```
In[172]:= (* Initialisation *)
(* Evaluate before start writing "real code" *)
(* Usage e.g.: "ld [Spacekey]" becomes "⊨",
so writing "a ld 5" turns into "a ⊨ 5" *)
SetOptions[EvaluationNotebook[],
            InputAutoReplacements → {(* special AceGen assignment operators: *)
   "ld" → "⊨", "ls" → "⊢", "rd" → "⫤", "rs" → "⊣",
                                    (* brackets and symbols: *) "dbl" → "⟦",
   "dbr" → "⟧", "lcb" → "{", "rcb" → "}", "lsb" → "[", "rsb" → "]", "->" → "→",
                                    (* shortcuts for
    starting/ending a comment block: *) "co" → "(*", "cc" → "*)"
                                   }
          ]
(* Output the current time,
so we know when AceGen has been executed the last time *)
Now

Out[173]= Thu 13 Jun 2024 13:00:25 GMT+2

In[174]:= (*initialization*)
ClearAll["Global`*"](*all variables are initially cleared*)

(* NAME OF SUBROUTINE YOU WANT TO PRODUCE *)
NAME = "stabilisation_Q1LES _2D";

In[176]:= << AceGen`; (*AceGen is started*)
```

In[177]:=
```
shape_functions_Quad4 [ξ1_, ξ2_] :=
  (
     1 / 4 * {(1 - ξ1) × (1 - ξ2), (*N1*)
    (1 + ξ1) × (1 - ξ2), (*N2*)
    (1 + ξ1) × (1 + ξ2), (*N3*)
    (1 - ξ1) × (1 + ξ2)(*N4*)
    }
  );
quadrature_rule_full  :=
(
        s1by3 ⊨ Sqrt[1 / 3];
        QP_coords ⊨ Transpose[{{-s1by3 , -s1by3}, (*Q1*)
            {+s1by3 , -s1by3}, (*Q2*)
            {+s1by3 , +s1by3}, (*Q3*)
            {-s1by3 , +s1by3}(*Q4*)
                              }];
        QP_weights ⊨ {1, 1, 1, 1};
        Return[{QP_coords , QP_weights }];
  );
quadrature_rule_reduced  :=
(
         QP_coords ⊨ {{0}, {0}};  (*Q5*)
         QP_weights ⊨ {4};

         Return[{QP_coords , QP_weights }];
);
```

In[180]:=
```
dofs_per_node = 2;
n_nodes = 4;
ndtot = n_nodes * dofs_per_node ;
```

```
(* Programming language, Mode: Debug/Prototype/Optimal *)
SMSInitialize[NAME, "Language" → "Fortran", "Mode" → "Optimal"];
(* Create the module named NAME with all inputs and outputs *)
(* Inputs:
 - X: (4,2) array with undeformed nodal
   coordinates of the 4 nodes of this 2D quadrilateral element
  - u: (8) array with displacement values (degrees of freedom) listed as
       { u_N1_x, u_N1_y, u_N2_x, u_N2_y, ..., u_N4_y } for each Node N1...
     N4 with components {x,y}
    - bulkModkappa: bulk modulus of the base material
    - shearModmu: shear modulus of the base material
      - HGscale: hourglass control coefficient,
e.g. 1e-4 (higher values cause stronger stabilisation)
 - istif: integer/boolean to request the stiffness matrix,
istif=1: stiffness matrix is requested thus computed therein,
istif=0: stiffness matrix is not requested and not computed herein
   Outputs:
  - forceHG: (8) array of internal force components for hourglass
      stabilisation for each degree of freedom of the current element
     - stiffHG: (8x8) array with component of the stiffness
     matrix for the stabilisation ´for the current element
*)
SMSModule[NAME, Real[X$$[n_nodes, dofs_per_node],
   u$$[n_nodes, dofs_per_node], bulkModkappa$$, shearModmu$$, HGscale$$],
  Integer[istif$$], Real[forceHG$$[ndtot], stiffHG$$[ndtot, ndtot]],
  "Input" → {X$$, u$$, bulkModkappa$$, shearModmu$$, HGscale$$, istif$$},
  "Output" → {forceHG$$, stiffHG$$}];
```

```
In[185]:= (* Input declaration / copy Acegen variables to Mathematica variables *)
 XIO ⊨ SMSReal[Table[X$$[iNode, jdof], {iNode, n_nodes}, {jdof, dofs_per_node}]];
 uIO ⊨ SMSReal[Table[u$$[iNode, jdof], {iNode, n_nodes}, {jdof, dofs_per_node}]];
 pe ⊨ Flatten[uIO];

 κ ⊨ SMSReal[bulkModkappa$$];
 μ ⊨ SMSReal[shearModmu$$];
 λ ⊨ κ - 2/3*μ;

 HGscaleValue ⊨ SMSReal[HGscale$$];
 istif ⊨ SMSInteger[istif$$];
```

In[193]:=
```
(* Initialise output variables to zero
 (can be included optionally, if not done by the caller subroutine) *)
 (* SMSExport[Table[0,{idof,ndtot}],forceHG$$];
  SMSExport[Table[0,{idof,ndtot},{jdof,ndtot}],stiffHG$$];*)
```

In[194]:=
```
(* Standard QP coordinates and weights
 for full integration (FuI) of stabilisation *)
 {QP_coords_FuI , QP_weights_FuI } ⊨ quadrature_rule_full ;
(* Reduced integration (RI) for energy compensation *)
 {QP_coords_RI , QP_weights_RI } ⊨ quadrature_rule_reduced ;
(* Merge the two lists, with first FuI followed by RI *)
 QP_coords = Join[QP_coords_FuI , QP_coords_RI , 2];
 QP_weights = Join[QP_weights_FuI , QP_weights_RI ];
```

In[198]:=
```
(* Loop over combined list of QPs *)
 SMSDo[qpoint, 1, 5];
   (* Using SMSPart to access qpoint's entry: *)
    ξ1 ⊢ SMSReal[SMSPart[QP_coords⟦1⟧, qpoint]];
    ξ2 ⊢ SMSReal[SMSPart[QP_coords⟦2⟧, qpoint]];
   Ξ = {ξ1, ξ2};
   weight_qpoint ⊢ SMSPart[QP_weights, qpoint];
   Nh ⊨ shape_functions_Quad4 [ξ1, ξ2];
   X ⊢ SMSFreeze[Nh . XIO];
   u ⊨ Nh . uIO;
   𝕁e ⊨ SMSD[X, Ξ];
   Jed ⊨ Det[𝕁e];
   ℍ ⊨ SMSD[u, X, "Dependency" → {Ξ, X, SMSInverse[𝕁e]}];
   ϵ_2D ⊨ 1/2 * (ℍᵀ + ℍ);
   ϵ ⊨ ( ϵ_2D⟦1, 1⟧  ϵ_2D⟦1, 2⟧  0
         ϵ_2D⟦2, 1⟧  ϵ_2D⟦2, 2⟧  0
         0           0           1 );

   (* For the fully integrated QPs we add the
 stabilising energy with a positive HG scaling factor *)
    SMSIf[qpoint ≤ 4];
      HGscale ⫤ HGscaleValue * 1;
    (* and for the reduced integrated centre QP (qpoint==9),
we remove the "same" energy (1 QP with weight=8)
 leaving "only" the hourglass stabilising energy *)
    SMSElse[];
      HGscale ⊣ HGscaleValue * (-1);
    SMSEndIf[HGscale];
```

```
(* Linear elastic energy *)
W ⊨ HGscale * (λ/2 * (Tr[ϵ])^2 + μ * Tr[ϵ . ϵ]);

(* Compute the residual/force and the stiffness matrix *)
SMSDo[m, 1, ndtot];
 ℝgm ⊨ Jed * SMSD[W, pe, m];
 SMSExport[weight_qpoint * ℝgm, forceHG$$[m], "AddIn" → True];
 SMSDo[n, 1, ndtot];
  𝕂gmn ⊨ SMSIf[istif == 1, SMSD[ℝgm, pe, n], 0.0];
  SMSExport[weight_qpoint * 𝕂gmn, stiffHG$$[m, n], "AddIn" → True];
 SMSEndDo[];
SMSEndDo[];

SMSEndDo[]; (*End Gauss Quadrature Loop*)
```

In[226]:=
```
(* write output file *)
SMSWrite[NAME, "LocalAuxiliaryVariables " → True ];

(* print file on screen *)
NAME_FileExtension = Which[SMSLanguage == "Fortran",
   ".f", SMSLanguage == "Matlab", ".m", SMSLanguage == "C++", ".cpp"];
FilePrint[StringJoin[NAME, NAME_FileExtension ]]
```

File: stabilisation_Q1LES _2D.cpp **Size:** 4317 **Time:** 2

Method	stabilisation_Q1LES _2D
No.Formulae	88
No.Leafs	1137

```
/*************************************************************
* AceGen    7.505 Linux (16 Aug 22)                         *
*           Co. J. Korelc  2020           13 Jun 24 13:00:28 *
**************************************************************
User     : Full professional version
Notebook : stabilisation_Q1LES_2D
Evaluation time                 : 2 s     Mode   : Optimal
Number of formulae              : 88      Method: Automatic
Subroutine                      : stabilisation_Q1LES_2D  size: 1137
Total size of Mathematica  code : 1137 subexpressions
Total size of C code            : 3689 bytes */
#include "sms.h"

/******************* S U B R O U T I N E *********************/
void stabilisation_Q1LES_2D(double  X[4][2],double  u[4][2],double
     (*bulkModkappa),double  (*shearModmu),double  (*HGscale),int  (*istif)
     ,double  forceHG[8],double  stiffHG[8][8])
{
```

```
double  v[343];
int i23,i77,i86,b74,b87,b88,b117;
v[161]=1e0;
v[162]=1e0;
v[163]=1e0;
v[164]=1e0;
v[165]=4e0;
v[156]=-0.5773502691896257e0;
v[157]=-0.5773502691896257e0;
v[158]=0.5773502691896257e0;
v[159]=0.5773502691896257e0;
v[160]=0e0;
v[151]=-0.5773502691896257e0;
v[152]=0.5773502691896257e0;
v[153]=0.5773502691896257e0;
v[154]=-0.5773502691896257e0;
v[155]=0e0;
v[1]=X[0][0];
v[2]=X[0][1];
v[3]=X[1][0];
v[112]=v[1]-v[3];
v[4]=X[1][1];
v[110]=v[2]-v[4];
v[5]=X[2][0];
v[108]=v[3]-v[5];
v[6]=X[2][1];
v[106]=v[4]-v[6];
v[7]=X[3][0];
v[113]=v[5]-v[7];
v[107]=v[1]-v[7];
v[8]=X[3][1];
v[111]=v[6]-v[8];
v[105]=v[2]-v[8];
v[9]=u[0][0];
v[10]=u[0][1];
v[11]=u[1][0];
v[12]=u[1][1];
v[13]=u[2][0];
v[14]=u[2][1];
v[15]=u[3][0];
v[16]=u[3][1];
v[18]=(*shearModmu);
v[102]=2e0*v[18];
v[19]=(*bulkModkappa)+(-2e0/3e0)*v[18];
v[20]=(*HGscale);
b87=(*istif)==1;
for(i23=1;i23<=5;i23++){
 v[24]=v[150+i23];
 v[40]=(-1e0+v[24])/4e0;
 v[41]=(-1e0-v[24])/4e0;
 v[46]=v[105]*v[40]+v[106]*v[41];
 v[44]=v[107]*v[40]+v[108]*v[41];
 v[25]=v[155+i23];
 v[42]=(1e0+v[25])/4e0;
 v[39]=(-1e0+v[25])/4e0;
```

```
v[45]=v[110]*v[39]+v[111]*v[42];
v[43]=v[112]*v[39]+v[113]*v[42];
v[26]=v[160+i23];
v[47]=-(v[44]*v[45])+v[43]*v[46];
v[48]=-(v[46]/v[47]);
v[62]=-(v[42]*v[48]);
v[54]=-(v[39]*v[48]);
v[49]=v[44]/v[47];
v[65]=-(v[42]*v[49]);
v[56]=-(v[39]*v[49]);
v[50]=-(v[45]/v[47]);
v[63]=v[40]*v[50];
v[58]=v[41]*v[50];
v[51]=v[43]/v[47];
v[66]=v[40]*v[51];
v[60]=v[41]*v[51];
v[52]=v[54]+v[63];
v[53]=v[56]+v[66];
v[55]=-v[54]+v[58];
v[57]=-v[56]+v[60];
v[59]=-v[58]+v[62];
v[61]=-v[60]+v[65];
v[64]=-v[62]-v[63];
v[67]=-v[65]-v[66];
v[68]=v[11]*v[55]+v[13]*v[59]+v[15]*v[64]+v[52]*v[9];
v[83]=v[10]*v[52]+v[12]*v[55]+v[11]*v[57]+v[14]*v[59]+v[13]*v[61]+v[16]*v[64]+v[15]*v[67]
 +v[53]*v[9];
v[197]=v[53]*v[83];
v[198]=v[52]*v[83];
v[199]=v[57]*v[83];
v[200]=v[55]*v[83];
v[201]=v[61]*v[83];
v[202]=v[59]*v[83];
v[203]=v[67]*v[83];
v[204]=v[64]*v[83];
v[71]=v[10]*v[53]+v[12]*v[57]+v[14]*v[61]+v[16]*v[67];
v[81]=v[19]*(1e0+v[68]+v[71]);
if(i23<=4){
 v[75]=v[20];
} else {
 v[75]=-v[20];
};
v[103]=v[18]*v[75];
v[96]=v[102]*v[75];
v[80]=v[75]*(v[102]*v[71]+v[81]);
v[82]=v[75]*(v[102]*v[68]+v[81]);
v[181]=v[52]*v[82];
v[182]=v[53]*v[80];
v[183]=v[55]*v[82];
v[184]=v[57]*v[80];
v[185]=v[59]*v[82];
v[186]=v[61]*v[80];
v[187]=v[64]*v[82];
v[188]=v[67]*v[80];
if(b87){
```

```
    v[220]=0e0;
    v[221]=v[53];
    v[222]=0e0;
    v[223]=v[57];
    v[224]=0e0;
    v[225]=v[61];
    v[226]=0e0;
    v[227]=v[67];
    v[212]=v[52];
    v[213]=0e0;
    v[214]=v[55];
    v[215]=0e0;
    v[216]=v[59];
    v[217]=0e0;
    v[218]=v[64];
    v[219]=0e0;
    v[228]=v[53];
    v[229]=v[52];
    v[230]=v[57];
    v[231]=v[55];
    v[232]=v[61];
    v[233]=v[59];
    v[234]=v[67];
    v[235]=v[64];
   } else {
   };
   for(i77=1;i77<=8;i77++){
    forceHG[i77-1]+=(v[180+i77]+v[103]*v[196+i77])*v[26]*v[47];
    for(i86=1;i86<=8;i86++){
     if(b87){
      v[91]=v[211+i77]*v[47];
      v[92]=v[219+i77]*v[47];
      v[97]=v[19]*v[75]*(v[91]+v[92]);
      v[95]=v[92]*v[96]+v[97];
      v[98]=v[91]*v[96]+v[97];
      v[236]=v[52]*v[98];
      v[237]=v[53]*v[95];
      v[238]=v[55]*v[98];
      v[239]=v[57]*v[95];
      v[240]=v[59]*v[98];
      v[241]=v[61]*v[95];
      v[242]=v[64]*v[98];
      v[243]=v[67]*v[95];
      v[100]=v[235+i86]+v[103]*v[227+i77]*v[227+i86]*v[47];
     } else {
      v[100]=0e0;
     };
     stiffHG[i77-1][i86-1]+=v[100]*v[26];
    };/* end for */
   };/* end for */
  };/* end for */
  };
```