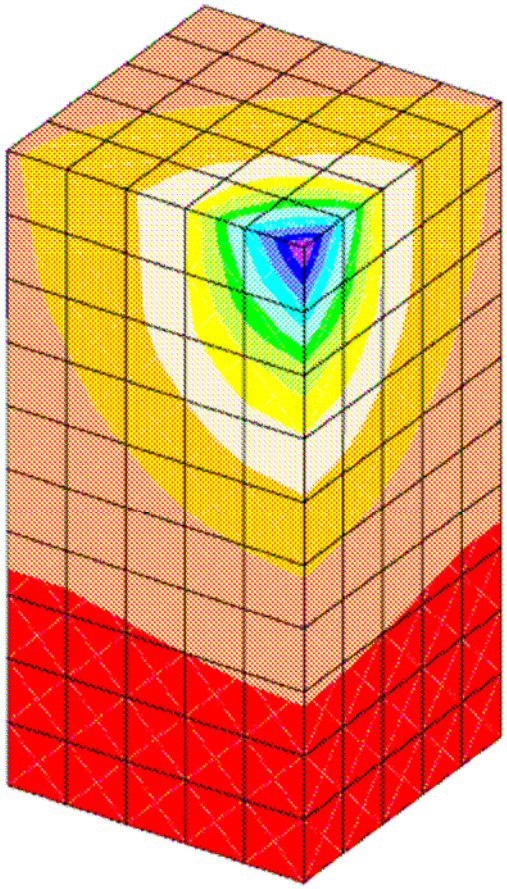
ASEN 5367

Comparison of Tetrahedral and Brick Elements for Linear Elastic Analysis

Term Project AFEM Spring 2006



Date: 05/09/2006

Submitted to: Professor Carlos A. Felippa

By
Jochen Carl
Dana Müller-Hoeppe
Morgan Meadows

ABSTRACT

In this term paper, the performances of elastic solid continuum finite elements are compared modeling both hexagonal and tetrahedral meshes. Elements with linear and quadratic shape functions are used to model three different tests. The first test is a clamped cantilever beam with a T cross-section, which is deformed due to an applied displacement at the free end. In the second test, a distorted element mesh is applied to a beam of square cross-section. In the last test, a load distributed over a small area is applied to an incompressible block.
For the tests, the overkill solutions will serve as reference values, which means taking the finest mesh possible and using the best element for the calculation. The tests results are compared in convergence studies. Finally, the results of the performance tests are discussed and explained theoretically.

TABLE OF CONTENTS

ABSTRACT....2
TABLE OF CONTENTS....3
TABLE OF EQUATIONS....3
TABLE OF FIGURES....3
INTRODUCTION....5
BASIC PRINCIPLES....5
Basic Equations and General Computation Steps....5
Elements....8
Linear Tetrahedron....8
Quadratic (10 node) Tetrahedron....9
The Tri-linear (8 node) Hexahedron....10
The Tri-quadratic (27 node) Hexahedron....11
Patch test....12
PERFORMANCE TESTS....15
T-Beam Test....16
Distortion test....20
Block test....23
CONCLUSION....26
REFERENCES....27
APPENDIX....27
Element Files....27
Linear brick....27
Quadratic brick....35
Linear tetrahedron....43
Patch test....58
Input Files for Performance Tests....69
Beam....69
Distortion....71
Block....72

TABLE OF EQUATIONS

Equation 1: TPE functional....5
Equation 2: Element stiffness matrix....6

TABLE OF FIGURES

Figure 1: Linear tetrahedron....8
Figure 2: Quadratic tetrahedron....9
Figure 3: Linear hexahedron....10
Figure 4: Quadratic hexahedron....11

Figure 5: Applied force and boundary conditions for the FPT with tension in the z-direction 13
Figure 6: Displacement in the z-direction for the FPT with tension in the z-direction..... 14
Figure 7: Stress σ_{zz} for the FPT with tension in the z-direction..... 14
Figure 8 : Applied shear and boundary conditions for the FPT with shear on the x- and y-planes 15
Figure 9: Stress σ_{xy} for the FPT with shear on the x- and y-planes..... 15
Figure 10: T-beam under applied end deformation..... 16
Figure 11: Meshes for the T-Beam using bricks and tetrahedrons 16
Figure 12: Boundary conditions for the T-Beam..... 17
Figure 13: Stress σ_{xx} for the quadratic brick..... 17
Figure 14: Stress σ_{xx} for the quadratic tetrahedron 18
Figure 15: Relative error of the sum of the reaction forces for all elements for the beam test..... 19
Figure 16: Bending behavior of a quadratic and a linear brick 20
Figure 17: Geometry of the beam used for the distortion test 20
Figure 18: Meshes for the distortion test using bricks and tetrahedrons..... 21
Figure 19: Boundary conditions for the distortion test..... 21
Figure 20: Relative error of the sum of the reaction forces for all elements for the distortion test..... 22
Figure 21: Mesh and applied load for the block test..... 23
Figure 22: Boundary conditions for block test 24
Figure 23: Dispalcement in z-direction for the linear brick 24
Figure 24: Displacement in z-direction for the quadratic brick..... 25
Figure 25: Incompressibility locking 25

Table 1: Gauss point coordinates and weight for quadratic tetrahedron..... 10
Table 2: Gauss point coordinates and weights for the linear hexahedron..... 11
Table 3: Gauss point coordinates and weights for the quadratic hexahedron 12
Table 4: Sum of reaction forces for all elements and different mesh resolutions..... 18
Table 5: Sum of reaction forces for all elements and different mesh resolutions..... 22

INTRODUCTION

Previous studies have shown, that the performance of finite elements greatly depends on their basic shape, element distortion, the polynomial order, completeness of polynomial functions, integration techniques, material incompressibility and loading.

First, the basic equations and the steps of thee computation are discussed generally. Then, the elements that are used for the performance tests are introduced, namely the linear and quadratic tetrahedron (T1 and T2, respectively), and the linear and quadratic brick (Q1 and Q2, respectively), are introduced. The functionality of the elements is tested with the help of the patch test, which is shortly discussed, and the results are presented for some cases for the linear brick. Then, the first test, the bending of a beam with a T cross-section, is presented. The results of the convergence study comparing the relative error of the sum of the nodal forces in the plane where the displacement is applied are discussed, as well as reasons for the different performance of the elements. This is repeated for the distorted beam with a square cross section. Finally, a nearly incompressible block is considered. However, as the meshes required to perform a convergence study get so large that the necessary computer to solve the systems was unavailable, there are only results for the linear and the quadratic brick, showing the displacement due to loading on a small area of the block surface.

As a conclusion, elements that remove some of the problems of the elements used here are suggested.

BASIC PRINCIPLES

Basic Equations and General Computation Steps

The Total Potential Energy (TPE) functional is given by

$$\Pi_{\mathrm{TPE}}[u_i] = \tfrac{1}{2}\int_V \sigma_{ij}^u \, e_{ij}^u \, dV - \int_V b_i \, u_i \, dV - \int_{S_t} \hat{t}_i \, u_i \, dS.$$

Equation 1: TPE functional

Here, no body forces $\mathbf{b_i}$ are used, and the surface tractions $\hat{t}_i$ are considered by the lumped nodal forces, where the lumping is done by FEAP. The goal is to calculate the element stiffness matrix, as the assembling of the global stiffness matrix as well as taking account of the boundary conditions is done by FEAP.
The stiffness matrix is given by

$$\mathbf{K}^{(e)} = \sum_{i=1}^{p_1} \sum_{j=1}^{p_2} \sum_{k=1}^{p_3} w_i w_j w_k \mathbf{B}_{ijk}^T \mathbf{E} \mathbf{B}_{ijk} J_{ik\ell}$$

Equation 2: Element stiffness matrix

Here, **E** is the material matrix, given by

$$E = \frac{E}{(1-2\nu)(1+\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1 & \nu & 0 & 0 & 0 \\ \nu & \nu & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix}$$

The **B** matrix is used to express the derivatives of the displacements, $\boldsymbol{\varepsilon} = \mathbf{B}\mathbf{u}_\mathbf{i}$, with

$$B = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \end{bmatrix}$$

However, as can be seen in the next section, the shape functions that are used to interpolate the geometry,

$$1 = \sum_{i=1}^{n} N_i, \qquad x = \sum_{i=1}^{n} x_i N_i, \qquad y = \sum_{i=1}^{n} y_i N_i, \qquad z = \sum_{i=1}^{n} z_i N_i,$$

as well as the displacements,

$$u_x = \sum_{i=1}^{n} u_{xi} N_i, \qquad u_y = \sum_{i=1}^{n} u_{yi} N_i, \qquad u_z = \sum_{i=1}^{n} u_{zi} N_i$$

using the isoparametric concept, are given in local coordinates[1].
The steps to achieve the global stiffness matrix are the following:

- Take the derivative of the shape functions w.r.t. the local coordinates for a given Gauss integration point with given coordinates and weight w_i, where for the bricks the 1d shape functions and weights can be used and multiplied like in (Equation 2: Element stiffness matrix), but for the tetrahedrons the total shape functions and weight for each Gauss integration point is given.
- Calculate the derivatives of the shape functions w.r.t. the global coordinates using the Jacobian matrix **J** with the following procedure:

$$\begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{bmatrix} = \boldsymbol{J} \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \\ \frac{\partial}{\partial \mu} \end{bmatrix}, \qquad \boldsymbol{J} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} & \frac{\partial \mu}{\partial x} \\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} & \frac{\partial \mu}{\partial y} \\ \frac{\partial \xi}{\partial z} & \frac{\partial \eta}{\partial z} & \frac{\partial \mu}{\partial z} \end{bmatrix}$$

  where **J** is obtained by inverting

$$\boldsymbol{J}^{-1} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \mu} & \frac{\partial y}{\partial \mu} & \frac{\partial z}{\partial \mu} \end{bmatrix},$$

  which can be easily computed.
- Use the derivatives of the shape functions w.r.t. the global coordinates in combinations with the nodal displacements, $\mathbf{u}_i$, to obtain the **B** matrix.
- Calculate the stiffness matrix.
- Repeat this procedure for each Gauss integration point, adding the contribution of each Gauss integration point to the element stiffness matrix.

According to the element, the number of Gauss integration points p_i (in each direction for the brick and the total number of Gauss integration points for the tetrahedrons) is chosen, as shown in the next section.
After the displacements are calculated using the global stiffness matrix, the stresses at the Gauss integration points are recovered after calculation the strains with the help of the relation $\boldsymbol{\varepsilon} = \mathbf{B}\mathbf{u}_i$. The Gauss integration points are also the superconvergent points for the bricks, and although the tetrahedrons do not have superconvergent points, the Gauss points still provide the most accurate values for the stress. Therefore, the stresses at the Gauss integration points are projected to the element nodes[2].

[1] Felippa, C.A.: AFEM Notes. 2006, Boulder.

[2] Zienkiewicz, O.C. and Taylor, R.L.: The Finite Element Method, Volume 1: The Basis. Butterworth-Heinemann, 2000, Oxford.

Elements

The hexagonal and tetrahedral elements used for the tests are briefly introduced in the following. For each element, the geometry, the chosen coordinate system, the shape functions and the Gauss integration points are shown[3][4].

Linear Tetrahedron

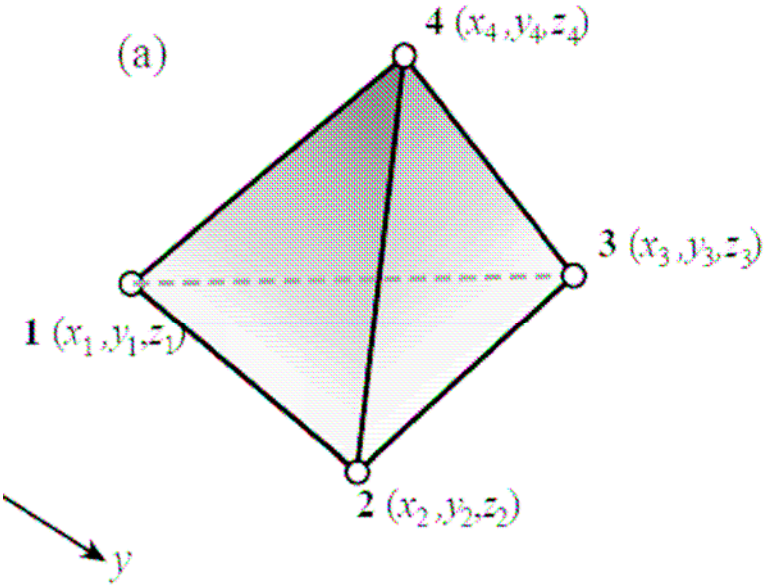


Figure 1: Linear tetrahedron

For the four-node tetrahedron shown in (Figure 1: Linear tetrahedron) the shape functions are tri-linear, the origin of the cartesian coordinate system is placed at one node, like shown for the quadratic tetrahedron.
The shape functions are:

$N_1 = 1 - \xi - \eta - \zeta$
$N_2 = \xi$

[3] Felippa, C.A.: AFEM Notes. 2006, Boulder.

[4] Wriggers, P.: Nichtlineare Finite-Elemente Methoden. Springer, 2001, Heidelberg.

$N_3 = \eta$
$N_4 = \zeta,$

This element requires only one Gauss integration point situated at

$$(x, y, z) = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}),$$

with a weight of

w = 1/6.

Quadratic (10 node) Tetrahedron

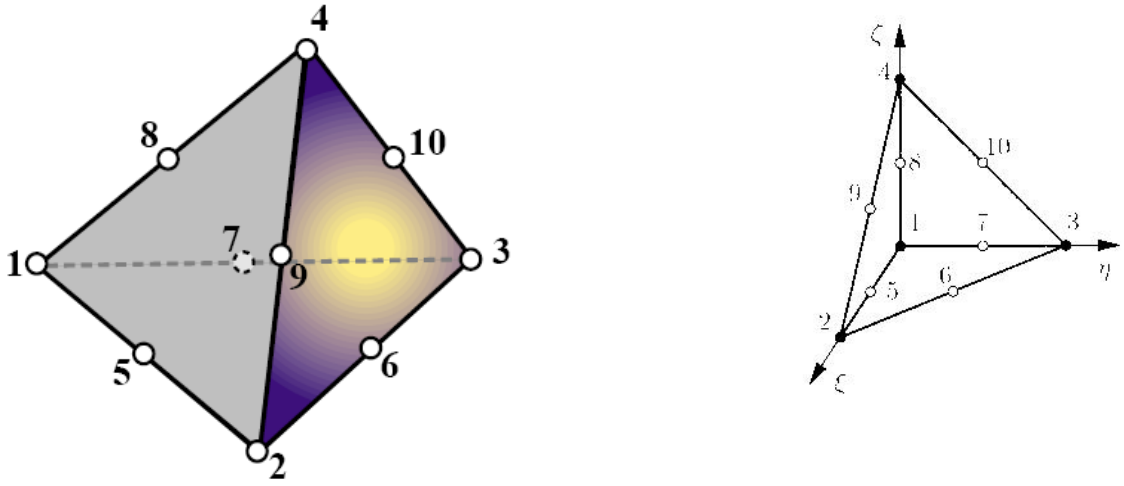


Figure 2: Quadratic tetrahedron

The shape functions for the quadratic tetrahedron are tri-quadratic, and the origin of the Cartesian coordinate system is again placed at one node, like shown in (Figure 2: Quadratic tetrahedron).

The ten shape functions are given by

$N_1 = \lambda(2\lambda - 1)$
$N_2 = \xi(2\xi - 1)$
$N_3 = \eta(2\eta - 1)$
$N_4 = \zeta(2\zeta - 1)$
$N_5 = 4\xi\lambda$
$N_6 = 4\xi\eta$
$N_7 = 4\eta\lambda$
$N_8 = 4\zeta\lambda$

$N_9 = 4\xi\zeta$
$N_{10} = 4\eta\zeta,$

with
$\lambda = 1 - \xi - \eta - \zeta.$

Here, five Gauss points are necessary for the numerical integration, the coordinates and weights of which are given in (Table 1: Gauss point coordinates and weight for quadratic tetrahedron).

Gauss point coordinates $(x, y, z)_i$	Weights w_i
$(\frac{1}{4}, \frac{1}{4}, \frac{1}{4})$	-2/15
$(\frac{1}{6}, \frac{1}{6}, \frac{1}{6})$	3/40
$(\frac{1}{6}, \frac{1}{6}, \frac{1}{2})$	3/40
$(\frac{1}{6}, \frac{1}{2}, \frac{1}{6})$	3/40
$(\frac{1}{2}, \frac{1}{6}, \frac{1}{6})$	3/40

Table 1: Gauss point coordinates and weight for quadratic tetrahedron

The Tri-linear (8 node) Hexahedron

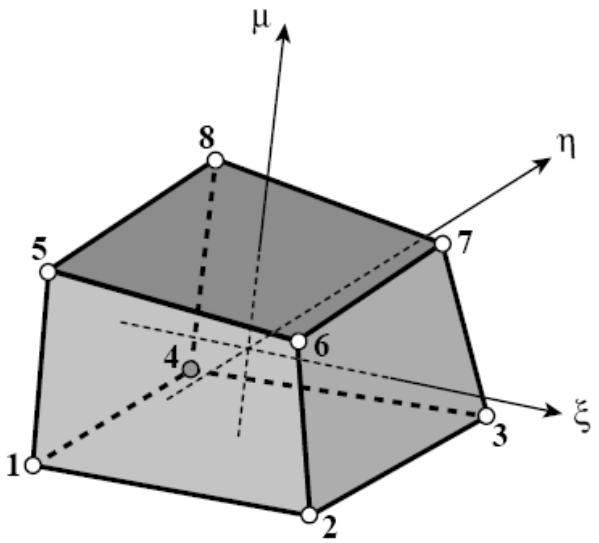


Figure 3: Linear hexahedron

For the brick element the Cartesian coordinate system is placed in the center of the brick, like shown in (Figure 3: Linear hexahedron). Here, the eight shape functions can be

expressed in a more convenient way than for the tetrahedrons. For each shape function N_i the coordinates of the i'th Gauss integration point are inserted for (ξ_i, η_i, μ_i) into

$N_i = (1+ \xi\xi_i)(1+ \eta\eta_i)(1+ \mu\mu_i)/8;$

Two Gauss integration points are necessary in each direction. They are given in (Table 2: Gauss point coordinates and weights for the linear hexahedron) for the ξ-directions and are the same in the other directions.

Gauss point coordinates ξ_i	Weights w_i
$-\frac{1}{\sqrt{3}}$	1
$\frac{1}{\sqrt{3}}$	1

Table 2: Gauss point coordinates and weights for the linear hexahedron

The Tri-quadratic (27 node) Hexahedron

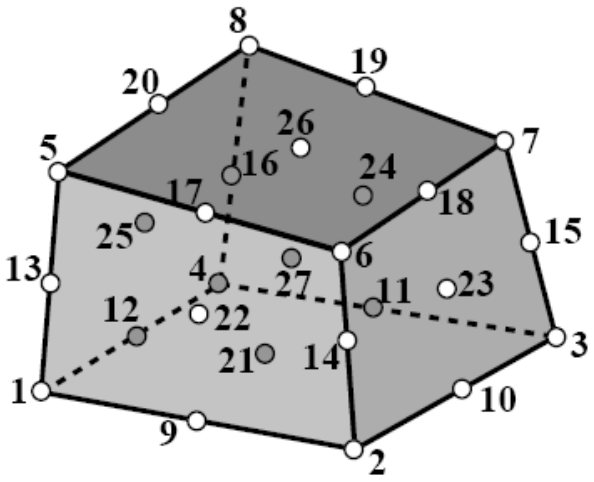


Figure 4: Quadratic hexahedron

The quadratic hexahedron is shown in (Figure 4: Quadratic hexahedron). The quadratic shape functions in 1d are

$N_{1\eta}=0.5\eta(\eta-1)$
$N_{2\eta}=0.5\eta(\eta+1)$
$N_{3\eta}=1-\eta^2$

They can be written for each directions for the 3d case and then the 27 possible combinations, taking one of the shape functions for each direction are used to obtain the tri-quadratic shape functions for all 27 Gauss integration points.
For the numeric integration, three Gauss integrations points are used in each direction, giving a total number of $3^3 = 27$ Gauss points. The 1d coordinates and weights are given in (Table 3: Gauss point coordinates and weights for the quadratic hexahedron), where the 3d weight factors are obtained by multiplying the corresponding weights for all three directions.

Gauss point coordinates η_i	Weight w_i
$-\sqrt{\frac{3}{5}}$	$\frac{5}{9}$
0	$\frac{8}{9}$
$\sqrt{\frac{3}{5}}$	$\frac{5}{9}$

Table 3: Gauss point coordinates and weights for the quadratic hexahedron

The code for all four elements is given in the appendix.

Patch test

With displacement and force patch tests the functionality of the four chosen elements, i.e. the continuity, completeness and stability of the used elements is tested[5]. The inputfile for the patch test for the linear brick is given in the appendix. It can be easily changed to include the other elements instead. Also, only some results for the linear brick are shown here, as the results are the same for all elements if they work.
The configuration for the patch tests has to be in a way that there is a least one node that is not on a boundary, which in 3d requires at least eight elements.
The first set of patch tests are the displacement patch test (DPT). First, constant displacements and rotations are applied to test the solid-body translation and rotation. This leads to six different cases for a 3d element. First, the translations are tested by applying a displacement of 1 in each direction, resulting in the first three cases. The displacement is applied to all nodes except the inner one. The resulting displacement for this node has to be the same like for the other nodes. Then, the solid body rotations are testes by applying a displacement of the negative y-coordinate of the node in the x-direction and a displacement of the positive x-coordinate of the node in the y-direction to the nodes on the boundary and checking if the resulting displacement of the inner node corresponds to this scheme. The same concept is used to test all three solid-body

[5] Felippa, C.A.: AFEM Notes. 2006, Boulder.

rotations. Then, the six cases of constant strain are tested. First, a displacement of the corresponding x-coordinate of the node is applied in the x-direction on each node except the inner one, and the displacement of the inner node is then checked. The same is done in the y- and z-direction. Then, a displacement of the corresponding y-coordinate is applied in the x-direction and a displacement of the corresponding x-coordinate in the y-direction, respectively for the other two possible combinations, to test the shear response.
As the results of the displacement patch tests are not too interesting, they are not listed here. The displacement patch tests are fulfilled for all elements.
The second set of patch tests is the force patch test (FPT). First, a constant tensile surface traction is applied to a surface in each direction in turn. In (Figure 5: Applied force and boundary conditions for the FPT with tension in the z-direction) the setup for the force patch tests where a surface traction of 10 is applied in the z-direction and the corresponding displacement boundary conditions, indicated by the short red lines, is shown. In these cases, the displacement of the test configuration is supposed to be linear, and the stresses are supposed to be constant and equal to the applied surface traction. As can be seen from (Figure 6: Displacement in the z-direction for the FPT with tension in the z-direction), showing the displacement in the z-direction, and figure (Figure 7: Stress σ_{zz} for the FPT with tension in the z-direction), showing the stress σ_{zz}, the patch test is fulfilled. The same results are obtained for the other directions.

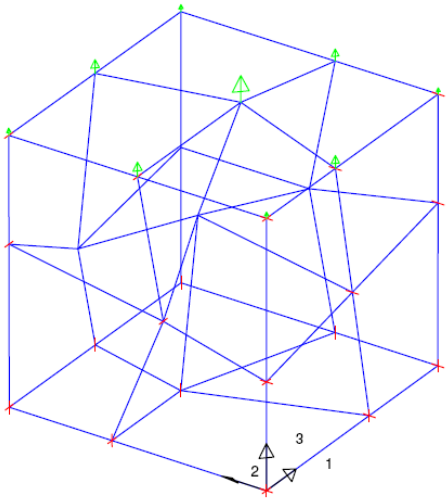


Figure 5: Applied force and boundary conditions for the FPT with tension in the z-direction

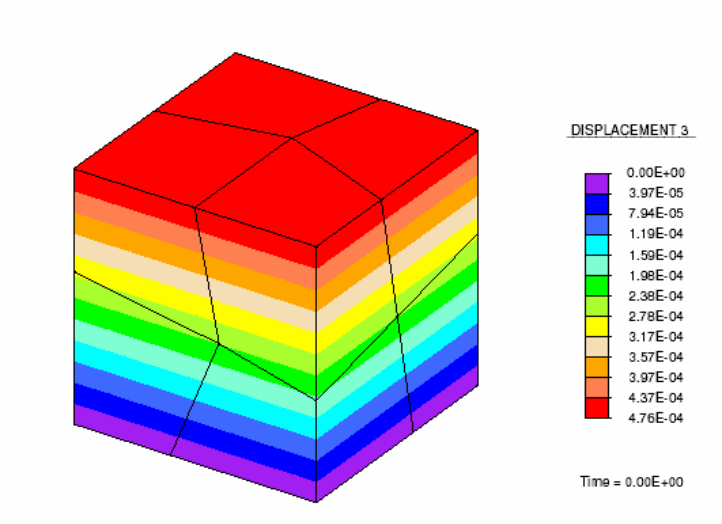


Figure 6: Displacement in the z-direction for the FPT with tension in the z-direction

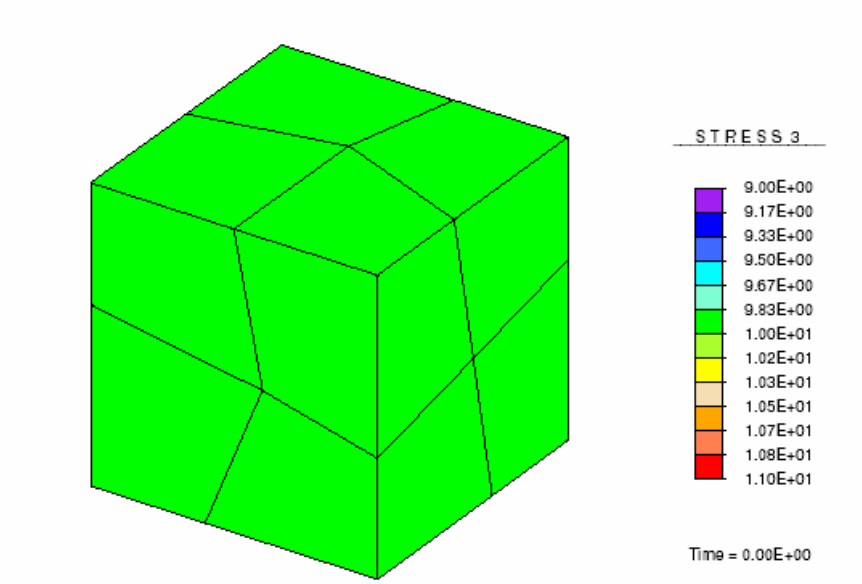


Figure 7: Stress σ_{zz} for the FPT with tension in the z-direction

Finally, the element is tested for shear. Here, the case where a surface traction in the y-direction is applied on the planes x=10 (positive traction) and x=0 (negative traction), and a traction in the x-direction on the planes y=10 (positive traction) and y=0 (negative traction), using minimum boundary conditions to avoid translations and solid body rotations, like shown in (Figure 8 : Applied shear and boundary conditions for the FPT with shear on the x- and y-planes). The other two shear cases are applying the corresponding force boundary conditions on the y- and z-planes and the x- and z-planes,

respectively. The σ_{xy} stress has to be equal to the applied shear traction of 10. The result for σ_{xy} is shown in (Figure 9: Stress σ_{xy} for the FPT with shear on the x- and y-planes). Although the color is not uniform in the plot, it can be seen from the color bar that the stress is indeed correct, FEAP is interpolating over the numeric error.

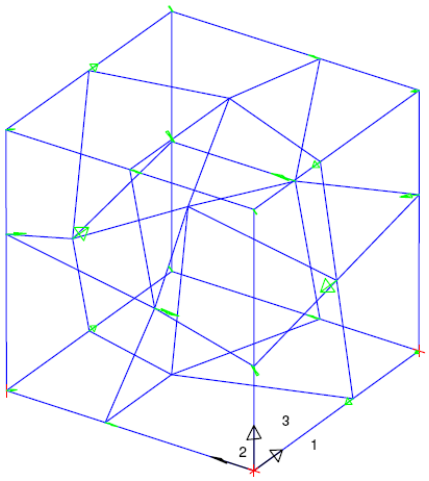


Figure 8 : Applied shear and boundary conditions for the FPT with shear on the x- and y-planes

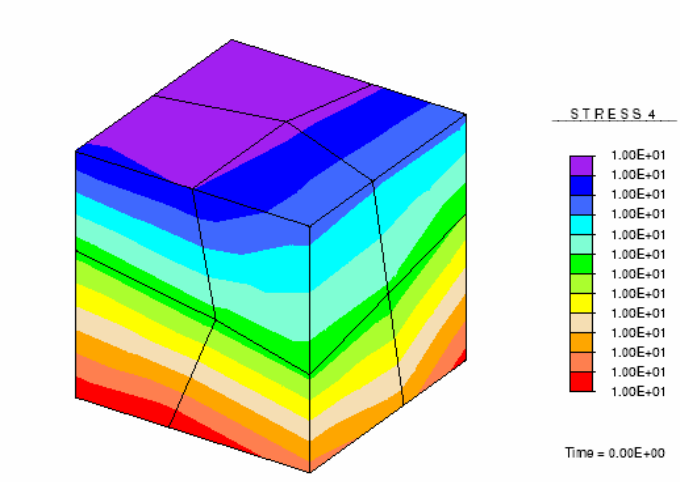


Figure 9: Stress σ_{xy} for the FPT with shear on the x- and y-planes

PERFORMANCE TESTS

There are a number of benchmark problems to test the performance of different elements. Among the most common for solid elements are bending of a beam, the distortion test and applying a small surface load on a nearly incompressible block. These tests are introduced in the following, although the convergence of the nearly incompressible block could not be studied due to computer limits.

T-Beam Test

The clamped beam was chosen with the dimensions shown in (Figure 10: T-beam under applied end deformation). Shear is applied to the plane x=50 by the means of a displacement of d=0.5 of this entire plane. As the displacement is only 1% of the beam length, the deformation is small enough so that using linear finite elements is appropriate.

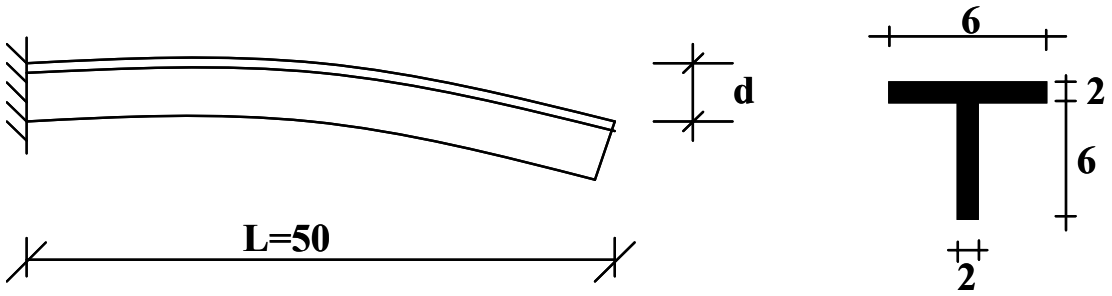


Figure 10: T-beam under applied end deformation

To save computational cost, only half the beam is modeled, as shown in (Figure 11: Meshes for the T-Beam using bricks and tetrahedrons), where the coarsest mesh is shown for both bricks and tetrahedrons, and symmetry boundary conditions are applied, as shown in (Figure 12: Boundary conditions for the T-Beam).

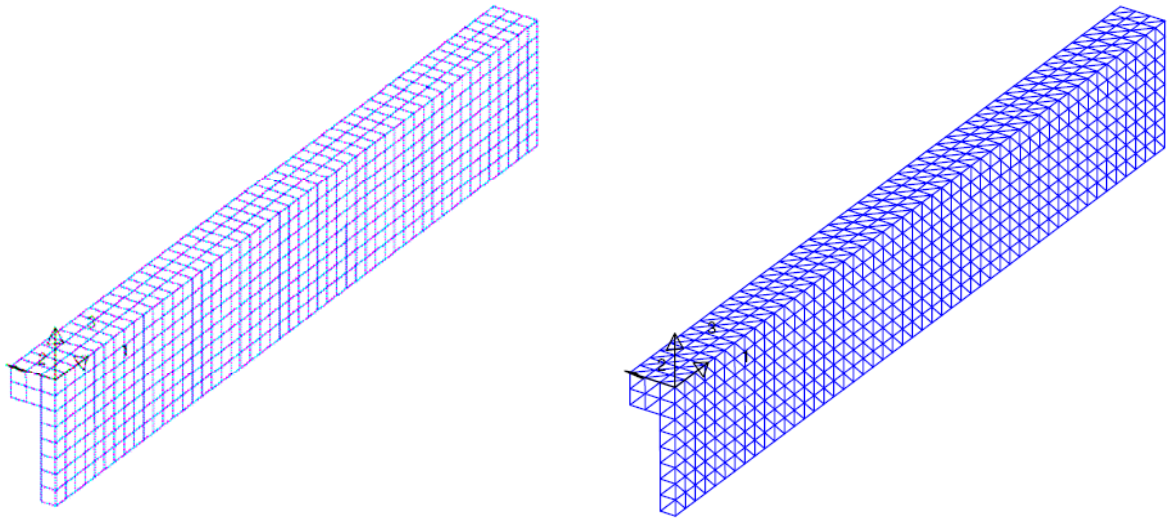


Figure 11: Meshes for the T-Beam using bricks and tetrahedrons

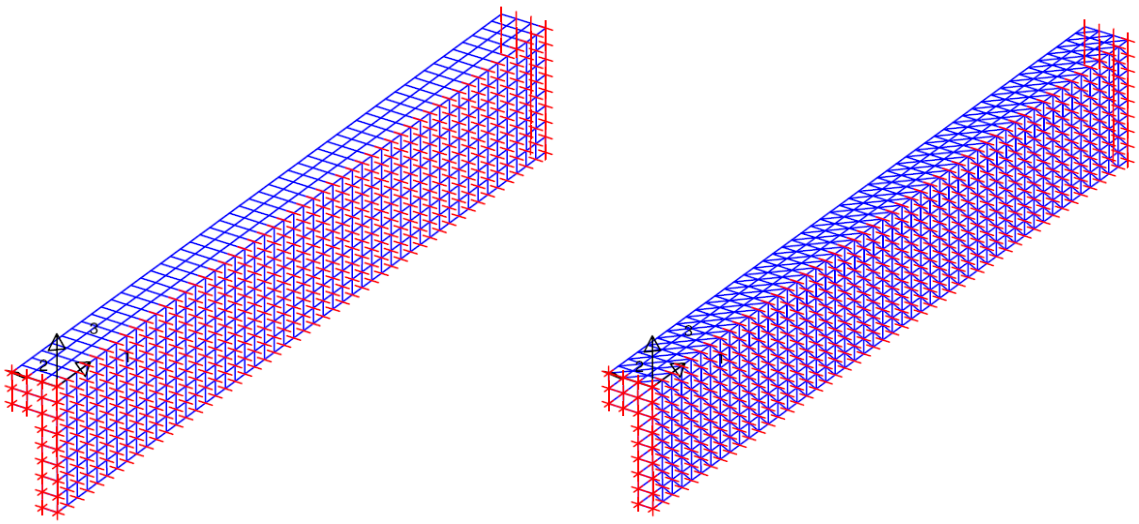


Figure 12: Boundary conditions for the T-Beam

The resulting σ_{xx} stress distribution for the quadratic brick and the quadratic tetrahedron for the coarsest mesh is shown in (Figure 13: Stress σ_{xx} for the quadratic brick) and (Figure 14: Stress σ_{xx} for the quadratic tetrahedron) respectively.

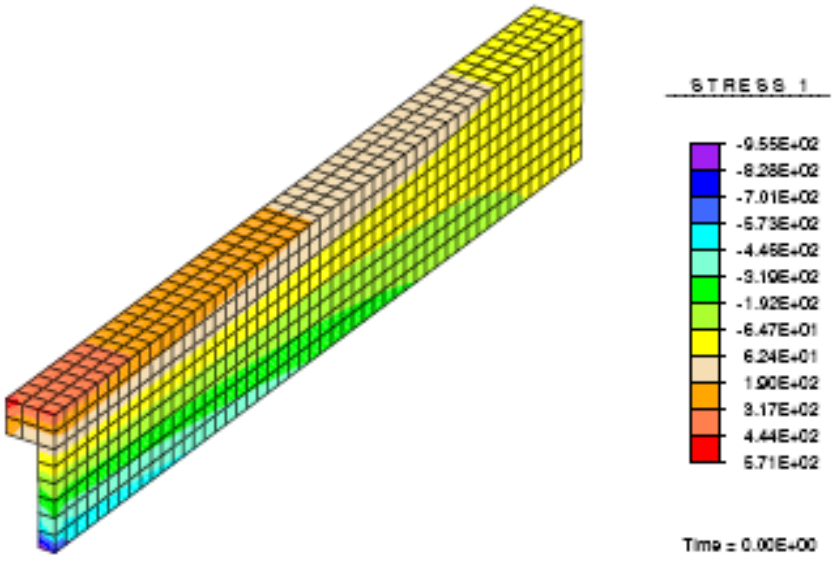


Figure 13: Stress σ_{xx} for the quadratic brick

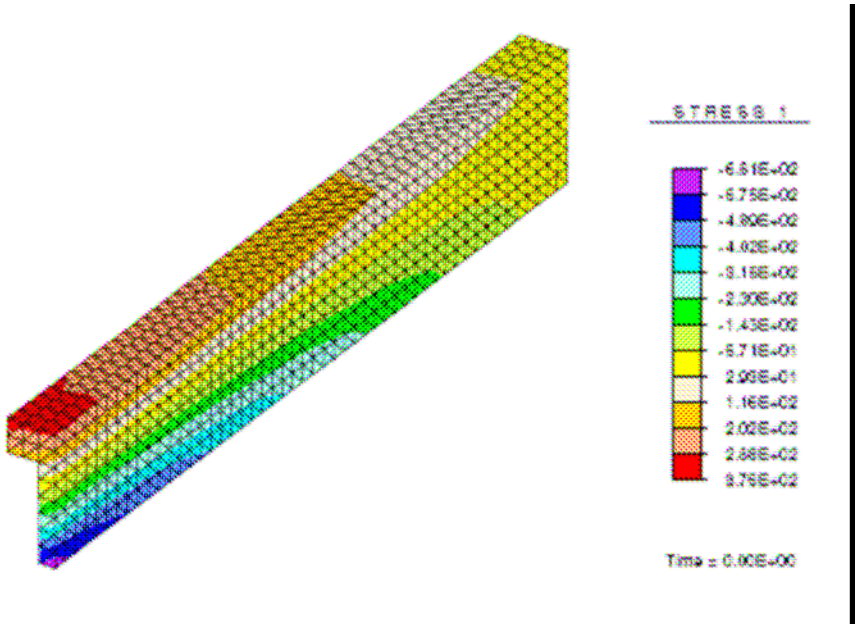


Figure 14: Stress σ_{xx} for the quadratic tetrahedron

However, the stress is not used directly to compare the performance of the different elements, but the sum of the nodal forces in the z-direction in the plane where the displacement is applied, at x=50. In (Table 4: Sum of reaction forces for all elements and different mesh resolutions) the sums of the nodal forces are listed for the different elements for different resolutions, where each resolution contains twice as many elements in each direction as the preceding. As the quadratic elements require at least two elements in each direction, no solution is obtained for the otherwise coarsest possible mesh. The relative error to the overkill solution in dependence of the mesh resolution is shown in (Figure 15: Relative error of the sum of the reaction forces for all elements), where the logarithmic values are plotted on axis. The last value for the quadratic brick is not shown, as this is used for the overkill solution, so the error is necessarily zero.

Resolution	T1	Q1	T2	Q2
1326	-181.95135	-169.18190		
7474	-171.16206	-167.81086	-167.43344	-167.35965
48642	-168.29125	-167.41347	-167.30943	-167.28170
347266	-167.52554	-167.29277	-167.26344	-167.25246

Table 4: Sum of reaction forces for all elements and different mesh resolutions

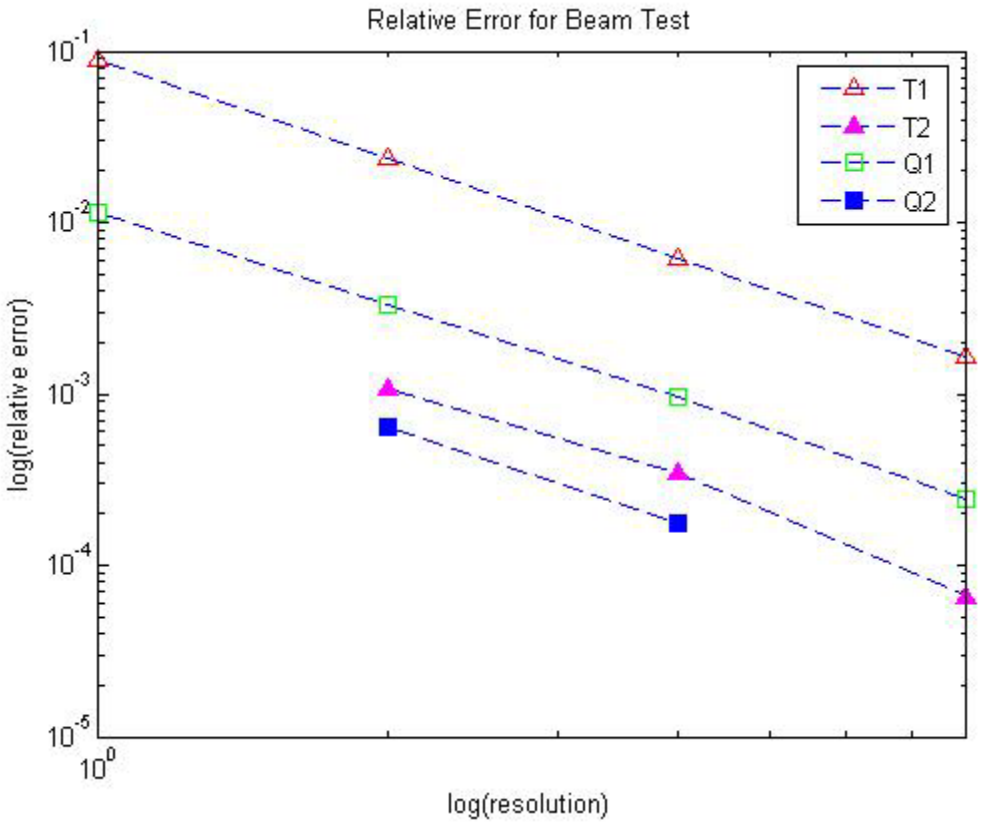


Figure 15: Relative error of the sum of the reaction forces for all elements for the beam test

It can be observed that the quadratic elements perform better than the linear ones, and that the brick perform better than the tetrahedrons. As can be seen from (Table 4: Sum of reaction forces for all elements and different mesh resolutions), the solution is better if the magnitude of the reactions forces is smaller. Therefore, the linear elements are obviously too stiff, a higher force is required to obtain a certain displacement than for the quadratic elements.

This behavior is due to bending locking. As shown in (**Figure 16: Bending behavior of a quadratic and a linear brick**), an effect called bending locking is responsible for bending locking. Quadratic shape functions are able to represent the actual deformation of the element much better than linear shape functions. If the same force is applied, the quadratic element deforms like the left element in (**Figure 16: Bending behavior of a quadratic and a linear brick**), while the linear element deforms like the right element in (**Figure 16: Bending behavior of a quadratic and a linear brick**). Obviously, the right element's reaction is to stiff, the displacement is too small. The other way around, when the same displacement is applied to linear and quadratic elements, a larger force results for the linear element.

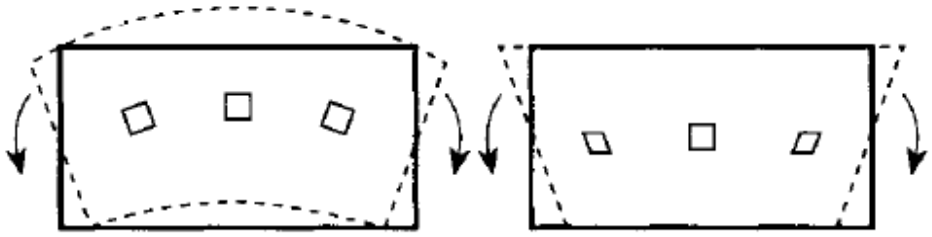


Figure 16: Bending behavior of a quadratic and a linear brick

Another way of comparing the quality of the solutions is to compare the eigenvalues of the stiffness matrices. The eigenvalues of the accurate solution are smaller than the ones of the too stiff solution, so the quality of the solution is the better the closer the eigenvalues are to the correct solution.
The bricks perform better than the tetrahedrons because the shape functions of the tetrahedrons are not able to describe the deformations modes as well as the shape functions for the brick, therefore, their error is greater to begin with.
The test shows that for quadratic elements, a fairly good solution can be obtained with a coarser mesh than with linear elements. However, the computational cost for quadratic elements is higher, e.g. there has to be a loop over 27 Gauss integration points instead of eight for the bricks. As an example, the CPU times for the beam with the finest mesh are 778.08 (T1), 694.61 (Q1), 1161.06 (T2) and 1615.01 (Q2). Therefore, the best option for each case, depending on the required accuracy of the solution, may be different.

Distortion test

The distortion test is to observe the physical behavior of the elements throughout the volume. A well defined element has an aspect ratio, ratio of the longest dimension to the shortest, close to one. Typically as the aspect ratio, distortion, increase the inaccuracy of the solution increases. Best results are obtained if an element is compact and regular[6]. For the distortion test, a clamped beam with a square cross section was chosen, with dimensions as shown in (Figure 17: Geometry of the beam used for the distortion test).

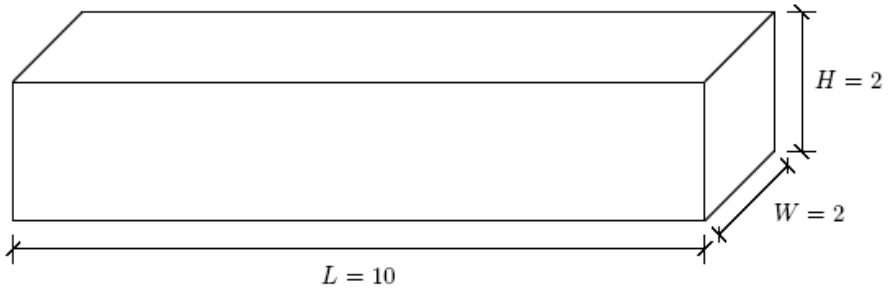


Figure 17: Geometry of the beam used for the distortion test

[6] Logan, D.L.: A First Course in the Finite Element Method. Brooks/Cole, 2000, Pacific Grove.

Again, only half the beam is modeled to save computational cost. The meshes for the coarsest resolution are shown in (Figure 18: Meshes for the distortion test using bricks and tetrahedrons) for both the brick and the tetrahedron, and the boundary conditions are shown in (Figure 19: Boundary conditions for the distortion test), including the symmetry boundary conditions. A displacement of d=0.1, again being in the range of a small displacement, is applied at x=10, and again the sum of the nodal forces in the z-direction in the plane at x=10 are compared.

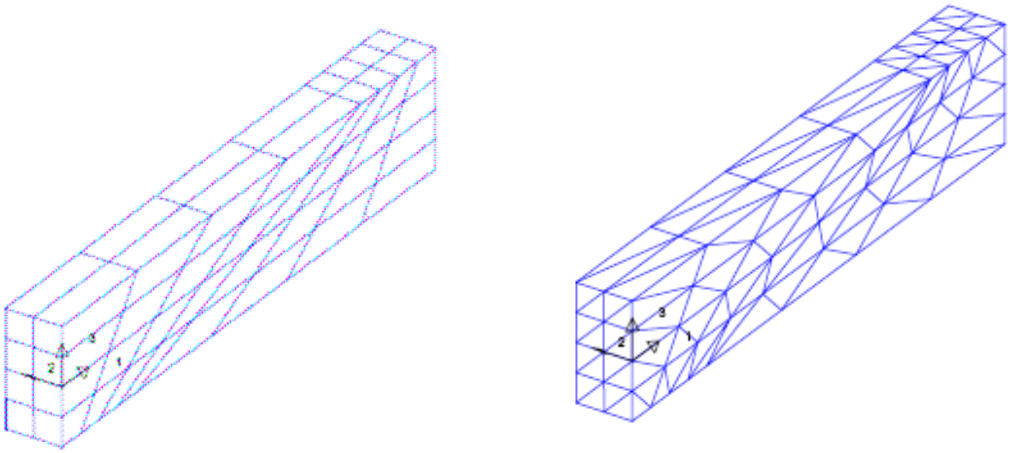


Figure 18: Meshes for the distortion test using bricks and tetrahedrons

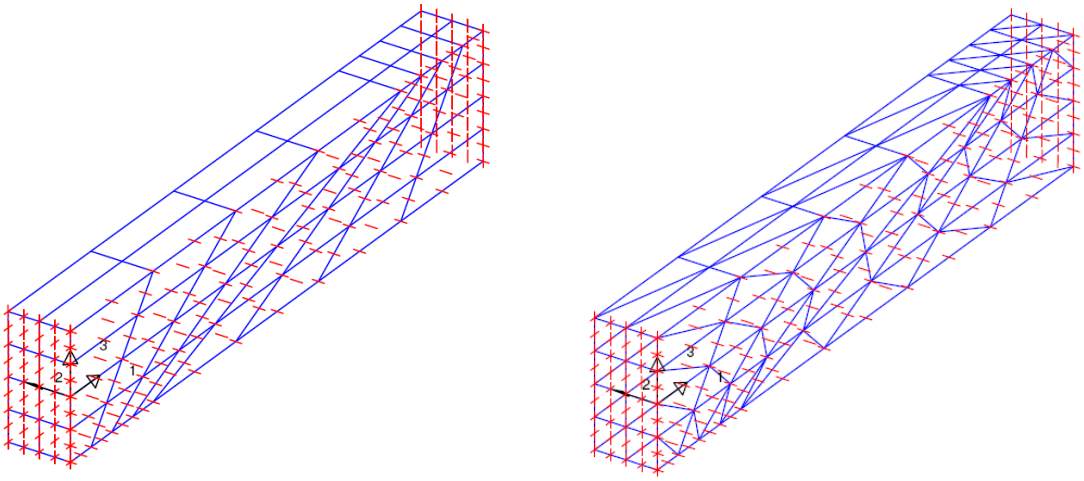


Figure 19: Boundary conditions for the distortion test

The resulting sums of the reactions forces at x=10 are shown in (Table 5: Sum of reaction forces for all elements and different mesh resolutions) for all elements and different resolutions. For the overkill solution, undistorted elements are used, as the goal is to

achieve the best solution possible. In (Figure 20: Relative error of the sum of the reaction forces for all elements for the distortion test), the relative error to the overkill solution is shown with respect to the mesh resolution, again using logarithmic scaling for both axes.

Resolution	T1	Q1	T2	Q2
8x2x4	156.13872	94.707573	74.282366	42.697313
16x4x8	87.048170	62.070269	44.457280	41.255605
32x8x16	58.268110	48.273627	41.305353	41.030872
64x16x32	46.530740	43.076042	41.020278	40.989481
128x32x64	42.516936	41.524465	40.984624	40.977960
overkill				40.977228

Table 5: Sum of reaction forces for all elements and different mesh resolutions

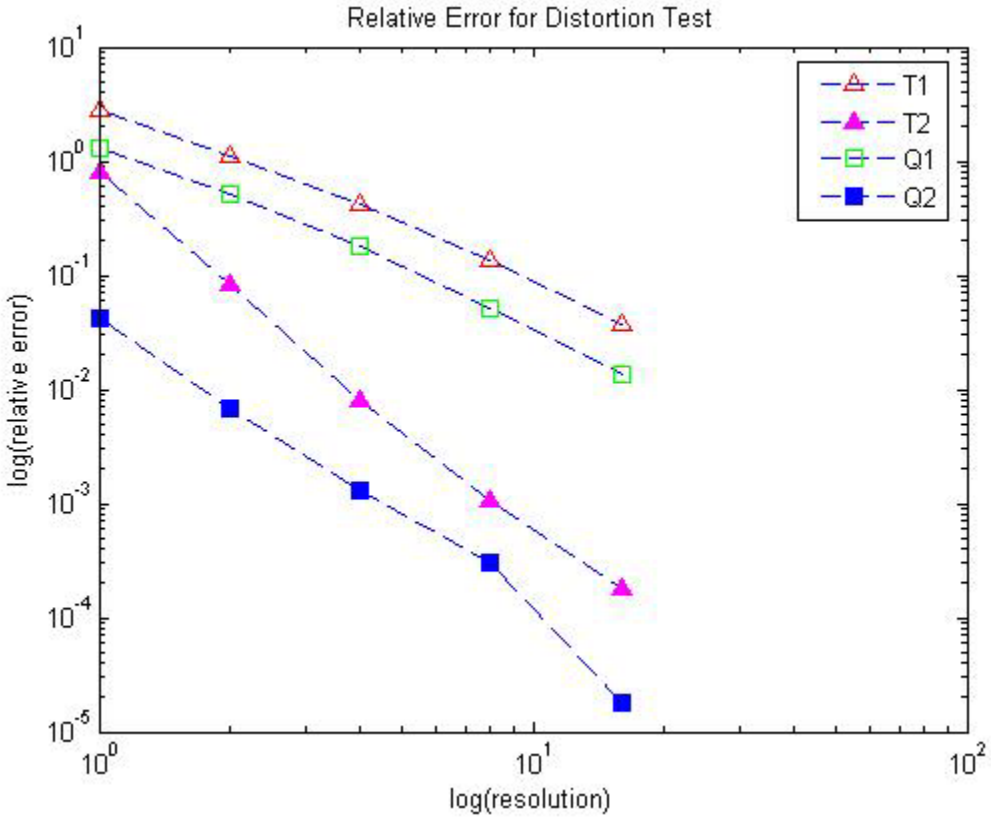


Figure 20: Relative error of the sum of the reaction forces for all elements for the distortion test

Again, the quadratic elements perform better than the linear ones, and the bricks perform better than the tetrahedrons due to the reasons stated above. However, it can be observed that the quadratic elements do not perform as well for coarse meshes as they do if the elements are not distorted. This happens because due to the distortion, the tri-quadratic shape functions cannot be displayed correctly any more, they are only really tri-quadratic in the reference element. The tetrahedrons have the same order of convergence as the bricks, however, they have a greater error to begin with, as the shape functions are not able to describe the deformations modes as well as the shape functions for the brick. The

Block test

Here, a nearly incompressible block with the dimensions 20x20x20 with a Poisson' ratio of ν=0.4999 was loaded with a traction of 200 on a small part of its surface, as shown in (Figure 21: Mesh and applied load for the block test). Only a quarter of the block is modeled, as it is double symmetric, and the symmetry boundary conditions are shown as in (Figure 22: Boundary conditions for block test). The block is restricted from moving in the z-direction at z=0.
Unfortunately, it was not possible to compute enough meshes to do a convergence study. However, the displacements in the z-direction for the linear and quadratic brick are shown in (Figure 23: Displacement in z-direction for the linear brick) and (Figure 24: Displacement in z-direction for the quadratic brick), respectively, and it can be seen that the displacements for the quadratic brick are of an order of a magnitude higher than for the linear brick.

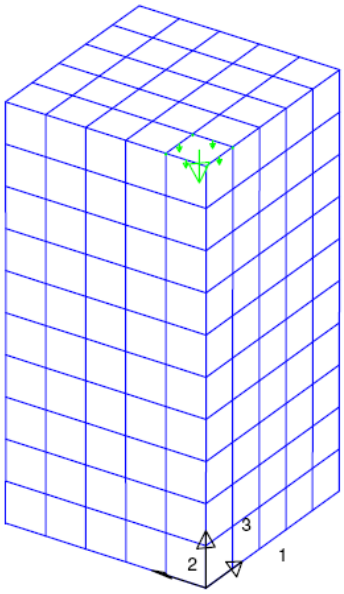


Figure 21: Mesh and applied load for the block test

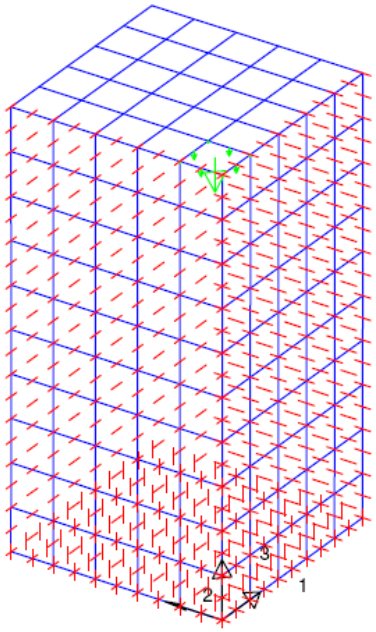


Figure 22: Boundary conditions for block test

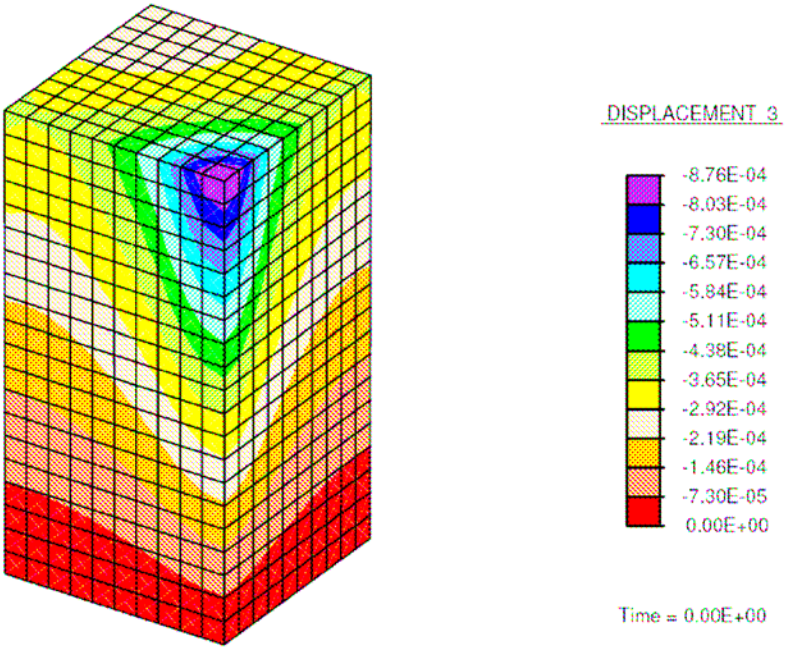


Figure 23: Displacement in z-direction for the linear brick

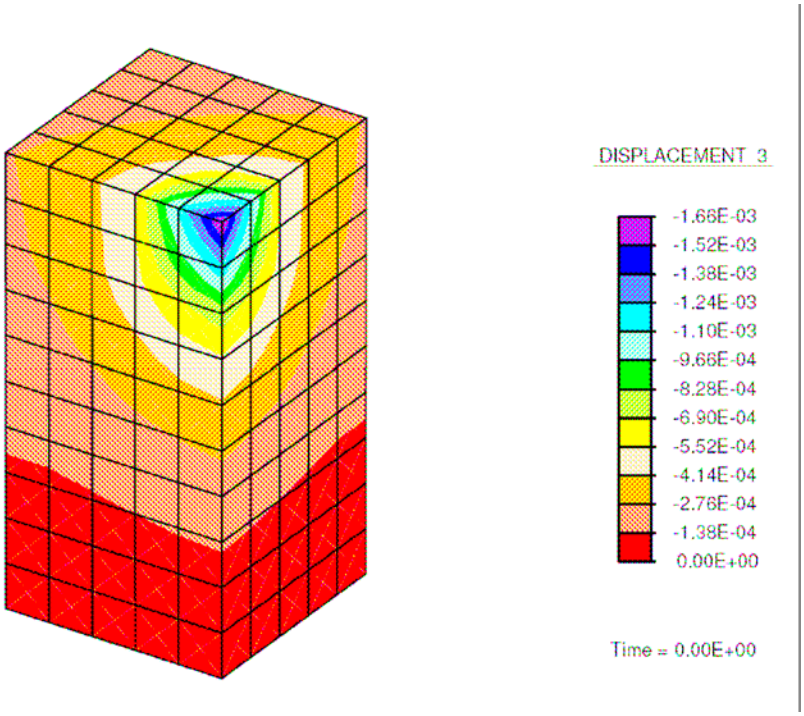


Figure 24: Displacement in z-direction for the quadratic brick

Here, the reason for the smaller displacement for the linear brick is incompressibility locking and can be best explained with the help of the example shown in (Figure 25: Incompressibility locking). A single element clamped like shown there under a single force wants to deform as the blue lines indicate. However, as the deformation must be volume preserving, the linear shape functions only allow a displacement of the node where the force is applied along the circle indicated in (Figure 25: Incompressibility locking), and this would be normal to the applied force. Therefore, the node is not displaced at all in this case.

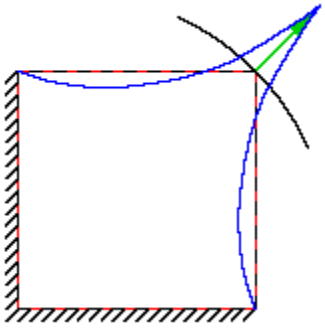


Figure 25: Incompressibility locking

CONCLUSION

In this term paper the performance of four isoparametric TPE elements is compared. The results are as expected: Quadratic elements perform better than linear elements because they do not lock, and bricks perform better then tetrahedrons because….

The tests performed here show the limitations due to locking and distortion sensitivity of the isoparametric elements. They are easy to implement and comparatively fast, especially the linear elements, but the performance for the tests considered here is questionable. A very fine mesh is necessary to obtain high quality results, and therefore high computer capacity. The quadratic elements perform better, even for a relatively coarse mesh, but as it has more Gauss integration points, the computational cost is higher compared to using the linear element for the same mesh.

These problems can be resolved using different element formulations that are especially able to deal with the problems encountered here.

For example, fixed-up isoparametric elements using reduced, selective or directional integration or incompatible displacement modes are also easy to implement and fix or at least reduce the problems regarding locking and distortion sensitivity. However, they often lead to regression bugs like hourglassing or invariance loss.

The mixed Hellinger-Reissner formulation, using the displacements and the stresses as master fields, are especially useful for incompressible materials, but they are difficult to implement and do not necessarily lead to the expected improvements[7].

Another possibility is using an element formulation that incorporates the analytical solution of bending, torsion and higher-order hourglassing and therefore does neither lock nor has the problems of the fixed-up isoparametric elements[8]. However, this element so far has its limitations as the analytical solutions are valid for rectangular parallelepipeds only, and not for arbitrary bricks.

Overall, the performed tests confirmed the expected results of the four different elements and showed clearly their strengths and weaknesses.

[7] Felippa, C.A.: AFEM Notes. 2006, Boulder.

[8] Nadler, B. and Rubin, M.B.: A new 3-D finite element for nonlinear elasticity using the theory of a Cosserat point, Int. J. Solids and Structures 40, 4585-4614, 2003.

REFERENCES

1. Felippa, C.A.: AFEM Notes. 2006, Boulder.

2. Logan, D.L.: A First Course in the Finite Element Method. Brooks/Cole, 2000, Pacific Grove.

3. Nadler, B. and Rubin, M.B.: A new 3-D finite element for nonlinear elasticity using the theory of a Cosserat point, Int. J. Solids and Structures 40, 4585-4614, 2003.

4. Wriggers, P.: Nichtlineare Finite-Elemente Methoden. Springer, 2001, Heidelberg.

5. Zienkiewicz, O.C. and Taylor, R.L.: The Finite Element Method, Volume 1: The Basis. Butterworth-Heinemann, 2000, Oxford.

APPENDIX

Element Files

Linear brick

```
c     3d 8node brick element (Q1)

      subroutine elmt01(d,ul,xl,ix,tl,s,p,ndf,ndm,nst,isw)

      implicit none

c     Include Files for own  FEAP-variables

      include 'comblk.h'
      include 'iofile.h'
      include 'eldata.h'
      include 'prstrs.h'
      include 'cdata.h'
      include 'strnum.h'


c     FEAP-variables

      integer ix(4), ndf , ndm , nst , isw
      real*8  d(*), ul(ndf,*), xl(ndm,*), tl(*), s(nst,*), p(nst)

c     own variables

      real*8 emod, nue  ! elastic modulus, poissons ratio
      real*8 sg(4,8)  ! gauss point coordinates and weights
      real*8 shp(4,8)  ! shape functions
```

```
      real*8 xsj  ! determinant of jacobian
      real*8 dd(6,6), matfac  ! material matrix, help-factor
      real*8 dvol  ! = determinant of jacobian * weight for gauss point
      real*8 b(6,3,8)  ! B-matrix
      real*8 eps(6)  ! strains (voigt)
      real*8 sigma(6)  ! stresses (voigt)
      real*8 sigp(12)  ! all values that are projected on the nodes,
                       ! stresses (1-6) and strains (7-12)
      integer ig,i1,i2,i3,i4,i5,i6,ih,j,k,m,ll,jj,kk,lint,l


c     ----------------------------------------------------------------
c     isw=1: read and plot material data
      if(isw.eq.1) then


      call pinput(d(1),2)
      write(iow,*)'   Linear brick element'
      write(iow,*)'   ------------------------'
      write(iow,*)'      Elastic modulus  E = ',d(1)
      write(iow,*)'      Poissons ratio   nue = ',d(2)
      write(*,*)'   Linear brick element'
      write(*,*)'   ------------------------'
      write(*,*)'     Elastic modulus  E = ',d(1)
      write(*,*)'     Poissons ratio   nue = ',d(2)

      call plbrk8(iel)

      end if


c     ----------------------------------------------------------------
c     isw=3 or isw=6: calculate stiffness matrix and residual
      if((isw.eq.3).or.(isw.eq.6)) then

      l = 2  ! 2 gauss points in each direction


c     material matrix
      emod=d(1)
      nue=d(2)

      call pzero(dd,36)
      matfac = emod/((1.d0-2.d0*nue)*(1.d0+nue))
      dd(1,1) = matfac*(1.d0-nue)
      dd(2,2) = dd(1,1)
      dd(3,3) = dd(1,1)
      dd(4,4) = matfac*(1.d0-2.d0*nue)/2.d0
      dd(5,5) = dd(4,4)
      dd(6,6) = dd(4,4)
      dd(1,2) = matfac*nue
      dd(2,1) = dd(1,2)
      dd(1,3) = dd(1,2)
      dd(3,1) = dd(1,2)
      dd(2,3) = dd(1,2)
      dd(3,2) = dd(1,2)
```

```
c     gauss points and weights
      call int3dlinbrick(l,lint,sg)

c     loop over all gauss points
      do l = 1,lint

c     shape functions for current gauss point
        call shp3dlinbrick(sg(1,l),xsj,shp,xl,ndm,nel)

        dvol = sg(4,l)*xsj

c     B-matrix
        call pzero(b,144)
        do i1 = 1,nel
          b(1,1,i1) = shp(1,i1)
          b(2,2,i1) = shp(2,i1)
          b(3,3,i1) = shp(3,i1)
          b(4,1,i1) = shp(2,i1)
          b(4,2,i1) = shp(1,i1)
          b(5,2,i1) = shp(3,i1)
          b(5,3,i1) = shp(2,i1)
          b(6,1,i1) = shp(3,i1)
          b(6,3,i1) = shp(1,i1)
        end do

c     strains
        call pzero(eps,6)
        do i1 = 1,nel
          eps(1) = eps(1) + b(1,1,i1) * ul(1,i1)  ! eps_11
          eps(2) = eps(2) + b(2,2,i1) * ul(2,i1)  ! eps_22
          eps(3) = eps(3) + b(3,3,i1) * ul(3,i1)  ! eps_33
          eps(4) = eps(4) + b(4,1,i1) * ul(1,i1)
     +                    + b(4,2,i1) * ul(2,i1)  ! 2*eps_12
          eps(5) = eps(5) + b(5,2,i1) * ul(2,i1)
     +                    + b(5,3,i1) * ul(3,i1)  ! 2*eps_23
          eps(6) = eps(6) + b(6,1,i1) * ul(1,i1)
     +                    + b(6,3,i1) * ul(3,i1)  ! 2*eps_31
        end do

c     stresses
        call pzero(sigma,6)
        do i1 = 1,6
          do i2 = 1,6
            sigma(i1) = sigma(i1) + dd(i1,i2)*eps(i2)
          end do
        end do

c     residual (negative)
      do i1=1,3
        do i2=1,8
          do i3=1,6
      p(i1+3*(i2-1)) = p(i1+3*(i2-1)) - sigma(i3)*b(i3,i1,i2)*dvol
          end do
        end do
      end do
```

```
c     stiffness matrix
      do i1=1,8
        do i2=1,3
          do i3=1,8
            do i4=1,3
              do i5=1,6
                do i6=1,6
      s(i2+3*(i1-1),i4+3*(i3-1))=s(i2+3*(i1-1),i4+3*(i3-1))
     +       +b(i5,i2,i1)*dd(i5,i6)*b(i6,i4,i3)*dvol
                end do
              end do
            end do
          end do
        end do
      end do


      end do  ! end loop over all gauss points

      end if

c     ----------------------------------------------------------------

c     isw=4 or isw=8: calculate strains, stresses and projected nodal stresses
      if((isw.eq.4).or.(isw.eq.8)) then

      l = 2  ! 2 gauss points in each direction


c     material matrix
      emod=d(1)
      nue=d(2)

      call pzero(dd,36)
      matfac = emod/((1.d0-2.d0*nue)*(1.d0+nue))
      dd(1,1) = matfac*(1.d0-nue)
      dd(2,2) = dd(1,1)
      dd(3,3) = dd(1,1)
      dd(4,4) = matfac*(1.d0-2.d0*nue)/2.d0
      dd(5,5) = dd(4,4)
      dd(6,6) = dd(4,4)
      dd(1,2) = matfac*nue
      dd(2,1) = dd(1,2)
      dd(1,3) = dd(1,2)
      dd(3,1) = dd(1,2)
      dd(2,3) = dd(1,2)
      dd(3,2) = dd(1,2)

c     gauss points and weights
      call int3dlinbrick(l,lint,sg)

c     loop over all gauss points
      do l = 1,lint

c     shape functions for current gauss point
```

```
      call shp3dlinbrick(sg(1,l),xsj,shp,xl,ndm,nel)

      dvol = sg(4,l)*xsj

c     B-matrix
      call pzero(b,144)
      do i1 = 1,nel
        b(1,1,i1) = shp(1,i1)
        b(2,2,i1) = shp(2,i1)
        b(3,3,i1) = shp(3,i1)
        b(4,1,i1) = shp(2,i1)
        b(4,2,i1) = shp(1,i1)
        b(5,2,i1) = shp(3,i1)
        b(5,3,i1) = shp(2,i1)
        b(6,1,i1) = shp(3,i1)
        b(6,3,i1) = shp(1,i1)
      end do

c     strains
      call pzero(eps,6)
      do i1 = 1,nel
        eps(1) = eps(1) + b(1,1,i1) * ul(1,i1)  ! eps_11
        eps(2) = eps(2) + b(2,2,i1) * ul(2,i1)  ! eps_22
        eps(3) = eps(3) + b(3,3,i1) * ul(3,i1)  ! eps_33
        eps(4) = eps(4) + b(4,1,i1) * ul(1,i1)
     +                  + b(4,2,i1) * ul(2,i1)  ! 2*eps_12
        eps(5) = eps(5) + b(5,2,i1) * ul(2,i1)
     +                  + b(5,3,i1) * ul(3,i1)  ! 2*eps_23
        eps(6) = eps(6) + b(6,1,i1) * ul(1,i1)
     +                  + b(6,3,i1) * ul(3,i1)  ! 2*eps_31
      end do

c     stresses
      call pzero(sigma,6)
      do i1 = 1,6
        do i2 = 1,6
          sigma(i1) = sigma(i1) + dd(i1,i2)*eps(i2)
        end do
      end do

      if(isw.ne.8) then
        write(iow,*)'  sigma_11', sigma(1)
        write(iow,*)'  sigma_22', sigma(2)
        write(iow,*)'  sigma_33', sigma(3)
        write(iow,*)'  sigma_12', sigma(4)
        write(iow,*)'  sigma_23', sigma(5)
        write(iow,*)'  sigma_31', sigma(6)
      else
        call pzero(sigp,12)
        sigp(1) = sigma(1)
        sigp(2) = sigma(2)
        sigp(3) = sigma(3)
        sigp(4) = sigma(4)
        sigp(5) = sigma(5)
        sigp(6) = sigma(6)
        sigp(7) = eps(1)
```

```
          sigp(8) = eps(2)
          sigp(9) = eps(3)
          sigp(10) = 0.5*eps(4)
          sigp(11) = 0.5*eps(5)
          sigp(12) = 0.5*eps(6)

          istv = 12  ! number of projected values

c     calculated projected nodal stresses and strains (lumped projection)

          call stcn33linbrick(ix,ul,shp,xsj,sigp,hr(nph),hr(nph+numnp),
     +               sg,l,ndm,nel,numnp,n)
        endif

      end do ! loop over gauss points

      end if


      end

c-----------------------------------------------------------------------
c-----------------------------------------------------------------------
c
c     subroutines
c
c-----------------------------------------------------------------------
c-----------------------------------------------------------------------


c-----------------------------------------------------------------------
c     gauss integration
c-----------------------------------------------------------------------

      subroutine int3dlinbrick(ll,lint,s)
      implicit none
c
      real*8 s(4,8)
      real*8 gp(2,2),wg(2,2)
      integer j,k,m,ll,jj,kk,lint


c     2 gauss points in each direction
      gp(2,1)=-0.577350269189626
      wg(2,1)=1.d0
      gp(2,2)= 0.577350269189626
      wg(2,2)=1.d0


      lint=ll*ll*ll  ! total number of gauss points

      do j=1,ll  ! index for z-coordinate
        jj=(j-1)*ll*ll
        do k=1,ll  ! index for y-coordinate
          kk=(k-1)*ll
          do m=1,ll  ! index for x-coordinate
```

```
          s(1,jj+kk+m)=gp(ll,m) ! x-coordinate of gauss point
          s(2,jj+kk+m)=gp(ll,k) ! y-coordinate of gauss point
          s(3,jj+kk+m)=gp(ll,j) ! z-coordinate of gauss point
          s(4,jj+kk+m)=wg(ll,m)*wg(ll,k)*wg(ll,j)  ! weight of gauss point
        end do
       end do
      end do

      return
      end


c-----------------------------------------------------------------------
c     shape functions
c-----------------------------------------------------------------------

      subroutine shp3dlinbrick(ss,xsj,shp,xl,ndm,nel)
      implicit none

      integer ndm,nel,i,j,k
      real*8 s1(8),s2(8),s3(8),xsj,a1,a2,a3,c1,c2,c3
      real*8 ss(3),shp(4,*),xl(ndm,*),xs(3,3),ad(3,3)
      data s1/-0.5, 0.5, 0.5,-0.5,-0.5, 0.5,0.5,-0.5/
      data s2/-0.5,-0.5, 0.5, 0.5,-0.5,-0.5,0.5, 0.5/
      data s3/-0.5,-0.5,-0.5,-0.5, 0.5, 0.5,0.5, 0.5/

c     shape functions
       do i = 1,8
         a1 = 0.5+s1(i)*ss(1)  ! shape function in xi-direction
         a2 = 0.5+s2(i)*ss(2)  ! shape function in eta-direction
         a3 = 0.5+s3(i)*ss(3)  ! shape function in mu-direction
         shp(1,i) = s1(i)*a2*a3  ! dN/d xi
         shp(2,i) = s2(i)*a1*a3  ! dN/d eta
         shp(3,i) = s3(i)*a1*a2  ! dN/d mu
         shp(4,i) = a1*a2*a3  ! actual shape function N
       end do


c     jacobian transformation
      do i = 1,3
       do j = 1,3
         xs(i,j) = 0.d0
         do k = 1,nel
           xs(i,j) = xs(i,j)+xl(j,k)*shp(i,k)
         end do
       end do
      end do

c     adjoint to jacobian
      ad(1,1) = xs(2,2)*xs(3,3)-xs(2,3)*xs(3,2)
      ad(1,2) = xs(1,3)*xs(3,2)-xs(1,2)*xs(3,3)
      ad(1,3) = xs(1,2)*xs(2,3)-xs(1,3)*xs(2,2)
      ad(2,1) = xs(2,3)*xs(3,1)-xs(2,1)*xs(3,3)
      ad(2,2) = xs(1,1)*xs(3,3)-xs(1,3)*xs(3,1)
      ad(2,3) = xs(1,3)*xs(2,1)-xs(1,1)*xs(2,3)
      ad(3,1) = xs(2,1)*xs(3,2)-xs(2,2)*xs(3,1)
```

```
      ad(3,2) = xs(1,2)*xs(3,1)-xs(1,1)*xs(3,2)
      ad(3,3) = xs(1,1)*xs(2,2)-xs(1,2)*xs(2,1)

c     determinant of jacobian
      xsj = xs(1,1)*ad(1,1)+xs(1,2)*ad(2,1)+xs(1,3)*ad(3,1)

c     jacobian inverse
      do i = 1,3
        do j = 1,3
          xs(i,j) = ad(i,j)/xsj
        end do
      end do

c     derivatives with repect to global coordinates

      do k = 1,nel
        c1 = xs(1,1)*shp(1,k)+xs(1,2)*shp(2,k)+xs(1,3)*shp(3,k)  ! dN/dx
        c2 = xs(2,1)*shp(1,k)+xs(2,2)*shp(2,k)+xs(2,3)*shp(3,k)  ! dN/dy
        c3 = xs(3,1)*shp(1,k)+xs(3,2)*shp(2,k)+xs(3,3)*shp(3,k)  ! dN/dz
        shp(1,k) = c1  ! store dN/dx instead of dN/d xi
        shp(2,k) = c2  ! store dN/dy instead of dN/d eta
        shp(3,k) = c3  ! store dN/dz instead of dN/d mu
      end do

      return
      end

c-----------------------------------------------------------------------
c     stress and strain projection
c-----------------------------------------------------------------------

      subroutine stcn33linbrick(ix,ul,shp,xsj,sigp,dt,st,sg,l,ndm,
     +                          nel, numnp,n)
      implicit none
c
      include  'iofile.h'
      include  'strnum.h'
c
      integer   ix(*),i,j,k,l,ll,ndm,nel,numnp,n
      real*8    ul(ndm,*),dt(*),st(numnp,*),sigp(*)
      real*8    shp(4,*),sg(4,*),xsj,xg

      do j = 1,nel  ! loop over element nodes
        ll = ix(j)  ! global node number
        xg = xsj*shp(4,j)*sg(4,j)
        dt(ll) = dt(ll)+xg  ! diagonal matrix
        do i=1,istv  ! loop over projected values
          st(ll,i) = st(ll,i)+sigp(i)*xg
        end do
      end do

      return
      end
```

Quadratic brick

```
c     3d 27node brick element (Q2)

      subroutine elmt02(d,ul,xl,ix,tl,s,p,ndf,ndm,nst,isw)

      implicit none

c     Include Files for own  FEAP-variables

      include 'comblk.h'
      include 'iofile.h'
      include 'eldata.h'
      include 'prstrs.h'
      include 'cdata.h'
      include 'strnum.h'


c     FEAP-variables

      integer ix(4), ndf , ndm , nst , isw
      real*8  d(*), ul(ndf,*), xl(ndm,*), tl(*), s(nst,*), p(nst)

c     own variables

      real*8 emod, nue  ! elastic modulus, poissons ratio
      real*8 sg(4,27)  ! gauss point coordinates and weights
      real*8 shp(4,27)  ! shape functions
      real*8 xsj  ! determinant of jacobian
      real*8 dd(6,6), matfac  ! material matrix, help-factor
      real*8 dvol  ! = determinant of jacobian * weight for gauss point
      real*8 b(6,3,27)  ! B-matrix
      real*8 eps(6)  ! strains (voigt)
      real*8 sigma(6)  ! stresses (voigt)
      real*8 sigp(12)  ! all values that are projected on the nodes,
                  ! stresses (1-6) and strains (7-12)
      integer ig,i1,i2,i3,i4,i5,i6,ih,j,k,m,ll,jj,kk,lint,l


c     ----------------------------------------------------------------
c     isw=1: read and plot material data
      if(isw.eq.1) then


      call pinput(d(1),2)
      write(iow,*)'   Quadratic brick element'
      write(iow,*)'   ------------------------'
      write(iow,*)'      Elastic modulus  E = ',d(1)
      write(iow,*)'      Poissons ratio   nue = ',d(2)
      write(*,*)'   Quadratic brick element'
      write(*,*)'   ------------------------'
      write(*,*)'     Elastic modulus  E = ',d(1)
      write(*,*)'     Poissons ratio   nue = ',d(2)
```

```
      call plbk27(iel)

      end if


c     ----------------------------------------------------------------
c     isw=3 or isw=6: calculate stiffness matrix and residual
      if((isw.eq.3).or.(isw.eq.6)) then

      l = 3  ! 3 gauss points in each direction


c     material matrix
      emod=d(1)
      nue=d(2)

      call pzero(dd,36)
      matfac = emod/((1.d0-2.d0*nue)*(1.d0+nue))
      dd(1,1) = matfac*(1.d0-nue)
      dd(2,2) = dd(1,1)
      dd(3,3) = dd(1,1)
      dd(4,4) = matfac*(1.d0-2.d0*nue)/2.d0
      dd(5,5) = dd(4,4)
      dd(6,6) = dd(4,4)
      dd(1,2) = matfac*nue
      dd(2,1) = dd(1,2)
      dd(1,3) = dd(1,2)
      dd(3,1) = dd(1,2)
      dd(2,3) = dd(1,2)
      dd(3,2) = dd(1,2)

c     gauss points and weights
      call int3dquadbrick(l,lint,sg)

c     loop over all gauss points
      do l = 1,lint

c     shape functions for current gauss point
        call shp3dquadbrick(sg(1,l),xsj,shp,xl,ndm,nel)

        dvol = sg(4,l)*xsj

c     B-matrix
        call pzero(b,486)
        do i1 = 1,nel
          b(1,1,i1) = shp(1,i1)
          b(2,2,i1) = shp(2,i1)
          b(3,3,i1) = shp(3,i1)
          b(4,1,i1) = shp(2,i1)
          b(4,2,i1) = shp(1,i1)
          b(5,2,i1) = shp(3,i1)
          b(5,3,i1) = shp(2,i1)
          b(6,1,i1) = shp(3,i1)
          b(6,3,i1) = shp(1,i1)
        end do
```

```
c     strains
        call pzero(eps,6)
        do i1 = 1,nel
          eps(1) = eps(1) + b(1,1,i1) * ul(1,i1)  ! eps_11
          eps(2) = eps(2) + b(2,2,i1) * ul(2,i1)  ! eps_22
          eps(3) = eps(3) + b(3,3,i1) * ul(3,i1)  ! eps_33
          eps(4) = eps(4) + b(4,1,i1) * ul(1,i1)
     +                    + b(4,2,i1) * ul(2,i1)  ! 2*eps_12
          eps(5) = eps(5) + b(5,2,i1) * ul(2,i1)
     +                    + b(5,3,i1) * ul(3,i1)  ! 2*eps_23
          eps(6) = eps(6) + b(6,1,i1) * ul(1,i1)
     +                    + b(6,3,i1) * ul(3,i1)  ! 2*eps_31
        end do

c     stresses
        call pzero(sigma,6)
        do i1 = 1,6
          do i2 = 1,6
            sigma(i1) = sigma(i1) + dd(i1,i2)*eps(i2)
          end do
        end do

c     residual (negative)
      do i1=1,3
        do i2=1,27
          do i3=1,6
      p(i1+3*(i2-1)) = p(i1+3*(i2-1)) - sigma(i3)*b(i3,i1,i2)*dvol
          end do
        end do
      end do

c     stiffness matrix
      do i1=1,27
        do i2=1,3
          do i3=1,27
            do i4=1,3
              do i5=1,6
                do i6=1,6
      s(i2+3*(i1-1),i4+3*(i3-1))=s(i2+3*(i1-1),i4+3*(i3-1))
     +       +b(i5,i2,i1)*dd(i5,i6)*b(i6,i4,i3)*dvol
                end do
              end do
            end do
          end do
        end do
      end do


      end do  ! end loop over all gauss points

      end if

c     ----------------------------------------------------------------

c     isw=4 or isw=8: calculate strains, stresses and projected nodal stresses
```

```
      if((isw.eq.4).or.(isw.eq.8)) then

      l = 3  ! 2 gauss points in each direction


c     material matrix
      emod=d(1)
      nue=d(2)

      call pzero(dd,36)
      matfac = emod/((1.d0-2.d0*nue)*(1.d0+nue))
      dd(1,1) = matfac*(1.d0-nue)
      dd(2,2) = dd(1,1)
      dd(3,3) = dd(1,1)
      dd(4,4) = matfac*(1.d0-2.d0*nue)/2.d0
      dd(5,5) = dd(4,4)
      dd(6,6) = dd(4,4)
      dd(1,2) = matfac*nue
      dd(2,1) = dd(1,2)
      dd(1,3) = dd(1,2)
      dd(3,1) = dd(1,2)
      dd(2,3) = dd(1,2)
      dd(3,2) = dd(1,2)

c     gauss points and weights
      call int3dquadbrick(l,lint,sg)

c     loop over all gauss points
      do l = 1,lint

c     shape functions for current gauss point
        call shp3dquadbrick(sg(1,l),xsj,shp,xl,ndm,nel)

        dvol = sg(4,l)*xsj

c     B-matrix
        call pzero(b,486)
        do i1 = 1,nel
          b(1,1,i1) = shp(1,i1)
          b(2,2,i1) = shp(2,i1)
          b(3,3,i1) = shp(3,i1)
          b(4,1,i1) = shp(2,i1)
          b(4,2,i1) = shp(1,i1)
          b(5,2,i1) = shp(3,i1)
          b(5,3,i1) = shp(2,i1)
          b(6,1,i1) = shp(3,i1)
          b(6,3,i1) = shp(1,i1)
        end do

c     strains
        call pzero(eps,6)
        do i1 = 1,nel
          eps(1) = eps(1) + b(1,1,i1) * ul(1,i1)  ! eps_11
          eps(2) = eps(2) + b(2,2,i1) * ul(2,i1)  ! eps_22
          eps(3) = eps(3) + b(3,3,i1) * ul(3,i1)  ! eps_33
          eps(4) = eps(4) + b(4,1,i1) * ul(1,i1)
```

```
     +                  + b(4,2,i1) * ul(2,i1)  ! 2*eps_12
           eps(5) = eps(5) + b(5,2,i1) * ul(2,i1)
     +                  + b(5,3,i1) * ul(3,i1)  ! 2*eps_23
           eps(6) = eps(6) + b(6,1,i1) * ul(1,i1)
     +                  + b(6,3,i1) * ul(3,i1)  ! 2*eps_31
         end do

c     stresses
         call pzero(sigma,6)
         do i1 = 1,6
           do i2 = 1,6
             sigma(i1) = sigma(i1) + dd(i1,i2)*eps(i2)
           end do
         end do

         if(isw.ne.8) then
           write(iow,*)'  sigma_11', sigma(1)
           write(iow,*)'  sigma_22', sigma(2)
           write(iow,*)'  sigma_33', sigma(3)
           write(iow,*)'  sigma_12', sigma(4)
           write(iow,*)'  sigma_23', sigma(5)
           write(iow,*)'  sigma_31', sigma(6)
         else
           call pzero(sigp,12)
           sigp(1) = sigma(1)
           sigp(2) = sigma(2)
           sigp(3) = sigma(3)
           sigp(4) = sigma(4)
           sigp(5) = sigma(5)
           sigp(6) = sigma(6)
           sigp(7) = eps(1)
           sigp(8) = eps(2)
           sigp(9) = eps(3)
           sigp(10) = 0.5*eps(4)
           sigp(11) = 0.5*eps(5)
           sigp(12) = 0.5*eps(6)

           istv = 12  ! number of projected values

c     calculated projected nodal stresses and strains (lumped projection)

           call stcn33quadbrick(ix,ul,shp,xsj,sigp,hr(nph),
     +               hr(nph+numnp),sg,l,ndm,nel,numnp,n)
         endif

       end do ! loop over gauss points

       end if


       end

c-----------------------------------------------------------------------
c-----------------------------------------------------------------------
c
c     subroutines
```

```
c
c-----------------------------------------------------------------------
c-----------------------------------------------------------------------


c-----------------------------------------------------------------------
c     gauss integration
c-----------------------------------------------------------------------

      subroutine int3dquadbrick(ll,lint,s)
      implicit none
c
      real*8 s(4,27)
      real*8 gp(3,3),wg(3,3)
      integer j,k,m,ll,jj,kk,lint


c     3 gauss points in each direction
      gp(3,1)=-0.774596669241483
      wg(3,1)=0.555555555555556
      gp(3,2)= 0.d0
      wg(3,2)=0.888888888888889
      gp(3,3)= 0.774596669241483
      wg(3,3)=0.555555555555556


      lint=ll*ll*ll  ! total number of gauss points

      do j=1,ll  ! index for z-coordinate
        jj=(j-1)*ll*ll
        do k=1,ll  ! index for y-coordinate
          kk=(k-1)*ll
          do m=1,ll  ! index for x-coordinate
            s(1,jj+kk+m)=gp(ll,m) ! x-coordinate of gauss point
            s(2,jj+kk+m)=gp(ll,k) ! y-coordinate of gauss point
            s(3,jj+kk+m)=gp(ll,j) ! z-coordinate of gauss point
            s(4,jj+kk+m)=wg(ll,m)*wg(ll,k)*wg(ll,j)  ! weight of gauss point
          end do
        end do
      end do

      return
      end


c-----------------------------------------------------------------------
c     shape functions
c-----------------------------------------------------------------------

      subroutine shp3dquadbrick(ss,xsj,shp,xl,ndm,nel)
      implicit none

      integer ir(27),is(27),it(27)
      integer ndm,nel,i,j,k
      real*8 xsj,c1,c2,c3
      real*8 ss(3),shp(4,*),xl(ndm,*),xs(3,3),ad(3,3)
```

```
      real*8 nr(3),dr(3),ns(3),ds(3),nt(3),dt(3)

      data      ir /1,2,2,1, 1,2,2,1, 3,2,3,1,
     &             3,2,3,1, 1,2,2,1, 3,3,1,2,3,3, 3/
      data      is /1,1,2,2, 1,1,2,2, 1,3,2,3,
     &             1,3,2,3, 1,1,2,2, 3,3,3,3,1,2, 3/
      data      it /1,1,1,1, 2,2,2,2, 1,1,1,1,
     &             2,2,2,2, 3,3,3,3, 1,2,3,3,3,3, 3/


c     set 1-d shape functions for each local direction

c     shape funtions in xi-direction
      nr(1) =  0.5d0 * ss(1) * (ss(1) - 1.d0)
      nr(2) =  0.5d0 * ss(1) * (ss(1) + 1.d0)
      nr(3) =  1.0d0 - ss(1) *  ss(1)

c     shape funtions in eta-direction
      ns(1) =  0.5d0 * ss(2) * (ss(2) - 1.d0)
      ns(2) =  0.5d0 * ss(2) * (ss(2) + 1.d0)
      ns(3) =  1.0d0 - ss(2) *  ss(2)

c     shape funtions in mu-direction
      nt(1) =  0.5d0 * ss(3) * (ss(3) - 1.d0)
      nt(2) =  0.5d0 * ss(3) * (ss(3) + 1.d0)
      nt(3) =  1.0d0 - ss(3) *  ss(3)

c     derivatives
      dr(1) =  ss(1) - 0.5d0
      dr(2) =  ss(1) + 0.5d0
      dr(3) = -ss(1) - ss(1)

c     derivatives
      ds(1) =  ss(2) - 0.5d0
      ds(2) =  ss(2) + 0.5d0
      ds(3) = -ss(2) - ss(2)

c     derivatives
      dt(1) =  ss(3) - 0.5d0
      dt(2) =  ss(3) + 0.5d0
      dt(3) = -ss(3) - ss(3)

c     set local 3-d shape functions

      do i = 1,27
        shp(1,i) = dr(ir(i)) * ns(is(i)) * nt(it(i)) ! d N/ d xi
        shp(2,i) = nr(ir(i)) * ds(is(i)) * nt(it(i)) ! d N/ d eta
        shp(3,i) = nr(ir(i)) * ns(is(i)) * dt(it(i)) ! d N/ d mu
        shp(4,i) = nr(ir(i)) * ns(is(i)) * nt(it(i)) ! N
      end do


c     jacobian transformation
      do i = 1,3
        do j = 1,3
          xs(i,j) = 0.d0
```

```
        do k = 1,nel
          xs(i,j) = xs(i,j)+xl(j,k)*shp(i,k)
        end do
      end do
    end do

c     adjoint to jacobian
      ad(1,1) = xs(2,2)*xs(3,3)-xs(2,3)*xs(3,2)
      ad(1,2) = xs(1,3)*xs(3,2)-xs(1,2)*xs(3,3)
      ad(1,3) = xs(1,2)*xs(2,3)-xs(1,3)*xs(2,2)
      ad(2,1) = xs(2,3)*xs(3,1)-xs(2,1)*xs(3,3)
      ad(2,2) = xs(1,1)*xs(3,3)-xs(1,3)*xs(3,1)
      ad(2,3) = xs(1,3)*xs(2,1)-xs(1,1)*xs(2,3)
      ad(3,1) = xs(2,1)*xs(3,2)-xs(2,2)*xs(3,1)
      ad(3,2) = xs(1,2)*xs(3,1)-xs(1,1)*xs(3,2)
      ad(3,3) = xs(1,1)*xs(2,2)-xs(1,2)*xs(2,1)

c     determinant of jacobian
      xsj = xs(1,1)*ad(1,1)+xs(1,2)*ad(2,1)+xs(1,3)*ad(3,1)

c     jacobian inverse
      do i = 1,3
        do j = 1,3
          xs(i,j) = ad(i,j)/xsj
        end do
      end do

c     derivatives with repect to global coordinates

      do k = 1,nel
        c1 = xs(1,1)*shp(1,k)+xs(1,2)*shp(2,k)+xs(1,3)*shp(3,k)  ! dN/dx
        c2 = xs(2,1)*shp(1,k)+xs(2,2)*shp(2,k)+xs(2,3)*shp(3,k)  ! dN/dy
        c3 = xs(3,1)*shp(1,k)+xs(3,2)*shp(2,k)+xs(3,3)*shp(3,k)  ! dN/dz
        shp(1,k) = c1  ! store dN/dx instead of dN/d xi
        shp(2,k) = c2  ! store dN/dy instead of dN/d eta
        shp(3,k) = c3  ! store dN/dz instead of dN/d mu
      end do

      return
      end

c-----------------------------------------------------------------------
c     stress and strain projection
c-----------------------------------------------------------------------

      subroutine stcn33quadbrick(ix,ul,shp,xsj,sigp,dt,st,sg,l,ndm,
     +                     nel, numnp,n)
      implicit none
c
      include  'iofile.h'
      include  'strnum.h'
c
      integer   ix(*),i,j,k,l,ll,ndm,nel,numnp,n
      real*8    ul(ndm,*),dt(*),st(numnp,*),sigp(*)
      real*8    shp(4,*),sg(4,*),xsj,xg
```

```
      do j = 1,nel  ! loop over element nodes
        ll = ix(j)  ! global node number
        xg = xsj*shp(4,j)*sg(4,j)
        dt(ll) = dt(ll)+xg  ! diagonal matrix
        do i=1,istv  ! loop over projected values
          st(ll,i) = st(ll,i)+sigp(i)*xg
        end do
      end do

      return
      end
```

Linear tetrahedron

```
c     3d linear tetraheder element

      subroutine elmt03(d,ul,xl,ix,tl,s,p,ndf,ndm,nst,isw)

      implicit none

c     Include Files for own  FEAP-variables

      include 'comblk.h'
      include 'iofile.h'
      include 'eldata.h'
      include 'prstrs.h'
      include 'cdata.h'
      include 'strnum.h'


c     FEAP-variables

      integer ix(4), ndf , ndm , nst , isw
      real*8  d(*), ul(ndf,*), xl(ndm,*), tl(*), s(nst,*), p(nst)

c     own variables

      real*8 emod, nue  ! elastic modulus, poissons ratio
      real*8 sg(4,1)  ! gauss point coordinates and weights
      real*8 shp(4,4)  ! shape functions
      real*8 xsj  ! determinant of jacobian
      real*8 dd(6,6), matfac  ! material matrix, help-factor
      real*8 dvol  ! = determinant of jacobian * weight for gauss point
      real*8 b(6,3,4)  ! B-matrix
      real*8 eps(6)  ! strains (voigt)
      real*8 sigma(6)  ! stresses (voigt)
      real*8 sigp(12)  ! all values that are projected on the nodes,
                       ! stresses (1-6) and strains (7-12)
      integer ig,i1,i2,i3,i4,i5,i6,ih,j,k,m,ll,jj,kk,lint,l


c     ----------------------------------------------------------------
c     isw=1: read and plot material data
```

```
      if(isw.eq.1) then


      call pinput(d(1),2)
      write(iow,*)'   Linear tetrahedron'
      write(iow,*)'   ------------------------'
      write(iow,*)'      Elastic modulus  E = ',d(1)
      write(iow,*)'      Poissons ratio   nue = ',d(2)
      write(*,*)'   Linear tetrahedron'
      write(*,*)'   ------------------------'
      write(*,*)'     Elastic modulus  E = ',d(1)
      write(*,*)'     Poissons ratio   nue = ',d(2)

      call plbrk8(iel)

      end if


c     ----------------------------------------------------------------
c     isw=3 or isw=6: calculate stiffness matrix and residual
      if((isw.eq.3).or.(isw.eq.6)) then

c     material matrix
      emod=d(1)
      nue=d(2)

      call pzero(dd,36)
      matfac = emod/((1.d0-2.d0*nue)*(1.d0+nue))
      dd(1,1) = matfac*(1.d0-nue)
      dd(2,2) = dd(1,1)
      dd(3,3) = dd(1,1)
      dd(4,4) = matfac*(1.d0-2.d0*nue)/2.d0
      dd(5,5) = dd(4,4)
      dd(6,6) = dd(4,4)
      dd(1,2) = matfac*nue
      dd(2,1) = dd(1,2)
      dd(1,3) = dd(1,2)
      dd(3,1) = dd(1,2)
      dd(2,3) = dd(1,2)
      dd(3,2) = dd(1,2)

c     gauss point and weight
      call int3dlintet(sg)


c     shape functions for gauss point
      call shp3dlintet(sg,xsj,shp,xl,ndm,nel)

      dvol = sg(4,1)*xsj

c     B-matrix
        call pzero(b,72)
        do i1 = 1,nel
          b(1,1,i1) = shp(1,i1)
          b(2,2,i1) = shp(2,i1)
          b(3,3,i1) = shp(3,i1)
```

```
          b(4,1,i1) = shp(2,i1)
          b(4,2,i1) = shp(1,i1)
          b(5,2,i1) = shp(3,i1)
          b(5,3,i1) = shp(2,i1)
          b(6,1,i1) = shp(3,i1)
          b(6,3,i1) = shp(1,i1)
        end do

c     strains
        call pzero(eps,6)
        do i1 = 1,nel
          eps(1) = eps(1) + b(1,1,i1) * ul(1,i1)  ! eps_11
          eps(2) = eps(2) + b(2,2,i1) * ul(2,i1)  ! eps_22
          eps(3) = eps(3) + b(3,3,i1) * ul(3,i1)  ! eps_33
          eps(4) = eps(4) + b(4,1,i1) * ul(1,i1)
     +                    + b(4,2,i1) * ul(2,i1)  ! 2*eps_12
          eps(5) = eps(5) + b(5,2,i1) * ul(2,i1)
     +                    + b(5,3,i1) * ul(3,i1)  ! 2*eps_23
          eps(6) = eps(6) + b(6,1,i1) * ul(1,i1)
     +                    + b(6,3,i1) * ul(3,i1)  ! 2*eps_31
        end do

c     stresses
        call pzero(sigma,6)
        do i1 = 1,6
          do i2 = 1,6
            sigma(i1) = sigma(i1) + dd(i1,i2)*eps(i2)
          end do
        end do

c     residual (negative)
      do i1=1,3
        do i2=1,1
          do i3=1,6
      p(i1+3*(i2-1)) = p(i1+3*(i2-1)) - sigma(i3)*b(i3,i1,i2)*dvol
          end do
        end do
      end do

c     stiffness matrix
      do i1=1,4
        do i2=1,3
          do i3=1,4
            do i4=1,3
              do i5=1,6
                do i6=1,6
      s(i2+3*(i1-1),i4+3*(i3-1))=s(i2+3*(i1-1),i4+3*(i3-1))
     +       +b(i5,i2,i1)*dd(i5,i6)*b(i6,i4,i3)*dvol
                end do
              end do
            end do
          end do
        end do
      end do

      end if
```

```
c     ------------------------------------------------------------------

c     isw=4 or isw=8: calculate strains, stresses and projected nodal stresses
      if((isw.eq.4).or.(isw.eq.8)) then

c     material matrix
      emod=d(1)
      nue=d(2)

      call pzero(dd,36)
      matfac = emod/((1.d0-2.d0*nue)*(1.d0+nue))
      dd(1,1) = matfac*(1.d0-nue)
      dd(2,2) = dd(1,1)
      dd(3,3) = dd(1,1)
      dd(4,4) = matfac*(1.d0-2.d0*nue)/2.d0
      dd(5,5) = dd(4,4)
      dd(6,6) = dd(4,4)
      dd(1,2) = matfac*nue
      dd(2,1) = dd(1,2)
      dd(1,3) = dd(1,2)
      dd(3,1) = dd(1,2)
      dd(2,3) = dd(1,2)
      dd(3,2) = dd(1,2)

c     gauss point and weight
      call int3dlintet(sg)

c     shape functions for gauss point
        call shp3dlintet(sg,xsj,shp,xl,ndm,nel)

        dvol = sg(4,1)*xsj

c     B-matrix
        call pzero(b,72)
        do i1 = 1,nel
          b(1,1,i1) = shp(1,i1)
          b(2,2,i1) = shp(2,i1)
          b(3,3,i1) = shp(3,i1)
          b(4,1,i1) = shp(2,i1)
          b(4,2,i1) = shp(1,i1)
          b(5,2,i1) = shp(3,i1)
          b(5,3,i1) = shp(2,i1)
          b(6,1,i1) = shp(3,i1)
          b(6,3,i1) = shp(1,i1)
        end do

c     strains
        call pzero(eps,6)
        do i1 = 1,nel
          eps(1) = eps(1) + b(1,1,i1) * ul(1,i1)  ! eps_11
          eps(2) = eps(2) + b(2,2,i1) * ul(2,i1)  ! eps_22
          eps(3) = eps(3) + b(3,3,i1) * ul(3,i1)  ! eps_33
          eps(4) = eps(4) + b(4,1,i1) * ul(1,i1)
     +                    + b(4,2,i1) * ul(2,i1)  ! 2*eps_12
          eps(5) = eps(5) + b(5,2,i1) * ul(2,i1)
```

```
     +                  + b(5,3,i1) * ul(3,i1)  ! 2*eps_23
          eps(6) = eps(6) + b(6,1,i1) * ul(1,i1)
     +                  + b(6,3,i1) * ul(3,i1)  ! 2*eps_31
        end do

c     stresses
        call pzero(sigma,6)
        do i1 = 1,6
          do i2 = 1,6
            sigma(i1) = sigma(i1) + dd(i1,i2)*eps(i2)
          end do
        end do

        if(isw.ne.8) then
          write(iow,*)'  sigma_11', sigma(1)
          write(iow,*)'  sigma_22', sigma(2)
          write(iow,*)'  sigma_33', sigma(3)
          write(iow,*)'  sigma_12', sigma(4)
          write(iow,*)'  sigma_23', sigma(5)
          write(iow,*)'  sigma_31', sigma(6)
        else
          call pzero(sigp,12)
          sigp(1) = sigma(1)
          sigp(2) = sigma(2)
          sigp(3) = sigma(3)
          sigp(4) = sigma(4)
          sigp(5) = sigma(5)
          sigp(6) = sigma(6)
          sigp(7) = eps(1)
          sigp(8) = eps(2)
          sigp(9) = eps(3)
          sigp(10) = 0.5*eps(4)
          sigp(11) = 0.5*eps(5)
          sigp(12) = 0.5*eps(6)

          istv = 12  ! number of projected values

c     calculated projected nodal stresses and strains (lumped projection)

          call stcn33lintet(ix,ul,shp,xsj,sigp,hr(nph),
     +              hr(nph+numnp),sg,l,ndm,nel,numnp,n)
        endif

      end if


      end

c-----------------------------------------------------------------------
c-----------------------------------------------------------------------
c
c     subroutines
c
c-----------------------------------------------------------------------
c-----------------------------------------------------------------------
```

```
c-----------------------------------------------------------------------
c     gauss integration
c-----------------------------------------------------------------------

      subroutine int3dlintet(s)
      implicit none
c
      real*8 s(4,1)
      s(1,1) = 0.25d0
      s(2,1) = 0.25d0
      s(3,1) = 0.25d0
      s(4,1) = 1.d0/6.d0

      return
      end


c-----------------------------------------------------------------------
c     shape functions
c-----------------------------------------------------------------------

      subroutine shp3dlintet(ss,xsj,shp,xl,ndm,nel)
      implicit none

      integer ndm,nel,i,j,k
      real*8 xsj,c1,c2,c3
      real*8 ss(3),shp(4,*),xl(ndm,*),xs(3,3),ad(3,3)

      call pzero(shp,16)
      shp(1,1) = -1.d0  ! d N1 / d xi
      shp(2,1) = -1.d0  ! d N1 / d eta
      shp(2,1) = -1.d0  ! d N1 / d mu
      shp(4,1) = 1.d0-ss(1)-ss(2)-ss(3)  ! N1

      shp(1,2) = 1.d0  ! d N2/ d xi
      shp(4,2) = ss(1) ! N2

      shp(2,3) = 1.d0  ! d N3/d eta
      shp(4,3) = ss(2) ! N3

      shp(3,4) = 1.d0  ! d N4/ d mu
      shp(4,4) = ss(3) ! N4

c     jacobian transformation
      do i = 1,3
        do j = 1,3
          xs(i,j) = 0.d0
          do k = 1,nel
            xs(i,j) = xs(i,j)+xl(j,k)*shp(i,k)
          end do
        end do
      end do

c     adjoint to jacobian
      ad(1,1) = xs(2,2)*xs(3,3)-xs(2,3)*xs(3,2)
```

```
      ad(1,2) = xs(1,3)*xs(3,2)-xs(1,2)*xs(3,3)
      ad(1,3) = xs(1,2)*xs(2,3)-xs(1,3)*xs(2,2)
      ad(2,1) = xs(2,3)*xs(3,1)-xs(2,1)*xs(3,3)
      ad(2,2) = xs(1,1)*xs(3,3)-xs(1,3)*xs(3,1)
      ad(2,3) = xs(1,3)*xs(2,1)-xs(1,1)*xs(2,3)
      ad(3,1) = xs(2,1)*xs(3,2)-xs(2,2)*xs(3,1)
      ad(3,2) = xs(1,2)*xs(3,1)-xs(1,1)*xs(3,2)
      ad(3,3) = xs(1,1)*xs(2,2)-xs(1,2)*xs(2,1)

c     determinant of jacobian
      xsj = xs(1,1)*ad(1,1)+xs(1,2)*ad(2,1)+xs(1,3)*ad(3,1)

c     jacobian inverse
      do i = 1,3
        do j = 1,3
          xs(i,j) = ad(i,j)/xsj
        end do
      end do

c     derivatives with repect to global coordinates

      do k = 1,nel
        c1 = xs(1,1)*shp(1,k)+xs(1,2)*shp(2,k)+xs(1,3)*shp(3,k)  ! dN/dx
        c2 = xs(2,1)*shp(1,k)+xs(2,2)*shp(2,k)+xs(2,3)*shp(3,k)  ! dN/dy
        c3 = xs(3,1)*shp(1,k)+xs(3,2)*shp(2,k)+xs(3,3)*shp(3,k)  ! dN/dz
        shp(1,k) = c1  ! store dN/dx instead of dN/d xi
        shp(2,k) = c2  ! store dN/dy instead of dN/d eta
        shp(3,k) = c3  ! store dN/dz instead of dN/d mu
      end do

      return
      end

c-----------------------------------------------------------------------
c     stress and strain projection
c-----------------------------------------------------------------------

      subroutine stcn33lintet(ix,ul,shp,xsj,sigp,dt,st,sg,l,ndm,
     +                        nel, numnp,n)
      implicit none
c
      include  'iofile.h'
      include  'strnum.h'
c
      integer   ix(*),i,j,k,l,ll,ndm,nel,numnp,n
      real*8    ul(ndm,*),dt(*),st(numnp,*),sigp(*)
      real*8    shp(4,*),sg(4,*),xsj,xg

      do j = 1,nel  ! loop over element nodes
        ll = ix(j)  ! global node number
        xg = xsj*shp(4,j)*sg(4,1)
        dt(ll) = dt(ll)+xg  ! diagonal matrix
        do i=1,istv  ! loop over projected values
          st(ll,i) = st(ll,i)+sigp(i)*xg
        end do
      end do
```

```
      return
      end
```

Quadratic tetrahedron

```
c     3d quadratic tetraheder element

      subroutine elmt04(d,ul,xl,ix,tl,s,p,ndf,ndm,nst,isw)

      implicit none

c     Include Files for own  FEAP-variables

      include 'comblk.h'
      include 'iofile.h'
      include 'eldata.h'
      include 'prstrs.h'
      include 'cdata.h'
      include 'strnum.h'


c     FEAP-variables

      integer ix(*), ndf , ndm , nst , isw
      real*8  d(*), ul(ndf,*), xl(ndm,*), tl(*), s(nst,*), p(nst)

c     own variables

      real*8 emod, nue  ! elastic modulus, poissons ratio
      real*8 sg(4,5)    ! gauss point coordinates and weights
      real*8 shp(4,10)  ! shape functions
      real*8 xsj        ! determinant of jacobian
      real*8 dd(6,6), matfac  ! material matrix, help-factor
      real*8 dvol       ! = determinant of jacobian * weight for gauss point
      real*8 bmat(6,3,10)  ! B-matrix
      real*8 eps(6)     ! strains (voigt)
      real*8 sigma(6)   ! stresses (voigt)
      real*8 sigp(12)   ! all values that are projected on the nodes,
                        ! stresses (1-6) and strains (7-12)
      integer i1,i2,i3,i4,i5,i6,j,k,lint,l


c     ----------------------------------------------------------------
c     isw=1: read and plot material data
      if(isw.eq.1) then

        call dinput(d(1),2)
        write(iow,*)'   Quadratic tetrahedron'
        write(iow,*)'   ------------------------'
        write(iow,*)'      Elastic modulus  E = ',d(1)
```

```
      write(iow,*)'      Poissons ratio   nue = ',d(2)
      write(*,*)'   Quadratic tetrahedron'
      write(*,*)'   ------------------------'
      write(*,*)'    Elastic modulus  E = ',d(1)
      write(*,*)'    Poissons ratio   nue = ',d(2)

      call pltet10(iel)

      istv = 12              ! number of projected values

     end if


c    ------------------------------------------------------------------
c    isw=3 or isw=6: calculate stiffness matrix and residual
     if((isw.eq.3).or.(isw.eq.6)) then

c    material matrix
      emod=d(1)
      nue=d(2)

      call pzero(dd,36)
      matfac = emod/((1.d0-2.d0*nue)*(1.d0+nue))
      dd(1,1) = matfac*(1.d0-nue)
      dd(2,2) = dd(1,1)
      dd(3,3) = dd(1,1)
      dd(4,4) = matfac*(1.d0-2.d0*nue)/2.d0
      dd(5,5) = dd(4,4)
      dd(6,6) = dd(4,4)
      dd(1,2) = matfac*nue
      dd(2,1) = dd(1,2)
      dd(1,3) = dd(1,2)
      dd(3,1) = dd(1,2)
      dd(2,3) = dd(1,2)
      dd(3,2) = dd(1,2)

c    gauss points and weights
      call int3dquadtet(sg)

c    loop over all gauss points
      do l = 1,5

c    shape functions for current gauss point
        call shp3dquadtet(sg(1,l),xsj,shp,xl,ndm,nel)

        dvol = sg(4,l)*xsj

c    B-matrix
        call pzero(bmat,180)
        do i1 = 1,nel
          bmat(1,1,i1) = shp(1,i1)
          bmat(2,2,i1) = shp(2,i1)
          bmat(3,3,i1) = shp(3,i1)
          bmat(4,1,i1) = shp(2,i1)
          bmat(4,2,i1) = shp(1,i1)
          bmat(5,2,i1) = shp(3,i1)
```

```
          bmat(5,3,i1) = shp(2,i1)
          bmat(6,1,i1) = shp(3,i1)
          bmat(6,3,i1) = shp(1,i1)
        end do

c     strains
        call pzero(eps,6)
        do i1 = 1,nel
          eps(1) = eps(1) + bmat(1,1,i1) * ul(1,i1)  ! eps_11
          eps(2) = eps(2) + bmat(2,2,i1) * ul(2,i1)  ! eps_22
          eps(3) = eps(3) + bmat(3,3,i1) * ul(3,i1)  ! eps_33
          eps(4) = eps(4) + bmat(4,1,i1) * ul(1,i1)
     +                    + bmat(4,2,i1) * ul(2,i1)  ! 2*eps_12
          eps(5) = eps(5) + bmat(5,2,i1) * ul(2,i1)
     +                    + bmat(5,3,i1) * ul(3,i1)  ! 2*eps_23
          eps(6) = eps(6) + bmat(6,1,i1) * ul(1,i1)
     +                    + bmat(6,3,i1) * ul(3,i1)  ! 2*eps_31
        end do

c     stresses
        call pzero(sigma,6)
        do i1 = 1,6
          do i2 = 1,6
            sigma(i1) = sigma(i1) + dd(i1,i2)*eps(i2)
          end do
        end do

c     residual (negative)
        do i1=1,3
          do i2=1,10
            do i3=1,6
      p(i1+3*(i2-1)) = p(i1+3*(i2-1)) - sigma(i3)*bmat(i3,i1,i2)*dvol
            end do
          end do
        end do

c     stiffness matrix
        do i1=1,10
          do i2=1,3
            do i3=1,10
              do i4=1,3
                do i5=1,6
                  do i6=1,6
      s(i2+3*(i1-1),i4+3*(i3-1))=s(i2+3*(i1-1),i4+3*(i3-1))
     +       +bmat(i5,i2,i1)*dd(i5,i6)*bmat(i6,i4,i3)*dvol
                  end do
                end do
              end do
            end do
          end do
        end do

       end do  ! end loop over all gauss points

      end if
```

```
c     ----------------------------------------------------------------

c     isw=4 or isw=8: calculate strains, stresses and projected nodal stresses
      if((isw.eq.4).or.(isw.eq.8)) then

c     material matrix
       emod=d(1)
       nue=d(2)

       call pzero(dd,36)
       matfac = emod/((1.d0-2.d0*nue)*(1.d0+nue))
       dd(1,1) = matfac*(1.d0-nue)
       dd(2,2) = dd(1,1)
       dd(3,3) = dd(1,1)
       dd(4,4) = matfac*(1.d0-2.d0*nue)/2.d0
       dd(5,5) = dd(4,4)
       dd(6,6) = dd(4,4)
       dd(1,2) = matfac*nue
       dd(2,1) = dd(1,2)
       dd(1,3) = dd(1,2)
       dd(3,1) = dd(1,2)
       dd(2,3) = dd(1,2)
       dd(3,2) = dd(1,2)

c     gauss points and weights
       call int3dquadtet(sg)

c     loop over all gauss points
       do l = 1,5

c     shape functions for current gauss point
         call shp3dquadtet(sg(1,l),xsj,shp,xl,ndm,nel)

         dvol = sg(4,l)*xsj

c     B-matrix
         call pzero(bmat,180)
         do i1 = 1,nel
           bmat(1,1,i1) = shp(1,i1)
           bmat(2,2,i1) = shp(2,i1)
           bmat(3,3,i1) = shp(3,i1)
           bmat(4,1,i1) = shp(2,i1)
           bmat(4,2,i1) = shp(1,i1)
           bmat(5,2,i1) = shp(3,i1)
           bmat(5,3,i1) = shp(2,i1)
           bmat(6,1,i1) = shp(3,i1)
           bmat(6,3,i1) = shp(1,i1)
         end do

c     strains
         call pzero(eps,6)
         do i1 = 1,nel
           eps(1) = eps(1) + bmat(1,1,i1) * ul(1,i1)  ! eps_11
           eps(2) = eps(2) + bmat(2,2,i1) * ul(2,i1)  ! eps_22
           eps(3) = eps(3) + bmat(3,3,i1) * ul(3,i1)  ! eps_33
           eps(4) = eps(4) + bmat(4,1,i1) * ul(1,i1)
```

```
     +                  + bmat(4,2,i1) * ul(2,i1)  ! 2*eps_12
          eps(5) = eps(5) + bmat(5,2,i1) * ul(2,i1)
     +                  + bmat(5,3,i1) * ul(3,i1)  ! 2*eps_23
          eps(6) = eps(6) + bmat(6,1,i1) * ul(1,i1)
     +                  + bmat(6,3,i1) * ul(3,i1)  ! 2*eps_31
        end do

c     stresses
        call pzero(sigma,6)
        do i1 = 1,6
          do i2 = 1,6
            sigma(i1) = sigma(i1) + dd(i1,i2)*eps(i2)
          end do
        end do

        if(isw.ne.8) then
          write(iow,*)'  sigma_11', sigma(1)
          write(iow,*)'  sigma_22', sigma(2)
          write(iow,*)'  sigma_33', sigma(3)
          write(iow,*)'  sigma_12', sigma(4)
          write(iow,*)'  sigma_23', sigma(5)
          write(iow,*)'  sigma_31', sigma(6)
        else
          call pzero(sigp,12)
          sigp(1) = sigma(1)
          sigp(2) = sigma(2)
          sigp(3) = sigma(3)
          sigp(4) = sigma(4)
          sigp(5) = sigma(5)
          sigp(6) = sigma(6)
          sigp(7) = eps(1)
          sigp(8) = eps(2)
          sigp(9) = eps(3)
          sigp(10) = 0.5*eps(4)
          sigp(11) = 0.5*eps(5)
          sigp(12) = 0.5*eps(6)

c     calculated projected nodal stresses and strains (lumped projection)

          call stcn33quadtet(ix,ul,shp,xsj,sigp,hr(nph),
     +         hr(nph+numnp),sg,l,ndm,nel,numnp,n)
        endif

       end do  ! end loop over all gauss points

      end if


      end

c-----------------------------------------------------------------------
c-----------------------------------------------------------------------
c
c     subroutines
c
c-----------------------------------------------------------------------
```

```
c-----------------------------------------------------------------------


c-----------------------------------------------------------------------
c     gauss integration
c-----------------------------------------------------------------------

      subroutine int3dquadtet(s)
      implicit none
c
      real*8 s(4,*)

      s(1,1) = 0.25d0
      s(2,1) = 0.25d0
      s(3,1) = 0.25d0
      s(4,1) = -2.d0/15.d0

      s(1,2) = 1.d0/6.d0
      s(2,2) = 1.d0/6.d0
      s(3,2) = 1.d0/6.d0
      s(4,2) = 3.d0/40.d0

      s(1,3) = 1.d0/6.d0
      s(2,3) = 1.d0/6.d0
      s(3,3) = 0.5d0
      s(4,3) = 3.d0/40.d0

      s(1,4) = 1.d0/6.d0
      s(2,4) = 0.5d0
      s(3,4) = 1.d0/6.d0
      s(4,4) = 3.d0/40.d0

      s(1,5) = 0.5d0
      s(2,5) = 1.d0/6.d0
      s(3,5) = 1.d0/6.d0
      s(4,5) = 3.d0/40.d0

      return
      end


      subroutine int3dquadtet2(s)
      implicit none
c
      real*8 s(4,*)
      real*8 alpha, beta

      alpha = (5.d0+3.d0*dsqrt(5.d0))/20.d0
      beta = (5.d0-dsqrt(5.d0))/20.d0

      s(1,1) = beta
      s(2,1) = beta
      s(3,1) = beta
      s(4,1) = 1.d0/24.d0

      s(1,2) = alpha
```

```
      s(2,2) = beta
      s(3,2) = beta
      s(4,2) = 1.d0/24.d0

      s(1,3) = beta
      s(2,3) = alpha
      s(3,3) = beta
      s(4,3) = 1.d0/24.d0

      s(1,4) = beta
      s(2,4) = beta
      s(3,4) = alpha
      s(4,4) = 1.d0/24.d0

      return
      end


c-----------------------------------------------------------------------
c     shape functions
c-----------------------------------------------------------------------

      subroutine shp3dquadtet(ss,xsj,shp,xl,ndm,nel)
      implicit none

      integer ndm,nel,i,j,k
      real*8 xsj,c1,c2,c3, lambda
      real*8 ss(3),shp(4,*),xl(ndm,*),xs(3,3),ad(3,3)


      call pzero(shp,40)
      lambda = 1.d0 - ss(1) - ss(2) - ss(3)

      shp(1,1) = 1.d0 - 4.d0*lambda  ! d N1 / d xi
      shp(2,1) = 1.d0 - 4.d0*lambda  ! d N1 / d eta
      shp(3,1) = 1.d0 - 4.d0*lambda  ! d N1 / d mu
      shp(4,1) = lambda*(2.d0*lambda-1.0)  ! N1

      shp(1,2) = 4.d0*ss(1)-1.d0  ! d N2 / d xi
      shp(4,2) = ss(1)*(2.d0*ss(1)-1.d0) ! N2

      shp(2,3) = 4.d0*ss(2)-1.d0  ! d N3 / d eta
      shp(4,3) = ss(2)*(2.d0*ss(2)-1.d0) ! N3

      shp(3,4) = 4.d0*ss(3)-1.d0  ! d N4 / d mu
      shp(4,4) = ss(3)*(2.d0*ss(3)-1.d0) ! N4

      shp(1,5) = 4.d0*lambda-4.d0*ss(1)  ! d N5 / d xi
      shp(2,5) = -4.d0*ss(1)  ! d N5 / d eta
      shp(3,5) = -4.d0*ss(1)  ! d N5 / d mu
      shp(4,5) = 4.d0*ss(1)*lambda  ! N5

      shp(1,6) = 4.d0*ss(2)  ! d N6 / d xi
      shp(2,6) = 4.d0*ss(1)  ! d N6 / d eta
      shp(4,6) = 4.d0*ss(1)*ss(2)  ! N6
```

```
      shp(1,7) = -4.d0*ss(2)  ! d N7 / d xi
      shp(2,7) = 4.d0*lambda-4.d0*ss(2)  ! d N7 / d eta
      shp(3,7) = -4.d0*ss(2)  ! d N7 / d mu
      shp(4,7) = 4.d0*ss(2)*lambda  ! N7

      shp(1,8) = -4.d0*ss(3)  ! d N8 / d xi
      shp(2,8) = -4.d0*ss(3)  ! d N8 / d eta
      shp(3,8) = 4.d0*lambda-4.d0*ss(3)  ! d N8 / d mu
      shp(4,8) = 4.d0*ss(3)*lambda  ! N8

      shp(1,9) = 4.d0*ss(3)  ! d N9 / d xi
      shp(3,9) = 4.d0*ss(1)  ! d N9 / d mu
      shp(4,9) = 4.d0*ss(1)*ss(3)  ! N9

      shp(2,10) = 4.d0*ss(3)  ! d N10 / d eta
      shp(3,10) = 4.d0*ss(2)  ! d N10 / d mu
      shp(4,10) = 4.d0*ss(2)*ss(3)  ! N10

c     jacobian transformation
      do i = 1,3
       do j = 1,3
        xs(i,j) = 0.d0
        do k = 1,nel
         xs(i,j) = xs(i,j)+xl(j,k)*shp(i,k)
        end do
       end do
      end do

c     adjoint to jacobian
      ad(1,1) = xs(2,2)*xs(3,3)-xs(2,3)*xs(3,2)
      ad(1,2) = xs(1,3)*xs(3,2)-xs(1,2)*xs(3,3)
      ad(1,3) = xs(1,2)*xs(2,3)-xs(1,3)*xs(2,2)
      ad(2,1) = xs(2,3)*xs(3,1)-xs(2,1)*xs(3,3)
      ad(2,2) = xs(1,1)*xs(3,3)-xs(1,3)*xs(3,1)
      ad(2,3) = xs(1,3)*xs(2,1)-xs(1,1)*xs(2,3)
      ad(3,1) = xs(2,1)*xs(3,2)-xs(2,2)*xs(3,1)
      ad(3,2) = xs(1,2)*xs(3,1)-xs(1,1)*xs(3,2)
      ad(3,3) = xs(1,1)*xs(2,2)-xs(1,2)*xs(2,1)

c     determinant of jacobian
      xsj = xs(1,1)*ad(1,1)+xs(1,2)*ad(2,1)+xs(1,3)*ad(3,1)

c     jacobian inverse
      do i = 1,3
       do j = 1,3
        xs(i,j) = ad(i,j)/xsj
       end do
      end do

c     derivatives with repect to global coordinates

      do k = 1,nel
       c1 = xs(1,1)*shp(1,k)+xs(1,2)*shp(2,k)+xs(1,3)*shp(3,k)  ! dN/dx
       c2 = xs(2,1)*shp(1,k)+xs(2,2)*shp(2,k)+xs(2,3)*shp(3,k)  ! dN/dy
       c3 = xs(3,1)*shp(1,k)+xs(3,2)*shp(2,k)+xs(3,3)*shp(3,k)  ! dN/dz
       shp(1,k) = c1  ! store dN/dx instead of dN/d xi
```

```
        shp(2,k) = c2  ! store dN/dy instead of dN/d eta
        shp(3,k) = c3  ! store dN/dz instead of dN/d mu
      end do

      return
      end

c-----------------------------------------------------------------------
c     stress and strain projection
c-----------------------------------------------------------------------

      subroutine stcn33quadtet(ix,ul,shp,xsj,sigp,dt,st,sg,l,ndm,
     +                         nel, numnp,n)
      implicit none
c
      include  'iofile.h'
      include  'strnum.h'
c
      integer   ix(*),i,j,k,l,ll,ndm,nel,numnp,n
      real*8    ul(ndm,*),dt(*),st(numnp,*),sigp(*)
      real*8    shp(4,*),sg(4,*),xsj,xg

      do j = 1,nel  ! loop over element nodes
        ll = ix(j)  ! global node number
        xg = xsj*shp(4,j)*sg(4,1)
        dt(ll) = dt(ll)+xg  ! diagonal matrix
        do i=1,istv  ! loop over projected values
          st(ll,i) = st(ll,i)+sigp(i)*xg
        end do
      end do

      return
      end
```

Patch test

Because of similarities, only the patch test implementation for the linear brick is shown below:

```
feap
,,,3,3,8

bloc
cart,1,1,1,,,,10
1,   0,  0,  0
2,   6,  0,  0
3,   4,  6,  0
4,   0,  6,  0
5,   0,  0,  4
6,   5,  0,  5
7,   5,  6,  6
```

8, 0, 4, 5

bloc
cart,1,1,1,,,,10
1, 6, 0, 0
2, 10, 0, 0
3, 10, 4, 0
4, 4, 6, 0
5, 5, 0, 5
6, 10, 0, 6
7, 10, 5, 5
8, 5, 6, 6

bloc
cart,1,1,1,,,,10
1, 0, 6, 0
2, 4, 6, 0
3, 5, 10, 0
4, 0, 10, 0
5, 0, 4, 5
6, 5, 6, 6
7, 4, 10, 4
8, 0, 10, 6

bloc
cart,1,1,1,,,,10
1, 4, 6, 0
2, 10, 4, 0
3, 10, 10, 0
4, 5, 10, 0
5, 5, 6, 6
6, 10, 5, 5
7, 10, 10, 5
8, 4, 10, 4

bloc
cart,1,1,1,,,,10
1, 0, 0, 4
2, 5, 0, 5
3, 5, 6, 6
4, 0, 4, 5
5, 0, 0, 10
6, 4, 0, 10
7, 6, 5, 10
8, 0, 5, 10

bloc
cart,1,1,1,,,,10
1, 5, 0, 5
2, 10, 0, 6
3, 10, 5, 5
4, 5, 6, 6
5, 4, 0, 10
6, 10, 0, 10
7, 10, 4, 10
8, 6, 5, 10

```
bloc
cart,1,1,1,,,,10
1,   0,  4,  5
2,   5,  6,  6
3,   4, 10,  4
4,   0, 10,  6
5,   0,  5, 10
6,   6,  5, 10
7,   5, 10, 10
8,   0, 10, 10

bloc
cart,1,1,1,,,,10
1,   5,  6,  6
2,  10,  5,  5
3,  10, 10,  5
4,   4, 10,  4
5,   6,  5, 10
6,  10,  4, 10
7,  10, 10, 10
8,   5, 10, 10

! boundary conditions for DPT
!eboun
!1  0.0 1 1 1
!1 10.0 1 1 1
!2  0.0 1 1 1
!2 10.0 1 1 1
!3  0.0 1 1 1
!3 10.0 1 1 1

! 1. r-mode (ux = 1, uy = 0, uz = 0)
!cdisp
!node  0  0  0 1 0 0
!node  6  0  0 1 0 0
!node 10  0  0 1 0 0
!node  0  6  0 1 0 0
!node  4  6  0 1 0 0
!node 10  4  0 1 0 0
!node  0 10  0 1 0 0
!node  5 10  0 1 0 0
!node 10 10  0 1 0 0
!node  0  0  4 1 0 0
!node  5  0  5 1 0 0
!node 10  0  6 1 0 0
!node  0  4  5 1 0 0
!node 10  5  5 1 0 0
!node  0 10  6 1 0 0
!node  4 10  4 1 0 0
!node 10 10  5 1 0 0
!node  0  0 10 1 0 0
!node  4  0 10 1 0 0
!node 10  0 10 1 0 0
!node  0  5 10 1 0 0
!node  6  5 10 1 0 0
```

```
!node 10  4 10 1 0 0
!node  0 10 10 1 0 0
!node  5 10 10 1 0 0
!node 10 10 10 1 0 0

! 2. r-mode (ux = 0, uy = 1, uz = 0)
!cdisp
!node  0  0  0 0 1 0
!node  6  0  0 0 1 0
!node 10  0  0 0 1 0
!node  0  6  0 0 1 0
!node  4  6  0 0 1 0
!node 10  4  0 0 1 0
!node  0 10  0 0 1 0
!node  5 10  0 0 1 0
!node 10 10  0 0 1 0
!node  0  0  4 0 1 0
!node  5  0  5 0 1 0
!node 10  0  6 0 1 0
!node  0  4  5 0 1 0
!node 10  5  5 0 1 0
!node  0 10  6 0 1 0
!node  4 10  4 0 1 0
!node 10 10  5 0 1 0
!node  0  0 10 0 1 0
!node  4  0 10 0 1 0
!node 10  0 10 0 1 0
!node  0  5 10 0 1 0
!node  6  5 10 0 1 0
!node 10  4 10 0 1 0
!node  0 10 10 0 1 0
!node  5 10 10 0 1 0
!node 10 10 10 0 1 0

! 3. r-mode (ux = 0, uy = 0, uz = 1)
!cdisp
!node  0  0  0 0 0  1
!node  6  0  0 0 0  1
!node 10  0  0 0 0  1
!node  0  6  0 0 0  1
!node  4  6  0 0 0  1
!node 10  4  0 0 0  1
!node  0 10  0 0 0  1
!node  5 10  0 0 0  1
!node 10 10  0 0 0  1
!node  0  0  4 0 0  1
!node  5  0  5 0 0  1
!node 10  0  6 0 0  1
!node  0  4  5 0 0  1
!node 10  5  5 0 0  1
!node  0 10  6 0 0  1
!node  4 10  4 0 0  1
!node 10 10  5 0 0  1
!node  0  0 10 0 0  1
!node  4  0 10 0 0  1
!node 10  0 10 0 0  1
```

```
!node  0  5 10 0 0  1
!node  6  5 10 0 0  1
!node 10  4 10 0 0  1
!node  0 10 10 0 0  1
!node  5 10 10 0 0  1
!node 10 10 10 0 0  1

! 4. r-mode (ux = -y, uy = x, uz = 0)
!cdisp
!node  0  0  0   -0  0 0
!node  6  0  0   -0  6 0
!node 10  0  0   -0 10 0
!node  0  6  0   -6  0 0
!node  4  6  0   -6  4 0
!node 10  4  0   -4 10 0
!node  0 10  0  -10  0 0
!node  5 10  0  -10  5 0
!node 10 10  0  -10 10 0
!node  0  0  4   -0  0 0
!node  5  0  5   -0  5 0
!node 10  0  6   -0 10 0
!node  0  4  5   -4  0 0
!node 10  5  5   -5 10 0
!node  0 10  6  -10  0 0
!node  4 10  4  -10  4 0
!node 10 10  5  -10 10 0
!node  0  0 10   -0  0 0
!node  4  0 10   -0  4 0
!node 10  0 10   -0 10 0
!node  0  5 10   -5  0 0
!node  6  5 10   -5  6 0
!node 10  4 10   -4 10 0
!node  0 10 10  -10  0 0
!node  5 10 10  -10  5 0
!node 10 10 10  -10 10 0

! 5. r-mode (ux = 0, uy = -z, uz = y)
!cdisp
!node  0  0  0 0  -0  0
!node  6  0  0 0  -0  0
!node 10  0  0 0  -0  0
!node  0  6  0 0  -0  6
!node  4  6  0 0  -0  6
!node 10  4  0 0  -0  4
!node  0 10  0 0  -0 10
!node  5 10  0 0  -0 10
!node 10 10  0 0  -0 10
!node  0  0  4 0  -4  0
!node  5  0  5 0  -5  0
!node 10  0  6 0  -6  0
!node  0  4  5 0  -5  4
!node 10  5  5 0  -5  5
!node  0 10  6 0  -6 10
!node  4 10  4 0  -4 10
!node 10 10  5 0  -5 10
!node  0  0 10 0 -10  0
```

```
!node  4  0 10 0 -10  0
!node 10  0 10 0 -10  0
!node  0  5 10 0 -10  5
!node  6  5 10 0 -10  5
!node 10  4 10 0 -10  4
!node  0 10 10 0 -10 10
!node  5 10 10 0 -10 10
!node 10 10 10 0 -10 10

! 6. r-mode (ux = z, uy = 0, uz = -x)
!cdisp
!node  0  0  0   0 0   -0
!node  6  0  0   0 0   -6
!node 10  0  0   0 0  -10
!node  0  6  0   0 0   -0
!node  4  6  0   0 0   -4
!node 10  4  0   0 0  -10
!node  0 10  0   0 0   -0
!node  5 10  0   0 0   -5
!node 10 10  0   0 0  -10
!node  0  0  4   4 0   -0
!node  5  0  5   5 0   -5
!node 10  0  6   6 0  -10
!node  0  4  5   5 0   -0
!node 10  5  5   5 0  -10
!node  0 10  6   6 0   -0
!node  4 10  4   4 0   -4
!node 10 10  5   5 0  -10
!node  0  0 10  10 0   -0
!node  4  0 10  10 0   -4
!node 10  0 10  10 0  -10
!node  0  5 10  10 0   -0
!node  6  5 10  10 0   -6
!node 10  4 10  10 0  -10
!node  0 10 10  10 0   -0
!node  5 10 10  10 0   -5
!node 10 10 10  10 0  -10

! 1. e-mode (ux = x, uy = 0, uz = 0)
!cdisp
!node  0  0  0   0  0  0
!node  6  0  0   6  0  0
!node 10  0  0  10  0  0
!node  0  6  0   0  0  0
!node  4  6  0   4  0  0
!node 10  4  0  10  0  0
!node  0 10  0   0  0  0
!node  5 10  0   5  0  0
!node 10 10  0  10  0  0
!node  0  0  4   0  0  0
!node  5  0  5   5  0  0
!node 10  0  6  10  0  0
!node  0  4  5   0  0  0
!node 10  5  5  10  0  0
!node  0 10  6   0  0  0
!node  4 10  4   4  0  0
```

```
!node 10 10  5  10  0  0
!node  0  0 10   0  0  0
!node  4  0 10   4  0  0
!node 10  0 10  10  0  0
!node  0  5 10   0  0  0
!node  6  5 10   6  0  0
!node 10  4 10  10  0  0
!node  0 10 10   0  0  0
!node  5 10 10   5  0  0
!node 10 10 10  10  0  0

! 2. e-mode (ux = 0, uy = y, uz = 0)
!cdisp
!node  0  0  0    0  0  0
!node  6  0  0    0  0  0
!node 10  0  0    0  0  0
!node  0  6  0    0  6  0
!node  4  6  0    0  6  0
!node 10  4  0    0  4  0
!node  0 10  0    0 10  0
!node  5 10  0    0 10  0
!node 10 10  0    0 10  0
!node  0  0  4    0  0  0
!node  5  0  5    0  0  0
!node 10  0  6    0  0  0
!node  0  4  5    0  4  0
!node 10  5  5    0  5  0
!node  0 10  6    0 10  0
!node  4 10  4    0 10  0
!node 10 10  5    0 10  0
!node  0  0 10    0  0  0
!node  4  0 10    0  0  0
!node 10  0 10    0  0  0
!node  0  5 10    0  5  0
!node  6  5 10    0  5  0
!node 10  4 10    0  4  0
!node  0 10 10    0 10  0
!node  5 10 10    0 10  0
!node 10 10 10    0 10  0

! 3. e-mode (ux = 0, uy = 0, uz = z)
!cdisp
!node  0  0  0   0 0  0
!node  6  0  0   0 0  0
!node 10  0  0   0 0  0
!node  0  6  0   0 0  0
!node  4  6  0   0 0  0
!node 10  4  0   0 0  0
!node  0 10  0   0 0  0
!node  5 10  0   0 0  0
!node 10 10  0   0 0  0
!node  0  0  4   0 0  4
!node  5  0  5   0 0  5
!node 10  0  6   0 0  6
!node  0  4  5   0 0  5
!node 10  5  5   0 0  5
```

```
!node  0 10  6   0 0  6
!node  4 10  4   0 0  4
!node 10 10  5   0 0  5
!node  0  0 10   0 0 10
!node  4  0 10   0 0 10
!node 10  0 10   0 0 10
!node  0  5 10   0 0 10
!node  6  5 10   0 0 10
!node 10  4 10   0 0 10
!node  0 10 10   0 0 10
!node  5 10 10   0 0 10
!node 10 10 10   0 0 10

! 4. e-mode (ux = y, uy = x, uz = 0)
!cdisp
!node  0  0  0  0  0  0
!node  6  0  0  0  6  0
!node 10  0  0  0 10  0
!node  0  6  0  6  0  0
!node  4  6  0  6  4  0
!node 10  4  0  4 10  0
!node  0 10  0 10  0  0
!node  5 10  0 10  5  0
!node 10 10  0 10 10  0
!node  0  0  4  0  0  0
!node  5  0  5  0  5  0
!node 10  0  6  0 10  0
!node  0  4  5  4  0  0
!node 10  5  5  5 10  0
!node  0 10  6 10  0  0
!node  4 10  4 10  4  0
!node 10 10  5 10 10  0
!node  0  0 10  0  0  0
!node  4  0 10  0  4  0
!node 10  0 10  0 10  0
!node  0  5 10  5  0  0
!node  6  5 10  5  6  0
!node 10  4 10  4 10  0
!node  0 10 10 10  0  0
!node  5 10 10 10  5  0
!node 10 10 10 10 10  0

! 5. e-mode (ux = 0, uy = z, uz = y)
!cdisp
!node  0  0  0  0   0  0
!node  6  0  0  0   0  0
!node 10  0  0  0   0  0
!node  0  6  0  0   0  6
!node  4  6  0  0   0  6
!node 10  4  0  0   0  4
!node  0 10  0  0   0 10
!node  5 10  0  0   0 10
!node 10 10  0  0   0 10
!node  0  0  4  0   4  0
!node  5  0  5  0   5  0
!node 10  0  6  0   6  0
```

```
!node  0  4  5  0   5  4
!node 10  5  5  0   5  5
!node  0 10  6  0   6 10
!node  4 10  4  0   4 10
!node 10 10  5  0   5 10
!node  0  0 10  0  10  0
!node  4  0 10  0  10  0
!node 10  0 10  0  10  0
!node  0  5 10  0  10  5
!node  6  5 10  0  10  5
!node 10  4 10  0  10  4
!node  0 10 10  0  10 10
!node  5 10 10  0  10 10
!node 10 10 10  0  10 10

! 6. e-mode (ux = z, uy = 0, uz = x)
!cdisp
!node  0  0  0  0  0  0
!node  6  0  0  0  0  6
!node 10  0  0  0  0 10
!node  0  6  0  0  0  0
!node  4  6  0  0  0  4
!node 10  4  0  0  0 10
!node  0 10  0  0  0  0
!node  5 10  0  0  0  5
!node 10 10  0  0  0 10
!node  0  0  4  4  0  0
!node  5  0  5  5  0  5
!node 10  0  6  6  0 10
!node  0  4  5  5  0  0
!node 10  5  5  5  0 10
!node  0 10  6  6  0  0
!node  4 10  4  4  0  4
!node 10 10  5  5  0 10
!node  0  0 10 10  0  0
!node  4  0 10 10  0  4
!node 10  0 10 10  0 10
!node  0  5 10 10  0  0
!node  6  5 10 10  0  6
!node 10  4 10 10  0 10
!node  0 10 10 10  0  0
!node  5 10 10 10  0  5
!node 10 10 10 10  0 10


! boundary conditions for FPT
!eboun
!1 0.0 1 0 0
!2 0.0 0 1 0
!3 0.0 0 0 1


! 1. FPT, tension in x-direction
!csur
!surface
!1, 10.0  0.0  0.0 10.0
```

```
!2, 10.0 10.0  0.0 10.0
!3, 10.0 10.0 10.0 10.0
!4, 10.0  0.0 10.0 10.0

! help tension in x-direction
!csur
!surface
!1, 0.0  0.0  0.0 10.0
!2, 0.0  0.0 10.0 10.0
!3, 0.0 10.0 10.0 10.0
!4, 0.0 10.0  0.0 10.0

! 2. FPT, tension in y-direction
!csur
!surface
!1,  0.0 10.0  0.0 10.0
!2,  0.0 10.0 10.0 10.0
!3, 10.0 10.0 10.0 10.0
!4, 10.0 10.0  0.0 10.0

! help tension in y-direction
!csur
!surface
!1,  0.0  0.0  0.0 10.0
!2, 10.0  0.0  0.0 10.0
!3, 10.0  0.0 10.0 10.0
!4,  0.0  0.0 10.0 10.0

! 3. FPT, tension in z-direction
!csur
!surface
!1,  0.0  0.0 10.0 10.0
!2, 10.0  0.0 10.0 10.0
!3, 10.0 10.0 10.0 10.0
!4,  0.0 10.0 10.0 10.0

! help tension in z-direction
!csur
!surface
!1,  0.0  0.0  0.0 10.0
!2,  0.0 10.0  0.0 10.0
!3, 10.0 10.0  0.0 10.0
!4, 10.0  0.0  0.0 10.0

! boundary conditions for xy-shear tests
cboun
node  0.0  0.0  0.0 1 1 1
node 10.0  0.0  0.0 0 1 1
node  0.0 10.0  0.0 0 0 1

! 4. FPT, shear on x- and y-plane
forc
10 0  -6.1667E+01 6.1667E+01
12 0  0          1.2917E+02
14 0  -1.2500E+02 1.1667E+02
16 0  0          2.5000E+02
```

```
28 0  6.2500E+01 7.0833E+01
32 0  1.4167E+02 1.3333E+02
46 0  -6.1667E+01 4.6667E+01
48 0  0          1.2083E+02
64 0  7.0833E+01 7.0833E+01
19 0  6.1667E+01 -6.6667E+01
20 0  1.1167E+02
23 0  1.0833E+02 -1.3750E+02
24 0  2.5000E+02
55 0  5.5000E+01 -5.7500E+01
56 0  1.3833E+02
1  0  -6.1667E+01 -5.8333E+01
3  0  0          -1.2667E+02
5  0  -1.2500E+02 -1.1250E+02
7  0  0          -2.5167E+02
37 0  -6.1667E+01 -6.5833E+01
39 0  0          -1.2333E+02
 2 0 -1.2500E+02
 6 0 -2.5333E+02
38 0 -1.2500E+02


! boundary conditions for yz-shear tests
!cboun
!node  0.0  0.0  0.0 1 1 1
!node  0.0 10.0  0.0 1 0 0
!node  0.0  0.0 10.0 1 1 0

! 5. FPT, shear on y- and z-plane
!forc
!19 0  0  -4.6667E+01 6.1667E+01
!20 0  0  -1.1000E+02 1.1167E+02
!23 0  0  0           1.0833E+02
!24 0  0  0           2.5000E+02
!28 0  0  -7.1667E+01 6.2500E+01
!32 0  0  0           1.4167E+02
!55 0  0  6.6667E+01  5.5000E+01
!56 0  0  1.2833E+02  1.3833E+02
!64 0  0  6.5833E+01  7.0833E+01
!37 0  0  5.8333E+01 -6.1667E+01
!38 0  0  1.2167E+02 -1.2500E+02
!39 0  0  1.3750E+02
!40 0  0  2.5083E+02
!46 0  0  5.8333E+01 -6.1667E+01
!48 0  0  1.1250E+02
! 1 0  0 -8.0000E+01 -6.1667E+01
! 2 0  0 -1.4000E+02  -1.2500E+02
! 5 0  0 0            -1.2500E+02
! 6 0  0 0            -2.5333E+02
!10 0  0 -5.3333E+01  -6.1667E+01
!14 0  0 0            -1.2500E+02
! 3 0  0 -1.1333E+02
! 4 0  0 -2.4833E+02
!12 0  0 -1.3667E+02
```

```
! boundary conditions for xz-shear tests
!cboun
!node  0.0  0.0  0.0 1 1 1
!node 10.0  0.0  0.0 0 1 0
!node  0.0  0.0 10.0 1 1 0

! 6. FPT, shear on x- and z-plane
!forc
!10 0  -5.3333E+01 0 6.1667E+01
!12 0  -1.3667E+02 0 1.2917E+02
!14 0  0           0 1.1667E+02
!16 0  0           0 2.5000E+02
!28 0  -7.1667E+01 0 7.0833E+01
!32 0  0           0 1.3333E+02
!46 0  5.8333E+01  0 4.6667E+01
!48 0  1.1250E+02  0 1.2083E+02
!64 0  6.5833E+01  0 7.0833E+01
!37 0  5.8333E+01  0 -6.5833E+01
!38 0  1.2167E+02
!39 0  1.3750E+02  0 -1.2333E+02
!40 0  2.5083E+02
!55 0  6.6667E+01  0 -5.7500E+01
!56 0  1.2833E+02
! 1 0  -8.0000E+01 0 -5.8333E+01
! 3 0  -1.1333E+02 0 -1.2667E+02
! 5 0  0           0 -1.1250E+02
! 7 0  0           0 -2.5167E+02
!19 0  -4.6667E+01 0 -6.6667E+01
!23 0  0           0 -1.3750E+02
! 2 0  -1.4000E+02
! 4 0  -2.4833E+02
!20 0  -1.1000E+02


mate,1
user,1
210000.0, 0.3

end
tie
inte
stop
```

Input Files for Performance Tests

Beam

```
feap
,,,3,3,8  ! change
```

```
bloc
cart,2,12,100,,,,10 ! change
1   0  0  -8
2   0  1  -8
3   0  1  -2
4   0  0  -2
5  50  0  -8
6  50  1  -8
7  50  1  -2
8  50  0  -2

bloc
cart,6,4,100,,,,10 ! change
1   0  0  -2
2   0  3  -2
3   0  3   0
4   0  0   0
5  50  0  -2
6  50  3  -2
7  50  3   0
8  50  0   0


eboun
1  0.0 1 1 1
2  0.0 0 1 0
1 50.0 0 0 1

edisp
1 50.0 0 0 -0.5

!csur
!surf
!1,  50 0 -2 1
!2,  50 3 -2 1
!3,  50 3  0 1
!4,  50 0  0 1

!csur
!surf
!1, 50 0 -8 1
!2, 50 1 -8 1
!3, 50 1 -2 1
!4, 50 0 -2 1


mate,1
user,1              ! change
210000.0, 0.3

end
tie
opti
inte
stop
```

Distortion

```
feap
,,,3,3,8

para
a1=10.0
a2=2.0
a3=2.0
a=3.0
ex=8
ey=4
ez=8
p=30.0

nopr

bloc
cart,ey,ez,ex,,,,10
1   0.0d0        0.d0  -0.5*a3
2   0.0d0      0.5*a2  -0.5*a3
3   0.0d0      0.5*a2   0.5*a3
4   0.0d0        0.d0   0.5*a3
5   0.5*a1-a     0.d0  -0.5*a3
6   0.5*a1-a   0.5*a2  -0.5*a3
7   0.5*a1+a   0.5*a2   0.5*a3
8   0.5*a1+a     0.d0   0.5*a3

block
cart,ey,ez,ex,,,,10
1   0.5*a1-a     0.d0  -0.5*a3
2   0.5*a1-a   0.5*a2  -0.5*a3
3   0.5*a1+a   0.5*a2   0.5*a3
4   0.5*a1+a     0.d0   0.5*a3
5   a1           0.d0  -0.5*a3
6   a1         0.5*a2  -0.5*a3
7   a1         0.5*a2   0.5*a3
8   a1           0.d0   0.5*a3
```

```
eboun,add
1  0.d0  1,0,1
2  0.d0  0,1,0
1  a1    0,0,1


edisp
1 a1 0 0 0.1


mate,1
user,1
210000.0, 0.3

end
tie
inte
stop
```

Block

```
feap
,,,3,3,8


bloc
cart,10,10,20,,,,10
1   0  0   0
2   5  0   0
3   5  5   0
4   0  5   0
5   0  0  10
6   5  0  10
7   5  5  10
8   0  5  10


eboun
1  0.0 1 0 0
```

2 0.0 0 1 0
3 0.0 0 0 1

csur
surf
1, 0 0 10 -200
2, 1 0 10 -200
3, 1 1 10 -200
4, 0 1 10 -200

mate,1
user,1
210000.0, 0.4999

end
tie
opti
inte
stop