《非线性有限元方法》期中大作业

张家瑞 2401111734

目录

1 线弹性问题

1.1 介绍

本章节介绍的 MATLAB 程序，可以实现二线弹性平面问题的求解。目前能够实现的可调节功能有：模拟单元的设置、网格划分节点数的设置、单元类型的选取、边界条件的设置。

1.2 程序结构

该代码的主要结构为：网格划分与处理、设置边界条件、设置材料属性、单元 B 矩阵与刚度矩阵的计算、外力条件的施加、强制边界条件的施加、总体方程的求解、后处理（应力应变的计算等）。整体流程图如图 1 所示。

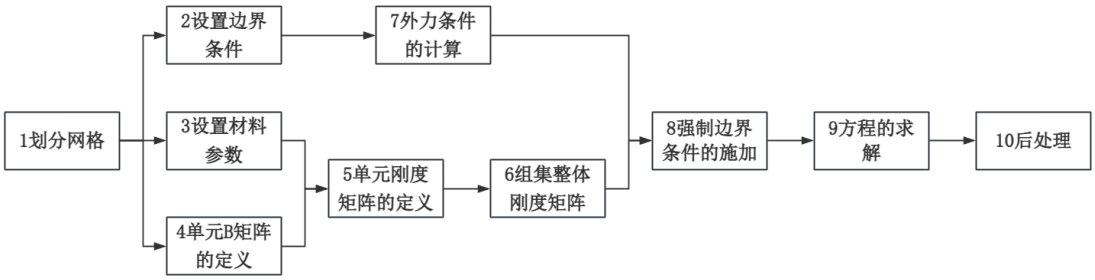


图 1 线弹性问题 MATLAB 代码流程图

1.2.1 划分网格（generate_mesh 函数）

通过 generate_mesh 函数实现网格的生成。其中通过全局变量 flag 来定义单元类型，当 flag=1 时为一次三角形单元；当 flag=2 时为双线性四边形单元。在函数中定义网格划分数（局部变量 nx 与 ny 分别为水平与垂直方向的划分数）以及模拟单元的几何设置。x_a 为节点的坐标、elem 为生成单元中的节点信息。其中节点的编号顺序为从先纵向后横向，单元中的节点编号为逆时针排列。在生成单元中节点信息时，先选取出四个节点，再根据类型定义为三角形或四边形单元。代码如下所示。

```
function [x_a,elem]=generate_mesh(flag)
  % flag = 1 时生成三角形单元，flag = 2 时生成四边形单元
  % 网格划分参数
  nx = 10;  % 水平划分数
  ny = 5;  % 垂直划分数
  % 预分配节点坐标数组
  x_a = zeros((nx+1)*(ny+1), 2);
  index = 0;  % 变量定义
```

```
    for ix = 0:nx
        % 水平位置参数 xi 以及对应 x 坐标
        xi = ix/nx;
        xcoord = 2 * xi;  % x 从 0 到 2
        % 每列的底部 y 坐标与顶部 y 坐标
        y_bot = 0.5 * xi;  % 当 xi=0 时为 0，xi=1 时为 0.5
        y_top = 1;        % 顶部始终为 1
        for iy = 0:ny
            eta = iy/ny;
            % 线性插值计算 y 坐标，当 eta=0 时取底部 y，eta=1 时取顶部 y
            ycoord = (1-eta)*y_bot + eta*y_top;
            index = index + 1;
            x_a(index, :) = [xcoord, ycoord];
        end
    end
    % 生成单元中的节点信息
    % 采用的节点编号规则：先沿 y 方向（每一列）排列，再沿 x 方向排列
    % 对于每个单元节点编号顺序为逆时针
    elem = [];
    for ix = 1:nx
        for iy = 1:ny
            n1 = (ix-1)*(ny+1) + iy;
            n2 = n1 + 1;
            n3 = ix*(ny+1) + iy;
            n4 = n3 + 1;
            if flag == 1
                % 三角形单元，将四边形单元分为两个三角形
                elem = [elem; n1, n3, n2];
                elem = [elem; n2, n3, n4];
            elseif flag == 2
                % 四边形单元
                elem = [elem; n1, n3, n4, n2];
            end
        end
    end
end
```

在划分网格后，通过 triplot 或 quadplot 函数画出网格。

1.2.2 高斯点物理坐标和单元面积计算（g_center 函数）

在网格生成后，需要针对每个单元计算其高斯点坐标以及单元面积。其中输入信息为节点的坐标 x_a 以及单元节点信息 elem，输出信息为每个单元的质心坐标 xg（因为采用单点积分）以及每个单元面积 Area。通过坐标平均计算质

心坐标，通过通用的多边形面积计算公式来计算面积

$$S = \frac{1}{2}\left|\sum_{i=1}^{n}(x_i y_{i+1} - x_{i+1} y_i)\right|,$$

通过顶点加权计算质心坐标

$$xg_1 = \frac{1}{6A}\sum_{i=1}^{n}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i),$$

$$xg_2 = \frac{1}{6A}\sum_{i=1}^{n}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i).$$

特别地，当单元面积比较小时，设置质心坐标按照顶点坐标的平均值进行选取。代码如下所示。

```
function [xg,Area]=g_center(x_a,elem)
  %  根据单元节点坐标计算单元的质心（xg）和面积（Area）。
  %  可处理三角形与四边形（或更一般的凸多边形）。
  %  x_a  - 节点坐标矩阵，每行为一个节点的横纵坐标
  %  elem - 单元连通关系，每行表示一个单元的节点序号（节点按逆时针顺序排列）
  %  xg   - 每个单元的质心坐标，矩阵大小为节点数乘 2
  %  Area - 每个单元的面积，列向量
  elements = size(elem, 1);
  xg = zeros(elements, 2);
  Area = zeros(elements, 1);
  eps = 1e-12;
  for e = 1:elements
     nodes = elem(e, :);          % 单元中所有节点的索引
     coords = x_a(nodes, :);      % 单元各节点的坐标，要求按单元边界连续排序
     n = size(coords, 1);         % 多边形的顶点数（3 或 4）
     a_sum = 0;
     Cx_sum = 0;
     Cy_sum = 0;
     % 采用通用的多边形公式计算面积与质心累加量
     for i = 1:n
        j = mod(i, n) + 1;  % 保证 i+1 超出范围时回到第 1 个节点
        cross_val = coords(i,1)*coords(j,2) - coords(j,1)*coords(i,2);
        a_sum = a_sum + cross_val;
        Cx_sum = Cx_sum + (coords(i,1) + coords(j,1)) * cross_val;
        Cy_sum = Cy_sum + (coords(i,2) + coords(j,2)) * cross_val;
     end
     A = 0.5 * a_sum;
```

```
        % 若 A 为负值，说明节点顺序为逆时针，取绝对值同时调整质心累加量的符号
        if A < 0
            A = -A;
            Cx_sum = -Cx_sum;
            Cy_sum = -Cy_sum;
        end
        Area(e) = A;
        % 若面积非常接近于 0，则质心直接取节点均值（防止数值除零问题）
        if abs(A) < eps
            xg(e, :) = mean(coords, 1);
        else
            xg(e, :) = [Cx_sum, Cy_sum] / (6*A);
        end
    end
end
```

1.2.3 设置边界条件（displa、dist 函数）

边界条件分为位移边界条件与外力边界条件，其中输入信息包括节点坐标 x_a，约束信息 A、B，单元节点信息 elem。输出信息包括位移约束 boundary，节点位移 disp，节点表面面积 l_area（为后续分配面力做准备）。因此 A 为位移边界条件信息、B 为外力边界条件信息。通过对指定的约束位置进行赋值来得到 boundary 与 disp。由于该问题中施加的力或位移约束均为完整的一条边界，因此约束 A、B 只需要四个信息。如果我们施加的约束为某条边界上的部分区间，则需要引入新的参数（详情见下一章）。代码如下所示。

```
function [boundary,disp]=displa(x_a,A)
    nNodes = size(x_a, 1);
    boundary = zeros(2*nNodes, 1);  % 初始化为 (2*nNodes,1) 向量
    disp = zeros(2*nNodes, 1);    % 位移向量同样调整
    tol = 1e-6;  % 数值判别容差
    % 遍历所有给定的边界条件
    for i = 1:size(A, 1)
        axisLabel = A(i, 1);  % 1 表示 x 方向，2 表示 y 方向
        loc = A(i, 2);       % 边界坐标位置
        direction = A(i, 3);  % 1:只约束 x，2:只约束 y，3:都约束
        value = A(i, 4);     % 指定位移值
        % 根据 axisLabel 找到满足条件的节点索引
        if axisLabel == 1
            indices = find(abs(x_a(:,1) - loc) < tol);
        elseif axisLabel == 2
            indices = find(abs(x_a(:,2) - loc) < tol);
        else
            indices = [];
```

```
        end
        % 对满足条件的每个节点，根据 direction 分别赋值
        for idx = indices'
            if direction == 1
                % 约束 x 方向（前 nNodes 个位置）
                boundary(2*idx-1) = 1;
                disp(2*idx-1) = value;
            elseif direction == 2
                % 约束 y 方向（后 nNodes 个位置）
                boundary(2*idx) = 1;
                disp(2*idx) = value;
            elseif direction == 3
                % 同时约束 x 和 y 方向
                boundary(2*idx-1) = 1;
                boundary(2*idx) = 1;
                disp(2*idx-1) = value;
                disp(2*idx) = value;
            end
        end
    end
end


function [l_area]=dist(x_a,elem,B)
    nNodes = size(x_a, 1);
    l_area = zeros(nNodes, 1); % 初始化每个节点的“面积”或边长为 0
    tol = 1e-6; % 数值容差
    % 遍历每一条给定的边界条件
    for i = 1:size(B, 1)
        axisLabel = B(i, 1); % 1 表示垂直边界（x=const），2 表示水平边界（y=const）
        loc = B(i, 2);       % 边界所在坐标值
        if axisLabel == 1
            % 垂直边界：选取所有满足 x = loc 的节点，排序依据其 y 坐标
            indices = find(abs(x_a(:,1) - loc) < tol);
            if isempty(indices)
                continue;
            end
            [~, order] = sort(x_a(indices, 2));
            indices = indices(order);
            % 若仅有一个点，则赋值为 0（或可根据需要赋予其他数值）
            if isscalar(indices)
                l_area(indices(1)) = l_area(indices(1)) + 0;
            else
                coords = x_a(indices, 2); % y 坐标
```

```
            dists = diff(coords);
            % 第一个点：距离取相邻点一半
            l_area(indices(1)) = l_area(indices(1)) + dists(1)/2;
            % 最后一个点
            l_area(indices(end)) = l_area(indices(end)) + dists(end)/2;
            % 中间各点：前后距离之和的一半
            for k = 2:length(indices)-1
                l_area(indices(k)) = l_area(indices(k)) + (dists(k-1) + dists(k))/2;
            end
        end
    elseif axisLabel == 2
        % 水平边界：选取所有满足 y = loc 的节点，排序依据其 x 坐标
        indices = find(abs(x_a(:,2) - loc) < tol);
        if isempty(indices)
            continue;
        end
        [~, order] = sort(x_a(indices, 1));
        indices = indices(order);
        if isscalar(indices)
            l_area(indices(1)) = l_area(indices(1)) + 0;
        else
            coords = x_a(indices, 1);  % x 坐标
            dists = diff(coords);
            l_area(indices(1)) = l_area(indices(1)) + dists(1)/2;
            l_area(indices(end)) = l_area(indices(end)) + dists(end)/2;
            for k = 2:length(indices)-1
                l_area(indices(k)) = l_area(indices(k)) + (dists(k-1) + dists(k))/2;
            end
        end
    end
end
```

1.2.4 设置材料参数（main 程序中）

本问题为平面应力问题，材料参数杨氏模量为3×10^7Pa，泊松比为 0.3，材料参数的定义放于主程序中。通过 properties 来存储材料参数。代码如下所示。

```
E  = 3.0e7; % Young's modulus [Pa]
nu = 0.3;   % Poisson ratio
properties(1)=E;
properties(2)=nu;
```

1.2.5 单元 B 矩阵的定义（bm 函数）

输入信息包括单元形函数在单元重心处的导数 dp、单元节点信息 elem、空间维度 sp，输出信息为 B 矩阵。其中单元 B 矩阵的求解为

$$B(q)=\begin{pmatrix} \frac{\partial N_I^q}{\partial x} & 0 & \frac{\partial N_J^q}{\partial x} & 0 & \dots & \frac{\partial N_K^q}{\partial x} & 0 \\ 0 & \frac{\partial N_I^q}{\partial y} & 0 & \frac{\partial N_J^q}{\partial y} & \dots & 0 & \frac{\partial N_K^q}{\partial y} \\ \frac{\partial N_I^q}{\partial y} & \frac{\partial N_I^q}{\partial x} & \frac{\partial N_J^q}{\partial y} & \frac{\partial N_J^q}{\partial x} & \dots & \frac{\partial N_K^q}{\partial y} & \frac{\partial N_K^q}{\partial x} \end{pmatrix}, q \in 1,2,\dots,M.$$

代码如下所示。

```
function [B]=bm(dp,elem,sp)
  % 组装每个单元的 B 矩阵
  % dp   - 单元形函数在单元重心处的导数（cell 数组），每个 cell 内部为 (n_node × 2) 矩阵
  % elem - 单元节点信息（用于指明单元节点数）
  % sp   - 空间维度
  % B    - B 矩阵，采用 cell 数组形式保存，每个 cell 存储对应单元的 B 矩阵
  nElem = size(elem, 1);     % 单元数
  B = cell(nElem, 1);        % 初始化 B 矩阵 cell 数组
  for i = 1:nElem
    % 取出第 i 个单元的导数矩阵（n_node×2），其中各行为节点在 x,y 方向的形函数导数
    deriv = dp{i};
    nNodes = size(deriv, 1);
    % 初始化当前单元 B 矩阵，尺寸 3 x (2*nNodes)
    Bi = zeros(3, sp * nNodes);
    % 依次将各节点的导数按照要求填入 B 矩阵中：
    %   第一行：存放 dN/dx，隔一个位置后置 0
    %   第二行：存放 dN/dy，前面插入 0
    %   第三行：左右排列，依次是 dN/dy 和 dN/dx
    for j = 1:nNodes
      Bi(1, 2*j-1) = deriv(j, 1);    % dN_j/dx
      Bi(2, 2*j)   = deriv(j, 2);    % dN_j/dy
      Bi(3, 2*j-1) = deriv(j, 2);    % dN_j/dy
      Bi(3, 2*j)   = deriv(j, 1);    % dN_j/dx
    end
    B{i} = Bi;
  end
end
```

1.2.6 刚度矩阵的定义与组装（K_matrix 函数）

通过 K_matrix 函数实现单元刚度矩阵的计算与总刚度矩阵的组装。输入信息包括单元 B 矩阵，单元节点信息 elem，节点坐标 x_a，单元面积 Area，材料属性 properties。输出信息为全局刚度矩阵 K。针对单点积分的情况，有单元刚

度矩阵

$$K_e = \int_{\Omega_e} B^T C B \mathrm{d}\Omega = B^T C B \times \mathrm{Area},$$

其中 Area 为单元面积，B 为上节中求解的矩阵，C 为材料本构矩阵，对于平面应力问题有

$$C = \frac{E}{1-\nu^2}\begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \dfrac{1-\nu}{2} \end{pmatrix}.$$

在计算出单元刚度矩阵后，通过对应单元的节点编号，组集得到总体的刚度矩阵。代码如下所示。

```
function [K]=K_matrix(B,elem,x_a,Area,properties)
    % K_matrix: 组装全局刚度矩阵
    % B        - 单元 B 矩阵，cell 数组形式，每个 cell 内为单元的 B 矩阵
    % elem     - 单元节点信息
    % x_a      - 全部节点坐标矩阵，每行为一个节点坐标 [x, y]
    % Area     - 单元面积
    % properties- 材料属性向量，这里要求 properties(1)=E, properties(2)=nu
    % K        - 全局刚度矩阵
    % 二维问题，每个节点有 2 个自由度（u_x, u_y）
    nNodes = size(x_a, 1);
    ndof = nNodes * 2;
    K = zeros(ndof, ndof);
    % 构造材料本构矩阵 C
    % 采用平面应力情况下的 C 矩阵：
    E = properties(1);
    nu = properties(2);
    C = E / (1 - nu^2) * [1,   nu,        0;
                          nu,  1,         0;
                          0,   0,  (1 - nu) / 2];
    nElem = size(elem, 1);
    for e = 1:nElem
        % 取出第 e 个单元的节点编号（忽略填充为 0 的部分）
        nodes = elem(e, :);
        nodes = nodes(nodes > 0);
        nElemNodes = length(nodes);
        % 构造单元的全局自由度数组：对于节点 i，其自由度编号为 2*i-1 (u_x)和 2*i
(u_y)
        dof = zeros(1, nElemNodes * 2);
```

```
        for iNode = 1:nElemNodes
            globalNode = nodes(iNode);
            dof(2*iNode-1) = 2 * globalNode - 1;
            dof(2*iNode)   = 2 * globalNode;
        end
        % 取出该单元的 B 矩阵，存储在 cell 数组 B{e}
        Bi = B{e};
        % 计算单元刚度矩阵 Ke = B'*C*B*Area
        Ke = Bi' * C * Bi * Area(e);
        % 装配到全局刚度矩阵 K 中（注意索引映射）
        K(dof, dof) = K(dof, dof) + Ke;
    end
end
```

1.2.7 外力条件的计算（F_vector 函数）

在之前设定了应力边界条件后，通过该函数装配全局力向量。其中输入参数包括节点坐标 x_a，施加载荷 Load，节点对应的边界面积 l_area，输出信息为全局力向量 F。根据节点的边界面积进行力的分配。代码如下所示。

```
function [F]=F_vector(x_a,Load,l_area)
    % F_vector: 装配全局力向量
    % x_a    : 节点坐标矩阵，尺寸为 (nNodes x 2)
    % Load   : 施加在边界上的 traction [Fx, Fy]
    % l_area : 每个节点对应的边界面积（或边界线长），尺寸 (nNodes x 1)
    % F      : 全局力向量，尺寸为 (2*nNodes, 1)，对应于每个节点的 [Fx; Fy]
    nNodes = size(x_a, 1);
    % 初始化全局力向量，二维问题每个节点有两个自由度
    F = zeros(2 * nNodes, 1);
    % 对于每个节点，把对应的边界面积与施加的 traction 相乘
    for i = 1:nNodes
        % 若节点不在边界（l_area(i)==0），其贡献自然为 0
        F(2*i-1) = l_area(i) * Load(1);  % x 方向分量
        F(2*i)   = l_area(i) * Load(2);  % y 方向分量
    end
end
```

1.2.8 强制边界条件的施加（Enforce_BC 函数）

在进行求解前，需要根据位移边界条件对总体刚度矩阵进行改装，在这里采用“赋 0 赋 1”法。将对应节点的对角线设置为 1，对应的行列设置为 0，外力向量设置为对应的位移值。

```
function [F,K]=Enforce_BC(F,K,boundary,disp,x_a)
    % Enforce_BC: 施加位移边界条件，通过修正刚度矩阵 K 与力向量 F 来保证在求解时满足 Dirichlet 边界条件
```

```
    % F      - 原始全局力向量
    % K      - 原始全局刚度矩阵
    % boundary- 每个节点每个自由度的约束标志矩阵 (nNodes x 2)，取值 1 表示受约束
    % disp   - 每个节点每个自由度的预定位移 (nNodes x 2)
    % x_a    - 节点坐标矩阵（本函数中未直接用到）
    % F      - 修正后的全局力向量
    % K      - 修正后的全局刚度矩阵
    nNodes = size(x_a, 1); % 节点数
    nDOFs = 2 * nNodes;
    % 对于二维问题，每个节点有 2 个自由度
    for dof = 1:nDOFs
        if boundary(dof) == 1  % 如果第 dof 个自由度受约束
            % 将刚度矩阵的对应行和列置零，对角线置 1
            K(dof, :) = 0;
            K(:, dof) = 0;
            K(dof, dof) = 1;
            % 将力向量赋值为预设的位移值
            F(dof) = disp(dof);
        end
    end
end
```

1.3 结果分析

改写程序后，按照题 2 中的工况进行设置。值得注意的是，该问题为平面应力问题，但是在后处理文件 constitutive.m 中，采用的是平面应变问题的材料本构矩阵，因此需要进行修改。

在本问题中，边界条件施加为 A(1,:)=[1 0 3 0]; B(1,:)=[2 max(x_a(:,2))]; Load = [0, -2e4]。

1.3.1 自编程序计算结果

分别采用三角形与四边形单元，考虑了不同网格数量的模拟，结果如图 2 所示。我们可以发现，当单元数量过少时，算出的结果与其它情况有很大的偏差。我们可以通过增加网格数量来增加计算精度。针对不同类型单元中 A 点 y 方向位移如表 1 所示，其中 A 点坐标为(2,1)。我们可以发现随着网格数量的增加，位移结果也趋近于稳定，同时经过验证，该结果也与 Abaqus 模拟中的-0.0285 相近。在下一节中，我们将考虑不同的网格数来探究收敛率。

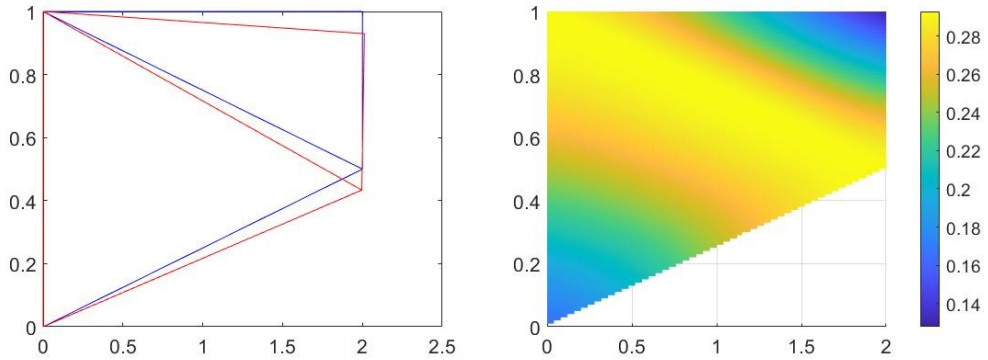


图 2 三角形单元 2 个单元位移图与压力云图

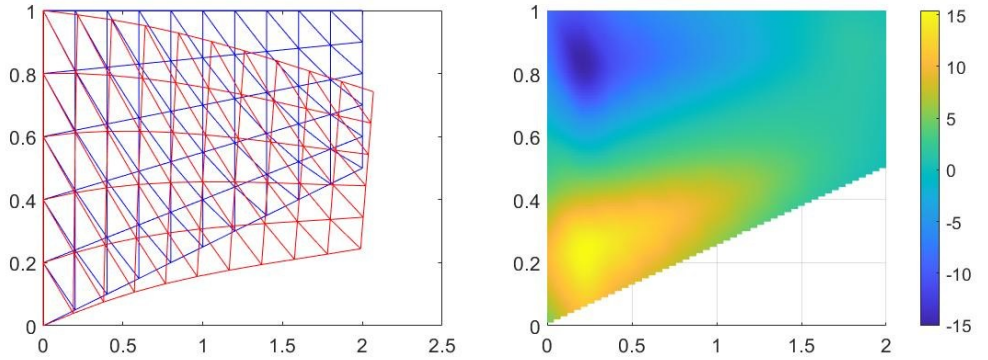


图 3 三角形单元 100 个单元位移图与压力云图

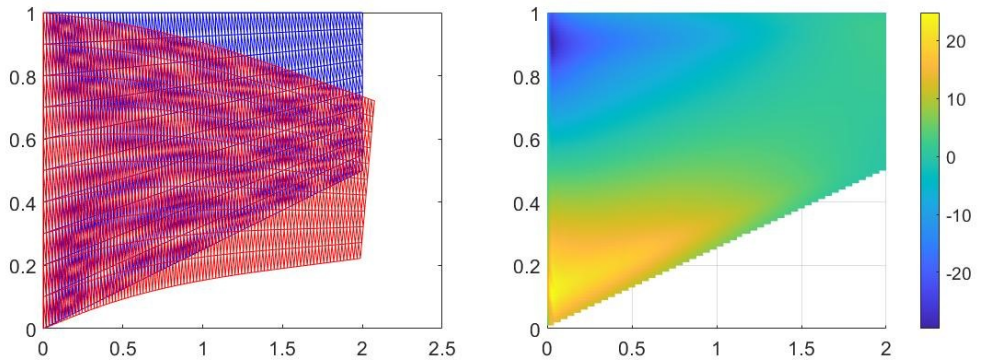


图 4 三角形单元 2000 个单元位移图与压力云图

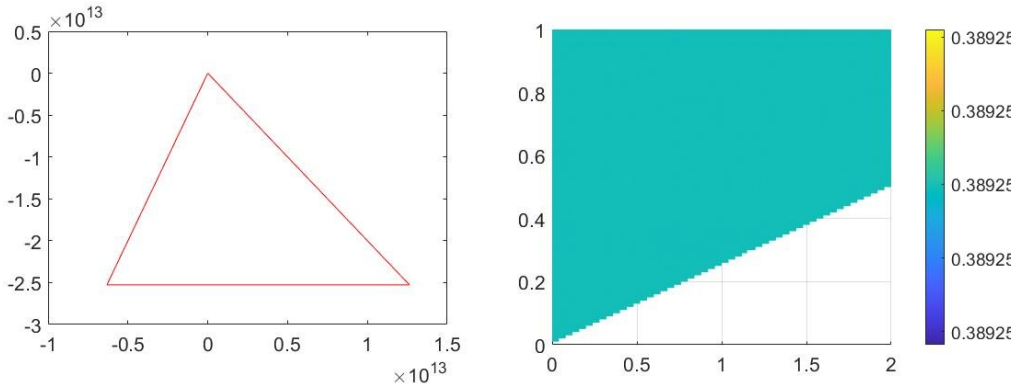


图 5 四边形单元 1 个单元位移图与压力云图

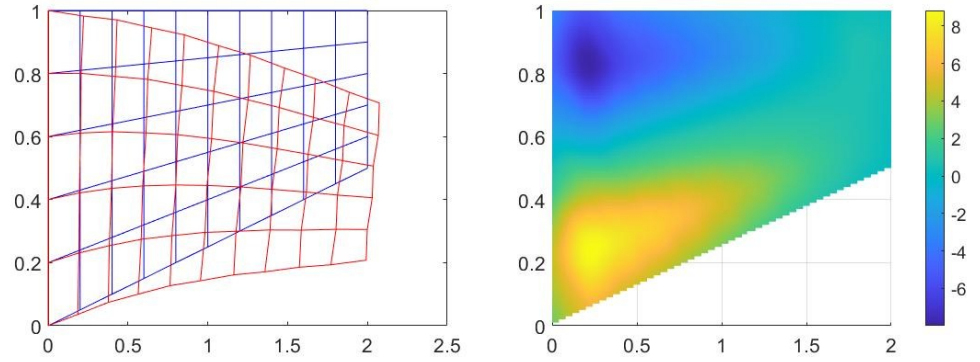


图 6 四边形单元 50 个单元位移图与压力云图

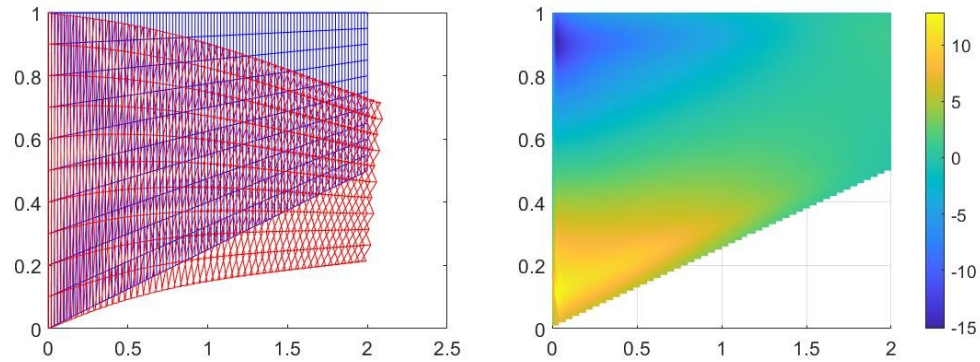


图 7 四边形单元 1000 个单元位移图与压力云图

	2 或 1	100 或 50	2000 或 1000
三角形单元	-0.00704068061680902	-0.025767014051750	-0.0279354967557116
四边形单元	-2.526451350588235e+12	-0.029343668663840	-0.028701401614540

表 1 不同类型单元中 A 点 y 方向位移

1.3.2 自编程序收敛性分析

我们考虑使用 A 点的 y 方向位移进行收敛性分析。分别考虑三角形单元与四边形单元。为了统一标准，再次设定 x 方向划分网格的数量 nx 为 y 方向划分网格数量 ny 的两倍。使用 1/ny 作为网格精细度的表述标准。

由于

$$|u_h - u| = ch^{\alpha},$$

其中u_h为数值计算得到 A 点 y 方向的位移，u为真实位移，在这里我们采用 ny 取 50 时的解作为近似真实解，通过对方程两边取对数，可以得到

$$\ln|u_h - u| = \alpha \ln h + \ln c,$$

其中斜率α即为收敛率。

首先考察三角形单元，A 点 y 方向的位移随网格精细度变化以及收敛率拟合如图 8 所示。收敛率为 2.07784。

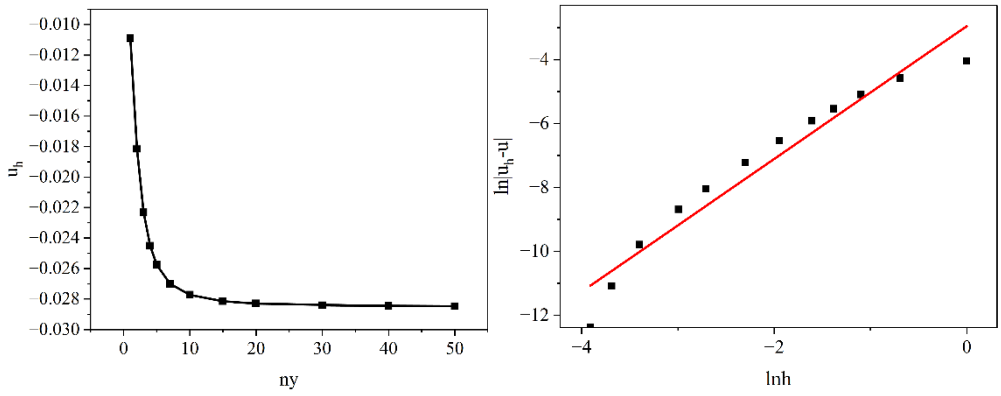
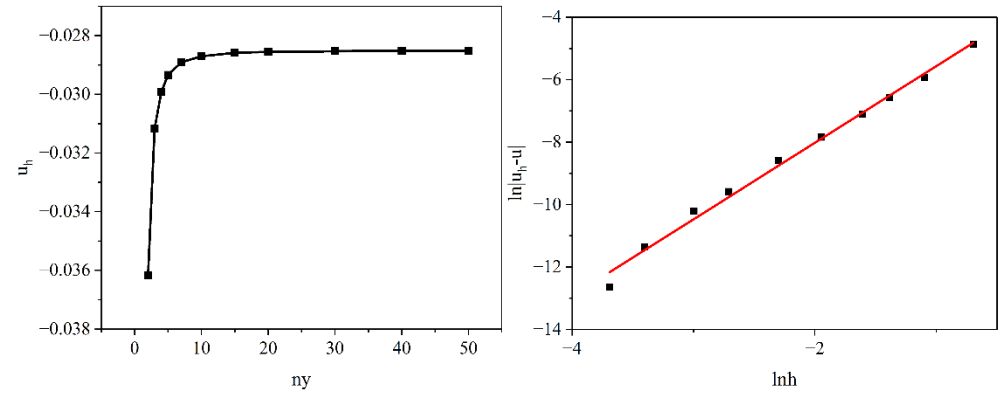


图 8 三角形单元位移随网格精细度变化以及收敛率拟合

接下来考察四边形单元，A 点 y 方向的位移随网格精细度变化以及收敛率拟合如所示。收敛率为 2.45313。



1.4 收获与展望

通过与 Abaqus 结果对比以及收敛性分析，此版本改写程序效果较好，且随着网格数的增加，迅速收敛到真实解。值得注意的是，我们计算得到四边形单元的收敛性相较于三角形单元更好一些，并且收敛的方向也是不同的，也就是说，三角形单元的计算变形结果偏小，而四边形单元的计算变形结果偏大。

当然该程序也存在一定的不足：无法对指定边界上指定区域加载边界条件（这一问题将在下个章节中解决）；未考虑四边形单元中的完全积分情况；对于一些参数的设定需要在对应的子程序中修改；求解问题仅限于二维的线弹性问题；设定的几何形状不够普适。将会在之后的工作中进行改进。

2 超弹性稳态问题

2.1 介绍

本章节介绍的 MATLAB 程序，可以实现二线超弹性稳态问题的求解，其中默认采用近不可压缩 Neo-Hookean 材料。目前能够实现的可调节功能有：模拟单元的设置、网格划分节点数的设置、单元类型的选取、边界条件的设置。

2.2 程序结构

该代码的主要结构为：网格划分与处理、设置边界条件、设置材料属性、单元 B 矩阵与刚度矩阵的计算、外力条件的施加、强制边界条件的施加、总体方程的求解、迭代计算、后处理（应力应变的计算等）。整体流程图如图 9 图 1 所示。

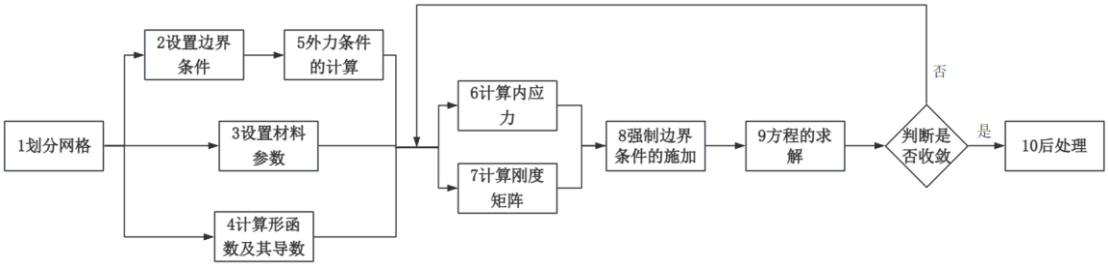


图 9 超弹性稳态问题 MATLAB 代码流程图

2.2.1 划分网格（generate_mesh 函数）

划分网格基本原理与上章方式相同，由于在本问题中考虑的为正方形板，因此将长度与划分网格数定义为全局变量，可在主程序中直接修改划分网格数量。输入信息包括 x 与 y 方向的长度以及划分网格数，划分网格单元类型，输出的为节点坐标 x_a 以及单元节点信息 elem。

```
function [x_a, elem] = generate_mesh(Lx, Ly, nx, ny, flag)
% Lx: x 方向的长度(mm)
% Ly: y 方向的长度(mm)
% nx: 沿着 x 方向划分网格数
% ny: 沿着 y 方向划分网格数
% flag: 单元类型 (1 为三角形, 2 为四边形)
% 设置节点
x_a = zeros((nx+1)*(ny+1), 2);
index = 0;
for ix = 0:nx
    xi = ix / nx;
```

```
    xcoord = Lx * xi;  % x 坐标从 0 到 Lx(mm)
    for iy = 0:ny
        yi = iy / ny;
        ycoord = Ly * yi;  %y 坐标从 0 到 Ly(mm)
        index = index + 1;
        x_a(index, :) = [xcoord, ycoord];
    end
end
% 建立单元划分
elem = [];
for ix = 1:nx
    for iy = 1:ny
        n1 = (ix-1)*(ny+1) + iy;
        n2 = n1 + 1;
        n3 = ix*(ny+1) + iy;
        n4 = n3 + 1;
        if flag == 1
            % 三角形单元
            elem(end+1, :) = [n1, n3, n2];
            elem(end+1, :) = [n2, n3, n4];
        else
            % 四边形单元
            elem(end+1, :) = [n1, n3, n4, n2];
        end
    end
end
end
```

2.2.2 高斯点物理坐标和单元面积计算（g_center 函数）

与 1.2.2 高斯点物理坐标和单元面积计算（g_center 函数）中函数相同。

2.2.3 设置边界条件（displa、dist 函数）

其中力边界条件设置函数 dist 与 1.2.3 设置边界条件（displa、dist 函数）中函数相同。而在本问题中考虑在部分区域施加了位移边界条件，因此对 displa 函数进行优化。A 矩阵改为含有六个参数，前四个参数与原本的设定相同，后两个参数规定了位移边界条件施加在另一个坐标轴上的区间范围。代码如下所示。

```
function [boundary,disp]=displa(x_a,A)
    % 初始化
    % displa: 根据 A 矩阵施加位移边界条件，支持子区域范围
 nNodes = size(x_a, 1);
    boundary = zeros(nNodes, 2);
```

```
disp     = zeros(nNodes, 2);
tol = 1e-6;
for i = 1:size(A, 1)
    axisLabel = A(i, 1);
    loc       = A(i, 2);
    direction = A(i, 3);
    value     = A(i, 4);
    % 检查是否给出了子区域范围
    if size(A,2) >= 6 && ~isnan(A(i,5)) && ~isnan(A(i,6))
        rangeMin = A(i,5);
        rangeMax = A(i,6);
    else
        rangeMin = -inf;
        rangeMax = +inf;
    end

    % 确定主轴和副轴
    if axisLabel == 1
        primaryCoord   = x_a(:,1);
        secondaryCoord = x_a(:,2);
    else
        primaryCoord   = x_a(:,2);
        secondaryCoord = x_a(:,1);
    end
    % 查找满足主轴位置和副轴范围的节点
    idxs = find(abs(primaryCoord - loc) < tol & ...
                secondaryCoord >= rangeMin - tol & ...
                secondaryCoord <= rangeMax + tol);

    % 施加约束
    for idx = idxs'
        if direction == 1 || direction == 3
            % x 方向约束
            boundary(idx, 1) = 1;
            disp(idx, 1)     = value;
        end
        if direction == 2 || direction == 3
            % y 方向约束
            boundary(idx, 2) = 1;
            disp(idx, 2)     = value;
        end
    end
end
```

```
end
```

2.2.4 设置材料参数（main 程序中）

本问题为平面应力问题，材料参数杨氏模量为$4.00889806 \times 10^{11}$Pa，剪切模量为$8.0194 \times 10^{7}$Pa，材料参数的定义放于主程序中。通过 properties 来存储材料参数。代码如下所示。

```
K0  = 400889.806e6; % initial bulk modulus
mu0 = 80.194e6; % initial shear modulus
properties(1) = K0;
properties(2) = mu0;
```

2.2.5 外力条件的计算（fext_vector 函数）

与 1.2.7 外力条件的计算（F_vector 函数）中函数相同。

2.2.6 显式求解（implicit_solver 函数）

在本程序中，采用显式求解。输入信息包括外力 F_ext。边界条件 boundary，节点坐标 x_a，材料属性 properties，单元节点信息 elem，形函数 DN 及其倒数 DNs，位移迭代初值 u_old。输出信息为求得的节点位移 u_new。通过外力减去内力作为收敛的判断条件。首先根据位移计算出节点内力，之后组集刚度矩阵，通过迭代算法得到位移增量，进而不断求解。代码如下所示。

```
function [u_new]=implicit_solver(F_ext, boundary, x_a, properties, elem, DN, DNs, u_old)
  tol = 1e-12;
  MaxIter = 10;
  u_new = u_old;
  for i = 1:MaxIter
    F_int = fint_vector(x_a, u_new, DNs, properties, elem);
    dF = F_ext - F_int;
    K = K_matrix(x_a, elem, u_new, DN, DNs, properties);
    if norm(dF) < tol   % 收敛性判断
      fprintf('Converged at iteration %d\n', i);
      break;
    end
    prescribed_disp = zeros(size(u_new));  % 位移约束
    [dF_mod, K_mod] = Enforce_BC(dF, K, boundary, prescribed_disp, x_a);
    % 增量求解
    delta_u = K_mod \ dF_mod;
    u_new = u_new + delta_u;
  end
end
```

2.2.7 计算节点内力（fint_vector 函数）

单元内部力的计算是保证正确计算的关键一个环节，输入信息包括节点位

移 x_a，节点位移 u，单元形函数导数 DNs，材料属性 properties，单元节点信息 elem。其中第二 PK 应力为

$$S = 2\frac{\partial \Psi}{\partial C} = \kappa J(J-1)C^{-1} + \mu J^{-\frac{2}{3}}\left(I - \frac{1}{3}I_1 C^{-1}\right),$$

其中第一项为提及部分应力，第二项为等容部分应力。第一 PK 应力可直接通过

$$P = FS,$$

求得。之后将每个单元求得的内力进行组装，即可得到全局的内力向量。代码如下所示。

```
function [Fint] = fint_vector(x_a, u, DNs, properties, elem)
    [nodes, ~] = size(x_a);
    Fint = zeros(2*nodes, 1);
    mu = properties(2);
    kappa = properties(1);
    for e = 1:size(elem,1)
        nodes_e = elem(e,:);
        n_nodes = length(nodes_e);    % 单元节点数（3 或 4）
        Xe = x_a(nodes_e,:);          % 单元参考坐标 [n_nodes x 2]
        Ue = u(reshape([2*nodes_e-1; 2*nodes_e], [], 1)); % 位移向量 [2*n_nodes x 1]
        % 获取形状函数导数 [n_nodes x 2] (dN/dx, dN/dy)
        gradN = DNs{e};
        %严格计算位移梯度张量 F = I + Grad(u)
        grad_ux = gradN' * Ue(1:2:end); % [2 x 1]
        grad_uy = gradN' * Ue(2:2:end); % [2 x 1]
        GradU = [grad_ux, grad_uy]';    % [2 x 2] 矩阵
        F = eye(2) + GradU;             % 变形梯度
        %计算第二类 Piola-Kirchhoff 应力 S
        J = det(F);
        C = F' * F;                     % 右 Cauchy-Green 张量
        I1 = trace(C);
        C_inv = inv(C);
        % 体积部分和等容部分应力
        S_vol = kappa * (J - 1) * J * C_inv;
        S_iso = mu * J^(-2/3) * (eye(2) - (1/3)*I1 * C_inv);
        S = S_iso + S_vol; % 总应力
        %第一类 Piola-Kirchhoff 应力 P = F*S
        P = F * S; % [2 x 2]
        %将 P 转换为 Voigt 形式 [3 x 1]：P_voigt = [P11; P22; P12]
        P_voigt = [P(1,1); P(2,2); P(1,2)];
        %组装 B 矩阵（兼容三角形和四边形单元）
```

```
        B = zeros(3, 2*n_nodes);
        for i = 1:n_nodes
            dN_dx = gradN(i,1);
            dN_dy = gradN(i,2);
            % B 矩阵结构：
            % [dN/dx  0     ... ;
            %  0     dN/dy ... ;
            % dN/dy dN/dx ... ]
            B(1, 2*i-1) = dN_dx;
            B(2, 2*i)   = dN_dy;
            B(3, 2*i-1) = dN_dy;
            B(3, 2*i)   = dN_dx;
        end
        %单元内部力计算（考虑单元面积）
        [~, Area_e] = g_center(x_a, elem(e,:)); % 获取单元面积
        F_int_e = B' * P_voigt * Area_e;       % [2*n_nodes x 1]
        %组装到全局力向量
        indices = reshape([2*nodes_e-1; 2*nodes_e], [], 1);
        Fint(indices) = Fint(indices) + F_int_e;
    end
end
```

2.2.8 计算刚度矩阵（K_matrix 函数）

在本节中组集总的刚度矩阵，基本原理与上章中的函数相同，但是需要进行修改。因为刚度矩阵为四阶张量，我们需要将其转化为二阶矩阵进行运算。同时，四阶张量刚度矩阵为

$$C^{tan} = 2\frac{\partial S}{\partial C} = 4\frac{\partial^2 \Psi}{\partial C^2}$$

$$= \frac{2}{3}\mu J^{-\frac{2}{3}}\left[-I \otimes C^{-1} - C^{-1} \otimes I + \frac{I_1}{2}\left(C^{-1} \underline{\otimes} C^{-1} + C^{-1} \overline{\otimes} C^{-1}\right) + \frac{I_1}{3} C^{-1} \otimes C^{-1}\right].$$

代码如下所示。

```
function [K] = K_matrix(x_a, elem, u_new, DN, DNs, properties)
    [nodes, ~] = size(x_a);
    K = sparse(2*nodes, 2*nodes);
    mu = properties(2);
    kappa = properties(1);
    I2 = eye(2);  % 提前定义单位阵
    for e = 1:size(elem,1)
        nodes_e = elem(e,:);
        n_nodes = length(nodes_e);
        Xe = x_a(nodes_e,:);
```

```
dofs = reshape([2*nodes_e - 1; 2*nodes_e], [], 1);
Ue = u_new(dofs); % 真实位移
gradN = DNs{e};
%计算变形梯度
grad_ux = gradN' * Ue(1:2:end);
grad_uy = gradN' * Ue(2:2:end);
GradU = [grad_ux, grad_uy]';
F = eye(2) + GradU;
J = det(F);
%C 张量
C = F' * F;
C_inv = inv(C);
I1 = trace(C);
%构造四阶刚度张量 Ctan 并映射为 4x4 矩阵
Ctan = zeros(2,2,2,2);
for m = 1:2
   for n = 1:2
      for p = 1:2
         for q = 1:2
            % 等容部分
            term_iso = (2/3) * mu * J^(-2/3) * ( ...
               - I2(m,n)*C_inv(p,q) - C_inv(m,n)*I2(p,q) + ...
               0.5*(C_inv(m,p)*C_inv(n,q) + C_inv(m,q)*C_inv(n,p)) + ...
               (1/3)*I1*C_inv(m,n)*C_inv(p,q) );
            % 体积部分
            term_vol = kappa * J * ( ...
               (2*J - 1)*C_inv(m,n)*C_inv(p,q) - ...
               (J - 1)*(C_inv(m,p)*C_inv(n,q) + C_inv(m,q)*C_inv(n,p)) );
            Ctan(m,n,p,q) = term_iso + term_vol;
         end
      end
   end
end
%构造 B 矩阵（4x2n）
B = zeros(4, 2*n_nodes);
for i = 1:n_nodes
   dN_dx = gradN(i,1);
   dN_dy = gradN(i,2);
   B(1, 2*i-1) = dN_dx;
   B(2, 2*i)   = dN_dy;
   B(3, 2*i-1) = dN_dy;
   B(3, 2*i)   = dN_dx;
   B(4, 2*i-1) = dN_dx;
```

```
            B(4, 2*i)   = dN_dy;
        end
        %映射四阶张量为 D 矩阵 (4x4)
        D = zeros(4,4);
        for I = 1:2
            for J = 1:2
                for P = 1:2
                    for Q = 1:2
                        row = 2*(I-1) + J;
                        col = 2*(P-1) + Q;
                        D(row, col) = Ctan(I,J,P,Q);
                    end
                end
            end
        end
        %单元面积（积分权重）
        [~, Area_e] = g_center(x_a, elem(e,:));
        %单元刚度矩阵
        Ke = B' * D * B * Area_e;
        %组装全局刚度矩阵
        indices = reshape([2*nodes_e-1; 2*nodes_e], [], 1);
        K(indices, indices) = K(indices, indices) + Ke;
    end
end
```

2.2.9 强制边界条件的施加（Enforce_BC 函数）

与 1.2.8 强制边界条件的施加（Enforce_BC 函数）中函数相同。

2.2.10 应力计算（getStress 函数）

在本节中，根据求得的位移，计算出节点上的 Cauchy 应力、第一 PK 应力以及第二 PK 应力。代码如下所示。

```
function [sigma, p]=getStress(x_a, u, DNs, properties, elem)
% sigma: Cauchy stress
% p: pressure
    [elements, ~] = size(elem);
    sigma = zeros(elements, 3); % 存储 σ _xx, σ _yy, σ _xy
    p = zeros(elements, 1);
    mu = properties(2);
    kappa = properties(1);
    for e = 1:elements
        nodes_e = elem(e,:);
        n_nodes = length(nodes_e);     % 单元节点数（3 或 4）
        Xe = x_a(nodes_e,:);           % 参考坐标 [n_nodes x 2]
```

```
        Ue = u(reshape([2*nodes_e-1; 2*nodes_e], [], 1)); % 位移向量 [2n_nodes x 1]
        % 获取形状函数导数矩阵 [n_nodes x 2]
        gradN = DNs{e};
        % 严格计算位移梯度张量 F = I + Grad(u)
        grad_ux = gradN' * Ue(1:2:end); % [2 x 1] (dUx/dX, dUx/dY)
        grad_uy = gradN' * Ue(2:2:end); % [2 x 1] (dUy/dX, dUy/dY)
        GradU = [grad_ux, grad_uy]';    % [2 x 2] 矩阵
        F = eye(2) + GradU;             % 变形梯度
        J = det(F);                     % Jacobian 行列式
        % 右 Cauchy-Green 张量 C = F'*F
        C = F' * F;
        I1 = trace(C);
        C_inv = inv(C);
        % 第二类 Piola-Kirchhoff 应力 S
        S_vol = kappa * (J - 1) * J * C_inv;  % 体积部分
        S_iso = mu * J^(-2/3) * (eye(2) - (1/3)*I1 * C_inv); % 等容部分
        S = S_iso + S_vol;                    % 总应力
        % 第一类 Piola-Kirchhoff 应力 P = F*S
        P = F * S;                            % [2 x 2]
        % Cauchy 应力 σ = (1/J) * P * F'
        sigma_e = (1/J) * P * F';             % [2 x 2]
        sigma(e, :) = [sigma_e(1,1), sigma_e(2,2), sigma_e(1,2)];
        p(e) = -trace(sigma_e)/3;             % 压力（负静水压）
    end
end
```

2.3 结果分析

改写程序后，按照题 4 中的工况进行设置。

在本问题中，边界条件施加为 A = zeros(4,6); A(1,1:6) = [1, 0, 1, 0, 0, 10]; A(2,1:6) = [2, 0, 2, 0, 0, 10]; A(3,1:6) = [2, 10, 1, 0, 0, 10]; A(4,1:6) = [2, 10, 2, -v, 0, 5]; B(1,:)=[0 0]。

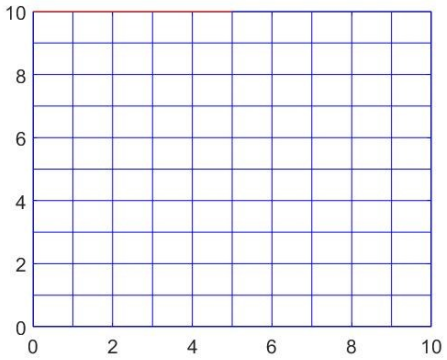


图 10 超弹性计算结果

然而计算结果如图 10 超弹性计算结果所示，结果很难达到收敛。通过逐步分析判断得到发散的原因为在显式迭代求解过程中，位移增量计算并不收敛，很快出先虚部，这是因为在计算过程中雅可比行列式变为负数，进而是的内力计算出现虚部。针对该问题目前还没有较好的解决方案。本人也尝试使用 Abaqus 进行模拟，但可惜结果也不收敛。

2.4 收获与展望

在本节中，研究了超弹性稳态问题，通过编程熟悉了迭代求解的思路，通过亲自实践更加直观的理解了课堂上学习的理论知识，并且了解了内力计算以及刚度矩阵计算的思想。在本节中增加的边界条件设置方式可以灵活运用于上一章中。但是最终的程序未能收敛，这也是之后要解决的问题。