新型非对称八节点平面单元的有限元代码实现及效果检验

张靖雅，张家瑞，王奕

目录

一、原理

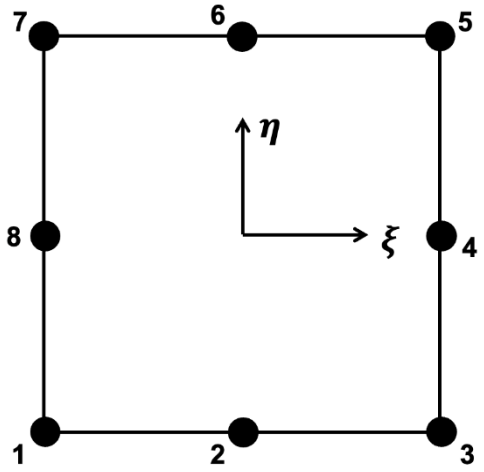


图 1 八节点平面单元示意图

在图 1 所示的八节点平面单元中，要求该单元能够准确复现完备的二次多项式位移场

$$u(x,y)=a_1+a_2x+a_3y+a_4x^2+a_5xy+a_6y^2+a_7x^2y+a_8xy^2 \tag{1}$$

$$v(x,y)=b_1+b_2x+b_3y+b_4x^2+b_5xy+b_6y^2+b_7x^2y+b_8xy^2 \tag{2}$$

前六项构成了完备的二次多项式，而最后两项只是为了使单项式项的数量与该单元的节点总数保持一致，故其的选择具有一定的任意性。设插值函数为M_i，则位移场可被近似表示为

$$\hat{u}^{(e)}(x,y)=\sum_{i=1}^{8}M_i\hat{u}_i^{(e)} \tag{3}$$

$$\hat{v}^{(e)}(x,y)=\sum_{i=1}^{8}M_i\hat{v}_i^{(e)} \tag{4}$$

$\hat{u}_i^{(e)},\hat{v}_i^{(e)}$为第 i 个单元的节点位移值。为了使该单元能够在其内部任意一点精确复现由式(1)和式(2)所给出的多项式位移场，必须满足以下条件：

$$u(x,y)=\hat{u}^{(e)}(x,y) \tag{5}$$

$$v(x,y)=\hat{v}^{(e)}(x,y) \tag{6}$$

经典的 Serendipity 八节点单元形函数为

$$\mathbf{N}=[N_1,N_2,...,N_8] \tag{7}$$

$$N_i=\frac{1}{4}(1+\xi\xi_i)(1+\eta\eta_i)(\xi\xi_i+\eta\eta_i-1) \tag{8}$$

$$N_j=\frac{1}{2}\left(1-\xi^2\right)\left(1+\eta\eta_j\right) \tag{9}$$

$$N_k=\frac{1}{2}\left(1+\xi\xi_k\right)\left(1-\eta^2\right) \tag{10}$$

其中ξ,η为自然坐标，i为角节点，j为$\xi=0$对应的边中点，k为$\eta=0$对应的边中点。此外，由式(5)和式(6)可以推出，形函数须满足八个条件

$$\sum_{i=1}^{8}M_i=1 \tag{11}$$

$$\sum_{i=1}^{8}M_i x_i=x \tag{12}$$

$$\sum_{i=1}^{8}M_i y_i=y \tag{13}$$

$$\sum_{i=1}^{8}M_i x_i^2=x^2 \tag{14}$$

$$\sum_{i=1}^{8}M_i x_i y_i=xy \tag{15}$$

$$\sum_{i=1}^{8}M_i y_i^2=y^2 \tag{16}$$

$$\sum_{i=1}^{8}M_i x_i^2 y_i=x^2 y \tag{17}$$

$$\sum_{i=1}^{8}M_i x_i y_i^2=xy^2 \tag{18}$$

则在度量坐标下形函数也可以表示为

$$\mathbf{M}=\mathbf{P}^{-1}\mathbf{p}\left(x,y\right) \tag{19}$$

其中

$$\mathbf{M}=\left[M_1,M_2,...,M_8\right]^{\mathrm{T}} \tag{20}$$

$$\mathbf{p}\left(x,y\right)=\left[1,x,y,x^2,xy,y^2,x^2y,xy^2\right]^{\mathrm{T}} \tag{21}$$

$$\mathbf{P}=\begin{bmatrix} 1 & \cdots & 1 \\ x_1 & \cdots & x_8 \\ y_1 & \cdots & y_8 \\ x_1^2 & \cdots & x_8^2 \\ x_1y_1 & \cdots & x_8y_8 \\ y_1^2 & \cdots & y_8^2 \\ x_1^2y_1 & \cdots & x_8^2y_8 \\ x_1y_1^2 & \cdots & x_8y_8^2 \end{bmatrix} \tag{22}$$

为了区分，将式(19)-(22)表示的形函数称为度量形函数，式(7)-(10)表示的形函数称为参数形函数。

根据连续性要求，形函数必须使位移模型在单元内部和单元之间至少分别满足 C^1 和 C^0 的连续性条件，这两种形函数都满足单元内部连续性的最低要求。对于八节点单元，若要实现单元间的位移连续性，需满足在边界上的任意一点，所有不位于该边界上的节点的形函数值之和必须为零。当单元几何为正方形或矩形时，度量形函数满足这一要求，但对于任意形状的单元，并不能保持满足这一要求。参数形函数能在任意几何形状下满足单元间位移连续性要求，但其无法满足二次多项式完备性的要求。理想情况下，我们希望有一组形函数能够同时满足完备性要求和在任意形状的单元几何下实现单元间的位移连续性。然而，要构造这样的形函数是非常困难的，而且对于当前的公式推导而言，这样的形函数也并非必需。

在虚功原理

$$\int_\Omega \delta\boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{\sigma}\mathrm{d}\Omega-\int_\Omega \delta\mathbf{u}^{\mathrm{T}}\mathbf{b}\mathrm{d}\Omega-\int_\Gamma \delta\mathbf{u}^{\mathrm{T}}\mathbf{t}\mathrm{d}\Gamma-\delta\mathbf{u}_c^{\mathrm{T}}\mathbf{f}_c=0 \tag{23}$$

中，包含虚应变与虚位移项，故虚位移必须至少在单元内部满足 C^1 阶的连续性，在单元间满足 C^0 阶连续性。在单元内部满足 C^1 阶的连续性，只需确保位移场中包含式(1)(2)右侧的前三项，因此并不一定要满足更高阶的完备性要求，可以使用参数形函数来对虚位移进行插值。

$$\delta\mathbf{u}^{(e)}=\delta\overline{\mathbf{u}}^{(e)}=\mathbf{N}\delta\overline{\mathbf{u}}_n^{(e)} \tag{24}$$

$$\delta\boldsymbol{\varepsilon}^{(e)}=\delta\overline{\boldsymbol{\varepsilon}}^{(e)}=\overline{\mathbf{B}}\delta\overline{\boldsymbol{\varepsilon}}_n^{(e)} \tag{25}$$

$$\overline{\mathbf{B}}=\mathbf{L}\mathbf{N} \tag{26}$$

式(23)中的 $\boldsymbol{\sigma}$ 为真实应力场，若使用 $\overline{\boldsymbol{\sigma}}=\mathbf{D}\overline{\boldsymbol{\varepsilon}}$，则会引入二阶的误差，使用度量形函数可以提高插值应力场的精度

$$\hat{\mathbf{u}}^{(e)} = \mathbf{M}\hat{\mathbf{u}}_n^{(e)} \tag{27}$$

$$\hat{\boldsymbol{\varepsilon}}^{(e)} = \hat{\mathbf{B}}\hat{\boldsymbol{\varepsilon}}_n^{(e)} \tag{28}$$

$$\hat{\mathbf{B}} = \mathbf{LM} \tag{29}$$

$$\hat{\boldsymbol{\sigma}}^{(e)} = \mathbf{D}\hat{\boldsymbol{\varepsilon}}^{(e)} \tag{30}$$

且无论单元几何形状如何，都可以满足条件。则虚功原理可写为

$$\mathbf{A}_{e=1}^{M}\left(\begin{array}{l}\int_{\Omega^{(e)}} \delta\bar{\mathbf{u}}_n^{(e)^{\mathrm{T}}}(\bar{\mathbf{B}}^{\mathrm{T}}\mathbf{D}\hat{\mathbf{B}})\hat{\mathbf{u}}_n^{(e)}\mathrm{d}\Omega^{(e)} - \int_{\Omega^{(e)}} \delta\bar{\mathbf{u}}_n^{(e)^{\mathrm{T}}}\mathbf{N}^{\mathrm{T}}\mathbf{b}^{(e)}\mathrm{d}\Omega^{(e)} \\ -\int_{\Gamma^{(e)}} \delta\bar{\mathbf{u}}_n^{(e)^{\mathrm{T}}}\mathbf{N}^{\mathrm{T}}\mathbf{t}^{(e)}\mathrm{d}\Gamma^{(e)}\end{array}\right) = 0 \tag{31}$$

通过执行常规的有限元组装过程，并利用虚位移的任意性，可以得到如下系统方程

$$\mathbf{KU} = \mathbf{F} \tag{32}$$

其中

$$\mathbf{K} = \mathbf{A}_{e=1}^{N}(\mathbf{K}^{(e)}) \tag{33}$$

$$\mathbf{F} = \mathbf{A}_{e=1}^{N}(\mathbf{f}_b^{(e)} + \mathbf{f}_t^{(e)}) \tag{34}$$

$$\mathbf{K}^{(e)} = \int_{\Omega^{(e)}} \bar{\mathbf{B}}^{\mathrm{T}}\mathbf{D}\hat{\mathbf{B}}\mathrm{d}\Omega^{(e)} \tag{35}$$

$$\mathbf{f}_b^{(e)} = \int_{\Omega^{(e)}} \mathbf{N}^{\mathrm{T}}\mathbf{b}^{(e)}\mathrm{d}\Omega^{(e)} \tag{36}$$

$$\mathbf{f}_t^{(e)} = \int_{\Gamma^{(e)}} \mathbf{N}^{\mathrm{T}}\mathbf{t}^{(e)}\mathrm{d}\Gamma^{(e)} \tag{37}$$

相比于传统的 Galerkin 法选择相同的试函数与插值函数，这里试函数与插值函数的选择不同，刚度矩阵 $\mathbf{K}$ 非对称。

二、刚度矩阵的计算

要具体求出度量形函数的解析表达式十分复杂，由于只在确定刚度矩阵时涉及到度量形函数，则只需求出在各个积分点上度量形函数的导数的值即可。接下来，分别求出度量形函数和参数形函数对应的 $\mathbf{B}$ 矩阵。

度量形函数对应的 $\hat{\mathbf{B}}$

$$\hat{\mathbf{B}} = \begin{bmatrix} \partial\mathbf{M}/\partial x & \mathbf{0} \\ \mathbf{0} & \partial\mathbf{M}/\partial y \\ \partial\mathbf{M}/\partial y & \partial\mathbf{M}/\partial x \end{bmatrix} = \begin{bmatrix} \mathbf{P}^{-1}\mathbf{p}_{,x} & \mathbf{0} \\ \mathbf{0} & \mathbf{P}^{-1}\mathbf{p}_{,y} \\ \mathbf{P}^{-1}\mathbf{p}_{,y} & \mathbf{P}^{-1}\mathbf{p}_{,x} \end{bmatrix} \tag{38}$$

其中

$$\begin{aligned}\mathbf{p}_{,x}&=\left[0,1,0,2x,y,0,2xy,y^2\right]^{\mathrm{T}}\\ \mathbf{p}_{,y}&=\left[0,0,1,0,x,2y,x^2,2xy\right]^{\mathrm{T}}\end{aligned}\tag{39}$$

则在积分点 α 处

$$\left(\mathbf{P}^{-1}\mathbf{p}_{,x}\right)_{\alpha}=\begin{bmatrix}1 & \cdots & 1\\ x_1 & \cdots & x_8\\ y_1 & \cdots & y_8\\ x_1^2 & \cdots & x_8^2\\ x_1y_1 & \cdots & x_8y_8\\ y_1^2 & \cdots & y_8^2\\ x_1^2y_1 & \cdots & x_8^2y_8\\ x_1y_1^2 & \cdots & x_8y_8^2\end{bmatrix}^{-1}\begin{bmatrix}0 & 1 & 0 & 2x_{\alpha} & y_{\alpha} & 0 & 2x_{\alpha}y_{\alpha} & y_{\alpha}^2\end{bmatrix}^{\mathrm{T}}\tag{40}$$

$$\left(\mathbf{P}^{-1}\mathbf{p}_{,y}\right)_{\alpha}=\begin{bmatrix}1 & \cdots & 1\\ x_1 & \cdots & x_8\\ y_1 & \cdots & y_8\\ x_1^2 & \cdots & x_8^2\\ x_1y_1 & \cdots & x_8y_8\\ y_1^2 & \cdots & y_8^2\\ x_1^2y_1 & \cdots & x_8^2y_8\\ x_1y_1^2 & \cdots & x_8y_8^2\end{bmatrix}^{-1}\begin{bmatrix}0 & 0 & 1 & 0 & x_{\alpha} & 2y_{\alpha} & x_{\alpha}^2 & 2x_{\alpha}y_{\alpha}\end{bmatrix}^{\mathrm{T}}\tag{41}$$

参数形函数对参数坐标的导数

$$\frac{\partial N_i}{\partial\xi}=\frac{1}{4}\xi_i\left(1+\eta\eta_i\right)\left(2\xi\xi_i+\eta\eta_i\right)\tag{42}$$

$$\frac{\partial N_j}{\partial\xi}=-\xi\left(1+\eta\eta_j\right)\tag{43}$$

$$\frac{\partial N_k}{\partial\xi}=\frac{1}{2}\xi_k\left(1-\eta^2\right)\tag{44}$$

$$\frac{\partial N_i}{\partial\eta}=\frac{1}{4}\eta_i\left(1+\xi\xi_i\right)\left(\xi\xi_i+2\eta\eta_i\right)\tag{45}$$

$$\frac{\partial N_j}{\partial\eta}=\frac{1}{2}\eta_j\left(1-\xi^2\right)\tag{46}$$

$$\frac{\partial N_k}{\partial\eta}=-\eta\left(1+\xi\xi_k\right)\tag{47}$$

若采用 2×2 高斯积分，则积分点坐标为$\left(\xi_{\alpha},\eta_a\right)=\left(\pm\frac{1}{\sqrt{3}},\pm\frac{1}{\sqrt{3}}\right)$，则参数形函数

对应的$\bar{\mathbf{B}}$

$$\bar{\mathbf{B}}_\alpha = \begin{bmatrix} \partial\mathbf{N}/\partial x & \mathbf{0} \\ \mathbf{0} & \partial\mathbf{N}/\partial y \\ \partial\mathbf{N}/\partial y & \partial\mathbf{N}/\partial x \end{bmatrix} \tag{48}$$

对应至图 1 的节点编号，可显式表达为

$$\left(\frac{\partial\mathbf{N}}{\partial\xi}\right)_\alpha = \begin{bmatrix} \frac{1}{4}(1-\eta_\alpha)(2\xi_\alpha+\eta_\alpha) \\ -\xi_\alpha(1-\eta_\alpha) \\ \frac{1}{4}(1-\eta_\alpha)(2\xi_\alpha-\eta_\alpha) \\ \frac{1}{2}(1-\eta_\alpha^2) \\ \frac{1}{4}(1+\eta_\alpha)(2\xi_\alpha+\eta_\alpha) \\ -\xi_\alpha(1+\eta_\alpha) \\ \frac{1}{4}(1+\eta_\alpha)(2\xi_\alpha-\eta_\alpha) \\ -\frac{1}{2}(1-\eta_\alpha^2) \end{bmatrix} \tag{49}$$

$$\left(\frac{\partial\mathbf{N}}{\partial\eta}\right)_\alpha = \begin{bmatrix} \frac{1}{4}(1-\xi_\alpha)(\xi_\alpha+2\eta_\alpha) \\ -\frac{1}{2}(1-\xi_\alpha^2) \\ -\frac{1}{4}(1+\xi_\alpha)(\xi_\alpha-2\eta_\alpha) \\ -\eta(1+\xi_\alpha) \\ \frac{1}{4}(1+\xi_\alpha)(\xi_\alpha+2\eta_\alpha) \\ \frac{1}{2}(1-\xi_\alpha^2) \\ \frac{1}{4}(1-\xi_\alpha)(-\xi_\alpha+2\eta_\alpha) \\ -\eta(1-\xi_\alpha) \end{bmatrix} \tag{50}$$

则有单元刚度矩阵

$$\mathbf{K}^{(e)} = \sum_{\alpha=1}^{4} w_\alpha \bar{\mathbf{B}}_\alpha^{\mathrm{T}} \mathbf{D} \hat{\mathbf{B}}_\alpha \det(\mathbf{J}_\alpha) \tag{51}$$

其中

$$\mathbf{J}_\alpha = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} \tag{52}$$

$$\begin{aligned} \mathbf{J}_\alpha(1,1) &= \frac{1}{4}\left[-x_1 + x_3 + x_5 - x_7 + \eta_\alpha\left(x_1 - x_3 + x_5 - x_7\right)\right] \\ \mathbf{J}_\alpha(1,2) &= \frac{1}{4}\left[-y_1 + y_3 + y_5 - y_7 + \eta_\alpha\left(y_1 - y_3 + y_5 - y_7\right)\right] \\ \mathbf{J}_\alpha(2,1) &= \frac{1}{4}\left[-x_1 - x_3 + x_5 + x_7 + \xi_\alpha\left(x_1 - x_3 + x_5 - x_7\right)\right] \\ \mathbf{J}_\alpha(2,2) &= \frac{1}{4}\left[-y_1 - y_3 + y_5 + y_7 + \xi_\alpha\left(y_1 - y_3 + y_5 - y_7\right)\right] \end{aligned} \tag{53}$$

按照自由度对应组合即可得出全局刚度矩阵。

三、有限元求解代码实现

本文中修改版本的 Fortran 代码采用一种新型非对称八节点平面单元，可以实现二维平面应变问题的有限元求解，优点是在二次位移场下不受网格畸变影响。程序框架如图 2 所示，其中需要修改的部分使用红色标注。

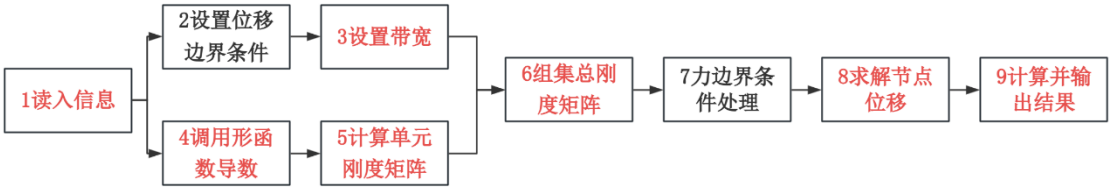


图 2 程序求解框架

本文中程序部分进行的修改如表 1 所示。表 1 中左侧为原有的代码功能，右侧为为适应非对称八节点平面单元进行修改后的代码功能，具体代码修改见附录。

表 1 程序修改内容

原代码功能	修改代码功能
浮点数类型 real*4	浮点数类型 real*8
传统刚度矩阵	新型非对称刚度矩阵
变带宽存储刚度矩阵	全局存储刚度矩阵
单元节点单元顺时针排序	单元节点单元逆时针排序

Cholesky 分解法求解	LU 分解法求解
输出单元中心点应力	额外输出每个单元节点应力
输入文件进行网格划分	直接输入网格与节点信息

各部分程序修改如下所示：

3.1 数据类型修改

在原有程序中，采用 real 格式定义浮点数，但该数据格式精度不够，因此所有浮点数定义修改为 real*8。

3.2 单元刚度矩阵修改

该部分是程序修改的核心内容，单元刚度矩阵按照公式(51)求解。在原本程序中，单元刚度矩阵定义为

$$\mathbf{K}^{(e)}=\sum_{\alpha=1}^{4} w_{\alpha}\bar{\mathbf{B}}_{\alpha}^{\mathrm{T}}\mathbf{D}\bar{\mathbf{B}}_{\alpha}\det\left(\mathbf{J}_{\alpha}\right) \tag{54}$$

相较于公式(51)，式中的第二个 $\bar{\mathbf{B}}_{\alpha}$ 矩阵应该修改为 $\hat{\mathbf{B}}_{\alpha}$。需要在原有基础上增加 $\hat{\mathbf{B}}_{\alpha}$ 的计算模块。主程序中代码设置如下所示：

```
call metric_shape_and_B(coord, points(i,1), points(i,2), Bhat)
call det_compute(coord, points(i,1), points(i,2), det)
```

特别注意的是，在求解 $\hat{\mathbf{B}}_{\alpha}$ 时，需要首先计算 $\mathbf{P}^{-1}$，需要对 8×8 矩阵求逆，调用模块为：

```
call inverse_8x8(P, invP)
```

3.3 全局存储刚度矩阵

由于采用非对称刚度矩阵，原程序中采用的变带宽半存储模式无法使用，由于问题本身规模不是很大，在此直接采用全局刚度矩阵存储，主程序中代码修改为：

```
allocate(kb(neq,neq),loads(0:neq)); kb=.0
```

在组集总刚度矩阵时候，需要采用整体存储方式，则需对 formkb 模块修改。

3.4 单元节点单元逆时针排序

原代码中四边形八节点单元中，局部节点排序为顺时针，但是论文中的推导是基于逆时针排序的节点。针对 geometry_8qx 模块中的 num 与 coord 赋值、shape_der 模块中的 der 赋值进行重新排序。在此采用节点对调的方式直接进行修改。

3.5 LU 分解法求解

在进行位移求解时，由于原刚度矩阵为对称正定的，采用 Cholesky 分解法求解。在本问题中刚度矩阵为非对称矩阵，因此采用LU分解法，主程序中调用LU 分解如下：

```
call lu_solver_full(kb, loads(1:neq), neq)
```

对于 LU 分解直接采用默认主元的简单方式求解。

3.6 结果输出修改

在原代码中，只有对每个单元的中心位置应力进行输出，为了更好的展现结果以及进行结果对比，在此增加了每个单元中所有单元节点的应力输出，以及每个单元中高斯点的应力输出，包含三个方向的应力、应力强度以及 Von-Mises 应力。这一功能直接在主程序中后接实现的。

3.7 输入文件修改

在原本程序中，只能模拟矩形材料，且只能按照均匀的规则网格进行单元划分，因此需要额外建立新的程序进行单元网格划分，并对输入信息进行修改如下：

```
elements_1: do iel = 1 , nels
read (10,*) num(1),num(2),num(3),num(4),num(5),num(6),num(7),num(8)
g_num(:,iel) = num;  call num_to_g(num,nf,g)
g_g(:,iel)=g  ;  if(nband<bandwidth(g))nband=bandwidth(g)
end do elements_1
elements_2: do iel = 1 , nn
read (10,*) g_coord(1,iel),g_coord(2,iel)
end do elements_2
```

四、算例验证

修改程序后，为确保程序的正确性，首先针对规则网格单元进行验证。采用原程序中自带算例：固定薄板的部分受压问题 P53.DAT 作为输入文件（可见 PROGRAMMING THE FINITE ELEMENT METHOD 第五版 P181-184），经过计算得到结果，并与参考书中的结果进行对比，节点位移与节点应力结果与原程序显示一致，证明针对规则网格，本程序同样适用，满足最基本的要求。运行结果见附录。

接下来，我们针对论文中的算例进行测试，需要注意的是，程序可以自定

义选取高斯积分点数量。 在本文中，采用四点高斯积分作为算例验证。

4.1 算例一验证

考虑如图 3 所示的梁弯曲问题，材料属性以及几何参数定义与论文中相同，分别采用四种网格划分方式进行计算。

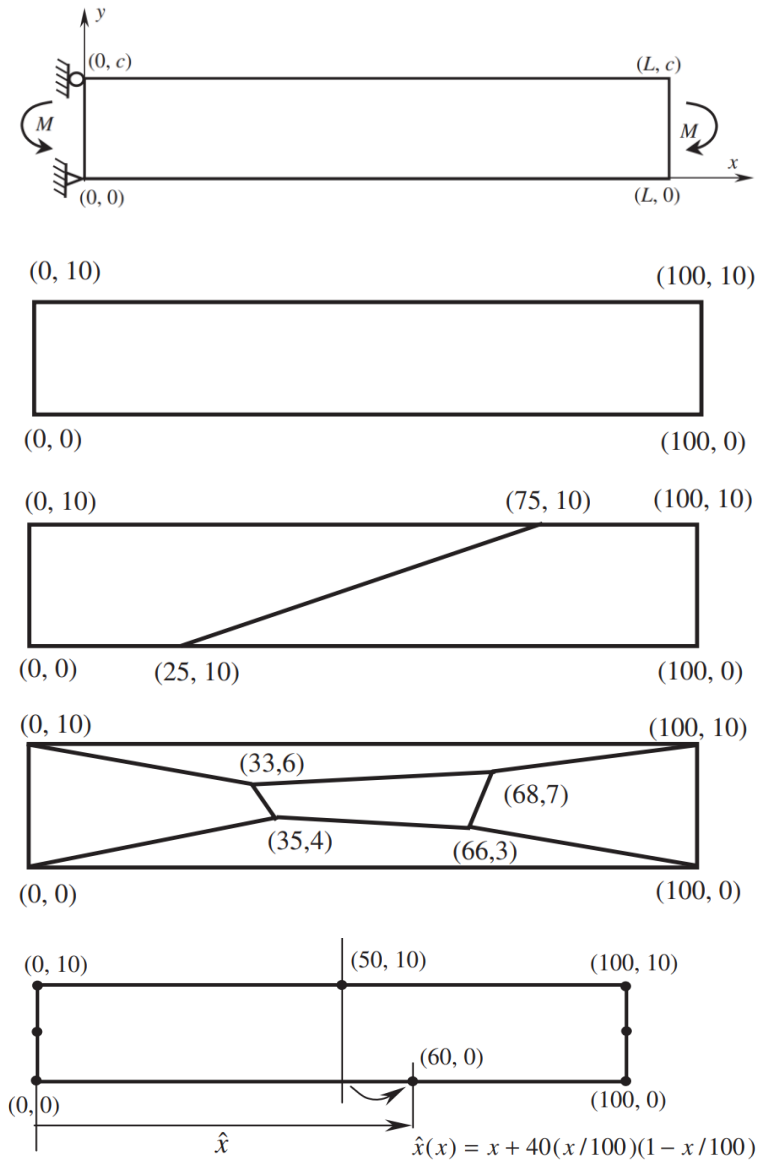


图 3 算例一验证

输入文件与结果文件见附录，在此挑选出关键节点信息进行对比，计算结果与理论结果对比如表 2 所示。

表 2 算例一本程序数值解与精确解对比

网格	变量	本程序数值解	精确解
网格一	$\sigma_x(0,0)$	12620	-120

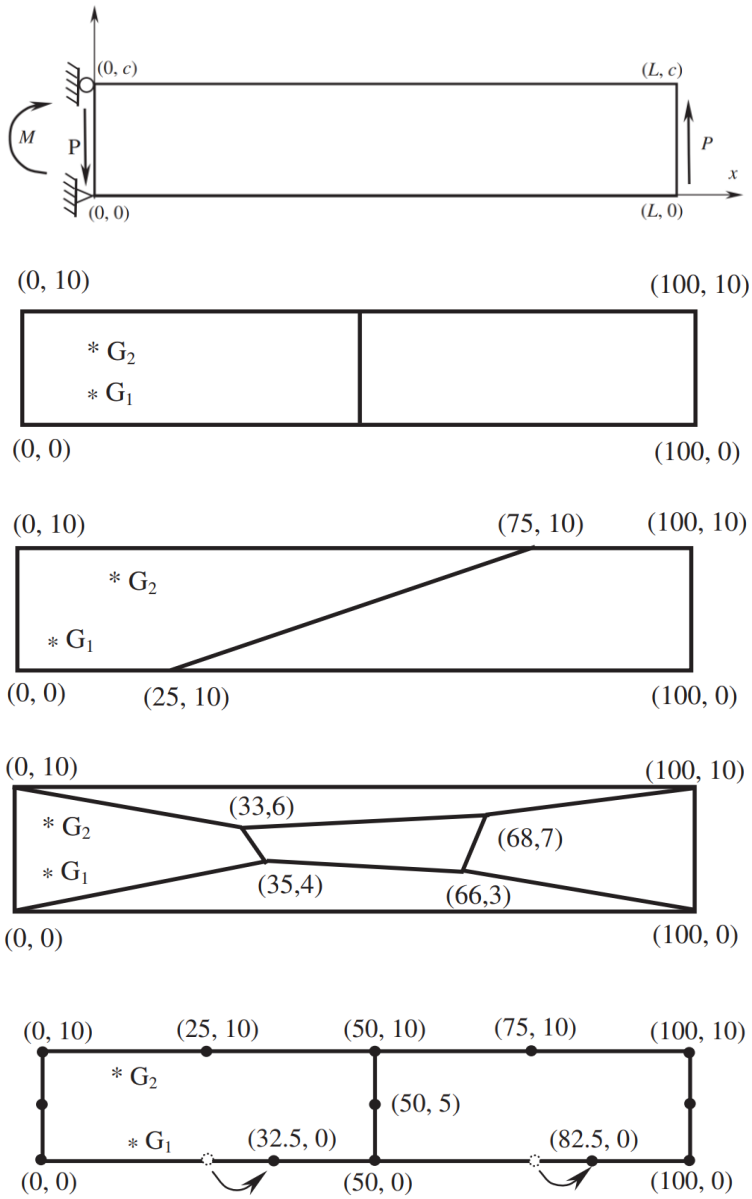
	$\sigma_x(0,10)$	12860	120
	$v(100,0)$	-0.011	-0.012
网格二	$\sigma_x(0,0)$	-120	-120
	$\sigma_x(0,10)$	120	120
	$v(100,0)$	-0.011	-0.012
网格三	$\sigma_x(0+,0)$	-126	-120
	$\sigma_x(0,0+)$	-118	-120
	$\sigma_x(0,10-)$	117	120
	$\sigma_x(0+,10)$	112	120
	$v(100,0)$	-0.011	-0.012
网格四	$\sigma_x(0,0)$	4373	-120
	$\sigma_x(0,10)$	3275	120
	$v(100,0)$	-0.011	-0.012

通过结果对比可以看到，针对网格二与网格三，计算得到的结果与真实解非常接近，因为网格一与网格四只采用了单个网格，本身计算的误差大，而且我们可以发现，采用网格四得到的结果相对误差较小，这也说明本程序针对不规则网格有更好的计算效果。且与传统对称八节点单元相比，针对网格二与网格三可以取得更加准确的计算结果。

此外，经过测试，针对网格一与网格四，当采用九点高斯积分时，可以得到更好的结果。

4.2 算例二验证

考虑如图 4 所示的梁弯曲问题，材料属性以及几何参数定义与论文中相同，分别采用四种网格划分方式进行计算。



$$\hat{x}(x) = x + 40(x/100)(1 - x/100)$$

图 4 算例二验证

计算结果与理论结果相符，结果对比如表 3 所示。与论文中结果吻合。

表 3 算例二本程序数值解与精确解对比

网格	变量	本程序数值解	精确解
网格一	$\sigma_x(G_1)$	54.64	61.9615
	$\sigma_x(G_2)$	-54.64	-61.9615
	$v(100,0)$	0.008265	0.008046
网格二	$\sigma_x(G_1)$	87.76	64.0748

	$\sigma_x(G_2)$	-85.80	-59.8483
	$v(100,0)$	0.006190	0.008046
网格三	$\sigma_x(G_1)$	79.32	53.3626
	$\sigma_x(G_2)$	-50.93	-53.5031
	$v(100,0)$	0.005134	0.008046
网格四	$\sigma_x(G_1)$	81.56	59.2295
	$\sigma_x(G_2)$	-68.46	-61.2295
	$v(100,0)$	0.006536	0.008046

五、小组成员及分工

姓名	学号	工作内容
张靖雅	2401111735	论文阅读、理论推导、统筹规划、算例编写、报告一、二部分撰写
张家瑞	2401111734	程序编写、算例验证、报告三、四、附录部分撰写
王奕	2401213249	算例验证、报告四部分撰写

附录

附录 A 程序代码

计算 $\hat{\mathbf{B}}_\alpha$ 模块

```
subroutine metric_shape_and_B(coord, xi, eta, Bhat)
implicit none
real*8, intent(in)  :: coord(8,2);  real*8, intent(in)  :: xi, eta;  real*8, intent(out) :: Bhat(3,16)
integer :: i, j;  real*8 :: x, y, poly(8), invP(8,8), P(8,8), N(8);  real*8 :: dMdx(8), dMdy(8)
! Step 1: 计算参数形函数值 N(ξ,η)（用于坐标插值）
N(1) = 0.25*(1.0 - xi)*(1.0 - eta)*(-xi - eta - 1.0);  N(2) = 0.5 *(1.0 - eta)*(1.0 - xi*xi)
N(3) = 0.25*(1.0 + xi)*(1.0 - eta)*(xi - eta - 1.0);  N(4) = 0.5 *(1.0 + xi)*(1.0 - eta*eta)
N(5) = 0.25*(1.0 + xi)*(1.0 + eta)*(xi + eta - 1.0);  N(6) = 0.5 *(1.0 + eta)*(1.0 - xi*xi)
N(7) = 0.25*(1.0 - xi)*(1.0 + eta)*(-xi + eta - 1.0);  N(8) = 0.5 *(1.0 - xi)*(1.0 - eta*eta)
! Step 2: 坐标插值
x = 0.0;  y = 0.0
do j = 1,8
  x = x + N(j) * coord(j,1);  y = y + N(j) * coord(j,2)
end do
! Step 3: 构造矩阵 P
do i = 1, 8
  P(1,i) = 1.0;  P(2,i) = coord(i,1);  P(3,i) = coord(i,2);  P(4,i) = coord(i,1)**2
  P(5,i) = coord(i,1)*coord(i,2);  P(6,i) = coord(i,2)**2;  P(7,i) = coord(i,1)**2 * coord(i,2)
  P(8,i) = coord(i,1) * coord(i,2)**2
end do
! Step 4: 求 inv(P)
call inverse_8x8(P, invP)
! Step 5: 求 dM/dx
poly(1) = 0.0;  poly(2) = 1.0;  poly(3) = 0.0;  poly(4) = 2.0 * x;  poly(5) = y;  poly(6) = 0.0
poly(7) = 2.0 * x * y;  poly(8) = y**2
dMdx = matmul(invP, poly)
! Step 6: 求 dM/dy
poly(1) = 0.0;  poly(2) = 0.0;  poly(3) = 1.0;  poly(4) = 0.0;  poly(5) = x;  poly(6) = 2.0 * y
poly(7) = x**2;  poly(8) = 2.0 * x * y
dMdy = matmul(invP, poly)
! Step 8: 构造 Bhat (3x16)
Bhat = 0.0
do i = 1, 8
```

```
  j = 2 * i
  Bhat(1,j-1) = dMdx(i);  Bhat(2,j  ) = dMdy(i);  Bhat(3,j-1) = dMdy(i);  Bhat(3,j  ) = dMdx(i)
end do
end subroutine metric_shape_and_B
```

8×8 矩阵求逆

```
subroutine inverse_8x8(A, Ainv)
implicit none
real*8, intent(in)  :: A(8,8);  real*8, intent(out) :: Ainv(8,8);  real*8 :: aug(8,16)
integer :: i, j, pivot_row;  real*8 :: factor, pivot
! 构造增广矩阵 [A | I]
aug(:,1:8) = A;  aug(:,9:16) = 0.0
do i = 1,8
  aug(i,i+8) = 1.0
end do
! 高斯消元带主元选取
do i = 1,8
  ! 主元选取（最大行交换）
  pivot = abs(aug(i,i));  pivot_row = i
  do j = i+1,8
    if (abs(aug(j,i)) > pivot) then
      pivot = abs(aug(j,i));  pivot_row = j
    end if
  end do
  ! 若主元为 0，则矩阵奇异
  if (pivot < 1.0e-12) then
    print*, "Matrix is singular or nearly singular"
    stop
  end if
  ! 行交换
  if (pivot_row /= i) then
    aug(i,:) = aug(i,:) + aug(pivot_row,:);  aug(pivot_row,:) = aug(i,:) - aug(pivot_row,:)
    aug(i,:) = aug(i,:) - aug(pivot_row,:)
  end if
  ! 归一化主行
  aug(i,:) = aug(i,:) / aug(i,i)
  ! 对其他行进行消元
```

```
  do j = 1,8
    if (j /= i) then
      factor = aug(j,i);  aug(j,:) = aug(j,:) - factor * aug(i,:)
    end if
  end do
end do
! 取出右半部分为逆矩阵
Ainv = aug(:,9:16)
end subroutine inverse_8x8
```

计算雅可比矩阵特征值模块

```
subroutine det_compute(coord, xi, eta, det)
implicit none
real*8, intent(in)  :: coord(8,2)
real*8, intent(in)  :: xi, eta
real*8, intent(out) :: det
real*8 :: Jet(2,2)
! Step 1: 计算参数形函数值 N(ξ,η)（用于坐标插值）
Jet(1,1) = 0.25*(-coord(1,1)+coord(3,1)+coord(5,1)-coord(7,1) &
          +eta*(coord(1,1)-coord(3,1)+coord(5,1)-coord(7,1)))
Jet(1,2) = 0.25*(-coord(1,2)+coord(3,2)+coord(5,2)-coord(7,2) &
          +eta*(coord(1,2)-coord(3,2)+coord(5,2)-coord(7,2)))
Jet(2,1) = 0.25*(-coord(1,1)-coord(3,1)+coord(5,1)+coord(7,1) &
          +xi*(coord(1,1)-coord(3,1)+coord(5,1)-coord(7,1)))
Jet(2,2) = 0.25*(-coord(1,2)-coord(3,2)+coord(5,2)+coord(7,2) &
          +xi*(coord(1,2)-coord(3,2)+coord(5,2)-coord(7,2)))
det = determinant(Jet)
end subroutine det_compute
```

组集全局刚度矩阵 formkb 模块

```
subroutine formkb(kb, km, g)
implicit none
real*8, intent(in)    :: km(:,:)     ！单元局部刚度矩阵
real*8, intent(inout) :: kb(:,:)     ！全局刚度矩阵 (已声明为 kb(neq, neq))
integer, intent(in)   :: g(:)        ！单元自由度编号
integer :: idof, i, j;  idof = size(km,1)
do i = 1, idof
```

```
 if (g(i) > 0) then
   do j = 1, idof
     if (g(j) > 0) then
       kb(g(i), g(j)) = kb(g(i), g(j)) + km(i,j)
     end if
   end do
 end if
end do
end subroutine formkb
```

geometry_8qx 单元节点修改

```
num(1)=(ip-1)*(3*nye+2)+2*iq+1; num(2)=num(1)-1; num(3)=num(1)-2
num(4)=(ip-1)*(3*nye+2)+2*nye+iq+1;num(5)=ip*(3*nye+2)+2*iq-1
num(6)=num(5)+1; num(7)=num(5)+2; num(8)=num(4)+1
coord(1:3,1)=(ip-1)*aa; coord(5:7,1)=ip*aa
coord(4,1)=.5*(coord(3,1)+coord(5,1))
coord(8,1)=.5*(coord(7,1)+coord(1,1))
coord(1,2)=-iq*bb; coord(7:8,2)=-iq*bb; coord(3:5,2)=-(iq-1)*bb
coord(2,2)=.5*(coord(1,2)+coord(3,2))
coord(6,2)=.5*(coord(5,2)+coord(7,2))
```

shape_der 单元节点修改

```
case(8)
der(1,1)=etam*(2.*xi+eta);  der(1,8)=-8.*etam*etap
der(1,7)=etap*(2.*xi-eta);  der(1,6)=-4.*etap*xi
der(1,5)=etap*(2.*xi+eta);  der(1,4)=8.*etap*etam
der(1,3)=etam*(2.*xi-eta);  der(1,2)=-4.*etam*xi
der(2,1)=xim*(xi+2.*eta);  der(2,8)=-4.*xim*eta
der(2,7)=xim*(2.*eta-xi);  der(2,6)=8.*xim*xip
der(2,5)=xip*(xi+2.*eta);  der(2,4)=-4.*xip*eta
der(2,3)=xip*(2.*eta-xi);  der(2,2)=-8.*xim*xip
```

LU 分解求解模块

```
subroutine lu_solver_full(kb, f, n)
implicit none
integer, intent(in) :: n;  real*8 , intent(in out) :: kb(n,n)   ！输入刚度矩阵，输出 LU 因子
real*8 , intent(in out) :: f(n)     ！输入载荷，输出位移解
```

```
integer :: i, j, k;  real*8 :: sum
! LU 分解
do k = 1, n
  if (abs(kb(k,k)) < 1.0d-12) then
    print *, "Zero pivot encountered at row ", k
    stop
  end if
  do i = k+1, n
    kb(i,k) = kb(i,k) / kb(k,k)
    do j = k+1, n
      kb(i,j) = kb(i,j) - kb(i,k) * kb(k,j)
    end do
  end do
end do
! 前代：求解 L * y = f
do i = 2, n
  sum = 0.0d0
  do j = 1, i-1
    sum = sum + kb(i,j) * f(j)
  end do
  f(i) = f(i) - sum
end do
! 后代：求解 U * x = y
f(n) = f(n) / kb(n,n)
do i = n-1, 1, -1
  sum = 0.0d0
  do j = i+1, n
    sum = sum + kb(i,j) * f(j)
  end do
  f(i) = (f(i) - sum) / kb(i,i)
end do
end subroutine lu_solver_full
```

结果输出模块

```
write(11,'(a)') "Stresses at Gauss Points in Each Element:"
write(11,'(a)') "Format: Element No., Gauss Point No., Sigma_x, Sigma_y, Tau_xy"
elements_4: do iel = 1, nels
write(11,'(a,i5)') "Element No. ", iel
```

```
num = g_num(:, iel)
g   = g_g(:, iel)
coord = transpose(g_coord(:, num))
eld = loads(g)
do i = 1, nip
  ! 参数空间导数
  call shape_der(der, points, i)
  jac   = matmul(der, coord)
  call invert(jac)
  deriv = matmul(jac, der)
  ! B 矩阵与应力
  call beemat(bee, deriv)
  sigma = matmul(dee, matmul(bee, eld))
  ! 输出当前积分点的应力
  write(11,'(a,i2,a,3e12.4)') "  Gauss Point ", i, ": ", sigma(1), sigma(2), sigma(3)
end do
end do elements_4
! 输出每个单元的每个节点上的应力
write(11,'(a)') "Element nodal stresses :"
write(11,'(a)') "Sigma_x Sigma_y Tau_xy Total_sigma Von_Mises"
elements_5: do iel = 1 , nels
write(11,'(a,i5)') "Element No. ", iel
num = g_num(:,iel);  g   = g_g(:,iel);  coord = transpose(g_coord(:,num))    ! 8x2 坐标
eld = loads(g) ！当前单元自由度向量
do k = 1, 8
  ! 节点的自然坐标：8 节点次序约定
  select case (k)
    case(1); xi0=-1.0; eta0=-1.0
    case(2); xi0= 0.0; eta0=-1.0
    case(3); xi0= 1.0; eta0=-1.0
    case(4); xi0= 1.0; eta0= 0.0
    case(5); xi0= 1.0; eta0= 1.0
    case(6); xi0= 0.0; eta0= 1.0
    case(7); xi0=-1.0; eta0= 1.0
    case(8); xi0=-1.0; eta0= 0.0
  end select
  ! 参数形函数导数（虚位移）
```

```
call shape_der(der, reshape([xi0,eta0],[1,2]), 1);  jac   = matmul(der, coord);  call invert(jac)
deriv = matmul(jac, der);  call beemat(bee, deriv);  sigma = matmul(dee, matmul(bee, eld))
! 计算总应力强度
sig_total = sqrt(sigma(1)**2 + sigma(2)**2 + sigma(3)**2)
 Von_Mises = sqrt(sigma(1)**2 + sigma(2)**2 + 3*sigma(3)**2 - sigma(1)*sigma(2))
 ! 输出：局部节点号 + 应力张量分量 + 总应力
 write(11,'(a,i2,a,i5,a,3e12.4,e12.4,e12.4)') "  Node ", k, &
                                  " (global ", num(k), "): ", &
  sigma(1), sigma(2), sigma(3), sig_total, Von_Mises
end do
end do elements_5
```

附录 B 算例验证结果

The nodal displacements are :

Node	Displacement	
1	0.0000E+00	-0.5311E-05
2	-0.4211E-06	-0.5041E-05
3	-0.7222E-06	-0.3343E-05
4	-0.4211E-06	-0.1644E-05
5	0.0000E+00	-0.1375E-05
6	0.0000E+00	-0.4288E-05
7	0.3774E-06	-0.2786E-05
8	0.0000E+00	-0.1283E-05
9	0.0000E+00	-0.3243E-05
10	0.2708E-06	-0.2873E-05
11	0.3670E-06	-0.2229E-05
12	0.2708E-06	-0.1584E-05
13	0.0000E+00	-0.1214E-05
14	0.0000E+00	-0.2217E-05
15	0.2996E-06	-0.1671E-05
16	0.0000E+00	-0.1125E-05
17	0.0000E+00	-0.1379E-05
18	0.1370E-06	-0.1299E-05
19	0.1912E-06	-0.1114E-05
20	0.1370E-06	-0.9299E-06
21	0.0000E+00	-0.8498E-06
22	0.0000E+00	-0.6454E-06
23	0.1122E-06	-0.5571E-06

24 0.0000E+00 -0.4689E-06

25 0.0000E+00 0.0000E+00

26 0.0000E+00 0.0000E+00

27 0.0000E+00 0.0000E+00

28 0.0000E+00 0.0000E+00

29 0.0000E+00 0.0000E+00

Stresses at Gauss Points in Each Element:

Format: Element No., Gauss Point No., Sigma_x, Sigma_y, Tau_xy

Element No. 1

Gauss Point 1: -0.5135E+00 -0.1037E+01 -0.2481E-01

Gauss Point 2: -0.3507E+00 -0.7711E+00 0.1505E+00

Gauss Point 3: -0.1513E+00 -0.8971E+00 0.7274E-01

Gauss Point 4: -0.1589E+00 -0.6931E+00 0.2405E+00

Element No. 2

Gauss Point 1: -0.7783E-01 -0.2289E+00 0.1505E+00

Gauss Point 2: 0.8491E-01 0.3683E-01 -0.2481E-01

Gauss Point 3: -0.2697E+00 -0.3069E+00 0.2405E+00

Gauss Point 4: -0.2773E+00 -0.1029E+00 0.7274E-01

Element No. 3

Gauss Point 1: -0.1372E+00 -0.7668E+00 0.7382E-01

Gauss Point 2: -0.1907E+00 -0.5855E+00 0.1540E+00

Gauss Point 3: -0.1446E+00 -0.6452E+00 0.3542E-01

Gauss Point 4: -0.1913E+00 -0.5523E+00 0.9349E-01

Element No. 4

Gauss Point 1: -0.2378E+00 -0.4145E+00 0.1540E+00

Gauss Point 2: -0.2914E+00 -0.2332E+00 0.7382E-01

Gauss Point 3: -0.2373E+00 -0.4477E+00 0.9349E-01

Gauss Point 4: -0.2840E+00 -0.3548E+00 0.3542E-01

Element No. 5

Gauss Point 1: -0.1700E+00 -0.5996E+00 0.1848E-01

Gauss Point 2: -0.1971E+00 -0.5337E+00 0.5417E-01

Gauss Point 3: -0.2135E+00 -0.5594E+00 0.9897E-02

Gauss Point 4: -0.2136E+00 -0.5250E+00 0.3342E-01

Element No. 6

Gauss Point 1: -0.2315E+00 -0.4663E+00 0.5417E-01

Gauss Point 2: -0.2586E+00 -0.4004E+00 0.1848E-01

Gauss Point 3: -0.2150E+00 -0.4750E+00 0.3342E-01

Gauss Point 4: -0.2151E+00 -0.4406E+00 0.9897E-02

Element nodal stresses :

Sigma_x Sigma_y Tau_xy Total_sigma Von_Mises

Element No. 1

Node 1 (global 9): -0.8512E-01 -0.8108E+00 0.5965E-01 0.8174E+00 0.7787E+00

Node 2 (global 10): -0.2567E+00 -0.9127E+00 0.1837E+00 0.9657E+00 0.8751E+00

Node 3 (global 11): -0.2063E+00 -0.4966E+00 0.3455E+00 0.6392E+00 0.7381E+00

Node 4 (global 7): -0.6942E-01 -0.4379E+00 0.2714E+00 0.5199E+00 0.6223E+00

Node 5 (global 3): -0.4306E+00 -0.5927E+00 0.1944E+00 0.7580E+00 0.6284E+00

Node 6 (global 2): -0.7365E+00 -0.1101E+01 0.2126E-01 0.1325E+01 0.9724E+00

Node 7 (global 1): -0.8205E+00 -0.1092E+01 -0.1141E+00 0.1371E+01 0.1004E+01

Node 8 (global 6): -0.2038E+00 -0.8447E+00 -0.2574E-01 0.8693E+00 0.7648E+00

Element No. 2

Node 1 (global 11): -0.2222E+00 -0.5034E+00 0.3455E+00 0.6497E+00 0.7409E+00

Node 2 (global 12): -0.1719E+00 -0.8733E-01 0.1837E+00 0.2663E+00 0.3512E+00

Node 3 (global 13): -0.3435E+00 -0.1892E+00 0.5965E-01 0.3966E+00 0.3154E+00

Node 4 (global 8): -0.2248E+00 -0.1553E+00 -0.2574E-01 0.2745E+00 0.2043E+00

Node 5 (global 5): 0.3919E+00 0.9200E-01 -0.1141E+00 0.4184E+00 0.4063E+00

Node 6 (global 4): 0.3080E+00 0.1013E+00 0.2126E-01 0.3249E+00 0.2743E+00

Node 7 (global 3): 0.2043E-02 -0.4073E+00 0.1944E+00 0.4513E+00 0.5293E+00

Node 8 (global 7): -0.3591E+00 -0.5621E+00 0.2714E+00 0.7201E+00 0.6812E+00

Element No. 3

Node 1 (global 17): -0.1264E+00 -0.6000E+00 0.7151E-02 0.6132E+00 0.5480E+00

Node 2 (global 18): -0.1810E+00 -0.5857E+00 0.5632E-01 0.6156E+00 0.5285E+00

Node 3 (global 19): -0.2028E+00 -0.4951E+00 0.9372E-01 0.5432E+00 0.4606E+00

Node 4 (global 15): -0.1954E+00 -0.4919E+00 0.1411E+00 0.5478E+00 0.4937E+00

Node 5 (global 11): -0.2063E+00 -0.4966E+00 0.2125E+00 0.5782E+00 0.5676E+00

Node 6 (global 10): -0.1741E+00 -0.7199E+00 0.1420E+00 0.7541E+00 0.6955E+00

Node 7 (global 9): -0.1091E+00 -0.8668E+00 0.5965E-01 0.8756E+00 0.8242E+00

Node 8 (global 14): -0.1085E+00 -0.7294E+00 0.2140E-01 0.7378E+00 0.6827E+00

Element No. 4

Node 1 (global 19): -0.2257E+00 -0.5049E+00 0.9372E-01 0.5610E+00 0.4672E+00

Node 2 (global 20): -0.2476E+00 -0.4143E+00 0.5632E-01 0.4859E+00 0.3740E+00

Node 3 (global 21): -0.3022E+00 -0.4000E+00 0.7151E-02 0.5014E+00 0.3614E+00

Node 4 (global 16): -0.3200E+00 -0.2706E+00 0.2140E-01 0.4196E+00 0.3007E+00

Node 5 (global 13): -0.3195E+00 -0.1332E+00 0.5965E-01 0.3512E+00 0.2965E+00

Node 6 (global 12): -0.2545E+00 -0.2801E+00 0.1420E+00 0.4042E+00 0.3639E+00

Node 7 (global 11): -0.2222E+00 -0.5034E+00 0.2125E+00 0.5899E+00 0.5713E+00

Node 8 (global 15): -0.2332E+00 -0.5081E+00 0.1411E+00 0.5766E+00 0.5038E+00

Element No. 5

Node 1 (global | 25):	-0.2313E+00	-0.5396E+00	0.0000E+00	0.5871E+00	0.4689E+00
Node 2 (global | 26):	-0.2328E+00	-0.5432E+00	0.2182E-01	0.5914E+00	0.4736E+00
Node 3 (global | 27):	-0.2143E+00	-0.5000E+00	0.3302E-01	0.5450E+00	0.4382E+00
Node 4 (global | 23):	-0.2011E+00	-0.4944E+00	0.4920E-01	0.5360E+00	0.4390E+00
Node 5 (global | 19):	-0.2028E+00	-0.4951E+00	0.7668E-01	0.5405E+00	0.4511E+00
Node 6 (global | 18):	-0.1809E+00	-0.5855E+00	0.4723E-01	0.6146E+00	0.5257E+00
Node 7 (global | 17):	-0.1388E+00	-0.6291E+00	0.7151E-02	0.6442E+00	0.5726E+00
Node 8 (global | 22):	-0.1776E+00	-0.5812E+00	-0.2070E-02	0.6077E+00	0.5158E+00

Element No. 6

Node 1 (global | 27):	-0.2143E+00	-0.5000E+00	0.3302E-01	0.5450E+00	0.4382E+00
Node 2 (global | 28):	-0.1958E+00	-0.4568E+00	0.2182E-01	0.4974E+00	0.3987E+00
Node 3 (global | 29):	-0.1973E+00	-0.4604E+00	0.0000E+00	0.5009E+00	0.4000E+00
Node 4 (global | 24):	-0.2510E+00	-0.4188E+00	-0.2070E-02	0.4883E+00	0.3651E+00
Node 5 (global | 21):	-0.2897E+00	-0.3709E+00	0.7151E-02	0.4707E+00	0.3380E+00
Node 6 (global | 20):	-0.2477E+00	-0.4145E+00	0.4723E-01	0.4852E+00	0.3704E+00
Node 7 (global | 19):	-0.2257E+00	-0.5049E+00	0.7668E-01	0.5584E+00	0.4578E+00
Node 8 (global | 23):	-0.2274E+00	-0.5056E+00	0.4920E-01	0.5566E+00	0.4468E+00