```
(* This Mathematica script generates element matrices and exports *)
(* them as fortran code for the finite element described in *)
(* https://github.com/Ra-Na/C3D4C1-Abaqus-UEL *)

(* Clear cache *)
Remove["Global`*"]
deg = 3;

(* Define a complete 3D 3rd order polynomial *)
phi[xvek_, p_] := (
   monoms = {};
   For[iii = 0, iii <= deg, iii++,
    For[jjj = 0, jjj <= deg, jjj++,
      For[kkk = 0, kkk <= deg, kkk++,
        If[iii + jjj + kkk ≤ deg,
         monoms = Append[monoms, xvek[[1]] ^ iii * xvek[[2]] ^ jjj * xvek[[3]] ^ kkk]
        ];];];
   p.monoms);

(* Create vector of polynomial coefficients *)
nparams = Binomial[3 + deg, 3]
params = Table[ToExpression[StringJoin["p", ToString[i]]], {i, 1, nparams}];

(* Create DOF as a 4x4 Matrix: First index = node number,
second index={DOF,dDOF/dx,dDOF/dy,dDOF/dz} *)
dofs = Table[0, {j, 1, 4}, {k, 1, 4}];

dofs[[1, 1]] = phi1;
dofs[[1, 2]] = phi1x;
dofs[[1, 3]] = phi1y;
dofs[[1, 4]] = phi1z;
dofs[[2, 1]] = phi2;
dofs[[2, 2]] = phi2x;
dofs[[2, 3]] = phi2y;
dofs[[2, 4]] = phi2z;
dofs[[3, 1]] = phi3;
dofs[[3, 2]] = phi3x;
dofs[[3, 3]] = phi3y;
dofs[[3, 4]] = phi3z;
dofs[[4, 1]] = phi4;
dofs[[4, 2]] = phi4x;
dofs[[4, 3]] = phi4y;
dofs[[4, 4]] = phi4z;

(* Create node coordinates *)
nodecoords1 = {x1, y1, z1};
nodecoords2 = {x2, y2, z2};
nodecoords3 = {x3, y3, z3};
```

```
nodecoords4 = {x4, y4, z4};

(* For convenience, create DOF gradient vectors *)
nodegrads1 = {phi1x, phi1y, phi1z};
nodegrads2 = {phi2x, phi2y, phi2z};
nodegrads3 = {phi3x, phi3y, phi3z};
nodegrads4 = {phi4x, phi4y, phi4z};

(* Determine face centers *)
facecoords124 = (nodecoords1 + nodecoords2 + nodecoords4) / 3;
facecoords123 = (nodecoords1 + nodecoords2 + nodecoords3) / 3;
facecoords134 = (nodecoords1 + nodecoords3 + nodecoords4) / 3;
facecoords234 = (nodecoords2 + nodecoords3 + nodecoords4) / 3;

(* Interpolate values at face centers *)
interpol124 = FullSimplify[Flatten[{
      (phi1 + phi2 + phi4 +
         phi1 + (facecoords124 - nodecoords1).nodegrads1 +
         phi2 + (facecoords124 - nodecoords2).nodegrads2 +
         phi4 + (facecoords124 - nodecoords4).nodegrads4) / 6}]];

interpol123 = FullSimplify[Flatten[{
      (phi1 + phi2 + phi3 +
         phi1 + (facecoords123 - nodecoords1).nodegrads1 +
         phi2 + (facecoords123 - nodecoords2).nodegrads2 +
         phi3 + (facecoords123 - nodecoords3).nodegrads3) / 6}]];

interpol134 = FullSimplify[Flatten[{
      (phi1 + phi3 + phi4 +
         phi1 + (facecoords134 - nodecoords1).nodegrads1 +
         phi3 + (facecoords134 - nodecoords3).nodegrads3 +
         phi4 + (facecoords134 - nodecoords4).nodegrads4) / 6}]];

interpol234 = FullSimplify[Flatten[{
      (phi2 + phi3 + phi4 +
         phi2 + (facecoords234 - nodecoords2).nodegrads2 +
         phi3 + (facecoords234 - nodecoords3).nodegrads3 +
         phi4 + (facecoords234 - nodecoords4).nodegrads4) / 6}]];

(* Set up linear system for solving the 16 DOF +
 4 pDOF for the 20 polynomial coefficients *)
eqs = Flatten[{
     phi[nodecoords1, params] == dofs[[1, 1]],
     ((D[phi[{x, y, z}, params], x]) /. {x → x1, y → y1, z → z1}) == dofs[[1, 2]],
     ((D[phi[{x, y, z}, params], y]) /. {x → x1, y → y1, z → z1}) == dofs[[1, 3]],
     ((D[phi[{x, y, z}, params], z]) /. {x → x1, y → y1, z → z1}) == dofs[[1, 4]],
     phi[nodecoords2, params] == dofs[[2, 1]],
```

```
    ((D[phi[{x, y, z}, params], x]) /. {x → x2, y → y2, z → z2}) == dofs[[2, 2]],
    ((D[phi[{x, y, z}, params], y]) /. {x → x2, y → y2, z → z2}) == dofs[[2, 3]],
    ((D[phi[{x, y, z}, params], z]) /. {x → x2, y → y2, z → z2}) == dofs[[2, 4]],
    phi[nodecoords3, params] == dofs[[3, 1]],
    ((D[phi[{x, y, z}, params], x]) /. {x → x3, y → y3, z → z3}) == dofs[[3, 2]],
    ((D[phi[{x, y, z}, params], y]) /. {x → x3, y → y3, z → z3}) == dofs[[3, 3]],
    ((D[phi[{x, y, z}, params], z]) /. {x → x3, y → y3, z → z3}) == dofs[[3, 4]],
    phi[nodecoords4, params] == dofs[[4, 1]],
    ((D[phi[{x, y, z}, params], x]) /. {x → x4, y → y4, z → z4}) == dofs[[4, 2]],
    ((D[phi[{x, y, z}, params], y]) /. {x → x4, y → y4, z → z4}) == dofs[[4, 3]],
    ((D[phi[{x, y, z}, params], z]) /. {x → x4, y → y4, z → z4}) == dofs[[4, 4]],
    phi[facecoords124, params] == interpol124[[1]],
    phi[facecoords123, params] == interpol123[[1]],
    phi[facecoords134, params] == interpol134[[1]],
    phi[facecoords234, params] == interpol234[[1]]
   }];

(* Extract linear system components *)
coeffs = CoefficientArrays[eqs, params]
```

Out[•]= 20

Out[•]= {SparseArray[Specified elements: 20 Dimensions: {20}], SparseArray[Specified elements: 280 Dimensions: {20, 20}]}

```
(* Export element matrices as Fortran code *)

(* right hand side =
 20 expressions involving the DOF and the node's coordinates *)
(* rhs is linear in the DOF and nonlinear in the node's coordinates *)
rhs = -FullSimplify[Normal[coeffs[[1]]]] ;
(* DOF vector as intended to use in UEL *)
dofvek = {phi1, phi1x, phi1y, phi1z, phi2, phi2x, phi2y,
   phi2z, phi3, phi3x, phi3y, phi3z, phi4, phi4x, phi4y, phi4z};
(* now we produce the 16x20 matrix which maps the vector
 of degrees of freedom to rhs *)
map = CoefficientArrays[rhs, dofvek];
str = OpenWrite["~/code20.txt"];
WriteString[str, "\n! matrix2 maps from the 16 dof to the 20 pseudo-dof \n"]
WriteString[str, "       double precision matrix2(20,16)\n"]
For[i = 1, i ≤ nparams, i++,
 For[j = 1, j ≤ 16, j++,
  WriteString[str, StringJoin["       matrix2(", ToString[i], ",", ToString[j],
    ")=", ToString[FortranForm[ FullSimplify[map[[2, i, j]]] ]], "\n" ]]
 ]
]

(* coefficient matrix that maps polynomial coefficients p1...
 p20 onto the rhs *)
```

```
matrix = FullSimplify[Normal[coeffs[[2]]]];
(* we need the inverse of this matrix to
 calculate the polynomial coefficients from rhs(dof) *)
(* this is done numerically in Fortran,
here we export only the 20x20 matrix *)
WriteString[str, "\n! the q-matrix maps
   from the 20 polynomial coeffs. to the 20 pseudo-dof\n"]
WriteString[str, "      double precision q(20,20)\n"]
For[i = 1, i ≤ nparams, i ++,
 For[j = 1, j ≤ nparams, j ++,
  WriteString[str, StringJoin["      q(", ToString[i], ",",
    ToString[j], ")=", ToString[FortranForm[matrix[[i, j]]]], "\n" ]]
 ]
]

(* To determine the approximation and its
 first and second gradient at any given point *)
(* which is needed for the numerical integration we need the *)
(* polynomial coefficients and the monomials as 20-vectors *)
xvek = {x, y, z};
monome = {};
For[iii = 0, iii <= deg, iii ++,
  For[jjj = 0, jjj <= deg, jjj ++,
    For[kkk = 0, kkk <= deg, kkk ++,
      If[iii + jjj + kkk ≤ deg,
        monome = Append[monome, xvek[[1]] ^ iii * xvek[[2]] ^ jjj * xvek[[3]] ^ kkk]]
      ];];];

(* export shape func monomials *)
WriteString[str, "\n! the shape function monomials \n"]
WriteString[str, "      double precision n(20)\n"]
For[i = 1, i ≤ nparams, i ++,
 WriteString[str, StringJoin["      n(",
   ToString[i], ")=", ToString[FortranForm[monome[[i]]]], "\n"]]
]

(* export first gradient of shape func monomials *)
WriteString[str, "\n! the shape function monomials first gradients \n"]
WriteString[str, "      double precision gn(20,3)\n"]
For[i = 1, i ≤ nparams, i ++,
 For[j = 1, j ≤ 3, j ++,
  WriteString[str, StringJoin["      gn(", ToString[i], ",", ToString[j],
    ")=", ToString[FortranForm[D[monome[[i]], xvek[[j]]]]], "\n"]]
 ]
]

(* export second gradient of shape func monomials *)
WriteString[str, "\n! the shape function monomials second gradients \n"]
WriteString[str, "      double precision ggn(20,3,3)\n"]
```

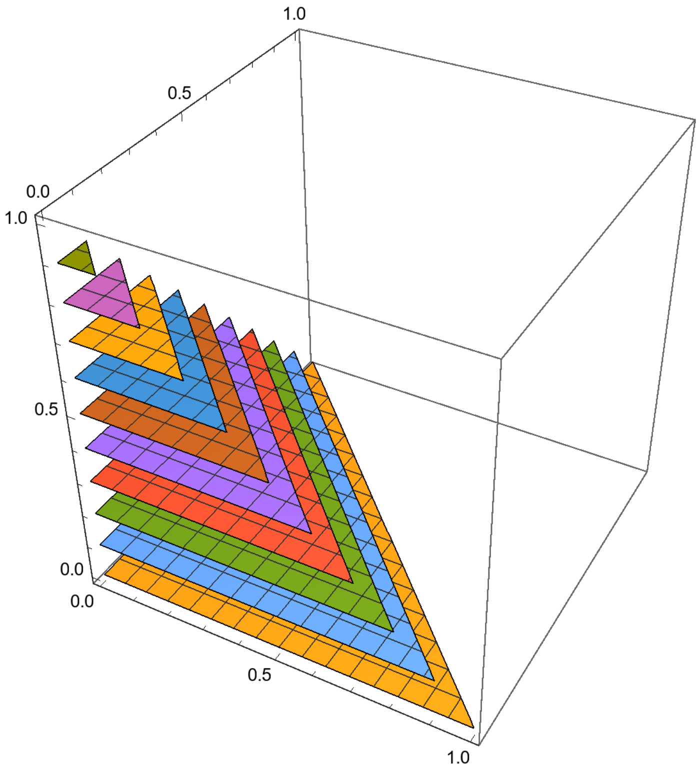
```
For[i = 1, i ≤ nparams, i++,
 For[j = 1, j ≤ 3, j++,
  For[k = 1, k ≤ 3, k++,
   WriteString[str,
    StringJoin["      ggn(", ToString[i], ",", ToString[j], ",", ToString[k],
     ")=", ToString[FortranForm[D[monome[[i]], xvek[[j]], xvek[[k]]]]], "\n"]]
  ]
 ]
]

Close[str]
```

Out[]= /home/gluege/code20.txt

```
In[]:= (* Plot interpolation for some a test cases *)
(* Coordinates for the reference tetrahedron *)
x1 = 0; y1 = 0; z1 = 0;
x2 = 1; y2 = 0; z2 = 0;
x3 = 0; y3 = 1; z3 = 0;
x4 = 0; y4 = 0; z4 = 1;
DOFsToPolyCoeffs = Inverse[matrix].map[[2]];
```

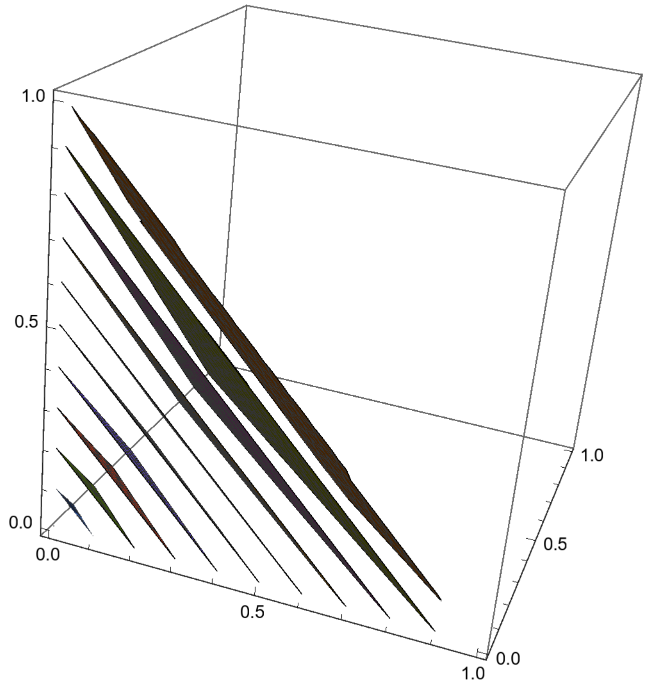
In[•]:=
```
(* constant gradient in <0,0,1> direction, linear field *)
(* DOF-vector: phi1, phi1x,phi1y,phi1z, phi2, ... *)
DOFS = {0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1};
PolyCoeffs = DOFsToPolyCoeffs.DOFS;
phi[x_, y_, z_] = monome.PolyCoeffs;
ContourPlot3D[phi[x, y, z], {x, 0, 1}, {y, 0, 1}, {z, 0, 1},
 RegionFunction → Function[{x, y, z}, Evaluate[x + y + z < 1]],
 Contours → {0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1}]
```

Out[•]=



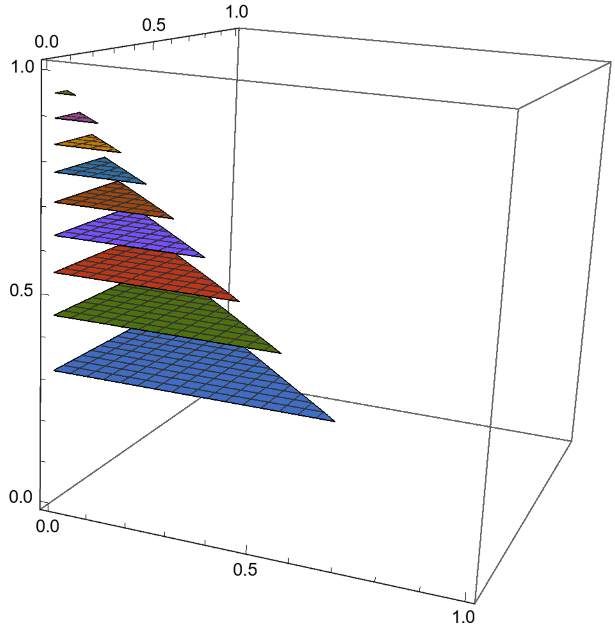
In[ ]:=
```
(* constant gradient in <1,1,1> direction, linear field *)
DOFS = {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
PolyCoeffs = DOFsToPolyCoeffs.DOFS;
phi[x_, y_, z_] = monome.PolyCoeffs;
ContourPlot3D[phi[x, y, z], {x, 0, 1}, {y, 0, 1}, {z, 0, 1},
 RegionFunction → Function[{x, y, z}, Evaluate[x + y + z < 1]],
 Contours → {0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1}]
```

Out[ ]=



In[•]:=
```
(* linear gradient in <0,0,1> direction, quadratic field *)
DOFS = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 2};
PolyCoeffs = DOFsToPolyCoeffs.DOFS;
phi[x_, y_, z_] = monome.PolyCoeffs;
ContourPlot3D[phi[x, y, z], {x, 0, 1}, {y, 0, 1}, {z, 0, 1},
 RegionFunction → Function[{x, y, z}, Evaluate[x + y + z < 1]],
 Contours → {0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1}]
```

Out[•]=



In[ ]:=
```
(* Radial field *)
DOFS = {0, 0, 0, 0,
   1, 2, 0, 0,
   1, 0, 2, 0,
   1, 0, 0, 2};
PolyCoeffs = DOFsToPolyCoeffs.DOFS;
phi[x_, y_, z_] = monome.PolyCoeffs;
k = 10;
ContourPlot3D[phi[x, y, z], {x, 0, 1}, {y, 0, 1}, {z, 0, 1},
 RegionFunction → Function[{x, y, z}, Evaluate[x + y + z < 1]],
 Contours → Table[0.1 i, {i, 0, k}]]
```

Out[ ]=