = elements.shape[0]

# 总自由度

total\_dof = dof \* total\_nodes

# 施加力大小及方向

total\_force = [0, -10000, 0];

print('有限元模型中，共计{0}个节点，{1}个单元，施加集中力节点{2}个，边界节点{3}个'

.format(total\_nodes, total\_elements, load\_nodes.shape[1], boundary\_nodes.shape[1]))

# 设置材料弹性模量

elastic = 210000

# 设置材料泊松比

poisson = 0.3

# 设置分析问题为三维问题

iopt = 4

# 计算d矩阵

d = cal\_d\_matrix(iopt, elastic, poisson)

# 为K、B矩阵预分配空间

K = np.zeros((total\_dof, total\_dof), dtype=np.float64)

B = np.zeros((6, 24, 8, total\_elements), dtype=np.float64)

# 设置积分点数量

integral\_nodes = 2

# 组装刚度矩阵

for e\_index in range(total\_elements):

e\_n\_index = elements[e\_index, 1:] - 1

eight\_nodes\_matrix = nodes[e\_n\_index, 1:]

print('组装六面体单元，{0}个刚度矩阵'.format(e\_index + 1))

[k, b] = cal\_k\_matrix(d, eight\_nodes\_matrix, integral\_nodes \* np.array([1, 1, 1]))

B[:, :, :, e\_index] = b

for row in range(8):

row\_index = e\_n\_index[row] # 刚度矩阵行节点编号

for col in range(8):

col\_index = e\_n\_index[col] # 刚度矩阵列节点编号

K[3 \* row\_index:3 \* (row\_index + 1), 3 \* col\_index:3 \* (col\_index + 1)] += \

k[3 \* row:3 \* (row + 1), 3 \* col:3 \* (col + 1)]

# %%

# 施加载荷条件与边界条件

F = load\_apply(load\_nodes, total\_nodes, dof, total\_force)

Constrain\_dofs = np.zeros([boundary\_nodes.shape[1], 3], dtype=np.int16)

# 限制三个自由度

for i in range(dof):

Constrain\_dofs[:, i] = (boundary\_nodes - 1) \* dof + i

if dof == 1:

Constrain = Constrain\_dofs[:, 0]

elif dof == 2:

Constrain = np.concatenate((Constrain\_dofs[:, 0], Constrain\_dofs[:, 1]))

elif dof == 3:

Constrain = np.concatenate((Constrain\_dofs[:, 0], Constrain\_dofs[:, 1], Constrain\_dofs[:, 2]))

else:

raise ValueError('dof not in [1, 2, 3]')

# K\_constrain、F\_constrain为施加完约束的K、F矩阵

K\_constrain = np.copy(K)

F\_constrain = np.copy(F)

# 删除约束节点对应的行和列

K\_constrain = np.delete(K\_constrain, Constrain, axis=0)

K\_constrain = np.delete(K\_constrain, Constrain, axis=1)

F\_constrain = np.delete(F\_constrain, Constrain, axis=0)

print("开始解方程")

time\_start = time.time()

# KU=F，求解U

U\_ = np.linalg.solve(K\_constrain, F\_constrain)

print("解方程结束")

print("耗时{}".format(time.time() - time\_start))

##

U = U\_

# %%

# 考虑边界条件后重新构建完整的位移向量

for i in range(boundary\_nodes.shape[1]):

index = boundary\_nodes[0][i] - 1

forward\_ = U[:3 \* (index)]

backward\_ = U[3 \* index:]

U = np.concatenate((forward\_, [0, 0, 0], backward\_))

# 提取每个节点的位移

U1 = U[::3]

U2 = U[1::3]

U3 = U[2::3]

# %%

# %%后处理，获得高斯积分点位置

gauss, \_ = gauss\_legendre\_1D(integral\_nodes)

# 插值数量，即积分点数量

inter\_num = pow(integral\_nodes, 3)

# 插值点坐标，即单元积分点坐标

inter\_points = np.zeros((inter\_num, 3))

time = 0

for intx in range(integral\_nodes):

x\_temp = gauss[intx]

for inty in range(integral\_nodes):

y\_temp = gauss[inty]

for intz in range(integral\_nodes):

z\_temp = gauss[intz]

inter\_points[time, :] = [x\_temp, y\_temp, z\_temp]

time += 1

# 每个单元有八个点，使用[1, x, y, z, xy, xz, yx, xyz]差值，目前仅支持积分点为2的计算工作

X = np.zeros((inter\_num, inter\_num))

for i in range(inter\_num):

inter\_point = inter\_points[i, :]

X[i, :] = poly\_(inter\_points[i, :])

inv\_X = np.linalg.inv(X)

# 待插顶点

equal\_nodes = np.array([[-1, -1, -1],

[-1, -1, 1],

[-1, 1, -1],

[-1, 1, 1],

[1, -1, -1],

[1, -1, 1],

[1, 1, -1],

[1, 1, 1]])

# %%

S\_Elements = np.zeros((8, 6, total\_elements), dtype=np.float16)

S\_Nodes = np.zeros((total\_nodes, 10), dtype=np.float16)

for element\_index in range(total\_elements):

element\_node\_index = elements[element\_index, 1:9]

u = np.zeros((24, 1))

for element\_node in range(8): # 找到单位位移列向量

node\_index = element\_node\_index[element\_node] - 1

u[element\_node \* 3:(element\_node + 1) \* 3] = np.array([U1[node\_index], U2[node\_index], U3[node\_index]]).reshape(

[3, 1])

S\_Element = np.zeros((8, 6))

for equal\_node in range(8): # 求积分点应力分量

b = B[:, :, equal\_node, element\_index]

S\_Element[equal\_node, :] = np.dot(d, np.dot(b, u)).reshape(-1)

S\_Elements[:, :, element\_index] = S\_Element # 存储积分点应力分量

for node\_element in range(8): # 差值每个单元节点

s\_node = np.zeros(6)

natural\_coor = equal\_nodes[node\_element] # 自然坐标

for s\_index in range(6): # 差值每个应力分量

s\_node[s\_index] = np.dot(poly\_(natural\_coor), np.dot(inv\_X, S\_Element[:, s\_index]))

node\_index = element\_node\_index[node\_element] - 1

S\_Nodes[node\_index, :6] += s\_node

S\_Nodes[node\_index, 9] += 1

for i in range(total\_nodes): #

S\_Nodes[i, :6] /= S\_Nodes[i, 9]

# %%

# 绘制三维散点图

fig1 = plt.figure()

ax1 = fig1.add\_subplot(111, projection='3d')

ax1.scatter(nodes[:, 1], nodes[:, 2], nodes[:, 3], c=S\_Nodes[:, 0], s=20, cmap='viridis')

ax1.set\_xlabel('X')

ax1.set\_ylabel('Y')

ax1.set\_zlabel('Z')

ax1.set\_title('Stress Component S11')

plt.show()

fig2 = plt.figure()

ax2 = fig2.add\_subplot(111, projection='3d')

ax2.scatter(nodes[:, 1], nodes[:, 2], nodes[:, 3], c=U1, s=20, cmap='viridis')

ax2.set\_xlabel('X')

ax2.set\_ylabel('Y')

ax2.set\_zlabel('Z')

ax2.set\_title('Stress Component S22')

plt.show()

fig3 = plt.figure()

ax3 = fig3.add\_subplot(111, projection='3d')

ax3.scatter(nodes[:, 1], nodes[:, 2], nodes[:, 3], c=S\_Nodes[:, 2], s=20, cmap='viridis')

ax3.set\_xlabel('X')

ax3.set\_ylabel('Y')

ax3.set\_zlabel('Z')

ax3.set\_title('Stress Component S33')

plt.show()

# 保存应力数据

np.savetxt("result/" + inp\_file + "S11.txt", S\_Nodes[:, 0])

np.savetxt("result/" + inp\_file + "S22.txt", S\_Nodes[:, 1])

np.savetxt("result/" + inp\_file + "S33.txt", S\_Nodes[:, 2])

np.savetxt("result/" + inp\_file + "S23.txt", S\_Nodes[:, 3])

np.savetxt("result/" + inp\_file + "S13.txt", S\_Nodes[:, 4])

np.savetxt("result/" + inp\_file + "S12.txt", S\_Nodes[:, 5])

# 保存位移数据

np.savetxt("result/" + inp\_file + "U1.txt", U1)

np.savetxt("result/" + inp\_file + "U2.txt", U2)

np.savetxt("result/" + inp\_file + "U3.txt", U3)

#%%

#生成可视化vtk文件

class FEDataModel:

"""有限元数据模型类"""

def \_\_init\_\_(self):

self.nodes = [] # 节点几何坐标

self.elements = [] # 单元拓扑信息

self.s = []

self.scalars = {} # 节点标量属性

self.vectors = {} # 节点向量属性

self.ugrid = vtk.vtkUnstructuredGrid() # 用于VTK可视化的数据模型

self.ugrid.Allocate(100)

# 得到节点坐标单元节点编号

def read\_nodes\_elements(self, node\_file, element\_file, s\_file):

with open(node\_file) as f:

for line in f.readlines():

line = line.strip("\n")

self.nodes.append(list(map(lambda x: float(x), line.split(",")))[1:])

f.close()

with open(element\_file) as f:

for line in f.readlines():

line = line.strip("\n")

self.elements.append(list(map(lambda x: int(x), line.split(",")))[1:])

f.close()

with open(s\_file) as f:

for line in f.readlines():

line = line.strip("\n")

self.s.append(list(map(lambda x: float(x), line.split(","))))

f.close()

nodes = vtk.vtkPoints()

for i in range(0, len(self.nodes)):

nodes.InsertPoint(i, self.nodes[i])

for i in range(0, len(self.elements)):

try:

hexahedron = vtk.vtkHexahedron()

for j in range(8):

hexahedron.GetPointIds().SetId(j, self.elements[i][j] - 1)

self.ugrid.InsertNextCell(hexahedron.GetCellType(), hexahedron.GetPointIds())

except Exception as err:

print("FEDataModel构建中遇到错误单元类型！")

print(err)

self.ugrid.SetPoints(nodes)

# 获得标量信息，应力、温度场等等

def read\_ntl(self):

scalar = self.s

# 存储标量值

scalars = vtk.vtkFloatArray()

scalars.SetName("S")

for i in range(0, len(scalar)):

scalars.InsertTuple1(i, scalar[i][0])

# 设定每个节点的标量值

self.ugrid.GetPointData().SetScalars(scalars)

def display(self):

renderer = vtk.vtkRenderer()

renWin = vtk.vtkRenderWindow()

renWin.AddRenderer(renderer)

iren = vtk.vtkRenderWindowInteractor()

iren.SetRenderWindow(renWin)

colors = vtk.vtkNamedColors()

ugridMapper = vtk.vtkDataSetMapper()

ugridMapper.SetInputData(self.ugrid)

ugridActor = vtk.vtkActor()

ugridActor.SetMapper(ugridMapper)

ugridActor.GetProperty().SetColor(colors.GetColor3d("AliceBlue"))

ugridActor.GetProperty().EdgeVisibilityOn()

renderer.AddActor(ugridActor)

renderer.SetBackground(colors.GetColor3d("AliceBlue"))

renderer.ResetCamera()

renderer.GetActiveCamera().Elevation(60.0)

renderer.GetActiveCamera().Azimuth(30.0)

renderer.GetActiveCamera().Dolly(1.2)

renWin.SetSize(640, 480)

# Interact with the data.

renWin.Render()

iren.Start()

def drawScalarField(self, scalar\_mapper, scalarRange, title):

# 定义颜色映射表

lut = vtk.vtkLookupTable()

lut.SetHueRange(0.5, 0.0) # 色调范围从红色到蓝色

lut.SetAlphaRange(1.0, 1.0) # 透明度范围

lut.SetValueRange(1.0, 1.0)

lut.SetSaturationRange(0.5, 0.5) # 颜色饱和度

lut.SetNumberOfTableValues(16)

lut.SetNumberOfColors(16) # 颜色个数

lut.SetRange(scalarRange)

lut.Build()

scalar\_mapper.SetScalarRange(scalarRange)

scalar\_mapper.SetLookupTable(lut)

scalar\_actor = vtk.vtkActor()

scalar\_actor.SetMapper(scalar\_mapper)

self.renderer.AddActor(scalar\_actor)

# 色标带

scalarBar = vtk.vtkScalarBarActor()

scalarBar.SetLookupTable(scalar\_mapper.GetLookupTable()) # 将颜色查找表传入窗口中的色标带

scalarBar.SetTitle(title)

scalarBar.SetNumberOfLabels(5)

self.renderer.AddActor2D(scalarBar)

def save\_vtk(self, filename):

writer = vtk.vtkXMLUnstructuredGridWriter()

writer.SetFileName(filename)

writer.SetInputData(self.ugrid)

writer.Write()

for i in ["11", "22", "33", "13", "23", "12"]:

model = FEDataModel()

model.read\_nodes\_elements("data/Nodes.txt", "data/Elements.txt", "result/" + inp\_file + "S"+i+".txt")

model.read\_ntl()

model.display()

model.save\_vtk("visualize/" + inp\_file + "S"+i+".vtk")

for i in ["1", "2", "3"]:

model = FEDataModel()

model.read\_nodes\_elements("data/Nodes.txt", "data/Elements.txt", "result/" + inp\_file + "U"+i+".txt")

model.read\_ntl()

model.display()

model.save\_vtk("visualize/" + inp\_file + "U"+i+".vtk")

print("Done")