

```

1 #ifndef MATERIAL_MODEL_HYPERELASTIC
2 #define MATERIAL_MODEL_HYPERELASTIC
3
4 #include "/src/Definitions.h"
5 #include "PJ2Utilities.h"
6 #include "Eigen/Eigen"
7 #include <iostream>
8 #include <Eigen/Dense>
9 using namespace Eigen;
10 using namespace std;
11
12 namespace MaterialModels {
13
14 class MaterialModelHyperelastic {
15
16 public:
17
18 // TODO: Given that DisplacementGradient, Stress, Strain and TangentMatrix are in
19 Voigt-notation,
20 // determine the number of rows and columns of each of those matrices
21 // REMINDER: The second template parameter for Matrix is the number or rows, the
22 third template
23 // parameter denotes the number of columns
24 typedef Matrix<double, 9, 1> DisplacementGradient;
25 typedef Matrix<double, 9, 1> Stress;
26 typedef Matrix<double, 9, 1> Strain;
27 typedef Matrix<double, 3, 3> StandardStrainMatrix;
28 typedef Matrix<double, 9, 9> TangentMatrix;
29
30 //
31 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
32 !!!!!!!!!!!!!!!
33
34 // CAREFUL: After changing the dimensions, please make sure to also change the
35 hard-coded 1's below. (Otherwise you will get a large Eigen-mistake)
36
37 //
38 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
39 !!!!!!!!!!!!!!!
40
41 // Constructor
42 MaterialModelHyperelastic(const double lambda, const double mu):
43 // TODO: Set the Lamé parameters _lambda and _mu based on the input
44 // CAUTION: _lambda and _mu are endowed with the 'const'-qualifier and hence can
45 only
46 // be defined in this way as part of the constructor.
47 _lambda(lambda),
48 _mu(mu){
49
50 }
51
52 Strain
53 computeStrain(const DisplacementGradient & displacementGradient) const {
54
55 // TODO: Evaluate the strain vector (here: F) based on the displacement gradient.
56 // NOTE: The functionality to convert from standard to Voigt notation can be found
57 // in the Utilities namespace (see PJ2Utilities.h) (it might come in handy
58 here...)
59 Strain strain = Strain::Zero();
60 Matrix<double,3,3> identityMatrix3x3 = Matrix<double,3,3>::Identity();
61 Matrix<double,9,1> identityMatrixVectorForm9x1 =
62 Utilities::convertTensorFromStandardToVoigt<3>(identityMatrix3x3);
63 for (int i = 0; i<9; i++){
64     strain(i,0) += identityMatrixVectorForm9x1(i,0) + displacementGradient(i,0);
65 };
66
67 // HINT: We will show once, how to convert a 3x3 matrix to a 9x1 matrix.
68 // You can remove this if you feel like you've understood the concept.
69 //Matrix<double,3,3> some3x3Matrix = Matrix<double,3,3>::Identity();
70 //Matrix<double,9,1> some9x1Matrix
71 // = Utilities::convertTensorFromStandardToVoigt<3>(some3x3Matrix);

```

```

64 //Matrix<double,3,3> reconverted3x3Matrix
65 // = Utilities::convertTensorFromVoigtToStandard<3>(some9x1Matrix);
66
67
68 // TODO: Remove the following ignoreUnusedVariables lines
69 //ignoreUnusedVariables(displacementGradient);
70 //ignoreUnusedVariables(reconverted3x3Matrix);
71
72 return strain;
73
74 };
75
76
77 double
78 computeEnergy(const DisplacementGradient & displacementGradient) const {
79
80 // Evaluate F based on displacementGradient using computeStrain
81 Strain strainVector = computeStrain(displacementGradient);
82 StandardStrainMatrix F =
83 Utilities::convertTensorFromVoigtToStandard<3>(strainVector);
84
85 // TODO: Remove the following ignoreUnusedVariables lines
86 //ignoreUnusedVariables(F);
87
88 // TODO: If necessary, evaluate useful scalars such as the I1 invariant
89 // and the volumetric expansion J based on strainMatrix
90
91 StandardStrainMatrix FMatrixTranspose = F.transpose();
92 double determinantOfF = F.determinant();
93 Matrix<double, 3, 3> CMatrix;
94 /*for (int i = 0; i <3; i++){
95     for (int j = 0; j<3; j++){
96         CMatrix (i,j) = F(i,j)* FMatrixTranspose(i,j);
97     }
98 }*/
99 CMatrix = FMatrixTranspose*F;
100 double traceOfC = 0.0;
101 for (int i = 0; i < 3; i++){
102     for (int j = 0; j<3; j++){
103         if (i ==j){
104             traceOfC += CMatrix(i,j);
105         }
106     }
107 }
108 // traceOfC = CMatrix.trace()
109 //double determinantOfF = 0.0;
110
111 // TODO: Evaluate the energy density
112 double energyDensity = 0.0;
113 energyDensity = _mu * (traceOfC - 3)/2 +
114 _lambda*log(determinantOfF)*log(determinantOfF)/2 - _mu * log(determinantOfF);
115
116 // TODO: Remove the following ignoreUnusedVariables line
117 //ignoreUnusedVariables(displacementGradient,strainVector);
118
119 // Return
120 return energyDensity;
121 }
122
123
124 Stress
125 computeStress(const DisplacementGradient & displacementGradient) const {
126
127 // TODO : Evaluate F based on displacementGradient using computeStrain
128 Strain strainVector = Strain::Zero();
129 strainVector = computeStrain(displacementGradient);
130 StandardStrainMatrix F =
131 Utilities::convertTensorFromVoigtToStandard<3>(strainVector);
132
133 // TODO: If necessary, evaluate useful scalars such as the I1 invariant

```

```

134 //         and the volumetric expansion J or tensors such as the right Cauchy-Green
135 //         tensor C or the inverse-transpose of F
136 double determinantOfF = F.determinant();
137 StandardStrainMatrix inverseTransposeOfF = F.inverse().transpose();
138
139 // TODO: Based on the possibly-useful scalars and tensors, you've obtained above,
140 //         determine the stress tensor.
141 Stress stress = Stress::Zero();
142 StandardStrainMatrix stressMatrixP =
143 Utilities::convertTensorFromVoigtToStandard<3>(stress);
144 stressMatrixP = _mu * F + _lambda * log (determinantOfF) * inverseTransposeOfF -
145 _mu * inverseTransposeOfF;
146 stress = Utilities::convertTensorFromStandardToVoigt<3>(stressMatrixP);
147
148 // TODO: Remove the following ignoreUnusedVariables line
149 //ignoreUnusedVariables(displacementGradient, strainVector);
150
151 // Return
152 return stress;
153 };
154
155 TangentMatrix
156 computeTangentMatrix(const DisplacementGradient & displacementGradient) const {
157
158 // TODO : Evaluate F based on displacementGradient using computeStrain
159 Strain strainVector = Strain::Zero();
160 strainVector = computeStrain(displacementGradient);
161 StandardStrainMatrix F =
162 Utilities::convertTensorFromVoigtToStandard<3>(strainVector);
163
164 // TODO: If necessary, evaluate useful scalars such as the I1 invariant
165 //         and the volumetric expansion J or tensors such as the left Cauchy-Green
166 //         tensor C or the inverse-transpose of F
167 double determinantOfF = F.determinant();
168 StandardStrainMatrix inverseOfF = F.inverse();
169
170
171
172 // TODO: We keep our lives simple and define a 4th order tensor as an array first.
173 //         1st: Set the right dimensions (i.e. replace the 1's by the correct
174 //         dimension)
175 //         2nd: Set all entries of tangentMatrixAsArray. You may have to use
176 //         nested for-loops.
177 array<array<array<array<double,3>,3>,3>,3> tangentMatrixAsArray;
178 for (int i = 0; i<3; i++){
179     for (int J = 0; J<3; J++){
180         for (int k = 0; k<3; k++){
181             for (int L = 0; L<3; L++){
182                 if ((i == k) && (J == L)){
183                     tangentMatrixAsArray [i][J][k][L] = _mu *(1 +
184                     inverseOfF(J,k) * inverseOfF(L,i)) + _lambda *
185                     (inverseOfF(L,k) * inverseOfF(J,i) - log
186                     (determinantOfF)*inverseOfF(J,k) * inverseOfF(L,i));
187                 }
188                 else {
189                     tangentMatrixAsArray [i][J][k][L] = _mu *(inverseOfF(J,k) *
190                     inverseOfF(L,i)) + _lambda * (inverseOfF(L,k) *
191                     inverseOfF(J,i) - log (determinantOfF)*inverseOfF(J,k) *
192                     inverseOfF(L,i));
193                 }
194             }
195         }
196     }
197 }
198
199 // TODO: Simply replace the 1 again by the right dimension
200 // We'll take care of the rest of the conversion. The functionality to convert
201 // between standard
202 // and Voigt is found in the Utilities namespace in PJ2Utilities.h.
203 TangentMatrix tangentMatrix

```

```
195         =
196         Utilities::convertFourthOrderTensorFromStandardToVoigt<3>(tangentMatrixAsArray);
197
198         // TODO: Remove the following ignoreUnusedVariables line
199         //ignoreUnusedVariables(displacementGradient, strainVector);
200
201
202         // Return
203         return tangentMatrix;
204     };
205
206
207
208     private:
209     const double _lambda;
210     const double _mu;
211
212     };
213 }
214 #endif // MATERIAL_MODEL_HYPERELASTIC
215
```