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

1.1. 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。

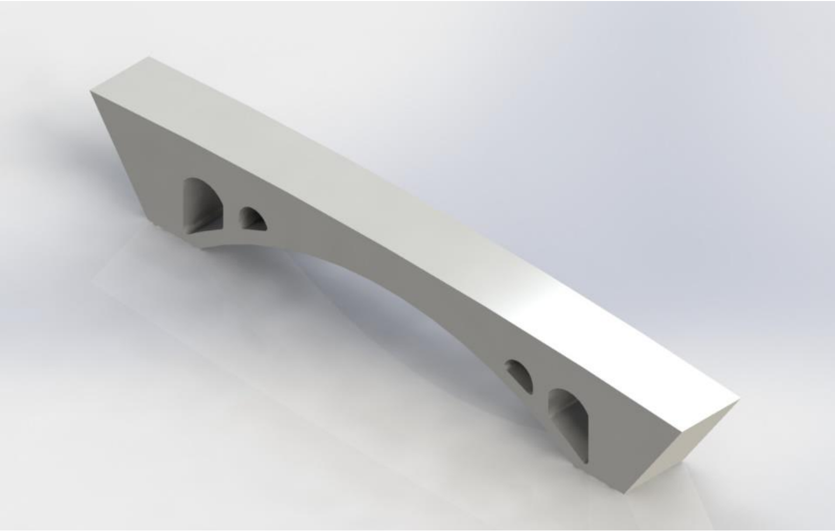


图 1 拱形待分析模型

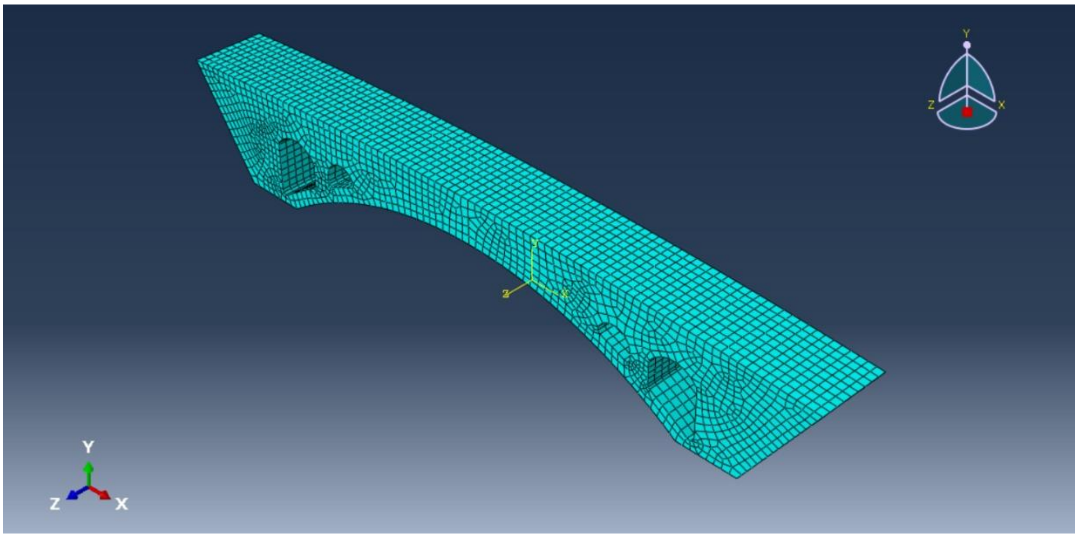


图 2 以 5.5 网格密度划分网格

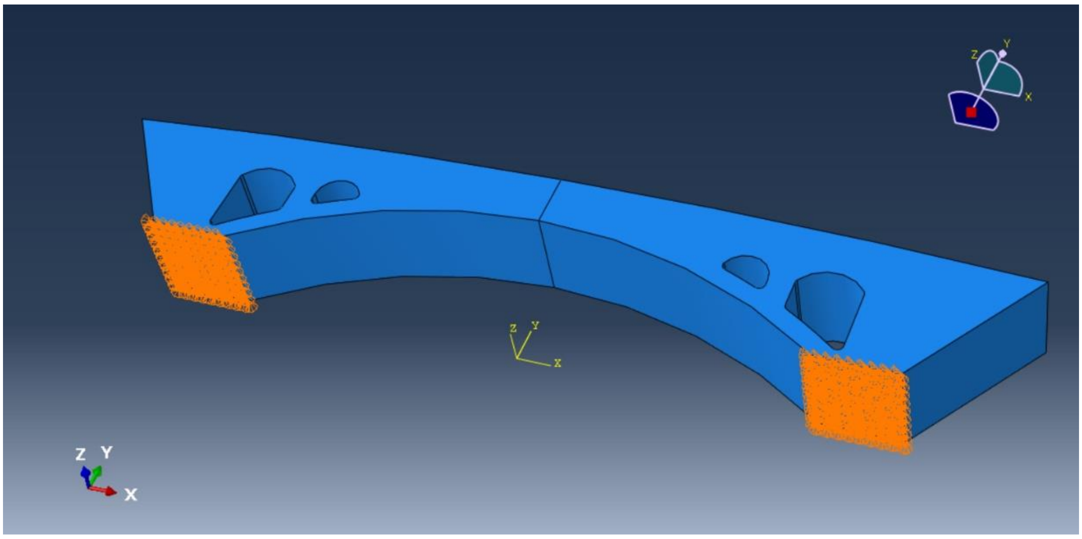


图 3 底部约束

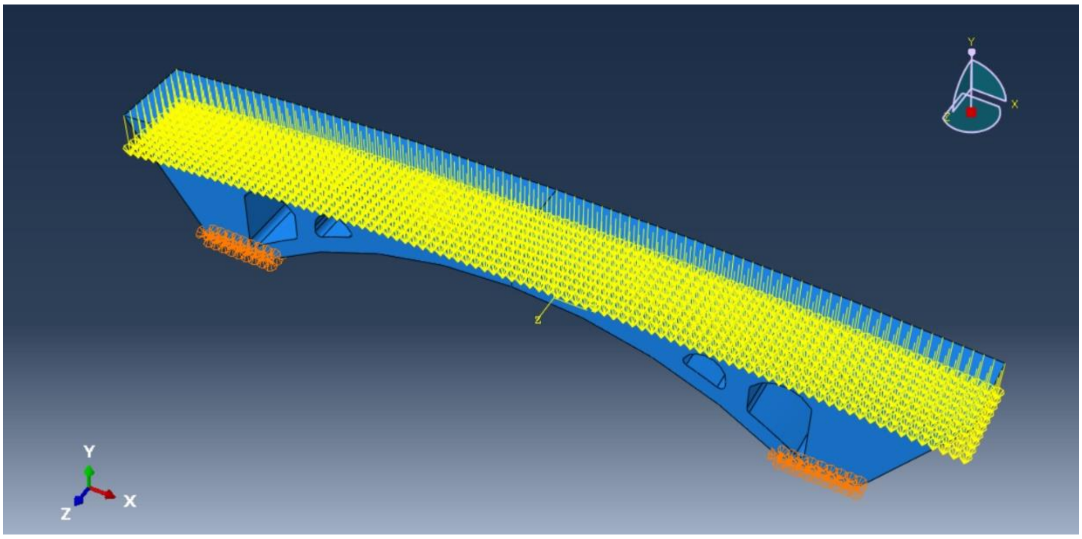


图 4 顶部集中力节点

1.2. 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。

2. Python-Code 文件夹介绍

2.1. 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。

2.2. 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；

2.3. 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。

2.4. 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。

2.5. 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。

2.6. 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。

3. 八结点立方体单元介绍

3.1. 单元节点

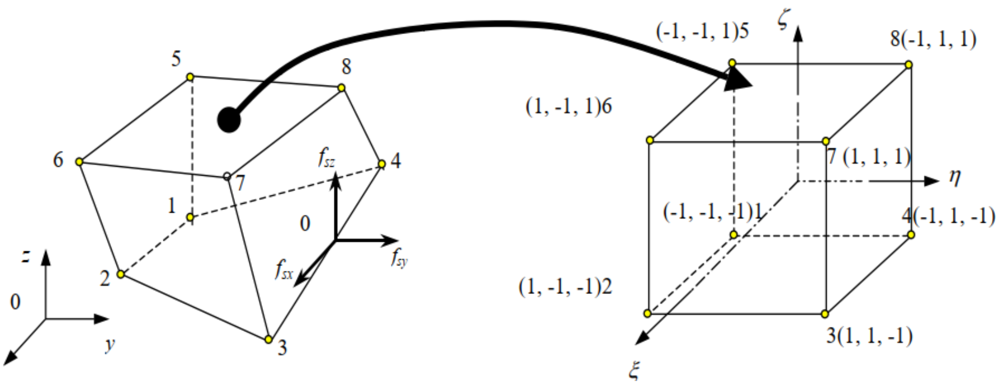


图 5 八结点立方体等参单元

3.2. 等参单元形函数

$$N_i = \frac{1}{8}(1+\xi_i\xi)(1+\eta_i\eta)(1+\zeta_i\zeta) \quad (i=1,2,\cdots,8)$$

$$\begin{cases} x = \sum_{i=1}^{8} N_i(\xi,\eta,\zeta)x_i \\ y = \sum_{i=1}^{8} N_i(\xi,\eta,\zeta)y_i \\ z = \sum_{i=1}^{8} N_i(\xi,\eta,\zeta)z_i \end{cases}$$

(ξ_i, η_i, ξ_i)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：

$$\begin{cases} \dfrac{\partial N_i}{\partial \xi} = \dfrac{1}{8}\xi_i(1+\eta_i\eta)(1+\zeta_i\zeta) \\ \dfrac{\partial N_i}{\partial \eta} = \dfrac{1}{8}\eta_i(1+\xi_i\xi)(1+\zeta_i\zeta) \quad i=(1,2,\cdots,8) \\ \dfrac{\partial N_i}{\partial \zeta} = \dfrac{1}{8}\zeta_i(1+\xi_i\xi)(1+\eta_i\eta) \end{cases}$$

假设节点位移矩阵：

$$\mathbf{q}^e = [u_1 \quad v_1 \quad w_1 \quad \cdots \quad \cdots \quad u_m \quad v_m \quad w_m]^T$$

根据式：

$$\boldsymbol{\varepsilon} = \mathbf{B}\mathbf{q}^e$$

得单元应变矩阵，该矩阵中的偏导数项根据下式求解：

$$
\mathbf{B}_i = \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_i}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_i}{\partial z} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} & 0 \\ 0 & \frac{\partial N_i}{\partial z} & \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} & 0 & \frac{\partial N_i}{\partial x} \end{bmatrix} \quad (i = 1, 2, \cdots, m)
$$

$$
\begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{Bmatrix} = \mathbf{J}^{-1} \begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{Bmatrix}
$$

$$
\mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{8} \frac{\partial N_i}{\partial \xi} x_i & \sum_{i=1}^{8} \frac{\partial N_i}{\partial \xi} y_i & \sum_{i=1}^{8} \frac{\partial N_i}{\partial \xi} z_i \\ \sum_{i=1}^{8} \frac{\partial N_i}{\partial \eta} x_i & \sum_{i=1}^{8} \frac{\partial N_i}{\partial \eta} y_i & \sum_{i=1}^{8} \frac{\partial N_i}{\partial \eta} z_i \\ \sum_{i=1}^{8} \frac{\partial N_i}{\partial \zeta} x_i & \sum_{i=1}^{8} \frac{\partial N_i}{\partial \eta} y_i & \sum_{i=1}^{8} \frac{\partial N_i}{\partial \eta} z_i \end{bmatrix}
$$

3.3. 刚度矩阵

刚度矩阵表达式为：

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

对积分区域为[-1 1]的被积函数，使用高斯积分点近似求积。

$$
\int_{-1}^{1} \int_{-1}^{1} \int_{-1}^{1} \mathbf{B}^T \mathbf{D} \mathbf{B} \left| \mathbf{J} \right| d\xi d\eta d\zeta = \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{l} w_i w_j w_k f(\xi_i, \eta_j, \zeta_k)
$$

3.4. 后处理

根据式：$\{\sigma\} = [D][B]\{u\}$ 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：

$$
\alpha_1 + \alpha_2 \xi + \alpha_3 \eta + \alpha_4 \zeta + \alpha_5 \xi \eta + \alpha_6 \xi \zeta + \alpha_7 \eta \zeta + \alpha_8 \xi \eta \zeta = \sigma
$$

对于多个单元的公共节点应力，进行取平均值处理。

4. 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

4.1. 线性二积分点 C3D8 单元

1)　位移对比

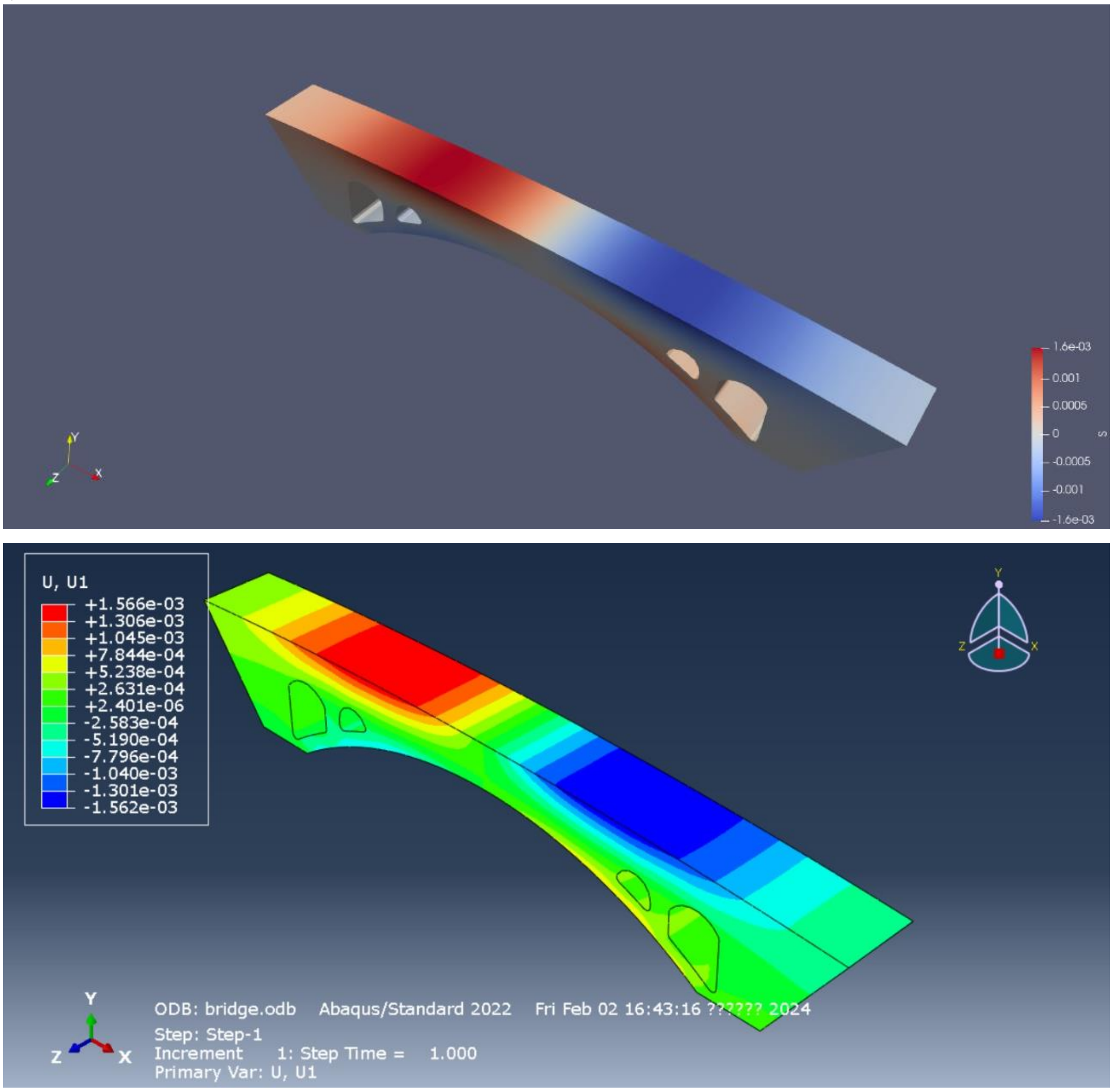
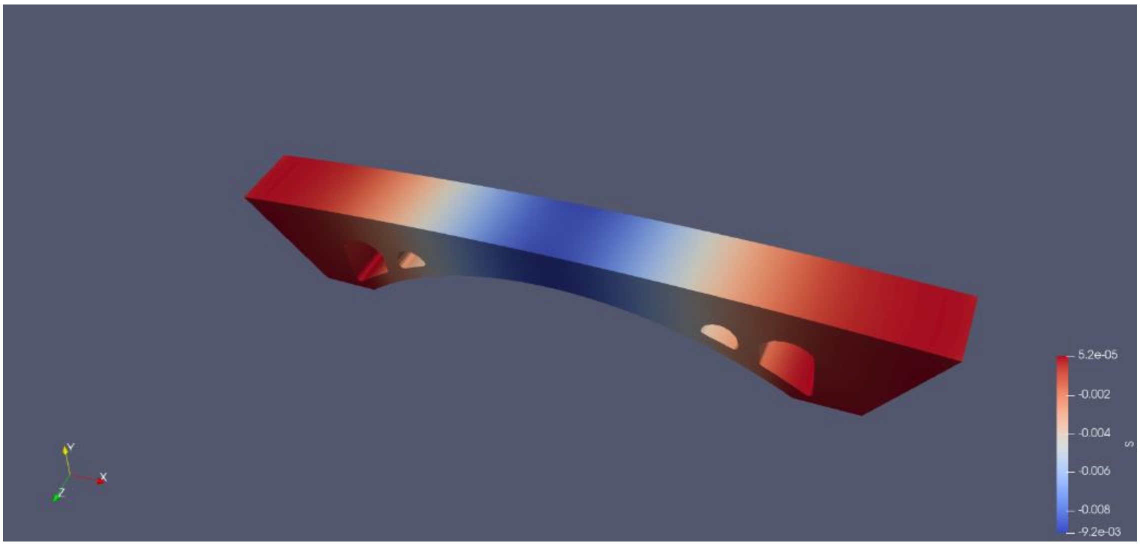


图 6 U1


5.2e-05
-0.002
-0.004
-0.006
-0.008
-9.2e-03
Y
Z
X

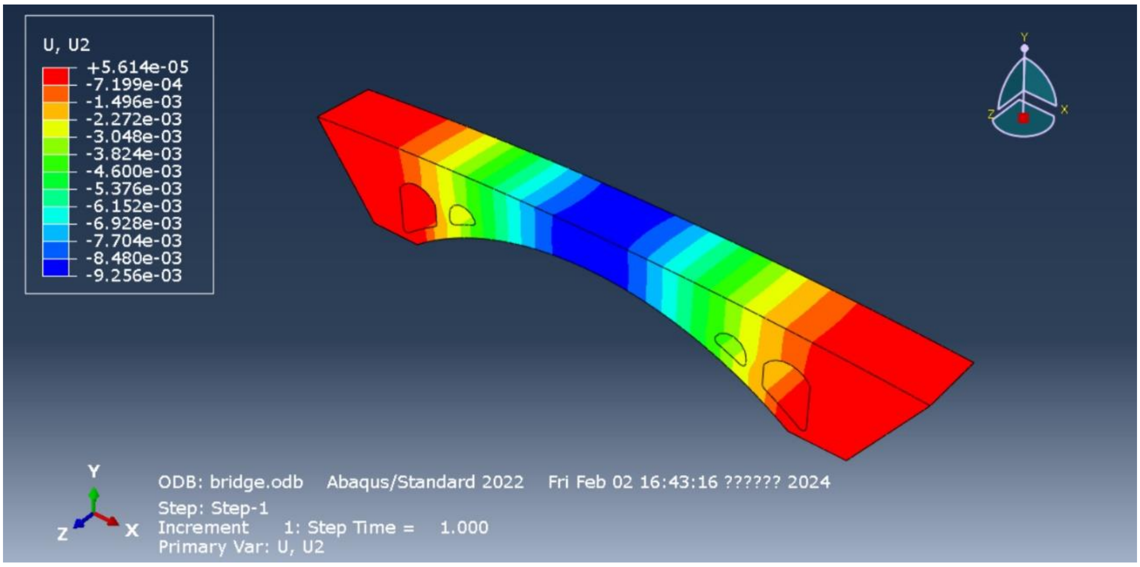
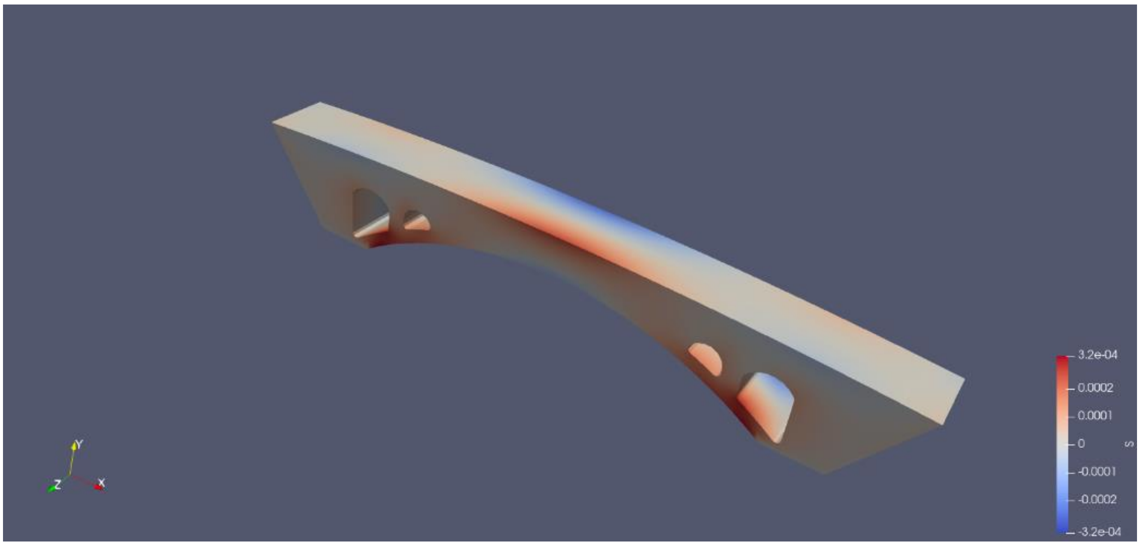

U, U2
+5.614e-05
-7.199e-04
-1.496e-03
-2.272e-03
-3.048e-03
-3.824e-03
-4.600e-03
-5.376e-03
-6.152e-03
-6.928e-03
-7.704e-03
-8.480e-03
-9.256e-03
ODB: bridge.odb Abaqus/Standard 2022 Fri Feb 02 16:43:16 ?????? 2024
Step: Step-1
Increment 1: Step Time = 1.000
Primary Var: U, U2
Y
Z
X

图 7 U2


3.2e-04
0.0002
0.0001
0
-0.0001
-0.0002
-3.2e-04
Y
Z
X

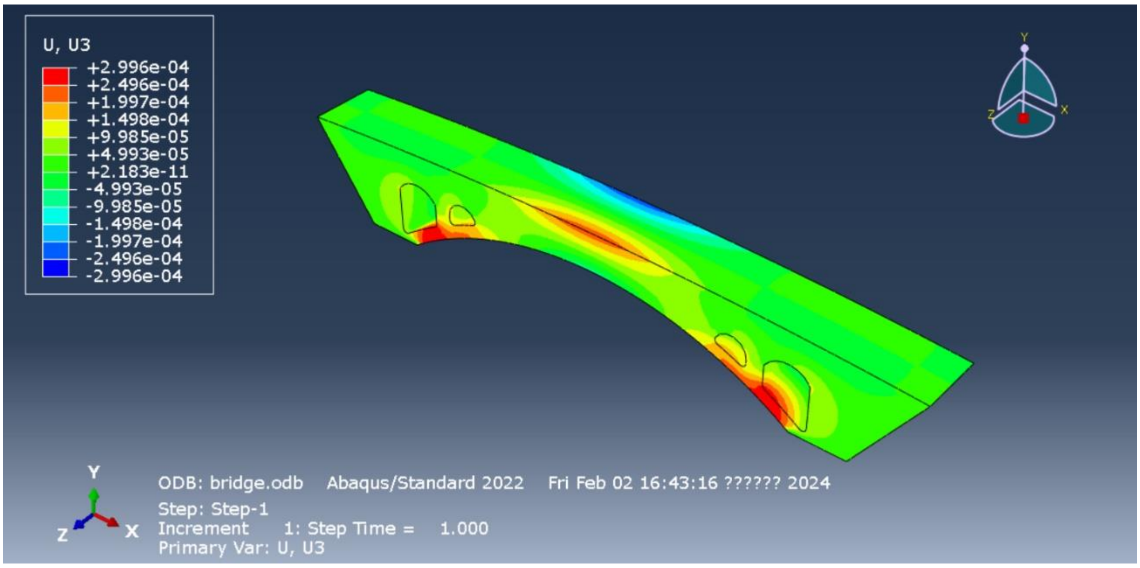


图 8 U3

2) 应力对比

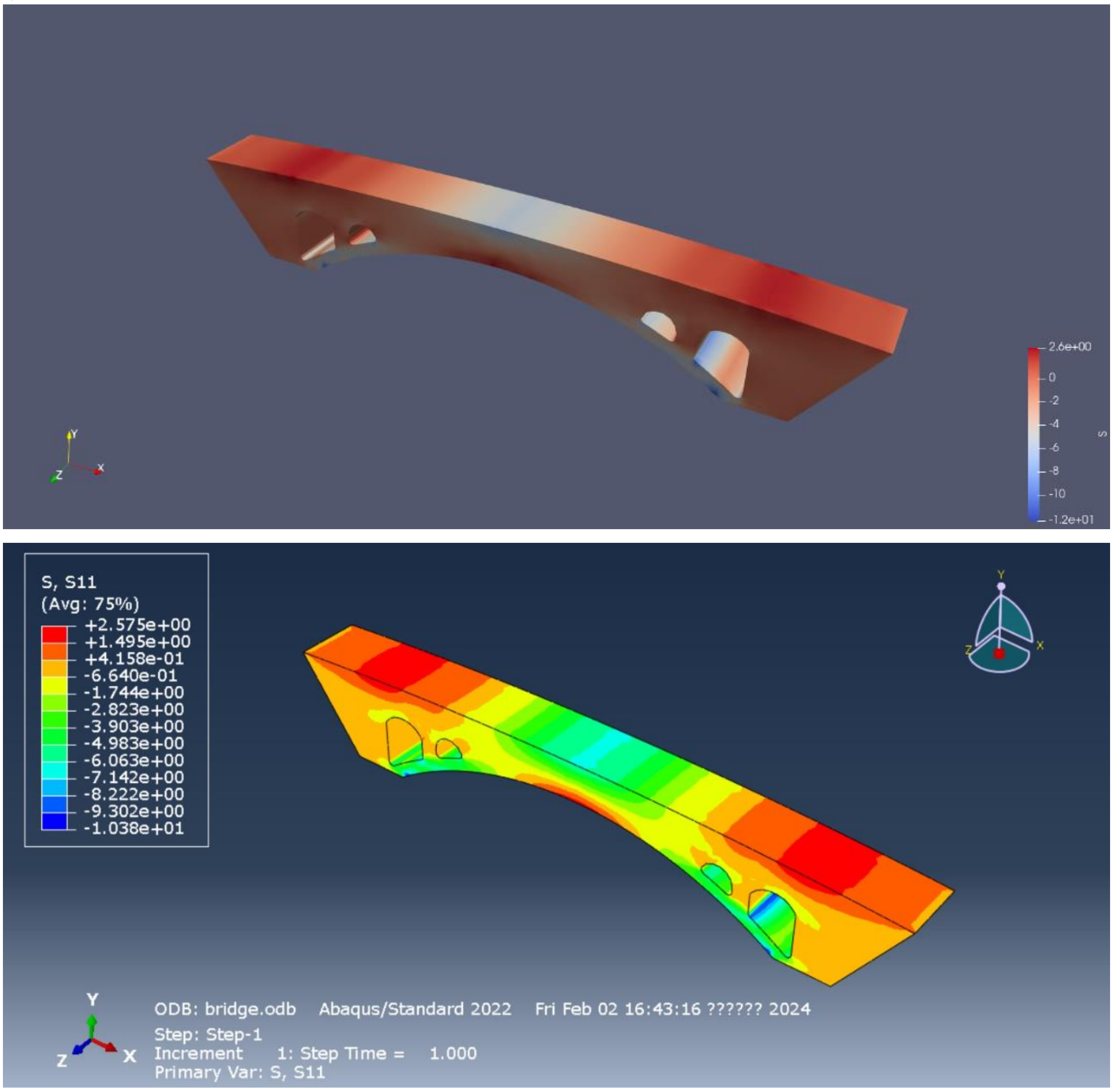


图 9 S11

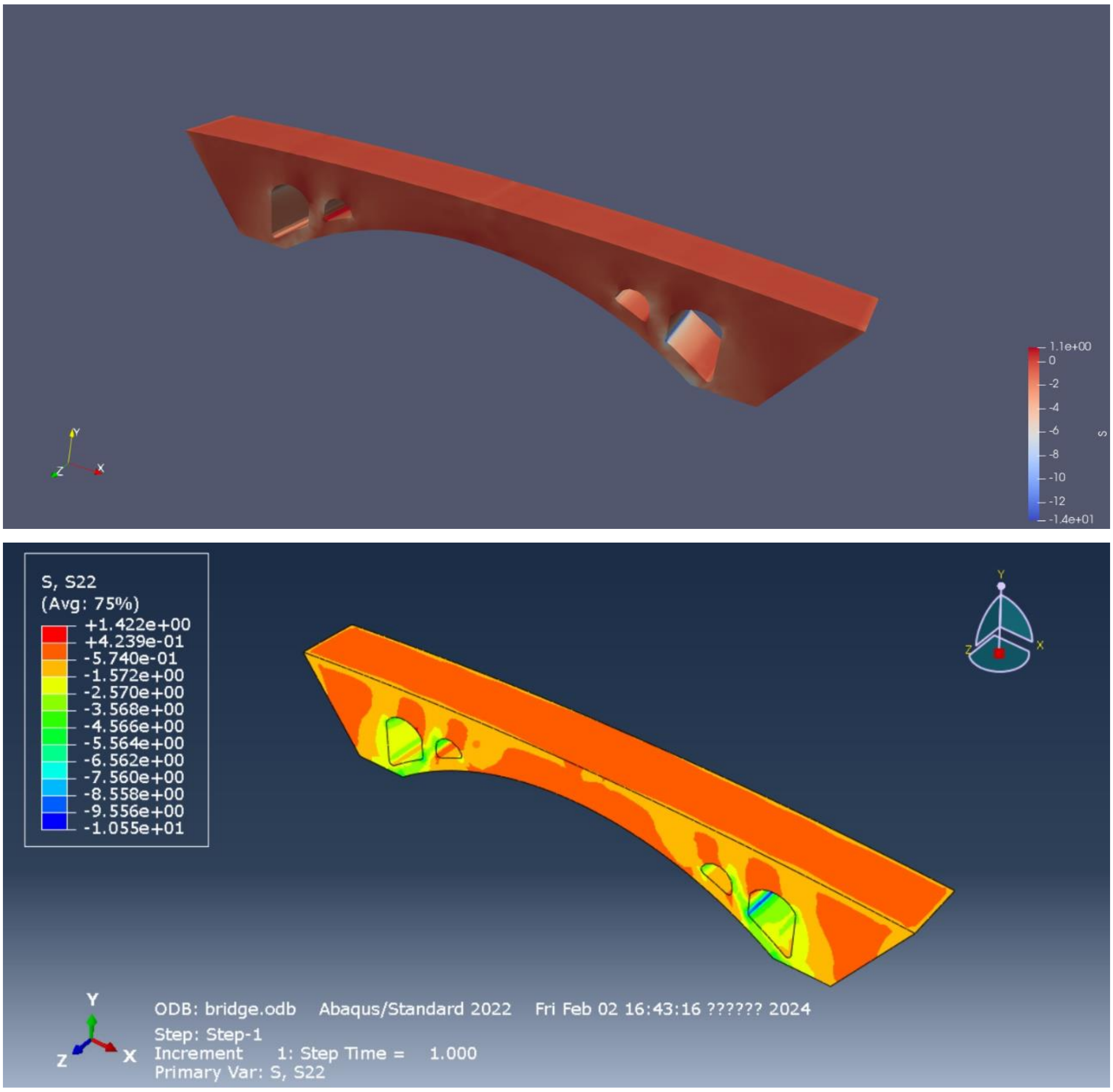
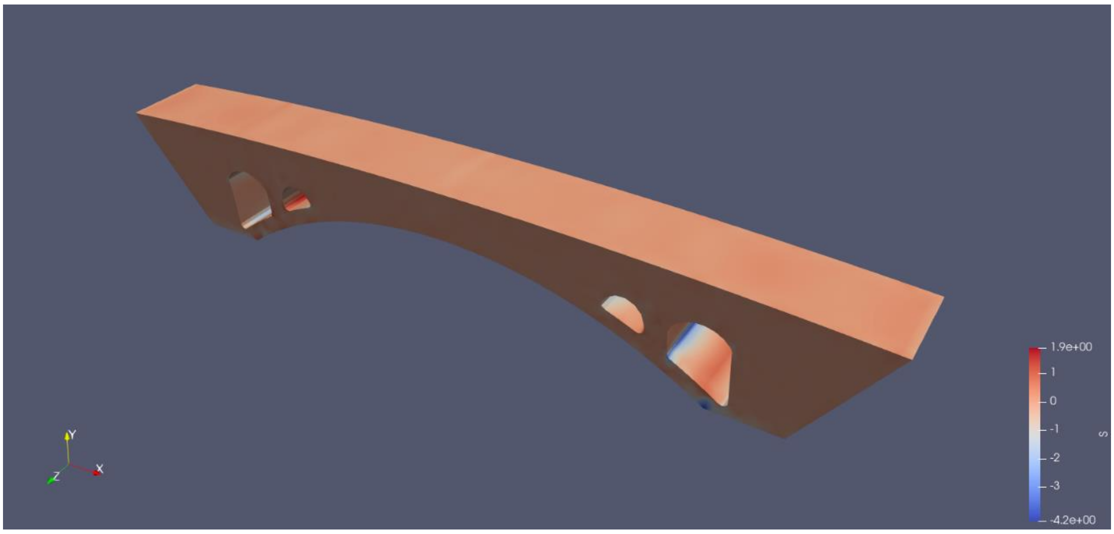

1.1e+00
0
-2
-4
-6
-8
-10
-12
-1.4e+01
S
Y
Z
X
S, S22
(Avg: 75%)
+1.422e+00
+4.239e-01
-5.740e-01
-1.572e+00
-2.570e+00
-3.568e+00
-4.566e+00
-5.564e+00
-6.562e+00
-7.560e+00
-8.558e+00
-9.556e+00
-1.055e+01
ODB: bridge.odb Abaqus/Standard 2022 Fri Feb 02 16:43:16 ?????? 2024
Step: Step-1
Increment 1: Step Time = 1.000
Primary Var: S, S22

图 10 S22


Y
Z
X
1.9e+00
1
0
-1
-2
-3
-4.2e+00
S

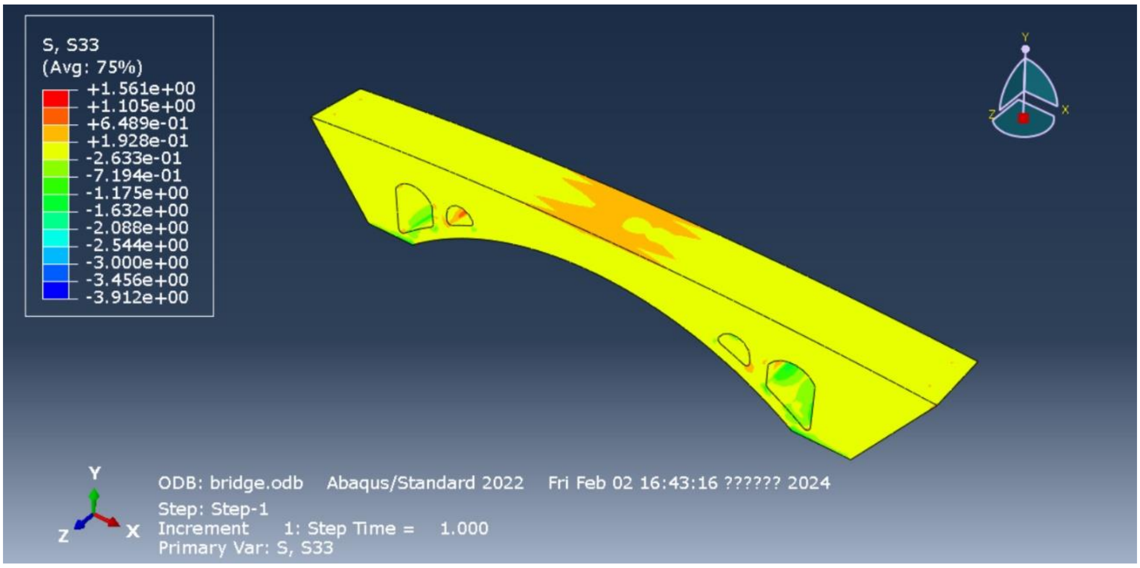
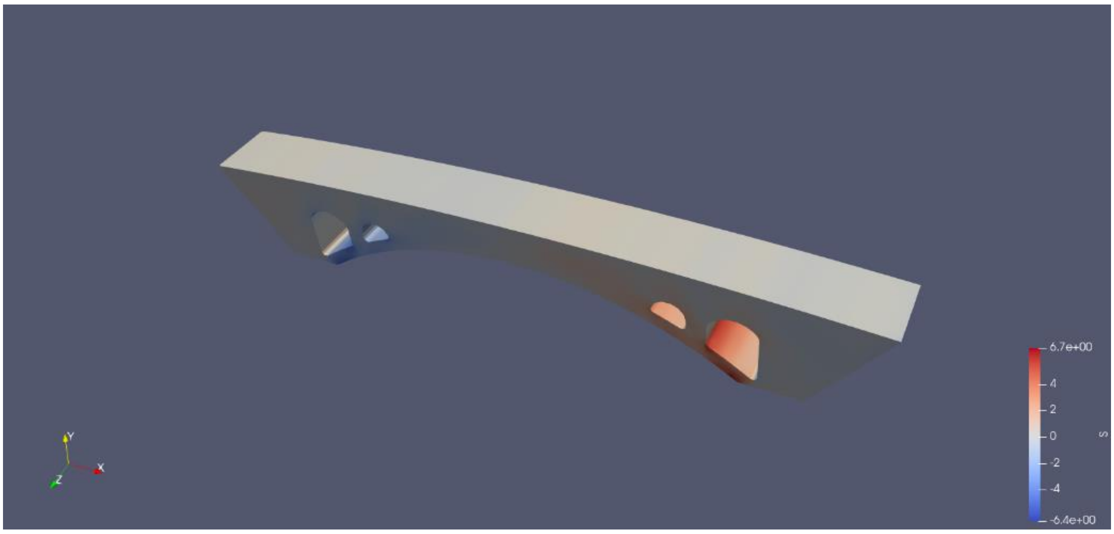


图 11 S33



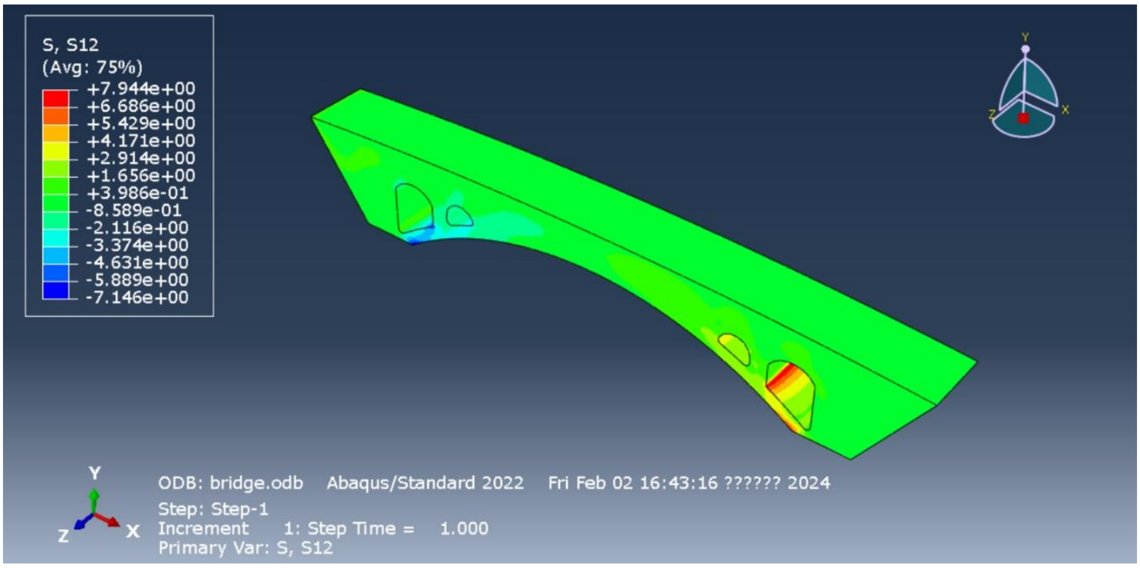
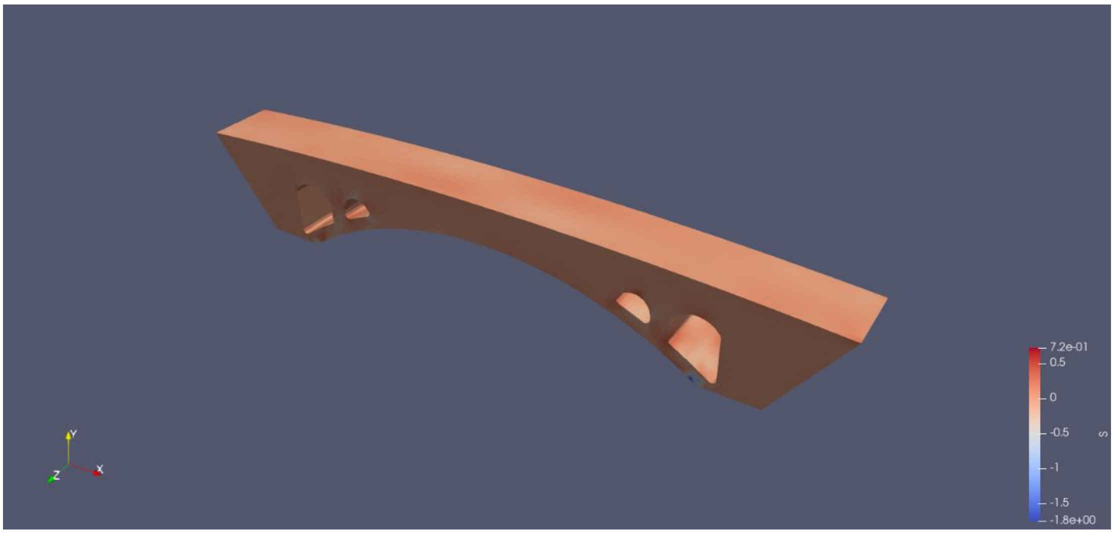


图 12 S12


7.2e-01
0.5
0
-0.5
-1
-1.5
-1.8e+00
S

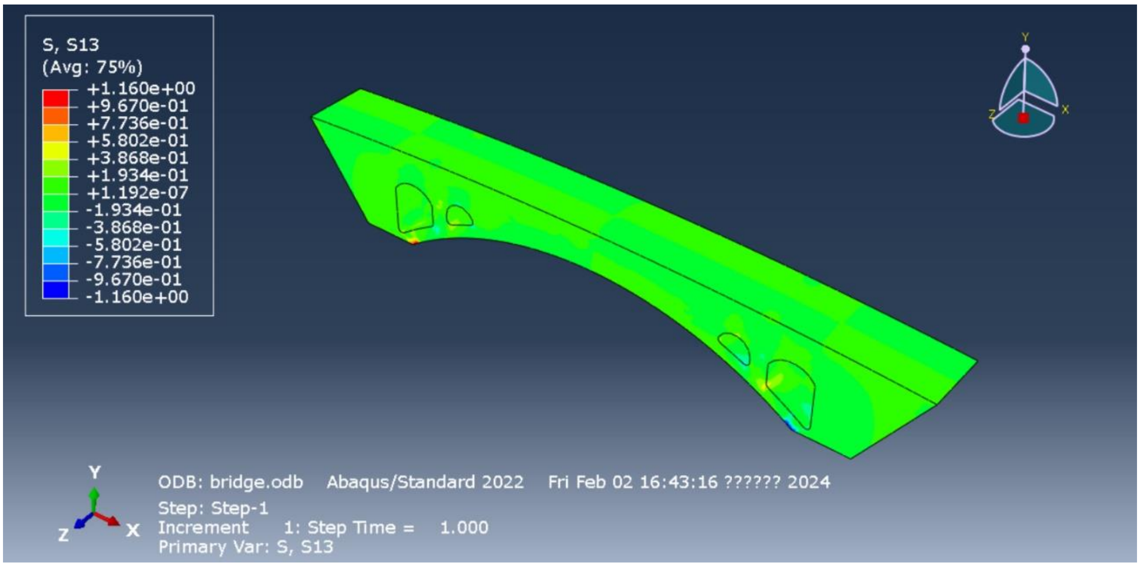
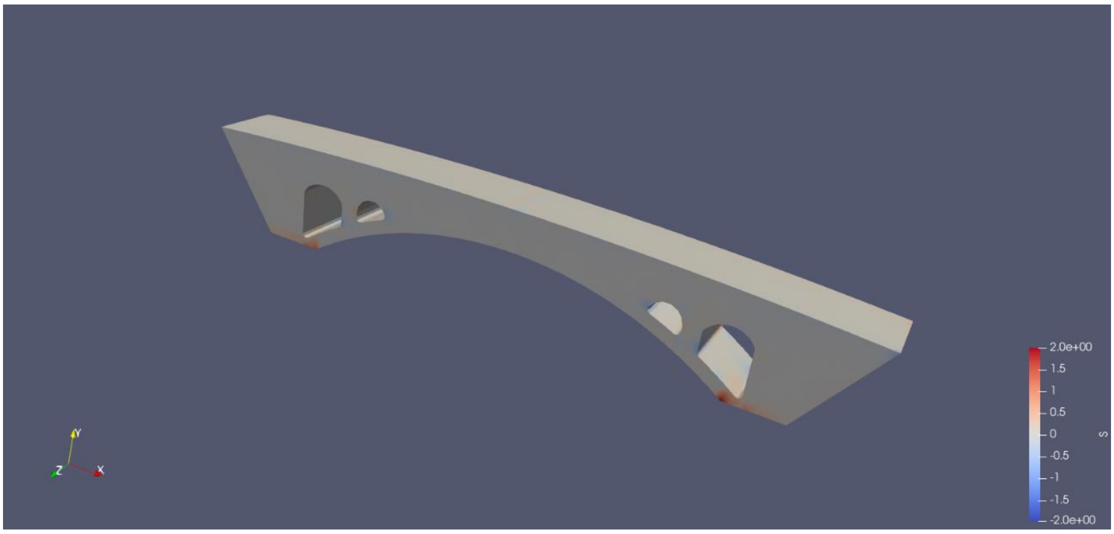

S, S13
(Avg: 75%)
+1.160e+00
+9.670e-01
+7.736e-01
+5.802e-01
+3.868e-01
+1.934e-01
+1.192e-07
-1.934e-01
-3.868e-01
-5.802e-01
-7.736e-01
-9.670e-01
-1.160e+00
ODB: bridge.odb Abaqus/Standard 2022 Fri Feb 02 16:43:16 ?????? 2024
Step: Step-1
Increment 1: Step Time = 1.000
Primary Var: S, S13

图 13 S13


2.0e+00
1.5
1
0.5
0
-0.5
-1
-1.5
-2.0e+00
S

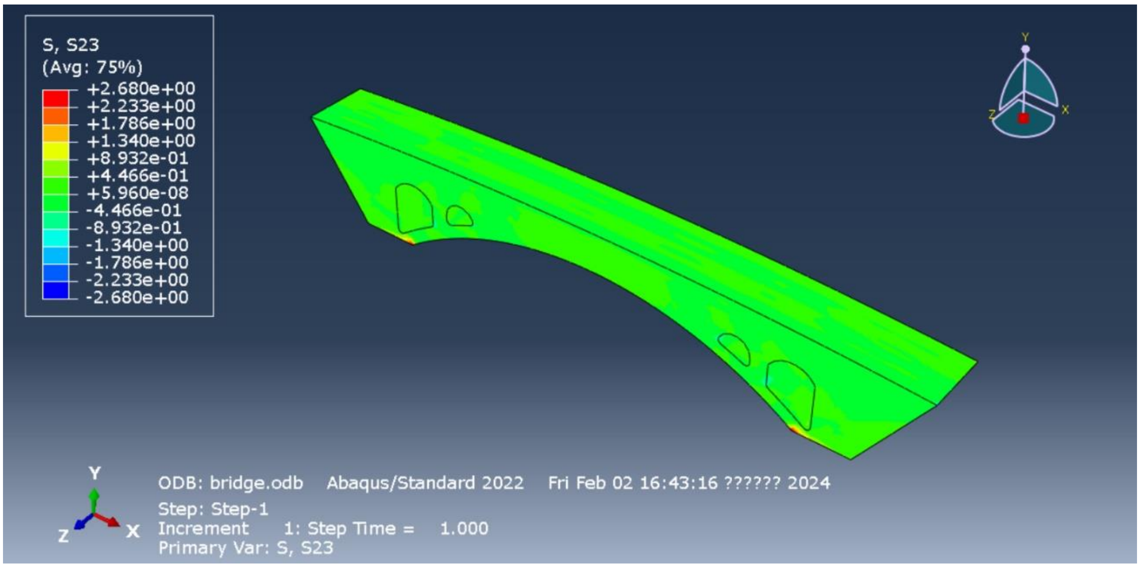


图 14 S23

3) 位移误差对比

1) 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。

4.2. 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()
```

```python
        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()
```

```python
            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")
```