

A

Táblázatok

Paraméter	Megnevezés
$A_i = 20 \text{ mm}$	nagyobb ellipszis féltengely
$B_i = 18 \text{ mm}$	kisebb ellipszis féltengely
$\theta_i = 70^\circ$	ellipszis nyílási szöge
$t_i = 2,5 \text{ mm}$	belső héj vastagsága
$P_c = 0,04 \text{ MPa}$	konstans nyomás a külső felületen
$\mu_1 = 0,62 \text{ MPa}$	Ogden modell paraméter
$\alpha_1 = 1,3$	Ogden modell paraméter
$n_{\text{RIKS},c} = 550$	RIKS iterációk száma, egyszerű modell
$n_{\text{RIKS},fc} = 150$	RIKS iterációk száma, "Fluid Cavity" modell
$P_{fc} = 0,04 \text{ MPa}$	"Fluid Cavity" előírt nyomása
$\rho = 1000 \text{ kg/m}^3$	anyag sűrűsége
$K_f = 200 \text{ GPa}$	anyag térfogati rugalmassági modulusza

A.1. táblázat. A 3.2-es szekcióban bevezetett végeselemes modellekhez használt paraméterek

B

PYTHON szkriptek

Figyelem!

A kimásolt szkriptek tabulálását futtatás előtt ellenőrizni kell a dokumentációba való beillesztés sortördelése miatt.

B.1. Modellezés

B.1.1. Belső héj modellezése

```
1 from abaqus import *
2 from abaqusConstants import *
3 from numpy import *
4 import mesh
5 import regionToolset
6 import xyPlot
7
8 # ABAQUS script to generate the inner cap of the actuator
9 # Usage: set the geometric data below, and run the script
10 # Working with ABAQUS v. 6.20 (2020)
11 # Created by Barnabas Piri, 2022
12
13 # MODEL NAME
14 NAME = 'innerCap_ConstPress'
15
16 # Geometric data for the inner cap (needs to be changed)
17 OP_ANGLE_inner = 70.0 # deg (Opening angle)
18 A_inner = 20.0 # mm (Semi axis in "x" direction)
19 B_inner = 18.0 # mm (Semi axis in "y" direction)
20 TH_inner = 2.5 # mm (Shell thickness)
21
22 # Geometric data for the outer cap (could be changed)
23 OP_ANGLE_outer = 90.0 # deg (Opening angle)
24 TH_outer = TH_inner / 2 # mm (Shell thickness)
```

```

25
26 TH_RING = TH_inner / 2 # mm (Shell thickness)
27
28 # Material properties
29 # First order Ogden model
30 mu = 0.62e6 # Pa
31 alpha = 1.3 # 1
32
33 # CALCULATIONS
34
35 # Functions
36 def deg2rad(angle_deg):
37     angle_rad = angle_deg / 180 * pi
38     return angle_rad
39
40 # Convert the angles to radians
41 OP_ANGLE_inner = deg2rad(OP_ANGLE_inner)
42 OP_ANGLE_outer = deg2rad(OP_ANGLE_outer)
43
44 # Convert the length units to SI (meters)
45 A_inner = A_inner / 1000
46 B_inner = B_inner / 1000
47 TH_inner = TH_inner / 1000
48 TH_outer = TH_outer / 1000
49 TH_RING = TH_RING / 1000
50
51 # Calculating the outer radius from the other parameters
52 r_ell = (A_inner * B_inner) / sqrt( (A_inner*sin(pi/2 - OP_ANGLE_inner))**2 +
53     (B_inner*cos(pi/2 - OP_ANGLE_inner))**2 )
54
55 Y6_CONST = r_ell * cos(OP_ANGLE_inner)
56
57 X6_CONST = sqrt( (1 - (Y6_CONST / (B_inner+TH_inner/2))**2 ) *
58     (A_inner + TH_inner/2)**2 )
59
60 A = 1 - cos(OP_ANGLE_outer)**2
61 B = -TH_outer
62 C = (TH_outer/2)**2 - X6_CONST**2
63
64 R_outer = (-B + sqrt(B**2 - 4*A*C)) / (2*A)
65
66 print("R_outer: ", R_outer)
67
68 # Calculating the outer cap's excentricity
69 DELTA = r_ell*cos(OP_ANGLE_inner) - R_outer*cos(OP_ANGLE_outer)
70
71 print("DELTA: ", DELTA)
72
73 # Coordinates of the points
74 P1 = [0, B_inner - TH_inner/2]
75 P2 = [0, B_inner + TH_inner/2]
76 P3 = [0, DELTA + R_outer - TH_outer/2]
77 P4 = [0, DELTA + R_outer + TH_outer/2]
78
79 P5 = [sqrt( (R_outer + TH_outer/2)**2 - (R_outer*cos(OP_ANGLE_outer))**2 ),
80     DELTA + R_outer*cos(OP_ANGLE_outer)]
81
82 P6_inner = [sqrt( (1 - (r_ell * cos(OP_ANGLE_inner) / (B_inner+TH_inner/2))**2 ) *
83     (A_inner + TH_inner/2)**2 ), r_ell*cos(OP_ANGLE_inner)]
84
85 P6_outer = [sqrt( (R_outer - TH_outer/2)**2 - (R_outer*cos(OP_ANGLE_outer))**2 ),
86     DELTA + R_outer*cos(OP_ANGLE_outer)]
87
88 # Calculating the ring radius
89 R_RING = P5[0] * 1.05

```

```

90 print("R_RING: ", R_RING)
91
92 P7 = [R_RING, P6_inner[1]]
93 P8 = [R_RING, P6_inner[1] - TH_RING]
94 P9 = [sqrt( (1 - ((P6_inner[1] - TH_RING) / (B_inner - TH_inner/2))**2 ) *
95         (A_inner - TH_inner/2)**2 ), P6_inner[1] - TH_RING]
96
97 ORIGO_inner = [0, 0]
98 ORIGO_outer = [0, DELTA]
99
100 # Making the inner cap
101 s = mdb.models['Model-1'].ConstrainedSketch(name='Sketch-InnerCap',
102                                             sheetSize=60e-3)
103
104 g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
105 s.setPrimaryObject(option=STANDALONE)
106
107 # Skteching
108 s.EllipseByCenterPerimeter(center=ORIGO_inner,axisPoint1=(A_inner + TH_inner/2, 0),
109                             axisPoint2=(0, B_inner + TH_inner/2))
110
111 s.EllipseByCenterPerimeter(center=ORIGO_inner,axisPoint1=(A_inner - TH_inner/2, 0),
112                             axisPoint2=(0, B_inner - TH_inner/2))
113 s.Line(point1=P7, point2=P6_inner)
114 s.Line(point1=P7, point2=P8)
115 s.Line(point1=P9, point2=P8)
116 s.Line(point1=P1, point2=P2)
117 s.autoTrimCurve(curve1=g[2], point1=(P2[0]-0.1e-3, P2[1]-0.1e-3 ))
118 s.autoTrimCurve(curve1=g[4], point1=(P1[0]-0.1e-3, P1[1]-0.1e-3 ))
119 s.autoTrimCurve(curve1=g[11], point1=(A_inner - TH_inner/2, +0.1e-3 ))
120 s.autoTrimCurve(curve1=g[10], point1=(A_inner + TH_inner/2, +0.1e-3 ))
121 s.autoTrimCurve(curve1=g[13], point1=(P6_inner[0]+0.1e-3, P6_inner[1]-0.1e-3))
122
123 # Creating and partitioning the part
124 s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=0.06)
125 s.sketchOptions.setValues(decimalPlaces=3, viewStyle=AXISYM)
126 s.setPrimaryObject(option=STANDALONE)
127 s.ConstructionLine(point1=(0.0, -0.03), point2=(0.0, 0.03))
128 s.retrieveSketch(sketch=mdb.models['Model-1'].sketches['Sketch-InnerCap'])
129 p = mdb.models['Model-1'].Part(name='Part-InnerCap', dimensionality=AXISYMMETRIC,
130                                type=DEFORMABLE_BODY)
131 p = mdb.models['Model-1'].parts['Part-InnerCap']
132 p.BaseShell(sketch=s)
133 s.unsetPrimaryObject()
134 session.viewports['Viewport: 1'].setValues(displayedObject=p)
135 del mdb.models['Model-1'].sketches['__profile__']
136
137 # Assembly instance
138 a = mdb.models['Model-1'].rootAssembly
139 p = mdb.models['Model-1'].parts['Part-InnerCap']
140 a.Instance(name='Part-InnerCap-1', part=p, dependent=ON)
141
142 # Set for the inner cap
143 f = p.faces # Set for the part
144 faces = f.getSequenceFromMask(mask=('[#1 ]', ), )
145 p.Set(faces=faces, name='Set-Part-InnerCap')
146 f1 = a.instances['Part-InnerCap-1'].faces # Set for the instance
147 faces1 = f1.getSequenceFromMask(mask=('[#1 ]', ), )
148 a.Set(faces=faces1, name='Set-InnerCap')
149
150 # Set for outer surface
151 e1 = a.instances['Part-InnerCap-1'].edges
152 edges1 = e1.getSequenceFromMask(mask=('[#4 ]', ), )
153 a.Set(edges=edges1, name='Set-PressSurface')
154

```

```

155 # Creating materials
156 mdb.models['Model-1'].Material(name='Material-Rubber-InnerCap')
157 mdb.models['Model-1'].materials['Material-Rubber-InnerCap']
158     .Density(table=((1000.0, ), ))
159
160 mdb.models['Model-1'].materials['Material-Rubber-InnerCap']
161     .Hyperelastic(materialType=ISOTROPIC, testData=OFF,
162     type=OGDEN, n=1, volumetricResponse=VOLUMETRIC_DATA,
163     table=((mu, alpha, 0), ))
164 mdb.models['Model-1'].HomogeneousSolidSection(name='Section-Rubber-InnerCap',
165     material='Material-Rubber-InnerCap', thickness=None)
166
167 # Assing sections
168 region = p.sets['Set-Part-InnerCap']
169 p.SectionAssignment(region=region, sectionName='Section-Rubber-InnerCap',
170     offset=0.0, offsetType=MIDDLE_SURFACE,
171     offsetField='', thicknessAssignment=FROM_SECTION)
172
173 # Meshing
174 MESHSIZE = (P2[1] - P1[1]) / 4
175
176 f = p.faces
177 pickedRegions = f.getSequenceFromMask(mask=('[#1 ]', ), )
178 p.setMeshControls(regions=pickedRegions, elemShape=QUAD, algorithm=MEDIAL_AXIS)
179 elemType1 = mesh.ElemType(elemCode=CAX4H, elemLibrary=STANDARD)
180 elemType2 = mesh.ElemType(elemCode=CAX3, elemLibrary=STANDARD)
181 faces = f.getSequenceFromMask(mask=('[#1 ]', ), )
182 pickedRegions =(faces, )
183 p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
184 p.seedPart(size=MESHSIZE, deviationFactor=0.1, minSizeFactor=0.1)
185 p.generateMesh()
186
187 # Riks-Static step
188 mdb.models['Model-1'].StaticRiksStep(name='Step-Riks', previous='Initial',
189 maxNumInc=550, initialArcInc=1e-10, minArcInc=1e-32, maxArcInc=0.05, nlgeom=ON)
190
191 # Boundary conditions
192 a = mdb.models['Model-1'].rootAssembly
193 e1 = a.instances['Part-InnerCap-1'].edges
194 edges1 = e1.getSequenceFromMask(mask=('[#8 ]', ), )
195 region = regionToolset.Region(edges=edges1)
196 mdb.models['Model-1'].XsymmBC(name='BC-Symmetry', createStepName='Initial',
197     region=region, localCsys=None)
198 a = mdb.models['Model-1'].rootAssembly
199 v1 = a.instances['Part-InnerCap-1'].vertices
200 verts1 = v1.getSequenceFromMask(mask=('[#4 ]', ), )
201 region = regionToolset.Region(vertices=verts1)
202 mdb.models['Model-1'].XasymBC(name='BC-Roller', createStepName='Initial',
203     region=region, localCsys=None)
204
205 # Creating load
206 a = mdb.models['Model-1'].rootAssembly
207 s1 = a.instances['Part-InnerCap-1'].edges
208 side1Edges1 = s1.getSequenceFromMask(mask=('[#4 ]', ), )
209 region = a.Surface(side1Edges=side1Edges1, name='Surf-Pressure')
210 mdb.models['Model-1'].Pressure(name='Load-Pressure', createStepName='Step-Riks',
211     region=region, distributionType=UNIFORM, field='', magnitude=30000.0)
212
213 # Creating the job
214 mdb.Job(name='Job-' + NAME, model='Model-1', description='',
215     type=ANALYSIS, atTime=None, waitMinutes=0, waitHours=0, queue=None,
216     memory=90, memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
217     explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, echoPrint=OFF,
218     modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSubroutine='',
219     scratch='', resultsFormat=ODB, multiprocessingMode=DEFAULT, numCpus=3,

```

```
220         numDomains=3, numGPUs=0)
221
222 # Viewport fitting
223 session.viewports['Viewport: 1'].view.fitView()
224
225 # History output settings
226 a = mdb.models['Model-1'].rootAssembly
227 a.regenerate()
228 session.viewports['Viewport: 1'].setValues(displayedObject=a)
229 session.viewports['Viewport: 1'].assemblyDisplay
230     .setValues(adaptiveMeshConstraints=ON)
231 regionDef=mdb.models['Model-1'].rootAssembly.sets['Set-PressSurface']
232 mdb.models['Model-1'].historyOutputRequests['H-Output-1'].setValues(
233     variables=('U1', 'U2'), region=regionDef, sectionPoints=DEFAULT,
234     rebar=EXCLUDE)
235
236 # Submitting job
237 mdb.jobs['Job-' + NAME].submit(consistencyChecking=OFF)
238 mdb.jobs['Job-' + NAME].waitForCompletion()
239
240 # Opening results
241 session.viewports['Viewport: 1'].setValues(displayedObject=a)
242 o3 = session.openOdb(name='D:/abaqus/Job-' + NAME + '.odb')
243 session.viewports['Viewport: 1'].setValues(displayedObject=o3)
244 session.viewports['Viewport: 1'].makeCurrent()
```

B.1.2. Fluid Cavity modellezés

```
1 # ABAQUS script to generate the actuator with fluid cavity
2 # Usage: run this script from ABAQUS, File, Run Script
3 # Working with ABAQUS v. 6.20 (2020)
4 # Created by Barnabas Piri, 2022
5
6 from abaqus import *
7 from abaqusConstants import *
8 from numpy import *
9 import section
10 import regionToolset
11 import displayGroupMdbToolset as dgm
12 import part
13 import material
14 import assembly
15 import step
16 import interaction
17 import load
18 import mesh
19 import optimization
20 import job
21 import sketch
22 import visualization
23 import xyPlot
24 import displayGroupOdbToolset as dgo
25 import connectorBehavior
26
27 # Setting work directory
28 RUNDIR = "E:\\abaqus\\"
29 os.chdir(RUNDIR)
30
31 # Model name
32 JOBNAME = 'TOT_GEOM_fc'
33 repFileName = 'E:\\abaqus\\volume_pressure.rpt'
34
35 # Making the model
36 s = mdb.models['Model-1'].ConstrainedSketch(name='Sketch-1', sheetSize=60e-3)
```

```

37 g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
38 s.setPrimaryObject(option=STANDALONE)
39
40 # Outer cap' material (rubber - steel)
41 OUTER_CAP_RUBBER = True
42
43 # Inner cap Ogden model parameters
44 mu = 620000.0 # Pa
45 alpha = 1.3 # 1
46
47 # Inner cap geometric parameters
48 OP_ANGLE_inner = 70.0 # deg
49 A_inner = 20.0 # mm
50 B_inner = 18.0 # mm
51 TH_inner = 2.5 # mm
52
53 # Outer cap geometric parameters
54 OP_ANGLE_outer = 90.0 # deg
55 TH_outer = TH_inner / 2 # mm
56
57 # Ring parameters
58 TH_RING = TH_inner / 2 # mm
59
60 # CALCULATIONS
61
62
63 # Functions
64 def deg2rad(angle_deg):
65     angle_rad = angle_deg / 180 * pi
66     return angle_rad
67
68
69 # Convert the angles to radians
70 OP_ANGLE_inner = deg2rad(OP_ANGLE_inner)
71 OP_ANGLE_outer = deg2rad(OP_ANGLE_outer)
72
73 # Convert the length units to SI (meters)
74 A_inner = A_inner / 1000
75 B_inner = B_inner / 1000
76 TH_inner = TH_inner / 1000
77 TH_outer = TH_outer / 1000
78 TH_RING = TH_RING / 1000
79 #R_RING = R_RING / 1000
80
81 # Calculating the outer radius from the other parameters
82
83 r_ell = (A_inner * B_inner) / sqrt( (A_inner*sin(pi/2 - OP_ANGLE_inner))**2 +
84     (B_inner*cos(pi/2 - OP_ANGLE_inner))**2 )
85
86 Y6_CONST = r_ell * cos(OP_ANGLE_inner)
87
88 X6_CONST = sqrt( (1 - (Y6_CONST / (B_inner+TH_inner/2))**2 ) *
89     (A_inner + TH_inner/2)**2 )
90
91 A = 1 - cos(OP_ANGLE_outer)**2
92 B = -TH_outer
93 C = (TH_outer/2)**2 - X6_CONST**2
94
95 R_outer = (-B + sqrt(B**2 - 4*A*C)) / (2*A)
96
97 # Calculating the outer cap's excentricity
98 DELTA = r_ell*cos(OP_ANGLE_inner) - R_outer*cos(OP_ANGLE_outer)
99
100 # Coordinates of the points
101 P1 = [0, B_inner - TH_inner/2]

```

```

102 P2 = [0, B_inner + TH_inner/2]
103 P3 = [0, DELTA + R_outer - TH_outer/2]
104 P4 = [0, DELTA + R_outer + TH_outer/2]
105
106 P5 = [sqrt( (R_outer + TH_outer/2)**2 - (R_outer*cos(OP_ANGLE_outer))**2 ),
107       DELTA + R_outer*cos(OP_ANGLE_outer)]
108
109 P6_inner = [sqrt( (1 - (r_ell * cos(OP_ANGLE_inner) / (B_inner+TH_inner/2))**2 ) *
110              (A_inner + TH_inner/2)**2 ), r_ell*cos(OP_ANGLE_inner)]
111
112 P6_outer = [sqrt( (R_outer - TH_outer/2)**2 - (R_outer*cos(OP_ANGLE_outer))**2 ),
113             DELTA + R_outer*cos(OP_ANGLE_outer)]
114
115 try:
116
117     P7 = [sqrt( (1 - (r_ell * cos(OP_ANGLE_inner) / (B_inner - TH_inner/2))**2 ) *
118            (A_inner - TH_inner/2)**2 ), r_ell*cos(OP_ANGLE_inner)]
119
120     # Calculating the ring radius
121     R_RING = P5[0] *1.05
122
123     P8 = [R_RING, P7[1]]
124     P9 = [R_RING, P7[1] - TH_RING]
125     P10 = [sqrt( (1 - ((P7[1] - TH_RING) / (B_inner - TH_inner/2))**2 ) *
126            (A_inner - TH_inner/2)**2 ), P7[1] - TH_RING]
127
128     ORIGO_inner = [0, 0]
129     ORIGO_outer = [0, DELTA]
130
131     if P2[1] > P3[1]:
132         Mdb()
133
134     else:
135         # Sketching
136         s.EllipseByCenterPerimeter(center=ORIGO_inner,
137                                    axisPoint1=(A_inner + TH_inner/2, 0),
138                                    axisPoint2=(0, B_inner + TH_inner/2))
139
140         s.EllipseByCenterPerimeter(center=ORIGO_inner,
141                                    axisPoint1=(A_inner - TH_inner/2, 0),
142                                    axisPoint2=(0, B_inner - TH_inner/2))
143
144         s.ArcByCenterEnds(center=ORIGO_outer, point1=P6_inner, point2=P3,
145                           direction=COUNTERCLOCKWISE)
146         s.ArcByCenterEnds(center=ORIGO_outer, point1=P5, point2=P4,
147                           direction=COUNTERCLOCKWISE)
148
149         s.Line(point1=P8, point2=P5)
150         s.Line(point1=P8, point2=P9)
151         s.Line(point1=P10, point2=P9)
152         s.Line(point1=P1, point2=P2)
153         s.Line(point1=P3, point2=P4)
154
155         s.autoTrimCurve(curve1=g[2],
156                        point1=(P2[0]-0.1e-3, P2[1]-0.1e-3 ))
157         s.autoTrimCurve(curve1=g[4],
158                        point1=(P1[0]-0.1e-3, P1[1]-0.1e-3 ))
159         s.autoTrimCurve(curve1=g[14],
160                        point1=(A_inner - TH_inner/2, +0.1e-3 ))
161         s.autoTrimCurve(curve1=g[13],
162                        point1=(A_inner + TH_inner/2, +0.1e-3 ))
163         s.autoTrimCurve(curve1=g[16],
164                        point1=(P6_inner[0]+0.1e-3, P6_inner[1]-0.1e-3))
165
166         # Creating and partitioning model

```

```

167 s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
168                                             sheetSize=0.06)
169 g, v1, d1, c = s.geometry, s.vertices, s.dimensions, s.constraints
170 s.sketchOptions.setValues(decimalPlaces=3, viewStyle=AXISYM)
171 s.setPrimaryObject(option=STANDALONE)
172 s.ConstructionLine(point1=(0.0, -0.03), point2=(0.0, 0.03))
173 s.FixedConstraint(entity=g[2])
174 s.retrieveSketch(sketch=mdb.models['Model-1'].sketches['Sketch-1'])
175 session.viewports['Viewport: 1'].view.fitView()
176 p = mdb.models['Model-1'].Part(name='Part-AxiPart',
177                               dimensionality=AXISYMMETRIC, type=DEFORMABLE_BODY)
178 p = mdb.models['Model-1'].parts['Part-AxiPart']
179 p.BaseShell(sketch=s)
180 s.unsetPrimaryObject()
181 p = mdb.models['Model-1'].parts['Part-AxiPart']
182 session.viewports['Viewport: 1'].setValues(displayedObject=p)
183 del mdb.models['Model-1'].sketches['__profile__']
184 p = mdb.models['Model-1'].parts['Part-AxiPart']
185 f = p.faces
186 pickedFaces = f.getSequenceFromMask(mask=('#1 ', ), )
187 v2, e1, d2 = p.vertices, p.edges, p.datums
188 p.PartitionFaceByShortestPath(point1=v2[0], point2=v2[6],
189                               faces=pickedFaces)
190
191 # Assembly instance
192 a = mdb.models['Model-1'].rootAssembly
193 a.Instance(name='Part-AxiPart-1', part=p, dependent=ON)
194
195 # Cavity reference point
196 a.ReferencePoint(point=(0.0, (P2[1] + P3[1])/2, 0.0))
197 mdb.models['Model-1'].rootAssembly.features.changeKey(
198     fromName='RP-1', toName='RP-SurfCavPoint')
199
200 # Set for inner cap
201 f = p.faces
202 faces = f.getSequenceFromMask(mask=('#2 ', ), )
203 p.Set(faces=faces, name='Set-Part-InnerCap')
204 f1 = a.instances['Part-AxiPart-1'].faces
205 faces1 = f1.getSequenceFromMask(mask=('#2 ', ), )
206 a.Set(faces=faces1, name='Set-InnerCap')
207
208 # Set for outer cap
209 faces = f.getSequenceFromMask(mask=('#1 ', ), )
210 p.Set(faces=faces, name='Set-Part-OuterCap')
211 faces1 = f1.getSequenceFromMask(mask=('#1 ', ), )
212 a.Set(faces=faces1, name='Set-OuterCap')
213
214 # Set for fluid cavity surface
215 e1 = a.instances['Part-AxiPart-1'].edges
216 edges1 = e1.getSequenceFromMask(mask=('#18 ', ), )
217 a.Set(edges=edges1, name='Set-FluidCavSurface')
218
219 # Creating materials
220 mdb.models['Model-1'].Material(name='Material-OuterCap')
221
222 if OUTER_CAP_RUBBER:
223     mdb.models['Model-1'].materials['Material-OuterCap']
224         .Density(table=((1000.0, ), ))
225
226     mdb.models['Model-1'].materials['Material-OuterCap']
227         .Hyperelastic(materialType=ISOTROPIC, testData=OFF,
228             type=OGDEN, n=3, volumetricResponse=VOLUMETRIC_DATA,
229             table=((620000.0, 1.3, 1180.0, 5.0, -9810.0, -2.0,
230                 0.0, 0.0, 0.0), ))
231

```

```

232     else:
233         mdb.models['Model-1'].materials['Material-OuterCap']
234             .Density(table=((7800.0, ), ), )
235
236         mdb.models['Model-1'].materials['Material-OuterCap']
237             .Elastic(table=((200000000000.0, 0.3), ))
238
239     mdb.models['Model-1'].HomogeneousSolidSection(name='Section-OuterCap',
240         material='Material-OuterCap', thickness=None)
241
242     mdb.models['Model-1'].Material(name='Material-Rubber-InnerCap')
243     mdb.models['Model-1'].materials['Material-Rubber-InnerCap']
244         .Density(table=((1000.0, ), ), )
245     mdb.models['Model-1'].materials['Material-Rubber-InnerCap']
246         .Hyperelastic(materialType=ISOTROPIC, testData=OFF,
247             type=OGDEN, n=1, volumetricResponse=VOLUMETRIC_DATA,
248             table=((mu, alpha, 0.0), ))
249
250     mdb.models['Model-1'].HomogeneousSolidSection(
251         name='Section-Rubber-InnerCap',
252         material='Material-Rubber-InnerCap', thickness=None)
253
254     # Assign sections
255     region = p.sets['Set-Part-InnerCap']
256     p.SectionAssignment(region=region,
257         sectionName='Section-Rubber-InnerCap',
258         offset=0.0, offsetType=MIDDLE_SURFACE,
259         offsetField='', thicknessAssignment=FROM_SECTION)
260
261     region = p.sets['Set-Part-OuterCap']
262     p.SectionAssignment(region=region,
263         sectionName='Section-OuterCap',
264         offset=0.0, offsetType=MIDDLE_SURFACE,
265         offsetField='', thicknessAssignment=FROM_SECTION)
266
267     # Meshing
268     if (P4[1] - P3[1]) < (P2[1] - P1[1]):
269         MESHSIZE = (P4[1] - P3[1]) / 4
270     else:
271         MESHSIZE = (P2[1] - P1[1]) / 4
272
273     f = p.faces
274     pickedRegions = f.getSequenceFromMask(mask=('[#3 ]', ), )
275     p.setMeshControls(regions=pickedRegions, elemShape=QUAD,
276         algorithm=MEDIAL_AXIS)
277
278     elemType1 = mesh.ElemType(elemCode=CAX4H, elemLibrary=STANDARD)
279     elemType2 = mesh.ElemType(elemCode=CAX3, elemLibrary=STANDARD)
280     faces = f.getSequenceFromMask(mask=('[#3 ]', ), )
281     pickedRegions =(faces, )
282     p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
283     p.seedPart(size=MESHSIZE, deviationFactor=0.1, minSizeFactor=0.1)
284     p.generateMesh()
285
286     # Riks-Static step
287     mdb.models['Model-1'].StaticRiksStep(name='Step-Riks', previous='Initial',
288         maxNumInc=350, initialArcInc=1e-10, minArcInc=1e-32, maxArcInc=0.01,
289         nlgeom=ON)
290
291     # Fluid-Cavity properties
292     mdb.models['Model-1'].FluidCavityProperty(name='IntProp-FluidCav',
293         fluidDensity=1000.0, useBulkModulus=True, bulkModulusTable=((
294         2000000000.0, ), ))
295
296     s1 = a.instances['Part-AxiPart-1'].edges

```

```

297     side1Edges1 = s1.getSequenceFromMask(mask=('[#18 ]', ), )
298     a.Surface(side1Edges=side1Edges1, name='Surf-FluidCav')
299     r1 = a.referencePoints
300     refPoints1=(r1[3], )
301     region1=regionToolset.Region(referencePoints=refPoints1)
302     region2=a-surfaces['Surf-FluidCav']
303     mdb.models['Model-1'].FluidCavity(name='Int-FluidCav',
304     createStepName='Initial', cavityPoint=region1, cavitySurface=region2,
305     interactionProperty='IntProp-FluidCav')
306
307     # Boundary conditions
308     e1 = a.instances['Part-AxiPart-1'].edges # Symmetry on rotational axis
309     edges1 = e1.getSequenceFromMask(mask=('[#24 ]', ), )
310     region = regionToolset.Region(edges=edges1)
311     mdb.models['Model-1'].XsymmBC(name='BC-Symmetry',
312     createStepName='Initial',
313     region=region, localCsys=None)
314
315     v1 = a.instances['Part-AxiPart-1'].vertices # Roller support
316     verts1 = v1.getSequenceFromMask(mask=('[#1 ]', ), )
317     region = regionToolset.Region(vertices=verts1)
318     mdb.models['Model-1'].XasymBC(name='BC-Roller', createStepName='Initial',
319     region=region, localCsys=None)
320
321     session.viewports['Viewport: 1'].assemblyDisplay
322     .setValues(step='Step-Riks')
323
324     mdb.models['Model-1'].FluidCavityPressureBC(name='BC-Pressure',
325     createStepName='Step-Riks', magnitude=40000.0, fixed=OFF,
326     fluidCavity='Int-FluidCav', amplitude=UNSET)
327
328     # History output settings
329     r1 = a.referencePoints
330     refPoints1=(r1[3], )
331     a.Set(referencePoints=refPoints1, name='Set-SurfCavPoint')
332     regionDef=mdb.models['Model-1'].rootAssembly.sets['Set-SurfCavPoint']
333     mdb.models['Model-1'].HistoryOutputRequest(name='H-Output-FluidCav',
334     createStepName='Step-Riks', variables=('PCAV', 'CVOL'),
335     region=regionDef, sectionPoints=DEFAULT, rebar=EXCLUDE)
336
337     # CCreating the job
338     mdb.Job(name=JOBNAME, model='Model-1', description='',
339     type=ANALYSIS, atTime=None, waitMinutes=0, waitHours=0, queue=None,
340     memory=90, memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
341     explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE,
342     echoPrint=OFF, modelPrint=OFF, contactPrint=OFF, historyPrint=OFF,
343     userSubroutine='', scratch='', resultsFormat=ODB,
344     multiprocessingMode=DEFAULT, numCpus=3,
345     numDomains=3, numGPUs=0)
346
347     # Viewport fitting
348     session.viewports['Viewport: 1'].view.fitView()
349
350     # Submitting job
351     mdb.jobs[JOBNAME].submit(consistencyChecking=OFF)
352     mdb.jobs[JOBNAME].waitForCompletion()
353
354     # Opening results
355     o3 = session.openOdb(name=RUNDIR + JOBNAME + '.odb')
356     session.viewports['Viewport: 1'].setValues(displayedObject=o3)
357     session.viewports['Viewport: 1'].makeCurrent()
358     a = mdb.models['Model-1'].rootAssembly
359     session.viewports['Viewport: 1'].setValues(displayedObject=a)
360     session.mdbData.summary()
361     session.viewports['Viewport: 1'].setValues(

```

```
362         displayedObject=session.odbs[RUNDIR + JOBNAME + '.odb'])
363
364     session.viewports['Viewport: 1'].assemblyDisplay.setValues(
365         optimizationTasks=OFF, geometricRestrictions=OFF, stopConditions=OFF)
366     odb = session.odbs[RUNDIR + JOBNAME + '.odb']
367     xy1 = xyPlot.XYDataFromHistory(odb=odb,
368         outputVariableName='Fluid cavity pressure: PCAV PI: rootAssembly ...'
369         '... Node 1 in NSET SET-SURFCAVPOINT',
370         suppressQuery=True, __linkedVpName__='Viewport: 1')
371
372     c1 = session.Curve(xyData=xy1)
373     xy2 = xyPlot.XYDataFromHistory(odb=odb,
374         outputVariableName='Fluid cavity volume: CVOL PI: rootAssembly ...'
375         '... Node 1 in NSET SET-SURFCAVPOINT',
376         suppressQuery=True, __linkedVpName__='Viewport: 1')
377
378     c2 = session.Curve(xyData=xy2)
379     xyp = session.XYPlot('XYPlot-12')
380     chartName = xyp.charts.keys()[0]
381     chart = xyp.charts[chartName]
382     chart.setValues(curvesToPlot=(c1, c2, ), appendMode=True)
383     session.charts[chartName].autoColor(lines=True, symbols=True)
384     session.viewports['Viewport: 1'].setValues(displayedObject=xyp)
385     xy1 = session.xyDataObjects['_temp_2']
386     xy2 = session.xyDataObjects['_temp_1']
387     xy3 = combine(xy1, xy2)
388     xyp = session.xyPlots['XYPlot-12']
389     chartName = xyp.charts.keys()[0]
390     chart = xyp.charts[chartName]
391     c1 = session.Curve(xyData=xy3)
392     chart.setValues(curvesToPlot=(c1, ), )
393     session.charts[chartName].autoColor(lines=True, symbols=True)
394
395     # Savint reportfile
396     session.mdbData.summary()
397     xyp = session.xyPlots['XYPlot-12']
398     session.viewports['Viewport: 1'].setValues(displayedObject=xyp)
399     x0 = session.xyDataObjects['_temp_3']
400     session.writeXYReport(fileName=repFileName, xyData=(x0, ))
401
402 except:
403     Mdb()
```

B.2. Paramétervizsgálat

B.2.1. Paraméterek listája

```
1 # ABAQUS script to generate the parameter sets for simulations
2 # Usage: set the geometric data below
3 # Woking with ABAQUS v. 6.20 (2020)
4 # Created by Barnabas Piri, 2022
5
6
7 A_range = [20.0]
8 B_range = [18.0, 19.0]
9 OPA_inner_range = [70.0, 80.0]
10 OPA_outer_range = [90.0]
11 TH_inner_range = [2.5, 3.5]
12 muVec_range = [[62000]]
13 alphaVec_range = [[1.3]]
14
15 RUNDIR = "E:\\abaqus\\"
```

B.2.2. Paraméterek futtatása

```

1 # ABAQUS script to generate the actuator with fluid cavity
2 # Usage: run this script from ABAQUS CMD
3 # Woking with ABAQUS v. 6.20 (2020)
4 # Created by Barnabas Piri, 2022
5
6 from re import M
7 from abaqus import *
8 from abaqusConstants import *
9 from numpy import *
10
11 def total_geom_sets(A_inner, B_inner, OPA_inner, OPA_outer,
12                    TH_inner, muVec, alphaVec):
13     import section
14     import regionToolset
15     import displayGroupMdbToolset as dgm
16     import part
17     import material
18     import assembly
19     import step
20     import interaction
21     import load
22     import mesh
23     import optimization
24     import job
25     import sketch
26     import visualization
27     import xyPlot
28     import displayGroupOdbToolset as dgo
29     import connectorBehavior
30
31     # Setting work directory
32     from parameters import RUNDIR
33     os.chdir(RUNDIR)
34
35     # Model name
36     jbnm = "TOT_GEOM-" + str(B_inner) + "-" + str(OPA_inner) + "-" + str(TH_inner)
37           + "-" + str(muVec[0]) + "-" + str(alphaVec[0])
38     JOBNAME = jbnm
39     JOBNAME = JOBNAME.replace('.', 'x')
40     JOBNAME_ODB = JOBNAME + ".odb"
41     repFileName = JOBNAME + ".rpt"
42
43     print('B_inner: ', B_inner)
44     print('OPA_inner: ', OPA_inner)
45     print('TH_inner: ', TH_inner)
46
47     # Making the model
48     s = mdb.models['Model-1'].ConstrainedSketch(name='Sketch-1', sheetSize=60e-3)
49     g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
50     s.setPrimaryObject(option=STANDALONE)
51
52     # Outer cap's material (rubber - steel)
53     OUTER_CAP_RUBBER = True
54
55     # Inner cap Ogden model parameters
56     muVec = muVec
57     alphaVec = alphaVec
58
59     # Inner cap geometric parameters
60     OP_ANGLE_inner = OPA_inner # deg
61     A_inner = A_inner # mm
62     B_inner = B_inner # mm

```

```
63 TH_inner = TH_inner # mm
64
65 # Outer cap geometric parameters
66 OP_ANGLE_outer = OPA_outer # deg
67 TH_outer = TH_inner / 2 # mm
68
69 # Ring parameters
70 TH_RING = TH_inner / 2 # mm
71
72 # CALCULATIONS
73
74
75 # Functions
76 def deg2rad(angle_deg):
77     angle_rad = angle_deg / 180 * pi
78     return angle_rad
79
80 # Convert the angles to radians
81 OP_ANGLE_inner = deg2rad(OP_ANGLE_inner)
82 OP_ANGLE_outer = deg2rad(OP_ANGLE_outer)
83
84 # Convert the length units to SI (meters)
85 A_inner = A_inner / 1000
86 B_inner = B_inner / 1000
87 TH_inner = TH_inner / 1000
88 TH_outer = TH_outer / 1000
89 TH_RING = TH_RING / 1000
90
91 r_ell = (A_inner * B_inner) / sqrt( (A_inner*sin(pi/2 - OP_ANGLE_inner))**2 +
92     (B_inner*cos(pi/2 - OP_ANGLE_inner))**2 )
93
94 Y6_CONST = r_ell * cos(OP_ANGLE_inner)
95
96 X6_CONST = sqrt( (1 - (Y6_CONST / (B_inner+TH_inner/2))**2 ) *
97     (A_inner + TH_inner/2)**2 )
98
99 A = 1 - cos(OP_ANGLE_outer)**2
100 B = -TH_outer
101 C = (TH_outer/2)**2 - X6_CONST**2
102
103 R_outer = (-B + sqrt(B**2 - 4*A*C)) / (2*A)
104
105 # Calculating the outer cap's excentricity
106 DELTA = r_ell*cos(OP_ANGLE_inner) - R_outer*cos(OP_ANGLE_outer)
107
108 # Coordinates of the points
109 P1 = [0, B_inner - TH_inner/2]
110 P2 = [0, B_inner + TH_inner/2]
111 P3 = [0, DELTA + R_outer - TH_outer/2]
112 P4 = [0, DELTA + R_outer + TH_outer/2]
113
114 P5 = [sqrt( (R_outer + TH_outer/2)**2 - (R_outer*cos(OP_ANGLE_outer))**2 ),
115     DELTA + R_outer*cos(OP_ANGLE_outer)]
116
117 P6_inner = [sqrt( (1 - (r_ell * cos(OP_ANGLE_inner)/(B_inner+TH_inner/2))**2 ) *
118     (A_inner + TH_inner/2)**2 ), r_ell*cos(OP_ANGLE_inner)]
119
120 P6_outer = [sqrt( (R_outer - TH_outer/2)**2 - (R_outer*cos(OP_ANGLE_outer))**2 ),
121     DELTA + R_outer*cos(OP_ANGLE_outer)]
122
123 try:
124     P7 = [sqrt( (1 - (r_ell * cos(OP_ANGLE_inner)/(B_inner - TH_inner/2))**2 ) *
125     (A_inner - TH_inner/2)**2 ), r_ell*cos(OP_ANGLE_inner)]
126
127     # Calculating the ring radius
```

```

128 R_RING = P5[0] * 1.05
129
130 P8 = [R_RING, P7[1]]
131 P9 = [R_RING, P7[1] - TH_RING]
132 P10 = [sqrt((1 - ((P7[1] - TH_RING) / (B_inner - TH_inner/2))**2) *
133         (A_inner - TH_inner/2)**2), P7[1] - TH_RING]
134
135 ORIGO_inner = [0, 0]
136 ORIGO_outer = [0, DELTA]
137
138 if P2[1] > P3[1] or OP_ANGLE_inner >= OP_ANGLE_outer:
139     Mdb()
140
141 else:
142     # Sketching
143     s.EllipseByCenterPerimeter(center=ORIGO_inner,
144                               axisPoint1=(A_inner + TH_inner/2, 0),
145                               axisPoint2=(0, B_inner + TH_inner/2))
146
147     s.EllipseByCenterPerimeter(center=ORIGO_inner,
148                               axisPoint1=(A_inner - TH_inner/2, 0),
149                               axisPoint2=(0, B_inner - TH_inner/2))
150
151     s.ArcByCenterEnds(center=ORIGO_outer, point1=P6_inner, point2=P3,
152                      direction=COUNTERCLOCKWISE)
153     s.ArcByCenterEnds(center=ORIGO_outer, point1=P5, point2=P4,
154                      direction=COUNTERCLOCKWISE)
155
156     s.Line(point1=P8, point2=P5)
157     s.Line(point1=P8, point2=P9)
158     s.Line(point1=P10, point2=P9)
159     s.Line(point1=P1, point2=P2)
160     s.Line(point1=P3, point2=P4)
161
162     s.autoTrimCurve(curve1=g[2],
163                    point1=(P2[0]-0.1e-3, P2[1]-0.1e-3))
164     s.autoTrimCurve(curve1=g[4],
165                    point1=(P1[0]-0.1e-3, P1[1]-0.1e-3))
166     s.autoTrimCurve(curve1=g[14],
167                    point1=(A_inner - TH_inner/2, +0.1e-3))
168     s.autoTrimCurve(curve1=g[13],
169                    point1=(A_inner + TH_inner/2, +0.1e-3))
170     s.autoTrimCurve(curve1=g[16],
171                    point1=(P6_inner[0]+0.1e-3, P6_inner[1]-0.1e-3))
172
173     # Creating and partitioning model
174     s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
175                                                  sheetSize=0.06)
176     g, v1, d1, c = s.geometry, s.vertices, s.dimensions, s.constraints
177     s.sketchOptions.setValues(decimalPlaces=3, viewStyle=AXISYM)
178     s.setPrimaryObject(option=STANDALONE)
179     s.ConstructionLine(point1=(0.0, -0.03), point2=(0.0, 0.03))
180     s.FixedConstraint(entity=g[2])
181     s.retrieveSketch(sketch=mdb.models['Model-1'].sketches['Sketch-1'])
182     session.viewports['Viewport: 1'].view.fitView()
183     p = mdb.models['Model-1'].Part(name='Part-AxiPart',
184                                   dimensionality=AXISYMMETRIC, type=DEFORMABLE_BODY)
185     p = mdb.models['Model-1'].parts['Part-AxiPart']
186     p.BaseShell(sketch=s)
187     s.unsetPrimaryObject()
188     p = mdb.models['Model-1'].parts['Part-AxiPart']
189     session.viewports['Viewport: 1'].setValues(displayedObject=p)
190     del mdb.models['Model-1'].sketches['__profile__']
191     p = mdb.models['Model-1'].parts['Part-AxiPart']
192     f = p.faces

```

```

193     pickedFaces = f.getSequenceFromMask(mask=('[#1 ]', ), )
194     v2, e1, d2 = p.vertices, p.edges, p.datums
195     p.PartitionFaceByShortestPath(point1=v2[0], point2=v2[6],
196                                 faces=pickedFaces)
197
198     # Assembly instance
199     a = mdb.models['Model-1'].rootAssembly
200     a.Instance(name='Part-AxiPart-1', part=p, dependent=ON)
201
202     # Cavity reference point
203     a.ReferencePoint(point=(0.0, (P2[1] + P3[1])/2, 0.0))
204     mdb.models['Model-1'].rootAssembly.features.changeKey(
205         fromName='RP-1', toName='RP-SurfCavPoint')
206
207     # Set for the inner cap
208     f = p.faces
209     faces = f.getSequenceFromMask(mask=('[#2 ]', ), )
210     p.Set(faces=faces, name='Set-Part-InnerCap')
211     f1 = a.instances['Part-AxiPart-1'].faces
212     faces1 = f1.getSequenceFromMask(mask=('[#2 ]', ), )
213     a.Set(faces=faces1, name='Set-InnerCap')
214
215     # Set for the outer cap
216     faces = f.getSequenceFromMask(mask=('[#1 ]', ), )
217     p.Set(faces=faces, name='Set-Part-OuterCap')
218     faces1 = f1.getSequenceFromMask(mask=('[#1 ]', ), )
219     a.Set(faces=faces1, name='Set-OuterCap')
220
221     # Set for fluid cavity surface
222     e1 = a.instances['Part-AxiPart-1'].edges
223     edges1 = e1.getSequenceFromMask(mask=('[#18 ]', ), )
224     a.Set(edges=edges1, name='Set-FluidCavSurface')
225
226     # Set for pole point
227     v1 = a.instances['Part-AxiPart-1'].vertices
228     verts1 = v1.getSequenceFromMask(mask=('[#10 ]', ), )
229     a.Set(vertices=verts1, name='Set-Pole')
230
231     # Creating materials
232     mdb.models['Model-1'].Material(name='Material-OuterCap')
233
234     # Automatic Ogden parameter definition
235     ogdenN = len(muVec)
236     ogdenParams = []
237
238     for i in range(0, ogdenN):
239         muCurrent = muVec[i]
240         alphaCurrent = alphaVec[i]
241         ogdenParams.append(muVec[i])
242         ogdenParams.append(alphaVec[i])
243
244     ogdenParams.extend([0.0 for i in range(ogdenN)])
245     ogdenParams = tuple(ogdenParams)
246
247     if OUTER_CAP_RUBBER:
248         mdb.models['Model-1'].materials['Material-OuterCap']
249             .Density(table=((1000.0, ), ))
250
251         mdb.models['Model-1'].materials['Material-OuterCap']
252             .Hyperelastic(materialType=ISOTROPIC, testData=OFF,
253                 type=OGDEN, n=3, volumetricResponse=VOLUMETRIC_DATA,
254                 table=((620000.0, 1.3, 1180.0, 5.0, -9810.0, -2.0,
255                     0.0, 0.0, 0.0), ))
256
257     else:

```

```

258         mdb.models['Model-1'].materials['Material-OuterCap']
259             .Density(table=((7800.0, ), ), )
260
261         mdb.models['Model-1'].materials['Material-OuterCap']
262             .Elastic(table=((200000000000.0, 0.3), ))
263
264     mdb.models['Model-1'].HomogeneousSolidSection(name='Section-OuterCap',
265         material='Material-OuterCap', thickness=None)
266
267     mdb.models['Model-1'].Material(name='Material-Rubber-InnerCap')
268     mdb.models['Model-1'].materials['Material-Rubber-InnerCap']
269         .Density(table=((1000.0, ), ), )
270
271     mdb.models['Model-1'].materials['Material-Rubber-InnerCap']
272         .Hyperelastic(materialType=ISOTROPIC, testData=OFF,
273             type=OGDEN, n=ogdenN, volumetricResponse=VOLUMETRIC_DATA,
274             table=(ogdenParams, ))
275
276     mdb.models['Model-1'].HomogeneousSolidSection(
277         name='Section-Rubber-InnerCap',
278         material='Material-Rubber-InnerCap',
279         thickness=None)
280
281
282     # Assign sections
283     region = p.sets['Set-Part-InnerCap']
284     p.SectionAssignment(region=region,
285         sectionName='Section-Rubber-InnerCap',
286         offset=0.0, offsetType=MIDDLE_SURFACE,
287         offsetField='',
288         thicknessAssignment=FROM_SECTION)
289
290     region = p.sets['Set-Part-OuterCap']
291     p.SectionAssignment(region=region,
292         sectionName='Section-OuterCap',
293         offset=0.0, offsetType=MIDDLE_SURFACE,
294         offsetField='',
295         thicknessAssignment=FROM_SECTION)
296
297     # Meshing
298     if (P4[1] - P3[1]) < (P2[1] - P1[1]):
299         MESHSIZE = (P4[1] - P3[1]) / 4
300     else:
301         MESHSIZE = (P2[1] - P1[1]) / 4
302
303     f = p.faces
304     pickedRegions = f.getSequenceFromMask(mask=('[#3 ]', ), )
305     p.setMeshControls(regions=pickedRegions, elemShape=QUAD,
306         algorithm=MEDIAL_AXIS)
307
308     elemType1 = mesh.ElemType(elemCode=CAX4H, elemLibrary=STANDARD)
309     elemType2 = mesh.ElemType(elemCode=CAX3, elemLibrary=STANDARD)
310     faces = f.getSequenceFromMask(mask=('[#3 ]', ), )
311     pickedRegions = (faces, )
312     p.setElementType(regions=pickedRegions,
313         elemTypes=(elemType1, elemType2))
314     p.seedPart(size=MESHSIZE, deviationFactor=0.1, minSizeFactor=0.1)
315     p.generateMesh()
316
317     # Riks-Static step
318     poleHeight = P2[1] - P5[1]
319     print('poleHeight: ', poleHeight)
320     regionDef=mdb.models['Model-1'].rootAssembly.sets['Set-Pole']
321     mdb.models['Model-1'].StaticRiksStep(name='Step-Riks',
322         previous='Initial', nodeOn=ON,

```

```

323         maximumDisplacement=-2*poleHeight, region=regionDef,
324         dof=2, maxNumInc=5000, initialArcInc=1e-10, minArcInc=1e-32,
325         maxArcInc=0.01, nlgeom=ON)
326
327     # Fluid-Cavity properties
328     mdb.models['Model-1'].FluidCavityProperty(name='IntProp-FluidCav',
329     fluidDensity=1000.0, useBulkModulus=True,
330     bulkModulusTable=((20000000000.0, ), ))
331     s1 = a.instances['Part-AxiPart-1'].edges
332     side1Edges1 = s1.getSequenceFromMask(mask=('[#18 ]', ), )
333     a.Surface(side1Edges=side1Edges1, name='Surf-FluidCav')
334     r1 = a.referencePoints
335     refPoints1=(r1[3], )
336     region1=regionToolset.Region(referencePoints=refPoints1)
337     region2=a-surfaces['Surf-FluidCav']
338     mdb.models['Model-1'].FluidCavity(name='Int-FluidCav',
339     createStepName='Initial', cavityPoint=region1, cavitySurface=region2,
340     interactionProperty='IntProp-FluidCav')
341
342     # Boundary conditions
343     e1 = a.instances['Part-AxiPart-1'].edges
344     edges1 = e1.getSequenceFromMask(mask=('[#24 ]', ), )
345     region = regionToolset.Region(edges=edges1)
346     mdb.models['Model-1'].XsymmBC(name='BC-Symmetry',
347     createStepName='Initial',
348     region=region, localCsys=None)
349
350     v1 = a.instances['Part-AxiPart-1'].vertices
351     verts1 = v1.getSequenceFromMask(mask=('[#1 ]', ), )
352     region = regionToolset.Region(vertices=verts1)
353     mdb.models['Model-1'].XasymmBC(name='BC-Roller',
354     createStepName='Initial',
355     region=region, localCsys=None)
356
357     session.viewports['Viewport: 1'].assemblyDisplay
358     .setValues(step='Step-Riks')
359
360     mdb.models['Model-1'].FluidCavityPressureBC(name='BC-Pressure',
361     createStepName='Step-Riks', magnitude=40000.0, fixed=OFF,
362     fluidCavity='Int-FluidCav', amplitude=UNSET)
363
364     # History output settings
365     r1 = a.referencePoints
366     refPoints1=(r1[3], )
367     a.Set(referencePoints=refPoints1, name='Set-SurfCavPoint')
368     regionDef=mdb.models['Model-1'].rootAssembly.sets['Set-SurfCavPoint']
369     mdb.models['Model-1'].HistoryOutputRequest(name='H-Output-FluidCav',
370     createStepName='Step-Riks', variables=('[PCAV', 'CVOL']),
371     region=regionDef, sectionPoints=DEFAULT, rebar=EXCLUDE)
372
373     # Creating the job
374     mdb.Job(name=JOBNAME, model='Model-1', description='',
375     type=ANALYSIS, atTime=None, waitMinutes=0, waitHours=0,
376     queue=None, memory=90, memoryUnits=PERCENTAGE,
377     getMemoryFromAnalysis=True, explicitPrecision=SINGLE,
378     nodalOutputPrecision=SINGLE, echoPrint=OFF,
379     modelPrint=OFF, contactPrint=OFF, historyPrint=OFF,
380     userSubroutine='', scratch='', resultsFormat=ODB,
381     multiprocessingMode=DEFAULT, numCpus=3,
382     numDomains=3, numGPUs=0)
383
384     # Viewport fitting
385     session.viewports['Viewport: 1'].view.fitView()
386
387     # Submitting job

```

```

388         mdb.jobs[JOBNAME].submit(consistencyChecking=OFF)
389         mdb.jobs[JOBNAME].waitForCompletion()
390
391         # Start new model
392         Mdb()
393         session.viewports['Viewport: 1'].setValues(displayedObject=None)
394     except:
395         Mdb()
396
397
398 # Import parameters
399 from parameters import *
400
401 # Run the simulation for all combination of the parameters
402 for A in A_range:
403     for B in B_range:
404         for OPAi in OPA_inner_range:
405             for OPAo in OPA_outer_range:
406                 for THi in TH_inner_range:
407                     for mu in muVec_range:
408                         for alpha in alphaVec_range:
409                             total_geom_sets(A, B, OPAi, OPAo, THi, mu, alpha)

```

B.2.3. Adatbázisok mentése

```

1 # Script to save results from the databases
2 # Usage: run this script from ABAQUS CMD
3 # Woking with ABAQUS v. 6.20 (2020)
4 # Created by Barnabas Piri, 2022
5
6 def total_geom_sets_save(A_inner, B_inner, OPA_inner,
7                           OPA_outer, TH_inner, muVec, alphaVec):
8     import section
9     import regionToolset
10    import displayGroupMdbToolset as dgm
11    import part
12    import material
13    import assembly
14    import step
15    import interaction
16    import load
17    import mesh
18    import optimization
19    import job
20    import sketch
21    import visualization
22    import xyPlot
23    import displayGroupOdbToolset as dgo
24    import connectorBehavior
25
26    # Setting work directory
27    from parameters import RUNDIR
28
29    os.chdir(RUNDIR)
30
31
32    # MODEL NAME
33    jbnm = "TOT_GEOM-" + str(B) + "-" + str(OPAi) + "-" + str(THi)
34          + "-" + str(mu[0]) + "-" + str(alpha[0])
35
36    JOBNAME = RUNDIR + jbnm
37    JOBNAME = JOBNAME.replace('.', 'x')
38    JOBNAME_ODB = JOBNAME + ".odb"
39    repFileName = JOBNAME + ".rpt"

```

```

40
41     try:
42         session.mdbData.summary()
43         o1 = session.openOdb(name=JOBNAME_ODB)
44         odb = session.odbs[JOBNAME_ODB]
45
46         xy_pressure = xyPlot.XYDataFromHistory(odb=odb,
47         outputVariableName='Fluid cavity pressure: PCAV PI:'...
48         ' rootAssembly Node 1 in NSET SET-SURFCAVPOINT',
49         steps=('Step-Riks', ), suppressQuery=True,
50         __linkedVpName__='Viewport: 1')
51
52         xy_volume = xyPlot.XYDataFromHistory(odb=odb,
53         outputVariableName='Fluid cavity volume: CVOL PI:'...
54         ' rootAssembly Node 1 in NSET SET-SURFCAVPOINT',
55         steps=('Step-Riks', ), suppressQuery=True,
56         __linkedVpName__='Viewport: 1')
57
58         xy_pv_curve = combine(xy_volume, xy_pressure, )
59
60         session.XYData(name='pv_curve', objectToCopy=xy_pv_curve,
61         sourceDescription='combine(pressure_volume, pressure_volume, )'
62         )
63
64         del session.xyDataObjects[xy_pressure.name]
65         del session.xyDataObjects[xy_volume.name]
66         del session.xyDataObjects[xy_pv_curve.name]
67
68         x0 = session.xyDataObjects['pv_curve']
69
70         session.writeXYReport(fileName=repFileName, xyData=(x0, ))
71
72     except:
73         Mdb()
74
75 from parameters import *
76
77 for A in A_range:
78     for B in B_range:
79         for OPAi in OPA_inner_range:
80             for OPAo in OPA_outer_range:
81                 for THi in TH_inner_range:
82                     for mu in muVec_range:
83                         for alpha in alphaVec_range:
84                             total_geom_sets_save(A, B, OPAi, OPAo, THi, mu, alpha)

```

B.2.4. Kiértékelés

```

1 # Script to evaluate results
2 # Usage: run this python script after saving the databases
3 # Working with ABAQUS v. 6.20 (2020)
4 # Created by Barnabas Piri, 2022
5
6 import pandas as pd
7 import matplotlib.pyplot as plt
8 from numpy import *
9 import scipy as scp
10 import csv
11
12 def pole_dist(A, B, OPAi, OPAo, THi):
13
14     # Inner cap geometric parameters
15     OP_ANGLE_inner = OPAi # deg
16     A_inner = A # mm

```

```

17 B_inner = B # mm
18 TH_inner = THi # mm
19
20 # Outer cap geometric parameters
21 OP_ANGLE_outer = OPAo # deg
22 TH_outer = TH_inner / 2 # mm Constant for now
23
24 # Ring parameters
25 TH_RING = TH_inner / 2 # mm Constant for now
26
27 # Calculations
28
29 def deg2rad(angle_deg):
30     angle_rad = angle_deg / 180 * pi
31     return angle_rad
32
33 # Convert the angles to radians
34 OP_ANGLE_inner = deg2rad(OP_ANGLE_inner)
35 OP_ANGLE_outer = deg2rad(OP_ANGLE_outer)
36
37 # Convert the length units to SI (meters)
38 A_inner = A_inner / 1000
39 B_inner = B_inner / 1000
40 TH_inner = TH_inner / 1000
41 TH_outer = TH_outer / 1000
42 TH_RING = TH_RING / 1000
43
44 r_ell = (A_inner * B_inner) / sqrt( (A_inner*sin(pi/2 - OP_ANGLE_inner))**2 +
45     (B_inner*cos(pi/2 - OP_ANGLE_inner))**2 )
46
47 Y6_CONST = r_ell * cos(OP_ANGLE_inner)
48
49 X6_CONST = sqrt( (1 - (Y6_CONST / (B_inner+TH_inner/2))**2 ) *
50     (A_inner + TH_inner/2)**2 )
51
52 A = 1 - cos(OP_ANGLE_outer)**2
53 B = -TH_outer
54 C = (TH_outer/2)**2 - X6_CONST**2
55
56 R_outer = (-B + sqrt(B**2 - 4*A*C)) / (2*A)
57
58 # Calculating the outer cap's excentricity
59 DELTA = r_ell*cos(OP_ANGLE_inner) - R_outer*cos(OP_ANGLE_outer)
60
61 # Coordinates of the points
62 P2 = [0, B_inner + TH_inner/2]
63 P3 = [0, DELTA + R_outer - TH_outer/2]
64 poleDist = P3[1] - P2[1]
65
66 return poleDist
67
68 def snapback_plot(filename):
69     reportFile = pd.read_fwf(filename, delimiter=' ', infer_nrows=1000)
70     reportFile = reportFile.drop(0) # dropping header
71     reportFile = reportFile.to_numpy() # convert to numpy array
72
73     x = reportFile[:,0] # separating volume
74     y = reportFile[:,1] # separating pressure
75     mu_0 = 0
76
77     for i in range(0, len(mu)):
78         mu_0 = mu_0 + mu[i]
79
80     x = x / (A/1000)**3 # Dimensionless data
81     y = y / (mu_0)

```

```

82
83 plt.grid(True)
84 plt.xlabel('Normalized Volume (1)')
85 plt.ylabel('Normalized Pressure (1)')
86
87 # Search algorithm for snapping
88
89 FIRST_SNAP = False # bools for snapping search
90 SECOND_SNAP = False
91
92 length = len(x) # getting the length of the data
93 remaining_x = [] # remaining datavectors to store intermediate information
94 remaining_y = []
95
96 for i in range(length-1):
97
98     # searching for first snap, and its index
99     if FIRST_SNAP == False and x[i+1] < x[i]:
100         FIRST_SNAP = True
101         first_snap_x = x[i]
102         first_snap_y = y[i]
103         index_of_first_snap = where(x == first_snap_x)[0][0]
104         index_of_first_snap = index_of_first_snap + 1
105
106     # searching for second snap, and its index
107     if SECOND_SNAP == False and FIRST_SNAP == True and x[i+1] > x[i]:
108         SECOND_SNAP = True
109         second_snap_x = x[i]
110         second_snap_y = y[i]
111         index_of_second_snap = where(x == second_snap_x)[0][0]
112
113     # searching for third snap, and its index
114     if SECOND_SNAP == True:
115         remaining_x.append(x[i])
116         third_snap_x = min(remaining_x, key=lambda x:abs(x-first_snap_x))
117         index_of_third_snap = where(x == third_snap_x)[0][0]
118         third_snap_y = y[index_of_third_snap]
119
120 if FIRST_SNAP == True and SECOND_SNAP == True: # plotting the snap-through
121     #print(f"first x: {index_of_first_snap}")
122     #print(f"second x: {index_of_second_snap}")
123     #print(f"third x: {index_of_third_snap}")
124
125 plt.grid(True) # setting up for plots
126 plt.xlabel('Normalized Volume (1)')
127 plt.ylabel('Normalized Pressure (1)')
128
129 plt.scatter(first_snap_x, first_snap_y, c='red')
130 plt.scatter(second_snap_x, second_snap_y, c='red')
131 plt.scatter(third_snap_x, third_snap_y, c='red')
132
133 v1 = x[0:index_of_first_snap]
134
135 v2 = x[index_of_first_snap-1:index_of_second_snap+1]
136
137 v3 = x[index_of_second_snap:index_of_third_snap+1]
138
139 p1 = y[0:index_of_first_snap]
140
141 p2 = y[index_of_first_snap-1:index_of_second_snap+1]
142
143 p3 = y[index_of_second_snap:index_of_third_snap+1]
144
145
146 v_tot = x[0:index_of_third_snap]

```

```

147     p_tot = y[0:index_of_third_snap]
148
149
150     plt.plot(v1, p1, c='red')
151     plt.plot(v2, p2, c='green')
152     plt.plot(v3, p3, c='blue')
153
154
155     # integrate under the curves to get the energy
156     int1 = scp.integrate.simps(p1, x=v1, even='avg')
157     int2 = -scp.integrate.simps(p2, x=v2, even='avg') # negative
158     int3 = scp.integrate.simps(p3, x=v3, even='avg')
159
160     tot_energy = int2 - int3
161     # when a figure pops up, save the corresponding data
162     # with 'ENTER' or discard with 'Q'
163
164     def close_figure(event):
165         if event.key == 'enter':
166             print('Adding data to export.')
167             poleDist = pole_dist(A, B, OPAi, OPAo, THi)
168             current_data = [B, OPAi, THi, tot_energy, poleDist]
169             data_ext.append(current_data)
170             plt.close(event.canvas.figure)
171         if event.key == 'q':
172             print('DEGENERATE.')
173             current_data = [B, OPAi, THi, tot_energy]
174             degen_ext.append(current_data)
175             plt.close(event.canvas.figure)
176
177     #event listener for closing
178     plt.gcf().canvas.mpl_connect('key_press_event', close_figure)
179     plt.show()
180
181     with open('data_ext.csv', 'w', newline='') as f: # writing the data
182         write = csv.writer(f)
183         write.writerows(data_ext)
184
185     with open('degen_ext.csv', 'w', newline='') as f:
186         write = csv.writer(f)
187         write.writerows(degen_ext)
188
189
190 from parameters import * # importing the parameters
191
192 # header of the exported datasheets
193 data_ext = [['B', 'OPAi', 'THi', 'tot_energy', 'pole_distance']]
194 degen_ext = [['degenerate curves'], ['B', 'OPAi', 'THi', 'tot_energy']]
195
196 # running the evaluation for every one of the simulation results
197 for A in A_range:
198     for B in B_range:
199         for OPAi in OPA_inner_range:
200             for OPAo in OPA_outer_range:
201                 for THi in TH_inner_range:
202                     for mu in muVec_range:
203                         for alpha in alphaVec_range:
204                             try:
205                                 print('--- NEW REPORT FILE ---')
206                                 print(f'B: {B}, OPAi: {OPAi}, THi: {THi}')
207                                 jbnm = "TOT_GEOM-" + str(B) + "-" + str(OPAi) +
208                                     "-" + str(THi) + "-" + str(mu[0]) + "-" +
209                                     str(alpha[0])
210
211                                 JOBNAME = RUNDIR + jbnm

```

```
212         JOBNAME = JOBNAME.replace('.', 'x')
213         repFileName = JOBNAME + ".rpt"
214         snapback_plot(repFileName)
215         print('\n')
216     except:
217         print('Report file can not be found:')
218         print(repFileName)
```


Irodalom

- [1] Deepak Trivedi, Christopher D. Rahn, William M. Kier és Ian D. Walker. „Soft robotics: Biological inspiration, state of the art, and future research”. *Applied Bionics and Biomechanics* 5.3 (2008. dec.), 99–117. old.
- [2] Johannes T. B. Overvelde, Tamara Kloeck, Jonas J. A. D’haen és Katia Bertoldi. „Amplifying the response of soft actuators by harnessing snap-through instabilities”. *Proceedings of the National Academy of Sciences* 112.35 (2015. aug.), 10863–10868. old.
- [3] Benjamin Gorissen, David Melancon, Nikolaos Vasios, Mehdi Torbati és Katia Bertoldi. „Inflatable soft jumper inspired by shell snapping”. *Science Robotics* 5.42 (2020. máj.).
- [4] Dassault Systèmes. *Abaqus version 6.20*. 2020.
- [5] Berezvai Szabolcs. „Constitutive modelling of compressible solids including viscoelastic-viscoplastic effects”. Doktori disszertáció. Budapesti Műszaki és Gazdaságtudományi Egyetem, Budapest, 2020.
- [6] Gerhard A. Holzapfel. „Nonlinear Solid Mechanics: A Continuum Approach for Engineering”. *Meccanica* 37.4/5 (2002), 489–490. old.
- [7] Allan F. Bower. *Applied Mechanics of Solids*. CRC Press, 2009. okt.
- [8] Jörgen Bergström. *Mechanics of Solid Polymers*. Elsevier, 2015.
- [9] Aniket Pal, Vanessa Restrepo, Deekalpa Goswami és Ramses V. Martinez. „Exploiting Mechanical Instabilities in Soft Robotics: Control, Sensing, and Actuation”. *Advanced Materials* 33.19 (2021. márc.), 2006939. old.
- [10] Yichao Tang, Yinding Chi, Jiefeng Sun, Tzu-Hao Huang, Omid H. Maghsoudi, Andrew Spence, Jianguo Zhao, Hao Su és Jie Yin. „Leveraging elastic instabilities for amplified performance: Spine-inspired high-speed and high-force soft robots”. *Science Advances* 6.19 (2020. máj.).
- [11] Zechen Xiong, Yufeng Su és Hod Lipson. *Fast Untethered Soft Robotic Crawler with Elastic Instability*. 2022.
- [12] Nan Hu és Rigoberto Burgueño. „Buckling-induced smart applications: recent advances and trends”. *Smart Materials and Structures* 24.6 (2015. máj.), 63001. old.

- [13] Nicholas W. Bartlett, Michael T. Tolley, Johannes T. B. Overvelde, James C. Weaver, Bobak Mosadegh, Katia Bertoldi, George M. Whitesides és Robert J. Wood. „A 3D-printed, functionally graded soft robot powered by combustion”. *Science* 349.6244 (2015. júl.), 161–165. old.
- [14] Carmel Majidi. „Soft Robotics: A Perspective—Current Trends and Prospects for the Future”. *Soft Robotics* 1.1 (2014. márc.), 5–11. old.
- [15] Nikolaos Vassios. *Nonlinear Analysis of Structures*. 2015.
- [16] Bashir-Ahmed Memon és Xiao-zu Su. „Arc-length technique for nonlinear finite element analysis”. *Journal of Zhejiang University-Science A* 5.5 (2004. márc.), 618–628. old.
- [17] M. A. Crisfield. „A fast incremental/iterative solution procedure that handles snap-through”. *Computers and Structures* 13.1-3 (1981. jún.), 55–62. old.
- [18] L. R. G. Treloar. „Stress-strain data for vulcanised rubber under various types of deformation”. *Transactions of the Faraday Society* 40 (1944), 59. old.