

# CHAPTER 1

## THE THIRD

### 1.1 Standard Linear Solid Model Curve Fit

</> Script 1: Python script used to fit a two-term standard linear solid model to creep </>  
data.

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Fri Nov 20 14:20:21 2020
4
5 @author: Kiffer
6 """
7
8 import lmfit as lf # lmfit
9 import numpy as np # numpy
10 import matplotlib.pyplot as plt
11 plt.rcParams['figure.figsize'] = [16, 9]
12 import pandas as pd
13 import os
14 import sys
15 import pdb
16 filePath = os.getcwd() # Location of Python script
17
18 dataSets = ['Jami', 'Lee', 'Kashani', 'Polymer']
19 dataSet = 1
20 dataSet = dataSets[dataSet]
21
22 def PronyR2(y, fit):
23     # R squared calculation
24     SS_tot = np.sum((y - np.mean(y))**2)
25     SS_res = np.sum((y - fit)**2)
26     Rsqd = 1 - SS_res/SS_tot
27     return Rsqd
28
29 # In[Jami Data for example]
30 if dataSet == 'Jami':
31     ##### Jami Creep Shear Test Data Curve Fit#####
32     dataName = 'Jami 2014'
33     Jami_data = pd.read_csv('Jami/Jami_Shear_Data.txt', sep="\t", header=0)
34     Jami_data.columns = ["Time", "NormShearCreep"]
35     Jami_data['NormRelaxData'] = 1/Jami_data.NormShearCreep
36
37     # Invert and compute the raw data
38     Raw_data_0 = 0.0214671 # first shear point in raw data
39     Jami_data['CreepData'] = Jami_data.NormShearCreep*Raw_data_0
40     Jami_data['RelaxData'] = 1/Jami_data['CreepData']
```

```

41
42     # Convert data to array
43     t = Jami_data.Time
44     # data = Jami_data.NormShearCreep
45     # data = Jami_data.NormRelaxData # Normalized relaxation data
46     data = Jami_data.CreepData
47     # data = Jami_data.RelaxData
48
49     # Units for plotting
50     CreepUnits = r'$\left(\mathrm{Pa}\right)$'
51     RelaxUnits = r'$\left(\frac{1}{\mathrm{Pa}}\right)$'
52     HorizontalUnits = r'Time (s)'
53
54 # In[Lee Digitized Data]
55
56 if dataSet == 'Lee':
57     # Read data from Lee1992 Viscoelastic material properties
58     # Digitized from http://getdata-graph-digitizer.com/
59     dataName = 'Lee 1992'
60     Lee_data = pd.read_csv('Lee1992_DigitizedData.txt', sep="\t", header=3)
61     Lee_data.columns = ["Time", "Compliance"]
62
63     t = Lee_data.Time
64
65     # Conversion is dyne/cm^2 to Pa is multiply by 0.1
66     data = Lee_data.Compliance*0.1
67
68     # Units for plotting
69     CreepUnits = r'$\left(\mathrm{Pa}\right)$'
70     RelaxUnits = r'$\left(\frac{1}{\mathrm{Pa}}\right)$'
71     HorizontalUnits = r'Time (s)'
72
73 # In[Kashani_2011]
74 if dataSet == 'Kashani':
75     # Porcine eyes
76     t = np.linspace(1, 600, 1000)
77
78     def modelEqn(t, J1, J2, T1, T2, eta_m):
79         """
80         Parameters
81         -----
82         J1 : TYPE
83             DESCRIPTION.
84         J2 : TYPE
85             DESCRIPTION.
86         T1 : TYPE
87             DESCRIPTION.
88         T2 : TYPE
89             DESCRIPTION.
90         eta_m : TYPE
91             DESCRIPTION.
92
93         Returns
94         -----
95         Compliance
96         """
97         return J1*(1 - np.exp(-t/T1)) + J2*(1 - np.exp(-t/T2)) + t/eta_m
98

```

```

99 J1,J2,T1,T2,eta_m = 1.36, 2.64, 1.77, 1.36, 1332.0
100 J_0 = modelEqn(1,J1,J2,T1,T2,eta_m) # Initial value
101 data = 1/(modelEqn(t,J1,J2,T1,T2,eta_m)/J_0) # invert to define relaxation
102 # I'm not sure how this equation works
103
104 # Read data from Lee1992 Viscoelastic material properties
105 # Digitized from http://getdata-graph-digitizer.com/
106 dataName = 'Kashani 2011'
107 Kashani_data = pd.read_csv('KashaniCreepData.csv', sep=",", header=1)
108 Kashani_data.columns = ["Time1", "Compliance1",
109                        "Time2", "Compliance2",
110                        "Time3", "Compliance3"]
111 t = Kashani_data.Time1.dropna() # get rid of NaN
112 Compliance = Kashani_data.Compliance1.dropna() # Pa # get rid of NaN
113 t = Kashani_data.Time2.dropna() # get rid of NaN
114 Compliance = Kashani_data.Compliance2.dropna() # Pa # get rid of NaN
115 # t = Kashani_data.Time3.dropna() # get rid of NaN
116 # Compliance = Kashani_data.Compliance3.dropna() # Pa # get rid of NaN
117
118 data = Compliance
119
120 # Units for plotting
121 CreepUnits = r'$\left(\mathrm{Pa}\right)$'
122 RelaxUnits = r'$\left(\frac{1}{\mathrm{Pa}}\right)$'
123 HorizontalUnits = r'Time (s)'
124
125 # In[polymer data]
126
127 if dataSet == 'Polymer':
128     ##### Creep Shear Test Data Curve Fit#####
129     dataName = 'Polymer 2019'
130     subFolder = 'Polymer'
131     # df = pd.read_csv(os.path.join(filePath, subFolder,
132     #                               'Dogbone1_Test1.csv'),
133     #                  sep=",", header=5)
134     df = pd.read_csv(os.path.join(filePath, subFolder,
135     #                               'Dogbone_1_StressRelaxation_2.csv'),
136     #                  sep=",", header=5)
137     # df = pd.read_csv(os.path.join(filePath, subFolder,
138     #                               'Dogbone1_StressRelaxation_3.csv'),
139     #                  sep=",", header=5)
140     df.columns = ["Time", "Extension","Load"]
141
142     # Convert dataframe to array
143     time = np.asarray(df['Time'].tolist())
144     extension = np.asarray(df['Extension'].tolist())
145     load = np.asarray(df['Load'].tolist())
146
147     # Specimen properties
148     width = 12/1000.0
149     thickness = 5.3/1000.0
150
151     # calculations
152     stress = load/(width*thickness)
153     strain = extension/100
154
155     # Determine where the stress begins to decrease from the max point in
156     # the array

```

```

157 StressRelax=stress[np.argmax(stress):-1];
158 TimeRelax=time[np.argmax(stress):-1]-time[np.argmax(stress)]
159
160 data = 1/StressRelax
161 t = TimeRelax
162
163 # Units for plotting
164 CreepUnits = r'$\left(\mathrm{Pa}\right)$'
165 RelaxUnits = r'$\left(\frac{1}{\mathrm{Pa}}\right)$'
166 HorizontalUnits = r'Displacement (x)'
167
168 # In[Lmfit]
169
170 def residual(pars, t, data=None):
171     """
172     Parameters
173     -----
174     pars : Ee and E1 and tau2 for Visco terms for the Standard Linear Solid
175           Model from Lin2020
176     t : time array being passed in
177     data : Data to be passed
178           through to compare to model data. The default is None.
179
180     Returns
181     -----
182     If no data is supplied, the return is the new model for plotting the final
183     curve
184
185     If data is supplied it will calculate the error between the actual
186     and known data
187     """
188
189     Ee = pars['Ee'].value # Instantaneous modulus
190     E1 = pars['E1'].value # Instantaneous modulus
191     tau2 = pars['tau2'].value # Time constant
192
193     # Standard Linear Solid Model (SLSM)
194     model = 1/Ee*(1 - E1/(E1 + Ee)*np.exp(-t/tau2))
195
196     if data is None:
197         return model
198     return model - data
199
200 def dresidual(pars, t, data=None):
201     """
202     Derivative of the function to return the jacobian for faster optimization
203     Parameters
204     -----
205     pars : Ee and E1 and tau2 for Visco terms for the Standard Linear Solid
206           Model from Lin2020
207     t : time array being passed in
208     data : Normalized relaxation data (1/creep compliance) to be passed
209           through to compare to model data. The default is None.
210
211     Returns
212     -----
213     The jacobian (partial derivatives with respect to unknown variables)
214     """

```

```

215 Ee = pars['Ee'].value # Instantaneous modulus
216 E1 = pars['E1'].value # Instantaneous modulus
217 tau2 = pars['tau2'].value # Time constant
218
219 jac = []
220
221 dCdEe = ((E1*Ee + (E1 + Ee)*(E1 - (E1 + Ee)*np.exp(t/tau2)))*
222          np.exp(-t/tau2)/(Ee**2*(E1 + Ee)**2))
223 dCdE1 = -np.exp(-t/tau2)/(E1 + Ee)**2
224 dCdtau2 = -E1*t*np.exp(-t/tau2)/(Ee*tau2**2*(E1 + Ee))
225
226 jac.append(dCdEe)
227 jac.append(dCdE1)
228 jac.append(dCdtau2)
229 return np.asarray(jac)
230
231 def SLSM(jac=None):
232     """
233     Parameters
234     -----
235     Add parameters to be fit using the SLS model
236
237     Returns
238     -----
239     out : Model output
240     t : Model time output
241     fit : Model fit output
242     """
243
244     # Specify parameters
245     fit_params = lf.Parameters() # initialize the class for parameters
246     fit_params.add('Ee', value = 1, min=0) # Instantaneous shear modulus
247     fit_params.add('E1', value = 1, min=0) # Total change in modulus
248     fit_params.add('tau2', value = 1) # Time constant
249
250     # Set up minimization class to be able to pass derivative in (Jacobian)
251     minClass = lf.Minimizer(residual, fit_params, fcn_args=(t,),
252                             fcn_kws={'data': data})
253     if jac is None:
254         # No jacobian
255         out = minClass.leastsq()
256     else:
257         # Yes jacobian
258         out = minClass.leastsq(Dfun=dresidual, col_deriv=1)
259     fit = residual(out.params, t) # run the model to fit the data
260
261     lf.report_fit(out) # modelpars=p_true, show_correl=True
262     print('\n\n\n')
263     return out, t, fit
264
265 # In[Prony series curve fit]
266 """ Information for running the curve-fit algorithm """
267 nu = 0.49
268 jac = True # Jacobian (if None, then don't include. If True, do include)
269
270 if jac is None:
271     print('No Jacobian')
272 elif jac is True:

```

```

273     print('Jacobian')
274 elif jac is False:
275     print('jac needs to be "None" or "True"')
276     sys.exit()
277
278 p = {} # empty dictionary
279 tfit = {} # empty dictionary
280 f = {} # empty dictionary
281
282 # run the curve fit
283 p['p'], tfit['tfit'], f['f'] = SLSM(jac)
284
285 # In[Plot data]
286
287 color_map = plt.cm.tab10
288
289 # Plot Relaxation
290 plt.plot(t, data, 'o', label=dataName + ' Data', linewidth=2, markersize=5,
291          color=color_map.colors[0])
292
293 rsqrd = PronyR2(data, f['f'])
294
295 plt.plot(tfit['tfit'], f['f'],
296          label='LMFIT 2-Term Standard Linear Solid Model' if jac is None else
297          'LMFIT 2-Term Standard Linear Solid Model with Jacobian, ' +
298          f'$r^2={rsqrd:.5}$', linewidth=2, color=color_map.colors[1])
299 plt.xlabel(HorizontalUnits, fontsize=18)
300 plt.ylabel(r'Creep Response ' + CreepUnits, fontsize=18)
301 plt.legend(loc = 'best', fontsize=14)
302 plt.grid(True, which='both', alpha=0.5)
303 plt.savefig("Figures/1LmFitSLSMCreep.pdf" if jac is None else
304            "Figures/1LmFitSLSMCreepJac.pdf", bbox_inches='tight')
305 plt.show()
306
307 # Plot Compliance
308 plt.plot(t, 1/data, '.', label=dataName + ' Data')
309 plt.plot(tfit['tfit'], 1/f['f'],
310          label='LMFIT Standard Linear Solid Model' if jac is None else
311          'LMFIT Standard Linear Solid Model with Jacobian')
312 plt.xlabel(HorizontalUnits, fontsize=18)
313 plt.ylabel(r'Relaxation Response ' + RelaxUnits, fontsize=18)
314 plt.legend(loc = 'best', fontsize=14)
315 plt.grid(True, which='both')
316 plt.savefig("Figures/2LmFitSLSMRelax.pdf" if jac is None else
317            "Figures/2LmFitSLSMRelaxJac.pdf", bbox_inches='tight')
318 plt.show()
319
320 # LogLog (To show different regions (Elastic, Retardation, Viscous))
321 plt.loglog(t, data, '.', label=dataName + ' Data')
322 plt.loglog(tfit['tfit'], f['f'],
323            label='LMFIT Standard Linear Solid Model' if jac is None else
324            'LMFIT Standard Linear Solid Model with Jacobian')
325 plt.xlabel(HorizontalUnits, fontsize=18)
326 plt.ylabel(r'Creep Response ' + CreepUnits, fontsize=18)
327 plt.legend(loc = 'best', fontsize=14)
328 plt.grid(True, which='both')
329 plt.savefig("Figures/3LmFitSLSMCreepLogLog.pdf" if jac is None else
330            "Figures/3LmFitSLSMCreepJacLogLog.pdf", bbox_inches='tight')

```

```

331 plt.show()
332
333 # Plot Compliance
334 plt.loglog(t, 1/data, '.', label=dataName + ' Data')
335 plt.plot(tfit['tfit'], 1/f['f'],
336          label='LMFIT Standard Linear Solid Model' if jac is None else
337          'LMFIT Standard Linear Solid Model with Jacobian')
338 plt.xlabel(HorizontalUnits, fontsize=18)
339 plt.ylabel(r'Relaxation Response ' + RelaxUnits, fontsize=18)
340 plt.legend(loc = 'best', fontsize=14)
341 plt.grid(True, which='both')
342 plt.savefig("Figures/4LmFitSLSMRelaxLogLog.pdf" if jac is None else
343            "Figures/4mFitSLSMRelaxJacLogLog.pdf", bbox_inches='tight')
344 plt.show()
345
346 # In[Extract Data]
347 # Extract data from LMFIT report
348
349 Ee = p['p'].params['Ee'].value
350 E1 = p['p'].params['E1'].value
351 Tau2 = p['p'].params['tau2'].value
352
353 # Write vitreous creep compliance data to a txt file for abaqus importing
354 file1 = open('Vitreous_SLSM_Constants_LMFIT.txt' if jac is None else
355            'Vitreous_SLSM_Constants_LMFIT_Jac.txt', "w")
356 str1 = ("Equation is in the form:  $C(t) = 1/Go*(1 - G1/(G1 +$ 
357          $+ Go)*np.exp(-t/tau2))$  # Standard Linear Solid Model (SLSM)")
358 str2 = ('Standard Linear Solid Model paper Lin2020 Figure 3, ' +
359         'equation 2 used in optimization')
360 str3 = 'Data set = ' + dataName
361 str4 = '' if jac is None else 'Jacobian was used to converge'
362 file1.write('\n'.join([str1, str2, str3, str4]) + '\n')
363
364 Eo = E1 + Ee # Instantaneous modulus
365
366 nu = 0.49
367 G_to_E_conversion = 2*(1+nu)
368
369 # Write to a txt file
370 file1.write('\n' + 79*'=')
371 file1.write('\nCalculated long term modulus (Goo) is: ' + str(Ee))
372 file1.write("\nCalculated modulus (G1) is: " + str(E1))
373 file1.write("\nCalculated instantaneous modulus (Go) is: " + str(Eo))
374 file1.write("\nCalculated time constant (Tau2) is: " + str(Tau2))
375 file1.write('\n' + 79*'.')
376 file1.write('\n  $E = G*2*(1+nu)$ , where  $nu = {}$ '.format(nu))
377 file1.write('\nCalculated long term modulus (Eoo) is: ' +
378            str(Ee*G_to_E_conversion))
379 file1.write("\nCalculated modulus (E1) is: " + str(E1*G_to_E_conversion))
380 file1.write("\nCalculated instantaneous modulus (Eo) is: " +
381            str(Eo*G_to_E_conversion))
382 file1.write("\n")
383 file1.close()

```

## 1.2 Prony Series Curve Fit

</> Script 2: Python script used to fit an  $n$ 'th-term Prony series model to creep data. </>

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Tue Mar 31 17:48:07 2020
4
5 @author: Kiffer
6 """
7 import lmfit as lf # lmfit
8 import numpy as np # numpy
9 import matplotlib.pyplot as plt
10 plt.rcParams['figure.figsize'] = [16, 9]
11 import pandas as pd
12 import os
13 import sys
14 import pdb
15 filePath = os.getcwd() # Location of Python script
16
17 dataSets = ['Jami', 'Lee', 'Kashani', 'Polymer', 'Tram']
18 dataSet = 4
19 dataSet = dataSets[dataSet]
20
21 def PronyR2(y, fit):
22     # R squared calculation
23     SS_tot = np.sum((y - np.mean(y))**2)
24     SS_res = np.sum((y - fit)**2)
25     Rsqd = 1 - SS_res/SS_tot
26     return Rsqd
27
28 # In[Jami Data for example]
29 if dataSet == 'Jami':
30     ##### Jami Creep Shear Test Data Curve Fit#####
31     dataName = 'Jami 2014'
32     Jami_data = pd.read_csv('Jami/Jami_Shear_Data.txt', sep="\t", header=0)
33     Jami_data.columns = ["Time", "NormShearCreep"]
34     Jami_data['NormRelaxData'] = 1/Jami_data.NormShearCreep
35
36     # Invert and compute the raw data
37     Raw_data_0 = 0.0214671 # first shear point in raw data
38     Jami_data['CreepData'] = Jami_data.NormShearCreep*Raw_data_0
39     Jami_data['RelaxData'] = 1/Jami_data['CreepData']
40
41     # Convert data to array
42     t = Jami_data.Time
43     # data = Jami_data.NormShearCreep
44     data = Jami_data.NormRelaxData # Normalized relaxation data
45     # data = Jami_data.CreepData
46     # data = Jami_data.RelaxData
47
48     # Units for plotting
49     CreepUnits = r'$\left(\mathrm{Pa}\right)$'
50     RelaxUnits = r'$\left(\frac{1}{\mathrm{Pa}}\right)$'
51     HorizontalUnits = r'Time (s)'
52
53 # In[Lee Digitized Data]
```



```

54
55 if dataSet == 'Lee':
56     # Read data from Lee1992 Viscoelastic material properties
57     # Digitized from http://getdata-graph-digitizer.com/
58     dataName = 'Lee 1992'
59     Lee_data = pd.read_csv('Lee1992_DigitizedData.txt', sep="\t", header=3)
60     Lee_data.columns = ["Time", "CreepDyne"]
61     # Conversion is dyne/cm^2 to Pa is multiply by 0.1
62     Lee_data["CreepPa"] = Lee_data.CreepDyne*0.1
63     Lee_data["RelaxPa"] = 1/Lee_data.CreepPa
64     Lee_data["CreepNormalized"] = Lee_data.CreepPa/Lee_data.CreepPa[0]
65     Lee_data["RelaxNormalized"] = Lee_data.RelaxPa/Lee_data.RelaxPa[0]
66
67     t = Lee_data.Time
68     data = Lee_data.RelaxNormalized
69
70     # Units for plotting
71     CreepUnits = r'$\left(\mathrm{Pa}\right)$'
72     RelaxUnits = r'$\left(\frac{1}{\mathrm{Pa}}\right)$'
73     HorizontalUnits = r'Time (s)'
74
75 # In[Kashani_2011]
76 if dataSet == 'Kashani':
77     # Porcine eyes
78     t = np.linspace(1, 600, 1000)
79
80     def modelEqn(t, J1, J2, T1, T2, eta_m):
81         """
82         Parameters
83         -----
84         J1 : TYPE
85             DESCRIPTION.
86         J2 : TYPE
87             DESCRIPTION.
88         T1 : TYPE
89             DESCRIPTION.
90         T2 : TYPE
91             DESCRIPTION.
92         eta_m : TYPE
93             DESCRIPTION.
94
95         Returns
96         -----
97         Compliance
98         """
99         return J1*(1 - np.exp(-t/T1)) + J2*(1 - np.exp(-t/T2)) + t/eta_m
100
101     J1, J2, T1, T2, eta_m = 1.36, 2.64, 1.77, 1.36, 1332.0
102     J_0 = modelEqn(1, J1, J2, T1, T2, eta_m) # Initial value
103     data = 1/(modelEqn(t, J1, J2, T1, T2, eta_m)/J_0) # invert to define relaxation
104     # I'm not sure how this equation works
105
106     # Read data from Lee1992 Viscoelastic material properties
107     # Digitized from http://getdata-graph-digitizer.com/
108     dataName = 'Kashani 2011'
109     Kashani_data = pd.read_csv('KashaniCreepData.csv', sep=",", header=1)
110     Kashani_data.columns = ["Time1", "Creep1",
111                             "Time2", "Creep2",

```

```

112         "Time3", "Creep3"]
113     # t = Kashani_data.Time1.dropna() # get rid of NaN
114     # Compliance = Kashani_data.Creep1.dropna() # Pa # get rid of NaN
115     t = Kashani_data.Time2.dropna() # get rid of NaN
116     Creep = Kashani_data.Creep2.dropna() # Pa # get rid of NaN
117     # t = Kashani_data.Time3.dropna() # get rid of NaN
118     # Compliance = Kashani_data.Creep3.dropna() # Pa # get rid of NaN
119
120     CreepNorm = Creep/Creep[0]
121     NormRelaxation = 1/CreepNorm
122     data = NormRelaxation
123
124     # Units for plotting
125     CreepUnits = r'$\left(\mathrm{Pa}\right)$'
126     RelaxUnits = r'$\left(\frac{1}{\mathrm{Pa}}\right)$'
127     HorizontalUnits = r'Time (s)'
128
129 # In[polymer data]
130
131 if dataSet == 'Polymer':
132     ##### Creep Shear Test Data Curve Fit#####
133     dataName = 'Polymer 2019'
134     subFolder = 'Polymer'
135     # df = pd.read_csv(os.path.join(filePath, subFolder,
136     #                               'Dogbone1_Test1.csv'),
137     #                  sep=",", header=5)
138     df = pd.read_csv(os.path.join(filePath, subFolder,
139     #                               'Dogbone_1_StressRelaxation_2.csv'),
140     #                  sep=",", header=5)
141     # df = pd.read_csv(os.path.join(filePath, subFolder,
142     #                               'Dogbone1_StressRelaxtion_3.csv'),
143     #                  sep=",", header=5)
144     df.columns = ["Time", "Extension", "Load"]
145
146     # Convert dataframe to array
147     time = np.asarray(df['Time'].tolist())
148     extension = np.asarray(df['Extension'].tolist())
149     load = np.asarray(df['Load'].tolist())
150
151     # Specimen properties
152     width = 12/1000.0
153     thickness = 5.3/1000.0
154
155     # calculations
156     stress = load/(width*thickness)
157     strain = extension/100
158
159     # Determine where the stress begins to decrease from the max point
160     # in the array
161     StressRelax=stress[np.argmax(stress):-1];
162     TimeRelax=time[np.argmax(stress):-1]-time[np.argmax(stress)]
163
164     data = StressRelax#/StressRelax[0] # Normalized stress relaxation
165     t = TimeRelax
166
167     # Units for plotting
168     CreepUnits = r'$\left(\mathrm{Pa}\right)$'
169     RelaxUnits = r'$\left(\frac{1}{\mathrm{Pa}}\right)$'

```

```

170     HorizontalUnits = r'Displacement (x)'
171
172 # In[Tram]
173
174 if dataSet == 'Tram':
175     dataName = 'Tram'
176     ExcelPath = 'Tram_2018_Creep_Data.xlsx'
177
178     df = pd.read_excel(os.path.join(filePath, ExcelPath), sheet_name=None)
179
180     path = 'Tram'
181
182     # Folder for general figures to be stored
183     TramFigures = os.path.join(path, 'Figures')
184     if not os.path.exists(TramFigures):
185         os.makedirs(TramFigures)
186
187     for i,j in enumerate(df.keys()):
188         if j == 'HU 0764 OS 1 Pa': # 9
189             # if j == 'HU2018-0074 OD 1 Pa': # 5
190             # if j == 'HU2018-0125 OS 1 Pa': # 4
191             # if j == 'HU2018-0125 OD 1 Pa': # 3
192
193             # Specific file name (adds the iteration number for organization)
194             specificName = str(i) + '_' + j
195
196             # Make directory for each data trace and associated images
197             specificPath = os.path.join(path, '{}_'.format(i) + j)
198             if not os.path.exists(specificPath):
199                 os.makedirs(specificPath)
200
201             sheeti = df[j] #.dropna() # Eliminate rows with NA
202
203             time = sheeti.iloc[1:-1,0].reset_index(drop=True)
204             creep = sheeti.iloc[1:-1,1].reset_index(drop=True)
205
206             time_constantStress = 6
207             # Shift values past region of ramp stress
208             creep = creep[time >= time_constantStress].reset_index(drop=True)
209             # Shift values past region of ramp stress
210             time = time[time >= time_constantStress].reset_index(drop=True)
211
212             # Convert pandas series to numpy arrays
213             timeArray = time.to_numpy(dtype='float')
214             creepArray = creep.to_numpy(dtype='float')
215
216             # Get rid of nan values from the data trace
217             timeArrayRemoveNans = timeArray[np.logical_not(np.isnan(creepArray))]
218             creepArrayRemoveNans = creepArray[np.logical_not(np.isnan(creepArray))]
219
220             # Start time at 0
221             timeArrayRemoveNans = timeArrayRemoveNans - timeArrayRemoveNans[0]
222
223             t = timeArrayRemoveNans
224             creepData = creepArrayRemoveNans
225             creepNorm = creepData/creepData[0]
226             data = 1/creepNorm
227

```

```

228 # In[Lmfit]
229
230 def residual(pars, t, data=None):
231     """
232     Parameters
233     -----
234     pars : g_k and Tau_k for Prony N'th order terms
235           The final parameter is the sum of the terms that needs to be less
236           than 1 for realistic thermodynamic properties
237     t : time array being passed in
238     data : Normalized relaxation data (1/creep compliance) to be passed
239           through to compare to model data. The default is None.
240
241     Returns
242     -----
243     If no data is supplied, the return is the new model for plotting the final
244     curve
245
246     If data is supplied it will calculate the error between the actual
247     and known data
248     """
249
250     # Extract g_k and tau_k from the pars class variable
251     g_k = []
252     tau_k = []
253     for key, value in pars.items():
254         if key.find('g') >= 0:
255             g_k.append(value.value)
256         elif key.find('T') >= 0:
257             tau_k.append(value.value)
258
259     if NormalizedData is True:
260         model = 1 # Normalized so this begins at 1
261     else:
262         G0 = pars['G0'].value # Instantaneous modulus
263         model = G0
264     for i in range(len(g_k)):
265         model -= g_k[i]*(1 - np.exp(-t/tau_k[i])) # Loop over prony terms
266
267     if data is None:
268         return model
269     return model - data
270
271 def dresidual(pars, t, data=None):
272     """
273     Derivative of the function to return the jacobian for faster optimization
274     Parameters
275     -----
276     pars : g_k and Tau_k for Prony N'th order terms
277           The final parameter is the sum of the terms that needs to be less
278           than 1 for realistic thermodynamic properties
279     t : time array being passed in
280     data : Normalized relaxation data (1/creep compliance) to be passed
281           through to compare to model data. The default is None.
282
283     Returns
284     -----
285     The jacobian (partial derivatives with respect to unknown variables)
286     """

```

```

286     # Extract g_k and tau_k from the pars class variable
287     g_k = []
288     tau_k = []
289     for key, value in pars.items():
290         if key.find('g') >= 0:
291             g_k.append(value.value)
292         elif key.find('T') >= 0:
293             tau_k.append(value.value)
294     jac = []
295     if NormalizedData is not True:
296         jac.append(np.ones(len(t))) # derivative of G(t) with respect to G0
297     for i in range(len(g_k)):
298         jac.append(-1 + np.exp(-t/tau_k[i]))
299         jac.append(g_k[i]*t*np.exp(-t/tau_k[i])/tau_k[i]**2)
300     return np.asarray(jac)
301
302
303 def PronyN(N, jac=None):
304     """
305     Parameters
306     -----
307     N : Number of parameters in the Prony Series fit
308
309     sumG ensures that the values for the individual springs divided by the G0
310     value sum to a value less than 1.
311
312     Returns
313     -----
314     out : Model output
315     t_fit : Model time output
316     fit : Model fit output
317     """
318     # Specify paramters bounds with a for loop for N terms
319     fit_params = lf.Parameters() # intialize the class for parameters
320     if NormalizedData is True:
321         # Used when normalized data
322         fit_params.add('G0', value = 1, vary=False)
323     else:
324         # Instantaneous shear modulus
325         fit_params.add('G0', value = 1, min=0, max=G_0_UpperLimit)
326     sumG = ''
327     for i in range(N):
328         if i == N-1:
329             # append g_k values for the constraint eqn
330             sumG = sumG + 'g_{}/G0'.format(i + 1)
331         else:
332             # append g_k values for the constraint eqn
333             sumG = sumG + 'g_{}/G0 + '.format(i + 1)
334
335     # If Normalized data the bounds of the values are [0,1],
336     # otherwise [0,infinity]
337     if NormalizedData is True:
338         # Used when normalized
339         fit_params.add('g_{}'.format(i + 1), value=0.1/N,
340                       min=0.0, max=1.0)
341         fit_params.add('Tau_{}'.format(i + 1), value=1, min=0.0)
342     else:
343         # Used when not normalized (1/N)

```

```

344         fit_params.add('g_{}'.format(i + 1), value=1, min=0.0)
345         # Polymer 0.00001*G_0_UpperLimit
346         fit_params.add('Tau_{}'.format(i + 1), value=1, min=0.0)
347
348         # comment this out if you want to relax the requirement for the
349         # sum of coefficients
350         if Constraint is True:
351             # Constraint eqn
352             fit_params.add('sumG', min=0, max=1, expr=sumG, vary=True)
353
354         # Set up minimization class to be able to pass derivative in (Jacobian)
355         minClass = lf.Minimizer(residual, fit_params, fcn_args=(t,),
356                                fcn_kws={'data': data})
357         if jac is None:
358             # No jacobian
359             out = minClass.leastsq()
360         else:
361             # Yes jacobian
362             out = minClass.leastsq(Dfun=dresidual, col_deriv=1)
363             # t_fit = np.linspace(0,max(t), 1000)
364             fit = residual(out.params, t) # t_fit
365
366         lf.report_fit(out) # modelpars=p_true, show_correl=True
367         print('\n\n\n')
368         return out, t, fit # t_fit
369
370 # In[Prony series curve fit]
371 """ Information for running the curve-fit algorithm """
372 pronyTerms = [1,2,3,4] # number of prony series terms to be plotted [List]
373 # Upper limit on the instantaneous shear modulus
374 G_0_UpperLimit = 1000 # 1000000
375 NormalizedData = True # Normalized data
376 # enforce the constraint where the sum of G_k's can't be more than 1
377 Constraint = True
378 nu = 0.49
379 jac = True # Jacobian (if None, then don't include. If True, do include)
380 if jac is None:
381     print('No Jacobian')
382 elif jac is True:
383     print('Jacobian')
384 elif jac is False:
385     print('jac needs to be "None" or "True"')
386     sys.exit()
387
388 print('Upper limit for instantaneous shear modulus is', G_0_UpperLimit)
389
390 p = {} # empty dictionary
391 tfit = {} # empty dictionary
392 f = {} # empty dictionary
393
394 # Loop over the number of prony terms to calculate the curve fit paramters
395 for i in pronyTerms:
396     A, B, C = PronyN(i, jac)
397     p['p_{}'.format(i)] = A
398     tfit['tfit_{}'.format(i)] = B
399     f['f_{}'.format(i)] = C
400
401 # In[Plot data]

```

```

402 E_0 = 1#1840
403 v_0 = 0.49
404 G_0 = E_0/(2*(1+v_0))
405 G_0 = 49.3075445
406 def Prony2(t,a,b,c,d):
407     return G_0*(1 - a*(1 - np.exp(-t/b)) - c*(1 - np.exp(-t/d)))
408 ABQtime = np.linspace(0, max(t), 300)
409 ABQfit_Norm = Prony2(0, 0.70134, 2.96389e-2, 0.19334, 0.47088)
410 ABQfit = Prony2(ABQtime, 0.70134, 2.96389e-2, 0.19334, 0.47088)
411
412 # Plot Relaxation
413 # plt.plot(ABQtime, ABQfit, 'b-', label='ABAQUS')
414 plt.plot(t, data, '.', label=dataName + ' Data')
415 for i in pronyTerms:
416     plt.plot(tfit['tfit{}'.format(i)], f['f{}'.format(i)],
417              label='LMFIT {} Prony terms'.format(i) if jac is None else
418              'LMFIT {} Prony terms with Jacobian'.format(i))
419 # plt.ylim(0.01,0.02)
420 plt.xlabel('Time (s)',fontsize=18)
421 plt.ylabel(r'Relaxation Modulus',fontsize=18) # Normalized
422 plt.title('Viscoelastic Response',fontsize=20)
423 plt.legend(loc = 'best', fontsize=14)
424 plt.grid(True, which='both')
425 plt.savefig("Figures/1LmFitRelax.pdf" if jac is None else
426            "Figures/1LmFitRelaxJac.pdf", bbox_inches='tight')
427 plt.show()
428
429
430 # Plot Compliance
431 # plt.plot(ABQtime, 1/ABQfit, 'b-', label='ABAQUS')
432
433 color_map = plt.cm.tab10
434
435 plt.plot(t, 1/data, ':o', label=dataName + ' Data', linewidth=2,
436          markersize=5, color=color_map.colors[0])
437
438 # Plot only the 4 term fit instead of all fits
439 for i in pronyTerms[-1:]:
440     rsqrd_i = PronyR2(1/data, 1/f[f'f{i}'])
441     plt.plot(tfit[f'tfit{i}'], 1/f[f'f{i}'],
442              label=f'LMFIT {i} Prony terms' if jac is None else
443              f'LMFIT {i} Prony terms with Jacobian, $r^2={rsqrd_i:.5}$',
444              linewidth=2, color=color_map.colors[1])
445
446 # plt.ylim(50,100)
447 plt.xlabel('Time (s)',fontsize=18)
448 plt.ylabel(r'Normalized Creep Compliance',fontsize=18)
449 # plt.title('Viscoelastic Response',fontsize=20)
450 plt.legend(loc = 'best', fontsize=14)
451 plt.grid(True, which='both', alpha=0.5)
452 plt.savefig("Figures/2LmFitCompliance.pdf" if jac is None else
453            "Figures/2LmFitComplianceJac.pdf", bbox_inches='tight')
454 plt.show()
455
456 # LogLog (To show different regions (Elastic, Retardation, Viscous))
457 plt.loglog(t, data, '.', label=dataName + ' Data')
458 for i in pronyTerms:
459     plt.loglog(tfit['tfit{}'.format(i)], f['f{}'.format(i)],

```

```

460         label='LMFIT {} Prony terms'.format(i) if jac is None else
461         'LMFIT {} Prony terms with Jacobian'.format(i))
462 plt.xlabel('Time (s)',fontsize=18)
463 plt.ylabel(r'Relaxation Modulus',fontsize=18)
464 plt.title('Viscoelastic Response',fontsize=20)
465 plt.legend(loc = 'best', fontsize=14)
466 plt.grid(True, which='both')
467 plt.savefig("Figures/3LmFitRelaxLogLog.pdf" if jac is None else
468             "Figures/3LmFitRelaxJacLogLog.pdf", bbox_inches='tight')
469 plt.show()
470
471 # Plot Compliance
472 plt.loglog(t, 1/data, '.', label=dataName + ' Data')
473 for i in pronyTerms:
474     plt.loglog(tfit['tfit{}'.format(i)], 1/f['f{}'.format(i)],
475               label='LMFIT {} Prony terms'.format(i) if jac is None else
476               'LMFIT {} Prony terms with Jacobian'.format(i))
477 plt.xlabel('Time (s)',fontsize=18)
478 plt.ylabel(r'Creep Compliance',fontsize=18)
479 plt.title('Viscoelastic Response',fontsize=20)
480 plt.legend(loc = 'best', fontsize=14)
481 plt.grid(True, which='both')
482 plt.savefig("Figures/4LmFitComplianceLogLog.pdf" if jac is None else
483             "Figures/4mFitComplianceJacLogLog.pdf", bbox_inches='tight')
484 plt.show()
485
486 # In[Normalized Plots]
487 # if NormalizedData == True:
488 #     # Plot Relaxation
489 #     data0 = data[0]
490 #     plt.plot(t, data/data0, '.', label=dataName + ' Data')
491 #     for i in pronyTerms:
492 #         ti = tfit['tfit{}'.format(i)] # Time
493 #         fi = f['f{}'.format(i)] # Curve fit data
494 #         fi0 = fi[0] # Normalization by the first data point data0#
495 #         plt.plot(ti, fi/fi0,
496 #                 label='LMFIT {} Prony terms'.format(i) if jac is None else
497 #                 'LMFIT {} Prony terms with Jacobian'.format(i))
498 #         # plt.ylim(0.01,0.02)
499 #         plt.xlabel('Time (s)',fontsize=18)
500 #         plt.ylabel(r'Normalized Relaxation Modulus',fontsize=18) # Normalized
501 #         plt.title('Viscoelastic Response',fontsize=20)
502 #         plt.legend(loc = 'best', fontsize=14)
503 #         plt.grid(True, which='both')
504 #     plt.savefig("Figures/5NormLmFitRelax.pdf" if jac is None else
505 #                 "Figures/5NormLmFitRelaxJac.pdf", bbox_inches='tight')
506 #     plt.show()
507
508 #     # Plot Compliance
509 #     plt.plot(t, 1/(data/data0), '.', label=dataName + ' Data')
510 #     for i in pronyTerms:
511 #         ti = tfit['tfit{}'.format(i)] # Time
512 #         fi = f['f{}'.format(i)] # Curve fit data
513 #         fi0 = fi[0] # Normalization by the first data point data0#
514 #         plt.plot(ti, 1/(fi/fi0),
515 #                 label='LMFIT {} Prony terms'.format(i) if jac is None else
516 #                 'LMFIT {} Prony terms with Jacobian'.format(i))
517 #         # plt.ylim(50,100)

```



```

518 # plt.xlabel('Time (s)',fontsize=18)
519 # plt.ylabel(r'Normalized Creep Compliance',fontsize=18)
520 # plt.title('Viscoelastic Response',fontsize=20)
521 # plt.legend(loc = 'best', fontsize=14)
522 # plt.grid(True, which='both')
523 # plt.savefig("Figures/6NormLmFitCompliance.pdf" if jac is None else
524 #             "Figures/6NormLmFitComplianceJac.pdf", bbox_inches='tight')
525 # plt.show()
526
527 # LogLog (To show different regions (Elastic, Retardation, Viscous))
528 # plt.loglog(t, data/data0, '.', label=dataName + ' Data')
529 # for i in pronyTerms:
530 #     ti = tfit['tfit{}'.format(i)] # Time
531 #     fi = f['f{}'.format(i)] # Curve fit data
532 #     fi0 = fi[0] # Normalization by the first data point data0#
533 #     plt.loglog(ti, fi,
534 #               label='LMFIT {} Prony terms'.format(i) if jac is None else
535 #               'LMFIT {} Prony terms with Jacobian'.format(i))
536 # plt.xlabel('Time (s)',fontsize=18)
537 # plt.ylabel(r'Normalized Relaxation Modulus',fontsize=18)
538 # plt.title('Viscoelastic Response',fontsize=20)
539 # plt.legend(loc = 'best', fontsize=14)
540 # plt.grid(True, which='both')
541 # plt.savefig("Figures/7NormLmFitRelaxLogLog.pdf" if jac is None else
542 #             "Figures/7NormLmFitRelaxJacLogLog.pdf",
543 #             bbox_inches='tight')
544 # plt.show()
545
546 # Plot Compliance
547 # plt.loglog(t, 1/(data/data0), '.', label=dataName + ' Data')
548 # for i in pronyTerms:
549 #     ti = tfit['tfit{}'.format(i)] # Time
550 #     fi = f['f{}'.format(i)] # Curve fit data
551 #     fi0 = fi[0] # Normalization by the first data point data0#
552 #     plt.loglog(ti, 1/(fi/fi0),
553 #               label='LMFIT {} Prony terms'.format(i) if jac is None else
554 #               'LMFIT {} Prony terms with Jacobian'.format(i))
555 # plt.xlabel('Time (s)',fontsize=18)
556 # plt.ylabel(r'Normalized Creep Compliance',fontsize=18)
557 # plt.title('Viscoelastic Response',fontsize=20)
558 # plt.legend(loc = 'best', fontsize=14)
559 # plt.grid(True, which='both')
560 # plt.savefig("Figures/8NormLmFitComplianceLogLog.pdf" if jac is None else
561 #             "Figures/8NormLmFitComplianceJacLogLog.pdf",
562 #             bbox_inches='tight')
563 # plt.show()
564
565
566 # In[Extract Data]
567 # Extract data from LMFIT report
568 optParams = {} # empty dictionary
569 # Loop over pronyTerms to extract the g_k and tau_k values for each N'th
570 # order fit
571 for i in pronyTerms:
572     g_k = [] # Shear modulus per Prony element
573     tau_k = [] # Time constant
574     G0_k = [] # Instantaneous shear modulus
575     for key, value in p['p{}'.format(i)].params.items():

```

```

576         if key.find('g') >= 0:
577             g_k.append(value.value)
578         elif key.find('T') >= 0:
579             tau_k.append(value.value)
580         elif key.find('G0') >= 0:
581             G0_k.append(value.value)
582     optParams['P{}'.format(i)] = g_k
583     optParams['T{}'.format(i)] = tau_k
584     optParams['G0{}'.format(i)] = G0_k
585
586     # Write vitreous creep compliance data to a txt file for abaqus importing
587     file1 = open('Vitreous_Prony_Constants_LMFIT.txt' if jac is None else
588                 'Vitreous_Prony_Constants_LMFIT_Jac.txt', "w")
589     str1 = ("Equation is in the form:  $G(t) = G_0(1 - \sum_{i=1}^N (g_k^P (1 - \exp(-t/\tau_k)))$ ")
590     str2 = '\t'.join(["Prony_#", "g_k^P", "k_i", "tau_k"])
591     str3 = 'Data set = ' + dataName
592     str4 = '' if jac is None else 'Jacobian was used to converge'
593     str5 = '' if Constraint is False else ('enforce the constraint where the " +
594                                           "sum of G_k/G_0's can't be more than 1")
595     str6 = '' if NormalizedData is True else ("Upper limit for G_0 is" +
596                                              " {}".format(G_0_UpperLimit))
597     file1.write('\n'.join([str1, str2, str3, str4, str5, str6]) + '\n')
598
599
600     # Loop over pronyTerm results to write to a txt file
601     for i in pronyTerms:
602         g_k = optParams['P{}'.format(i)] # shear modulus of prony term
603         tau_k = optParams['T{}'.format(i)] # Time constant
604         G_0 = optParams['G0{}'.format(i)][0] # Instantaneous shear modulus
605         file1.write('\n' + 79*'=' + '\n')
606         file1.write('\n' + 22*' ' + 'Prony series order ' + str(len(g_k)))
607         file1.write('\nCalculated instantaneous shear modulus (G_0) is: ' +
608                    str(G_0))
609         file1.write('\nKnown nu = {}'.format(nu))
610         file1.write("\nCalculated instantaneous Young's modulus (E_0) is: " +
611                    str(G_0*2*(1 + nu)) + '\n')
612         file1.write('\nNormalized coefficients\n')
613         for m in range(len(g_k)):
614             file1.write(''.join(['(' + str(g_k[m]), ', 0.0, ',
615                                 str(tau_k[m]) + '),']) + '\n')
616         print(sum(g_k), "g_k")
617         file1.write("\n")
618         # file1.write('Normalized coefficients (g_k/G_0)\n')
619         # for m in range(len(g_k)):
620             # file1.write(''.join(['(' + str(g_k[m]/G_0), ', 0.0, ',
621                                   str(tau_k[m]) + '),']) + '\n')
622         # print(sum(g_k), "g_k")
623         # file1.write("\n")
624     file1.close()

```

### 1.3 Raw Data Analysis

Identify Maximum and Steady-State Indices from the raw data in ??.

</> **Script 3:** Python script used to identify the maximum and steady-state time and index values from the raw data used for optimization. </>

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Aug 24 15:45:38 2020
4
5 @author: Kiffer2
6 """
7
8 import pandas as pd
9 import sqlite3
10 import glob
11 import os
12 import numpy as np
13 import matplotlib.pyplot as plt
14 import matplotlib.patches as mpatches
15 from matplotlib.pyplot import cm
16 import matplotlib.patheffects as pe
17 plt.rcParams['figure.figsize'] = [16, 9]
18 import pdb
19
20
21 def Least_Squares(x,y):
22     """
23     Calculate the slope and y-intercept using matrix math
24     x & y are the coordinates of points
25
26     parameters (X,Y) Data
27
28     Returns:
29     Curve fit data and parameters m*x + b
30     """
31     Z = np.ones((len(x),2))
32     Z[:,1] = x
33     # Calculate the matrix inverse for the constants of the regression
34     A = np.dot(np.linalg.inv(np.dot(Z.T,Z)),(np.dot(Z.T,y)))
35     linFit = x*A[1] + A[0]
36
37     # Stats
38     SS_tot = np.sum((y - np.mean(y))**2)
39     SS_res = np.sum((y - linFit)**2)
40     Rsqd = 1 - SS_res/SS_tot
41
42     return linFit, A, Rsqd
43
44 os.chdir('F:/Abaqus Working Directory/PeterComp/HumanData')
45
46 filePath = os.getcwd()
47
48 ExcelName = 'HumanData.xlsx'
49
50 ExcelPath = os.path.join(filePath, ExcelName)
51
52 conn = sqlite3.connect('HumanData.db')
53 c = conn.cursor()
54
55 c.execute('DROP TABLE IF EXISTS HumanData')
```

```

56 c.execute(''CREATE TABLE 'HumanData'(
57     'HumanID' TEXT,
58     'HumanAGE' REAL,
59     'HumanGender' TEXT,
60     'HumanLeftRight' TEXT,
61     'HumanRegion' TEXT,
62     'PostMortemHrs_Min' REAL,
63     'DateOfDeath' timestamp,
64     'TimeOfDeath' timestamp,
65     'EnucleationDate' timestamp,
66     'EnucleationTime' timestamp,
67     'DateOfTesting' timestamp,
68     'TimeOfTesting' timestamp,
69     'DiameterPostAnt' REAL,
70     'DiameterNasTemp' REAL,
71     'SSi' REAL,
72     'SSf' REAL,
73     'TFMax' REAL,
74     'DispMax' REAL,
75     'FMax' REAL,
76     'FSS' REAL,
77     'Slope10' REAL,
78     'Rsqr10' REAL,
79     'Slope20' REAL,
80     'Rsqr20' REAL,
81     'Slope30' REAL,
82     'Rsqr30' REAL,
83     'Slope0' REAL,
84     'Rsqr0' REAL,
85     'PeelVideoName' TEXT,
86     'PeelVideoHyperlink' TEXT,
87     'VideoComments' TEXT,
88     'LightMicroscopyImages' TEXT)
89 '')
90
91 df = pd.read_excel(ExcelPath, sheet_name=None)
92
93 """ Put values into a dictionary of dataframes """
94 HumanID = {}
95 HumanAge = {}
96 HumanGender = {}
97 HumanLeftRight = {}
98 HumanRegion = {}
99 PostMortemHrs_Min = {}
100 DateOfDeath = {}
101 TimeOfDeath = {}
102 EnucleationDate = {}
103 EnucleationTime = {}
104 DateOfTesting = {}
105 TimeOfTesting = {}
106 DiameterPostAnt = {}
107 DiameterNasTemp = {}
108 SSi = {}
109 SSf = {}
110 FMax = {}
111 FSS = {}
112 TFMax = {}
113 DispMax = {}

```

```

114 Slope10 = {}
115 Rsqrd10 = {}
116 Slope20 = {}
117 Rsqrd20 = {}
118 Slope30 = {}
119 Rsqrd30 = {}
120 Slope0 = {}
121 Rsqrd0 = {}
122 PeelVideoName = {}
123 PeelVideoHyperlink = {}
124 VideoComments = {}
125 LightMicroscopyImages = {}
126 time = {}
127 extension = {}
128 force = {}
129 for i in df.keys():
130     if len(i) <= 2: # Only look at data traces ... [Row,Col]
131         print(i)
132         HumanID['{}'.format(i)] = df['{}'.format(i)].iloc[0,1]
133         HumanAge['{}'.format(i)] = df['{}'.format(i)].iloc[1,1]
134         HumanGender['{}'.format(i)] = df['{}'.format(i)].iloc[2,1]
135         HumanLeftRight['{}'.format(i)] = df['{}'.format(i)].iloc[3,1]
136         HumanRegion['{}'.format(i)] = df['{}'.format(i)].iloc[4,1]
137         PostMortemHrs_Min['{}'.format(i)] = df['{}'.format(i)].iloc[5,1]
138         DateOfDeath['{}'.format(i)] = df['{}'.format(i)].iloc[6,1]
139         TimeOfDeath['{}'.format(i)] = df['{}'.format(i)].iloc[7,1]
140         EnucleationDate['{}'.format(i)] = df['{}'.format(i)].iloc[8,1]
141         EnucleationTime['{}'.format(i)] = df['{}'.format(i)].iloc[9,1]
142         DateOfTesting['{}'.format(i)] = df['{}'.format(i)].iloc[10,1]
143         TimeOfTesting['{}'.format(i)] = df['{}'.format(i)].iloc[11,1]
144         DiameterPostAnt['{}'.format(i)] = df['{}'.format(i)].iloc[12,1]
145         DiameterNasTemp['{}'.format(i)] = df['{}'.format(i)].iloc[13,1]
146         SSI['{}'.format(i)] = df['{}'.format(i)].iloc[18,6]
147         SSf['{}'.format(i)] = df['{}'.format(i)].iloc[19,6]
148         FMax['{}'.format(i)] = df['{}'.format(i)].iloc[22,6]
149         FSS['{}'.format(i)] = df['{}'.format(i)].iloc[21,6]
150         PeelVideoName['{}'.format(i)] = df['{}'.format(i)].iloc[26,6]
151         PeelVideoHyperlink['{}'.format(i)] = df['{}'.format(i)].iloc[27,6]
152         VideoComments['{}'.format(i)] = df['{}'.format(i)].iloc[28,6]
153         LightMicroscopyImages['{}'.format(i)] = df['{}'.format(i)].iloc[29,6]
154
155         """ Data Traces """
156         time['{}'.format(i)] =
157             → pd.to_numeric(df['{}'.format(i)].iloc[17:-1,0].reset_index(drop=True))
158         extension['{}'.format(i)] =
159             → pd.to_numeric(df['{}'.format(i)].iloc[17:-1,1].reset_index(drop=True))
160         force['{}'.format(i)] =
161             → pd.to_numeric(df['{}'.format(i)].iloc[17:-1,2].reset_index(drop=True))
162
163         if str(FMax['{}'.format(i)]) != 'nan':
164             # Time at max force
165             TFMaxi = time['{}'.format(i)][force['{}'.format(i)].loc[lambdax: x ==
166                 → FMax['{}'.format(i)].index.values[0]]
167             TFMax['{}'.format(i)] = TFMaxi
168
169             # Slope calculation 10 seconds before max peel force

```

```

166     maxIndex = force['{}'.format(i)].loc[lambda x: x ==
    ↪ FMax['{}'.format(i)].index.values[0] # Location in the array for the
    ↪ max force
167     x = extension['{}'.format(i)][maxIndex-100:maxIndex] # Array from
    ↪ MaxIndex-100 (10 sec) to location of max force
168     y = force['{}'.format(i)][maxIndex-100:maxIndex] # Array from
    ↪ MaxIndex-100 (10 sec) to location of max force
169     curveFit, Params, Rsqrd = Least_Squares(x,y) # Perform least squares and
    ↪ return
170     Slope10['{}'.format(i)] = Params[1]
171     Rsqrd10['{}'.format(i)] = Rsqrd
172
173     # Slope calculation 20 seconds before max peel force
174     x = extension['{}'.format(i)][maxIndex-200:maxIndex] # Array from
    ↪ MaxIndex-200 (20 sec) to location of max force
175     y = force['{}'.format(i)][maxIndex-200:maxIndex] # Array from
    ↪ MaxIndex-200 (20 sec) to location of max force
176     curveFit, Params, Rsqrd = Least_Squares(x,y) # Perform least squares and
    ↪ return
177     Slope20['{}'.format(i)] = Params[1]
178     Rsqrd20['{}'.format(i)] = Rsqrd
179
180     # Slope calculation 30 seconds before max peel force
181     x = extension['{}'.format(i)][maxIndex-300:maxIndex] # Array from
    ↪ MaxIndex-300 (30 sec) to location of max force
182     y = force['{}'.format(i)][maxIndex-300:maxIndex] # Array from
    ↪ MaxIndex-300 (30 sec) to location of max force
183     curveFit, Params, Rsqrd = Least_Squares(x,y) # Perform least squares and
    ↪ return
184     Slope30['{}'.format(i)] = Params[1]
185     Rsqrd30['{}'.format(i)] = Rsqrd
186
187     # Slope calculation from zero to max peel force
188     x = extension['{}'.format(i)][0:maxIndex] # Array from 0 to location of
    ↪ max force
189     y = force['{}'.format(i)][0:maxIndex] # Array from 0 to location of max
    ↪ force
190     curveFit, Params, Rsqrd = Least_Squares(x,y) # Perform least squares and
    ↪ return
191     Slope0['{}'.format(i)] = Params[1]
192     Rsqrd0['{}'.format(i)] = Rsqrd
193
194     # Displacement at max force
195     DispMax['{}'.format(i)] = extension['{}'.format(i)][time['{}'.format(i)]
    ↪ == TFMaxi].values[0]
196
197 else:
198     TFMaxi = np.nan
199     TFMax['{}'.format(i)] = TFMaxi
200     Slope10['{}'.format(i)] = np.nan
201     Rsqrd10['{}'.format(i)] = np.nan
202     Slope20['{}'.format(i)] = np.nan
203     Rsqrd20['{}'.format(i)] = np.nan
204     Slope30['{}'.format(i)] = np.nan
205     Rsqrd30['{}'.format(i)] = np.nan
206     Slope0['{}'.format(i)] = np.nan
207     Rsqrd0['{}'.format(i)] = np.nan
208     DispMax['{}'.format(i)] = np.nan

```

```

209
210 """ Add data to SQL database """
211 HumanIDi = HumanID['{}'.format(i)]
212 HumanAgei = HumanAge['{}'.format(i)]
213 HumanGenderi = HumanGender['{}'.format(i)]
214 HumanLeftRighti = HumanLeftRight['{}'.format(i)]
215 HumanRegioni = HumanRegion['{}'.format(i)]
216 PostMortemHrs_Mini = PostMortemHrs_Min['{}'.format(i)]
217 DateOfDeathi = DateOfDeath['{}'.format(i)]
218 TimeOfDeathi = TimeOfDeath['{}'.format(i)]
219 EnucleationDatei = EnucleationDate['{}'.format(i)]
220 EnucleationTimei = EnucleationTime['{}'.format(i)]
221 DateOfTestingi = DateOfTesting['{}'.format(i)]
222 TimeOfTestingi = TimeOfTesting['{}'.format(i)]
223 DiameterPostAnti = DiameterPostAnt['{}'.format(i)]
224 DiameterNasTempi = DiameterNasTemp['{}'.format(i)]
225 SSii = SSi['{}'.format(i)]
226 SSfi = SSf['{}'.format(i)]
227 TFMaxi = TFMax['{}'.format(i)]
228 DispMaxi = DispMax['{}'.format(i)]
229 FMaxi = FMax['{}'.format(i)]
230 FSSi = FSS['{}'.format(i)]
231 Slope10i = Slope10['{}'.format(i)]
232 Rsqrd10i = Rsqrd10['{}'.format(i)]
233 Slope20i = Slope20['{}'.format(i)]
234 Rsqrd20i = Rsqrd20['{}'.format(i)]
235 Slope30i = Slope30['{}'.format(i)]
236 Rsqrd30i = Rsqrd30['{}'.format(i)]
237 Slope0i = Slope0['{}'.format(i)]
238 Rsqrd0i = Rsqrd0['{}'.format(i)]
239 PeelVideoNamei = PeelVideoName['{}'.format(i)]
240 PeelVideoHyperlinki = PeelVideoHyperlink['{}'.format(i)]
241 VideoCommentsi = VideoComments['{}'.format(i)]
242 LightMicroscopyImagei = LightMicroscopyImages['{}'.format(i)]
243
244 # add to sql database
245 # cur.execute('' INSERT INTO HumanData(HumanID) VALUES (?)'',(HumanIDi))
246 # conn.commit()
247 c.execute('' INSERT INTO HumanData(
248         HumanID,
249         HumanAge,
250         HumanGender,
251         HumanLeftRight,
252         HumanRegion,
253         PostMortemHrs_Min,
254         DateOfDeath,
255         TimeOfDeath,
256         EnucleationDate,
257         EnucleationTime,
258         DateOfTesting,
259         TimeOfTesting,
260         DiameterPostAnt,
261         DiameterNasTemp,
262         SSi,
263         SSf,
264         TFMax,
265         DispMax,
266         FMax,

```

```

267         FSS,
268         Slope10,
269         Rsqrd10,
270         Slope20,
271         Rsqrd20,
272         Slope30,
273         Rsqrd30,
274         Slope0,
275         Rsqrd0,
276         PeelVideoName,
277         PeelVideoHyperlink,
278         VideoComments,
279         LightMicroscopyImages
280     ) VALUES
↪     (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
281     , (HumanIDi,
282       HumanAgei,
283       HumanGenderi,
284       HumanLeftRighti,
285       HumanRegioni,
286       str(PostMortemHrs_Mini),
287       str(DateOfDeathi),
288       str(TimeOfDeathi),
289       str(EnucleationDatei),
290       str(EnucleationTimei),
291       str(DateOfTestingi),
292       str(TimeOfTestingi),
293       DiameterPostAnti,
294       DiameterNasTempi,
295       SSii,
296       SSfi,
297       TFMaxi,
298       DispMaxi,
299       FMaxi,
300       FSSi,
301       Slope10i,
302       Rsqrd10i,
303       Slope20i,
304       Rsqrd20i,
305       Slope30i,
306       Rsqrd30i,
307       Slope0i,
308       Rsqrd0i,
309       PeelVideoNamei,
310       PeelVideoHyperlinki,
311       VideoCommentsi,
312       LightMicroscopyImagesi))
313     conn.commit()
314
315
316     # In[1] Plot individual data traces in the Figures folder as well as the
317     # age group/region folder
318
319     # Make the new folder to put figure
320     folderPath = os.path.join(filePath, 'Figures')
321     if not os.path.exists(folderPath):
322         os.makedirs(folderPath)
323

```



```

324 """ Age Groups """
325 g1e = [] # 30 - 39 yrs Equator
326 g1p = [] # 30 - 39 yrs Posterior
327 g2e = [] # 40 - 49 yrs Equator
328 g2p = [] # 40 - 49 yrs Posterior
329 g3e = [] # 50 - 59 yrs Equator
330 g3p = [] # 50 - 59 yrs Posterior
331 g4e = [] # 60 - 69 yrs Equator
332 g4p = [] # 60 - 69 yrs Posterior
333 g5e = [] # 70 - 79 yrs Equator
334 g5p = [] # 70 - 79 yrs Posterior
335 g6e = [] # 80 - 89 yrs Equator
336 g6p = [] # 80 - 89 yrs Posterior
337
338 # specific for paper 3
339 g8e = [] # 30 - 59 yrs Equator
340 g8p = [] # 30 - 59 yrs Posterior
341 g9e = [] # 60 - 89 yrs Equator
342 g9p = [] # 60 - 89 yrs Posterior
343
344 """ Plot Data """
345 for i in df.keys():
346     if len(i) <= 2: # Only look at data traces ... [Row,Col]
347         print(i)
348
349         """ Data Traces """
350         time['{}'.format(i)] =
351             → pd.to_numeric(df['{}'.format(i)].iloc[17:-1,0].reset_index(drop=True))
352         extension['{}'.format(i)] =
353             → pd.to_numeric(df['{}'.format(i)].iloc[17:-1,1].reset_index(drop=True))
354         force['{}'.format(i)] =
355             → pd.to_numeric(df['{}'.format(i)].iloc[17:-1,2].reset_index(drop=True))
356
357         """ call data from pandas dataframe """
358         HumanIDi = HumanID['{}'.format(i)]
359         HumanAgei = HumanAge['{}'.format(i)]
360         HumanGenderi = HumanGender['{}'.format(i)]
361         HumanLeftRighti = HumanLeftRight['{}'.format(i)]
362         HumanRegioni = HumanRegion['{}'.format(i)]
363         PostMortemHrs_Mini = PostMortemHrs_Min['{}'.format(i)]
364         DateOfDeathi = DateOfDeath['{}'.format(i)]
365         TimeOfDeathi = TimeOfDeath['{}'.format(i)]
366         EnucleationDatei = EnucleationDate['{}'.format(i)]
367         EnucleationTimei = EnucleationTime['{}'.format(i)]
368         DateOfTestingi = DateOfTesting['{}'.format(i)]
369         TimeOfTestingi = TimeOfTesting['{}'.format(i)]
370         DiameterPostAnti = DiameterPostAnt['{}'.format(i)]
371         DiameterNasTempi = DiameterNasTemp['{}'.format(i)]
372         SSii = SSi['{}'.format(i)]
373         SSfi = SSf['{}'.format(i)]
374         TMaxi = TMax['{}'.format(i)]
375         DispMaxi = DispMax['{}'.format(i)]
376         FMaxi = FMax['{}'.format(i)]
377         FSSi = FSS['{}'.format(i)]
378         Slope10i = Slope10['{}'.format(i)]
379         Rsqrd10i = Rsqrd10['{}'.format(i)]
380         Slope20i = Slope20['{}'.format(i)]
381         Rsqrd20i = Rsqrd20['{}'.format(i)]

```

```

379 Slope30i = Slope30['{}'.format(i)]
380 Rsqrd30i = Rsqrd30['{}'.format(i)]
381 Slope0i = Slope0['{}'.format(i)]
382 Rsqrd0i = Rsqrd0['{}'.format(i)]
383 PeelVideoNamei = PeelVideoName['{}'.format(i)]
384 PeelVideoHyperlinki = PeelVideoHyperlink['{}'.format(i)]
385 VideoCommentsi = VideoComments['{}'.format(i)]
386 LightMicroscopyImagesi = LightMicroscopyImages['{}'.format(i)]
387
388 if HumanAgei >= 30 and HumanAgei < 40:
389     if HumanRegioni == 'Equator':
390         g1e.append(i)
391         folder = '30_39_Equator'
392     elif HumanRegioni == 'Posterior':
393         g1p.append(i)
394         folder = '30_39_Posterior'
395 elif HumanAgei >= 40 and HumanAgei < 50:
396     if HumanRegioni == 'Equator':
397         g2e.append(i)
398         folder = '40_49_Equator'
399     elif HumanRegioni == 'Posterior':
400         g2p.append(i)
401         folder = '40_49_Posterior'
402 elif HumanAgei >= 50 and HumanAgei < 60:
403     if HumanRegioni == 'Equator':
404         g3e.append(i)
405         folder = '50_59_Equator'
406     elif HumanRegioni == 'Posterior':
407         g3p.append(i)
408         folder = '50_59_Posterior'
409 elif HumanAgei >= 60 and HumanAgei < 70:
410     if HumanRegioni == 'Equator':
411         g4e.append(i)
412         folder = '60_69_Equator'
413     elif HumanRegioni == 'Posterior':
414         g4p.append(i)
415         folder = '60_69_Posterior'
416 elif HumanAgei >= 70 and HumanAgei < 80:
417     if HumanRegioni == 'Equator':
418         g5e.append(i)
419         folder = '70_79_Equator'
420     elif HumanRegioni == 'Posterior':
421         g5p.append(i)
422         folder = '70_79_Posterior'
423 elif HumanAgei >= 80 and HumanAgei < 90:
424     if HumanRegioni == 'Equator':
425         g6e.append(i)
426         folder = '80_89_Equator'
427     elif HumanRegioni == 'Posterior':
428         g6p.append(i)
429         folder = '80_89_Posterior'
430
431 # Category for the age group/region for paper 3
432 if HumanAgei >= 30 and HumanAgei < 60:
433     if HumanRegioni == 'Equator':
434         g8e.append(i)
435         folder2 = '30_59_Equator'
436     elif HumanRegioni == 'Posterior':

```

```

437         g8p.append(i)
438         folder2 = '30_59_Posterior'
439     elif HumanAgei >= 60 and HumanAgei < 90:
440         if HumanRegioni == 'Equator':
441             g9e.append(i)
442             folder2 = '60_89_Equator'
443         elif HumanRegioni == 'Posterior':
444             g9p.append(i)
445             folder2 = '60_89_Posterior'
446
447
448     # Make the new folder to put figure
449     folderPath = os.path.join(filePath, folder)
450     if not os.path.exists(folderPath):
451         os.makedirs(folderPath)
452
453     # specific for paper 3
454     folderPath2 = os.path.join(filePath, folder2)
455     if not os.path.exists(folderPath2):
456         os.makedirs(folderPath2)
457
458     """ Plot force vs time and force vs disp """
459     # Calculate gradient
460     n = i
461     gn = np.gradient(extension['{}'.format(n)], time['{}'.format(n)])
462
463     tn = time['{}'.format(n)]
464     fn = force['{}'.format(n)]
465     dn = extension['{}'.format(n)]
466
467     ssin = SSi['{}'.format(n)]
468     ssfn = SSf['{}'.format(n)]
469
470     # Check if steady state values exist, if they do create time/force array
471     if str(ssin) == 'nan':
472         timeLocSSIn = np.nan
473         timeLocSSFn = np.nan
474
475         avgFnVal = np.nan
476
477         sstnArray = np.array([ssin, ssfn])
478         ssfnArray = np.array([avgFnVal, avgFnVal])
479     else:
480         # Location for steady state start/stop in the time array
481         timeLocSSIn = tn.loc[lambdax: x == ssin].index.values[0]
482         timeLocSSFn = tn.loc[lambdax: x == ssfn].index.values[0]
483
484         # Average force value in between the steady state times
485         avgFnVal =
486             ↪ np.sum(fn[timeLocSSIn:timeLocSSFn])/len(range(timeLocSSIn, timeLocSSFn))
487
488         sstnArray = np.array([ssin, ssfn]) # Steady-state time array
489         ssfnArray = np.array([avgFnVal, avgFnVal]) # Steady-state force array
490         ssdnArray = np.array([dn[timeLocSSIn], dn[timeLocSSFn]]) # Steady-state
491             ↪ displacement array
492
493     FMaxi = FMax['{}'.format(n)]
494     if str(FMaxi) == 'nan':

```

```

493     FmaxTimeLoc = np.nan
494     tFmax = np.nan
495     dFmax = np.nan # location in displacement where F = max
496
497 else:
498     FmaxTimeLoc = fn.loc[lamba x: x == FMaxi].index.values[0]
499     tFmax = tn[FmaxTimeLoc]
500     dFmax = dn[FmaxTimeLoc] # location in displacement where F = max
501
502     # slope calculation for 10 seconds prior to the max peel force
503     maxIndex = force['{}'.format(i)].loc[lamba x: x ==
504     ↪ FMax['{}'.format(i)]].index.values[0] # Location in the array for the
505     ↪ max force
506     x10 = extension['{}'.format(i)][maxIndex-100:maxIndex] # Array from
507     ↪ maxIndex - 100 (10 sec) to location of max force
508     y = force['{}'.format(i)][maxIndex-100:maxIndex] # Array from maxIndex -
509     ↪ 100 (10 sec) to location of max force
510     curveFit10, Params10, Rsqrd10val = Least_Squares(x10,y) # Perform least
511     ↪ squares and return
512
513     # slope calculation for 20 seconds prior to the max peel force
514     x20 = extension['{}'.format(i)][maxIndex-200:maxIndex] # Array from
515     ↪ maxIndex - 200 (20 sec) to location of max force
516     y = force['{}'.format(i)][maxIndex-200:maxIndex] # Array from maxIndex -
517     ↪ 200 (20 sec) to location of max force
518     curveFit20, Params20, Rsqrd20val = Least_Squares(x20,y) # Perform least
519     ↪ squares and return
520
521     # slope calculation for 30 seconds prior to the max peel force
522     x30 = extension['{}'.format(i)][maxIndex-300:maxIndex] # Array from
523     ↪ maxIndex - 300 (30 sec) to location of max force
524     y = force['{}'.format(i)][maxIndex-300:maxIndex] # Array from maxIndex -
525     ↪ 300 (30 sec) to location of max force
526     curveFit30, Params30, Rsqrd30val = Least_Squares(x30,y) # Perform least
527     ↪ squares and return
528
529     # Slope calculation from zero to max peel force
530     x0 = extension['{}'.format(i)][0:maxIndex] # Array from 0 to location of
531     ↪ max force
532     y = force['{}'.format(i)][0:maxIndex] # Array from 0 to location of max
533     ↪ force
534     curveFit0, Params0, Rsqrd0val = Least_Squares(x0,y) # Perform least
535     ↪ squares and return
536
537 FSSi = avgFnVal
538
539 """
540 Plot force vs time
541 """
542 fign, axn = plt.subplots()
543
544 # Plot data trace and Max & Steady-State vs time
545 axn.plot(tn, fn*1e3, '-', color='k', linewidth=1, markersize=2, label = 'Trace
546 ↪ {}'.format(n))
547 if str(FMaxi) == 'nan' and str(ssin) == 'nan':
548     pass # used to be 'continue' but an error showed up
549
550 if str(FMaxi) != 'nan':

```

```

536         axn.plot(tFmax, FMaxi*1e3, '.', color='r', linewidth=1, markersize=20,
537                 ↪ label = 'Max Peel - {:.4f} (mN)'.format(FMaxi*1e3))
538
539     if str(ssin) != 'nan':
540         axn.plot(ssfnArray, ssfnArray*1e3, '-', color='c', linewidth=3,
541                 ↪ markersize=2, label = 'Steady State - {:.4f} (mN)'.format(FSSI*1e3))
542
543     axn.axhline(y=0, color='k')
544     axn.set_xlabel('Time (sec)', fontsize=18)
545
546     if str(FMaxi) == 'nan':
547         axn.set_ylim(-0.5, fn.max()*1e3 + 1)
548     else:
549         axn.set_ylim(-0.5, FMaxi*1e3 + 1)
550     axn.set_ylabel('Force (mN)', fontsize=18)
551     axn.set_title(HumanIDi + ', ' + HumanGenderi + ', ' + 'Age: ' +
552                 ↪ str(HumanAgei) + ', ' + HumanLeftRighti + ', ' + HumanRegioni,
553                 ↪ fontsize=20)
554     axn.grid(True, which='both')
555     axn.legend(loc='best', prop={"size":12})
556     fign.savefig(os.path.join(filePath, 'Figures/' + 'Trace_{}'.format(str(n)) +
557                     ↪ '_F_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
558     fign.savefig(os.path.join(filePath, folder, 'Trace_{}'.format(str(n)) +
559                     ↪ '_F_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
560     fign.savefig(os.path.join(filePath, folder2, 'Trace_{}'.format(str(n)) +
561                     ↪ '_F_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
562     plt.close()
563
564     """
565     Plot force vs displacement with slope
566     """
567     fign, axn = plt.subplots()
568
569     # Plot data trace and Max & Steady-State vs displacement
570     axn.plot(dn, fn*1e3, '-', color='k', linewidth=1, markersize=2, label = 'Trace
571             ↪ #{}'.format(n), alpha = 0.3)
572
573     if str(FMaxi) == 'nan' and str(ssin) == 'nan':
574         pass # used to be 'continue' but an error showed up
575
576     if str(FMaxi) != 'nan':
577         axn.plot(dFmax, FMaxi*1e3, '.', color='r', linewidth=1, markersize=20,
578                 ↪ label = 'Max Peel - {:.4f} (mN)'.format(FMaxi*1e3))
579         axn.plot(x0, curveFit0*1e3, '-', color='tab:blue', linewidth=1,
580                 ↪ label=r'Curve fit 0 (s) y = {:.4f}x + {:.4f} (mN), $r^2$ =
581                 ↪ {:.4f}'.format(Params0[1]*1e3, Params0[0]*1e3, Rsqrd0val))
582         axn.plot(x30, curveFit30*1e3, '-', color='tab:orange', linewidth=2,
583                 ↪ label=r'Curve fit Max - 30 (s) y = {:.4f}x + {:.4f} (mN), $r^2$ =
584                 ↪ {:.4f}'.format(Params30[1]*1e3, Params30[0]*1e3, Rsqrd30val))
585         axn.plot(x20, curveFit20*1e3, '-', color='tab:purple', linewidth=3,
586                 ↪ label=r'Curve fit Max - 20 (s) y = {:.4f}x + {:.4f} (mN), $r^2$ =
587                 ↪ {:.4f}'.format(Params20[1]*1e3, Params20[0]*1e3, Rsqrd20val))
588         axn.plot(x10, curveFit10*1e3, '-', color='tab:green', linewidth=4,
589                 ↪ label=r'Curve fit Max - 10 (s) y = {:.4f}x + {:.4f} (mN), $r^2$ =
590                 ↪ {:.4f}'.format(Params10[1]*1e3, Params10[0]*1e3, Rsqrd10val))
591
592     if str(ssin) != 'nan':

```

```

576         axn.plot(ssdnArray, ssfnArray*1e3, '-', color='c', linewidth=3,
577                 ↪ markersize=2, label = 'Steady State - {:.4f} (mN)'.format(FSSi*1e3))
578
579     axn.axhline(y=0, color='k')
580     axn.set_xlabel('Disp (mm)', fontsize=18)
581
582     if str(FMaxi) == 'nan':
583         axn.set_ylim(-0.5, fn.max()*1e3 + 1)
584     else:
585         axn.set_ylim(-0.5, FMaxi*1e3 + 1)
586
587     axn.set_ylabel('Force (mN)', fontsize=18)
588     axn.set_title(HumanIDi + ', ' + HumanGenderi + ', ' + 'Age: ' +
589                 ↪ str(HumanAgei) + ', ' + HumanLeftRighti + ', ' + HumanRegioni,
590                 ↪ fontsize=20)
591     axn.grid(True, which='both')
592     axn.legend(loc='best', prop={"size":12})
593     fign.savefig(os.path.join(filePath, 'Figures/' + 'Trace_{}'.format(str(n)) +
594                 ↪ '_F_vs_disp.png'), dpi=300, bbox_inches='tight') # Save figure
595     fign.savefig(os.path.join(filePath, folder, 'Trace_{}'.format(str(n)) +
596                 ↪ '_F_vs_disp.png'), dpi=300, bbox_inches='tight') # Save figure
597     fign.savefig(os.path.join(filePath, folder2, 'Trace_{}'.format(str(n)) +
598                 ↪ '_F_vs_disp.png'), dpi=300, bbox_inches='tight') # Save figure
599     plt.close()
600
601     # Write the txt file with the force, extension, time data to the folder
602     """ Print the Instron Data """
603     print("\nWriting out the Instron data...")
604     filename = os.path.join(filePath, folder,
605                 ↪ 'Trace_{}_Instron_Data'.format(str(n)) + '.txt')
606     filename2 = os.path.join(filePath, folder2,
607                 ↪ 'Trace_{}_Instron_Data'.format(str(n)) + '.txt')
608     outfile = open(filename, 'w')
609     outfile2 = open(filename2, 'w')
610     DataFile = ['Human ID:\t{}'.format(HumanIDi),
611                 'Human Age:\t{}'.format(HumanAgei),
612                 'Human Gender:\t{}'.format(HumanGenderi),
613                 'Human Left/Right:\t{}'.format(HumanLeftRighti),
614                 'Human Region:\t{}'.format(HumanRegioni),
615                 'Post Mortem Hrs_Min:\t{}'.format(PostMortemHrs_Mini),
616                 'Date of Death:\t{}'.format(DateOfDeathi),
617                 'Time of Death:\t{}'.format(TimeOfDeathi),
618                 'Enucleation Date:\t{}'.format(DateOfTestingi),
619                 'Enucleation Time:\t{}'.format(TimeOfTestingi),
620                 'Diameter Posterior Anterior (in):\t{}'.format(DiameterPostAnti),
621                 'Diameter Nasal Temporal (in):\t{}'.format(DiameterNasTempi),
622                 'SSi (s):\t{}'.format(SSii),
623                 'SSf (s):\t{}'.format(SSfi),
624                 'Time Max (s):\t{}'.format(TMaxi),
625                 'Disp Max (mm):\t{}'.format(DispMaxi),
626                 'FMax (mN):\t{}'.format(FMaxi*1e3),
627                 'FSS (mN):\t{}'.format(FSSi*1e3),
628                 'Slope 10 seconds before max to max force value
629                 ↪ (mN/m):\t{}'.format(Slope10i*1e3),
630                 'R^2 for linear regression 10 seconds before
631                 ↪ max:\t{}'.format(Rsqrd10i),
632                 'Slope 20 seconds before max to max force value
633                 ↪ (mN/m):\t{}'.format(Slope20i*1e3),

```

```

623         'R^2 for linear regression 20 seconds before
        ↪ max:\t{}'.format(Rsqrd20i),
624         'Slope 30 seconds before max to max force value
        ↪ (mN/m):\t{}'.format(Slope30i*1e3),
625         'R^2 for linear regression 30 seconds before
        ↪ max:\t{}'.format(Rsqrd30i),
626         'Slope from 0 to max force value
        ↪ (mN/m):\t{}'.format(Slope0i*1e3),
627         'R^2 for linear regression from 0 to max:\t{}'.format(Rsqrd0i),
628         'Peel Video Name:\t{}'.format(PeelVideoNamei),
629         'Peel Video Hyperlink:\t{}'.format(PeelVideoHyperlinki),
630         'Video Comments:\t{}'.format(VideoCommentsi),
631         'Light Microscopy Images:\t{}'.format(LightMicroscopyImagesi),
632         '\n',
633         'Time (s)\tExtension (mm)\tForce (N)']
634
635     HeaderWrite = '\n'.join(item for item in DataFile)
636     outfile.write(HeaderWrite)
637     outfile2.write(HeaderWrite)
638     for i,j in enumerate(tn):
639         line = '\n%f\t%f\t%f' % (j, dn[i], fn[i])
640         outfile.write(line)
641         outfile2.write(line)
642     outfile.close()
643     outfile2.close()
644     print("\nDone!")
645     print("\nThe output file will be named '{}".format(filename) + "'")
646
647     """ Plot the extension rate for last test """
648     fig2, ax2 = plt.subplots()
649     ax2.plot(tn, gn, '-', color='k', linewidth=1, markersize=2, label = '1')
650     ax2.set_xlim(0,2.5)
651     ax2.set_xlabel('Time (sec)',fontsize=18)
652     ax2.set_ylim(0,0.045)
653     ax2.set_ylabel('Velocity (mm/s)',fontsize=18)
654     ax2.set_title('Data Trace', fontsize=20)
655     ax2.grid(True, which='both')
656     lines = fig2.gca().get_lines()
657     show = [0]
658     legend1 = ax2.legend([lines[i] for i in show],[lines[i].get_label() for i in show],
        ↪ prop={"size":12}, loc='best', title = 'Trace')
659     ax2.add_artist(legend1)
660     plt.show()
661     plt.close()
662
663
664     # In[2]
665
666     """ Plot each age group data on top of one another """
667     os.chdir(filePath)
668     fileNames = next(os.walk('.'))[1]
669     print(fileNames)
670
671     for i in fileNames:
672         if i == 'Figures' or i == 'StatisticsFigures':
673             # skip these two folders
674             continue
675

```

```

676 elif i != '30_59_Equator' and i != '30_59_Posterior' and i != '60_89_Equator' and
    ↪ i != '60_89_Posterior':
677     print(i, 'Age decade')
678     subPath = os.path.join(filePath, i)
679     (folderName, directory) = os.path.split(subPath)
680     os.chdir(subPath)
681     subTxtFiles = [x for x in glob.glob('*.txt')] # Posterior/Equator
682     show = []
683     maxVals = []
684
685     color1 = iter(cm.Set1(np.linspace(0,1,len(subTxtFiles))))
686     color2 = iter(cm.Set1(np.linspace(0,1,len(subTxtFiles))))
687     color3 = iter(cm.Set1(np.linspace(0,1,len(subTxtFiles))))
688     fign, axn = plt.subplots()
689     for j,k in enumerate(subTxtFiles):
690         c1 = next(color1)
691         c2 = next(color2)
692         c3 = next(color3)
693         """ Read in the csv file """
694         dfValsn = pd.read_csv(os.path.join(subPath, k), sep="\t", nrows=29,
    ↪ header=None, names=['Var', 'Attribute'])
695
696         """ File Attributes """
697         HID = dfValsn['Attribute'][0] # ID
698         HAGE = dfValsn['Attribute'][1] # Age
699         HG = dfValsn['Attribute'][2] # Gender
700         HLR = dfValsn['Attribute'][3] # Left/Right
701         HR = dfValsn['Attribute'][4] # Region
702         HSSi = float(dfValsn['Attribute'][12]) # Steady state start time
703         HSSf = float(dfValsn['Attribute'][13]) # Steady state final time
704         HTMax = float(dfValsn['Attribute'][14]) # Time @ max force
705         HFMax = float(dfValsn['Attribute'][16]) # Value at max force
706         HFSS = float(dfValsn['Attribute'][17]) # Value at steady state
707
708         dfn = pd.read_csv(os.path.join(subPath, k), sep="\t", header=30)
709         dfn.columns = ['Time', 'Extension', 'Force']
710         tn = dfn.Time
711         dn = dfn.Extension
712         force = dfn.Force
713
714         # SS Array
715         ssTimeArray = np.array([HSSi, HSSf])
716         ssValArray = np.array([HFSS, HFSS])
717
718         axn.plot(tn, force*1e3, '-', color=c1, linewidth=1, markersize=2, label =
    ↪ '{}', {}, Age: {}'.format(HID, HLR, HAGE), alpha = 1)
719         if str(HFMax) == 'nan' and str(HSSi) == 'nan':
720             continue
721
722         if str(HFMax) != 'nan':
723             axn.plot(HTMax, HFMax, '.', color=c2, linewidth=1, markersize=20,
    ↪ label = 'Max Peel - {:.4f} (mN)'.format(HFMax),
    ↪ path_effects=[pe.Stroke(linewidth=4, foreground='k'),
    ↪ pe.Normal())
724
725         if str(HSSi) != 'nan':

```



```

726         axn.plot(ssTimeArray, ssValArray, '-', color=c3, linewidth=3,
727                 ↪ markersize=2, label = 'Steady State - {:.4f} (mN)'.format(HFSS),
728                 ↪ path_effects=[pe.Stroke(linewidth=5, foreground='k'),
729                 ↪ pe.Normal())])
730
731         # append to the list for plot identification
732         show.append(fign.gca().get_lines())
733
734         # append the max value
735         maxVals.append(HFMax)
736
737     path, subFolder = os.path.split(subPath) # Extract the folder name
738     axn.axhline(y=0, color='k')
739     axn.set_xlabel('Time (sec)', fontsize=18)
740     axn.set_ylim(-0.5, max(maxVals) + 1)
741     axn.set_ylabel('Force (mN)', fontsize=18)
742     axn.set_title(subFolder, fontsize=20)
743     axn.grid(True, which='both')
744     for i,j in enumerate(show):
745         graphLine = []
746         step = len(j)
747         if i == 0:
748             for k in range(0, len(j), 1):
749                 graphLine.append(k)
750         elif i > 0:
751             for k in range(len(show[i-1]), len(j), 1):
752                 graphLine.append(k)
753         linesn = fign.gca().get_lines()
754         legendn = axn.legend([linesn[i] for i in
755                 ↪ graphLine], [linesn[i].get_label() for i in graphLine],
756                 ↪ prop={"size":10}, loc=i+1, title = 'Data')
757         axn.add_artist(legendn)
758     plt.show()
759     fign.savefig(os.path.join(subPath, '{}_All'.format(directory) +
760             ↪ '_F_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
761     plt.close()
762
763     elif i == '30_59_Equator' or i == '30_59_Posterior' or i == '60_89_Equator' or i
764     ↪ == '60_89_Posterior':
765         # Plot the groups for paper 3 but don't include the max and steady state
766         ↪ value (All legend items are in a single legend)
767         print(i)
768         subPath = os.path.join(filePath, i)
769         (folderName, directory) = os.path.split(subPath)
770         os.chdir(subPath)
771         subTxtFiles = [x for x in glob.glob('*.txt')] # Posterior/Equator
772         show = []
773         maxVals = []
774
775         color1 = iter(cm.rainbow(np.linspace(0,1,len(subTxtFiles))))
776         color2 = iter(cm.rainbow(np.linspace(0,1,len(subTxtFiles))))
777         color3 = iter(cm.rainbow(np.linspace(0,1,len(subTxtFiles))))
778         fign, axn = plt.subplots()
779
780         for j,k in enumerate(subTxtFiles):
781             c1 = next(color1)
782             c2 = next(color2)
783             c3 = next(color3)

```

```

776 """ Read in the csv file """
777 dfValsn = pd.read_csv(os.path.join(subPath, k), sep="\t", nrows=29,
778 ↪ header=None, names=['Var', 'Attribute'])
779
780 """ File Attributes """
781 HID = dfValsn['Attribute'][0] # ID
782 HAGE = dfValsn['Attribute'][1] # Age
783 HG = dfValsn['Attribute'][2] # Gender
784 HLR = dfValsn['Attribute'][3] # Left/Right
785 HR = dfValsn['Attribute'][4] # Region
786 HSSi = float(dfValsn['Attribute'][12]) # Steady state start time
787 HSSf = float(dfValsn['Attribute'][13]) # Steady state final time
788 HTMax = float(dfValsn['Attribute'][14]) # Time @ max force
789 HDispMax = float(dfValsn['Attribute'][15]) # Disp @ max force
790 HFMax = float(dfValsn['Attribute'][16]) # Value at max force
791 HFSS = float(dfValsn['Attribute'][17]) # Value at steady state
792
793 dfn = pd.read_csv(os.path.join(subPath, k), sep="\t", header=30)
794 dfn.columns = ['Time', 'Extension', 'Force']
795 tn = dfn.Time
796 dn = dfn.Extension # mm
797 force = dfn.Force*1e3 # N ---> mN
798
799 # SS Array
800 ssTimeArray = np.array([HSSi, HSSf])
801 ssValArray = np.array([HFSS, HFSS])
802 ssDispArray = np.array([dn[tn == HSSi].values[0] if HSSi is not np.nan
803 ↪ else np.nan,
804 ↪ dn[tn == HSSf].values[0] if HSSf is not np.nan
805 ↪ else np.nan])
806
807 axn.plot(dn, force, '-', color=c1, linewidth=1, markersize=2, label = '{}',
808 ↪ '{}', Age: {}'.format(HID, HLR, HAGE), alpha=0.3)
809
810 if str(HFMax) != 'nan':
811     maxIndex = force[tn == HTMax].index.values[0] # Location in the array
812     ↪ for the max force
813
814     # slope calculation for 10 seconds prior to the max peel force
815     x10 = dn[maxIndex-100:maxIndex] # Array from maxIndex - 100 (10 sec)
816     ↪ to location of max force
817     y = force[maxIndex-100:maxIndex] # Array from maxIndex - 100 (10 sec)
818     ↪ to location of max force
819     curveFit10, Params10, Rsqrd10 = Least_Squares(x10,y) # Perform least
820     ↪ squares and return
821
822     # slope calculation for 20 seconds prior to the max peel force
823     x20 = dn[maxIndex-200:maxIndex] # Array from maxIndex - 200 (20 sec)
824     ↪ to location of max force
825     y = force[maxIndex-200:maxIndex] # Array from maxIndex - 200 (20 sec)
826     ↪ to location of max force
827     curveFit20, Params20, Rsqrd20 = Least_Squares(x20,y) # Perform least
828     ↪ squares and return
829
830     # slope calculation for 30 seconds prior to the max peel force
831     x30 = dn[maxIndex-300:maxIndex] # Array from maxIndex - 300 (30 sec)
832     ↪ to location of max force

```

```

821     y = force[maxIndex-300:maxIndex] # Array from maxIndex - 300 (30 sec)
      ↪ to location of max force
822     curveFit30, Params30, Rsqrd30 = Least_Squares(x30,y) # Perform least
      ↪ squares and return
823
824     # Slope calculation from zero to max peel force
825     x0 = dn[0:maxIndex] # Array from 0 to location of max force
826     y = force[0:maxIndex] # Array from 0 to location of max force
827     curveFit0, Params0, Rsqrd0 = Least_Squares(x0,y) # Perform least
      ↪ squares and return
828
829     # axn.plot(x0, curveFit0*1e3, ':', color='black', linewidth=1,
      ↪ label=r'_Curve fit 0 (s) y = {:.4f}x + {:.4f}
      ↪ (mN)'.format(Params0[1]*1e3, Params0[0]*1e3), alpha = 1)
830     axn.plot(x30, curveFit30, '-', color='green', linewidth=2,
      ↪ label=r'_Curve fit Max - 30 (s) y = {:.4f}x + {:.4f} (mN), $r^2$
      ↪ = {:.4f}'.format(Params30[1], Params30[0], Rsqrd30), alpha = 1)
831     axn.plot(x20, curveFit20, '-', color='blue', linewidth=3,
      ↪ label=r'_Curve fit Max - 20 (s) y = {:.4f}x + {:.4f} (mN), $r^2$
      ↪ = {:.4f}'.format(Params20[1], Params20[0], Rsqrd20), alpha = 1)
832     axn.plot(x10, curveFit10, '-', color='red', linewidth=4,
      ↪ label=r'_Curve fit Max - 10 (s) y = {:.4f}x + {:.4f} (mN), $r^2$
      ↪ = {:.4f}'.format(Params10[1], Params10[0], Rsqrd10), alpha = 1)
833
834     # # Plot the max force value
835     # axn.plot(HDispMax, HFMax, '.', color=c1, linewidth=1, markersize=20,
      ↪ label = 'Max Peel - {:.4f} (mN)'.format(HFMax),
      ↪ path_effects=[pe.Stroke(linewidth=4, foreground='k'),
      ↪ pe.Normal()) #
836
837     # if str(HSSi) != 'nan':
838     #     axn.plot(ssDispArray, ssValArray, '-', color=c1, linewidth=3,
      ↪ markersize=2, label = '_Steady State - {:.4f} (mN)'.format(HFSS),
      ↪ path_effects=[pe.Stroke(linewidth=5, foreground='k'), pe.Normal()],
      ↪ alpha = 0.5)
839
840     # append to the list for plot identification
841     show.append(fign.gca().get_lines())
842
843     # append the max value
844     maxVals.append(HFMax)
845
846     path, subFolder = os.path.split(subPath) # Extract the folder name
847     axn.axhline(y=0, color='k')
848     axn.set_xlabel('Displacement (mm)', fontsize=18)
849     axn.set_xlim(0, 9) # focus in on just ramp up
850     axn.set_ylim(-0.5, max(maxVals) + 1)
851     axn.set_ylabel('Force (mN)', fontsize=18)
852     axn.set_title(subFolder, fontsize=20)
853     # axn.grid(True, which='both')
854
855     # where some data has already been plotted to ax
856     handles, labels = axn.get_legend_handles_labels()
857
858     # Manually add items to the legend
859     fit_0 = mpatches.Patch(color='black', label=r'Curvefit (0 - $Time_{Max})$')
860     fit_30 = mpatches.Patch(color='green', label=r'Curvefit (30 s before
      ↪ $Time_{Max})$')

```

```

861 fit_20 = mpatches.Patch(color='blue', label=r'Curvefit (20 s before
    → $Time_{Max})$')
862 fit_10 = mpatches.Patch(color='red', label=r'Curvefit (10 s before
    → $Time_{Max})$')
863
864 # handles is a list, so append manual patch
865 # handles.append(fit_0)
866 handles.append(fit_30)
867 handles.append(fit_20)
868 handles.append(fit_10)
869
870 axn.legend(handles=handles, loc='best')
871
872 plt.show()
873 fign.savefig(os.path.join(subPath, '{}_All'.format(directory) +
    → '_F_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
874 plt.close()
875
876 os.chdir(filePath)

```

## 1.4 Equations

### 1.4.1 Residual Vector

Residual vector,  $\mathbf{r}$  calculations used in optimization for elastic modulus Eq. (1.1) and adhesion models Eq. (1.2) are written as:

$$\mathbf{r} = \begin{cases} \langle m, \max(\mathbf{RF}) \rangle_{Exp} - \langle m, \max(\mathbf{RF}) \rangle_{Sim} & , \text{ Modulus} \\ \langle \max(\mathbf{RF}), \overline{\mathbf{SS}} \rangle_{Exp} - \langle \max(\mathbf{RF}), \overline{\mathbf{SS}} \rangle_{Sim} & , \text{ Cohesive} \end{cases} \quad (1.1)$$

$$(1.2)$$

where  $\mathbf{r}$  is residual vector,  $m$  is the slope of the data trace,  $\mathbf{RF}$  is the simulated reaction force,  $\overline{\mathbf{SS}}$  is the average steady-state peel force vector, and  $Exp$  and  $Sim$  are both experimental and simulated data respectively.

## 1.5 Bond Model

### 1.5.1 Python Batch File

Abaqus 2016 was written in python 2.7 and therefore argparse was not around to pass parameter as input. Instead, arguments are passed in as command line (cmd) space separated commands. This script calls the subprocess module to call Abaqus python from python 3.8.5.

</> **Script 4: Python file that sets up the model parameters as input into the Abaqus** </>  
model.

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Oct 19 15:22:26 2020
4
5 @author: Kiffer2
6
7 This Python script does the following
8
9     1) Select input parameters
10    2) Generates the filename/description
11    3) Calls Abaqus to create the .inp file w/ attributes & runs the job
12    4) Creates a folder with the filename
13    5) Extracts data from the Abaqus.odb file and creates two output files
14    ↪ (Field/Hist)
15    6) Plots the data
16    7) Moves all files that have the same filename
17
18 """
19 import os
20 import sys
21 import numpy as np
22 import pandas as pd
23 import itertools as it # iteration tools (product fcn)
24 from scipy import *
25 import scipy.optimize as opt
26 import lmfit as lf # lmfit
27 import pdb
28 import subprocess
29 import pprint
30
31 # Define the location of the Abaqus Working Directory
32 # specific folder path where this file is located
33 pythonScriptPath = os.getcwd()
34 abqWD, pythonFiles = os.path.split(pythonScriptPath) # split file path
35
36 # pythonScriptCreateINP_Run_ABQ (pS_ABQ)
37 pS_ABQ = os.path.join(pythonFiles, 'Bond_T1_EyeModel_Generate_Abaqus.py')
38 # pythonScriptExtract (pSE)
39 pSE = os.path.join(pythonFiles, 'Bond_T1_EyeModel_DataExtract.py')
40
41 # In[Job Info]
42
43 optE_V = False
44 optBondParams = False
45 optBondParams_no_db = True
46 sweep = False
47
48 if optE_V == True:
49     optimization = 'E_V'
50
51     """ Tie interface """
52     tieInterface = True
53
54     """ Objective Function Flags """
```

```

55     slopeFlag = True
56     maxForceFlag = True
57     ssForceFlag = False # Only used for damage
58
59     """ Traction separation """
60     BondStatus = False # If "False" then a tied interface will be used
61     PDFMStatus = False # If "False" then no post damage will occur
62
63
64 if optBondParams == True:
65     optimization = 'FNFSdbufnufs'
66
67     """ Tie interface """
68     tieInterface = False
69
70     """ Objective Function Flags """
71     slopeFlag = False
72     maxForceFlag = True
73     ssForceFlag = True # Only used for damage
74
75     """ Traction separation """
76     BondStatus = True # If "False" then a tied interface will be used
77     PDFMStatus = True # If "False" then no post damage will occur
78
79
80 if optBondParams_no_db == True:
81     optimization = 'FNFSufnufs'
82
83     """ Tie interface """
84     tieInterface = False
85
86     """ Objective Function Flags """
87     slopeFlag = False
88     maxForceFlag = True
89     ssForceFlag = True # Only used for damage
90
91     """ Traction separation """
92     BondStatus = True # If "False" then a tied interface will be used
93     PDFMStatus = True # If "False" then no post damage will occur
94
95
96 if sweep == True:
97     optimization = None
98
99     """ Tie interface """
100    tieInterface = False
101
102    """ Objective Function Flags """
103    slopeFlag = False
104    maxForceFlag = True
105    ssForceFlag = True # Only used for damage
106
107    """ Traction separation """
108    BondStatus = True # If "False" then a tied interface will be used
109    PDFMStatus = True # If "False" then no post damage will occur
110
111
112 """ Objective Function Error Formulation """

```

```

113 objFunErr = []
114 objFunErr.append('Difference') # Experimental - Simulated
115 objFunErr.append('Ratio') # Experimental/Simulated
116 # (Experimental - Simulated)/Experimental
117 objFunErr.append('Relative uncertainty')
118 # Change to specific optimization parameter. If 'None', no optimization
119 objErr = objFunErr[0]
120 print('Objective function error formulation = ', objErr)
121
122 # Calculation for error
123 ErrorCalculation = []
124 ErrorCalculation.append('two-point method') # Slope, Peak force, SS Force
125 ErrorCalculation.append('data-trace method') # interpolated array
126
127 errorMethod = ErrorCalculation[0]
128 print('Error method calculation = ', errorMethod)
129
130 ''' Symmetry '''
131 # Split model in half and multiply output by 2
132 symmetry = False # *bond doesn't allow symmetric BCs
133
134 ''' Simplified '''
135 # Remove the rigid body on the plastic tab and glue
136 simplified = True
137
138 ''' Gravity '''
139 # Turn gravity on/off
140 gravity = False # Keep off until model is updated
141
142 # In[Comparison Data Trace]
143 compareDataFolder = 'PeelDataCompare'
144 specificDataTrace = 'Trace_51_Instron_Data.txt' # Data trace number
145 timeBeforePeak = 20 # Default is 20 seconds
146 dataCompare = os.path.join(abqWD, compareDataFolder, specificDataTrace)
147 dfValsn = pd.read_csv(dataCompare, sep="\t", nrows=29, header=None,
148                       names=['Var', 'Attribute'])
149
150 """ File Attributes """
151 HID = dfValsn['Attribute'][0]
152 HAGE = dfValsn['Attribute'][1]
153 HG = dfValsn['Attribute'][2]
154 HLR = dfValsn['Attribute'][3]
155 HR = dfValsn['Attribute'][4]
156 HSSi = float(dfValsn['Attribute'][12])
157 HSSf = float(dfValsn['Attribute'][13])
158 HTMax = float(dfValsn['Attribute'][14])
159 HDispMax = float(dfValsn['Attribute'][15])
160 HFMax = float(dfValsn['Attribute'][16]) # (mN)
161 HFSS = float(dfValsn['Attribute'][17])
162 # (mN/m) slope from 20 seconds prior to max force value
163 HSlope20 = float(dfValsn['Attribute'][20])
164
165 dfn = pd.read_csv(os.path.join(dataCompare), sep="\t", header=30)
166 dfn.columns = ['Time', 'Extension', 'Force']
167 tn = dfn.Time
168 dn = dfn.Extension
169 df = dfn.Force # (N)
170

```

```

171 maxForceMeasured = HFMax # Value from data trace
172 maxSlopeMeasured = HSlope20 # slope from 20 seconds prior to max force value
173 SS_Measured = HFSS # simulated steady state force
174
175 # In[Functions]
176
177 if BondStatus == False and PDFMStatus == True:
178     print('Unable to have PDFM without *Bond')
179     sys.exit()
180
181 """ Tic Toc to determine runtime """
182 def tic():
183     #Homemade version of matlab tic and toc functions
184     import time
185     global startTime_for_tictoc
186     startTime_for_tictoc = time.time()
187
188 def toc():
189     import time
190     if 'startTime_for_tictoc' in globals():
191         print("Elapsed time is " + str(time.time() - startTime_for_tictoc) +
192               " seconds.")
193         timeDiff = time.time() - startTime_for_tictoc
194         return timeDiff
195     else:
196         print("Toc: start time not set")
197
198 try:
199     os.environ.pop('PYTHONIOENCODING')
200 except KeyError:
201     pass
202
203 # Import modules that plot/move all abq files to the new foldername
204 from ParameterSelection import ReadRAWDataTrace
205 from Bond_T1_Data_Plot import PlotAbqData
206 from Move_ABQ_Files_To_Folder import MoveAbqFiles
207 from Bond_T1_Residual import findResidual
208
209 newLine = '\n' + 77* '-' + '\n'
210
211 def jobAttributes():
212     """
213     Input: parameters used to create the filename and job description
214
215     Output: namei, fileName, JobDescription
216     """
217
218     # Build the fileName
219     fi = [] # initialize array
220     fi.append('{}'.format(namei))
221     fi.append('g') if gravity == True else ''
222     fi.append('sym') if symmetry == True else ''
223     fi.append('t{}'.format(time))
224     fi.append('E1{}'.format(e1Seedi[0]))
225     fi.append('E2{}'.format(e2Seedi[0]))
226
227     if simplified == False:
228         fi.append('PT{}'.format(ptSeedi[0]))

```



```

229         fi.append('G{}'.format(gSeedi[0]))
230
231     fi.append('V1{}'.format(v1Seedi[0]))
232     fi.append('V2{}'.format(v2Seedi[0]))
233     fi.append('R{}'.format(rSeedi[0]))
234     fi.append('F{}'.format(massScaleFactori[0]))
235     fi.append('MS{}'.format(massScaleTimeIncrementi[0]))
236     fi.append('RE{: .0e}'.format(RetinaYoungsModulus_i))
237
238     if optimization is not None:
239         if optimization.find('E_V') == -1:
240             fi.append('VE{: .0e}'.format(VitreousYoungsModulus_i))
241
242     # If True, then damage initiation, If optimization, get rid of the title
243     # (Not an integer anymore)
244     if BondStatus == True and optimization is None:
245         fi.append('FN{}'.format(int(FNi[0])))
246         fi.append('FS{}'.format(int(FSi[0])))
247
248     # If True, then damage evolution, If optimization, get rid of the title
249     # (Not an integer anymore)
250     if ((BondStatus == True) and (PDFMStatus == True) and
251         (optimization is None)):
252         fi.append('db{}'.format(int(dbi[0])))
253         fi.append('ufn{}'.format(int(ufni[0])))
254         fi.append('ufs{}'.format(int(ufsi[0])))
255
256     # .format(optimization)) optimization flag (I.e. RE, VE, FN, FS,
257     # ufn, or none)
258     fi.append('opt') if optimization is not None else ''
259     fi.append('TIE') if tieInterface == True else ''
260
261     if sweep == True:
262         # get rid of all attributes because a sweep is taking place
263         fi = fi[0]
264
265     """ Build file name and description """
266     fileName = ''.join(item for item in fi)
267     # fix header so no decimals, math show up in title
268     fileName = fileName.replace('+', '_').replace('-', '_').replace('.', '_')
269     jobNameString = 'Job Name - {}'.format(fileName)
270
271     # used for simplification of script
272     # Large value
273     multStrL = ('\n\tgeometric multiplier = 2*{}, \n\tbase value = {}, ' +
274               '\n\tmodel value = {}'.format(multStrL, baseValue, modelValue))
275     # Small value
276     multStrS = ('\n\tgeometric multiplier = 0.5*{}, \n\tbase value = {}, ' +
277               '\n\tmodel value = {}'.format(multStrS, baseValue, modelValue))
278
279     # Build the model description
280     si = [] # initialize array
281     si.append(newLine)
282     si.append('{} = model name'.format(namei))
283     si.append(jobNameString)
284     si.append('(g) - Gravity') if gravity == True else si.append('No Gravity')
285     # update name in list
286     si.append('(sym) SYMMETRIC model (XY) Plane') if symmetry == True else ''

```

```

287 # update name in list
288 si.append('(t) Simulated time {} (s)'.format(time))
289
290 # Eye Holder
291 si.append(('(E1) Eye holder outside edge seed size (Max) (SINGLE BIAS): ' +
292          + multStrS + ' (m)').format(*e1Seedi))
293 si.append(('(E2) Eye holder inside edge seed size (Min): ' + multStrS +
294          + ' (m)').format(*e2Seedi))
295
296 # If simplified is in the title, get rid of glue and plastic tab
297 if simplified == False:
298     si.append(('(PT) Plastic tab seed size: ' + multStrS +
299             + ' (m)').format(*ptSeedi))
300     si.append(('(G) Glue seed size: ' + multStrS +
301             + ' (m)').format(*gSeedi))
302
303 # Vitreous
304 si.append(('(V1) Vitreous seed size max (side edge seed set)-' +
305          + '(SINGLE BIAS): ' + multStrS + ' (m)').format(*v1Seedi))
306 si.append(('(V2) Vitreous seed size min (top edge in contact with ' +
307          + 'retina): ' + multStrS + ' (m)').format(*v2Seedi))
308
309 # Retina
310 si.append(('(R) Retina seed size: ' + multStrS + ' (m)').format(*rSeedi))
311
312 # Mass scale factor
313 si.append(('(F) Mass scale factor: ' + multStrL +
314          + ').format(*massScaleFactori))
315
316 # Mass scale time increment
317 si.append(('(MS) Mass scale time increment: ' + multStrS +
318          + ' (s)').format(*massScaleTimeIncrementi))
319
320 # Material properties (Young's Modulus)
321 si.append("(RE) Retina Young's Modulus: model value = {} (Pa)"
322          .format(RetinaYoungsModulus_i))
323 si.append("(VE) Vitreous Young's Modulus: model value = {} (Pa)"
324          .format(VitreousYoungsModulus_i))
325
326 # If True, then damage initiation
327 if BondStatus == True:
328     si.append(('(FN) FN: ' + multStrL + ' (N)').format(*FNi))
329     si.append(('(FS) FS: ' + multStrL + ' (N)').format(*FSi))
330
331 # If True, then damage evolution
332 if BondStatus == True and PDFMStatus == True:
333     si.append(('(db) Bead size: ' + multStrL +
334             + ' (m**2)').format(*dbi))
335     si.append(('(ufn) Normal displacement: ' + multStrL +
336             + ' (m)').format(*ufni))
337     si.append(('(ufs) Shear displacement: ' + multStrL +
338             + ' (m)').format(*ufsi))
339
340 # Optimization
341 if optimization is not None:
342     si.append('Optimization of {}'.format(optimization))
343     si.append('Objective function error formulation is the ' +
344             + '{} calculation'.format(objErr))

```

```

345         si.append('Objective error calculation is the {}'.format(errorMethod))
346
347     if optimization == None:
348         si.append('Parametric sweep')
349         si.append('Objective function error formulation is the ' +
350                 '{} calculation'.format(objErr))
351         si.append('Objective error calculation is the {}'.format(errorMethod))
352
353     # Tied interface
354     if tieInterface == True:
355         si.append('Tied interface between the Retina and the Vitreous')
356
357     # Data trace being compared for optimization
358     si.append('The data trace being compared is: {}'.format(specificDataTrace))
359
360
361     si.append(newLine)
362
363     # Job description
364     jobDescription = '\n'.join(item for item in si)
365
366     print(newLine)
367     print(fileName)
368     print(newLine)
369     print(jobDescription)
370
371     # Write a .txt file with the file attributes
372     outfile = open(os.path.join(abqWD, fileName + '.txt'), 'w')
373     line = ('The file name indicates what parameters were used to define ' +
374           'the model\n')
375     outfile.write(line)
376     line = '\n' + jobDescription + '\n'
377     outfile.write(line)
378     outfile.close()
379     print(outfile)
380     return namei, fileName, jobDescription
381
382 def GenerateAbaqusModels():
383     """
384     Function used to call Command Line (Windows Batch file)
385
386     Parameters
387     -----
388     fileName : abaqus job with paramters
389
390     """
391     # ----- Step 2 -----#
392     # Generates the filename/description
393     modelName, fileName, jobDescription = jobAttributes()
394
395     # Strip job description from spaces and new lines
396     # replace new lines, spaces, equal signs
397     jobDescription = jobDescription.replace(' ', 'SPACE')
398     jobDescription = jobDescription.replace('\n', 'NEWLINE')
399     jobDescription = jobDescription.replace('\t', 'TAB')
400     jobDescription = jobDescription.replace('=', 'EQUALSSIGN')
401
402     print(newLine)

```

```

403
404 # ----- Step 3 -----#
405 # Calls Abagus to create the job with the filename just created and
406 # run the job
407
408 # Strip spaces and make strings
409 ABQ = []
410 ABQ.append(pS_ABQ) # python 2.7 script
411
412 # gravity
413 ABQ.append(', '.join([i.strip(' ') for i in str(gravity).split(',')]))
414
415 # symmetry
416 ABQ.append(', '.join([i.strip(' ') for i in str(symmetry).split(',')]))
417
418 # Simplified model
419 ABQ.append(', '.join([i.strip(' ') for i in str(simplified).split(',')]))
420
421 ABQ.append(modelName) # model name
422 ABQ.append(fileName) # file name
423
424 # time
425 ABQ.append(', '.join([i.strip(' ') for i in str(time).split(',')]))
426
427 # eye holder seed size 1
428 ABQ.append(', '.join([i.strip(' ') for i in str(e1Seedi).split(',')]))
429
430 # eye holder seed size 2
431 ABQ.append(', '.join([i.strip(' ') for i in str(e2Seedi).split(',')]))
432
433 # plastic tab seed size
434 ABQ.append(', '.join([i.strip(' ') for i in str(ptSeedi).split(',')]))
435
436 # glue seed size
437 ABQ.append(', '.join([i.strip(' ') for i in str(gSeedi).split(',')]))
438
439 # vitreous seed 1 size
440 ABQ.append(', '.join([i.strip(' ') for i in str(v1Seedi).split(',')]))
441
442 # vitreous seed 2 size
443 ABQ.append(', '.join([i.strip(' ') for i in str(v2Seedi).split(',')]))
444
445 # retina seed size
446 ABQ.append(', '.join([i.strip(' ') for i in str(rSeedi).split(',')]))
447
448 # mass scale factor
449 ABQ.append(', '.join([i.strip(' ') for i in
450                       str(massScaleFactori).split(',')]))
451
452 # mass scale time
453 ABQ.append(', '.join([i.strip(' ') for i in
454                       str(massScaleTimeIncrementi).split(',')]))
455
456 # Retina Young's Modulus
457 ABQ.append(', '.join([i.strip(' ') for i in
458                       str(RetinaYoungsModulus_i).split(',')]))
459
460 # Vitreous Young's Modulus

```

```

461 ABQ.append(', '.join([i.strip(' ') for i in
462                      str(VitreousYoungsModulus_i).split(',')]))
463
464 # BondStatus
465 ABQ.append(', '.join([i.strip(' ') for i in
466                      str(BondStatus).split(',')]))
467 ABQ.append(', '.join([i.strip(' ') for i in str(FNi).split(',')])) # FN
468 ABQ.append(', '.join([i.strip(' ') for i in str(FSi).split(',')])) # FS
469
470 # PDFMStatus
471 ABQ.append(', '.join([i.strip(' ') for i in
472                      str(PDFMStatus).split(',')]))
473 ABQ.append(', '.join([i.strip(' ') for i in str(dbi).split(',')])) # db
474 ABQ.append(', '.join([i.strip(' ') for i in str(ufni).split(',')])) # ufn
475 ABQ.append(', '.join([i.strip(' ') for i in str(ufsi).split(',')])) # ufs
476
477 # Optimization None/optimized parameters
478 ABQ.append(', '.join([i.strip(' ') for i in str(optimization).split(',')]))
479
480 # Tied interface
481 ABQ.append(', '.join([i.strip(' ') for i in str(tieInterface).split(',')]))
482
483 # Model description
484 ABQ.append(jobDescription)
485
486 ABQ_parse_string = 'abaqus cae noGUI={} --' + (len(ABQ)-1)*' {'
487
488 # # Used for debugging, comment out to copy/paste output to cmd window
489 # # # to check and see if it works
490 # print(ABQ_parse_string.format(*ABQ))
491 # pdb.set_trace()
492
493 cmd = subprocess.Popen(ABQ_parse_string.format(*ABQ),
494                        cwd=r'{}'.format(abqWD), stdin=subprocess.PIPE,
495                        stdout=subprocess.PIPE, stderr=subprocess.PIPE,
496                        shell=True).communicate()[0]
497
498 print(newLine)
499 print('Abaqus has generated the .inp and executed the job')
500
501 # ----- Step 4 -----#
502 # Creates a folder with the filename
503 folderDirectory = os.path.join(abqWD, fileName)
504 if not os.path.exists(folderDirectory):
505     os.makedirs(folderDirectory)
506 dataDirectory = os.path.join(folderDirectory, 'Output')
507 if not os.path.exists(dataDirectory):
508     os.makedirs(dataDirectory)
509 figuresDirectory = os.path.join(dataDirectory, 'Figures')
510 if not os.path.exists(figuresDirectory):
511     os.makedirs(figuresDirectory)
512 print(newLine)
513 print('New file location:\n{} \n'.format(folderDirectory))
514
515 # ----- Step 5 -----#
516 """
517 Extracts data from the Abaqus.odb file and creates two output files
518 (Field/Hist). Create the name to be parsed into ABQ from the command

```

```

519 line through a subprocess
520 """
521 ABQ = []
522 ABQ.append(pSE)
523 ABQ.append(fileName)
524 ABQ.append(gravity)
525 ABQ.append(symmetry)
526 ABQ.append(simplified)
527 ABQ.append(BondStatus)
528 ABQ.append(PDFMStatus)
529
530 ABQ_parse_string = 'abaqus python' + len(ABQ)*' {'
531
532 # # # Used for debugging, comment out to copy/paste output to cmd window
533 # # # to check and see if it works
534 # print(ABQ_parse_string.format(*ABQ))
535 # pdb.set_trace()
536
537 cmd = subprocess.Popen(ABQ_parse_string.format(*ABQ),
538                        cwd=r'{}'.format(abqWD), stdin=subprocess.PIPE,
539                        stdout=subprocess.PIPE, stderr=subprocess.PIPE,
540                        shell=True).communicate()[0]
541
542 print(newLine)
543 print('Abaqus has extracted Field/History output: ' +
544       '\n{} \n'.format(dataDirectory))
545
546 # ----- Step 6 -----#
547 # Plot data and store it in the variable name folder under "Figures"
548 print(fileName)
549 print(dataDirectory)
550 PlotAbqData(fileName, dataDirectory, dataCompare, BondStatus,
551             PDFMStatus)
552 print(newLine)
553 print('New data plots:\n{} \n'.format(figuresDirectory))
554
555 # ----- Step 7 -----#
556 # Move all abaqus files to the folder with the same name
557 MoveAbqFiles(fileName, folderDirectory, abqWD)
558 print(newLine)
559 print('Files have been moved to: \n{} \n'.format(dataDirectory))
560
561 # ----- Step 8 (Error for minimization) -----#
562 maxForceTime = 100 # s
563 # slope is (mN/m)
564 residVals = findResidual(fileName, dataDirectory, maxForceTime,
565                          dataCompare, objErr, slopeFlag, maxForceFlag,
566                          ssForceFlag)
567
568 # Unpack
569 slopeSimulated = residVals[0]
570 maxForceSimulated = residVals[1]
571 SSmeanSimulated = residVals[2]
572 SSmedianSimulated = residVals[3]
573 y_new_exp_disp = residVals[4]
574 y_new_sim_disp = residVals[5]
575
576 # (return slope, force, and maxucrt @ specified displacement)
577 fcnReturn = []

```

```

577     fcnReturn.append(fileName)
578     fcnReturn.append(slopeSimulated)
579     fcnReturn.append(maxForceSimulated)
580     fcnReturn.append(SSmeanSimulated)
581     fcnReturn.append(SSmedianSimulated)
582     fcnReturn.append(y_new_exp_disp)
583     fcnReturn.append(y_new_sim_disp)
584     return fcnReturn
585
586
587 def writeOutputData(fileNameList):
588     print("\nWriting out the Reaction Force data...")
589     filename = os.path.join(abqWD, 'FEAAttributes' + '.txt')
590     outfile = open(filename, 'w')
591     sep = '\t'
592     Header = [] # List of items for the Header
593     Header.append('FileName')
594     Header.append('Time')
595     Header.append('E1')
596     Header.append('E2')
597     Header.append('PT')
598     Header.append('G')
599     Header.append('V1')
600     Header.append('V2')
601     Header.append('R')
602     Header.append('F')
603     Header.append('MS')
604     Header.append('RE')
605     Header.append('VE')
606     Header.append('FN')
607     Header.append('FS')
608     Header.append('db')
609     Header.append('UFN')
610     Header.append('UFS')
611     Header.append('BondStatus')
612     Header.append('PDFMStatus')
613     Header.append('Optimization')
614     Header.append('TIE')
615     Header.append('errorListL2Norm')
616     Header.append('ObjectiveFunction')
617     Header.append('maxSlopeSimulated')
618     Header.append('maxForceSimulated')
619     Header.append('SSmeanSimulated')
620     Header.append('simTime')
621     line = sep.join(item for item in Header)
622     outfile.write(line)
623     outfile.write('\n')
624     outfile.write('\t'.join(str(item) for item in attributeArray_0))
625     for i in list(fileNameList):
626         outfile.write('\n')
627         tempList = [str(i[0])] # filename
628         for j in list(i[1]):
629             tempList.append(str(j)) # file attributes
630         tempList.append(str(i[2])) # sim time
631         outfile.write('\t'.join(str(item) for item in tempList))
632     outfile.close()
633     print("\nDone!")
634     print("\nThe output file will be named '{}".format(filename) + "'")

```

```

635     print("\nIt will be in the same working directory as your Abaqus model\n")
636
637     # Print File of tests ran in order
638     print("\nWriting out the Reaction Force data...")
639     filename = os.path.join(abqWD, 'FEAFileList' + '.txt')
640     outfile = open(filename, 'w')
641     line = 'FileName'
642     outfile.write(line)
643     for i in list(fileNameList):
644         line = '\n%s' % (i[0])
645         outfile.write(line)
646     outfile.close()
647     print("\nDone!")
648     print("\nThe output file will be named '{}'.format(filename) + ".")
649     print("\nIt will be in the same working directory as your Abaqus model\n")
650
651
652
653 if __name__ == '__main__':
654     # Run the function
655
656     # ----- Step 1 -----#
657     # T1
658     name = ['T1']
659
660     paramSelect = ReadRAWDataTrace(dataCompare, abqWD, timeBeforePeak)
661
662     t0, t1, tshift, fe = paramSelect # Unpack variables
663
664     if t0 > tshift:
665         # If the t1 value is greater than tshift, use tshift for
666         # the simulation time
667         # Shouldn't have to do this as this issue has been handled
668         t0 = tshift
669         print('updated the time to be the shift value')
670
671     # Determine which time to use (Max value or steady state)
672     if optE_V == True:
673         time = int(t0)
674
675     elif optBondParams == True or sweep == True:
676         time = int(t1)
677
678     elif optBondParams_no_db == True or sweep == True:
679         time = int(t1)
680
681     # Select input parameters
682
683     ''' First round of optimized material properties '''
684     VitreousYoungsModulus_0 = 399.4216617623035
685     FNValOpt = -8
686     FSValOpt = -8
687     dbValOpt = -25
688     ufnValOpt = -8
689     ufsValOpt = -8
690
691     """ Retina Young's Modulus """
692     RetinaYoungsModulus_0 = 11120.0 # Pa Optimized with the vitreous

```



```

693
694 """ Eye holder inside edge """
695 e1Seed_0 = 1 # Base seed
696 e1SeedArray = [] # Array of multipliers
697 n = 12 # number of increments
698 for i in range(11, n):
699     # Decrease mesh seed by a factor of 2
700     e1SeedArray.append([i, e1Seed_0, e1Seed_0*(0.5)**i])
701
702 """ Eye holder outside edge """
703 # This will most likely never get smaller (saves computational time)
704 e2Seed_0 = 1 # Base seed
705 e2SeedArray = []
706 n = 7 # number of increments
707 for i in range(6, n):
708     # Decrease mesh seed by a factor of 2
709     e2SeedArray.append([i, e2Seed_0, e2Seed_0*(0.5)**i])
710
711 """ Plastic tab """
712 ptSeed_0 = 1 # Plastic tab seed size
713 ptSeedArray = [] # Array of multipliers
714 n = 7 # number of increments
715 for i in range(6, n):
716     # Decrease mesh seed by a factor of 2
717     ptSeedArray.append([i, ptSeed_0, ptSeed_0*(0.5)**i])
718
719 """ Glue """
720 gSeed_0 = 1 # Glue seed size
721 gSeedArray = [] # Array of multipliers
722 n = 8 # number of increments
723 for i in range(7, n):
724     # Decrease mesh seed by a factor of 2
725     gSeedArray.append([i, gSeed_0, gSeed_0*(0.5)**i])
726
727 """ Vitreous """
728 # smaller seed size
729 v1Seed_0 = 1 # Vitreous (max seed size)
730 v1SeedArray = [] # Array of multipliers
731 # n = 11 # number of increments
732 # for i in range(10, n):
733 #     # Decrease mesh seed by a factor of 2
734 #     v1SeedArray.append([i, v1Seed_0, v1Seed_0*(0.5)**i])
735
736 # Comment out when parameters have been optimized
737 v1ValOpt = 11.38 # 10
738 v1SeedArray.append([v1ValOpt, v1Seed_0, v1Seed_0*(0.5)**v1ValOpt])
739
740 # larger seed size (should be factor of 4 times smaller ## 2 numbers)
741 v2Seed_0 = 1 # Vitreous (min seed size)
742 v2SeedArray = [] # Array of multipliers
743 n = 9 # number of increments
744 for i in range(8, n):
745     # Decrease mesh seed by a factor of 2
746     v2SeedArray.append([i, v2Seed_0, v2Seed_0*(0.5)**i])
747
748 """ Retina """
749 rSeed_0 = 1 # Base seed
750 rSeedArray = [] # Array of multipliers

```

```

751 # n = 11 # number of increments
752 # for i in range(10, n):
753 #     # Decrease mesh seed by a factor of 2
754 #     rSeedArray.append([i, rSeed_0, rSeed_0*(0.5)**i])
755
756 rValOpt = 11.3275 # 10 #
757 rSeedArray.append([rValOpt, rSeed_0, rSeed_0*(0.5)**rValOpt])
758
759
760 """ mass scale factor """
761 massScaleFactor_0 = 1
762 massScaleFactorArray = [] # Array of multipliers
763 n = 1 # number of increments
764 for i in range(0, n):
765     # Increase by a factor of 2
766     massScaleFactorArray.append([i, massScaleFactor_0,
767                                 massScaleFactor_0*2**i])
768
769 """ mass scale time increment """
770 massScaleTimeIncrement_0 = 1
771 massScaleTimeArray = [] # multiplier and value
772 n = 8 # number of increments
773 for i in range(7, n):
774     # Decrease by a factor of 2
775     massScaleTimeArray.append([i, massScaleTimeIncrement_0,
776                                massScaleTimeIncrement_0*(0.5)**i])
777
778 if massScaleTimeIncrement_0 == 0:
779     print('No Mass Scaling... This will take a while...ABAQUS is ' +
780           'deciding for us')
781
782 """ FN """
783 FN_0 = 1
784 FNArray = [] # Array of multipliers
785 # n = 10 # 10 works when using max stress criteria
786 # for i in range(9, n):
787 #     # Increase by a factor of 2
788 #     FNArray.append([i, FN_0, FN_0*(2)**i])
789
790 # Comment out when parameters have been optimized
791 FNArray.append([FNValOpt, FN_0, FN_0*(2)**FNValOpt])
792
793 """ FS """
794 FS_0 = 1
795 FSArray = [] # Array of multipliers
796 # n = 10
797 # for i in range(9, n):
798 #     # Increase by a factor of 2
799 #     FSArray.append([i, FS_0, FS_0*(2)**i])
800
801 # Comment out when parameters have been optimized
802 FSArray.append([FSValOpt, FS_0, FS_0*(2)**FSValOpt])
803
804 """ db """
805 db_0 = 1
806 dbArray = [] # Array of multipliers
807 # n = 10
808 # for i in range(9, n):

```

```

809     #     # Increase by a factor of 2
810     #     dbArray.append([i, db_0, db_0*(2)**i])
811
812     # Comment out when parameters have been optimized
813     dbArray.append([dbVal0pt, db_0, db_0*(2)**dbVal0pt])
814
815     """ ufn """
816     ufn_0 = 1
817     ufnArray = [] # Array of multipliers
818     # n = -8
819     # for i in range(-9, n):
820     #     # Increase by a factor of 2
821     #     ufnArray.append([i, ufn_0, ufn_0*(2)**i])
822
823     ufnArray.append([ufnVal0pt, ufn_0, ufn_0*(2)**ufnVal0pt])
824
825     """ ufs """
826     ufs_0 = 1
827     ufsArray = [] # Array of multipliers
828     # n = -8
829     # for i in range(-9, n):
830     #     # Increase by a factor of 2
831     #     ufsArray.append([i, ufs_0, ufs_0*(2)**i])
832
833     ufsArray.append([ufsVal0pt, ufs_0, ufs_0*(2)**ufsVal0pt])
834
835     errorList = np.nan # initial error
836
837     """ Attribute Array Initial Values """
838     attributeArray_0 = []
839     attributeArray_0.append('BaseVals')
840     attributeArray_0.append(time)
841     attributeArray_0.append(e1Seed_0)
842     attributeArray_0.append(e2Seed_0)
843     attributeArray_0.append(ptSeed_0)
844     attributeArray_0.append(gSeed_0)
845     attributeArray_0.append(v1Seed_0)
846     attributeArray_0.append(v2Seed_0)
847     attributeArray_0.append(rSeed_0)
848     attributeArray_0.append(massScaleFactor_0)
849     attributeArray_0.append(massScaleTimeIncrement_0)
850     attributeArray_0.append(RetinaYoungsModulus_0)
851     attributeArray_0.append(VitreousYoungsModulus_0)
852     attributeArray_0.append(BondStatus)
853     attributeArray_0.append(FN_0)
854     attributeArray_0.append(FS_0)
855     attributeArray_0.append(db_0)
856     attributeArray_0.append(PDFMStatus)
857     attributeArray_0.append(ufn_0)
858     attributeArray_0.append(ufs_0)
859     attributeArray_0.append(optimization)
860     attributeArray_0.append(tieInterface)
861     attributeArray_0.append(errorList)
862     attributeArray_0.append(objErr)
863     attributeArray_0.append('simTime')
864
865
866     fileNameList = [] # List of files

```

```

867 counter = 0
868
869 if optimization is not None:
870     """ If the optimization variable is not "None" then optimize the
871     specific variable beins passed through """
872
873     name = name[0]
874
875     # BondStatus = True # interested in bonding
876
877     # # post damage failure model (If False, ignore db, ufs, and ufn,
878     # # otherwise include them)
879     # pdfm = False
880
881     # Optimization method
882     # optName = 'NM' # Nelder-mead
883     # optName = 'P' # Powell
884     optName = 'C' # COBYLA
885     # optName = 'L' # LBFGSB
886     # optName = 'T' # Truncated Newton
887     # optName = 'S' # SLSQP
888     # optName = 'TC' # Trust-Constr
889
890     name0 = '_' .join([name, optName]) # used for optimization
891
892 def FEA_Residual(pars, data=None):
893     # Global variables
894     global counter
895     global name
896     global name0
897     global fileNameList
898     global time
899     global namei
900     global e1Seedi
901     global e2Seedi
902     global ptSeedi
903     global gSeedi
904     global v1Seedi
905     global v2Seedi
906     global rSeedi
907     global massScaleFactori
908     global massScaleTimeIncrementi
909     global RetinaYoungsModulus_i
910     global VitreousYoungsModulus_i
911     global FNi
912     global FSi
913     global dbi
914     global ufni
915     global ufsi
916
917     # Parameters used for optimization
918     global errorList
919
920     print('Iteration # ', counter)
921
922     tic() # Start time
923
924     e1Seedi = e1SeedArray[0] # Default array

```

```

925 e2Seedi = e2SeedArray[0] # Default array
926 ptSeedi = ptSeedArray[0] # Default array
927 gSeedi = gSeedArray[0] # Default array
928 v1Seedi = v1SeedArray[0] # Default array
929 v2Seedi = v2SeedArray[0] # Default array
930 rSeedi = rSeedArray[0] # Default array
931 massScaleFactori = massScaleFactorArray[0] # Default array
932 massScaleTimeIncrementi = massScaleTimeArray[0] # Default array
933 RetinaYoungsModulus_i = RetinaYoungsModulus_0 # Default value
934 VitreousYoungsModulus_i = VitreousYoungsModulus_0 # Default value
935 FNi = FNArray[0] # Default array
936 FSi = FSArray[0] # Default array
937 dbi = dbArray[0] # Default array
938 ufn_i = ufnArray[0] # Default array
939 ufsi = ufsArray[0] # Default array
940
941 # Extract the unknown parameters from the pars class variable
942 # Determine the multiplier for the title
943 for key, value in pars.items():
944
945     if key.find('ER') >= 0:
946         """ Retina Young's Modulus """
947         val = value.value
948         RetinaYoungsModulus_i = val
949
950     elif key.find('EV') >= 0:
951         """ Vitreous Young's Modulus """
952         val = value.value
953         VitreousYoungsModulus_i = val
954
955     elif key.find('FN') >= 0:
956         """  $F_n$  """
957         val = value.value
958         mult = np.log(val)/np.log(2) # multiplier
959         FNi = [mult, FN_0, val]
960
961     elif key.find('FS') >= 0:
962         """  $F_S$  """
963         val = value.value
964         mult = np.log(val)/np.log(2) # multiplier
965         FSi = [mult, FS_0, val]
966
967     elif key.find('db') >= 0:
968         """  $db$  """
969         val = value.value
970         mult = np.log(val)/np.log(2) # multiplier
971         dbi = [mult, db_0, val]
972
973     elif key.find('ufn') >= 0:
974         """  $u_{fn}$  """
975         val = value.value
976         mult = np.log(val)/np.log(2) # multiplier
977         ufn_i = [mult, ufn_0, val]
978
979     elif key.find('ufs') >= 0:
980         """  $u_{fs}$  """
981         val = value.value
982         mult = np.log(val)/np.log(2) # multiplier

```

```

983         ufsi = [mult, ufs_0, val]
984
985     # Keep track of simulation results by unique names with the count
986     # number. Comment out the second part to save file space if you
987     # are not interested in saving every single simulation
988     namei = name0 #+ str(counter)
989
990     # Error of the simulation
991     L2Normi = np.sqrt(np.dot(errorList, errorList))
992
993     # multipliers to be appended to the output file to show changes
994     # in parameters
995     aAM = [] # attributeArrayMultipliar
996     aAM.append(time)
997     aAM.append(e1Seedi[0])
998     aAM.append(e2Seedi[0])
999     aAM.append(ptSeedi[0])
1000     aAM.append(gSeedi[0])
1001     aAM.append(v1Seedi[0])
1002     aAM.append(v2Seedi[0])
1003     aAM.append(rSeedi[0])
1004     aAM.append(massScaleFactori[0])
1005     aAM.append(massScaleTimeIncremeni[0])
1006     aAM.append(RetinaYoungsModulus_i)
1007     aAM.append(VitreousYoungsModulus_i)
1008     aAM.append(BondStatus)
1009     aAM.append(FNi[0])
1010     aAM.append(FSi[0])
1011     aAM.append(PDFMStatus)
1012     aAM.append(dbi[0])
1013     aAM.append(ufni[0])
1014     aAM.append(ufsi[0])
1015     aAM.append(optimization)
1016     aAM.append(tieInterface)
1017     aAM.append(L2Normi)
1018     aAM.append(objErr)
1019
1020     # Call the function
1021     # Runs jobs and saves file names
1022     funReturn = GenerateAbaqusModels()
1023     fileName = funReturn[0]
1024     maxSlopeSimulated = funReturn[1]
1025     maxForceSimulated = funReturn[2]
1026     SSmeanSimulated = funReturn[3]
1027     SSmedianSimulated = funReturn[4]
1028     y_new_exp_disp = funReturn[5]
1029     y_new_sim_disp = funReturn[6]
1030
1031     # add the simulated outputs to the data file
1032     aAM.append(maxSlopeSimulated)
1033     aAM.append(maxForceSimulated)
1034     aAM.append(SSmedianSimulated)
1035
1036     # Determine the measure of error used for optimization
1037     # Let the data trace being passed in act as the comparison
1038     maxSlopeMeasured, maxForceMeasured = data
1039
1040     # Error calculation

```

```

1041 errorDict = {} # Dictionary
1042 if objErr == 'Difference':
1043     errorDict['slope'] = (maxSlopeMeasured - maxSlopeSimulated) if
        ↪ slopeFlag == True else []
1044     errorDict['maxForce'] = (maxForceMeasured - maxForceSimulated) if
        ↪ maxForceFlag == True else []
1045     errorDict['ssForce'] = (SS_Measured - SSmeanSimulated) if
        ↪ ssForceFlag == True else []
1046 elif objErr == 'Ratio':
1047     errorDict['slope'] = (1 - maxSlopeMeasured / maxSlopeSimulated) if
        ↪ slopeFlag == True else []
1048     errorDict['maxForce'] = (1 - maxForceMeasured / maxForceSimulated) if
        ↪ maxForceFlag == True else []
1049     errorDict['ssForce'] = (1 - SS_Measured / SSmeanSimulated) if
        ↪ ssForceFlag == True else []
1050 elif objErr == 'Relative uncertainty':
1051     errorDict['slope'] = ((maxSlopeMeasured -
        ↪ maxSlopeSimulated)/maxSlopeMeasured) if slopeFlag == True else []
1052     errorDict['maxForce'] = ((maxForceMeasured -
        ↪ maxForceSimulated)/maxForceMeasured) if maxForceFlag == True else
        ↪ []
1053     errorDict['ssForce'] = ((SS_Measured -
        ↪ SSmedianSimulated)/SS_Measured) if ssForceFlag == True
        ↪ else []
1054 else:
1055     print('Error in MaxForceError')
1056     sys.exit()
1057
1058 # Error array values
1059 errorList = list(errorDict.values()) # convert to list
1060 errorList = [x for x in errorList if x] # get rid of empty values
1061 L2Normi = np.sqrt(np.dot(errorList, errorList))
1062
1063 # Calculate residual
1064 residual = y_new_exp_disp - y_new_sim_disp # residual
1065
1066 # Calculate L2Norm
1067 L2Norm = np.sqrt(np.dot(residual, residual))
1068
1069 simulationTime = toc() # Determine run time
1070 # appends the fileName & File Attributes
1071 fileNameList.append([fileName, aAM,
1072     simulationTime])
1073 print('{} Error calculation: '.format(objErr), errorList)
1074 print('L2 norm objective calculation', L2Normi)
1075 print('L2 Norm residual', L2Norm)
1076 print('Done\n\n')
1077 counter += 1
1078
1079 # Determine which calculation is going to be used for optimization
1080 if errorMethod == 'two-point method':
1081     FEA_Residual = errorList
1082 elif errorMethod == 'data-trace method':
1083     FEA_Residual = residual
1084
1085 return FEA_Residual
1086
1087 maxFuncEval = 200

```

```

1088     tolVal = 1e-4
1089
1090     # Use the data variable to input the max slope and force from the
1091     # known data trace
1092     data = [maxSlopeMeasured, maxForceMeasured]
1093
1094     # Initial, Upper, and Lower bounds for parameters
1095
1096     # Young's Modulus - Retina
1097     ER_i = 5000 # Pa
1098     ER_LB = 50 # Pa
1099     ER_UB = 11000 # Pa
1100
1101     # Young's Modulus - Vitreous
1102     EV_i = 172 # Pa (Prony series calculation)
1103     # EV_i = 500 # Pa (Trying higher initial guess)
1104     EV_LB = 50 # Pa
1105     EV_UB = 2100 # Pa
1106
1107     # Traction-Separation Behavior
1108     FN_i = 2**-6.083194128688112 # Force [N]
1109     FN_LB = 2**-12 # Force [N]
1110     FN_UB = 2**-5 # Force [N]
1111
1112     FS_i = 2**-5.372267011053392 # Force [N]
1113     FS_LB = 2**-12 # Force [N]
1114     FS_UB = 2**-5 # Force [N]
1115
1116     db_i = 2**-25 # Area [m**2]
1117     db_LB = 2**-30 # Area [m**2]
1118     db_UB = 2**-20 # Area [m**2]
1119
1120     # Damage Initiation Behavior
1121     ufn_i = 2**-6.470108340925128 # Disp [m]
1122     ufn_LB = 2**-12 # Disp [m]
1123     ufn_UB = 2**-5 # Disp [m]
1124
1125     ufs_i = 2**-6.470108340925128 # -11.169201472977056 # Disp [m]
1126     ufs_LB = 2**-20 # Disp [m]
1127     ufs_UB = 2**-5 # Disp [m]
1128
1129     # Specify parameters
1130     fit_params = lf.Parameters() # intialize the class for parameters
1131
1132     # Retina young's modulus
1133     if optimization.find('E_R') >= 0:
1134         fit_params.add('ER', value = ER_i, min=ER_LB, max=ER_UB, vary=True)
1135
1136     # Vitreous Young's Modulus
1137     if optimization.find('E_V') >= 0:
1138         fit_params.add('EV', value = EV_i, min=EV_LB, max=EV_UB, vary=True)
1139
1140     # parameter for making the retina stiffer than the vitreous
1141     if optimization.find('E_R') >= 0 and optimization.find('E_V') >= 0:
1142         fit_params.add('StiffDelta', value = 0.01, min=0, vary=True)
1143         # Constraint to allow vitreous to be not as stiff as the retina
1144         fit_params.add('stiffnessConstraint', expr = 'EV - StiffDelta')
1145

```



```

1146 # FN
1147 if optimization.find('FN') >= 0:
1148     fit_params.add('FN', value = FN_i, min=FN_LB, max=FN_UB,
1149                     vary=True)
1150
1151 # FS
1152 if optimization.find('FS') >= 0:
1153     fit_params.add('FS', value = FS_i, min=FS_LB, max=FS_UB,
1154                     vary=True)
1155
1156 # db
1157 if optimization.find('db') >= 0:
1158     fit_params.add('db', value = db_i, min=db_LB, max=db_UB,
1159                     vary=True)
1160
1161 # ufn
1162 if optimization.find('ufn') >= 0:
1163     fit_params.add('ufn', value = ufn_i, min=ufn_LB, max=ufn_UB,
1164                     vary=True)
1165
1166 # ufs
1167 if optimization.find('ufs') >= 0:
1168     fit_params.add('ufs', value = ufs_i, min=ufs_LB, max=ufs_UB,
1169                     vary=True)
1170
1171 # Set up minimization class
1172 minClass = lf.Minimizer(FEA_Residual, fit_params,
1173                          fcn_kws={'data': data},
1174                          max_nfev = maxFuncEval) # fcn_args=(x,),
1175
1176 # (Different methods can be used here) Uses an array
1177 # out = minClass.leastsq() # Levenberg-Marquardt
1178
1179 # single scalar value
1180 # out = minClass.scalar_minimize(method='Nelder-Mead', tol=tolVal)
1181
1182 # single scalar value (if the objective function returns an array,
1183 # the sum of the squares of the array will be used (L2Norm))
1184 out = minClass.scalar_minimize(method='Cobyla', tol=tolVal)
1185
1186 lf.report_fit(out) # modelpars=p_true, show_correl=True
1187
1188 # Write data to txt files
1189 writeOutputData(fileNameList)
1190
1191 else:
1192
1193     # Number of simulations to perform (Simulation Batch Total)
1194     SBT = []
1195     SBT.append(len(name))
1196     SBT.append(len(e1SeedArray))
1197     SBT.append(len(e2SeedArray))
1198     SBT.append(len(ptSeedArray))
1199     SBT.append(len(gSeedArray))
1200     SBT.append(len(v1SeedArray))
1201     SBT.append(len(v2SeedArray))
1202     SBT.append(len(rSeedArray))
1203     SBT.append(len(massScaleFactorArray))

```

```

1204 SBT.append(len(massScaleTimeArray))
1205 SBT.append(len(FNArray))
1206 SBT.append(len(FSArray))
1207 SBT.append(len(dbArray))
1208 SBT.append(len(ufnArray))
1209 SBT.append(len(ufsArray))
1210
1211 ZipArray = []
1212 ZipArray.append(max(SBT)*name)
1213 ZipArray.append(max(SBT)*e1SeedArray)
1214 ZipArray.append(max(SBT)*e2SeedArray)
1215 ZipArray.append(max(SBT)*ptSeedArray)
1216 ZipArray.append(max(SBT)*gSeedArray)
1217 ZipArray.append(max(SBT)*v1SeedArray)
1218 ZipArray.append(max(SBT)*v2SeedArray)
1219 ZipArray.append(max(SBT)*rSeedArray)
1220 ZipArray.append(max(SBT)*massScaleFactorArray)
1221 ZipArray.append(max(SBT)*massScaleTimeArray)
1222 ZipArray.append(max(SBT)*FNArray)
1223 ZipArray.append(max(SBT)*FSArray)
1224 ZipArray.append(max(SBT)*dbArray)
1225 ZipArray.append(max(SBT)*ufnArray)
1226 ZipArray.append(max(SBT)*ufsArray)
1227
1228 # Iterate over the different combinations of parameters
1229 # If varying one parameter, then use iter.product(items in list...)
1230 # If varying multiple parameters, use zip*max(SBT)*items in list...
1231
1232 for (namei,
1233      e1Seedi,
1234      e2Seedi,
1235      ptSeedi,
1236      gSeedi,
1237      v1Seedi,
1238      v2Seedi,
1239      rSeedi,
1240      massScaleFactori,
1241      massScaleTimeIncrementi,
1242      FNi,
1243      FSi,
1244      dbi,
1245      ufn,
1246      ufsi) in zip(*ZipArray):
1247
1248     tic() # Start time
1249     counter += 1
1250     print(counter, 'of ', max(*SBT))
1251
1252     # set the i'th value to the initial value (Updated in
1253     # optimization algorithm)
1254     RetinaYoungsModulus_i = RetinaYoungsModulus_0
1255     VitreousYoungsModulus_i = VitreousYoungsModulus_0
1256
1257     # Call the function
1258     # Runs jobs and saves file names
1259     funReturn = GenerateAbaqusModels()
1260     fileName = funReturn[0]
1261     maxSlopeSimulated = funReturn[1]

```

```

1262 maxForceSimulated = funReturn[2]
1263 SSmeanSimulated = funReturn[3]
1264 SSmedianSimulated = funReturn[4]
1265 y_new_exp_disp = funReturn[5]
1266 y_new_sim_disp = funReturn[6]
1267
1268 # Error calculation
1269 errorDict = {} # Dictionary
1270 if objErr == 'Difference':
1271     errorDict['slope'] = (maxSlopeMeasured - maxSlopeSimulated) if
1272     ↪ slopeFlag == True else []
1273     errorDict['maxForce'] = (maxForceMeasured - maxForceSimulated) if
1274     ↪ maxForceFlag == True else []
1275     errorDict['ssForce'] = (SS_Measured - SSmeanSimulated) if
1276     ↪ ssForceFlag == True else []
1277 elif objErr == 'Ratio':
1278     errorDict['slope'] = (1 - maxSlopeMeasured / maxSlopeSimulated) if
1279     ↪ slopeFlag == True else []
1280     errorDict['maxForce'] = (1 - maxForceMeasured / maxForceSimulated) if
1281     ↪ maxForceFlag == True else []
1282     errorDict['ssForce'] = (1 - SS_Measured / SSmeanSimulated) if
1283     ↪ ssForceFlag == True else []
1284 elif objErr == 'Relative uncertainty':
1285     errorDict['slope'] = ((maxSlopeMeasured -
1286     ↪ maxSlopeSimulated)/maxSlopeMeasured) if slopeFlag == True else []
1287     errorDict['maxForce'] = ((maxForceMeasured -
1288     ↪ maxForceSimulated)/maxForceMeasured) if maxForceFlag == True else
1289     ↪ []
1290     errorDict['ssForce'] = ((SS_Measured -
1291     ↪ SSmedianSimulated)/SS_Measured) if ssForceFlag == True
1292     ↪ else []
1293 else:
1294     print('Error in MaxForceError')
1295     sys.exit()
1296
1297 # Error array values
1298 errorList = list(errorDict.values()) # convert to list
1299 errorList = [x for x in errorList if x] # get rid of empty values
1300
1301 # Error of the simulation
1302 L2Normi = np.sqrt(np.dot(errorList, errorList))
1303
1304 # Calculate residual
1305 residual = y_new_exp_disp - y_new_sim_disp # residual
1306
1307 # Calculate L2Norm
1308 L2Norm = np.sqrt(np.dot(residual, residual))
1309
1310 # multipliers to be appended to the output file to show changes
1311 # in parameters
1312 aAM = [] # attributeArrayMultipliar
1313 aAM.append(time)
1314 aAM.append(e1Seedi[0])
1315 aAM.append(e2Seedi[0])
1316 aAM.append(ptSeedi[0])
1317 aAM.append(gSeedi[0])
1318 aAM.append(v1Seedi[0])
1319 aAM.append(v2Seedi[0])

```

```

1309     aAM.append(rSeedi[0])
1310     aAM.append(massScaleFactori[0])
1311     aAM.append(massScaleTimeIncrementi[0])
1312     aAM.append(RetinaYoungsModulus_i)
1313     aAM.append(VitreousYoungsModulus_i)
1314     aAM.append(BondStatus)
1315     aAM.append(FNi[0])
1316     aAM.append(FSi[0])
1317     aAM.append(PDFMStatus)
1318     aAM.append(dbi[0])
1319     aAM.append(ufni[0])
1320     aAM.append(ufsi[0])
1321     aAM.append(optimization)
1322     aAM.append(tieInterface)
1323     aAM.append(L2Normi)
1324     aAM.append(objErr)
1325     aAM.append(maxSlopeSimulated)
1326     aAM.append(maxForceSimulated)
1327     aAM.append(SSmedianSimulated)
1328
1329     simulationTime = toc() # Determine run time
1330     # appends the fileName & File Attributes
1331     fileNameList.append([fileName, aAM,
1332                          simulationTime])
1333     print('{} Error calculation: '.format(objErr), errorList)
1334     print('L2 norm objective calculation', L2Normi)
1335     print('L2 Norm residual', L2Norm)
1336     print('Done')
1337
1338     # Write data to txt files
1339     writeOutputData(fileNameList)

```

## 1.5.2 Input Parameter Selection

Determine input parameters to the Abaqus model. The following script not only determines maximum and steady-state peel force, but also integrates the force-displacement curve from the maximum force to the beginning of the steady-state peel as the failure energy input to the cohesive optimization routine.

</> **Script 5: Parameter selection script that determines the updated time at the maximum and steady-state peel force after linear extrapolation to the origin.** </>

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Jan 19 15:07:50 2021
4
5  @author: Kiffer2
6  """
7
8  import numpy as np
9  import pandas as pd
10 import os

```

```

11 import sys
12 import matplotlib.pyplot as plt
13 from matplotlib.pyplot import cm
14 import matplotlib.patheffects as pe
15 from matplotlib.patches import Polygon
16 plt.rcParams['figure.figsize'] = [16, 9]
17 from scipy import interpolate
18 import pdb
19
20
21 # # Define the location of the Abaqus Working Directory
22 # # specific folder path where this file is located
23 # pythonScriptPath = os.getcwd()
24 # abqWD, pythonFiles = os.path.split(pythonScriptPath) # split file path
25
26 # filePath = os.getcwd() # current working directory
27 # codePath, pythonFolder = os.path.split(filePath) # split file path
28 # HWPPath, codesFolder = os.path.split(codePath) # split file path
29
30 # expDataPath = 'experimentalData' # folder of data files
31 # dataPath = os.path.join(HWPPath, expDataPath) # Path to data files
32
33 def Least_Squares(x,y):
34     """
35     Calculate the slope and y-intercept using matrix math
36     x & y are the coordinates of points
37
38     parameters (X,Y) Data
39
40     Returns:
41     Curve fit data and parameters  $m \cdot x + b$ , R squared value
42     """
43     Z = np.ones((len(x),2))
44     Z[:,1] = x
45     # Calculate the matrix inverse for the constants of the regression
46     A = np.dot(np.linalg.inv(np.dot(Z.T,Z)),(np.dot(Z.T,y)))
47     linFit = x*A[1] + A[0]
48
49     # Stats
50     SS_tot = np.sum((y - np.mean(y))**2)
51     SS_res = np.sum((y - linFit)**2)
52     Rsqd = 1 - SS_res/SS_tot
53
54     return linFit, A, Rsqd
55
56 def myformat(x):
57     myexp = int(np.floor(np.log10(x)))
58     xout = x*10**(-myexp)
59     strout = '{:.4f}'.format(xout) + '\cdot 10^{-' + '{}'.format(myexp) + '}'
60     return strout
61
62
63 # In[previous data]
64
65 def ReadRAWDataTrace(dataPath, abqWD, timeBeforePeak):
66     """
67     Inputs: dataPath - file path to raw data
68     abqWD: abaqus working directory

```

```

69     timeBeforePeak: number of seconds prior to the peak where data will
70         be extrapolated to the origin for curve fitting
71     """
72
73     timeBeforePeak = timeBeforePeak*10 # Convert s --> cs (10 data points/sec)
74
75     # Eliminate the file extension
76     dataPathNoExt = dataPath.split('.txt')[0]
77
78     # Determine the specific file name
79     fileDir, dataCompare = os.path.split(dataPathNoExt)
80
81     """ Read in the csv file """
82     dfValsn = pd.read_csv(dataPath, sep="\t", nrows=29, header=None,
83         names=['Var', 'Attribute'])
84
85     """ File Attributes """
86     HID = dfValsn['Attribute'][0]
87     HAGE = dfValsn['Attribute'][1]
88     HG = dfValsn['Attribute'][2]
89     HLR = dfValsn['Attribute'][3]
90     HR = dfValsn['Attribute'][4]
91     HSSi = float(dfValsn['Attribute'][12])
92     HSSf = float(dfValsn['Attribute'][13])
93     HTMax = float(dfValsn['Attribute'][14])
94     HDispMax = float(dfValsn['Attribute'][15])
95     HFMax = float(dfValsn['Attribute'][16]) # (mN)
96     HFSS = float(dfValsn['Attribute'][17])
97     # slope from 20 seconds prior to max force value
98     HSlope20 = float(dfValsn['Attribute'][20]) # (mN/m)
99
100     dfn = pd.read_csv(dataPath, sep="\t", header=30)
101     dfn.columns = ['Time', 'Extension', 'Force']
102     dfn_time = dfn.Time
103     dfn_extension = dfn.Extension # mm
104     dfn_force = dfn.Force*1e3 # N ---> mN
105
106     # SS Array
107     ssTimeArray = np.array([HSSi, HSSf])
108     ssValArray = np.array([HFSS, HFSS])
109
110     # slope calculation for 20 seconds prior to the max peel force
111     # (Experimental Data)
112     maxIndex = dfn_time[dfn_time == HTMax].index.values[0]
113
114     # to location of max force
115     # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
116     t_n = dfn_extension[maxIndex - timeBeforePeak:maxIndex]
117     # to location of max force
118     # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
119     y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
120     # Perform least squares and return
121     curveFit_n, Params_n, R_Squared_n = Least_Squares(t_n,y)
122
123     # Shift extension data so that the linear region is extrapolated through
124     # the origin
125     shift = abs(Params_n[0]/Params_n[1])*0
126     dfn_extension = dfn_extension - shift

```

```

127
128 # Now that the data has been shifted, recalculate the linear regression
129 # using the reduced data set
130
131 # to location of max force
132 # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
133 t_n = dfn_extension[maxIndex - timeBeforePeak:maxIndex]
134 # to location of max force
135 # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
136 y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
137 # Perform least squares and return
138 curveFit_n, Params_n, R_Squared_n = Least_Squares(t_n,y)
139
140 # # Slope of the curve up to the max force !!!(from the simulated data)!!!
141 # adjustDisp = min(dn, key=lambda x:abs(x - dfn_extension[maxIndex]))
142 # index = RF[dn == adjustDisp].index.values[0]
143 # simulationCriteria = index # Time before peak force for curve fitting
144 # # Array from 0 to location of max force
145 # x = dn[index - simulationCriteria:index]
146 # # Array from 0 to location of max force
147 # y = RF[index - simulationCriteria:index]
148 # # Perform least squares
149 # curveFit, Params, R_Squared = Least_Squares(x,y)
150
151 # # Updated force at specific max disp with adjusted value (Simulated data)
152 # specificTime = maxForceTime
153 # actualDisp = min(dn, key=lambda x:abs(x - dfn_extension[maxIndex]))
154 # force_at_Displacement = RF[dn == actualDisp].values[0]
155
156 # # Simulated max force
157 # simMaxForce = RF.max() # maximum simulated force value
158 # simMaxDisp = dn[RF == simMaxForce] # displacement at the max force value
159
160 # Max peel force displacement at max and steady state
161 dfn_max_Displacement = dfn_extension[dfn_time == HTMax]
162 # Didn't seem to work here
163 # dfn_ss_Displacement = np.array([dfn_extension[dfn_time == HSSi],
164 #                                dfn_extension[dfn_time == HSSf]]).flatten()
165 dfn_ss_Displacement = [dfn_extension[dfn_time == HSSi].values[0],
166                        dfn_extension[dfn_time == HSSf].values[0]]
167
168 # In[Simulated Trace]
169
170 # dataDirectory = 'D:\Downloads\experimentalData'
171
172 # fileName = ('output_Field_S25CohesiveXLvitDiff_CT250S11' +
173 #             'SFONS7RE1e_04VE5e_02opt.txt')
174
175 # df = pd.read_csv(os.path.join(dataDirectory, fileName),
176 #                  sep="\t", header=0)
177
178 # Header = [] # Header information for the dataframe
179 # Header.append('Frame') # h1
180 # Header.append('Time') # h2
181 # Header.append('RF_y_dot') # h3
182 # Header.append('RFx') # h4
183 # Header.append('RFy') # h5
184 # Header.append('RFz') # h6

```

```

185 # Header.append('Nodal_Force') # h7
186 # Header.append('Tab_Displacement') # h8
187 # Header.append('Bond_Displacement') # h9
188 # Header.append('Stress') # h10
189 # Header.append('AVG_CS_MAXSCRT') # h11
190 # Header.append('AVG_CSDMG') # h12
191 # df.columns = Header
192
193 # tt = df.Time
194 # RF = df.RF_y_dot*1000 # N to mN
195 # dn = df.Tab_Displacement*1000 # m
196
197 # In[Plots]
198
199 """ Plots """
200 # Plot the data trace to compare the simulated results with the force
201 # displacement curves
202 fig, ax = plt.subplots()
203 ax.plot(dfn_extension, dfn_force, '-', color='r', linewidth=1,
204         markersize=2, label = '{} , Age: {}'.format(HID, HAGE),
205         alpha = 0.5)
206
207 if str(HFMax) == 'nan' and str(HSSi) == 'nan':
208     print('No max or steady state')
209     pass
210
211 if str(HFMax) != 'nan':
212     ax.plot(dfn_max_Dis, HFMax, '.', color='k', linewidth=1,
213             markersize=20,
214             label = 'Max Peel - {:.4f} (mN)'.format(HFMax),
215             path_effects=[pe.Stroke(linewidth=4, foreground='k'),
216                           pe.Normal()])
217     ax.plot(t_n, curveFit_n, '-', color='tab:blue', linewidth=2,
218             label=r'Curve fit Max - {}'.format(int(timeBeforePeak/10)) +
219             ' (s) y = {:.4f}x '.format(Params_n[1]) +
220             '+ {:.4f} (mN), '.format(Params_n[0]) +
221             '$r^2$ = {:.4f}'.format(R_Squared_n),
222             alpha = 1)
223
224 if str(HSSi) != 'nan':
225     ax.plot(dfn_ss_Dis, ssValArray, '-', color='c', linewidth=3,
226             markersize=2,
227             label = 'Steady State - {:.4f} (mN)'.format(HFSS),
228             path_effects=[pe.Stroke(linewidth=5,
229                                     foreground='k'),
230                           pe.Normal()])
231
232 # Make the shaded region for the entire integral
233 a = dfn_max_Dis.values[0] # dfn_ss_Dis[0]
234 b = dfn_ss_Dis[0] # dfn_ss_Dis[1]
235
236 # Make the shaded region include the square below
237 adjust = 0 # 0 or 1 to get rid of the small square
238
239 # Filter the data in between the bounds
240 dfn_ext_adjust = dfn_extension[(dfn_extension >= a) & (dfn_extension < b)]
241 dnf_force_adjust = dfn_force[(dfn_extension >= a) & (dfn_extension < b)]
242

```



```

243     verts = [(a, HFSS*adjust),
244               *zip(dfn_ext_adjust, dnf_force_adjust),
245               (b, HFSS*adjust)]
246     poly = Polygon(verts, facecolor='0.8', edgecolor='0.5')
247     ax.add_patch(poly)
248
249     # Integral area
250     Integral = np.trapz(dnf_force_adjust - HFSS*adjust, dfn_ext_adjust)
251
252     # Centroid for plotting
253     CentroidX = 1/Integral*(np.trapz(dfn_ext_adjust*(dnf_force_adjust -
254                                         HFSS*adjust),
255                                         dfn_ext_adjust))
256     CentroidY = 1/Integral*(np.trapz((dnf_force_adjust**2 -
257                                         (HFSS*adjust)**2*adjust)/2,
258                                         dfn_ext_adjust))
259
260     # ax.text(b, (HFMax + HFSS)/2, r'\int_a^b f(x)\mathrm{d}x=' +
261     # myformat(Integral*1e-6) + '$ (J)', horizontalalignment='center',
262     # fontsize=20)
263     # ax.plot([0.5*max(dfn_extension), CentroidX], [0.5*max(dfn_force),
264     # CentroidY])
265
266     prop = dict(arrowstyle="->",head_width=0.4, head_length=0.8", shrinkA=0,
267                 shrinkB=0)
268     # ax.arrow(0.5*max(dfn_extension), 0.5*max(dfn_force),
269     # CentroidX - 0.5*max(dfn_extension),
270     # CentroidY - 0.5*max(dfn_force),
271     # head_width=0.1, head_length=0.1)
272     ax.annotate("", xy=(CentroidX, CentroidY), xytext=(0.5*max(dfn_extension),
273                                                         0.5*max(dfn_force)),
274                arrowprops=prop)
275
276     ax.text(0.5*max(dfn_extension), 0.52*max(dfn_force),
277            r'\int_a^b f(x)\mathrm{d}x=' + myformat(Integral*1e-6) + '$ (J)',
278            horizontalalignment='center', fontsize=20)
279
280     ax.spines['right'].set_visible(False)
281     ax.spines['top'].set_visible(False)
282     ax.xaxis.set_ticks_position('bottom')
283
284     ax.set_xticks((a, b))
285     ax.set_xticklabels(('${}\$'.format(a), '{}\$'.format(b)))
286     ax.set_yticks((HFSS, HFMax))
287     ax.set_yticklabels(('${:.5}\$'.format(HFSS), '{}:.5}\$'.format(HFMax)))
288
289     ##### Plot Data #####
290     plt.axhline(0,color='black') # x = 0
291     plt.axvline(0,color='black') # y = 0
292     plt.ylabel('Force (mN)',fontsize=18)
293     plt.xlabel('Displacement (mm)',fontsize=18)
294     plt.title('Vitreous',fontsize=20)
295     plt.grid()
296     plt.legend(loc = 'best',fontsize = 'medium')
297     plt.savefig(os.path.join(abqWD, 'GcSelection.pdf'), dpi=300,
298               bbox_inches='tight')
299     # plt.show()
300     plt.close()

```

```

301
302
303
304     # """ Derivative of the data trace """
305     # fig, ax = plt.subplots()
306
307     # deriv = np.gradient(dfn_force, dfn_extension)
308
309     # ax.plot(dfn_extension, deriv)
310     # ax.set_ylim(-100, 100) # maxRFList
311     # plt.show()
312
313     # In[Time plot]
314
315     # slope calculation for n seconds prior to the max peel force
316     # (Experimental Data)
317     maxIndex = dfn_time[dfn_time == HTMax].index.values[0]
318
319     # to location of max force
320     # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
321     t_n = dfn_time[maxIndex - timeBeforePeak:maxIndex]
322     y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
323     # Perform least squares and return
324     curveFit_n, Params_n, R_Squared_n = Least_Squares(t_n, y)
325
326     # Shift extension data so that the linear region is extrapolated
327     # through the origin
328     shift_time = abs(Params_n[0]/Params_n[1])*1
329     if Params_n[0] > 0:
330         # shift time data for visual purposes
331         dfn_time_shift = dfn_time + shift_time
332         dfn_ss_time_shift = ssTimeArray + shift_time
333         HTMax_shift = HTMax + shift_time
334     else:
335         # shift time data for visual purposes
336         dfn_time_shift = dfn_time - shift_time
337         dfn_ss_time_shift = ssTimeArray - shift_time
338         HTMax_shift = HTMax - shift_time
339
340
341     # Curve fit the shifted displacement
342     maxIndex = dfn_time[dfn_time == HTMax].index.values[0]
343
344     # to location of max force
345     # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
346     t_n = dfn_time_shift[maxIndex - timeBeforePeak:maxIndex]
347     y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
348     # Perform least squares and return
349     curveFit_n, Params_n, R_Squared_n = Least_Squares(t_n, y)
350
351     # to location of max force
352     # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
353     x_n = dfn_extension[maxIndex - timeBeforePeak:maxIndex]
354     # Perform least squares
355     curveFit_n_disp, Params_n_disp, R_Squared_n_disp = Least_Squares(x_n, y)
356
357     # Shift extension data so that the linear region is extrapolated through
358     # the origin

```

```

359 shift_disp = abs(Params_n_disp[0]/Params_n_disp[1])*1
360 if Params_n[0] > 0:
361     dfn_extension_shift = dfn_extension + shift_disp
362     dfn_ss_Disp_shift = dfn_ss_Disp + shift_disp
363 else:
364     dfn_extension_shift = dfn_extension - shift_disp
365     dfn_ss_Disp_shift = dfn_ss_Disp - shift_disp
366
367 # to location of max force
368 # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
369 x_n = dfn_extension_shift[maxIndex - timeBeforePeak:maxIndex]
370 # Perform least squares
371 curveFit_n_disp, Params_n_disp, R_Squared_n_disp = Least_Squares(x_n, y)
372
373
374 Fmax_t_shift = dfn_time_shift[maxIndex]
375 fit_t = np.linspace(0, Fmax_t_shift, 200) # Selected value
376
377 # true max
378 # fit_t = np.linspace(0, dfn_time_shift[np.argmax(dfn_force)], 200)
379 Fmax_x_shift = dfn_extension_shift[maxIndex]
380
381 # true max
382 # fit_x = np.linspace(0, dfn_extension_shift[np.argmax(dfn_force)], 200)
383 fit_x = np.linspace(0, Fmax_x_shift, 200) # Selected value
384
385 def fit(params, x):
386     b, m = params
387     return m*x + b
388
389 fit_vals_y_time = fit(Params_n, fit_t)
390 fit_vals_y_force = fit(Params_n_disp, fit_x)
391
392 ''' Reaction force vs. time shifted '''
393 fig, ax = plt.subplots()
394 ax.plot(dfn_time_shift, dfn_force,
395         label=r'Data - {}'.format(dataCompare.split('.')[0]))
396 ax.plot(fit_t, fit_vals_y_time, '--', label=r'Assumed linear region')
397 ax.plot(Fmax_t_shift, dfn_force[maxIndex], 'o', markersize=10,
398         label=r'Time at peak = {:.4f} (s)'.format(max(fit_t)))
399
400 ax.plot(dfn_ss_time_shift, ssValArray, '-', color='c', linewidth=3,
401         markersize=2, label = 'Steady State - {:.4f} (mN)'.format(HFSS),
402         path_effects=[pe.Stroke(linewidth=5, foreground='k'), pe.Normal()])
403
404 ax.plot([], [], 'w',
405         label='Start SS time = {:.4f} (s)'.format(min(dfn_ss_time_shift)))
406 ax.plot([], [], 'w',
407         label='End SS time = {:.4f} (s)'.format(max(dfn_ss_time_shift)))
408
409 plt.axhline(0,color='black')
410 plt.axvline(0,color='black')
411
412 plt.ylabel('Force (mN)',fontsize=18)
413 plt.xlabel('Time from extrapolated zero (s)',fontsize=18)
414 plt.legend(loc='best')
415 # plt.xlim([0, max(dfn_time_shift)])
416 plt.savefig(os.path.join(abqWD, 'SimulationTime.pdf'), dpi=300,

```

```

417         bbox_inches='tight')
418     # plt.show()
419     plt.close()
420
421     ''' Reaction force vs. displacement shifted '''
422     fig, ax = plt.subplots()
423     ax.plot(dfn_extension_shift, dfn_force,
424             label=r'Data - {}'.format(dataCompare.split('.')[0]))
425     ax.plot(fit_x, fit_vals_y_force, '--', label=r'Assumed linear region')
426     ax.plot(Fmax_x_shift, dfn_force[maxIndex], 'o', markersize=10,
427             label=r'Time at peak = {:.4f} (s)'.format(max(fit_t)))
428
429     ax.plot(dfn_ss_Displacement_shift, ssValArray, '-', color='c', linewidth=3,
430             markersize=2, label = 'Steady State - {:.4f} (mN)'.format(HFSS),
431             path_effects=[pe.Stroke(linewidth=5, foreground='k'), pe.Normal()])
432
433     ax.plot([], [], 'w',
434             label='Start SS time = {:.4f} (s)'.format(min(dfn_ss_time_shift)))
435     ax.plot([], [], 'w',
436             label='End SS time = {:.4f} (s)'.format(max(dfn_ss_time_shift)))
437
438     plt.axhline(0, color='black')
439     plt.axvline(0, color='black')
440
441     plt.ylabel('Force (mN)', fontsize=18)
442     plt.xlabel('Displacement (mm)', fontsize=18)
443     plt.legend(loc='best')
444     # plt.xlim([0, max(dfn_time_shift)])
445     plt.savefig(os.path.join(abqWD, 'SimulationDisp.pdf'), dpi=300,
446               bbox_inches='tight')
447     # plt.show()
448     plt.close()
449
450     # In[Interpolated Experimental Data]
451
452     # create array from 0 max peel force (linear equation fit from above)
453     # populate a pandas dataframe
454     # merge the data frame with the data above from the peak force to the end
455     # use the interp1d fcn to interpolate between data
456     # pass the simulated data into the interpolation
457
458     # Time greater than the shift intersection point
459     t_exp = dfn_time_shift[dfn_time_shift >= 0]
460     x_exp = dfn_extension_shift[dfn_time_shift >= 0]
461     y_exp = dfn_force[dfn_time_shift >= 0]
462
463     # data frame with original data
464     dfdata = pd.DataFrame(np.array([t_exp, x_exp, y_exp]).T,
465                           columns=['t', 'x', 'y'])
466
467     # Select time beyond the max time to the end of the data
468     t_geq_max = dfn_time_shift[maxIndex:]
469     x_geq_max = dfn_extension_shift[maxIndex:]
470     y_geq_max = dfn_force[maxIndex:]
471
472     # dataframe of data points from the max value to the end
473     dfgmax = pd.DataFrame(np.array([t_geq_max, x_geq_max, y_geq_max]).T,
474                           columns=['t', 'x', 'y'])

```

```

475
476 # data frame of points from zero to the max value
477 linArray = np.array([fit_t, fit_x, fit_vals_y_time])
478 dfLin = pd.DataFrame(linArray.T, columns=['t', 'x', 'y'])
479
480 # create the new data frame of linear points up to the peak and all points
481 # beyond
482 dfNew = dfLin.append(dfgmax, ignore_index=True)
483
484 # # Interpolate the experimental data
485 # n_data_pts = 100
486 # Time at the peak (shifted)
487 # start_point_time = tt[RF.argmax()]# - shift
488 # Disp at the peak (shifted)
489 # start_point_disp = dn[RF.argmax()]# - shift_disp
490 # f_exp_time = interpolate.interp1d(dfNew['t'], dfNew['y'])
491 # f_exp_disp = interpolate.interp1d(dfNew['x'], dfNew['y'])
492 # t_new_exp = np.linspace(start_point_time, tt[tt.argmax()],
493 #                          n_data_pts) # (s)
494 # x_new_exp = np.linspace(start_point_disp, dn[tt.argmax()],
495 #                          n_data_pts) # (mm)
496 # y_new_exp_time = f_exp_time(t_new_exp) # Interpolate `interp1d`
497 # y_new_exp_disp = f_exp_disp(x_new_exp) # Interpolate `interp1d`
498
499 # In[Interpolated Simulated Trace]
500
501 # # Interpolate the simulated data
502 # f_sim_time = interpolate.interp1d(tt, RF)
503 # f_sim_disp = interpolate.interp1d(dn, RF)
504 # t_new_sim = np.linspace(start_point_time, tt[tt.argmax()],
505 #                          n_data_pts) # (s)
506 # x_new_sim = np.linspace(start_point_disp, dn[tt.argmax()],
507 #                          n_data_pts) # (mm)
508 # y_new_sim_time = f_sim_time(t_new_sim) # Interpolate `interp1d`
509 # y_new_sim_disp = f_sim_disp(x_new_sim) # Interpolate `interp1d`
510
511 # In[Plots]
512 # ''' Time curve '''
513 # fit, ax = plt.subplots()
514 # ax.plot()
515 # ax.plot(dfdata['t'], dfdata['y'], label='Original Shifted Data',
516 #         alpha = 0.5)
517 # ax.plot(dfNew['t'], dfNew['y'], label='Merged Data',
518 #         alpha = 0.5)
519 # ax.plot(t_new_exp, y_new_exp_time, '--',
520 #         label='Interp Experimental Data')
521 # ax.plot(tt, RF, label='Simulated Data')
522 # ax.plot(t_new_sim, y_new_sim_time, ':', label='Interp Simulated Data')
523 # ax.set_xlim([0, 300])
524 # ax.set_xlabel('Time (s)', fontsize=14)
525 # ax.set_ylabel('Force (N)', fontsize=14)
526 # ax.legend(loc='best', fontsize=14)
527 # ax.grid('on')
528 # plt.savefig(os.path.join(abqWD, 'interp1d_Time.pdf'), dpi=300,
529 #             bbox_inches='tight')
530 # plt.show()
531
532 # ''' Displacement curve '''

```

```

533 # fit, ax = plt.subplots()
534 # ax.plot()
535 # ax.plot(dfdata['x'], dfdata['y'], label='Original Shifted Data',
536 #         alpha = 0.5)
537 # ax.plot(dfNew['x'], dfNew['y'], label='Merged Data',
538 #         alpha = 0.5)
539 # ax.plot(x_new_exp, y_new_exp_disp, '--',
540 #         label='Interp Experimental Data')
541 # ax.plot(dn, RF, label='Simulated Data')
542 # ax.plot(x_new_sim, y_new_sim_disp, ':', label='Interp Simulated Data')
543 # ax.set_xlim([0, max(dn)])
544 # ax.set_xlabel('Displacement (mm)', fontsize=14)
545 # ax.set_ylabel('Force (N)', fontsize=14)
546 # ax.legend(loc='best', fontsize=14)
547 # ax.grid('on')
548 # plt.savefig(os.path.join(abqWD, 'interp1d_Displ.pdf'), dpi=300,
549 #             bbox_inches='tight')
550 # plt.show()
551
552 # ''' Displacement curve only showing interpolated data '''
553 # abs residual calculation
554 # residual = abs(y_new_exp_disp - y_new_sim_disp)
555 # L2Norm = np.dot(residual, residual)
556
557 # fit, ax = plt.subplots()
558 # ax.plot()
559 # ax.plot(x_new_exp, y_new_exp_disp, '-', label='Interp Experimental Data')
560 # ax.plot(x_new_sim, y_new_sim_disp, '-', label='Interp Simulated Data')
561 # ax.plot(x_new_sim, residual, ':',
562 #         label=r'Residual =  $\| \exp - \text{sim} \|$ ', alpha = 0.8)
563 # ax.plot([], [], color='white',
564 #         label=r' $L^2$  norm = {:.4f}'.format(L2Norm))
565 # ax.axhline(color='k', linewidth=0.25)
566 # ax.set_xlim([0, max(x_new_exp)])
567 # ax.set_xlabel('Displacement (mm)', fontsize=14)
568 # ax.set_ylabel('Force (N)', fontsize=14)
569 # ax.legend(loc='best', fontsize=14)
570 # ax.grid('on')
571 # plt.savefig(os.path.join(abqWD, 'interp1d_Displ_clean.pdf'), dpi=300,
572 #             bbox_inches='tight')
573 # plt.show()
574
575 print('Output files have been printed to determine the appropriate ' +
576       'parameters for the simulation')
577
578 returnArray = [max(fit_t), max(dfn_ss_time_shift), HTMax_shift,
579               Integral*1e-6]
580 return returnArray
581
582 if __name__ == '__main__':
583     # Run the function
584
585     # fileName = sys.argv[-2]
586     # savePath = sys.argv[-1]
587
588     ReadRAWDDataTrace(fileName, abqWD, timeBeforePeak)

```

### 1.5.3 Abaqus Python Script

</> Script 6: Abaqus python script used to create the input file (.inp) and execute the simulation. </>

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Oct 18 22:12:05 2020
4
5 @author: Kiffer2
6 """
7
8 """ abaqus cae -noGUI abaqusMacros.py """
9
10 # -*- coding: mbcs -*-
11 # Do not delete the following import lines
12 from abaqus import *
13 from abaqusConstants import *
14 import __main__
15
16 import section
17 import regionToolset
18 import displayGroupMdbToolset as dgm
19 import part
20 import material
21 import assembly
22 import step
23 import interaction
24 import load
25 import mesh
26 import optimization
27 import job
28 import sketch
29 import visualization
30 import xyPlot
31 import displayGroupOdbToolset as dgo
32 import connectorBehavior
33 import numpy as np
34 import os
35 import sys
36
37 # location of the folder
38 # specific folder path where this file is located # os.getcwd()
39 pythonScriptPath = os.path.abspath("file")
40 abqWD, pythonFiles = os.path.split(pythonScriptPath) # split file path
41
42 # StepFile = 'Adult Human Eye holder Assembly.STEP'
43 # StepFile = 'Adult Human Eye holder Assembly 2 Step.STEP'
44 # Constrained
45 # StepFile = 'Adult Human Eye holder Assembly Constrained Bottom.STEP'
46
47 # Trimmed to prevent element distortion on low elastic modulus curve fits
48 StepFile = ('Adult Human Eye holder Assembly Constrained Bottom Trimmed ' +
49             'Retina.STEP')
50
51 SolidWorksDir = 'SolidWorksStepFiles' # Folder name
52
```

```

53 # Combine folder directory
54 SolidWorksStepFile = os.path.join(SolidWorksDir, StepFile)
55
56 # In[Non-symmetric model]
57
58 def ImportStepEyeConstrained():
59     """ Use with the constrained bottom STEP file """
60     step = mdb.openStep(os.path.join(abqWD, SolidWorksStepFile),
61                          scaleFromFile=OFF)
62
63     abqModel.PartFromGeometryFile(name='V', geometryFile=step, bodyNum=1,
64                                   combine=False, dimensionality=THREE_D,
65                                   type=DEFORMABLE_BODY)
66     abqModel.PartFromGeometryFile(name='E', geometryFile=step, bodyNum=2,
67                                   combine=False, dimensionality=THREE_D,
68                                   type=DISCRETE_RIGID_SURFACE)
69     abqModel.PartFromGeometryFile(name='R', geometryFile=step, bodyNum=3,
70                                   combine=False, dimensionality=THREE_D,
71                                   type=DEFORMABLE_BODY)
72     abqModel.PartFromGeometryFile(name='T', geometryFile=step, bodyNum=4,
73                                   combine=False, dimensionality=THREE_D,
74                                   type=DISCRETE_RIGID_SURFACE)
75     abqModel.PartFromGeometryFile(name='G', geometryFile=step, bodyNum=5,
76                                   combine=False, dimensionality=THREE_D,
77                                   type=DISCRETE_RIGID_SURFACE)
78
79
80 def Retina_Mat_Prop(RetinaProp):
81     retina_E = RetinaProp # Passed in young's modulus
82     Retina_Description = """
83     Actually used the value from Chen 2014
84     E = 11.12 KPa
85
86     -----
87     Density (kg/m^3)
88     1100 -----> Esposito_2013
89
90     """
91     abqModel.Material(name='Retina', description=Retina_Description)
92     abqModel.materials['Retina'].Density(table=((1100.0, ), ))
93     abqModel.materials['Retina'].Elastic(table=((retina_E, 0.49), ))
94
95     # Assign the section to the part
96     abqModel.HomogeneousSolidSection(name='Retina_Section', material='Retina',
97                                      thickness=None)
98
99 def Vitreous_Mat_Prop(vitreousProp):
100     vitreous_E = vitreousProp # Passed in young's modulus
101     Vitreous_Description = """
102     -----
103     Density (kg/m^3)
104     950 -----> Esposito_2013
105
106     -----
107
108     # Tram 2018 Viscoelasticity data
109     # 4 Term Prony (Tram Data # 5 HU2018-0074 OD 1 Pa)
110     (0.1486397420159951, 0.0, 331.4796231072498),

```



```

111 (0.12469207412616717, 0.0, 3.388868494747128),
112 (0.29059507092540404, 0.0, 15.59692349525066),
113 (0.1591569334281, 0.0, 69.85134248442381)
114 """
115     abqModel.Material(name='Vitreous', description=Vitreous_Description)
116     abqModel.materials['Vitreous'].Density(table=((950.0, ), ))
117     ''' Using Lin2020 Paper to relate SLSM curve fit parameters to physical
118     values. Prony 4 Term (Long term) initial guess 172.77874855377468
119     optimization of E'''
120     abqModel.materials['Vitreous'].Elastic(moduli=LONG_TERM,
121                                           table=((vitreous_E, 0.49), ))
122     # Prony 4 Term calculated from normalized data
123     abqModel.materials['Vitreous'].Viscoelastic(
124         domain=TIME, time=PRONY, table=(
125             # Tram Data # 5
126             (0.1486397420159951, 0.0, 331.4796231072498),
127             (0.12469207412616717, 0.0, 3.388868494747128),
128             (0.29059507092540404, 0.0, 15.59692349525066),
129             (0.1591569334281, 0.0, 69.85134248442381)))
130
131     # Assign the section to the part
132     abqModel.HomogeneousSolidSection(name='Vitreous_Section',
133                                     material='Vitreous', thickness=None)
134
135 def E_Features():
136     ''' Eye holder features '''
137     p = abqModel.parts['E']
138
139     # Remove shell
140     c = p.cells
141     p.RemoveCells(cellList = c[0:1])
142
143     # Reference point
144     p.ReferencePoint(point=(0.0, 0.0, 0.0))
145
146     # Add E-set to the reference point
147     r = p.referencePoints
148     refPoints=(r[3], )
149     p.Set(referencePoints=refPoints, name='E_RP_Set')
150
151     # Edge seed sets
152     e = p.edges
153     edges = e.getSequenceFromMask(mask=('[#400f000 #1402 ]', ), )
154     p.Set(edges=edges, name='E_Edge_Seed_Set')
155
156     edges = e.getSequenceFromMask(mask=('[#f1ff0fff #2838 ]', ), )
157     p.Set(edges=edges, name='E_Outside_Edge_Seed_Set')
158
159     # Surfaces
160     s = p.faces
161     side1Faces = s.getSequenceFromMask(mask=('[#1ffff ]', ), )
162     p.Surface(side1Faces=side1Faces, name='E_Surf')
163
164
165 def G_Features():
166     ''' Glue features '''
167     p = abqModel.parts['G']
168     c = p.cells

```

```

169
170     # Remove cells for rigid body
171     p.RemoveCells(cellList = c[0:1])
172
173     # Reference point
174     p.ReferencePoint(point=(9.799E-03, 5.657E-03, 2.54E-03))
175
176     # Define the reference point for the rigid body
177     r = p.referencePoints
178     refPoints=(r[3], )
179     p.Set(referencePoints=refPoints, name='G_RP_Set')
180
181     # # Create sets
182     f = p.faces
183     faces = f.getSequenceFromMask(mask=('#3f ', ), )
184     p.Set(faces=faces, name='G_Set')
185     faces = f.getSequenceFromMask(mask=('#20 ', ), )
186     p.Set(faces=faces, name='G_T_Set')
187     faces = f.getSequenceFromMask(mask=('#1 ', ), )
188     p.Set(faces=faces, name='G_R_Set')
189
190     # Create surfaces
191     s = p.faces
192     side1Faces = s.getSequenceFromMask(mask=('#3f ', ), )
193     p.Surface(side1Faces=side1Faces, name='G_Surf')
194     side1Faces = s.getSequenceFromMask(mask=('#20 ', ), )
195     p.Surface(side1Faces=side1Faces, name='G_T_Surf')
196     side1Faces = s.getSequenceFromMask(mask=('#1 ', ), )
197     p.Surface(side1Faces=side1Faces, name='G_R_Surf')
198
199
200 def T_Features():
201     ''' Plastic Tab features '''
202     p = abqModel.parts['T']
203     c = p.cells
204
205     # Remove cells for rigid body
206     p.RemoveCells(cellList = c[0:1])
207
208     # Reference point
209     p.ReferencePoint(point=(16.241E-03, 9.74E-03, 13.E-06))
210
211     # Define the reference point for the rigid body
212     r = p.referencePoints
213     refPoints=(r[3], )
214     p.Set(referencePoints=refPoints, name='T_RP_Set')
215
216     # Create sets
217     f = p.faces
218     faces = f.getSequenceFromMask(mask=('#ff ', ), )
219     p.Set(faces=faces, name='T_Set')
220     f = p.faces
221     faces = f.getSequenceFromMask(mask=('#2 ', ), )
222     p.Set(faces=faces, name='T_G_Set')
223
224     # Create surfaces
225     s = p.faces
226     side1Faces = s.getSequenceFromMask(mask=('#ff ', ), )

```

```

227 p.Surface(side1Faces=side1Faces, name='T_Surf')
228 side1Faces = s.getSequenceFromMask(mask=('[#2 ]', ), )
229 p.Surface(side1Faces=side1Faces, name='T_G_Surf')
230
231
232 def R_Features():
233     ''' Retina features '''
234     p = abqModel.parts['R']
235     c = p.cells
236     cells = c.getSequenceFromMask(mask=('[#1 ]', ), )
237     p.Set(cells=cells, name='R_Set')
238
239     f = p.faces
240     faces = f.getSequenceFromMask(mask=('[#3 ]', ), )
241     p.Set(faces=faces, name='R_G_Set')
242
243     faces = f.getSequenceFromMask(mask=('[#4 ]', ), )
244     p.Set(faces=faces, name='R_V_Set')
245
246     s = p.faces
247     side1Faces = s.getSequenceFromMask(mask=('[#ff ]', ), )
248     p.Surface(side1Faces=side1Faces, name='R_Surf')
249
250     side1Faces = s.getSequenceFromMask(mask=('[#3 ]', ), )
251     p.Surface(side1Faces=side1Faces, name='R_G_Surf')
252
253     side1Faces = s.getSequenceFromMask(mask=('[#4 ]', ), )
254     p.Surface(side1Faces=side1Faces, name='R_V_Surf_BOND')
255
256     # Assign section
257     region = p.sets['R_Set']
258     p.SectionAssignment(region=region, sectionName='Retina_Section',
259                        offset=0.0, offsetType=MIDDLE_SURFACE, offsetField='',
260                        thicknessAssignment=FROM_SECTION)
261
262
263
264 def PartitionRetinaOnVitreous():
265     ''' Vitreous features additional partitions for creating the surface for
266     bonding'''
267     p = abqModel.parts['V']
268
269     # Partition V along the width of the retina
270     p.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=-0.00254)
271     abqModel.parts['V'].features.changeKey(fromName='Datum plane-1',
272                                           toName='Retina_Width_Neg_Z')
273
274     p.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=0.00254)
275     abqModel.parts['V'].features.changeKey(fromName='Datum plane-1',
276                                           toName='Retina_Width_Pos_Z')
277
278     # Create a datum plnd along the z axis plane
279     p.DatumAxisByPrincipalAxis(principalAxis=ZAXIS)
280     p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset=0.0)
281
282     # Create the rotated datum planes
283     d = p.datums
284     p.DatumPlaneByRotation(plane=d[5], axis=d[4], angle=18.75)

```

```

285 p.DatumPlaneByRotation(plane=d[5], axis=d[4], angle=-18.75)
286
287 ''' Partition the surface of the retina on the vitreous '''
288 p = abqModel.parts['V']
289 c, d = p.cells, p.datums
290 pickedCells = c.getSequenceFromMask(mask=('#440 '), )
291 p.PartitionCellByDatumPlane(datumPlane=d[3], cells=pickedCells)
292 pickedCells = c.getSequenceFromMask(mask=('#408 '), )
293 p.PartitionCellByDatumPlane(datumPlane=d[2], cells=pickedCells)
294 pickedCells = c.getSequenceFromMask(mask=('#22 '), )
295 p.PartitionCellByDatumPlane(datumPlane=d[6], cells=pickedCells)
296 pickedCells = c.getSequenceFromMask(mask=('#140 '), )
297 p.PartitionCellByDatumPlane(datumPlane=d[7], cells=pickedCells)
298
299
300 def Vitreous_Features():
301     ''' Assign specific features to the vitreous '''
302     p = abqModel.parts['V']
303     c, f, s = p.cells, p.faces, p.faces
304
305     # Sets
306     cells = c.getSequenceFromMask(mask=('#ffffff '), )
307     p.Set(cells=cells, name='V_Set')
308     faces = f.getSequenceFromMask(mask=('#5090 '), )
309     p.Set(faces=faces, name='V_R_Set')
310
311     # Surfaces
312     side1Faces = s.getSequenceFromMask(mask=('#1805090 #3 #ff0 '), )
313     p.Surface(side1Faces=side1Faces, name='V_Surf')
314     side1Faces = s.getSequenceFromMask(mask=('#5090 '), )
315     p.Surface(side1Faces=side1Faces, name='V_R_Surf_BOND')
316
317     # Assign the section to the part
318     region = p.sets['V_Set']
319     p.SectionAssignment(region=region,
320                        sectionName='Vitreous_Section',
321                        offset=0.0,
322                        offsetType=MIDDLE_SURFACE,
323                        offsetField='',
324                        thicknessAssignment=FROM_SECTION)
325
326
327 def V_Partition_XYZ_Axis():
328     ''' Partition the sphere along the x, y, z axis '''
329     p = abqModel.parts['V']
330     c, v, e, d = p.cells, p.vertices, p.edges, p.datums
331     pickedCells = c.getSequenceFromMask(mask=('#1 '), )
332     p.PartitionCellByPlaneThreePoints(point1=v[1],
333                                       point2=v[0],
334                                       point3=v[3],
335                                       cells=pickedCells)
336
337     pickedCells = c.getSequenceFromMask(mask=('#3 '), )
338     p.PartitionCellByPlaneThreePoints(point1=v[0],
339                                       point2=v[4],
340                                       point3=v[2],
341                                       cells=pickedCells)
342

```

```

343 pickedCells = c.getSequenceFromMask(mask=('[#f ]', ), )
344 p.PartitionCellByPlaneThreePoints(point1=v[5],
345                                   point2=v[2],
346                                   point3=v[4],
347                                   cells=pickedCells)
348
349
350 def V_Internal_Sphere():
351     sphereRadius = 0.008 # radius of the internal sphere for meshing
352
353     s1 = abqModel.ConstrainedSketch(name='__profile__', sheetSize=0.1)
354     g, v, d, c1 = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
355     s1.sketchOptions.setValues(decimalPlaces=3)
356     s1.setPrimaryObject(option=STANDALONE)
357     s1.ConstructionLine(point1=(0.0, -0.05), point2=(0.0, 0.05))
358     s1.FixedConstraint(entity=g[2])
359     s1.ArcByCenterEnds(center=(0.0, 0.0),
360                       point1=(0.0, sphereRadius),
361                       point2=(0.0, -sphereRadius),
362                       direction=CLOCKWISE)
363     s1.Line(point1=(0.0, sphereRadius),
364            point2=(0.0, -sphereRadius))
365     s1.VerticalConstraint(entity=g[4], addUndoState=False)
366     s1.PerpendicularConstraint(entity1=g[3], entity2=g[4], addUndoState=False)
367     p = abqModel.Part(name='V_internal',
368                      dimensionality=THREE_D,
369                      type=DEFORMABLE_BODY)
370     p = abqModel.parts['V_internal']
371     p.BaseSolidRevolve(sketch=s1, angle=360.0, flipRevolveDirection=OFF)
372     s1.unsetPrimaryObject()
373     p = abqModel.parts['V_internal']
374     del abqModel.sketches['__profile__']
375
376
377 def mergeV():
378     ''' Merge the internal sphere with the vitreous '''
379     a = abqModel.rootAssembly
380     a.InstanceFromBooleanMerge(name='V_Merge',
381                               instances=(a.instances['V-1'],
382                                         a.instances['V_internal-1']), ),
383     keepIntersections=ON,
384     originalInstances=DELETE,
385     domain=GEOMETRY)
386
387     # Clean up file names after merge
388     del abqModel.parts['V']
389     del abqModel.parts['V_internal']
390
391     abqModel.parts.changeKey(fromName='V_Merge', toName='V')
392     a = abqModel.rootAssembly
393     a.regenerate()
394     abqModel.rootAssembly.features.changeKey(fromName='V_Merge-1',
395                                              toName='V-1')
396
397     a.regenerate()
398
399
400 def AssembleV_for_Merging():

```

```

401     a1 = abqModel.rootAssembly
402     a1.DatumCsysByDefault(CARTESIAN)
403     p = abqModel.parts['V']
404     a1.Instance(name='V-1', part=p, dependent=ON)
405     p = abqModel.parts['V_internal']
406     a1.Instance(name='V_internal-1', part=p, dependent=ON)
407
408
409 def E_Mesh(InsideSeed, OutsideSeed):
410     p = abqModel.parts['E']
411     e = p.edges
412     pickedEdges = e.getSequenceFromMask(mask=('[#400f000 #1402 ]', ), )
413     p.seedEdgeBySize(edges=pickedEdges,
414                      size=0.0005,
415                      deviationFactor=0.1,
416                      minSizeFactor=0.1,
417                      constraint=FINER)
418     pickedEdges = e.getSequenceFromMask(mask=('[#f1ff0fff #2838 ]', ), )
419     p.seedEdgeBySize(edges=pickedEdges,
420                      size=0.00342673,
421                      deviationFactor=0.1,
422                      minSizeFactor=0.1,
423                      constraint=FINER)
424     # (unique node numbering)
425     p.setValues(startNodeLabel=1000000, startElemLabel=1000000)
426     p.generateMesh()
427
428
429 def G_Mesh(seed):
430     p = abqModel.parts['G']
431     p.seedPart(size=seed, deviationFactor=0.1, minSizeFactor=0.1)
432     f = p.faces
433     pickedRegions = f.getSequenceFromMask(mask=('[#3f ]', ), )
434     p.setMeshControls(regions=pickedRegions, elemShape=QUAD)
435     elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=EXPLICIT)
436     elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=EXPLICIT)
437     f = p.faces
438     faces = f.getSequenceFromMask(mask=('[#3f ]', ), )
439     pickedRegions = (faces, )
440     p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
441     # (unique node numbering)
442     p.setValues(startNodeLabel=2000000, startElemLabel=2000000)
443     p.generateMesh()
444
445
446 def T_Mesh(seed):
447     p = abqModel.parts['T']
448     p.seedPart(size=seed, deviationFactor=0.1, minSizeFactor=0.1)
449     f = p.faces
450     pickedRegions = f.getSequenceFromMask(mask=('[#ff ]', ), )
451     p.setMeshControls(regions=pickedRegions, elemShape=QUAD)
452     elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=EXPLICIT)
453     elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=EXPLICIT)
454     f = p.faces
455     faces = f.getSequenceFromMask(mask=('[#ff ]', ), )
456     pickedRegions = (faces, )
457     p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
458     # (unique node numbering)

```

```

459 p.setValues(startNodeLabel=3000000, startElemLabel=3000000)
460 p.generateMesh()
461
462
463 def R_Mesh(seed):
464     p = abqModel.parts['R']
465     p.seedPart(size=seed, deviationFactor=0.1, minSizeFactor=0.1)
466     c, e = p.cells, p.edges
467     pickedRegions = c.getSequenceFromMask(mask=['[#1 ]', ], )
468     p.setMeshControls(regions=pickedRegions,
469                       technique=SWEEP,
470                       algorithm=ADVANCING_FRONT)
471     p.setSweepPath(region=c[0], edge=e[10], sense=FORWARD)
472     elemType1 = mesh.ElemType(elemCode=C3D8R,
473                               elemLibrary=EXPLICIT,
474                               kinematicSplit=AVERAGE_STRAIN,
475                               secondOrderAccuracy=ON,
476                               hourglassControl=ENHANCED,
477                               distortionControl=ON,
478                               lengthRatio=0.100000001490116)
479     elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
480     elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT)
481     c = p.cells
482     cells = c.getSequenceFromMask(mask=['[#1 ]', ], )
483     pickedRegions = (cells, )
484     p.setElementType(regions=pickedRegions,
485                     elemTypes=(elemType1, elemType2, elemType3))
486     p.generateMesh()
487     # (unique node numbering)
488     p.setValues(startNodeLabel=4000000, startElemLabel=4000000)
489     p.generateMesh()
490
491
492 def VitreousMesh(v1Seed, v2Seed):
493     ''' Specify tetrahedral elements '''
494     p = abqModel.parts['V']
495     c = p.cells
496     pickedRegions = c.getSequenceFromMask(mask=['[#86f800 ]', ], )
497     p.setMeshControls(regions=pickedRegions, elemShape=TET, technique=FREE)
498     elemType1 = mesh.ElemType(elemCode=C3D20R)
499     elemType2 = mesh.ElemType(elemCode=C3D15)
500     elemType3 = mesh.ElemType(elemCode=C3D10)
501     cells = c.getSequenceFromMask(mask=['[#86f800 ]', ], )
502     pickedRegions = (cells, )
503     p.setElementType(regions=pickedRegions,
504                     elemTypes=(elemType1, elemType2, elemType3))
505
506     ''' Specify hexahedral elements '''
507     elemType1 = mesh.ElemType(elemCode=C3D8R, elemLibrary=EXPLICIT)
508     elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
509     elemType3 = mesh.ElemType(elemCode=C3D4,
510                               elemLibrary=EXPLICIT,
511                               secondOrderAccuracy=ON,
512                               distortionControl=ON,
513                               lengthRatio=0.100000001490116)
514     cells = c.getSequenceFromMask(mask=['[#86f800 ]', ], )
515     pickedRegions = (cells, )
516     p.setElementType(regions=pickedRegions,

```

```

517         elemTypes=(elemType1, elemType2, elemType3))
518
519     elemType1 = mesh.ElemType(elemCode=C3D8R,
520                               elemLibrary=EXPLICIT,
521                               kinematicSplit=AVERAGE_STRAIN,
522                               secondOrderAccuracy=ON,
523                               hourglassControl=ENHANCED,
524                               distortionControl=ON,
525                               lengthRatio=0.100000001490116)
526     elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
527     elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT)
528     cells = c.getSequenceFromMask(mask=('[#7907ff ]', ), )
529     pickedRegions = (cells, )
530     p.setElementType(regions=pickedRegions,
531                     elemTypes=(elemType1, elemType2, elemType3))
532
533     # Seed the entire part
534     p.seedPart(size=v2Seed, deviationFactor=0.1, minSizeFactor=0.1)
535
536     # Seed the retina interface
537     e = p.edges
538     pickedEdges = e.getSequenceFromMask(mask=('[#ffffff #7fec0fff #80012 ]',
539                                             ), )
540     p.seedEdgeBySize(edges=pickedEdges,
541                     size=v1Seed,
542                     deviationFactor=0.1,
543                     minSizeFactor=0.1,
544                     constraint=FINER)
545
546     # Seed the bias edges
547     e = p.edges
548     pickedEdges1 = e.getSequenceFromMask(mask=('[#0 #104000 #10001 ]', ), )
549     pickedEdges2 = e.getSequenceFromMask(mask=('[#0 #80020000 #900000 ]', ), )
550     p.seedEdgeByBias(biasMethod=SINGLE,
551                     end1Edges=pickedEdges1,
552                     end2Edges=pickedEdges2,
553                     minSize=v1Seed,
554                     maxSize=v2Seed,
555                     constraint=FINER)
556
557     # (unique node numbering)
558     p.setValues(startNodeLabel=5000000, startElemLabel=5000000)
559     p.generateMesh()
560
561
562     def QuadraticTetVitreous():
563         # Vitreous
564         p = abqModel.parts['V']
565         c = p.cells
566         pickedRegions = c.getSequenceFromMask(mask=('[#9f ]', ), )
567         p.deleteMesh(regions=pickedRegions)
568         p.setMeshControls(regions=pickedRegions, elemShape=TET, technique=FREE)
569         elemType1 = mesh.ElemType(elemCode=UNKNOWN_HEX, elemLibrary=EXPLICIT)
570         elemType2 = mesh.ElemType(elemCode=UNKNOWN_WEDGE, elemLibrary=EXPLICIT)
571         elemType3 = mesh.ElemType(elemCode=C3D10M, elemLibrary=EXPLICIT)
572         cells = c.getSequenceFromMask(mask=('[#9f ]', ), )
573         pickedRegions = (cells, )
574         p.setElementType(regions=pickedRegions,

```



```

575         elemTypes=(elemType1, elemType2, elemType3))
576 elemType1 = mesh.ElemType(elemCode=UNKNOWN_HEX, elemLibrary=EXPLICIT)
577 elemType2 = mesh.ElemType(elemCode=UNKNOWN_WEDGE, elemLibrary=EXPLICIT)
578 elemType3 = mesh.ElemType(elemCode=C3D10M,
579                             elemLibrary=EXPLICIT,
580                             secondOrderAccuracy=ON,
581                             distortionControl=ON,
582                             lengthRatio=0.100000001490116)
583 c = p.cells
584 p.setElementType(regions=pickedRegions,
585                  elemTypes=(elemType1, elemType2, elemType3))
586 p.generateMesh()
587
588
589 def QuadraticTetRetina():
590     # Retina
591     p = abqModel.parts['R']
592     c = p.cells
593     pickedRegions = c.getSequenceFromMask(mask=('[#1 ]', ), )
594     p.deleteMesh(regions=pickedRegions)
595     p.setMeshControls(regions=pickedRegions, elemShape=TET, technique=FREE)
596     elemType1 = mesh.ElemType(elemCode=UNKNOWN_HEX, elemLibrary=EXPLICIT)
597     elemType2 = mesh.ElemType(elemCode=UNKNOWN_WEDGE, elemLibrary=EXPLICIT)
598     elemType3 = mesh.ElemType(elemCode=C3D10M, elemLibrary=EXPLICIT)
599     c = p.cells
600     cells = c.getSequenceFromMask(mask=('[#1 ]', ), )
601     pickedRegions = (cells, )
602     p.setElementType(regions=pickedRegions,
603                      elemTypes=(elemType1, elemType2, elemType3))
604     elemType1 = mesh.ElemType(elemCode=UNKNOWN_HEX, elemLibrary=EXPLICIT)
605     elemType2 = mesh.ElemType(elemCode=UNKNOWN_WEDGE, elemLibrary=EXPLICIT)
606     elemType3 = mesh.ElemType(elemCode=C3D10M,
607                               elemLibrary=EXPLICIT,
608                               secondOrderAccuracy=ON,
609                               distortionControl=ON,
610                               lengthRatio=0.100000001490116)
611     p.setElementType(regions=pickedRegions,
612                      elemTypes=(elemType1, elemType2, elemType3))
613     p.generateMesh()
614
615
616 def Assembly():
617     a1 = abqModel.rootAssembly
618     a1.DatumCsysByDefault(CARTESIAN)
619     p = abqModel.parts['E']
620     a1.Instance(name='E-1', part=p, dependent=ON)
621     p = abqModel.parts['G']
622     a1.Instance(name='G-1', part=p, dependent=ON)
623     p = abqModel.parts['R']
624     a1.Instance(name='R-1', part=p, dependent=ON)
625     p = abqModel.parts['T']
626     a1.Instance(name='T-1', part=p, dependent=ON)
627     p = abqModel.parts['V']
628     a1.Instance(name='V-1', part=p, dependent=ON)
629
630
631 def GravityStep(time, prevStep, scaleFactor, MSTI, stepName, descrip):
632     abqModel.ExplicitDynamicsStep(name=stepName,

```

```

633         previous=prevStep,
634         description=descrip,
635         timePeriod=time,
636         massScaling=((SEMI_AUTOMATIC,
637                     MODEL,
638                     AT_BEGINNING,
639                     scaleFactor,
640                     MSTI,
641                     BELOW_MIN, 0, 0, 0.0, 0.0, 0,
642                     None), ),
643         nlgeom=ON)
644
645
646 def General_Contact(stepName, cIP):
647     # Rename the two variables
648     GC_IP = 'IntProp-GC' # Interaction property
649     GC = 'General_Contact' # General Contact name
650     # cIP = 'cohesive_IntProp' # cohesive interaction property name
651     abqModel.ContactProperty(GC_IP)
652
653     GC_IntProp = abqModel.interactionProperties[GC_IP] # simplify code
654
655     # if gravity == True:
656     # Gravity keeps the vitreous from energetically moving after peeling
657     GC_IntProp.TangentialBehavior(formulation=PENALTY,
658                                  directionality=ISOTROPIC,
659                                  slipRateDependency=OFF,
660                                  pressureDependency=OFF,
661                                  temperatureDependency=OFF,
662                                  dependencies=0,
663                                  table=((0.2, ), ),
664                                  shearStressLimit=None,
665                                  maximumElasticSlip=FRACTION,
666                                  fraction=0.005,
667                                  elasticSlipStiffness=None)
668
669     # else:
670     # # Prevent the vitreous from sliding inside the eye holder
671     # GC_IntProp.TangentialBehavior(formulation=ROUGH)
672
673     GC_IntProp.NormalBehavior(pressureOverclosure=HARD,
674                              allowSeparation=ON,
675                              constraintEnforcementMethod=DEFAULT)
676
677     abqModel.ContactExp(name=GC, createStepName=stepName)
678
679     GC_Int = abqModel.interactions[GC] # simplify code
680     GC_Int.includedPairs.setValuesInStep(stepName=stepName, useAllstar=ON)
681     GC_Int.contactPropertyAssignments.appendInStep(stepName=stepName,
682                                                    assignments=((GLOBAL,
683                                                                SELF,
684                                                                GC_IP),
685                                                                ))
686
687
688 def updateGeneralContact(stepName, Knn, Kss, Ktt, damageInitiation,
689                          tn, ts, tt, damageEvolution, FE):
690     ''' Specify the cohesive surface behavior between the retina and vitreous
        during the step after the gravity step '''

```

```

691 # Simplify
692 GC = 'General_Contact'
693 cp = 'cohesivePeel'
694
695 abqModel.ContactProperty(cp)
696
697 CP_IP = abqModel.interactionProperties[cp]
698 CP_IP.TangentialBehavior(formulation=PENALTY,
699                          directionality=ISOTROPIC,
700                          slipRateDependency=OFF,
701                          pressureDependency=OFF,
702                          temperatureDependency=OFF,
703                          dependencies=0,
704                          table=((0.2, ), ),
705                          shearStressLimit=None,
706                          maximumElasticSlip=FRACTION,
707                          fraction=0.005,
708                          elasticSlipStiffness=None)
709
710 CP_IP.CohesiveBehavior(defaultPenalties=OFF,
711                        table=((Knn, Kss, Ktt), ))
712 # eligibility=INITIAL_NODES,
713
714 CP_IP.Damage(criterion=MAX_STRESS,
715             initTable=((tn, ts, tt), ),
716             useEvolution=ON,
717             evolutionType=ENERGY,
718             evolTable=((FE, ), ),
719             useStabilization=ON,
720             viscosityCoef=1e-05)
721
722 GCI = abqModel.interactions[GC]
723 if gravity == True:
724     GCI.contactPropertyAssignments.changeValuesInStep(stepName=stepName,
725                                                         index=1,
726                                                         value=cp)
727 else:
728     r11=abqModel.rootAssembly.instances['R-1'].surfaces['R_V_Surf_BOND']
729     r12=abqModel.rootAssembly.instances['V-1'].surfaces['V_R_Surf_BOND']
730     GCI.contactPropertyAssignments.appendInStep(stepName=stepName,
731                                                  assignments=((r11, r12,
732                                                                cp), ))
733
734
735 def smoothGravity():
736     abqModel.SmoothStepAmplitude(name='smoothGravity', timeSpan=STEP,
737                                  data=((0.0, 0.0), (100.0, 1.0)))
738     abqModel.loads['Gravity'].setValues(amplitude='smoothGravity',
739                                         distributionType=UNIFORM, field='')
740
741
742 def turnTieCohesive(stepName, cohTieName):
743     ''' Simulate the tie constraint with cohesive surface '''
744     GC = 'General_Contact'
745     CTG = cohTieName # Simplify
746     abqModel.ContactProperty(CTG)
747
748     # Simplify

```

```

749 CTG_IP = abqModel.interactionProperties[CTG]
750 GC_IP = abqModel.interactions[GC]
751
752 CTG_IP.CohesiveBehavior(eligibility=INITIAL_NODES)
753 r11=abqModel.rootAssembly.instances['R-1'].surfaces['R_V_Surf_BOND']
754 r12=abqModel.rootAssembly.instances['V-1'].surfaces['V_R_Surf_BOND']
755 GC_IP.contactPropertyAssignments.appendInStep(stepName=stepName,
756                                               assignments=((r11,
757                                                           r12,
758                                                           CTG), ))
759
760
761 def peelStepPostGravity(time, stepName, previousStep, descrip, scaleFactor,
762                        MSTI):
763     ''' step after the gravity phase '''
764     abqModel.ExplicitDynamicsStep(name=stepName,
765                                   previous=previousStep,
766                                   description=descrip,
767                                   timePeriod=time,
768                                   massScaling=((SEMI_AUTOMATIC,
769                                                MODEL, AT_BEGINNING,
770                                                scaleFactor, MSTI, BELOW_MIN,
771                                                0, 0, 0.0, 0.0, 0, None), ))
772
773
774 def F_output(stepName):
775     FOutputInterval = 50 # Double the data points (Default is 20)
776     # Whole Model Fieldoutput (RF, U, NFORC)
777     abqModel.FieldOutputRequest(name='F-Output-1',
778                                 createStepName=stepName,
779                                 variables=('RF',
780                                           'U',
781                                           'NFORC'),
782                                 numIntervals=FOutputInterval)
783
784     # Set specific field output (Retina LE & S)
785     regionDef=abqModel.rootAssembly.allInstances['R-1'].sets['R_Set']
786     abqModel.FieldOutputRequest(name='Retina_LE_S',
787                                 createStepName=stepName,
788                                 variables=('LE',
789                                           'S'),
790                                 numIntervals=FOutputInterval,
791                                 region=regionDef,
792                                 sectionPoints=DEFAULT,
793                                 rebar=EXCLUDE)
794
795     # Set specific field output (Vitreous LE & S)
796     regionDef=abqModel.rootAssembly.allInstances['V-1'].sets['V_Set']
797     abqModel.FieldOutputRequest(name='Vitreous_LE_S',
798                                 createStepName=stepName,
799                                 variables=('LE',
800                                           'S'),
801                                 numIntervals=FOutputInterval,
802                                 region=regionDef,
803                                 sectionPoints=DEFAULT,
804                                 rebar=EXCLUDE)
805
806     # # Set specific field output (Rigid Body U & RF)

```

```

807 # regionDef=abqModel.rootAssembly.allInstances['G-1'].sets['G_RP_Set']
808 # abqModel.FieldOutputRequest(name='Glue_U_RF',
809 #                               createStepName=stepName,
810 #                               variables=('U', 'RF'),
811 #                               numIntervals=FOutputInterval,
812 #                               region=regionDef,
813 #                               sectionPoints=DEFAULT,
814 #                               rebar=EXCLUDE)
815
816 # # Contact Pair field output (CP - CStress, CDisp, CForce)
817 # abqModel.FieldOutputRequest(name='CF_Output_R_V',
818 #                               createStepName=stepName,
819 #                               variables=('CStress', 'CDisp', 'CForce'),
820 #                               numIntervals=FOutputInterval,
821 #                               interactions=('CP-R-V', ),
822 #                               sectionPoints=DEFAULT,
823 #                               rebar=EXCLUDE)
824
825
826 # # Contact Pair (Bond) field output (CP - CStress, CDisp, CForce)
827 # abqModel.FieldOutputRequest(name='CF_Output_BOND',
828 #                               createStepName=stepName,
829 #                               variables=('CStress', 'CDisp', 'CForce'),
830 #                               interactions=('CP_BOND', ),
831 #                               sectionPoints=DEFAULT,
832 #                               rebar=EXCLUDE)
833
834
835 def H_output(stepName):
836     # Internal/Kinetic Energy
837     abqModel.HistoryOutputRequest(name='H-Output-1',
838                                     createStepName=stepName,
839                                     variables=('ALLIE',
840                                                 'ALLKE'))
841
842     # # Define specific reaction force on the glue reference point
843     # a = abqModel.rootAssembly
844     # regionDef=a.allInstances['G-1'].sets['G_RP_Set']
845     # abqModel.HistoryOutputRequest(name='G_RP_Output_U_RF_RM',
846     #                               createStepName=stepName,
847     #                               variables=('U1', 'U2', 'U3',
848     #                                         'RF1', 'RF2', 'RF3',
849     #                                         'RM1', 'RM2', 'RM3'),
850     #                               region=regionDef,
851     #                               sectionPoints=DEFAULT,
852     #                               rebar=EXCLUDE)
853
854     # # Define specific CFN between the vitreous and retina
855     # regionDef=a.allInstances['V-1'].sets['V_R_Set']
856     # abqModel.HistoryOutputRequest(name='Contact-VR-Hist',
857     #                               createStepName=stepName,
858     #                               variables=('CFN1', 'CFN2', 'CFN3', 'CAREA'),
859     #                               region=regionDef,
860     #                               sectionPoints=DEFAULT,
861     #                               rebar=EXCLUDE)
862
863     # # Define specific CFN between the vitreous and retina
864     # abqModel.HistoryOutputRequest(name='Contact_CP-R-V',

```

```

865         #                                     createStepName=stepName,
866         #                                     variables=('CFN1', 'CFN2', 'CFN3', 'CAREA'),
867         #                                     interactions=('CP-R-V', ),
868         #                                     sectionPoints=DEFAULT,
869         #                                     rebar=EXCLUDE)
870
871     # # Define specific CFN between the retina and glue
872     # abqModel.HistoryOutputRequest(name='Contact_CP-R-G',
873     #                               createStepName=stepName,
874     #                               variables=('CFN1', 'CFN2', 'CFN3', 'CAREA'),
875     #                               interactions=('CP-R-G', ),
876     #                               sectionPoints=DEFAULT,
877     #                               rebar=EXCLUDE)
878
879
880 def CP_RV():
881     abqModel.ContactProperty('CP_R_V_Int_Prop')
882     a = abqModel.rootAssembly
883     s1 = a.instances['R-1'].faces
884     side1Faces1 = s1.getSequenceFromMask(mask=('#4 ', ), )
885     a.Surface(side1Faces=side1Faces1, name='CP-R_V')
886     region1=a-surfaces['CP-R_V']
887     s1 = a.instances['V-1'].faces
888     side1Faces1 = s1.getSequenceFromMask(mask=('#5090 ', ), )
889     a.Surface(side1Faces=side1Faces1, name='CP-V_R')
890     region2=a-surfaces['CP-V_R']
891     abqModel.SurfaceToSurfaceContactExp(name='CP-R-V',
892                                         createStepName='Initial',
893                                         master=region1,
894                                         slave=region2,
895                                         sliding=FINITE,
896                                         interactionProperty='CP_R_V_Int_Prop',
897                                         weightingFactorType=SPECIFIED,
898                                         weightingFactor=1.0,
899                                         initialClearance=OMIT,
900                                         datumAxis=None,
901                                         clearanceRegion=None)
902
903
904 def Amp():
905     abqModel.SmoothStepAmplitude(name='TD_amp',
906                                  timeSpan=STEP,
907                                  data=((0.0, 0.0),
908                                         (30.0, 2e-05))
909                                  )
910     abqModel.SmoothStepAmplitude(name='omega',
911                                  timeSpan=STEP,
912                                  data=((0.0, 0.0),
913                                         (30.0, 0.000909174))
914                                  )
915
916
917 def EHR_BC_Fixed(stepName):
918     a = abqModel.rootAssembly
919     region = a.instances['E-1'].sets['E_RP_Set']
920     abqModel.VelocityBC(name='EHR', createStepName=stepName, region=region,
921                         v1=0.0, v2=0.0, v3=0.0, vr1=0.0, vr2=0.0, vr3=0.0,
922                         amplitude='omega', localCsys=None,

```

```

923         distributionType=UNIFORM, fieldName='')
924
925
926 def EHR_BC(stepName):
927     a = abqModel.rootAssembly
928     region = a.instances['E-1'].sets['E_RP_Set']
929     abqModel.VelocityBC(name='EHR',
930                          createStepName=stepName,
931                          region=region,
932                          v1=0.0, v2=0.0, v3=0.0,
933                          vr1=0.0, vr2=0.0, vr3=-1.0,
934                          amplitude='omega',
935                          localCsys=None,
936                          distributionType=UNIFORM,
937                          fieldName='')
938
939
940 def Retina_Displacement_BC(stepName):
941     a = abqModel.rootAssembly
942     region = a.instances['R-1'].sets['R_G_Set']
943     abqModel.VelocityBC(name='R_Vel',
944                          createStepName=stepName,
945                          region=region,
946                          v1=0.866092,
947                          v2=0.499884,
948                          v3=UNSET,
949                          vr1=UNSET,
950                          vr2=UNSET,
951                          vr3=UNSET,
952                          amplitude='TD_amp',
953                          localCsys=None,
954                          distributionType=UNIFORM,
955                          fieldName='')
956
957
958 def Write_Job(jobName, modelName, jobDescription):
959     mdb.Job(name=jobName,
960            model=modelName,
961            description=jobDescription,
962            type=ANALYSIS,
963            atTime=None,
964            waitMinutes=0,
965            waitHours=0,
966            queue=None,
967            memory=90,
968            memoryUnits=PERCENTAGE,
969            explicitPrecision=DOUBLE,
970            nodalOutputPrecision=SINGLE,
971            echoPrint=OFF,
972            modelPrint=OFF,
973            contactPrint=OFF,
974            historyPrint=OFF,
975            userSubroutine='',
976            scratch='',
977            resultsFormat=ODB,
978            parallelizationMethodExplicit=DOMAIN,
979            numDomains=14,
980            activateLoadBalancing=False,

```

```

981         multiprocessingMode=DEFAULT,
982         numCpus=14)
983
984
985 def Save_INP(jobName):
986     mdb.jobs[jobName].writeInput(consistencyChecking=OFF)
987
988
989 def VR_Tie():
990     a = abqModel.rootAssembly
991     slaveSurf=a.instances['V-1'].surfaces['V_R_Surf_BOND']
992     mastSurf=a.instances['R-1'].surfaces['R_V_Surf_BOND']
993     abqModel.Tie(name='RV_Tie',
994                 master=mastSurf,
995                 slave=slaveSurf,
996                 positionToleranceMethod=COMPUTED,
997                 adjust=OFF,
998                 tieRotations=ON,
999                 constraintEnforcement=SURFACE_TO_SURFACE,
1000                 thickness=ON)
1001     return '_VR_Tie'
1002
1003
1004 def keywordBlockR_G_SET_NodeNum():
1005     # Work here because the abaqus default is to use Nset Generate
1006     # Think about this method. Used to be V_R_Set
1007     a = abqModel
1008     modelkwb = a.keywordBlock
1009     assembly = a.rootAssembly
1010
1011     # Synch edits to modelkwb with those made in the model. We don't need
1012     # access to *nodes and *elements as they would appear in the inp file,
1013     # so set the storeNodesAndElements arg to False.
1014     modelkwb.synchVersions(storeNodesAndElements=False)
1015
1016     # Search the modelkwb for the desired insertion point. If it is found, we
1017     # break the loop, storing the line number, and then write our keywords
1018     # using the insert method (which actually inserts just below the specified
1019     # line number, fyi).
1020     line_num = 0
1021     for n, line in enumerate(modelkwb.sisBlocks):
1022         if line.find('*Nset, nset=R_G_Set') >= 0:
1023             line_num = n
1024             break
1025     if line_num:
1026         line = line.replace('\n', ',') # replaces the new line with commas
1027
1028     if line.find('generate') == -1:
1029         if line[-1] == ',':
1030             # if the node list ends with a comma
1031             # split up the string into ints
1032             nList = [int(i) for i in line[20:-1].split(',')]
1033         else:
1034             # split up the string into ints
1035             nList = [int(i) for i in line[20:].split(',')]
1036
1037         nodeNum = len(nList) # count the number of nodes
1038     else:

```



```

1039         # use the equation in ABQ documentation to determine the number
1040         # of nodes in the set
1041         # nNodes = (n2-n1)/increment
1042         # split up the string into ints
1043         nList = [int(i) for i in line[30:].split(',')]
1044         n1 = nList[0] # first node
1045         n2 = nList[1] # last node
1046         increment = nList[2]
1047         nodeNum = (n2 - n1)/increment + 1
1048         print(nodeNum, 'nodes in nset=R_G_Set')
1049     else:
1050         e = ("Error: R_G_Set was not found in the Model KeywordBlock to " +
1051             "determine the number of nodes in the nodeSet.")
1052         raise Exception(" ".join(e))
1053     return nodeNum
1054
1055
1056 def keywordBlockBond(MSTI, scaleFactor, FN, FS, db, ufn, ufs):
1057     a = abqModel
1058     modelkwb = a.keywordBlock
1059     assembly = a.rootAssembly
1060
1061     #if assembly.isOutOfDate:
1062     #    assembly.regenerate()
1063
1064     # Synch edits to modelkwb with those made in the model. We don't need
1065     # access to *nodes and *elements as they would appear in the inp file,
1066     # so set the storeNodesAndElements arg to False.
1067     modelkwb.synchVersions(storeNodesAndElements=False)
1068
1069     # Search the modelkwb for the desired insertion point. If it is found, we
1070     # break the loop, storing the line number, and then write our keywords
1071     # using the insert method (which actually inserts just below the specified
1072     # line number, fyi).
1073
1074     # The abaqus keyword "*BOND" needs to be defined after the "*STEP" keyword
1075     # in the .inp file. Search for the "*Fixed Mass Scaling ..." term to
1076     # add text specific to the bond
1077     keywordSearch = ('*Fixed Mass Scaling, ' +
1078                     'dt={}, type=below min, '.format(MSTI) +
1079                     'factor={:.0f}'.format(scaleFactor))
1080
1081     line_num = 0
1082     for n, line in enumerate(modelkwb.sieBlocks):
1083         if line == keywordSearch:
1084             line_num = n
1085             break
1086     if line_num:
1087         kwds = ('*Contact Pair, interaction=Bond_Int_Prop, mechanical ' +
1088               'constraint=KINEMATIC, weight=0., ' +
1089               'cpset=CP_Bond\nV-1.V_R_SURF_BOND, ' +
1090               'R-1.R_V_SURF_BOND\n*Surface Interaction, ' +
1091               'name=Bond_Int_Prop\n' +
1092               '*BOND\nV-1.V_R_SET, {}'.format(FN) +
1093               ', {}'.format(FS) +
1094               ', {}'.format(db) +
1095               ', '+, {}'.format(ufn) +
1096               ', {}'.format(ufs))
1097         modelkwb.insert(position=line_num, text=kwds)

```

```

1097     else:
1098         e = ("Error: Mass Scaling was not found in the Model KeywordBlock.")
1099         raise Exception(" ".join(e))
1100
1101
1102 def keywordBlockBondNPDFM(FN, FS):
1103     a = abqModel
1104     modelkwb = a.keywordBlock
1105     assembly = a.rootAssembly
1106
1107     #if assembly.isOutOfDate:
1108     #    assembly.regenerate()
1109
1110     # Synch edits to modelkwb with those made in the model. We don't need
1111     # access to *nodes and *elements as they would appear in the inp file,
1112     # so set the storeNodesAndElements arg to False.
1113     modelkwb.synchVersions(storeNodesAndElements=False)
1114
1115     # Search the modelkwb for the desired insertion point. If it is found, we
1116     # break the loop, storing the line number, and then write our keywords
1117     # using the insert method (which actually inserts just below the specified
1118     # line number, fyi).
1119     line_num = 0
1120     for n, line in enumerate(modelkwb.sieBlocks):
1121         # Use this line because CP from defined in code doesn't seem to work..
1122         if line == '*Contact Property Assignment\n , , IntProp-GC':
1123             line_num = n
1124             break
1125     if line_num:
1126         kwds = ('*Contact Pair, interaction=Bond_Int_Prop, mechanical ' +
1127                'constraint=KINEMATIC, weight=0., ' +
1128                'cpset=CP_Bond\nV-1.V_R_SURF_BOND, ' +
1129                'R-1.R_V_SURF_BOND\n*Surface Interaction, ' +
1130                'name=Bond_Int_Prop\n' +
1131                '*BOND\nV-1.V_R_SET, {}'.format(FN) +
1132                ', {}'.format(FS) +
1133                ', , ')
1134         modelkwb.insert(position=line_num, text=kwds)
1135     else:
1136         e = ("Error: Mass Scaling was not found in the Model KeywordBlock.")
1137         raise Exception(" ".join(e))
1138
1139
1140 def keywordBlockBondHistOutput():
1141     a = abqModel
1142     modelkwb = a.keywordBlock
1143     assembly = a.rootAssembly
1144
1145     # Synch edits to modelkwb with those made in the model. We don't need
1146     # access to *nodes and *elements as they would appear in the inp file,
1147     # so set the storeNodesAndElements arg to False.
1148     modelkwb.synchVersions(storeNodesAndElements=False)
1149
1150     # Search the modelkwb for the desired insertion point. If it is found, we
1151     # break the loop, storing the line number, and then write our keywords
1152     # using the insert method (which actually inserts just below the specified
1153     # line number, fyi).
1154     line_num = 0

```

```

1155     for n, line in enumerate(modelkwb.sieBlocks):
1156         #print(n, line)
1157         if line == "*Energy Output\nALLIE, ALLKE":
1158             line_num = n
1159             break
1160     if line_num:
1161         kwds = '*Contact Output, Nset=V-1.V_R_SET\nBONDSTAT, BONDLOAD'
1162         modelkwb.insert(position=line_num, text=kwds)
1163     else:
1164         e = ("Error: Bond Output was not found in the Model KeywordBlock.")
1165         raise Exception(" ".join(e))
1166
1167
1168 # Added 9/11/2020
1169 def keywordBlockBondFieldOutput():
1170     a = abqModel
1171     modelkwb = a.keywordBlock
1172     assembly = a.rootAssembly
1173
1174     modelkwb.synchVersions(storeNodesAndElements=False)
1175     # Add history output for the contact
1176     line_num = 0
1177     for n, line in enumerate(modelkwb.sieBlocks):
1178         if line == '*Element Output, directions=YES\nNFORC, ':
1179             line_num = n
1180             break
1181     if line_num:
1182         kwds = ('** FIELD OUTPUT: CF_Output_R_V\n\n**\n*Contact Output, ' +
1183                'cpset=CP_Bond\nCDISP, CFORCE, CSTRESS')
1184         modelkwb.insert(position=line_num, text=kwds)
1185     else:
1186         e = ("Error: Bond Output was not found in the Model KeywordBlock.")
1187         raise Exception(" ".join(e))
1188
1189
1190 def Submit_job(jobname):
1191     myJob = mdb.jobs[jobname]
1192     try:
1193         myJob.submit(consistencyChecking=OFF)
1194         myJob.waitForCompletion()
1195     except:
1196         print(str(datetime.datetime.now())+' stop by error!')
1197         pass
1198
1199
1200 def FEA():
1201     """
1202     Function that generates FEA code to model vitreoretinal adhesion
1203
1204     # Steps are as follows:
1205     1 - Create new model database
1206     2 - Import SolidWorks STEP file (Includes all parts)
1207     3 - Material property definitions
1208     4 - Part features (Element & Node Sets & Reference Points ...)
1209     5 - Mesh parts (Specify seed size)
1210     6 - Assembly
1211     7 - Step (Dynamic Explicit with Mass Scaling)
1212     8 - Outputs (Field & History)

```

```

1213     9 - Contact (General Contact)
1214     10 - Contact pair (Retina/Vitreous - Bonded Surface)
1215     11 - Tie Constraint (Retina - Glue)
1216     12 - Amplitude definition
1217     13 - BC's'
1218     14 - Submit Job :)
1219 """
1220
1221 # Import SolidWorks STEP file
1222 ImportStepEyeConstrained()
1223
1224 # Mat Props
1225 Retina_Mat_Prop(RetinaProp)
1226 Vitreous_Mat_Prop(VitreousProp)
1227
1228 # # Part Geometry/RPs/Sets/Surfaces
1229 E_Features()
1230 G_Features()
1231 T_Features()
1232 R_Features()
1233
1234 # Internal sphere to reduce mesh
1235 V_Partition_XYZ_Axis()
1236 V_Internal_Sphere()
1237 AssembleV_for_Merging()
1238 mergeV()
1239
1240 # Features on the vitreous
1241 PartitionRetinaOnVitreous()
1242 Vitreous_Features()
1243
1244 # Seed & Mesh parts
1245 E_Mesh(e1Seed, e2Seed) # Max/min
1246 G_Mesh(gSeed)
1247 T_Mesh(ptSeed)
1248 R_Mesh(rSeed)
1249 VitreousMesh(v1Seed, v2Seed)
1250
1251 # Assembly
1252 Assembly()
1253
1254 # Eliminate the glue and tab from the model
1255 a = abqModel.rootAssembly
1256 a.features['G-1'].suppress()
1257 a.features['T-1'].suppress()
1258
1259 # Gravity Step
1260 previousStep = 'Initial'
1261 if gravity == True:
1262     stepName = 'Gravity_Step'
1263     descrip = ('Applying gravity to the model and letting the ' +
1264               'vitreous and retina settle')
1265
1266     GravityStep(200, previousStep, scaleFactor, 0.03125, stepName, descrip)
1267     Gravity(stepName)
1268     smoothGravity()
1269
1270 # Interactions

```

```

1271     cohTieName = 'Cohesive_Gravity_Tie'
1272     General_Contact(stepName, cohTieName)
1273
1274     # Interaction properties
1275     turnTieCohesive(stepName, cohTieName)
1276
1277     # Zero movement boundary conditions
1278     Amp()
1279     EHR_BC_Fixed(stepName)
1280
1281     # # Model outputs for gravity step
1282     F_output(stepName)
1283     H_output(stepName)
1284
1285     previousStep = stepName # Update the previous step to be gravity
1286 else:
1287     ''' General contact ''' # fix here if no gravity is specified
1288     peelCoh = 'Cohesive_Peel_Int'
1289     General_Contact(previousStep, peelCoh)
1290
1291
1292     # # Peel Step
1293     stepName = 'Peel_Test_Dynamic_Explicit'
1294     descrip = 'Peel the retina away from the vitreous (rotational peel test)'
1295     peelStepPostGravity(time, stepName, previousStep, descrip, scaleFactor,
1296                        MSTI)
1297
1298     # Interactions
1299     # CP_RV() # comment out?
1300
1301     # Boundary conditions
1302     Amp()
1303     EHR_BC(stepName)
1304     Retina_Displacement_BC(stepName)
1305
1306     # Model Outputs
1307     F_output(stepName)
1308     H_output(stepName)
1309
1310     if tieInterface == True:
1311         ''' If The VR interface is tied '''
1312         VR_Tie()
1313     else:
1314         ''' If bonding is true, then add the *Bond info to the .inp file '''
1315         # determine number of nodes in the nodeSet
1316         nodeNum = keywordBlockR_G_SET_NodeNum()
1317
1318         # Bonding
1319         # divide the bond tension load by the number of nodes in the set
1320         FN_Norm = FN/nodeNum
1321         # divide the bond shear load by the number of nodes in the set
1322         FS_Norm = FS/nodeNum
1323
1324     if PDFMStatus == True:
1325         # Include post damage failure model
1326         # FN, FS, SpotWeldRadius, ufn, ufs
1327         keywordBlockBond(MSTI, scaleFactor, FN_Norm,
1328                        FS_Norm, db, ufn, ufs)

```

```

1329
1330     else:
1331         # Do not include post damage failure model
1332         keywordBlockBondNPDFM(FN_Norm, FS_Norm)
1333
1334         keywordBlockBondHistOutput()
1335         keywordBlockBondFieldOutput()
1336
1337     # Undo the spacing to pass in the job description
1338     global jobDescription
1339     # replace new lines, spaces, equal signs
1340     jobDescription = jobDescription.replace('NEWLINE', '\n')
1341     jobDescription = jobDescription.replace('TAB', '\t')
1342     jobDescription = jobDescription.replace('SPACE', ' ')
1343     jobDescription = jobDescription.replace('EQUALSSIGN', '=')
1344
1345     Write_Job(jobName, modelName, jobDescription)
1346     print('Job has been written')
1347     Save_INP(jobName)
1348     Submit_job(jobName)
1349     print('Job has been submitted')
1350     del mdb.models['Model-1']
1351
1352
1353 # In[Main import info]
1354
1355
1356 if __name__ == '__main__':
1357     """ Run the following function """
1358
1359     # Print File of tests & attributes ran in order to make sure they are
1360     # being properly pass through
1361     print("\nWriting out the Argument Data...")
1362     filename = os.path.join(abqWD, 'FEAArgumentData' + '.txt')
1363     outfile = open(filename, 'w')
1364     outfile.write('sys.argv\n')
1365     outfile.write('\n'.join(sys.argv)) # write all arguments passed into abaqus
1366     outfile.close()
1367     print("\nDone!")
1368     print("\nThe output file will be named '{}'.format(filename) + ''")
1369     print("\nIt will be in the same working directory as your Abaqus model\n")
1370
1371     # # Testing when importing into abaqus script
1372     # gravity = eval('False') # gravity
1373     # symmetry = eval('False') # symmetry
1374     # simplified = eval('True') # simplified model (not used anymore)
1375     # modelName = 'ABQScript' # model name
1376     # jobName = 'jobNameTest' # file name/job name
1377     # time = float('50')
1378     # e1Seed = '[10,1,0.0009765625]'
1379     # e2Seed = '[8,1,0.00390625]'
1380     # ptSeed = '[6,1,0.015625]'
1381     # gSeed = '[7,1,0.0078125]'
1382     # v1Seed = '[10,1,0.0009765625]' #
1383     ↪ '[11.38,1,0.00037521366730664343]' #
1384     # v2Seed = '[8,1,0.00390625]'
1385     # rSeed = '[10,1,0.0009765625]'
1386     # scaleFactor = '[0,1,1]'

```

```

1386 # MSTI = '[4,1,0.0625]' # MassScaleTimeIncrement
1387 # RetinaProp = float('11120.0') # Young's modulus for retina
1388 # VitreousProp = float('400') # Young's modulus for vitreous
1389 # BondStatus = eval('True')
1390 # FN = '[-5,1,0.03125]'
1391 # FS = '[-5,1,0.03125]'
1392 # PDFMStatus = eval('True') # True/False - convert to bool
1393 # db = '[-5,1,0.03125]'
1394 # ufn = '[-5,1,0.03125]'
1395 # ufs = '[-5,1,0.03125]'
1396 # OptimizationStatus = 'None' # None/variables to be optimized
1397 # tieInterface = eval('False') # True/False - convert to bool
1398 # jobDescription = """Test Model via Abaqus Run Script"""
1399
1400 # Pass in arguments from previous file Strip the brackets from the strings
1401 gravity = sys.argv[-27]
1402 symmetry = sys.argv[-26]
1403 simplified = sys.argv[-25]
1404 modelName = sys.argv[-24] # model name
1405 jobName = sys.argv[-23] # file name/job name
1406 time = float(sys.argv[-22])
1407 e1Seed = sys.argv[-21]
1408 e2Seed = sys.argv[-20]
1409 ptSeed = sys.argv[-19]
1410 gSeed = sys.argv[-18]
1411 v1Seed = sys.argv[-17]
1412 v2Seed = sys.argv[-16]
1413 rSeed = sys.argv[-15]
1414 scaleFactor = sys.argv[-14]
1415 MSTI = sys.argv[-13]
1416 RetinaProp = float(sys.argv[-12]) # Young's modulus for retina
1417 VitreousProp = float(sys.argv[-11]) # Young's modulus for vitreous
1418 BondStatus = eval(sys.argv[-10])
1419 FN = sys.argv[-9]
1420 FS = sys.argv[-8]
1421 PDFMStatus = eval(sys.argv[-7]) # True/False - convert to bool
1422 db = sys.argv[-6]
1423 ufn = sys.argv[-5]
1424 ufs = sys.argv[-4]
1425 OptimizationStatus = sys.argv[-3] # None/variables to be optimized
1426 tieInterface = sys.argv[-2]
1427 jobDescription = sys.argv[-1] # String
1428
1429 """ Convert the strings back to lists of floats """
1430 e1SeedStrip = str(e1Seed)[1:-1] # Strip the brackets from the string
1431 e1SeedList = [float(i) for i in e1SeedStrip.split(',')]
1432 e1Seed = e1SeedList[2] # value
1433
1434 e2SeedStrip = str(e2Seed)[1:-1] # Strip the brackets from the string
1435 e2SeedList = [float(i) for i in e2SeedStrip.split(',')]
1436 e2Seed = e2SeedList[2] # value
1437
1438 ptSeedStrip = str(ptSeed)[1:-1] # Strip the brackets from the string
1439 ptSeedList = [float(i) for i in ptSeedStrip.split(',')]
1440 ptSeed = ptSeedList[2] # value
1441
1442 gSeedStrip = str(gSeed)[1:-1] # Strip the brackets from the string
1443 gSeedList = [float(i) for i in gSeedStrip.split(',')]

```

```

1444 gSeed = gSeedList[2] # value
1445
1446 v1SeedStrip = str(v1Seed)[1:-1] # Strip the brackets from the string
1447 v1SeedList = [float(i) for i in v1SeedStrip.split(',')]
1448 v1Seed = v1SeedList[2] # value
1449
1450 v2SeedStrip = str(v2Seed)[1:-1] # Strip the brackets from the string
1451 v2SeedList = [float(i) for i in v2SeedStrip.split(',')]
1452 v2Seed = v2SeedList[2] # value
1453
1454 rSeedStrip = str(rSeed)[1:-1] # Strip the brackets from the string
1455 rSeedList = [float(i) for i in rSeedStrip.split(',')]
1456 rSeed = rSeedList[2] # value
1457
1458 # Strip the brackets from the string
1459 scaleFactorStrip = str(scaleFactor)[1:-1]
1460 scaleFactorList = [float(i) for i in scaleFactorStrip.split(',')]
1461 scaleFactor = scaleFactorList[2] # value
1462
1463 # Strip the brackets from the string
1464 # MassScaleTimeIncrement
1465 MSTIStrip = str(MSTI)[1:-1]
1466 MSTIList = [float(i) for i in MSTIStrip.split(',')]
1467 MSTI = MSTIList[2] # value
1468
1469 # Strip the brackets from the string
1470 FNStrip = str(FN)[1:-1]
1471 FNList = [float(i) for i in FNStrip.split(',')]
1472 FN = FNList[2] # value
1473
1474 # Strip the brackets from the string
1475 FSSStrip = str(FS)[1:-1]
1476 FSList = [float(i) for i in FSSStrip.split(',')]
1477 FS = FSList[2] # value
1478
1479 dbStrip = str(db)[1:-1] # Strip the brackets from the string
1480 dbList = [float(i) for i in dbStrip.split(',')]
1481 db = dbList[2] # value
1482
1483 ufnStrip = str(ufn)[1:-1] # Strip the brackets from the string
1484 ufnList = [float(i) for i in ufnStrip.split(',')]
1485 ufn = ufnList[2] # value
1486
1487 ufsStrip = str(ufs)[1:-1] # Strip the brackets from the string
1488 ufsList = [float(i) for i in ufsStrip.split(',')]
1489 ufs = ufsList[2] # value
1490
1491 """ Write the FEA Code """
1492 Mdb()
1493 modelDescription = ('Measure adhesion between the retina & vitreous of ' +
1494                    'the human eye using the Abaqus *Bond technique')
1495 abqModel = mdb.Model(name=modelName,
1496                      description=modelDescription,
1497                      modelType=STANDARD_EXPLICIT,
1498                      copyInteractions=ON,
1499                      copyConstraints=ON)
1500
1501 # Call the function

```



### 1.5.4 Abaqus Extract Data Script

</> Script 7: Python script used to extract data from the output database file (.odb). </>

```

1  """
2  Created on Wed Jun 17 16:48:49 2020
3
4  @author: Kiffer Creveling
5  Instructions:
6      1) Save this script in a folder containing your ODB file
7      2) Open a command window and navigate to your directory containing this script
8      ↪ 3) Create a .bat file
9      3) Issue the command to call the script and extract data:
10         abaqus python -c "import BpT; BpT.data_extract('xxxxxx.odb')"
11  """
12  # *****
13  from odbAccess import *
14  import odbAccess as oa
15  from sys import argv, exit
16  from abaqusConstants import *
17  from textRepr import *
18  import pdb
19  import numpy as np
20  import os
21
22  """ Pass arguments into this script """
23  script = sys.argv[0]
24  jobName = sys.argv[1]
25  gravity = eval(sys.argv[2]) # True/False
26  symmetry = eval(sys.argv[3]) # True/False
27  simplified = eval(sys.argv[4]) # True/False
28  BondStatus = eval(sys.argv[5]) # True/False
29  PDFMStatus = eval(sys.argv[6]) # True/False # not used in the extraction
30
31  def openOdb(jobName):
32      """
33      Function used to locate the .odb given a file name
34
35      Parameters
36      -----
37      jobName : Name of the ABAQUS .odb file
38
39      Returns
40      -----
41      odb : Abaqus output file
42      """
43      if jobName.endswith('.odb'):
44          odbFile = jobName
45          try:
46              odb=oa.openOdb(path=odbFile, readOnly=TRUE)
47              print("\nOpening the odb file... (.odb was specified)")
48              return odb
49          except:

```

```

50         print("ERROR: Unable to open the specified odb %s. Exiting."
51               % odbFile)
52         exit(0)
53
54     else:
55         odbFile = jobName + '.odb'
56         # Try opening the odb file
57         try:
58             odb=oa.openOdb(path=odbFile, readOnly=TRUE)
59             print("\nOpening the odb file... (Searching for .odb)")
60             return odb
61         except:
62             print("ERROR: Unable to open the specified odb %s. Exiting."
63                   % odbFile)
64             exit(0)
65
66 def data_extract(jobName):
67     """
68     Function used to extract data from the .odb file
69
70     Parameters
71     -----
72     jobName : The name of ABAQUS .odb file
73
74     Returns
75     -----
76     Two files of data used for plotting
77     """
78
79     # due to symmetry multiply the values by 2
80     if symmetry == True:
81         mult = 2
82     else:
83         mult = 1
84
85     frames = []
86     try:
87         odb = openOdb(jobName)
88     except:
89         print(os.getcwd())
90         print("Looks like there is a problem with the job name or odb file")
91
92     theta = 30
93     LoadCellDirection = [np.cos(theta*np.pi/180), np.sin(theta*np.pi/180), 0]
94
95     """ Field Output data arrays """
96     RF = []
97
98     # vector components of the reaction force
99     RFx = []
100     RFy = []
101     RFz = []
102
103     U_top = [] # values to append
104     U_bot = [] # values to append
105     Nforc = []
106
107     # Used to calculate bond distance

```

```

108 R_bot = [] # bottom of retina
109 V_top = [] # top of vitreous
110 Bond_disp = [] # Bond separation distance
111
112 CnormF_RV = []
113 CnormF_VR = []
114 Cpress_RV = []
115 Cpress_VR = []
116 Cshear1_RV = []
117 Cshear1_VR = []
118 Cshear2_RV = []
119 Cshear2_VR = []
120 CshearF_RV = []
121 CshearF_VR = []
122
123 # Cpress_RV = []
124 # Cpress_VR = []
125 Cpress_RV_AVG = []
126 Cpress_VR_AVG = []
127 frames = [] # List of frames
128 time = [] # Time array
129
130 # Used for reaction force simplicity further in the code
131 temp = [] # Temporary array used for iterating (Clears after each iteration)
132 tempx = []
133 tempy = []
134 tempz = []
135
136 # List variables for exporting data
137 CnormF_RV_List = []
138 CnormF_VR_List = []
139 Cpress_RV_List = []
140 Cpress_VR_List = []
141 Cshear1_RV_List = []
142 Cshear1_VR_List = []
143 Cshear2_RV_List = []
144 Cshear2_VR_List = []
145 CshearF_RV_List = []
146 CshearF_VR_List = []
147
148 """ History Output data arrays """
149 Hist_Time = []
150 IE = []
151 KE = []
152 CAreaCP_RG = []
153 CAreaCP_GR = []
154 CAreaCP_RV = []
155 CAreaCP_VR = []
156 CFNCP_RG = []
157 CFNCP_GR = []
158 CFNCP_RV = []
159 CFNCP_VR = []
160 Glue_RP_RF = []
161
162 """ Loop over the field outputs """
163 step = odb.steps.keys() # determines the step in the abaqus odb file (typically
    ↪ displacement)
164 disp_step = step[0] # Defines the step as a variable name

```

```

165 for frame, odbFrame in enumerate(odb.steps[disp_step].frames):
166     frames.append(frame) # Construct a list of all of the frames
167
168     """ Extract ODB fieldOutputs """
169     fieldOutput = odbFrame.fieldOutputs
170
171     # Print the time during the simulation
172     print(odbFrame.description)
173     time.append(odbFrame.frameValue)
174
175     """ Abaqus Instances (Parts) """
176     odbInstance = odb.rootAssembly.instances
177
178     if simplified == False:
179         # If Simp is not in the title
180         part_E = odbInstance.keys(0)[0]
181         part_G = odbInstance.keys(0)[1]
182         part_R = odbInstance.keys(0)[2]
183         part_T = odbInstance.keys(0)[3]
184         part_V = odbInstance.keys(0)[4]
185
186     elif simplified == True:
187         # If simplification exists, omit the glue & tab
188         part_E = odbInstance.keys(0)[0]
189         part_R = odbInstance.keys(0)[1]
190         part_V = odbInstance.keys(0)[2]
191     else:
192         print('Error in part definitions')
193
194     """ Nodal displacements """
195     fieldObject_U = fieldOutput['U'] # displacements
196
197     if simplified == False:
198         # If Simp is not in the title
199
200         # Glue
201         Displacements =
202         ↪ fieldObject_U.getSubset(region=odbInstance[part_G].nodeSets['G_RP_SET'])
203         for Uyi in Displacements.values: # Loops over each node in the "SET"
204             ↪ defined by the displacement
205             Uyi_vec = [Uyi.data[0], Uyi.data[1], Uyi.data[2]]
206             # Find the magnitude
207             temp.append(np.dot(Uyi_vec, LoadCellDirection)) # Creates a list of
208             ↪ displacements in the "SET"
209
210         SU = np.sum(temp) # Sums up the list of displacements from the "SET"
211         AvgU_top = SU/len(temp) # Divide by the number of nodes in the set to get
212             ↪ average
213         U_top.append(AvgU_top) # Adds the total displacement to the U-array by
214             ↪ summing across each step
215         temp = [] # Clear the array for the next iteration in the loop
216
217     elif simplified == True:
218         # If simplification exists, omit the values
219
220         Displacements =
221         ↪ fieldObject_U.getSubset(region=odbInstance[part_R].nodeSets['R_G_SET'])

```

```

216         for Uyi in Displacements.values: # Loops over each node in the "SET"
217             ↳ defined by the displacement
218             Uyi_vec = [Uyi.data[0], Uyi.data[1], Uyi.data[2]]
219             # Find the magnitude
220             temp.append(np.dot(Uyi_vec, LoadCellDirection)) # Creates a list of
221             ↳ displacements in the "SET"
222
223         SU = np.sum(temp) # Sums up the list of displacements from the "SET"
224         AvgU_top = SU/len(temp) # Divide by the number of nodes in the set to get
225         ↳ average
226         U_top.append(AvgU_top) # Adds the total displacement to the U-array by
227         ↳ summing across each step
228         temp = [] # Clear the array for the next iteration in the loop
229
230     else:
231         print('Error in nodal displacements')
232
233     """ Bond Distance """
234     Displacements =
235     ↳ fieldObject_U.getSubset(region=odbInstance[part_R].nodeSets['R_V_SET'])
236     for Uyi in Displacements.values: # Loops over each node in the "SET" defined
237     ↳ by the displacement
238         Uyi_vec = [Uyi.data[0], Uyi.data[1], Uyi.data[2]]
239         # Find the magnitude
240         temp.append(np.dot(Uyi_vec, LoadCellDirection)) # Creates a list of
241         ↳ displacements in the "SET"
242
243     SU = np.sum(temp) # Sums up the list of displacements from the "SET"
244     AvgR_bot = SU/len(temp) # Divide by the number of nodes in the set to get
245     ↳ average
246     R_bot.append(AvgR_bot) # Adds the total displacement to the U-array by
247     ↳ summing across each step
248     temp = [] # Clear the array for the next iteration in the loop
249
250     Displacements =
251     ↳ fieldObject_U.getSubset(region=odbInstance[part_V].nodeSets['V_R_SET'])
252     for Uyi in Displacements.values: # Loops over each node in the "SET" defined
253     ↳ by the displacement
254         Uyi_vec = [Uyi.data[0], Uyi.data[1], Uyi.data[2]]
255         # Find the magnitude
256         temp.append(np.dot(Uyi_vec, LoadCellDirection)) # Creates a list of
257         ↳ displacements in the "SET"
258
259     SU = np.sum(temp) # Sums up the list of displacements from the "SET"
260     AvgV_top = SU/len(temp) # Divide by the number of nodes in the set to get
261     ↳ average
262     V_top.append(AvgV_top) # Adds the total displacement to the U-array by
263     ↳ summing across each step
264     temp = [] # Clear the array for the next iteration in the loop
265
266     # average difference in nodal positions between the *bonded surfaces
267     Bond_disp.append(AvgR_bot - AvgV_top)
268
269     if BondStatus == True:
270         # if fieldOutput.has_key('CNORMF
271         ↳ ASSEMBLY_R-1_R_V_SURF_BOND/ASSEMBLY_V-1_V_R_SURF_BOND') == 1: # if the
272         ↳ repository has the item

```

```

257 # 'CNORMF ASSEMBLY_CP-R_V/ASSEMBLY_CP-V_R' # This was used when the
    ↳ contact pair was being defined by abaqus. Now that it is defined as
    ↳ a keyword, it is slightly different
258
259 # Contact Force
260 fieldObject_CNORMF_RV = fieldOutput.keys()[0]
261 fieldObject_CNORMF_VR = fieldOutput.keys()[1]
262
263 # Contact Stress
264 fieldObject_CPRESS_RV = fieldOutput.keys()[2]
265 fieldObject_CPRESS_VR = fieldOutput.keys()[3]
266
267 # Contact Shear1
268 fieldObject_CSHEAR1_RV = fieldOutput.keys()[4]
269 fieldObject_CSHEAR1_VR = fieldOutput.keys()[5]
270
271 # Contact Shear2
272 fieldObject_CSHEAR2_RV = fieldOutput.keys()[6]
273 fieldObject_CSHEAR2_VR = fieldOutput.keys()[7]
274
275 # Contact ShearF
276 fieldObject_CSHEARF_RV = fieldOutput.keys()[8]
277 fieldObject_CSHEARF_VR = fieldOutput.keys()[9]
278
279 """ Contact Force (Retina-Vitreous) """
280 # Retina-Vitreous contact stress
281 for CnF_RV_i in fieldOutput[fieldObject_CNORMF_RV].values:
282     temp.append(CnF_RV_i.data*mult)
283
284 S_CnF_RV = np.sum(temp) # Sums up the list of stress from the "SET"
285 CnormF_RV_List.append(temp) # append the list of nodal values
286 CnormF_RV.append(S_CnF_RV) # Adds the total stress to the stress-array by
    ↳ summing across each step
287 temp = [] # Clear the array for the next iteration in the loop
288
289 """ Contact Force (Vitreous-Retina) """
290 # Retina-Vitreous contact stress
291 for CnF_VR_i in fieldOutput[fieldObject_CNORMF_VR].values:
292     temp.append(CnF_VR_i.data*mult)
293
294 S_CnF_VR = np.sum(temp) # Sums up the list of stress from the "SET"
295 CnormF_VR_List.append(temp) # append the list of nodal values
296 CnormF_VR.append(S_CnF_VR) # Adds the total stress to the stress-array by
    ↳ summing across each step
297 temp = [] # Clear the array for the next iteration in the loop
298
299 """ Contact Stress (Retina-Vitreous) """
300 # Retina-Vitreous contact stress
301 for CP_RV_i in fieldOutput[fieldObject_CPRESS_RV].values:
302     temp.append(CP_RV_i.data*mult)
303
304 S_CP_RV = np.sum(temp) # Sums up the list of stress from the "SET"
305 Cpress_RV_List.append(temp)
306 Cpress_RV.append(S_CP_RV) # Adds the total stress to the stress-array by
    ↳ summing across each step
307
    ↳ Cpress_RV_AVG.append(S_CP_RV/len(fieldOutput[fieldObject_CPRESS_RV].values))
308 temp = [] # Clear the array for the next iteration in the loop

```

```

309
310 """ Contact Stress (Vitreous-Retina) """
311 # Retina-Vitreous contact stress
312 for CP_VR_i in fieldOutput[fieldObject_CPRESS_VR].values:
313     temp.append(CP_VR_i.data*mult)
314
315 S_CP_VR = np.sum(temp) # Sums up the list of stress from the "SET"
316 Cpress_VR_List.append(temp)
317 Cpress_VR.append(S_CP_VR) # Adds the total stress to the stress-array by
318     → summing across each step
319
320     → Cpress_VR_AVG.append(S_CP_VR/len(fieldOutput[fieldObject_CPRESS_VR].values))
321 temp = [] # Clear the array for the next iteration in the loop
322
323 """ Contact Shear1 (Retina-Vitreous) """
324 # Retina-Vitreous contact stress
325 for Cs1_RV_i in fieldOutput[fieldObject_CSHEAR1_RV].values:
326     temp.append(Cs1_RV_i.data*mult)
327
328 S_Cs1_RV = np.sum(temp) # Sums up the list of stress from the "SET"
329 Cshear1_RV_List.append(temp) # append the list of nodal values
330 Cshear1_RV.append(S_Cs1_RV) # Adds the total stress to the stress-array
331     → by summing across each step
332 temp = [] # Clear the array for the next iteration in the loop
333
334 """ Contact Shear1 (Vitreous-Retina) """
335 # Retina-Vitreous contact stress
336 for Cs1_VR_i in fieldOutput[fieldObject_CSHEAR1_VR].values:
337     temp.append(Cs1_VR_i.data*mult)
338
339 S_Cs1_VR = np.sum(temp) # Sums up the list of stress from the "SET"
340 Cshear1_VR_List.append(temp) # append the list of nodal values
341 Cshear1_VR.append(S_Cs1_VR) # Adds the total stress to the stress-array
342     → by summing across each step
343 temp = [] # Clear the array for the next iteration in the loop
344
345 """ Contact Shear2 (Retina-Vitreous) """
346 # Retina-Vitreous contact stress
347 for Cs2_RV_i in fieldOutput[fieldObject_CSHEAR2_RV].values:
348     temp.append(Cs2_RV_i.data*mult)
349
350 S_Cs2_RV = np.sum(temp) # Sums up the list of stress from the "SET"
351 Cshear2_RV_List.append(temp) # append the list of nodal values
352 Cshear2_RV.append(S_Cs2_RV) # Adds the total stress to the stress-array
353     → by summing across each step
354 temp = [] # Clear the array for the next iteration in the loop
355
356 """ Contact Shear2 (Vitreous-Retina) """
357 # Retina-Vitreous contact stress
358 for Cs2_VR_i in fieldOutput[fieldObject_CSHEAR2_VR].values:
359     temp.append(Cs2_VR_i.data*mult)
360
361 S_Cs2_VR = np.sum(temp) # Sums up the list of stress from the "SET"
362 Cshear2_VR_List.append(temp) # append the list of nodal values
363 Cshear2_VR.append(S_Cs2_VR) # Adds the total stress to the stress-array
364     → by summing across each step
365 temp = [] # Clear the array for the next iteration in the loop

```

```

361         """ Contact ShearF (Retina-Vitreous) """
362         # Retina-Vitreous contact stress
363         for CsF_RV_i in fieldOutput[fieldObject_CSHEARF_RV].values:
364             temp.append(CsF_RV_i.data*mult)
365
366         S_CsF_RV = np.sum(temp) # Sums up the list of stress from the "SET"
367         CshearF_RV_List.append(temp) # append the list of nodal values
368         CshearF_RV.append(S_CsF_RV) # Adds the total stress to the stress-array
369         → by summing across each step
370         temp = [] # Clear the array for the next iteration in the loop
371
372         """ Contact ShearF (Vitreous-Retina) """
373         # Retina-Vitreous contact stress
374         for CsF_VR_i in fieldOutput[fieldObject_CSHEARF_VR].values:
375             temp.append(CsF_VR_i.data*mult)
376
377         S_CsF_VR = np.sum(temp) # Sums up the list of stress from the "SET"
378         CshearF_VR_List.append(temp) # append the list of nodal values
379         CshearF_VR.append(S_CsF_VR) # Adds the total stress to the stress-array
380         → by summing across each step
381         temp = [] # Clear the array for the next iteration in the loop
382
383     else:
384         # append nans if not available
385         CnormF_RV.append(np.nan)
386         CnormF_VR.append(np.nan)
387         Cpress_RV.append(np.nan)
388         Cpress_RV_AVG.append(np.nan)
389         Cpress_VR.append(np.nan)
390         Cpress_VR_AVG.append(np.nan)
391         Cshear1_RV.append(np.nan)
392         Cshear1_VR.append(np.nan)
393         Cshear2_RV.append(np.nan)
394         Cshear2_VR.append(np.nan)
395         CshearF_RV.append(np.nan)
396         CshearF_VR.append(np.nan)
397
398         CnormF_RV_List.append([np.nan, np.nan, np.nan])
399         CnormF_VR_List.append([np.nan, np.nan, np.nan])
400         Cpress_RV_List.append([np.nan, np.nan, np.nan])
401         Cpress_VR_List.append([np.nan, np.nan, np.nan])
402         Cshear1_RV_List.append([np.nan, np.nan, np.nan])
403         Cshear1_VR_List.append([np.nan, np.nan, np.nan])
404         Cshear2_RV_List.append([np.nan, np.nan, np.nan])
405         Cshear2_VR_List.append([np.nan, np.nan, np.nan])
406         CshearF_RV_List.append([np.nan, np.nan, np.nan])
407         CshearF_VR_List.append([np.nan, np.nan, np.nan])
408
409         print('No CPRESS... ** Updating with NaNs')
410
411     """ Contact Node Lists """
412     R_V_SetNodeNames = []
413     V_R_SetNodeNames = []
414     for i, NodeLabeli in
415         → enumerate(oddbInstance[part_R].nodeSets['R_V_SET'].nodes):
416         R_V_SetNodeNames.append(NodeLabeli.label)

```



```

415 for i, NodeLabeli in
416     → enumerate(oddbInstance[part_V].nodeSets['V_R_SET'].nodes):
417         V_R_SetNodeNames.append(NodeLabeli.label)
418
419 """ Reaction forces """
420 # fieldObject_RF = fieldOutput['RF'] # reaction forces
421 # # Glue-Retina RP set-forces
422 # Reaction_Forces =
423     → fieldObject_RF.getSubset(region=oddbInstance[part_G].nodeSets['G_RP_SET'])
424 # for RFi in Reaction_Forces.values: # Loops over each node in the "SET"
425     → defined by the reaction force
426 # RFi_vec = [RFi.data[0], RFi.data[1], RFi.data[2]]
427 # # Find the component in the direction of the load cell
428 # temp.append(np.dot(RFi_vec, LoadCellDirection)) # Creates a list of
429     → reaction forces in the "SET"
430
431 # SRF = np.sum(temp) # Sums up the list of reaction forces from the "SET"
432 # RF.append(SRF) # Adds the total reaction force to the RF-array by summing
433     → across each step
434 # temp = [] # Clear the array for the next iteration in the loop
435
436 fieldObject_RF = fieldOutput['RF'] # reaction forces
437 if simplified == False:
438     # If Simp is not in the title
439
440     # Glue-Retina G_RP_Set Reaction forces
441     Reaction_Forces =
442         → fieldObject_RF.getSubset(region=oddbInstance[part_G].nodeSets['G_RP_SET'])
443
444 elif simplified == True:
445
446     # Retina R_G_Set Reaction forces
447     Reaction_Forces =
448         → fieldObject_RF.getSubset(region=oddbInstance[part_R].nodeSets['R_G_SET'])
449
450 else:
451     print('Error in RF output')
452
453 for RFi in Reaction_Forces.values: # Loops over each node in the "SET"
454     → defined by the reaction force
455     RFxi = RFi.data[0]
456     RFyi = RFi.data[1]
457     RFzi = RFi.data[2]
458     RFi_vec = [RFxi, RFyi, RFzi]
459
460     # Find the component in the direction of the load cell
461     temp.append(np.dot(RFi_vec, LoadCellDirection)*mult) # Creates a list of
462         → reaction forces in the "SET"
463
464     tempx.append(RFxi) # X reaction forces along the R_G_SET
465     tempy.append(RFyi) # Y reaction forces along the R_G_SET
466     tempz.append(RFzi) # Z reaction forces along the R_G_SET
467
468 SRF = np.sum(temp) # Sums up the list of reaction forces from the "SET"
469 RF.append(SRF) # Adds the total reaction force to the RF-array by summing
470     → across each step
471 temp = [] # Clear the array for the next iteration in the loop
472
473 SRFx = np.sum(tempx)

```

```

463     RFx.append(SRFX)
464
465     SRFY = np.sum(tempy)
466     RFy.append(SRFY)
467
468     SRFZ = np.sum(tempz)
469     RFz.append(SRFZ)
470
471     """ Nodal Forces """
472     # Forces at the nodes of an element from both the hourglass and the regular
473     # deformation modes of that element (negative of the internal forces in
474     # the global coordinate system). The specified position in data and results
475     # file requests is ignored.
476
477     if fieldOutput.has_key('NFORC1') == 1: # Searches if the repository has the
478     ↪ value
479         fieldObject_NFORC1 = fieldOutput['NFORC1'] # Normal force 1
480         fieldObject_NFORC2 = fieldOutput['NFORC2'] # Normal force 2
481         fieldObject_NFORC3 = fieldOutput['NFORC3'] # Normal force 3
482
483         # Retina nodal forces on the glue interface
484         nodeSet_R_G_SET = odbInstance[part_R].nodeSets['R_G_SET']
485         NF1 = fieldObject_NFORC1.getSubset(region=nodeSet_R_G_SET)
486         NF2 = fieldObject_NFORC2.getSubset(region=nodeSet_R_G_SET)
487         NF3 = fieldObject_NFORC3.getSubset(region=nodeSet_R_G_SET)
488
489         for NFi in range(len(NF1.values)): # Loops over each node in the "SET"
490         ↪ defined by the reaction force
491             NFi_vec = [NF1.values[NFi].data, NF2.values[NFi].data,
492             ↪ NF3.values[NFi].data]
493             NFi_veclabel = [NF1.values[NFi].nodeLabel, NF1.values[NFi].data,
494             ↪ NF2.values[NFi].nodeLabel, NF2.values[NFi].data,
495             ↪ NF3.values[NFi].nodeLabel, NF3.values[NFi].data]
496             # Find the component in the direction of the load cell
497             temp.append(np.dot(NFi_vec, LoadCellDirection)*mult) # Creates a list
498             ↪ of reaction forces in the "SET"
499
500         SNf = np.sum(temp) # Sums up the list of reaction forces from the "SET"
501         Nforc.append(SNf*-1) # Adds the total reaction force to the RF-array by
502         ↪ summing across each step (negative indicates the direction, which is
503         ↪ opposite of tension when -1)
504         temp = [] # Clear the array for the next iteration in the loop
505     else:
506         Nforc.append(0)
507         print('No NFORC... ** Updating with 0')
508
509     """ Loop over the history outputs """
510     # odb.steps[disp_step].historyRegions.keys() List all of the items in the
511     ↪ dictionary
512     odbHistoryRegion = odb.steps[disp_step].historyRegions
513     odbHistAssem = 'Assembly ASSEMBLY'
514     Assembly = odbHistoryRegion[odbHistAssem]
515
516     # Energy output
517     ALLIE_KE = Assembly.historyOutputs.keys()[0]
518     Hist_ELEM = Assembly.historyOutputs.keys()[1]
519     Whole_Model_Energy = Assembly.historyOutputs
520     Internal_Energy = Whole_Model_Energy.keys()[0] # Internal energy

```

```

512 Kinetic_Energy = Whole_Model_Energy.keys()[1] # Kintic energy
513 for i, j in enumerate(Whole_Model_Energy[Internal_Energy].data):
514     Hist_Time.append(j[0]) # History Output Time Array
515     IE.append(j[1]) # Internal Energy
516     KE.append(Whole_Model_Energy[Kinetic_Energy].data[i][1]) # Kinetic Energy
517
518 if BondStatus == True:
519     # if jobName.find('VRTie') == -1:
520         # Figure out how to extract these for each node in the connected set
521         """ Bond Loads """
522         # This is an array of bond load per node
523         V_R_SetNodeLength = len(oddbInstance[part_V].nodeSets['V_R_SET'].nodes) #
524         → length of the V_R_Set node list
525         BondNodeNames = odbHistoryRegion.keys()[-V_R_SetNodeLength:]
526
527         # array of bond status and bond load
528         BondStat = []
529         BondLoad = []
530
531         # used for iterating and clearing
532         temp1 = []
533         temp2 = []
534
535         # loop over the length of the BondStat/BondLoad list
536         for m,n in
537             → enumerate(oddbHistoryRegion[BondNodeNames[0]].historyOutputs['BONDSTAT'].data):
538                 # loop over the length of the bond node list and append each time step
539                 for i,BondNodeNames_i in enumerate(BondNodeNames):
540                     → temp1.append(oddbHistoryRegion[BondNodeNames_i].historyOutputs['BONDSTAT'].data[m][1])
541                     → )
542
543                     → temp2.append(oddbHistoryRegion[BondNodeNames_i].historyOutputs['BONDLOAD'].data[m][1]*mu
544
545                 # build the arrays for BondStat/BondLoad
546                 BondStat.append(temp1)
547                 BondLoad.append(temp2)
548
549                 # clear the arrays
550                 temp1 = []
551                 temp2 = []
552         else:
553             print('No bonding,VR interface is tied')
554
555     # # Contact
556     # odbHistElementSetPIBATCH = odbHistoryRegion.keys()[1] # ElementSet PIBATCH
557     # elementSetPIBATCH = odbHistoryRegion[odbHistElementSetPIBATCH]
558     # eC = elementSetPIBATCH.historyOutputs # Element contact
559
560     # if jobName.find('Si') == -1:
561     #     # If Simp is not in the title
562     #     cAreaCP_RG = eC.keys()[0]
563     #     cAreaCP_RV = eC.keys()[1]
564     #     cAreaCP_GR = eC.keys()[2]
565     #     cAreaCP_VR = eC.keys()[3]
566     #     CFN1CP_RG = eC.keys()[4]
567     #     CFN1CP_RV = eC.keys()[5]
568     #     CFN1CP_GR = eC.keys()[6]

```

```

565 #     CFN1CP_VR = eC.keys()[7]
566 #     CFN2CP_RG = eC.keys()[8]
567 #     CFN2CP_RV = eC.keys()[9]
568 #     CFN2CP_GR = eC.keys()[10]
569 #     CFN2CP_VR = eC.keys()[11]
570 #     CFN3CP_RG = eC.keys()[12]
571 #     CFN3CP_RV = eC.keys()[13]
572 #     CFN3CP_GR = eC.keys()[14]
573 #     CFN3CP_VR = eC.keys()[15]
574
575 # elif jobName.find('Si') >= 0:
576 #     # If simplification omit the tab and glue
577 #     cAreaCP_RV = eC.keys()[0]
578 #     cAreaCP_VR = eC.keys()[1]
579 #     CFN1CP_RV = eC.keys()[2]
580 #     CFN1CP_VR = eC.keys()[3]
581 #     CFN2CP_RV = eC.keys()[4]
582 #     CFN2CP_VR = eC.keys()[5]
583 #     CFN3CP_RV = eC.keys()[6]
584 #     CFN3CP_VR = eC.keys()[7]
585 # else:
586 #     print('Error in Hist Output Names')
587 # Bond_Nodes = energyHistRegion.historyOutputs.keys()[2:-1]
588
589 # Glue Reference point
590 if simplified == False:
591     # If Simp is not in the title
592
593     odbHist_gRP = odbHistoryRegion.keys()[1]
594     gRP_Hist = odbHistoryRegion[odbHist_gRP]
595     gRP_Hist = gRP_Hist.historyOutputs
596     gRP_HistRF1 = gRP_Hist.keys()[0]
597     gRP_HistRF2 = gRP_Hist.keys()[1]
598     gRP_HistRF3 = gRP_Hist.keys()[2]
599     gRP_HistU1 = gRP_Hist.keys()[6]
600     gRP_HistU2 = gRP_Hist.keys()[7]
601     gRP_HistU3 = gRP_Hist.keys()[8]
602
603 elif simplified == True:
604     # If simplification, omit the tab and glue
605     print('Simplification')
606 else:
607     print('Error in simplification')
608
609 # for i,j in enumerate(Internal_Energy.data):
610 #     Hist_Time.append(j[0]) # History Output Time Array
611
612 #     # Energy array
613 #     IE.append(j[1]*mult) # Internal Energy
614 #     KE.append(Kinetic_Energy.data[i][1]*mult) # Kinetic Energy
615
616 #     if jobName.find('Si') == -1:
617 #         # If Simp is not in the title
618
619 #         # Contact area arrays, not sure if these need to be multiplied by 2
620 ↪ (Check .ODB)
621 #         CAreaCP_RG.append(eC[cAreaCP_RG].data[i][1])
622 #         CAreaCP_GR.append(eC[cAreaCP_GR].data[i][1])

```

```

622 #         CAreaCP_RV.append(eC[cAreaCP_RV].data[i][1])
623 #         CAreaCP_VR.append(eC[cAreaCP_VR].data[i][1])
624
625 #         # Create a vector for CP RG
626 #         CFNCP_RG_vec = [eC[CFN1CP_RG].data[i][1], eC[CFN2CP_RG].data[i][1],
↪ eC[CFN3CP_RG].data[i][1]]
627 #         # Find the component in the direction of the load cell
628 #         CFNCP_RG.append(np.dot(CFNCP_RG_vec, LoadCellDirection)*mult)
629
630 #         # Create a vector for CP GR
631 #         CFNCP_GR_vec = [eC[CFN1CP_GR].data[i][1], eC[CFN2CP_GR].data[i][1],
↪ eC[CFN3CP_GR].data[i][1]]
632 #         # Find the component in the direction of the load cell
633 #         CFNCP_GR.append(np.dot(CFNCP_GR_vec, LoadCellDirection)*mult)
634
635 #         # Create a vector for CP RV
636 #         CFNCP_RV_vec = [eC[CFN1CP_RV].data[i][1], eC[CFN2CP_RV].data[i][1],
↪ eC[CFN3CP_RV].data[i][1]]
637 #         # Find the component in the direction of the load cell
638 #         CFNCP_RV.append(np.dot(CFNCP_RV_vec, LoadCellDirection)*mult)
639
640 #         # Create a vector for CP VR
641 #         CFNCP_VR_vec = [eC[CFN1CP_VR].data[i][1], eC[CFN2CP_VR].data[i][1],
↪ eC[CFN3CP_VR].data[i][1]]
642 #         # Find the component in the direction of the load cell
643 #         CFNCP_VR.append(np.dot(CFNCP_VR_vec, LoadCellDirection)*mult)
644
645 #         # Create a vector for the Glue Reference point
646 #         Glue_RP_RF_vec = [gRP_Hist[gRP_HistRF1].data[i][1],
↪ gRP_Hist[gRP_HistRF2].data[i][1], gRP_Hist[gRP_HistRF3].data[i][1]]
647 #         # Find the component in the direction of the load cell
648 #         Glue_RP_RF.append(np.dot(Glue_RP_RF_vec, LoadCellDirection)*mult)
649
650 #     elif jobName.find('Si') >= 0:
651 #         # Contact area arrays
652 #         CAreaCP_RG.append(np.nan)
653 #         CAreaCP_GR.append(np.nan)
654 #         CAreaCP_RV.append(eC[cAreaCP_RV].data[i][1])
655 #         CAreaCP_VR.append(eC[cAreaCP_VR].data[i][1])
656
657 #         # Create a vector for CP RG
658 #         CFNCP_RG_vec = [np.nan, np.nan, np.nan]
659 #         # Find the component in the direction of the load cell
660 #         CFNCP_RG.append(np.dot(CFNCP_RG_vec, LoadCellDirection)*mult)
661
662 #         # Create a vector for CP GR
663 #         CFNCP_GR_vec = [np.nan, np.nan, np.nan]
664 #         # Find the component in the direction of the load cell
665 #         CFNCP_GR.append(np.dot(CFNCP_GR_vec, LoadCellDirection)*mult)
666
667 #         # Create a vector for CP RV
668 #         CFNCP_RV_vec = [eC[CFN1CP_RV].data[i][1], eC[CFN2CP_RV].data[i][1],
↪ eC[CFN3CP_RV].data[i][1]]
669 #         # Find the component in the direction of the load cell
670 #         CFNCP_RV.append(np.dot(CFNCP_RV_vec, LoadCellDirection)*mult)
671
672 #         # Create a vector for CP VR

```

```

673 #         CFNCP_VR_vec = [eC[CFN1CP_VR].data[i][1], eC[CFN2CP_VR].data[i][1],
    ↪ eC[CFN3CP_VR].data[i][1]]
674 #         # Find the component in the direction of the load cell
675 #         CFNCP_VR.append(np.dot(CFNCP_VR_vec, LoadCellDirection)*mult)
676
677 #         # Create a vector for the Glue Reference point
678 #         Glue_RP_RF_vec = [np.nan, np.nan, np.nan]
679 #         # Find the component in the direction of the load cell
680 #         Glue_RP_RF.append(np.dot(Glue_RP_RF_vec, LoadCellDirection)*mult)
681 #     else:
682 #         print('Error in hist output data with simplification')
683
684 # if jobName.find('VRTie') == -1: # if VRTie is not in the title
685 #     """ Bond Loads """
686 #     # This is an array of bond load per node
687 #     V_R_SetNodeLength = len(oddbInstance[part_V].nodeSets['V_R_SET'].nodes) #
    ↪ length of the V_R_Set node list
688 #     BondNodeNames = oddbHistoryRegion.keys()[-V_R_SetNodeLength:]
689
690 #     # array of bond status and bond load
691 #     BondStat = []
692 #     BondLoad = []
693
694 #     # used for iterating and clearing
695 #     temp1 = []
696 #     temp2 = []
697
698 #     # loop over the length of the BondStat/BondLoad list
699 #     for m,n in
    ↪ enumerate(oddbHistoryRegion[BondNodeNames[0]].historyOutputs['BONDSTAT'].data):
700 #         # loop over the length of the bond node list and append each time step
701 #         for i,BondNodeNames_i in enumerate(BondNodeNames):
702 #
    ↪ temp1.append(oddbHistoryRegion[BondNodeNames_i].historyOutputs['BONDSTAT'].data[n][1])
703 #
    ↪ temp2.append(oddbHistoryRegion[BondNodeNames_i].historyOutputs['BONDLOAD'].data[n][1])
704
705 #     # build the arrays for BondStat/BondLoad
706 #     BondStat.append(temp1)
707 #     BondLoad.append(temp2)
708
709 #     # clear the arrays
710 #     temp1 = []
711 #     temp2 = []
712 # else:
713 #     print('No bond because the VR interface is tied')
714
715 """ Specify folder name where the files go..."""
716 folderName = jobName
717 folder_sub_directory = 'Output'
718
719 """ Print the oddbFieldOutput Data """
720 print("\nWriting out the load data...")
721 filename = os.path.join(folderName, folder_sub_directory, 'output_Field_'
    + jobName + '.txt')
722
723 outfile = open(filename,'w')
724
725 Header = [] # Header information for the dataframe

```

```

726 Header.append('frame')
727 Header.append('Time [s]')
728 Header.append('Reaction force dotted in y direction [N]')
729 Header.append('Reaction force X [N]')
730 Header.append('Reaction force Y [N]')
731 Header.append('Reaction force Z [N]')
732 Header.append('Sum NForc (Glue-Retina Set) [N]')
733 Header.append('CnormF_RV [N]')
734 Header.append('CnormF_VR [N]')
735 Header.append('Cpress_RV [Pa]')
736 Header.append('Cpress_VR [Pa]')
737 Header.append('AVG_Cpress_RV_AVG [Pa]')
738 Header.append('AVG_Cpress_VR_AVG [Pa]')
739 Header.append('Cshear1_RV [Pa]')
740 Header.append('Cshear1_VR [Pa]')
741 Header.append('Cshear2_RV [Pa]')
742 Header.append('Cshear2_VR [Pa]')
743 Header.append('CshearF_RV [Pa]')
744 Header.append('CshearF_VR [Pa]')
745 Header.append('Glue Displacements [m]')
746 Header.append('Bond Displacements [m]')
747 lineWrite = '\t'.join(str(item) for item in Header)
748 outfile.write(lineWrite)
749
750 for i in frames:
751
752     lineNums = []
753     lineNums.append(time[i])
754     lineNums.append(RF[i])
755     lineNums.append(RFx[i])
756     lineNums.append(RFy[i])
757     lineNums.append(RFz[i])
758     lineNums.append(Nforc[i])
759     lineNums.append(CnormF_RV[i])
760     lineNums.append(CnormF_VR[i])
761     lineNums.append(Cpress_RV[i])
762     lineNums.append(Cpress_VR[i])
763     lineNums.append(Cpress_RV_AVG[i])
764     lineNums.append(Cpress_VR_AVG[i])
765     lineNums.append(Cshear1_RV[i])
766     lineNums.append(Cshear1_VR[i])
767     lineNums.append(Cshear2_RV[i])
768     lineNums.append(Cshear2_VR[i])
769     lineNums.append(CshearF_RV[i])
770     lineNums.append(CshearF_VR[i])
771     lineNums.append(U_top[i])
772     lineNums.append(Bond_disp[i])
773
774     # format the list to have a float with twenty decimal places
775     # Add floats
776     formatted_list = ['{:.20f}'.format(item) for item in lineNums]
777     line = '\n' + '{}\t'.format(i) + '\t'.join(str(item) for item in
778                                                formatted_list)
779     outfile.write(line)
780
781 outfile.close()
782
783 print("\nDone!")

```

```

784 print("\nThe output file will be named '{}'.format(filename) + '"')
785 print("\nIt will be in the same working directory as your Abaqus model\n")
786
787 """ Print the odbHistoryOutput Data """
788 print("\nWriting out the History Output data...")
789 filename = os.path.join(folderName, 'Output', 'output_History_' +
790                          jobName + '.txt')
791 outfile = open(filename, 'w')
792
793 Header = []
794 Header.append('frame')
795 Header.append('Time [s]')
796 Header.append('Internal Energy [J]')
797 Header.append('Kinetic Energy [J]')
798 lineWrite = '\t'.join(str(item) for item in Header)
799 outfile.write(lineWrite)
800
801 for i, j in enumerate(Hist_Time):
802     line = []
803     line.append('{}'.format(i)) # Integer for frame number
804     line.append('{:.10f}'.format(j))
805     line.append('{:.30f}'.format(IE[i]))
806     line.append('{:.30f}'.format(KE[i]))
807     lineWrite = '\n' + '\t'.join(str(item) for item in line)
808     outfile.write(lineWrite)
809
810 outfile.close()
811
812 print("\nDone!")
813 print("\nThe output file will be named '{}'.format(filename) + '"')
814 print("\nIt will be in the same working directory as your Abaqus model\n")
815
816 if BondStatus == True:
817     # if jobName.find('VRTie') == -1: # if VRTie is not in the jobName
818     """ Cube Info Plots """
819     """ Print the odbHistoryOutput BondStat Data """
820     print("\nWriting out the History Output Bond data...")
821     filename = os.path.join(folderName, 'Output', 'BONDSTAT_' + jobName + '.txt')
822     outfile = open(filename, 'w')
823     outfile.write('Time (s)\t' + '\t'.join(str(item) for item in BondNodeNames))
824     for i, j in enumerate(BondStat):
825         outfile.write('\n')
826         tempList = [Hist_Time[i]]
827         for k in list(j):
828             tempList.append(k)
829         outfile.write('\t'.join(str(item) for item in tempList))
830     outfile.close()
831     print("\nDone!")
832     print("\nThe output file will be named '{}'.format(filename) + '"')
833     print("\nIt will be in the same working directory as your Abaqus model\n")
834
835     """ Print the odbHistoryOutput BondLoad Data """
836     print("\nWriting out the History Output Bond data...")
837     filename = os.path.join(folderName, 'Output', 'BONDLOAD_' + jobName + '.txt')
838     outfile = open(filename, 'w')
839     outfile.write('Time (s)\t' + '\t'.join(str(item) for item in BondNodeNames))
840     for i, j in enumerate(BondLoad):
841         outfile.write('\n')

```



```

842         tempList = [Hist_Time[i]]
843         for k in list(j):
844             tempList.append(k)
845         outfile.write('\t'.join(str(item) for item in tempList))
846     outfile.close()
847     print("\nDone!")
848     print("\nThe output file will be named '{}".format(filename) + "'")
849     print("\nIt will be in the same working directory as your Abaqus model\n")
850
851     """ Print the odbFieldOutput CnormF_RV Data """
852     print("\nWriting out the Field Output CnormF_RV data...")
853     filename = os.path.join(folderName, 'Output', 'CnormF_RV_' + jobName +
854                             → '.txt')
854     outfile = open(filename, 'w')
855     xyz = ['X', 'Y', 'Z']
856     header = []
857     for R_V_SetXYZi in list(R_V_SetNodeNames):
858         for m in range(3):
859             header.append('R-' + str(R_V_SetXYZi) + xyz[m])
860     outfile.write('Time (s)\t' + '\t'.join(item for item in header))
861     for i, Nodei in enumerate(CnormF_RV_List):
862         outfile.write('\n')
863         tempList = [time[i]]
864         for XYZ in list(Nodei):
865             for XYZi in list(XYZ):
866                 tempList.append(XYZi)
867             outfile.write('\t'.join(str(item) for item in tempList))
868     outfile.close()
869     print("\nDone!")
870     print("\nThe output file will be named '{}".format(filename) + "'")
871     print("\nIt will be in the same working directory as your Abaqus model\n")
872
873     """ Print the odbFieldOutput CnormF_VR Data """
874     print("\nWriting out the Field Output CnormF_VR data...")
875     filename = os.path.join(folderName, 'Output', 'CnormF_VR_' + jobName +
876                             → '.txt')
876     outfile = open(filename, 'w')
877     xyz = ['X', 'Y', 'Z']
878     header = []
879     for V_R_SetXYZi in list(V_R_SetNodeNames):
880         for m in range(3):
881             header.append('V-' + str(V_R_SetXYZi) + xyz[m])
882     outfile.write('Time (s)\t' + '\t'.join(item for item in header))
883     for i, Nodei in enumerate(CnormF_VR_List):
884         outfile.write('\n')
885         tempList = [time[i]]
886         for XYZ in list(Nodei):
887             for XYZi in list(XYZ):
888                 tempList.append(XYZi)
889             outfile.write('\t'.join(str(item) for item in tempList))
890     outfile.close()
891     print("\nDone!")
892     print("\nThe output file will be named '{}".format(filename) + "'")
893     print("\nIt will be in the same working directory as your Abaqus model\n")
894
895     """ Print the odbFieldOutput Cpress_RV Data """
896     print("\nWriting out the Field Output Cpress_RV data...")

```

```

897 filename = os.path.join(folderName, 'Output', 'Cpress_RV_' + jobName +
898     ↳ '.txt')
899 outfile = open(filename, 'w')
900 header = []
901 for R_V_SetXYZi in list(R_V_SetNodeNames):
902     header.append('R-' + str(R_V_SetXYZi))
903 outfile.write('Time (s)\t' + '\t'.join(item for item in header))
904 for i,Nodei in enumerate(Cpress_RV_List):
905     outfile.write('\n')
906     tempList = [time[i]]
907     for ni in list(Nodei):
908         tempList.append(ni)
909     outfile.write('\t'.join(str(item) for item in tempList))
910 outfile.close()
911 print("\nDone!")
912 print("\nThe output file will be named '{}".format(filename) + "'")
913 print("\nIt will be in the same working directory as your Abaqus model\n")
914
915 """ Print the odbFieldOutput Cpress_VR Data """
916 print("\nWriting out the Field Output Cpress_VR data...")
917 filename = os.path.join(folderName, 'Output', 'Cpress_VR_' + jobName +
918     ↳ '.txt')
919 outfile = open(filename, 'w')
920 header = []
921 for V_R_SetXYZi in list(V_R_SetNodeNames):
922     header.append('V-' + str(V_R_SetXYZi))
923 outfile.write('Time (s)\t' + '\t'.join(item for item in header))
924 for i,Nodei in enumerate(Cshear1_RV_List):
925     outfile.write('\n')
926     tempList = [time[i]]
927     for ni in list(Nodei):
928         tempList.append(ni)
929     outfile.write('\t'.join(str(item) for item in tempList))
930 outfile.close()
931 print("\nDone!")
932 print("\nThe output file will be named '{}".format(filename) + "'")
933 print("\nIt will be in the same working directory as your Abaqus model\n")
934
935 """ Print the odbFieldOutput Cshear1_RV Data """
936 print("\nWriting out the Field Output Cshear1_RV data...")
937 filename = os.path.join(folderName, 'Output', 'Cshear1_RV_' + jobName +
938     ↳ '.txt')
939 outfile = open(filename, 'w')
940 header = []
941 for R_V_SetXYZi in list(R_V_SetNodeNames):
942     header.append('R-' + str(R_V_SetXYZi))
943 outfile.write('Time (s)\t' + '\t'.join(item for item in header))
944 for i,Nodei in enumerate(Cshear1_RV_List):
945     outfile.write('\n')
946     tempList = [time[i]]
947     for ni in list(Nodei):
948         tempList.append(ni)
949     outfile.write('\t'.join(str(item) for item in tempList))
950 outfile.close()
951 print("\nDone!")
952 print("\nThe output file will be named '{}".format(filename) + "'")
953 print("\nIt will be in the same working directory as your Abaqus model\n")

```

```

952     """ Print the odbFieldOutput Cshear1_VR Data """
953     print("\nWriting out the Field Output Cshear1_VR data...")
954     filename = os.path.join(folderName, 'Output', 'Cshear1_VR_' + jobName +
955         ↪ '.txt')
956     outfile = open(filename, 'w')
957     header = []
958     for V_R_SetXYZi in list(V_R_SetNodeNames):
959         header.append('V-' + str(V_R_SetXYZi))
960     outfile.write('Time (s)\t' + '\t'.join(item for item in header))
961     for i, Nodei in enumerate(Cshear1_VR_List):
962         outfile.write('\n')
963         tempList = [time[i]]
964         for ni in list(Nodei):
965             tempList.append(ni)
966         outfile.write('\t'.join(str(item) for item in tempList))
967     outfile.close()
968     print("\nDone!")
969     print("\nThe output file will be named '{}".format(filename) + "'")
970     print("\nIt will be in the same working directory as your Abaqus model\n")
971
972     """ Print the odbFieldOutput Cshear2_RV Data """
973     print("\nWriting out the Field Output Cshear2_RV data...")
974     filename = os.path.join(folderName, 'Output', 'Cshear2_RV_' + jobName +
975         ↪ '.txt')
976     outfile = open(filename, 'w')
977     header = []
978     for R_V_SetXYZi in list(R_V_SetNodeNames):
979         header.append('R-' + str(R_V_SetXYZi))
980     outfile.write('Time (s)\t' + '\t'.join(item for item in header))
981     for i, Nodei in enumerate(Cshear2_RV_List):
982         outfile.write('\n')
983         tempList = [time[i]]
984         for ni in list(Nodei):
985             tempList.append(ni)
986         outfile.write('\t'.join(str(item) for item in tempList))
987     outfile.close()
988     print("\nDone!")
989     print("\nThe output file will be named '{}".format(filename) + "'")
990     print("\nIt will be in the same working directory as your Abaqus model\n")
991
992     """ Print the odbFieldOutput Cshear2_VR Data """
993     print("\nWriting out the Field Output Cshear2_VR data...")
994     filename = os.path.join(folderName, 'Output', 'Cshear2_VR_' + jobName +
995         ↪ '.txt')
996     outfile = open(filename, 'w')
997     header = []
998     for V_R_SetXYZi in list(V_R_SetNodeNames):
999         header.append('V-' + str(V_R_SetXYZi))
1000     outfile.write('Time (s)\t' + '\t'.join(item for item in header))
1001     for i, Nodei in enumerate(Cshear2_VR_List):
1002         outfile.write('\n')
1003         tempList = [time[i]]
1004         for ni in list(Nodei):
1005             tempList.append(ni)
1006         outfile.write('\t'.join(str(item) for item in tempList))
1007     outfile.close()
1008     print("\nDone!")
1009     print("\nThe output file will be named '{}".format(filename) + "'")

```

```

1007     print("\nIt will be in the same working directory as your Abaqus model\n")
1008
1009     """ Print the odbFieldOutput CshearF_RV Data """
1010     print("\nWriting out the Field Output CshearF_RV data...")
1011     filename = os.path.join(folderName, 'Output', 'CshearF_RV_' + jobName +
1012                             ↪ '.txt')
1012     outfile = open(filename, 'w')
1013     xyz = ['X', 'Y', 'Z']
1014     header = []
1015     for R_V_SetXYZi in list(R_V_SetNodeNames):
1016         for m in range(3):
1017             header.append('R-' + str(R_V_SetXYZi) + xyz[m])
1018     outfile.write('Time (s)\t' + '\t'.join(item for item in header))
1019     for i, Nodei in enumerate(CshearF_RV_List):
1020         outfile.write('\n')
1021         tempList = [time[i]]
1022         for XYZ in list(Nodei):
1023             for XYZi in list(XYZ):
1024                 tempList.append(XYZi)
1025             outfile.write('\t'.join(str(item) for item in tempList))
1026     outfile.close()
1027     print("\nDone!")
1028     print("\nThe output file will be named '{}".format(filename) + "'")
1029     print("\nIt will be in the same working directory as your Abaqus model\n")
1030
1031     """ Print the odbFieldOutput CshearF_VR Data """
1032     print("\nWriting out the Field Output CshearF_VR data...")
1033     filename = os.path.join(folderName, 'Output', 'CshearF_VR_' + jobName +
1034                             ↪ '.txt')
1034     outfile = open(filename, 'w')
1035     xyz = ['X', 'Y', 'Z']
1036     header = []
1037     for V_R_SetXYZi in list(V_R_SetNodeNames):
1038         for m in range(3):
1039             header.append('V-' + str(V_R_SetXYZi) + xyz[m])
1040     outfile.write('Time (s)\t' + '\t'.join(item for item in header))
1041     for i, Nodei in enumerate(CshearF_VR_List):
1042         outfile.write('\n')
1043         tempList = [time[i]]
1044         for XYZ in list(Nodei):
1045             for XYZi in list(XYZ):
1046                 tempList.append(XYZi)
1047             outfile.write('\t'.join(str(item) for item in tempList))
1048     outfile.close()
1049     print("\nDone!")
1050     print("\nThe output file will be named '{}".format(filename) + "'")
1051     print("\nIt will be in the same working directory as your Abaqus model\n")
1052
1053     else:
1054         print('No bonding because the VR Interface is tied')
1055     return
1056
1057 # Run the function
1058 data_extract(jobName)

```

## 1.5.5 Plotting Script

```
</> Script 8: Python script used to create plots for each simulation. </>
1 # -*- coding: utf-8 -*-
2 """
3 Created on Fri Apr 03 11:58:40 2020
4
5 @author: Kiffer Creveling
6 python3
7 """
8 # Packages & path folder
9 #from sys import argv, exit
10 #sys.path.append(r'F:\Abaqus Working Directory')
11 import pandas as pd
12 import matplotlib.pyplot as plt
13 from matplotlib.pyplot import cm
14 import matplotlib.patheffects as pe
15 import numpy as np
16 import os
17 import os.path
18 import sys
19 import pdb
20 plt.rcParams['figure.figsize'] = [16, 9]
21
22 # jobName = sys.argv[1] # Extract the jobName from the previous script
23
24 #if __name__ == '__main__':
25 def plot_Field_Output(fileName, dataDirectory, dataCompare, BondStatus, PDFMStatus):
26
27     """ Field Output Data """
28     df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)
29
30     Header = [] # Header information for the dataframe
31     Header.append('Frame')
32     Header.append('Time')
33     Header.append('RF_y_dot')
34     Header.append('RFx')
35     Header.append('RFy')
36     Header.append('RFz')
37     Header.append('Nodal_Force')
38     Header.append('CnormF_RV')
39     Header.append('CnormF_VR')
40     Header.append('Cpress_RV')
41     Header.append('Cpress_VR')
42     Header.append('AVG_Cpress_RV_AVG')
43     Header.append('AVG_Cpress_VR_AVG')
44     Header.append('Cshear1_RV')
45     Header.append('Cshear1_VR')
46     Header.append('Cshear2_RV')
47     Header.append('Cshear2_VR')
48     Header.append('CshearF_RV')
49     Header.append('CshearF_VR')
50     Header.append('Glue_Displacements')
51     Header.append('Bond_Displacements')
52
53     df.columns = Header
```

```

54
55 t = df.Time
56 RF = df.RF_y_dot*1e3 # Convert from N to mN
57 NF = df.Nodal_Force*1e3 # Convert from N to mN
58 CnF_RV = df.CnormF_RV*1e3 # Convert from N to mN
59 CnF_VR = df.CnormF_VR*1e3 # Convert from N to mN
60 Cp_RV = df.Cpress_RV
61 Cp_VR = df.Cpress_VR
62 AVG_Cp_RV = df.AVG_Cpress_RV_AVG
63 AVG_Cp_VR = df.AVG_Cpress_VR_AVG
64 Cs1_RV = df.Cshear1_RV*1e3 # Convert from N to mN
65 Cs1_VR = df.Cshear1_VR*1e3 # Convert from N to mN
66 Cs2_RV = df.Cshear2_RV*1e3 # Convert from N to mN
67 Cs2_VR = df.Cshear2_VR*1e3 # Convert from N to mN
68 CsF_RV = df.CshearF_RV*1e3 # Convert from N to mN
69 CsF_VR = df.CshearF_VR*1e3 # Convert from N to mN
70 TD = df.Glue_Displacements*1e3 # Convert from m to mm
71 BD = df.Bond_Displacements*1e3 # Convert from m to mm
72
73 tabArea = 0.00002247 # m^2 (Used in the hand calc - not used anymore)
74
75 (figureName, ext) = os.path.splitext(fileName) # Split the file extension
76
77 """ Read in the csv file """
78 dfValsn = pd.read_csv(dataCompare, sep="\t", nrows=22, header=None,
79 ↪ names=['Var', 'Attribute'])
80
81 """ File Attributes """
82 HID = dfValsn['Attribute'][0]
83 HAGE = dfValsn['Attribute'][1]
84 HG = dfValsn['Attribute'][2]
85 HLR = dfValsn['Attribute'][3]
86 HR = dfValsn['Attribute'][4]
87 HSSi = float(dfValsn['Attribute'][12])
88 HSSf = float(dfValsn['Attribute'][13])
89 HTMax = float(dfValsn['Attribute'][14])
90 HDispMax = float(dfValsn['Attribute'][15])
91 HFMax = float(dfValsn['Attribute'][16]) # (mN)
92 HFSS = float(dfValsn['Attribute'][17])
93 HSlope20 = float(dfValsn['Attribute'][20]) # (mN/m) slope from 20 seconds prior
94 ↪ to max force value
95
96 dfn = pd.read_csv(os.path.join(dataCompare), sep="\t", header=30)
97 dfn.columns = ['Time', 'Extension', 'Force']
98 dfn_time = dfn.Time
99 dfn_extension = dfn.Extension # mm
100 dfn_force = dfn.Force*1e3 # convert from N to mN
101
102 # SS Array
103 ssTimeArray = np.array([HSSi, HSSf])
104 ssValArray = np.array([HFSS, HFSS])
105
106 # Max peel force displacement at max and steady state
107 dfn_max_Displ = dfn_extension[dfn_time == HTMax]
108 dfn_ss_Displ = np.array([dfn_extension[dfn_time == HSSi].values[0],
109 ↪ dfn_extension[dfn_time == HSSf].values[0]])

```

```

108 # Plot the data trace to compare the simulated results with the force
    ↪ displacement curves
109 plt.plot(dfn_extension, dfn_force, '-', color='r', linewidth=1, markersize=2,
    ↪ label = '{} , Age: {}'.format(HID, HAGE))
110 if str(HFMax) == 'nan' and str(HSSi) == 'nan':
111     print('No max or steady state')
112     pass
113
114 if str(HFMax) != 'nan':
115     plt.plot(dfn_max_Dis, HFMax, '.', color='k', linewidth=1, markersize=20,
    ↪ label = 'Max Peel - {:.4f} (mN)'.format(HFMax),
    ↪ path_effects=[pe.Stroke(linewidth=4, foreground='k'), pe.Normal()])
116
117 if str(HSSi) != 'nan':
118     plt.plot(dfn_ss_Dis, ssValArray, '-', color='c', linewidth=3, markersize=2,
    ↪ label = 'Steady State - {:.4f} (mN)'.format(HFSS),
    ↪ path_effects=[pe.Stroke(linewidth=5, foreground='k'), pe.Normal()])
119
120 """ Plots """
121 ##### Plot Data #####
122 plt.plot(TD, RF, '-', color='blue', linewidth=2, markersize=2, label = r'Simulated
    ↪ Reaction force  $\Sigma F_{\text{Retina}}$ ')
123 plt.xlabel('Displacement (mm)', fontsize=18)
124 plt.ylabel('Force (mN)', fontsize=18)
125 plt.title('Vitreous', fontsize=20)
126 plt.grid()
127 plt.legend(loc = 'best', fontsize = 'medium')
128 plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
    ↪ '_RF_vs_Dis.png'), dpi=300, bbox_inches='tight') # Save figure
129 plt.close()
130
131 # Plot the data trace to compare the simulated results
132 plt.plot(dfn_time, dfn_force, '-', color='r', linewidth=1, markersize=2, label =
    ↪ '{} , Age: {}'.format(HID, HAGE))
133 if str(HFMax) == 'nan' and str(HSSi) == 'nan':
134     print('No max or steady state')
135     pass
136
137 if str(HFMax) != 'nan':
138     plt.plot(HTMax, HFMax, '.', color='k', linewidth=1, markersize=20, label =
    ↪ 'Max Peel - {:.4f} (mN)'.format(HFMax),
    ↪ path_effects=[pe.Stroke(linewidth=4, foreground='k'), pe.Normal()])
139
140 if str(HSSi) != 'nan':
141     plt.plot(ssTimeArray, ssValArray, '-', color='c', linewidth=3, markersize=2,
    ↪ label = 'Steady State - {:.4f} (mN)'.format(HFSS),
    ↪ path_effects=[pe.Stroke(linewidth=5, foreground='k'), pe.Normal()])
142
143 """ Plots """
144 ##### Plot Data #####
145 plt.plot(t, RF, '-', color='blue', linewidth=2, markersize=2, label = r'Simulated
    ↪ Reaction force  $\Sigma F_{\text{Retina}}$ ')
146 plt.xlabel('Time (sec)', fontsize=18)
147 plt.ylabel('Force (mN)', fontsize=18)
148 plt.title('Vitreous', fontsize=20)
149 plt.grid()
150 plt.legend(loc = 'best', fontsize = 'medium')

```

```

151 plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
    ↳ '_RF_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
152 plt.close()
153
154 """ Sum Nodal Force Reaction force """
155 ##### Plot Data #####
156 plt.plot(t, NF, '-', color='blue', linewidth=2, markersize=2, label = 'Reaction
    ↳ force')
157 plt.xlabel('Time (sec)', fontsize=18)
158 plt.ylabel('Force (mN)', fontsize=18)
159 plt.title('Vitreous', fontsize=20)
160 plt.grid()
161 plt.legend(loc = 'best', fontsize = 'medium')
162 plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
    ↳ '_NForce_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
163 plt.close()
164
165 """ Compare sum RF vs Nforce """
166 ##### Plot Data #####
167 plt.plot(t, RF, '-', color='blue', linewidth=2, markersize=2, label = 'RF')
168 plt.plot(t, NF, ':', color='red', linewidth=2, markersize=2, label = 'NFORC')
169 plt.xlabel('Time (sec)', fontsize=18)
170 plt.ylabel('Force (mN)', fontsize=18)
171 plt.title('Vitreous', fontsize=20)
172 plt.grid()
173 plt.legend(loc = 'best', fontsize = 'medium')
174 plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
    ↳ '_RF_vs_NForce_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
175 plt.close()
176
177 """ Plot bond disp """
178 ##### Plot Data #####
179 plt.plot(t, BD, '-', color='blue', linewidth=2, markersize=2, label = 'Bond - Disp')
180 plt.plot(t, TD, '-', color='red', linewidth=2, markersize=2, label = 'Top - Disp')
181 plt.xlabel('Time (sec)', fontsize=18)
182 plt.ylabel('Bond Disp (mm)', fontsize=18)
183 plt.title('Vitreous', fontsize=20)
184 plt.grid()
185 plt.legend(loc = 'best', fontsize = 'medium')
186 plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
    ↳ '_disp_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
187 plt.close()
188
189 """ Contact Nforce """
190 ##### Plot Data #####
191 plt.plot(t, CnF_RV, '-', color='red', linewidth=2, markersize=2, label =
    ↳ r'CnormF$_{RV}$')
192 plt.plot(t, CnF_VR, ':', color='blue', linewidth=2, markersize=2, label =
    ↳ r'CnormF$_{VR}$')
193 plt.xlabel('Time (sec)', fontsize=18)
194 plt.ylabel('Force (mN)', fontsize=18)
195 plt.title('Contact Normal Force', fontsize=20)
196 plt.grid()
197 plt.legend(loc = 'best', fontsize = 'medium')
198 plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
    ↳ 'CnormF_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
199 plt.close()
200

```



```

201 """ Contact Cpress """
202 ##### Plot Data #####
203 plt.plot(t, Cp_RV, '-', color='red', linewidth=2, markersize=2, label =
    ↪ r'Cpress$_{RV}$')
204 plt.plot(t, Cp_VR, ':', color='blue', linewidth=2, markersize=2, label =
    ↪ r'Cpress$_{VR}$')
205 plt.xlabel('Time (sec)', fontsize=18)
206 plt.ylabel('Pressure (Pa)', fontsize=18)
207 plt.title('Contact pressure', fontsize=20)
208 plt.grid()
209 plt.legend(loc = 'best', fontsize = 'medium')
210 plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
    ↪ 'Cpress_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
211 plt.close()
212
213 """ Contact AVG_Cpress """
214 ##### Plot Data #####
215 plt.plot(t, AVG_Cp_RV, '-', color='red', linewidth=2, markersize=2, label = r'AVG
    ↪ Cpress$_{RV}$')
216 plt.plot(t, AVG_Cp_VR, ':', color='blue', linewidth=2, markersize=2, label = r'AVG
    ↪ Cpress$_{VR}$')
217 plt.xlabel('Time (sec)', fontsize=18)
218 plt.ylabel('Pressure (Pa)', fontsize=18)
219 plt.title('Contact pressure Average', fontsize=20)
220 plt.grid()
221 plt.legend(loc = 'best', fontsize = 'medium')
222 plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
    ↪ 'AVG_Cpress_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
223 plt.close()
224
225 """ Contact Cshear1 """
226 ##### Plot Data #####
227 plt.plot(t, Cs1_RV, '-', color='red', linewidth=2, markersize=2, label =
    ↪ r'Cshear$_{RV}$^1$')
228 plt.plot(t, Cs1_VR, ':', color='blue', linewidth=2, markersize=2, label =
    ↪ r'Cshear$_{VR}$^1$')
229 plt.xlabel('Time (sec)', fontsize=18)
230 plt.ylabel('Shear Force (mN)', fontsize=18)
231 plt.title('Contact shear 1 force', fontsize=20)
232 plt.grid()
233 plt.legend(loc = 'best', fontsize = 'medium')
234 plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
    ↪ 'Cshear1_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
235 plt.close()
236
237 """ Contact Cshear2 """
238 ##### Plot Data #####
239 plt.plot(t, Cs2_RV, '-', color='red', linewidth=2, markersize=2, label =
    ↪ r'Cshear$_{RV}$^2$')
240 plt.plot(t, Cs2_VR, ':', color='blue', linewidth=2, markersize=2, label =
    ↪ r'Cshear$_{VR}$^2$')
241 plt.xlabel('Time (sec)', fontsize=18)
242 plt.ylabel('Shear Force (mN)', fontsize=18)
243 plt.title('Contact shear 2 force', fontsize=20)
244 plt.grid()
245 plt.legend(loc = 'best', fontsize = 'medium')
246 plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
    ↪ 'Cshear2_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure

```

```

247 plt.close()
248
249 """ Contact CshearF """
250 ##### Plot Data #####
251 plt.plot(t, CsF_RV, '-',color='red',linewidth=2,markersize=2,label =
↳ r'Cshear$_{RV}$~F$')
252 plt.plot(t, CsF_VR, ':',color='blue',linewidth=2,markersize=2,label =
↳ r'Cshear$_{VR}$~F$')
253 plt.xlabel('Time (sec)',fontsize=18)
254 plt.ylabel('Shear Force (mN)',fontsize=18)
255 plt.title('Contact shear F force',fontsize=20)
256 plt.grid()
257 plt.legend(loc = 'best',fontsize = 'medium')
258 plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
↳ 'CshearF_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
259 plt.close()
260
261 def plot_History_Output(fileName, dataDirectory):
262 """ History Output Data """
263 df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)
264 df.columns = ["Frame","Time","Internal_Energy","Kinetic_Energy"]
265
266 t_h = df.Time
267 IE = df.Internal_Energy
268 KE = df.Kinetic_Energy
269
270 (figureName, ext) = os.path.splitext(fileName) # Split the file extension
271
272 """ Plots History Outputs """
273 ##### Plot Data #####
274 plt.plot(t_h, IE, '-',color='blue',linewidth=2,markersize=2,label = 'Internal
↳ Energy')
275 plt.plot(t_h, KE, '-',color='red',linewidth=2,markersize=2,label = 'Kinetic
↳ Energy')
276 plt.xlabel('Time (sec)',fontsize=18)
277 plt.ylabel('Energy (J)',fontsize=18)
278 plt.title('Energy',fontsize=20)
279 plt.grid()
280 plt.legend(loc = 'best',fontsize = 'medium')
281 plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
↳ '_Energy_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
282 plt.close()
283
284 ##### Plot Data #####
285 plt.semilogy(t_h, KE/IE, '-',color='blue',linewidth=2,markersize=2,label = r'Ratio
↳ $\frac{KE}{IE}$')
286 plt.semilogy(t_h,
↳ 0.1*np.ones(len(t_h)), '-',color='red',linewidth=2,markersize=2,label = '10%')
287 plt.xlabel('Time (sec)',fontsize=18)
288 plt.ylabel('Ratio of KE to IE',fontsize=18)
289 plt.title('Energy ratio',fontsize=20)
290 plt.grid()
291 plt.legend(loc = 'best',fontsize = 'medium')
292 plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
↳ '_Ratio_KE_IE_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
293 plt.close()
294
295 # ##### Plot Data #####

```

```

296 # plt.plot(t_h, gRP_RF, '-', color='blue', linewidth=2, markersize=2, label =
    ↳ r'G$_{RP}$')
297 # plt.xlabel('Time (sec)', fontsize=18)
298 # plt.ylabel('Reaction Force (mN)', fontsize=18)
299 # plt.title('Glue Reference Point History Output', fontsize=20)
300 # plt.grid()
301 # plt.legend(loc = 'best', fontsize = 'medium')
302 # plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
    ↳ '_Glue_RP_RF.png'), dpi=300, bbox_inches='tight') # Save figure
303 # plt.close()
304
305 # ##### Plot Data #####
306 # plt.plot(t_h, CFNCP_RG*1e3, '-', color='red', linewidth=2, markersize=2, label =
    ↳ r'CFNCP$_{RG}$')
307 # plt.plot(t_h, CFNCP_GR*1e3, ':', color='blue', linewidth=2, markersize=2, label =
    ↳ r'CFNCP$_{GR}$')
308 # plt.xlabel('Time (sec)', fontsize=18)
309 # plt.ylabel('Reaction Force (mN)', fontsize=18)
310 # plt.title('Contact Force CFNCP_RG/GR History Output', fontsize=20)
311 # plt.grid()
312 # plt.legend(loc = 'best', fontsize = 'medium')
313 # plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
    ↳ '_CFNCP_RG_GR_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
314 # plt.close()
315
316 # ##### Plot Data #####
317 # plt.plot(t_h, CFNCP_RV*1e3, '-', color='red', linewidth=2, markersize=2, label =
    ↳ r'CFNCP$_{RV}$')
318 # plt.plot(t_h, CFNCP_VR*1e3, ':', color='blue', linewidth=2, markersize=2, label =
    ↳ r'CFNCP$_{VR}$')
319 # plt.xlabel('Time (sec)', fontsize=18)
320 # plt.ylabel('Reaction Force (mN)', fontsize=18)
321 # plt.title('Contact Force CFNCP_RV/GR History Output', fontsize=20)
322 # plt.grid()
323 # plt.legend(loc = 'best', fontsize = 'medium')
324 # plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
    ↳ '_CFNCP_RV_VR_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
325 # plt.close()
326
327 # ##### Plot Data #####
328 # plt.plot(t_h, CAreaCP_RG, '-', color='red', linewidth=2, markersize=2, label =
    ↳ r'CAreaCP$_{RG}$')
329 # plt.plot(t_h, CAreaCP_GR, ':', color='blue', linewidth=2, markersize=2, label =
    ↳ r'CAreaCP$_{GR}$')
330 # plt.xlabel('Time (sec)', fontsize=18)
331 # plt.ylabel(r'CArea ($m^2$)', fontsize=18)
332 # plt.title('Contact Area CAreaCP_RG/GR History Output', fontsize=20)
333 # plt.grid()
334 # plt.legend(loc = 'best', fontsize = 'medium')
335 # plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
    ↳ 'CAreaCP_RG_GR_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
336 # plt.close()
337
338 # ##### Plot Data #####
339 # plt.plot(t_h, CAreaCP_RV*1e3, '-', color='red', linewidth=2, markersize=2, label =
    ↳ r'CAreaCP$_{RV}$')
340 # plt.plot(t_h, CAreaCP_VR*1e3, ':', color='blue', linewidth=2, markersize=2, label =
    ↳ r'CAreaCP$_{VR}$')

```

```

341 # plt.xlabel('Time (sec)',fontsize=18)
342 # plt.ylabel(r'CArea ($m^2$)',fontsize=18)
343 # plt.title('Contact Area CAreaCP_RV/VR History Output',fontsize=20)
344 # plt.grid()
345 # plt.legend(loc = 'best',fontsize = 'medium')
346 # plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
    ↳ 'CAreaCP_RV_VR_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
347 # plt.close()
348
349 print("Plots will be in the figures folder")
350
351 def plotFieldHist(FieldfileName, HistfileName, dataDirectory, dataCompare, BondStatus,
    ↳ PDFMStatus):
352     """ Field Output Data """
353     df1 = pd.read_csv(os.path.join(dataDirectory,FieldfileName), sep="\t", header=0)
354
355     Header = [] # Header information for the dataframe
356     Header.append('Frame')
357     Header.append('Time')
358     Header.append('RF_y_dot')
359     Header.append('RFx')
360     Header.append('RFy')
361     Header.append('RFz')
362     Header.append('Nodal_Force')
363     Header.append('CnormF_RV')
364     Header.append('CnormF_VR')
365     Header.append('Cpress_RV')
366     Header.append('Cpress_VR')
367     Header.append('AVG_Cpress_RV_AVG')
368     Header.append('AVG_Cpress_VR_AVG')
369     Header.append('Cshear1_RV')
370     Header.append('Cshear1_VR')
371     Header.append('Cshear2_RV')
372     Header.append('Cshear2_VR')
373     Header.append('CshearF_RV')
374     Header.append('CshearF_VR')
375     Header.append('Glue_Displacements')
376     Header.append('Bond_Displacements')
377
378     df1.columns = Header
379
380     t = df1.Time
381     RF = df1.RF_y_dot*1e3 # Convert from N to mN
382     NF = df1.Nodal_Force*1e3 # Convert from N to mN
383     CnF_RV = df1.CnormF_RV
384     CnF_VR = df1.CnormF_VR
385     Cp_RV = df1.Cpress_RV
386     Cp_VR = df1.Cpress_VR
387     AVG_Cp_RV = df1.AVG_Cpress_RV_AVG
388     AVG_Cp_VR = df1.AVG_Cpress_VR_AVG
389     Cs1_RV = df1.Cshear1_RV
390     Cs1_VR = df1.Cshear1_VR
391     Cs2_RV = df1.Cshear2_RV
392     Cs2_VR = df1.Cshear2_VR
393     CsF_RV = df1.CshearF_RV
394     CsF_VR = df1.CshearF_VR
395     TD = df1.Glue_Displacements*1e3 # Convert from m to mm
396     BD = df1.Bond_Displacements*1e3 # Convert from m to mm

```

```

397
398 (figureName, ext) = os.path.splitext(FieldfileName) # Split the file extension
399
400 """ History Output Data """
401 df2 = pd.read_csv(os.path.join(dataDirectory, HistfileName), sep="\t", header=0,
402 ↪ index_col=False)
403
404 df2.columns = ["Frame", "Time", "Internal_Energy", "Kinetic_Energy"]
405
406 t_h = df2.Time
407 IE = df2.Internal_Energy
408 KE = df2.Kinetic_Energy
409
410 (figureName, ext) = os.path.splitext(HistfileName) # Split the file extension
411
412 # ##### Plot Data #####
413 # plt.plot(t, RF, '-', color='blue', linewidth=2, markersize=2, label = r'Field
414 ↪ $G^{\text{RP}}$ RF')
415 # plt.plot(t, NF, '--', color='red', linewidth=2, markersize=2, label = 'Field NFORC
416 ↪ R')
417 # plt.plot(t_h, gRP_RF, ':', color='black', linewidth=2, markersize=2, label = r'Hist
418 ↪ $G^{\text{RP}}$ RF')
419 # plt.xlabel('Time (sec)', fontsize=18)
420 # plt.ylabel('Force (mN)', fontsize=18)
421 # plt.title('Reaction Force Compare Field to History Outputs', fontsize=20)
422 # plt.grid()
423 # plt.legend(loc = 'best', fontsize = 'medium')
424 # plt.savefig(os.path.join(dataDirectory, 'Figures/' +
425 ↪ 'CompareFieldtoHist_RF.png'), dpi=300, bbox_inches='tight') # Save figure
426 # plt.close()
427
428 def plot_BondStat_Output(fileName, dataDirectory):
429 """ BondStat Output Data """
430 df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)
431
432 t = df['Time (s)']
433
434 # determine the length of the number of bonded nodes
435 # linspace from 0 to 1 by the number of nodes for the y-position
436 # Loop over the number of bonded nodes and plot the y-th position vs time with
437 ↪ the color of the bond load on a single plot
438
439 fig1, ax1 = plt.subplots()
440 nRows = np.shape(df)[0]
441 nCols = np.shape(df)[1] - 1 # subtract the time column
442 y = np.linspace(0,1,nCols)
443 count = 0
444 for (colName, colData) in df.iteritems():
445     if colName.find('Time') == -1:
446         """ Plots History Outputs """
447         ##### Plot Data #####
448         sc = ax1.scatter(t, np.ones(nRows)*y[count], c=colData, cmap=cm.cool, s=5,
449 ↪ edgecolors='none', vmin=0, vmax=1)
450         count += 1 # update the counter
451     else:
452         continue
453
454 # plt.gray() # turns image to grayscale

```

```

448 plt.colorbar(sc)
449 ax1.set_xlabel('Time (sec)',fontsize=18)
450 ax1.set_ylabel('Bonded Nodes',fontsize=18)
451 ax1.set_title('BONDSTAT (Color indicates status)',fontsize=20)
452 (figureName, ext) = os.path.splitext(fileName) # Split the file extension
453 fig1.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
454     ↳ '_BONDSTAT_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
454 plt.close()
455
456 print("Plots will be in the figures folder")
457
458 def plot_BondLoad_Output(fileName, dataDirectory):
459     """ BondLoad Output Data """
460     df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)
461
462     t = df['Time (s)']
463
464     # determine the length of the number of bonded nodes
465     # linspace from 0 to 1 by the number of nodes for the y-position
466     # Loop over the number of bonded nodes and plot the y-th position vs time with
467     ↳ the color of the bond load on a single plot
468
469     fig1, ax1 = plt.subplots()
470     nRows = np.shape(df)[0]
471     nCols = np.shape(df)[1] - 1 # subtract the time column
472     y = np.linspace(0,1,nCols)
473     count = 0
474     for (colName, colData) in df.iteritems():
475         if colName.find('Time') == -1:
476             """ Plots History Outputs """
477             ##### Plot Data #####
478             sc = ax1.scatter(t, np.ones(nRows)*y[count], c=colData, cmap=cm.cool, s=5,
479                 ↳ edgecolors='none', vmin=0, vmax=1)
480             count += 1 # update the counter
481         else:
482             continue
483
484     # plt.gray() # turns image to grayscale
485     plt.colorbar(sc)
486     ax1.set_xlabel('Time (sec)',fontsize=18)
487     ax1.set_ylabel('Bonded Nodes',fontsize=18)
488     ax1.set_title('BONDLOAD (Color indicates status)',fontsize=20)
489     (figureName, ext) = os.path.splitext(fileName) # Split the file extension
490     fig1.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
491         ↳ '_BONDLOAD_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
492     plt.close()
493
494     print("Plots will be in the figures folder")
495
496 def PlotAbqData(fileName, dataDirectory, dataCompare, BondStatus, PDFMStatus):
497     # """ Change directory to correct path """
498     # filePath = os.getcwd()
499     # data_directory = os.path.join(filePath,jobName)
500     # figures_directory = os.path.join(filePath,jobName,'Figures')
501     # if not os.path.exists(figures_directory):
502     #     os.makedirs(figures_directory)

```

```

502     """ Call both functions to plot Field/History data """
503     field_files = [f for f in os.listdir(dataDirectory) if
504     ↪ os.path.isfile(os.path.join(dataDirectory, f)) and
505     ↪ f.startswith('output_Field')]
506     for fname in field_files:
507         plot_Field_Output(fname, dataDirectory, dataCompare, BondStatus, PDFMStatus)
508
509     history_files = [f for f in os.listdir(dataDirectory) if
510     ↪ os.path.isfile(os.path.join(dataDirectory, f)) and
511     ↪ f.startswith('output_History')]
512     for hname in history_files:
513         plot_History_Output(hname, dataDirectory)
514
515     for fname in field_files:
516         plotFieldHist(fname, hname, dataDirectory, dataCompare, BondStatus,
517         ↪ PDFMStatus)
518
519     if BondStatus == True:
520         BONDSTAT_files = [f for f in os.listdir(dataDirectory) if
521         ↪ os.path.isfile(os.path.join(dataDirectory, f)) and
522         ↪ f.startswith('BONDSTAT')]
523         for bname in BONDSTAT_files:
524             plot_BondStat_Output(bname, dataDirectory)
525
526         BONDLOAD_files = [f for f in os.listdir(dataDirectory) if
527         ↪ os.path.isfile(os.path.join(dataDirectory, f)) and
528         ↪ f.startswith('BONDLOAD')]
529         for bname in BONDLOAD_files:
530             plot_BondLoad_Output(bname, dataDirectory)

```

## 1.5.6 Residual Script For Optimization

</> **Script 9: Python script used to calculate the residual for the objective function** </>  
*used in the optimization routine.*

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Sat Nov 7 17:27:47 2020
4
5  @author: Kiffer2
6
7  """
8  import numpy as np
9  import pandas as pd
10 from scipy import interpolate
11 import matplotlib.pyplot as plt
12 from matplotlib.pyplot import cm
13 import matplotlib.path_effects as pe
14 import os
15 import os.path
16 import sys
17 import pdb
18
19 def Least_Squares(x, y):
20     """

```

```

21     Calculate the slope and y-intercept using matrix math
22     x & y are the coordinates of points
23
24     parameters (X,Y) Data
25
26     Returns:
27         Curve fit data and parameters  $m \cdot x + b$ , R squared value
28     """
29     Z = np.ones((len(x),2))
30     Z[:,1] = x
31     # Calculate the matrix inverse for the constants of the regression
32     A = np.dot(np.linalg.inv(np.dot(Z.T,Z)),(np.dot(Z.T,y)))
33     linFit = x*A[1] + A[0]
34
35     # Stats
36     SS_tot = np.sum((y - np.mean(y))**2)
37     SS_res = np.sum((y - linFit)**2)
38     Rsqd = 1 - SS_res/SS_tot
39
40     return linFit, A, Rsqd
41
42
43 def residualFcn(fileName, dataDirectory, maxForceTime, dataCompare, objErr,
44                 slopeFlag, maxForceFlag, ssForceFlag, timeBeforePeak):
45     """
46     Parameters
47     -----
48     fileName : Output txt file with the odb data
49     dataDirectory : Location of the output file
50
51     Returns
52     -----
53     Maximum force from the txt file
54     """
55
56     # In[Simulated data]
57     df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)
58
59     Header = [] # Header information for the dataframe
60     Header.append('Frame')
61     Header.append('Time')
62     Header.append('RF_y_dot')
63     Header.append('RFx')
64     Header.append('RFy')
65     Header.append('RFz')
66     Header.append('Nodal_Force')
67     Header.append('CnormF_RV')
68     Header.append('CnormF_VR')
69     Header.append('Cpress_RV')
70     Header.append('Cpress_VR')
71     Header.append('AVG_Cpress_RV_AVG')
72     Header.append('AVG_Cpress_VR_AVG')
73     Header.append('Cshear1_RV')
74     Header.append('Cshear1_VR')
75     Header.append('Cshear2_RV')
76     Header.append('Cshear2_VR')
77     Header.append('CshearF_RV')
78     Header.append('CshearF_VR')

```



```

79 Header.append('Retina_Glue_Top')
80 Header.append('Bond_Displacements')
81
82 df.columns = Header
83
84 tt = df.Time
85 RF = df.RF_y_dot*1e3 # Convert from N to mN
86 NF = df.Nodal_Force*1e3 # Convert from N to mN
87 CnF_RV = df.CnormF_RV*1e3 # Convert from N to mN
88 CnF_VR = df.CnormF_VR*1e3 # Convert from N to mN
89 Cp_RV = df.Cpress_RV
90 Cp_VR = df.Cpress_VR
91 AVG_Cp_RV = df.AVG_Cpress_RV_AVG
92 AVG_Cp_VR = df.AVG_Cpress_VR_AVG
93 Cs1_RV = df.Cshear1_RV*1e3 # Convert from N to mN
94 Cs1_VR = df.Cshear1_VR*1e3 # Convert from N to mN
95 Cs2_RV = df.Cshear2_RV*1e3 # Convert from N to mN
96 Cs2_VR = df.Cshear2_VR*1e3 # Convert from N to mN
97 CsF_RV = df.CshearF_RV*1e3 # Convert from N to mN
98 CsF_VR = df.CshearF_VR*1e3 # Convert from N to mN
99 dn = df.Retina_Glue_Top*1e3 # Convert from m to mm
100 BD = df.Bond_Displacements*1e3 # Convert from m to mm
101
102 # maybe try to output the maximum force at a specific time
103 specificTime = maxForceTime
104 actualTime = min(df['Time'], key=lambda x:abs(x - specificTime))
105 force_at_time = RF[df['Time'] == actualTime].values[0]
106
107 # In[Experimental data]
108 """ Read in the csv file """
109 dfValsn = pd.read_csv(os.path.join(dataCompare), sep="\t", nrows=29,
110                       header=None, names=['Var', 'Attribute'])
111
112 """ File Attributes """
113 HID = dfValsn['Attribute'][0]
114 HAGE = dfValsn['Attribute'][1]
115 HG = dfValsn['Attribute'][2]
116 HLR = dfValsn['Attribute'][3]
117 HR = dfValsn['Attribute'][4]
118 HSSi = float(dfValsn['Attribute'][12])
119 HSSf = float(dfValsn['Attribute'][13])
120 HTMax = float(dfValsn['Attribute'][14])
121 HDispMax = float(dfValsn['Attribute'][15])
122 HFMax = float(dfValsn['Attribute'][16]) # (mN)
123 HFSS = float(dfValsn['Attribute'][17]) # (mN)
124 # slope from 20 seconds prior to max force value
125 HSlope20 = float(dfValsn['Attribute'][20]) # (mN/m)
126
127 dfn = pd.read_csv(os.path.join(dataCompare), sep="\t", header=30)
128 dfn.columns = ['Time', 'Extension', 'Force']
129 dfn_time = dfn.Time
130 dfn_extension = dfn.Extension # mm
131 dfn_force = dfn.Force*1e3 # N ---> mN
132
133 # if fileName.find('sym') >= 0:
134 #     # divide all data trace values by 2
135 #     dfn_force = dfn_force/2
136 #     HFMax = HFMax/2

```

```

137     # HFSS = HFSS/2
138
139     # SS Array
140     ssTimeArray = np.array([HSSi, HSSf])
141     ssValArray = np.array([HFSS, HFSS])
142
143     # In[Experimental data isolate linear region up to peak]
144
145     # slope calculation for 20 seconds prior to the max peel force
146     # (Experimental Data)
147     maxIndex = dfn_time[dfn_time == HTMax].index.values[0]
148
149     # Convert to data array length
150     timeBeforePeak = timeBeforePeak*10
151
152     # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec) to location of max
153     ↪ force
154     x_n = dfn_extension[maxIndex - timeBeforePeak:maxIndex]
155     y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
156     # Perform least squares
157     curveFit_n, Params_n, R_Squared_n = Least_Squares(x_n, y)
158
159     # Shift extension data so that the linear region is extrapolated
160     # through the origin
161     shift_disp = abs(Params_n[0]/Params_n[1])
162     if Params_n[0] > 0:
163         dfn_extension_shift = dfn_extension + shift_disp
164
165     if min(dfn_extension_shift) > 0:
166         # Add zero to prevent mishaps with interpolation
167         dfn_extension_shift = [0] + dfn_extension_shift
168     else:
169         dfn_extension_shift = dfn_extension - shift_disp
170
171     # Now that the data has been shifted, recalculate the linear regression
172     # using the reduced data set
173     # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec) to location of max
174     ↪ force
175     x_n = dfn_extension_shift[maxIndex - timeBeforePeak:maxIndex]
176     y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
177     # Perform least squares
178     curveFit_n, Params_n, R_Squared_n = Least_Squares(x_n,y)
179
180     # Slope of the curve up to the max force !!!(from the simulated data)!!!
181     # find the closest simulated displacement to the experimental
182     # max displacement
183     # adjustDisp = min(dn, key=lambda x:abs(x - dfn_extension[maxIndex]))
184     # index = RF[dfn == adjustDisp].index.values[0] # index determination
185     # Index where the max reaction force is in the array
186     simMaxIndex = RF.idxmax()
187     simMaxForce = RF.max() # maximum simulated force value
188     simMaxDisp = dn[RF == simMaxForce] # displacement at the max force value
189
190     # If the max index is the second data point add one to it (Difficulty in
191     # selecting the pandas series value) to select the first two values in the
192     # pandas array it needs to be RF[0:2] instead of RF[0:1] but the index

```

```

192     # value of the max force is 1. Try to fix this issue
193     if simMaxIndex == 1:
194         simMaxIndex += 1
195
196     x = dn[0:simMaxIndex] # Array from 0 to location of max force/n
197     y = RF[0:simMaxIndex] # Array from 0 to location of max force/n
198     # Perform least squares
199     curveFit, Params, R_Squared = Least_Squares(x,y)
200
201     # Updated force at specific max disp with adjusted value (Simulated data)
202     specificTime = maxForceTime
203     actualDisp = min(dn, key=lambda x:abs(x - dfn_extension_shift[maxIndex]))
204     force_at_Displacement = RF[dn == actualDisp].values[0]
205
206     # Max peel force displacement at max and steady state
207     dfn_max_Displacement = dfn_extension_shift[dfn_time == HTMax]
208     dfn_ss_Displacement = [dfn_extension_shift[dfn_time == HSSi].values[0],
209                            dfn_extension_shift[dfn_time == HSSf].values[0]] # flatten()
210
211     """ Simulated Steady State calculation """
212     if simMaxIndex == len(RF):
213         simMaxGreaterIndex = len(RF) - 1
214     else:
215         # return the mean and median of the points after the peak force value
216         # This will always round down
217         simMaxGreaterIndex = int(simMaxIndex + (len(RF) - simMaxIndex)*(31/64))
218
219     # Steady state values from the max force index half way to the end
220     # Force values after the peak force
221     RF_SteadyState = RF[simMaxGreaterIndex:]
222     # Displacement values after the peak force
223     dn_SteadyState = dn[simMaxGreaterIndex:]
224
225     SSMean = np.mean(RF_SteadyState) # Mean
226     SSMedian = np.median(RF_SteadyState) # Median
227
228     # In[Plots]
229     """ Plots """
230     # Plot the experimental, simulated, and curve fit data
231
232     # Split the file extension
233     (figureName, ext) = os.path.splitext(fileName)
234
235     # Plot the data trace to compare the simulated results with the force
236     # displacement curves
237     plt.plot(dfn_extension_shift, dfn_force, '-', color='r', linewidth=1,
238             markersize=2, label = '{} , Age: {}'.format(HID, HAGE),
239             alpha = 0.5)
240
241     if str(HFMax) == 'nan' and str(HSSi) == 'nan':
242         print('No max or steady state')
243         pass
244
245     if str(HFMax) != 'nan':
246         plt.plot(dfn_max_Displacement, HFMax, '.', color='k', linewidth=1,
247                 markersize=20,
248                 label = 'Max Peel - {:.4f} (mN)'.format(HFMax),
249                 path_effects=[pe.Stroke(linewidth=4, foreground='k')],

```

```

250         pe.Normal())
251     plt.plot(x_n, curveFit_n, '-', color='tab:blue', linewidth=2,
252             label=r'Curve fit Max - {} (s)'.format(timeBeforePeak/10) +
253             'y = {:.4f}x + '.format(Params_n[1]) +
254             '{:.4f} (mN)', '.format(Params_n[0]) +
255             '$r^2$ = {:.4f}'.format(R_Squared_n), alpha = 1)
256
257     if str(HSSi) != 'nan':
258         plt.plot(dfn_ss_Displacement, ssValArray, '-', color='c', linewidth=3,
259                 markersize=2,
260                 label = 'Steady State - {:.4f} (mN)'.format(HFSS),
261                 path_effects=[pe.Stroke(linewidth=5, foreground='k'),
262                                pe.Normal()])
263
264     # Plot the simulated data
265     plt.plot(dn, RF, '-', color='blue', linewidth=2, markersize=2,
266             label = r'Simulated Reaction force $\Sigma F_{Retina}$')
267     plt.plot(x, curveFit, '-', color='tab:green', linewidth=2, markersize=2,
268             label= 'y = {:.4f}x + '.format(Params[1]) +
269             '{:.4f} (mN)', '.format(Params[0]) +
270             '$r^2$ = {:.4f}'.format(R_Squared))
271     plt.plot(simMaxDisp, simMaxForce, '.', color='tab:red', linewidth=1,
272             markersize = 20,
273             label = 'Simulated maximum Force {:.4f} (mN)'.format(simMaxForce))
274     plt.plot(dn_SteadyState, np.ones(len(RF_SteadyState))*SSMean, '-',
275             color='tab:gray', label = 'Simulated steady state force ' +
276             '{:.4f} (mN)'.format(np.mean(RF_SteadyState)))
277
278     # In[Error Calculation]
279     # error between slope, force, and steady-state value
280
281     maxSlopeMeasured = Params_n[1] # Experimental slope
282     maxSlopeSimulated = Params[1] # Simulated slope
283     maxForceMeasured = HFMax # Experimental max force
284     maxForceSimulated = simMaxForce # Simulated max force
285     SS_Measured = HFSS # Experimental SS force
286     SSmeanSimulated = SSMean # Simulated SS force (mean)
287     SSmedianSimulated = SSMedian # Simulated SS force (median)
288
289     # Error calculation
290     errorDict = {} # Dictionary
291     if objErr == 'Difference':
292         errorDict['slope'] = (maxSlopeMeasured - maxSlopeSimulated) if slopeFlag
293         → == True else []
294         errorDict['maxForce'] = (maxForceMeasured - maxForceSimulated) if
295         → maxForceFlag == True else []
296         errorDict['ssForce'] = (SS_Measured - SSmeanSimulated) if ssForceFlag
297         → == True else []
298     elif objErr == 'Ratio':
299         errorDict['slope'] = (1 - maxSlopeMeasured / maxSlopeSimulated) if
300         → slopeFlag == True else []
301         errorDict['maxForce'] = (1 - maxForceMeasured / maxForceSimulated) if
302         → maxForceFlag == True else []
303         errorDict['ssForce'] = (1 - SS_Measured / SSmeanSimulated) if
304         → ssForceFlag == True else []
305     elif objErr == 'Relative uncertainty':
306         errorDict['slope'] = ((maxSlopeMeasured -
307         → maxSlopeSimulated)/maxSlopeMeasured) if slopeFlag == True else []

```

```

301     errorDict['maxForce'] = ((maxForceMeasured -
302         ↪ maxForceSimulated)/maxForceMeasured) if maxForceFlag == True else []
302     errorDict['ssForce'] = ((SS_Measured - SSmedianSimulated)/SS_Measured)
303         ↪ if ssForceFlag == True else []
303 else:
304     print('Error in MaxForceError')
305     sys.exit()
306
307     # Error array values
308     errorList = list(errorDict.values()) # convert to list
309     errorList = [x for x in errorList if x] # get rid of empty values
310
311     # String for the error array
312     errorString = ', '.join('{:.4f}'.format(i) for i in errorList)
313
314     plt.plot([dfn_max_Disp, simMaxDisp], [HFMax, simMaxForce], '--',
315         linewidth = 1, color = 'magenta', label = r'Difference ' +
316         'between simulated & experiment max force: ' +
317         '{:.4f}'.format(HFMax - np.max(RF)))
318
319     # Plot the different conditions if they are to be compared
320     if slopeFlag == True:
321         plt.plot([], [], 'white', label = r'{} '.format(objErr) +
322             'between slopes is: ' +
323             '{:.4f}'.format(errorDict['slope']))
324
325     if maxForceFlag == True:
326         plt.plot([], [], 'white', label = r'{} '.format(objErr) +
327             'between max force is: ' +
328             '{:.4f}'.format(errorDict['maxForce']))
329
330     if ssForceFlag == True:
331         plt.plot([], [], 'white', label = r'{} '.format(objErr) +
332             'between steady state is: ' +
333             '{:.4f}'.format(errorDict['ssForce']))
334
335     plt.plot([], [], 'white',
336         label = r'Objective error array: [' + errorString + ']')
337     plt.plot([], [], 'white', label = r'Error $L^2$ Norm: ' +
338         '{:.4f}'.format(np.sqrt(np.dot(errorList, errorList))))
339
340     ##### Plot Data #####
341     plt.axhline(0, color='black')
342     plt.axvline(0, color='black')
343     plt.ylabel('Force (mN)',fontsize=18)
344     plt.xlabel('Distance (mm)',fontsize=18)
345     plt.title('Simulation vs. Experimental Data Trace',fontsize=20)
346     plt.grid()
347     plt.legend(loc = 'best',fontsize = 'medium')
348     plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
349         '_SlopeCompare.pdf'), dpi=300,
350         bbox_inches='tight')
351     plt.close()
352
353     # In[Calculate interpolated Experimental and Simulated data]
354
355     # slope calculation for 20 seconds prior to the max peel force
356     # (Experimental Data)

```

```

357 maxIndex = dfn_time[dfn_time == HTMax].index.values[0]
358 # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec) to location of max
    ↪ force
359 t_n = dfn_time[maxIndex - timeBeforePeak:maxIndex]
360 y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
361 # Perform least squares and return
362 curveFit_n, Params_n_time, R_Squared_n = Least_Squares(t_n, y)
363
364 # Shift extension data so that the linear region is extrapolated
365 # through the origin
366 shift_time = abs(Params_n_time[0]/Params_n_time[1])
367
368 # shift time data for visual purposes
369 if Params_n_time[0] > 0:
370     dfn_time_shift = dfn_time + shift_time
371
372     if min(dfn_time_shift) > 0:
373         # Add zero to prevent mishaps with interpolation
374         dfn_time_shift = [0] + dfn_time_shift
375 else:
376     dfn_time_shift = dfn_time - shift_time
377
378 # x array for the linear region leading up to the peak force
379 Fmax_t_shift = dfn_time_shift[maxIndex]
380 fit_t = np.linspace(0, Fmax_t_shift, 200) # Selected value
381 # fit_t = np.linspace(0, dfn_time_shift[np.argmax(dfn_force)], 200) # true max
382 Fmax_x_shift = dfn_extension_shift[maxIndex]
383 # fit_x = np.linspace(0, dfn_extension_shift[np.argmax(dfn_force)], 200) # true
    ↪ max
384 fit_x = np.linspace(0, Fmax_x_shift, 200) # Selected value
385
386 # create the linear region leading up to the peak force
387 def fit(params, x):
388     b, m = params
389     return m*x + b
390 fit_vals_y_time = fit(Params_n_time, fit_t)
391 fit_vals_y_force = fit(Params_n, fit_x)
392
393 # Trim the shifted experimental data to be greater than zero
394 t_exp = dfn_time_shift[dfn_time_shift >= 0]
395 x_exp = dfn_extension_shift[dfn_time_shift >= 0]
396 y_exp = dfn_force[dfn_time_shift >= 0]
397
398 # data frame with original data only shifted
399 dfdata = pd.DataFrame(np.array([t_exp, x_exp, y_exp]).T,
400                       columns=['t', 'x', 'y'])
401
402 # Select time beyond the max time to the end of the data
403 t_geq_max = dfn_time_shift[maxIndex:]
404 x_geq_max = dfn_extension_shift[maxIndex:]
405 y_geq_max = dfn_force[maxIndex:]
406
407 # dataframe of data points from the max value to the end
408 dfgmax = pd.DataFrame(np.array([t_geq_max, x_geq_max, y_geq_max]).T,
409                       columns=['t', 'x', 'y'])
410
411 # data frame of points from zero to the max value
412 linArray = np.array([fit_t, fit_x, fit_vals_y_force])

```

```

413 dfLin = pd.DataFrame(linArray.T, columns=['t', 'x', 'y'])
414
415 # create the new data frame of linear points up to the peak and all points
416 # beyond
417 dfNew = dfLin.append(dfgmax, ignore_index=True)
418
419 # Interpolate the experimental data
420 n_data_pts = 100
421 start_point_time = tt[RF.argmax()] # Time at the peak (simulated)
422 start_point_disp = dn[RF.argmax()] # Disp at the peak (simulated)
423 f_exp_time = interpolate.interp1d(dfNew['t'], dfNew['y'])
424 f_exp_disp = interpolate.interp1d(dfNew['x'], dfNew['y'])
425 t_new_exp = np.linspace(start_point_time, tt[tt.argmax()],
426                          n_data_pts) # (s)
427 x_new_exp = np.linspace(start_point_disp, dn[tt.argmax()],
428                          n_data_pts) # (mm)
429 y_new_exp_time = f_exp_time(t_new_exp) # Interpolate `interp1d`
430 y_new_exp_disp = f_exp_disp(x_new_exp) # Interpolate `interp1d`
431
432 # In[Interpolated Simulated Trace]
433
434 # Interpolate the simulated data
435 f_sim_time = interpolate.interp1d(tt, RF)
436 f_sim_disp = interpolate.interp1d(dn, RF)
437 t_new_sim = np.linspace(start_point_time, tt[tt.argmax()],
438                          n_data_pts) # (s)
439 x_new_sim = np.linspace(start_point_disp, dn[tt.argmax()],
440                          n_data_pts) # (mm)
441 y_new_sim_time = f_sim_time(t_new_sim) # Interpolate `interp1d`
442 y_new_sim_disp = f_sim_disp(x_new_sim) # Interpolate `interp1d`
443
444 # In[Plots]
445 ''' Time curve '''
446 fit, ax = plt.subplots()
447 ax.plot()
448 ax.plot(dfdata['t'], dfdata['y'], label='Original Shifted Data',
449         alpha = 0.5)
450 ax.plot(dfNew['t'], dfNew['y'], label='Merged Data',
451         alpha = 0.5)
452 ax.plot(t_new_exp, y_new_exp_time, '--', label='Interp Experimental Data')
453 ax.plot(tt, RF, label='Simulated Data')
454 ax.plot(t_new_sim, y_new_sim_time, ':', label='Interp Simulated Data')
455 ax.axhline(color='k')
456 ax.set_xlim([0, 300])
457 ax.set_xlabel('Time (s)', fontsize=14)
458 ax.set_ylabel('Force (N)', fontsize=14)
459 ax.legend(loc='best', fontsize=14)
460 ax.grid('on')
461 plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
462                          '_Interp_Time.pdf'), dpi=300,
463            bbox_inches='tight')
464 plt.close()
465
466 ''' Displacement curve '''
467 fit, ax = plt.subplots()
468 ax.plot()
469 ax.plot(dfdata['x'], dfdata['y'], label='Original Shifted Data',
470         alpha = 0.5)

```

```

471 ax.plot(dfNew['x'], dfNew['y'], label='Merged Data',
472         alpha = 0.5)
473 ax.plot(x_new_exp, y_new_exp_disp, '--', label='Interp Experimental Data')
474 ax.plot(dn, RF, label='Simulated Data')
475 ax.plot(x_new_sim, y_new_sim_disp, ':', label='Interp Simulated Data')
476 ax.axhline(color='k')
477 ax.set_xlim([0, max(dn)])
478 ax.set_xlabel('Displacement (mm)', fontsize=14)
479 ax.set_ylabel('Force (N)', fontsize=14)
480 ax.legend(loc='best', fontsize=14)
481 ax.grid('on')
482 plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
483                          '_Interp_Displ.pdf'), dpi=300,
484            bbox_inches='tight')
485 plt.close()
486
487 ''' Displacement curve only showing interpolated data '''
488 residual = y_new_exp_disp - y_new_sim_disp # residual calculation
489 L2Norm = np.sqrt(np.dot(residual, residual))
490
491 fit, ax = plt.subplots()
492 ax.plot()
493 ax.plot(x_new_exp, y_new_exp_disp, '-', label='Interp Experimental Data')
494 ax.plot(x_new_sim, y_new_sim_disp, '-', label='Interp Simulated Data')
495 ax.plot(x_new_sim, residual, ':', label=r'Residual = $(exp - sim)$',
496         alpha = 0.8)
497 ax.plot([], [], color='white', label=r'$L^2$ norm = {:.4f}'.format(L2Norm))
498 ax.axhline(color='k', linewidth=0.25)
499 ax.set_xlim([0, max(x_new_exp)])
500 ax.set_xlabel('Displacement (mm)', fontsize=14)
501 ax.set_ylabel('Force (N)', fontsize=14)
502 ax.legend(loc='best', fontsize=14)
503 ax.grid('on')
504 plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
505                          '_Interp_Displ_Clean.pdf'), dpi=300,
506            bbox_inches='tight')
507 plt.close()
508
509 returnList = [Params[1], simMaxForce, SSMean, SSMedian, y_new_exp_disp,
510              y_new_sim_disp]
511 return returnList
512
513 # In[Function that calls the nested function to compute the residual]
514 def findResidual(fileName, dataDirectory, maxForceTime, dataCompare, objErr,
515                 slopeFlag, maxForceFlag, ssForceFlag, timeBeforePeak):
516     """
517     Parameters
518     -----
519     fileName : Output txt file with the odb data
520     dataDirectory : Location of the output file
521
522     Returns
523     -----
524     maximumForce : Maximum force from the txt file
525     """
526
527     global residual
528     """ Call function to return max displacement """

```



```

529     ModelParamsFile = [f for f in os.listdir(dataDirectory)
530                         if os.path.isfile(os.path.join(dataDirectory, f))
531                         and f.startswith('output_Field')]
532     for mpFile in ModelParamsFile:
533         residual = residualFcn(mpFile, dataDirectory, maxForceTime,
534                               dataCompare, objErr, slopeFlag, maxForceFlag,
535                               ssForceFlag, timeBeforePeak)
536
537     return residual

```

## 1.5.7 Max Force Script

</> **Script 10:** Python script used to determine the max force for the bond model. </>

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Sun Jan 17 23:56:35 2021
4
5  @author: Kiffer2
6  """
7
8  import numpy as np
9  import pandas as pd
10 import matplotlib.pyplot as plt
11 from matplotlib.pyplot import cm
12 import matplotlib.path as pe
13 import os
14 import os.path
15 import sys
16 import pdb
17
18 def Least_Squares(x,y):
19     """
20     Calculate the slope and y-intercept using matrix math
21     x & y are the coordinates of points
22
23     parameters (X,Y) Data
24
25     Returns:
26     Curve fit data and parameters m*x + b, R squared value
27     """
28     Z = np.ones((len(x),2))
29     Z[:,1] = x
30     A = np.dot(np.linalg.inv(np.dot(Z.T,Z)),(np.dot(Z.T,y))) # Calculate the matrix
31     ↪ inverse for the constants of the regression
32     linFit = x*A[1] + A[0]
33
34     # Stats
35     SS_tot = np.sum((y - np.mean(y))**2)
36     SS_res = np.sum((y - linFit)**2)
37     Rsqd = 1 - SS_res/SS_tot
38
39     return linFit, A, Rsqd
40
41 def maxForce(fileName, dataDirectory, maxForceTime, dataCompare):
42     """

```

```

42 Parameters
43 -----
44 fileName : Output txt file with the odb data
45 dataDirectory : Location of the output file
46
47 Returns
48 -----
49 Maximum force from the txt file
50 """
51 df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)
52
53 Header = [] # Header information for the dataframe
54 Header.append('Frame')
55 Header.append('Time')
56 Header.append('RF_y_dot')
57 Header.append('RFx')
58 Header.append('RFy')
59 Header.append('RFz')
60 Header.append('Nodal_Force')
61 Header.append('CnormF_RV')
62 Header.append('CnormF_VR')
63 Header.append('Cpress_RV')
64 Header.append('Cpress_VR')
65 Header.append('AVG_Cpress_RV_AVG')
66 Header.append('AVG_Cpress_VR_AVG')
67 Header.append('Cshear1_RV')
68 Header.append('Cshear1_VR')
69 Header.append('Cshear2_RV')
70 Header.append('Cshear2_VR')
71 Header.append('CshearF_RV')
72 Header.append('CshearF_VR')
73 Header.append('Glue_Displacements')
74 Header.append('Bond_Displacements')
75
76 df.columns = Header
77
78 RF = df.RF_y_dot*1e3 # N to mN
79 dn = df.Glue_Displacements*1e3 # m to mm
80
81 # maybe try to output the maximum force at a specific time
82 specificTime = maxForceTime
83 actualTime = min(df['Time'], key=lambda x:abs(x - specificTime))
84 force_at_time = RF[df['Time'] == actualTime].values[0]
85
86 # Plot the experimental, simulated, and curve fit data
87
88 (figureName, ext) = os.path.splitext(fileName) # Split the file extension
89
90 """ Read in the csv file """
91 dfValsn = pd.read_csv(os.path.join(dataCompare), sep="\t", nrows=29, header=None,
92 ↪ names=['Var', 'Attribute'])
93
94 """ File Attributes """
95 HID = dfValsn['Attribute'][0]
96 HAGE = dfValsn['Attribute'][1]
97 HG = dfValsn['Attribute'][2]
98 HLR = dfValsn['Attribute'][3]
99 HR = dfValsn['Attribute'][4]

```

```

99 HSSi = float(dfValsn['Attribute'][12])
100 HSSf = float(dfValsn['Attribute'][13])
101 HTMax = float(dfValsn['Attribute'][14])
102 HDispMax = float(dfValsn['Attribute'][15])
103 HFMax = float(dfValsn['Attribute'][16]) # (mN)
104 HFSS = float(dfValsn['Attribute'][17])
105 HSlope20 = float(dfValsn['Attribute'][20]) # (mN/m) slope from 20 seconds prior
    ↳ to max force value
106
107 dfn = pd.read_csv(os.path.join(dataCompare), sep="\t", header=30)
108 dfn.columns = ['Time', 'Extension', 'Force']
109 dfn_time = dfn.Time
110 dfn_extension = dfn.Extension # mm
111 dfn_force = dfn.Force*1e3 # N ---> mN
112
113 # SS Array
114 ssTimeArray = np.array([HSSi, HSSf])
115 ssValArray = np.array([HFSS, HFSS])
116
117 # slope calculation for 20 seconds prior to the max peel force (Experimental
    ↳ Data)
118 maxIndex = dfn_time[dfn_time == HTMax].index.values[0]
119 x20 = dfn_extension[maxIndex-200:maxIndex] # Array from maxIndex - 200 (20 sec)
    ↳ to location of max force
120 y = dfn_force[maxIndex-200:maxIndex] # Array from maxIndex - 200 (20 sec) to
    ↳ location of max force
121 curveFit20, Params20, R_Squared20 = Least_Squares(x20,y) # Perform least squares
    ↳ and return
122
123 # Shift extension data so that the linear region is extrapolated through the
    ↳ origin
124 shift = abs(Params20[0]/Params20[1])
125 dfn_extension = dfn_extension - shift
126
127 # Now that the data has been shifted, recalculate the linear regression using the
    ↳ reduced data set
128 x20 = dfn_extension[maxIndex-200:maxIndex] # Array from maxIndex - 200 (20 sec)
    ↳ to location of max force
129 y = dfn_force[maxIndex-200:maxIndex] # Array from maxIndex - 200 (20 sec) to
    ↳ location of max force
130 curveFit20, Params20, R_Squared20 = Least_Squares(x20,y) # Perform least squares
    ↳ and return
131
132 # Slope of the curve up to the max force !!!(from the simulated data)!!!
133 adjustDisp = min(dn, key=lambda x:abs(x - dfn_extension[maxIndex]))
134 index = RF[dn == adjustDisp].index.values[0]
135 simulationCriteria = index # Time before peak force for curve fitting
136 x = dn[index - simulationCriteria:index] # Array from 0 to location of max force
137 y = RF[index - simulationCriteria:index] # Array from 0 to location of max force
138 curveFit, Params, R_Squared = Least_Squares(x,y) # Perform least squares and
    ↳ return
139
140 # Updated force at specific max disp with adjusted value (Simulated data)
141 specificTime = maxForceTime
142 actualDisp = min(dn, key=lambda x:abs(x - dfn_extension[maxIndex]))
143 force_at_Dis = RF[dn == actualDisp].values[0]
144
145 # Simulated max force

```

```

146 simMaxForce = RF.max() # maximum simulated force value
147 simMaxDisp = dn[RF == simMaxForce] # displacement at the max force value
148
149 # Max peel force displacement at max and steady state
150 dfn_max_Displacement = dfn_extension[dfn_time == HTMax]
151 # dfn_ss_Displacement = np.array([dfn_extension[dfn_time == HSSi], dfn_extension[dfn_time
    ↳ == HSSf]]).flatten() # Didn't seem to work here
152 dfn_ss_Displacement = [dfn_extension[dfn_time == HSSi].values[0], dfn_extension[dfn_time
    ↳ == HSSf].values[0]]
153
154 """ Plots """
155 # Plot the data trace to compare the simulated results with the force
    ↳ displacement curves
156 plt.plot(dfn_extension, dfn_force, '-', color='r', linewidth=1, markersize=2,
    ↳ label = '{} , Age: {}'.format(HID, HAGE), alpha = 0.5)
157
158 if str(HFMax) == 'nan' and str(HSSi) == 'nan':
159     print('No max or steady state')
160     pass
161
162 if str(HFMax) != 'nan':
163     plt.plot(dfn_max_Displacement, HFMax, '.', color='k', linewidth=1, markersize=20,
    ↳ label = 'Max Peel - {:.4f} (mN)'.format(HFMax),
    ↳ path_effects=[pe.Stroke(linewidth=4, foreground='k'), pe.Normal()])
164 plt.plot(x20, curveFit20, '-', color='tab:blue', linewidth=2, label=r'Curve
    ↳ fit Max - 20 (s) y = {:.4f}x + {:.4f} (mN), $r^2$ =
    ↳ {:.4f}'.format(Params20[1], Params20[0], R_Squared20), alpha = 1)
165
166 if str(HSSi) != 'nan':
167     plt.plot(dfn_ss_Displacement, ssValArray, '-', color='c', linewidth=3, markersize=2,
    ↳ label = 'Steady State - {:.4f} (mN)'.format(HFSS),
    ↳ path_effects=[pe.Stroke(linewidth=5, foreground='k'), pe.Normal()])
168
169 plt.plot(dn, RF, '-', color='blue', linewidth=2, markersize=2, label = r'Simulated
    ↳ Reaction force $\Sigma F_{\text{Retina}}$')
170 plt.plot(x, curveFit, '-', color='tab:green', linewidth=2, markersize=2, label= 'y
    ↳ = {:.4f}x + {:.4f} (mN), $r^2$ = {:.4f}'.format(Params[1], Params[0],
    ↳ R_Squared))
171 plt.plot(actualDisp, force_at_Displacement, '.', color='tab:orange', linewidth=1,
    ↳ markersize = 20, label = 'Force at max disp {:.4f}
    ↳ (mN)'.format(force_at_Displacement))
172 plt.plot(simMaxDisp, simMaxForce, '.', color='tab:red', linewidth=1, markersize =
    ↳ 20, label = 'Simulated maximum Force {:.4f} (mN)'.format(simMaxForce))
173
174 # error between slope and force value
175 plt.plot([actualDisp, dfn_max_Displacement], [force_at_Displacement, HFMax], '--', linewidth = 1,
    ↳ color = 'magenta', label = r'ABS difference between force @ peak values is:
    ↳ {:.4f}'.format(abs(HFMax - force_at_Displacement)))
176 plt.plot([], [], 'white', label = r'ABS difference between slopes is:
    ↳ {:.4f}'.format(abs(Params20[1] - Params[1])))
177 plt.plot([], [], 'white', label = r'ABS ratio between slopes is:
    ↳ {:.4f}'.format(abs(Params20[1] / Params[1])))
178
179 ##### Plot Data #####
180 plt.axhline(0,color='black') # x = 0
181 plt.axvline(0,color='black') # y = 0     plt.xlabel('Displacement
    ↳ (mm)',fontSize=18)
182 plt.ylabel('Force (mN)',fontSize=18)

```

```

183 plt.title('Vitreous',fontSize=20)
184 plt.grid()
185 plt.legend(loc = 'best',fontSize = 'medium')
186 plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
    ↳ '_SlopeCompare.png'),dpi=300, bbox_inches='tight') # Save figure
187 plt.close()
188
189 return Params[1], force_at_Displacement, np.max(RF) # Slope, force @ specified time, max
    ↳ force
190
191 def findMaxForce(fileName, dataDirectory, maxForceTime, dataCompare):
192     """
193     Parameters
194     -----
195     fileName : Output txt file with the odb data
196     dataDirectory : Location of the output file
197
198     Returns
199     -----
200     maximumForce : Maximum force from the txt file
201     """
202
203     global maximumForce
204     """ Call function to return max displacement """
205     ModelParamsFile = [f for f in os.listdir(dataDirectory) if
    ↳ os.path.isfile(os.path.join(dataDirectory, f)) and
    ↳ f.startswith('output_Field')]
206     for mpFile in ModelParamsFile:
207         maximumForce = maxForce(mpFile, dataDirectory, maxForceTime, dataCompare)
208
209     return maximumForce

```

## 1.5.8 Move Simulation Files To A Single Folder

</> **Script 11:** Python script used to move all of the output Abaqus files to a separate </> folder for better organization during optimization batch runs.

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Fri Jun 19 16:02:44 2020
4
5 @author: Kiffer Creveling
6 """
7
8 # importing os module
9 import os
10 import glob
11 import shutil
12
13 def MoveAbqFiles(fileName, folderDirectory, abqWD):
14
15     # """ Change directory to correct path """
16
17     # dataDirectory = os.path.join(abqWD, fileName)
18     # if not os.path.exists(dataDirectory):

```

```

19     #     os.makedirs(dataDirectory)
20
21     # List of files in the ABQ working directory with the same name as the
22     # 'fileName'
23     fileList = glob.glob('{}.*'.format(os.path.join(abqWD, fileName)))
24     for i in fileList:
25         if i == folderDirectory:
26             # Skip the file with the exact same name (i.e. Folder name...)
27             continue
28         source = os.path.join(abqWD,i)
29         destination = os.path.join(folderDirectory)
30         # copy (since shutil.move wouldn't overwrite)
31         dest = shutil.copy(source, destination)
32         os.remove(source) # remove the source file
33
34     return print('Files moved = :')

```

## 1.6 Cohesive Surface Model

### 1.6.1 Python batch file

Abaqus 2016 was written in python 2.7 and therefore argparse was not around to pass parameter as input. Instead, arguments are passed in as command line (cmd) space separated commands. This script calls the subprocess module to call Abaqus python from python 3.8.5.

**</> Script 12: Python file that sets up the model parameters as input into the Abaqus </> model.**

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Thu Jan 28 22:28:58 2021
4
5  @author: Kiffer Creveling
6
7  This Python script does the following
8
9      1) Select input parameters
10     2) Generates the filename/description
11     3) Calls Abaqus to create the .inp file w/ attributes & runs the job
12     4) Creates a folder with the filename
13     5) Extracts data from the Abaqus.odb file and creates two output files
14         (Field/Hist)
15     6) Plots the data
16     7) Moves all files that have the same filename
17
18  """
19
20  import os
21  import sys
22  import numpy as np

```

```

23 import pandas as pd
24 # import itertools as it # iteration tools (product fcn)
25 # from scipy import *
26 # import scipy.optimize as opt
27 import lmfit as lf
28 import pdb
29 import subprocess
30 import pprint
31
32 # Define the location of the Abaqus Working Directory
33 # specific folder path where this file is located
34 pythonScriptPath = os.getcwd()
35 abqWD, pythonFiles = os.path.split(pythonScriptPath) # split file path
36
37 # pythonScriptCreateINP_Run_ABQ (pS_ABQ)
38 pS_ABQ = os.path.join(pythonFiles, 'Cohesive_T3_EyeModel_Generate_Abaqus.py')
39 # pythonScriptExtract (pSE)
40 pSE = os.path.join(pythonFiles, 'Cohesive_T3_EyeModel_DataExtract.py')
41
42 # In[Job Info]
43
44 optE_V = True
45 optKsTsFE = False
46 sweep = False
47
48 if optE_V == True:
49     optimization = 'E_V'
50
51     """ Optimization of the vitreous using a tied interface """
52     # If "True" then abaqus uses a tied interface between the nodes
53     tieInterface = True
54
55     """ Objective Function Flags """
56     slopeFlag = True
57     maxForceFlag = True
58     ssForceFlag = False # Only used for damage
59
60     """ Traction separation """
61     DamageInitiation = False # If "False" then do not include damage initiation
62     DamageEvolution = False # If "False" then do not include damage evolution
63
64
65 if optKsTsFE == True:
66     optimization = 'K_nnK_ssK_ttt_nt_st_tFE'
67
68     """ Optimization of the vitreous using a tied interface """
69     # If "True" then abaqus uses a tied interface between the nodes
70     tieInterface = False
71
72     """ Objective Function Flags """
73     slopeFlag = False
74     maxForceFlag = True
75     ssForceFlag = True # Only used for damage
76
77     """ Traction separation """
78     DamageInitiation = True # If "False" then do not include damage initiation
79     DamageEvolution = True # If "False" then do not include damage evolution
80

```

```

81
82 if sweep == True:
83     optimization = None
84
85     """ Parametric sweep of the vitreous using a tied interface """
86     # If "True" then abaqus uses a tied interface between the nodes
87     tieInterface = False
88
89     """ Objective Function Flags """
90     slopeFlag = False
91     maxForceFlag = True
92     ssForceFlag = True # Only used for damage
93
94     """ Traction separation """
95     DamageInitiation = True # If "False" then do not include damage initiation
96     DamageEvolution = True # If "False" then do not include damage evolution
97
98
99 # # optimization info
100 # optList = []
101 # optList.append(None)
102 # optList.append('E_R')
103 # optList.append('E_V')
104 # optList.append('E_RE_V') # Retina and Vitreous Young's Modulus
105 # optList.append('K_nnK_ssK_tt') # Traction Separation Paramters
106 # Vitreous Young's Modulus and Traction Separation Parameters
107 # optList.append('E_VK_nnK_ssK_tt')
108 # optList.append('t_nt_st_t') # Damage initiation parameters
109 # optList.append('FE') # Damage evolution parameters
110 # optList.append('t_nt_st_tFE') # Damage initiation and evolution parameteres
111 # optList.append('K_nnK_ssK_ttt_nt_st_tFE') # All cohesive parameters
112 # All parameters except for retina young's modulus
113 # optList.append('E_VK_nnK_ssK_ttt_nt_st_tFE')
114
115 # Change to specific optimization parameter. If 'None', no optimization
116 # optimization = optList[2]
117 # print('Optimization parameters = ', optimization)
118
119 """ Optimization of the vitreous using a tied interface """
120 # # If "True" then abaqus uses a tied interface between the nodes
121 # tieInterface = False
122
123 # """ Objective Function Flags """
124 # slopeFlag = False
125 # maxForceFlag = True
126 # ssForceFlag = True # Only used for damage
127
128 # """ Traction separation """
129 # DamageInitiation = True # If "False" then do not include damage initiation
130 # DamageEvolution = True # If "False" then do not include damage evolution
131
132 """ Objective Function Error Formulation """
133 objFunErr = []
134 objFunErr.append('Difference') # Experimental - Simulated
135 objFunErr.append('Ratio') # Experimental/Simulated
136 # (Experimental - Simulated)/Experimental
137 objFunErr.append('Relative uncertainty')
138 # Change to specific optimization parameter. If 'None', no optimization

```



```

139 objErr = objFunErr[0]
140 print('Objective function error formulation = ', objErr)
141
142 # Calculation for error
143 ErrorCalculation = []
144 ErrorCalculation.append('two-point method') # Slope, Peak force, SS Force
145 ErrorCalculation.append('data-trace method') # interpolated array
146
147 errorMethod = ErrorCalculation[0]
148 print('Error method calculation = ', errorMethod)
149
150 ''' Symmetry '''
151 # Split model in half and multiply output by 2
152 symmetry = True
153
154 ''' Simplified '''
155 # Remove the rigid body on the plastic tab and glue
156 simplified = True
157
158 ''' Gravity '''
159 # Turn gravity on/off
160 gravity = False # Keep off until model is updated
161
162 # In[Comparison Data Trace]
163 compareDataFolder = 'PeelDataCompare'
164 specificDataTrace = 'Trace_45_Instron_Data.txt' # Data trace number
165 timeBeforePeak = 40 # Default is 20 seconds
166 dataCompare = os.path.join(abqWD, compareDataFolder, specificDataTrace)
167 dfValsn = pd.read_csv(dataCompare, sep="\t", nrows=29, header=None,
168                       names=['Var', 'Attribute'])
169
170 """ File Attributes """
171 HID = dfValsn['Attribute'][0]
172 HAGE = dfValsn['Attribute'][1]
173 HG = dfValsn['Attribute'][2]
174 HLR = dfValsn['Attribute'][3]
175 HR = dfValsn['Attribute'][4]
176 HSSi = float(dfValsn['Attribute'][12])
177 HSSf = float(dfValsn['Attribute'][13])
178 HTMax = float(dfValsn['Attribute'][14])
179 HDispMax = float(dfValsn['Attribute'][15])
180 HFMax = float(dfValsn['Attribute'][16]) # (mN)
181 HFSS = float(dfValsn['Attribute'][17])
182 # (mN/m) slope from 20 seconds prior to max force value
183 HSlope20 = float(dfValsn['Attribute'][20])
184
185 dfn = pd.read_csv(os.path.join(dataCompare), sep="\t", header=30)
186 dfn.columns = ['Time', 'Extension', 'Force']
187 tn = dfn.Time
188 dn = dfn.Extension
189 df = dfn.Force # (N)
190
191 maxForceMeasured = HFMax # Value from data trace
192 maxSlopeMeasured = HSlope20 # slope from 20 seconds prior to max force value
193 SS_Measured = HFSS # simulated steady state force
194
195 # In[Functions]
196

```

```

197 if DamageInitiation == False and DamageEvolution == True:
198     print('Unable to have DamageEvolution without DamageInitiation')
199     sys.exit()
200
201 """ Tic Toc to determine runtime """
202 def tic():
203     #Homemade version of matlab tic and toc functions
204     import time
205     global startTime_for_tictoc
206     startTime_for_tictoc = time.time()
207
208 def toc():
209     import time
210     if 'startTime_for_tictoc' in globals():
211         print("Elapsed time is " + str(time.time() - startTime_for_tictoc) +
212             " seconds.")
213         timeDiff = time.time() - startTime_for_tictoc
214         return timeDiff
215     else:
216         print("Toc: start time not set")
217
218 try:
219     os.environ.pop('PYTHONIOENCODING')
220 except KeyError:
221     pass
222
223 # Import modules that plot/move all abq files to the new foldername
224 from ParameterSelection import ReadRAWDataTrace
225 from Cohesive_T3_Data_Plot import PlotAbqData
226 from Cohesive_T3_Residual import findResidual
227 from Cohesive_T3_CSMAXSCRT_MaxValue import CSMAXSCRTAbqData
228 from Move_ABQ_Files_To_Folder import MoveAbqFiles
229
230 newLine = '\n' + 77*'-' + '\n'
231
232 def jobAttributes():
233     """
234     Input: parameters used to create the filename and job description
235
236     Output: namei, fileName, JobDescription
237     """
238
239     # Build the fileName
240     fi = [] # initialize array
241     fi.append('{}'.format(namei))
242     fi.append('g') if gravity == True else ''
243     fi.append('sym') if symmetry == True else ''
244     fi.append('t{}'.format(time))
245     fi.append('E1{}'.format(e1Seedi[0]))
246     fi.append('E2{}'.format(e2Seedi[0]))
247
248     if simplified == False:
249         fi.append('PT{}'.format(ptSeedi[0]))
250         fi.append('G{}'.format(gSeedi[0]))
251
252     fi.append('V1{}'.format(v1Seedi[0]))
253     fi.append('V2{}'.format(v2Seedi[0]))
254     fi.append('R{}'.format(rSeedi[0]))

```

```

255 fi.append('F{}'.format(massScaleFactori[0]))
256 fi.append('MS{}'.format(massScaleTimeIncrementi[0]))
257 fi.append('RE{:.0e}'.format(RetinaYoungsModulus_i))
258
259 if optimization is not None:
260     if optimization.find('E_V') == -1:
261         fi.append('VE{:.0e}'.format(VitreousYoungsModulus_i))
262
263 # If optimization, get rid of the title (Not an integer anymore)
264 if optimization is None:
265     fi.append('Kn{}'.format(int(Knni[0])))
266     fi.append('Ks{}'.format(int(Kssi[0])))
267     fi.append('Kt{}'.format(int(Ktti[0])))
268
269 # If True, then damage initiation, If optimization, get rid of the title
270 # (Not an integer anymore)
271 if DamageInitiation == True and optimization is None:
272     fi.append('tn{}'.format(int(tni[0])))
273     fi.append('ts{}'.format(int(tsi[0])))
274     fi.append('tt{}'.format(int(tti[0])))
275
276 # If True, then damage evolution, If optimization, get rid of the title
277 # (Not an integer anymore)
278 if ((DamageInitiation == True) and (DamageEvolution == True) and
279     (optimization is None)):
280     fi.append('FE{}'.format(int(FEi[0])))
281
282 # .format(optimization)) optimization flag (I.e. RE, VE, Knn, Kss,
283 # tn, or none)
284 fi.append('opt') if optimization is not None else ''
285 fi.append('TIE') if tieInterface == True else ''
286
287 if sweep == True:
288     # get rid of all attributes because a sweep is taking place
289     fi = fi[0]
290
291 """ Build file name and description """
292 fileName = ''.join(item for item in fi)
293 # fix header so no decimals, math show up in title
294 fileName = fileName.replace('+', '_').replace('-', '_').replace('.', '_')
295 jobNameString = 'Job Name - {}'.format(fileName)
296
297 # used for simplification of script
298 # Large value
299 multStrL = ('\n\tgeometric multiplier = 2*{}, \n\tbase value = {}, ' +
300             '\n\tmodel value = {}'.format(multStrL, baseValue, modelValue))
301 # Small value
302 multStrS = ('\n\tgeometric multiplier = 0.5*{}, \n\tbase value = {}, ' +
303             '\n\tmodel value = {}'.format(multStrS, baseValue, modelValue))
304
305 # Build the model description
306 si = [] # initialize array
307 si.append(newLine)
308 si.append('{} = model name'.format(namei))
309 si.append(jobNameString)
310 si.append('(g) - Gravity') if gravity == True else si.append('No Gravity')
311 # update name in list
312 si.append('(sym) SYMMETRIC model (XY) Plane') if symmetry == True else ''

```

```

313 # update name in list
314 si.append('(t) Simulated time {} (s)'.format(time))
315
316 # Eye Holder
317 si.append(('(E1) Eye holder outside edge seed size (Max) (SINGLE BIAS): '
318         + multStrS + ' (m)').format(*e1Seedi))
319 si.append(('(E2) Eye holder inside edge seed size (Min): ' + multStrS +
320         ' (m)').format(*e2Seedi))
321
322 # If simplified is in the title, get rid of glue and plastic tab
323 if simplified == False:
324     si.append(('(PT) Plastic tab seed size: ' + multStrS +
325             ' (m)').format(*ptSeedi))
326     si.append(('(G) Glue seed size: ' + multStrS + ' (m)').format(*gSeedi))
327
328 # Vitreous
329 si.append(('(V1) Vitreous seed size max (side edge seed set)-' +
330         '(SINGLE BIAS): ' + multStrS + ' (m)').format(*v1Seedi))
331 si.append(('(V2) Vitreous seed size min (top edge in contact with ' +
332         'retina): ' + multStrS + ' (m)').format(*v2Seedi))
333
334 # Retina
335 si.append(('(R) Retina seed size: ' + multStrS + ' (m)').format(*rSeedi))
336
337 # Mass scale factor
338 si.append(('(F) Mass scale factor: ' + multStrL +
339         '').format(*massScaleFactori))
340
341 # Mass scale time increment
342 si.append(('(MS) Mass scale time increment: ' + multStrS +
343         ' (s)').format(*massScaleTimeIncrementi))
344
345 # Material properties (Young's Modulus)
346 si.append("(RE) Retina Young's Modulus: model value = {} (Pa)"
347         .format(RetinaYoungsModulus_i))
348 si.append("(VE) Vitreous Young's Modulus: model value = {} (Pa)"
349         .format(VitreousYoungsModulus_i))
350
351 # Cohesive traction parameters
352 if tieInterface == False:
353     si.append(('(Kn) Knn: ' + multStrL + ' (Pa)').format(*Knni))
354     si.append(('(Ks) Kss: ' + multStrL + ' (Pa)').format(*Kssi))
355     si.append(('(Kt) Ktt: ' + multStrL + ' (Pa)').format(*Ktti))
356
357 # If True, then damage initiation
358 if DamageInitiation == True:
359     si.append(('(tn) tn: ' + multStrL + ' (Pa)').format(*tni))
360     si.append(('(ts) ts: ' + multStrL + ' (Pa)').format(*tsi))
361     si.append(('(tt) tt: ' + multStrL + ' (Pa)').format(*tti))
362
363 # If True, then damage evolution
364 if DamageInitiation == True and DamageEvolution == True:
365     si.append(('(FE) Fracture energy: ' + multStrL + ' (J)').format(*FEi))
366
367 # Optimization
368 if optimization is not None:
369     si.append('Optimization of {}'.format(optimization))
370     si.append('Objective function error formulation is the ' +

```

```

371         '{} calculation'.format(objErr))
372     si.append('Objective error calculation is the {}'.format(errorMethod))
373
374 if optimization == None:
375     si.append('Parametric sweep')
376     si.append('Objective function error formulation is the ' +
377         '{} calculation'.format(objErr))
378     si.append('Objective error calculation is the {}'.format(errorMethod))
379
380 # Tied interface
381 if tieInterface == True:
382     si.append('Tied interface between the Retina and the Vitreous')
383
384 # Data trace being compared for optimization
385 si.append('The data trace being compared is: {}'.format(specificDataTrace))
386
387 # Time shift info as it is a new capability
388 si.append('The time prior to the peak force time event used for ' +
389     'determining the linear region ' +
390     'was extended ({}).format(timeBeforePeak) +
391     'seconds before the actual peak')
392
393 si.append(newLine)
394
395 # Job description
396 jobDescription = '\n'.join(item for item in si)
397
398 print(newLine)
399 print(fileName)
400 print(newLine)
401 print(jobDescription)
402
403 # Write a .txt file with the file attributes
404 outfile = open(os.path.join(abqWD, fileName + '.txt'), 'w')
405 line = ('The file name indicates what parameters were used to define ' +
406     'the model\n')
407 outfile.write(line)
408 line = '\n' + jobDescription + '\n'
409 outfile.write(line)
410 outfile.close()
411 print(outfile)
412 return namei, fileName, jobDescription
413
414
415 def GenerateAbaqusModels():
416     """
417     Function used to call Command Line (Windows Batch file)
418
419     Parameters
420     -----
421     fileName : abaqus job with paramters
422
423     """
424     # ----- Step 2 -----#
425     # Generates the filename/description
426     modelName, fileName, jobDescription = jobAttributes()
427
428

```

```

429 # Strip job description from spaces and new lines
430 # replace new lines, spaces, equal signs
431 jobDescription = jobDescription.replace(' ', 'SPACE')
432 jobDescription = jobDescription.replace('\n', 'NEWLINE')
433 jobDescription = jobDescription.replace('\t', 'TAB')
434 jobDescription = jobDescription.replace('=', 'EQUALSSIGN')
435
436 print(newLine)
437
438 # ----- Step 3 -----#
439 # Calls Abaqus to create the job with the filename just created and
440 # run the job
441
442 # Strip spaces and make strings
443 ABQ = []
444 ABQ.append(pS_ABQ) # python 2.7 script
445
446 # gravity
447 ABQ.append(''.join([i.strip(' ') for i in str(gravity).split(',')]))
448
449 # symmetry
450 ABQ.append(''.join([i.strip(' ') for i in str(symmetry).split(',')]))
451
452 # Simplified model
453 ABQ.append(''.join([i.strip(' ') for i in str(simplified).split(',')]))
454
455 ABQ.append(modelName) # model name
456 ABQ.append(fileName) # file name
457
458 # time
459 ABQ.append(''.join([i.strip(' ') for i in str(time).split(',')]))
460
461 # eye holder seed size 1
462 ABQ.append(''.join([i.strip(' ') for i in str(e1Seedi).split(',')]))
463
464 # eye holder seed size 2
465 ABQ.append(''.join([i.strip(' ') for i in str(e2Seedi).split(',')]))
466
467 # plastic tab seed size
468 ABQ.append(''.join([i.strip(' ') for i in str(ptSeedi).split(',')]))
469
470 # glue seed size
471 ABQ.append(''.join([i.strip(' ') for i in str(gSeedi).split(',')]))
472
473 # vitreous seed 1 size
474 ABQ.append(''.join([i.strip(' ') for i in str(v1Seedi).split(',')]))
475
476 # vitreous seed 2 size
477 ABQ.append(''.join([i.strip(' ') for i in str(v2Seedi).split(',')]))
478
479 # retina seed size
480 ABQ.append(''.join([i.strip(' ') for i in str(rSeedi).split(',')]))
481
482 # mass scale factor
483 ABQ.append(''.join([i.strip(' ') for i in
484                     str(massScaleFactori).split(',')]))
485
486 # mass scale time

```

```

487 ABQ.append(', '.join([i.strip(' ') for i in
488                       str(massScaleTimeIncrementi).split(',')]))
489
490 # Retina Young's Modulus
491 ABQ.append(', '.join([i.strip(' ') for i in
492                       str(RetinaYoungsModulus_i).split(',')]))
493
494 # Vitreous Young's Modulus
495 ABQ.append(', '.join([i.strip(' ') for i in
496                       str(VitreousYoungsModulus_i).split(',')]))
497
498 # Cohesive behavior
499 ABQ.append(', '.join([i.strip(' ') for i in str(Knni).split(',')])) # Knn
500 ABQ.append(', '.join([i.strip(' ') for i in str(Kssi).split(',')])) # Kss
501 ABQ.append(', '.join([i.strip(' ') for i in str(Ktti).split(',')])) # Ktt
502
503 # DamageInitiation
504 ABQ.append(', '.join([i.strip(' ') for i in
505                       str(DamageInitiation).split(',')]))
506 ABQ.append(', '.join([i.strip(' ') for i in str(tni).split(',')])) # tn
507 ABQ.append(', '.join([i.strip(' ') for i in str(tsi).split(',')])) # ts
508 ABQ.append(', '.join([i.strip(' ') for i in str(tti).split(',')])) # tt
509
510 # DamageEvolution
511 ABQ.append(', '.join([i.strip(' ') for i in
512                       str(DamageEvolution).split(',')]))
513 ABQ.append(', '.join([i.strip(' ') for i in str(FEi).split(',')])) # FE
514
515 # Optimization None/optimized parameters
516 ABQ.append(', '.join([i.strip(' ') for i in str(optimization).split(',')]))
517
518 # Tied interface
519 ABQ.append(', '.join([i.strip(' ') for i in str(tieInterface).split(',')]))
520
521 # Model description
522 ABQ.append(jobDescription)
523
524 ABQ_parse_string = 'abaqus cae noGUI={} --' + (len(ABQ)-1)*' {'
525
526 ## Used for debugging, comment out to copy/paste output to cmd window
527 ## to check and see if it works
528 # print(ABQ_parse_string.format(*ABQ))
529 # pdb.set_trace()
530
531 cmd = subprocess.Popen(ABQ_parse_string.format(*ABQ),
532                        cwd=r'{}'.format(abqWD), stdin=subprocess.PIPE,
533                        stdout=subprocess.PIPE, stderr=subprocess.PIPE,
534                        shell=True).communicate()[0]
535
536 print(newLine)
537 print('Abaqus has generated the .inp and executed the job')
538
539 # ----- Step 4 -----#
540 # Creates a folder with the filename
541 folderDirectory = os.path.join(abqWD, fileName)
542 if not os.path.exists(folderDirectory):
543     os.makedirs(folderDirectory)
544 dataDirectory = os.path.join(folderDirectory, 'Output')

```

```

545 if not os.path.exists(dataDirectory):
546     os.makedirs(dataDirectory)
547 figuresDirectory = os.path.join(dataDirectory, 'Figures')
548 if not os.path.exists(figuresDirectory):
549     os.makedirs(figuresDirectory)
550 print(newLine)
551 print('New file location:\n{} \n'.format(folderDirectory))
552
553 # ----- Step 5 -----#
554 """
555 Extracts data from the Abaqus.odb file and creates two output files
556 (Field/Hist). Create the name to be parsed into ABQ from the command
557 line through a subprocess
558 """
559 ABQ = []
560 ABQ.append(pSE)
561 ABQ.append(fileName)
562 ABQ.append(gravity)
563 ABQ.append(symmetry)
564 ABQ.append(simplified)
565 ABQ.append(DamageInitiation)
566 ABQ.append(DamageEvolution)
567
568 ABQ_parse_string = 'abaqus python' + len(ABQ)*' {'
569
570 # # # # Used for debugging, comment out to copy/paste output to cmd window
571 # # # # to check and see if it works
572 # print(ABQ_parse_string.format(*ABQ))
573 # pdb.set_trace()
574
575 cmd = subprocess.Popen(ABQ_parse_string.format(*ABQ),
576                        cwd=r'{}'.format(abqWD), stdin=subprocess.PIPE,
577                        stdout=subprocess.PIPE, stderr=subprocess.PIPE,
578                        shell=True).communicate()[0]
579
580 print(newLine)
581 print('Abaqus has extracted Field/History output: ' +
582       '\n{} \n'.format(dataDirectory))
583
584 # ----- Step 6 -----#
585 # Plot data and store it in the variable name folder under "Figures"
586 print(fileName)
587 print(dataDirectory)
588 PlotAbqData(fileName, dataDirectory, dataCompare, DamageInitiation,
589             DamageEvolution)
590 print(newLine)
591 print('New data plots:\n{} \n'.format(figuresDirectory))
592
593 # ----- Step 7 -----#
594 # Move all abaqus files to the folder with the same name
595 MoveAbqFiles(fileName, folderDirectory, abqWD)
596 print(newLine)
597 print('Files have been moved to: \n{} \n'.format(dataDirectory))
598
599 # ----- Step 8 (Error for minimization) -----#
600 maxForceTime = 100 # s
601 # slope is (mN/m)
602 residVals = findResidual(fileName, dataDirectory, maxForceTime,
603                          dataCompare, objErr, slopeFlag, maxForceFlag,

```



```

603             ssForceFlag, timeBeforePeak)
604     # Unpack
605     slopeSimulated = residVals[0]
606     maxForceSimulated = residVals[1]
607     SSmeanSimulated = residVals[2]
608     SSmedianSimulated = residVals[3]
609     y_new_exp_disp = residVals[4]
610     y_new_sim_disp = residVals[5]
611
612     # -----Step 9 (Damage Initiation for minimization) -----#
613     if DamageInitiation == True:
614         retinaMaxUCRT, vitreousMaxUCRT = CSMA_XCRTAbqData(fileName,
615                                                         dataDirectory,
616                                                         maxForceTime,
617                                                         dataCompare)
618     else:
619         retinaMaxUCRT = np.nan
620         vitreousMaxUCRT = np.nan
621
622     # (return slope, force, and maxucrt @ specified displacement)
623     fcnReturn = []
624     fcnReturn.append(fileName)
625     fcnReturn.append(slopeSimulated)
626     fcnReturn.append(maxForceSimulated)
627     fcnReturn.append(SSmeanSimulated)
628     fcnReturn.append(SSmedianSimulated)
629     fcnReturn.append(retinaMaxUCRT)
630     fcnReturn.append(vitreousMaxUCRT)
631     fcnReturn.append(y_new_exp_disp)
632     fcnReturn.append(y_new_sim_disp)
633     return fcnReturn
634
635
636 def writeOutputData(fileNameList):
637     print("\nWriting out the Reaction Force data...")
638     filename = os.path.join(abqWD, 'FEAAttributes' + '.txt')
639     outfile = open(filename, 'w')
640     sep = '\t'
641     Header = [] # List of items for the header
642     Header.append('FileName')
643     Header.append('Time')
644     Header.append('E1')
645     Header.append('E2')
646     Header.append('PT')
647     Header.append('G')
648     Header.append('V1')
649     Header.append('V2')
650     Header.append('R')
651     Header.append('F')
652     Header.append('MS')
653     Header.append('RE')
654     Header.append('VE')
655     Header.append('Knn')
656     Header.append('Kss')
657     Header.append('Ktt')
658     Header.append('DamageInitiation')
659     Header.append('tn')
660     Header.append('ts')

```

```

661 Header.append('tt')
662 Header.append('DamageEvolution')
663 Header.append('FE')
664 Header.append('Optimization')
665 Header.append('TIE')
666 Header.append('errorListL2Norm')
667 Header.append('ObjectiveFunction')
668 Header.append('simTime')
669 line = sep.join(item for item in Header)
670 outfile.write(line)
671 outfile.write('\n')
672 outfile.write('\t'.join(str(item) for item in attributeArray_0))
673 for i in list(fileNameList):
674     outfile.write('\n')
675     tempList = [str(i[0])] # filename
676     for j in list(i[1]):
677         tempList.append(str(j)) # file attributes
678         tempList.append(str(i[2])) # sim time
679         outfile.write('\t'.join(str(item) for item in tempList))
680 outfile.close()
681 print("\nDone!")
682 print("\nThe output file will be named '{}".format(filename) + "'")
683 print("\nIt will be in the same working directory as your Abaqus model\n")
684
685 # Print File of tests ran in order
686 print("\nWriting out the Reaction Force data...")
687 filename = os.path.join(abqWD, 'FEAFileList' + '.txt')
688 outfile = open(filename, 'w')
689 line = 'FileName'
690 outfile.write(line)
691 for i in list(fileNameList):
692     line = '\n%s' % (i[0])
693     outfile.write(line)
694 outfile.close()
695 print("\nDone!")
696 print("\nThe output file will be named '{}".format(filename) + "'")
697 print("\nIt will be in the same working directory as your Abaqus model\n")
698
699
700
701 if __name__ == '__main__':
702     # Run the function
703
704     # ----- Step 1 -----#
705     # T3
706     name = ['T3']
707
708     paramSelect = ReadRAWDataTrace(dataCompare, abqWD, timeBeforePeak)
709
710     t0, t1, tshift, fe = paramSelect # Unpack variables
711
712     if t0 > tshift:
713         # If the t1 value is greater than tshfit, use tshift for
714         # the simulation time
715         # Shouldn't have to do this as this issue has been handled
716         t0 = tshift
717         print('updated the time to be the shift value')
718

```

```

719 # Determine which time to use (Max value or steady state)
720 if optE_V == True:
721     time = int(t0)
722     FEValOpt = fe
723
724 elif optKsTsFE == True or sweep == True:
725     time = int(t1)
726     FEValOpt = fe
727
728 # Select input parameters
729 # time = 97 # Simulation parameter time S25 shifted
730 # time = 250 # Simulation parameter time
731
732 ''' Optimized results using the updated optimization routine 2/11/21
733 using the larger vitreous strip model with first looking at the tied
734 interface between the vitreous and retina '''
735 VitreousYoungsModulus_0 = 524.265652
736 KnnValOpt = 26.312450336667535
737 KssValOpt = 25.620054908304496
738 KttValOpt = 27.028398378844223
739 tnValOpt = 18.51999887865916
740 tsValOpt = 17.98861859153288
741 ttValOpt = 10.906247748496245
742 # FEValOpt = -9.427062078905504
743
744 """ Vitreous Young's Modulus """
745
746 VitreousYoungsModulus_0 = 50.03617188307464 # optimized using Tie
747
748 """ Retina Young's Modulus """
749 RetinaYoungsModulus_0 = 11120.0 # Pa Optimized with the vitreous
750
751 """ Eye holder inside edge """
752 e1Seed_0 = 1 # Base seed
753 e1SeedArray = [] # Array of multipliers
754 n = 11 # number of increments
755 for i in range(10, n):
756     # Decrease mesh seed by a factor of 2
757     e1SeedArray.append([i, e1Seed_0, e1Seed_0*(0.5)**i])
758
759 """ Eye holder outside edge """
760 # This will most likely never get smaller (saves computational time)
761 e2Seed_0 = 1 # Base seed
762 e2SeedArray = []
763 n = 9 # number of increments
764 for i in range(8, n):
765     # Decrease mesh seed by a factor of 2
766     e2SeedArray.append([i, e2Seed_0, e2Seed_0*(0.5)**i])
767
768 """ Plastic tab """
769 ptSeed_0 = 1 # Plastic tab seed size
770 ptSeedArray = [] # Array of multipliers
771 n = 7 # number of increments
772 for i in range(6, n):
773     # Decrease mesh seed by a factor of 2
774     ptSeedArray.append([i, ptSeed_0, ptSeed_0*(0.5)**i])
775
776 """ Glue """

```

```

777 gSeed_0 = 1 # Glue seed size
778 gSeedArray = [] # Array of multipliers
779 n = 8 # number of increments
780 for i in range(7, n):
781     # Decrease mesh seed by a factor of 2
782     gSeedArray.append([i, gSeed_0, gSeed_0*(0.5)**i])
783
784 """ Vitreous """
785 # smaller seed size
786 v1Seed_0 = 1 # Vitreous (max seed size)
787 v1SeedArray = [] # Array of multipliers
788 n = 30 # number of increments
789 # for i in np.linspace(10, 12, n): # range(10, n):
790 #     # Decrease mesh seed by a factor of 2
791 #     v1SeedArray.append([i, v1Seed_0, v1Seed_0*(0.5)**i])
792
793 # Comment out when parameters have been optimized
794 v1ValOpt = 11.38 # (convergence value)
795 v1SeedArray.append([v1ValOpt, v1Seed_0, v1Seed_0*(0.5)**v1ValOpt])
796
797 # larger seed size (should be factor of 4 times smaller ## 2 numbers)
798 v2Seed_0 = 1 # Vitreous (min seed size)
799 v2SeedArray = [] # Array of multipliers
800 n = 9 # number of increments
801 # for i in range(8, n):
802 #     # Decrease mesh seed by a factor of 2
803 #     v2SeedArray.append([i, v2Seed_0, v2Seed_0*(0.5)**i])
804
805 # Comment out when parameters have been optimized
806 v2ValOpt = 8
807 v2SeedArray.append([v2ValOpt, v2Seed_0, v2Seed_0*(0.5)**v2ValOpt])
808
809 """ Retina """
810 rSeed_0 = 1 # Base seed
811 rSeedArray = [] # Array of multipliers
812 n = 30 # number of increments
813 # for i in np.linspace(10, 13.5, n): # range(10, n):
814 #     # Decrease mesh seed by a factor of 2
815 #     rSeedArray.append([i, rSeed_0, rSeed_0*(0.5)**i])
816
817 rValOpt = 11.3275 # (convergence value)
818 rSeedArray.append([rValOpt, rSeed_0, rSeed_0*(0.5)**rValOpt])
819
820 """ mass scale factor """
821 massScaleFactor_0 = 1
822 massScaleFactorArray = [] # Array of multipliers
823 n = 1 # number of increments
824 for i in range(0, n):
825     # Increase by a factor of 2
826     massScaleFactorArray.append([i, massScaleFactor_0,
827                                  massScaleFactor_0*2**i])
828
829 """ mass scale time increment """
830 massScaleTimeIncrement_0 = 1
831 massScaleTimeArray = [] # multiplier and value
832 n = 8 # number of increments
833 for i in range(7, n):
834     # Decrease by a factor of 2

```

```

835     massScaleTimeArray.append([i, massScaleTimeIncrement_0,
836                               massScaleTimeIncrement_0*(0.5)**i])
837
838 if massScaleTimeIncrement_0 == 0:
839     print('No Mass Scaling... This will take a while...ABAQUS is ' +
840           'deciding for us')
841
842     """ Knn """
843     Knn_0 = 1
844     KnnArray = [] # Array of multipliers
845     # n = 31 # number of increments # 23 works when R = 2e3, and V = 736 Pa
846     # for i in range(30, n):
847     #     # Increase by a factor of 2
848     #     KnnArray.append([i, Knn_0, Knn_0*(2)**i])
849
850     # Comment out when parameters have been optimized
851     KnnArray.append([KnnValOpt, Knn_0, Knn_0*(2)**KnnValOpt])
852
853     """ Kss """
854     Kss_0 = 1
855     KssArray = [] # Array of multipliers
856     # n = 31
857     # for i in range(30, n):
858     #     # Increase by a factor of 2
859     #     KssArray.append([i, Kss_0, Kss_0*(2)**i])
860
861     # Comment out when parameters have been optimized
862     KssArray.append([KssValOpt, Kss_0, Kss_0*(2)**KssValOpt])
863
864     """ Ktt """
865     Ktt_0 = 1
866     KttArray = [] # Array of multipliers
867     # n = 31
868     # for i in range(30, n):
869     #     # Increase by a factor of 2
870     #     KttArray.append([i, Ktt_0, Ktt_0*(2)**i])
871
872     # Comment out when parameters have been optimized
873     KttArray.append([KttValOpt, Ktt_0, Ktt_0*(2)**KttValOpt])
874
875     """ tn """
876     tn_0 = 1
877     tnArray = [] # Array of multipliers
878     # n = 10 # 10 works when using max stress criteria
879     # for i in range(9, n):
880     #     # Increase by a factor of 2
881     #     tnArray.append([i, tn_0, tn_0*(2)**i])
882
883     # Comment out when parameters have been optimized
884     tnArray.append([tnValOpt, tn_0, tn_0*(2)**tnValOpt])
885
886     """ ts """
887     ts_0 = 1
888     tsArray = [] # Array of multipliers
889     # n = 10
890     # for i in range(9, n):
891     #     # Increase by a factor of 2
892     #     tsArray.append([i, ts_0, ts_0*(2)**i])

```

```

893
894 # Comment out when parameters have been optimized
895 tsArray.append([tsValOpt, tn_0, tn_0*(2)**tsValOpt])
896
897 """ tt """
898 tt_0 = 1
899 ttArray = [] # Array of multipliers
900 # n = 10
901 # for i in range(9, n):
902 #     # Increase by a factor of 2
903 #     ttArray.append([i, tt_0, tt_0*(2)**i])
904
905 # Comment out when parameters have been optimized
906 ttArray.append([ttValOpt, tn_0, tn_0*(2)**ttValOpt])
907
908 """ FE """
909 FE_0 = 1
910 FEArray = [] # Array of multipliers
911 # n = -8
912 # for i in range(-9, n):
913 #     # Increase by a factor of 2
914 #     FEArray.append([i, FE_0, FE_0*(2)**i])
915
916 FEArray.append([FEValOpt, FE_0, FE_0*(2)**FEValOpt])
917
918 errorList = np.nan # initial error
919 slopeList = np.nan # Initial slope
920 FmaxList = np.nan # Initial max peel force
921 FSSList = np.nan # Initial steady-state peel force
922
923 """ Attribute Array Initial Values """
924 attributeArray_0 = []
925 attributeArray_0.append('BaseVals')
926 attributeArray_0.append(time)
927 attributeArray_0.append(e1Seed_0)
928 attributeArray_0.append(e2Seed_0)
929 attributeArray_0.append(ptSeed_0)
930 attributeArray_0.append(gSeed_0)
931 attributeArray_0.append(v1Seed_0)
932 attributeArray_0.append(v2Seed_0)
933 attributeArray_0.append(rSeed_0)
934 attributeArray_0.append(massScaleFactor_0)
935 attributeArray_0.append(massScaleTimeIncrement_0)
936 attributeArray_0.append(RetinaYoungsModulus_0)
937 attributeArray_0.append(VitreousYoungsModulus_0)
938 attributeArray_0.append(Knn_0)
939 attributeArray_0.append(Kss_0)
940 attributeArray_0.append(Ktt_0)
941 attributeArray_0.append(DamageInitiation)
942 attributeArray_0.append(tn_0)
943 attributeArray_0.append(ts_0)
944 attributeArray_0.append(tt_0)
945 attributeArray_0.append(DamageEvolution)
946 attributeArray_0.append(FE_0)
947 attributeArray_0.append(optimization)
948 attributeArray_0.append(tieInterface)
949 attributeArray_0.append(errorList)
950 attributeArray_0.append(objErr)

```

```

951 attributeArray_0.append(slopeList)
952 attributeArray_0.append(FmaxList)
953 attributeArray_0.append(FSSLList)
954 attributeArray_0.append('simTime')
955
956
957 fileNameList = [] # List of files
958 counter = 0
959
960 if optimization is not None:
961     """ If the optimization variable is not "None" then optimize the
962         specific variable beins passed through """
963
964     name = name[0]
965
966     # BondStatus = True # interested in bonding
967
968     # # post damage failure model (If False, ignore Ktt, ts, and tn,
969     # # otherwise include them)
970     # pdfm = False
971
972     # Optimization method
973     # optName = 'NM' # Nelder-mead
974     # optName = 'P' # Powell
975     optName = 'C' # COBYLA
976     # optName = 'L' # LBFGSB
977     # optName = 'T' # Truncated Newton
978     # optName = 'S' # SLSQP
979     # optName = 'TC' # Trust-Constr
980
981     name0 = '_' .join([name, optName]) # used for optimization
982
983 def FEA_Residual(pars, data=None):
984     # Global variables
985     global counter
986     global name
987     global name0
988     global fileNameList
989     global time
990     global namei
991     global e1Seedi
992     global e2Seedi
993     global ptSeedi
994     global gSeedi
995     global v1Seedi
996     global v2Seedi
997     global rSeedi
998     global massScaleFactori
999     global massScaleTimeIncrementi
1000     global RetinaYoungsModulus_i
1001     global VitreousYoungsModulus_i
1002     global Knni
1003     global Kssi
1004     global Ktti
1005     global tni
1006     global tsi
1007     global tti
1008     global FEi

```

```

1009
1010     # Parameters used for optimization
1011     global errorList
1012
1013     print('Iteration # ', counter)
1014
1015     tic() # Start time
1016
1017     e1Seedi = e1SeedArray[0] # Default array
1018     e2Seedi = e2SeedArray[0] # Default array
1019     ptSeedi = ptSeedArray[0] # Default array
1020     gSeedi = gSeedArray[0] # Default array
1021     v1Seedi = v1SeedArray[0] # Default array
1022     v2Seedi = v2SeedArray[0] # Default array
1023     rSeedi = rSeedArray[0] # Default array
1024     massScaleFactori = massScaleFactorArray[0] # Default array
1025     massScaleTimeIncrementi = massScaleTimeArray[0] # Default array
1026     RetinaYoungsModulus_i = RetinaYoungsModulus_0 # Default value
1027     VitreousYoungsModulus_i = VitreousYoungsModulus_0 # Default value
1028     Knni = KnnArray[0] # Default array
1029     Kssi = KssArray[0] # Default array
1030     Ktti = KttArray[0] # Default array
1031     tni = tnArray[0] # Default array
1032     tsi = tsArray[0] # Default array
1033     tti = ttArray[0] # Default array
1034     FEi = FEArray[0] # Default array
1035
1036     # Extract the unknown parameters from the pars class variable
1037     # Determine the multiplier for the title
1038     for key, value in pars.items():
1039
1040         if key.find('ER') >= 0:
1041             """ Retina Young's Modulus """
1042             val = value.value
1043             RetinaYoungsModulus_i = val
1044
1045         elif key.find('EV') >= 0:
1046             """ Vitreous Young's Modulus """
1047             val = value.value
1048             VitreousYoungsModulus_i = val
1049
1050         elif key.find('Knn') >= 0:
1051             """ Knn """
1052             val = value.value
1053             mult = np.log(val)/np.log(2) # multiplier
1054             Knni = [mult, Knn_0, val]
1055
1056         elif key.find('Kss') >= 0:
1057             """ Kss """
1058             val = value.value
1059             mult = np.log(val)/np.log(2) # multiplier
1060             Kssi = [mult, Kss_0, val]
1061
1062         elif key.find('Ktt') >= 0:
1063             """ Ktt """
1064             val = value.value
1065             mult = np.log(val)/np.log(2) # multiplier
1066             Ktti = [mult, Ktt_0, val]

```



```

1067
1068     elif key.find('tn') >= 0:
1069         """ tn """
1070         val = value.value
1071         mult = np.log(val)/np.log(2) # multiplier
1072         tni = [mult, tn_0, val]
1073
1074     elif key.find('ts') >= 0:
1075         """ ts """
1076         val = value.value
1077         mult = np.log(val)/np.log(2) # multiplier
1078         tsi = [mult, ts_0, val]
1079
1080     elif key.find('tt') >= 0:
1081         """ tt """
1082         val = value.value
1083         mult = np.log(val)/np.log(2) # multiplier
1084         tti = [mult, tt_0, val]
1085
1086     elif key.find('FE') >= 0:
1087         """ FE """
1088         val = value.value
1089         mult = np.log(val)/np.log(2) # multiplier
1090         FEi = [mult, FE_0, val]
1091
1092     # Keep track of simulation results by unique names with the count
1093     # number. Comment out the second part to save file space if you
1094     # are not interested in saving every single simulation
1095     namei = name0 #+ str(counter)
1096
1097     # Error of the simulation
1098     L2Normi = np.sqrt(np.dot(errorList, errorList))
1099
1100     # multipliers to be appended to the output file to show changes
1101     # in parameters
1102     aAM = [] # attributeArrayMultipliar
1103     aAM.append(time)
1104     aAM.append(e1Seedi[0])
1105     aAM.append(e2Seedi[0])
1106     aAM.append(ptSeedi[0])
1107     aAM.append(gSeedi[0])
1108     aAM.append(v1Seedi[0])
1109     aAM.append(v2Seedi[0])
1110     aAM.append(rSeedi[0])
1111     aAM.append(massScaleFactori[0])
1112     aAM.append(massScaleTimeIncrementi[0])
1113     aAM.append(RetinaYoungsModulus_i)
1114     aAM.append(VitreousYoungsModulus_i)
1115     aAM.append(Knni[0])
1116     aAM.append(Kssi[0])
1117     aAM.append(Ktti[0])
1118     aAM.append(DamageInitiation)
1119     aAM.append(tni[0])
1120     aAM.append(tsi[0])
1121     aAM.append(tti[0])
1122     aAM.append(DamageEvolution)
1123     aAM.append(FEi[0])
1124     aAM.append(optimization)

```

```

1125 aAM.append(tieInterface)
1126 aAM.append(L2Normi)
1127 aAM.append(objErr)
1128
1129 # Call the function
1130 # Runs jobs and saves file names
1131 funReturn = GenerateAbaqusModels()
1132 fileName = funReturn[0]
1133 maxSlopeSimulated = funReturn[1]
1134 maxForceSimulated = funReturn[2]
1135 SSmeanSimulated = funReturn[3]
1136 SSmedianSimulated = funReturn[4]
1137 retinaMaxUCRT = funReturn[5]
1138 vitreousMaxUCRT = funReturn[6]
1139 y_new_exp_disp = funReturn[7]
1140 y_new_sim_disp = funReturn[8]
1141
1142 # add the simulated outputs to the data file
1143 aAM.append(maxSlopeSimulated)
1144 aAM.append(maxForceSimulated)
1145 aAM.append(SSmedianSimulated)
1146
1147 # Determine the measure of error used for optimization
1148 # Let the data trace being passed in act as the comparison
1149 maxSlopeMeasured, maxForceMeasured = data
1150
1151 # Error calculation
1152 errorDict = {} # Dictionary
1153 if objErr == 'Difference':
1154     errorDict['slope'] = (maxSlopeMeasured - maxSlopeSimulated) if
1155         ↪ slopeFlag == True else []
1156     errorDict['maxForce'] = (maxForceMeasured - maxForceSimulated) if
1157         ↪ maxForceFlag == True else []
1158     errorDict['ssForce'] = (SS_Measured - SSmeanSimulated) if
1159         ↪ ssForceFlag == True else []
1160 elif objErr == 'Ratio':
1161     errorDict['slope'] = (1 - maxSlopeMeasured / maxSlopeSimulated) if
1162         ↪ slopeFlag == True else []
1163     errorDict['maxForce'] = (1 - maxForceMeasured / maxForceSimulated) if
1164         ↪ maxForceFlag == True else []
1165     errorDict['ssForce'] = (1 - SS_Measured / SSmeanSimulated) if
1166         ↪ ssForceFlag == True else []
1167 elif objErr == 'Relative uncertainty':
1168     errorDict['slope'] = ((maxSlopeMeasured -
1169         ↪ maxSlopeSimulated)/maxSlopeMeasured) if slopeFlag == True else []
1170     errorDict['maxForce'] = ((maxForceMeasured -
1171         ↪ maxForceSimulated)/maxForceMeasured) if maxForceFlag == True else
1172         ↪ []
1173     errorDict['ssForce'] = ((SS_Measured -
1174         ↪ SSmedianSimulated)/SS_Measured) if ssForceFlag == True
1175         ↪ else []
1176 else:
1177     print('Error in MaxForceError')
1178     sys.exit()
1179
1180 # Error array values
1181 errorList = list(errorDict.values()) # convert to list
1182 errorList = [x for x in errorList if x] # get rid of empty values

```

```

1172         L2Normi = np.sqrt(np.dot(errorList, errorList))
1173
1174         # Calculate residual
1175         residual = y_new_exp_disp - y_new_sim_disp # residual
1176
1177         # Calculate L2Norm
1178         L2Norm = np.sqrt(np.dot(residual, residual))
1179
1180         simulationTime = toc() # Determine run time
1181         # appends the fileName & File Attributes
1182         fileNameList.append([fileName, aAM,
1183                               simulationTime])
1184         print('{} Error calculation: '.format(objErr), errorList)
1185         print('L2 norm objective calculation', L2Normi)
1186         print('L2 Norm residual', L2Norm)
1187         print('Done\n\n\n')
1188         counter += 1
1189
1190         # Determine which calculation is going to be used for optimization
1191         if errorMethod == 'two-point method':
1192             FEA_Residual = errorList
1193         elif errorMethod == 'data-trace method':
1194             FEA_Residual = residual
1195
1196         return FEA_Residual
1197
1198     maxFuncEval = 200
1199     tolVal = 1e-4
1200
1201     # Use the data variable to input the max slope and force from the
1202     # known data trace
1203     data = [maxSlopeMeasured, maxForceMeasured]
1204
1205     # Initial, Upper, and Lower bounds for parameters
1206
1207     # Young's Modulus - Retina
1208     ER_i = 5000 # Pa
1209     ER_LB = 50 # Pa
1210     ER_UB = 11000 # Pa
1211
1212     # Young's Modulus - Vitreous
1213     EV_i = 172 # Pa (Prony series calculation)
1214     # EV_i = 500 # Pa (Trying higher initial guess)
1215     EV_LB = 50 # Pa
1216     EV_UB = 2100 # Pa
1217     # EV_UB = 400 # Pa (Lowering the upper bound)
1218
1219     # Traction-Separation Behavior
1220     # Knn_i = 2**18 # Stress [Pa]
1221     # Knn_i = 2**26.32642676301851 # better optimized guess
1222     Knn_i = 2**20.872765304828103 # Stress [Pa] # different guess Low E_v
1223     # Knn_LB = 2**22 # Stress [Pa]
1224     Knn_LB = 2**10 # Stress [Pa] # Try lowering the bound
1225     Knn_UB = 2**28 # Stress [Pa]
1226
1227     # Kss_i = 2**18 # Stress [Pa]
1228     # Kss_i = 2**27.387981486684094 # better optimized guess
1229     Kss_i = 2**26.094732037712763 # different guess Low E_v

```

```

1230 # Kss_LB = 2**22 # Stress [Pa]
1231 Kss_LB = 2**10 # Stress [Pa] # Try lowering the bound
1232 Kss_UB = 2**28 # Stress [Pa]
1233
1234 # Ktt_i = 2**18 # Stress [Pa]
1235 # Ktt_i = 2**27.88464867824286 # better optimized guess
1236 Ktt_i = 2**26.20110650892766 # different guess Low E_v
1237 # Ktt_LB = 2**22 # Stress [Pa]
1238 Ktt_LB = 2**10 # Stress [Pa] # Try lowering the bound
1239 Ktt_UB = 2**28 # Stress [Pa]
1240
1241 # Damage Initiation Behavior
1242 # tn_i = 2**9 # Stress [Pa]
1243 # tn_i = 2**18.51999887865916 # better optimized guess
1244 tn_i = 2**9.712181223168551 # different guess Low E_v
1245 tn_LB = 2**3 # Stress [Pa]
1246 tn_UB = 2**20 # Stress [Pa] 11 before
1247
1248 # ts_i = 2**9 # Stress [Pa]
1249 # ts_i = 2**17.98861859153288 # better optimized guess
1250 ts_i = 2**9.931687876075074 # different guess Low E_v
1251 ts_LB = 2**3 # Stress [Pa]
1252 ts_UB = 2**20 # Stress [Pa] 11 before
1253
1254 # tt_i = 2**9 # Stress [Pa]
1255 # tt_i = 2**10.906247748496245 # better optimized guess
1256 tt_i = 2**9.022372079206395 # different guess Low E_v
1257 tt_LB = 2**3 # Stress [Pa]
1258 tt_UB = 2**15 # Stress [Pa] 11 before
1259
1260 # Damage Evolution Behavior
1261 # FE_i = 3.738925970000001e-6 # Energy [J] ~ -18.028944662923816 # S25
1262 FE_i = 1.929e-6 # Energy [J] S47
1263 # FE_i = 2**-9.427062078905504 # better optimized guess
1264 FE_LB = 2**-30 # 2**0 # Energy [J]
1265 # FE_UB = 2**-8 # Energy [J] # small bounds on energy
1266 FE_UB = 2**0 # Energy [J] # increase bounds
1267
1268 # Specify parameters
1269 fit_params = lf.Parameters() # intialize the class for parameters
1270
1271 # Retina young's modulus
1272 if optimization.find('E_R') >= 0:
1273     fit_params.add('ER', value = ER_i, min=ER_LB, max=ER_UB, vary=True)
1274
1275 # Vitreous Young's Modulus
1276 if optimization.find('E_V') >= 0:
1277     fit_params.add('EV', value = EV_i, min=EV_LB, max=EV_UB, vary=True)
1278
1279 # parameter for making the retina stiffer than the vitreous
1280 if optimization.find('E_R') >= 0 and optimization.find('E_V') >= 0:
1281     fit_params.add('StiffDelta', value = 0.01, min=0, vary=True)
1282     # Constraint to allow vitreous to be not as stiff as the retina
1283     fit_params.add('stiffnessConstraint', expr = 'EV - StiffDelta')
1284
1285 # Knn
1286 if optimization.find('K_nn') >= 0:
1287     fit_params.add('Knn', value = Knn_i, min=Knn_LB, max=Knn_UB,

```

```

1288         vary=True)
1289
1290     # Kss
1291     if optimization.find('K_ss') >= 0:
1292         fit_params.add('Kss', value = Kss_i, min=Kss_LB, max=Kss_UB,
1293             vary=True)
1294
1295     # Ktt
1296     if optimization.find('K_tt') >= 0:
1297         fit_params.add('Ktt', value = Ktt_i, min=Ktt_LB, max=Ktt_UB,
1298             vary=True)
1299
1300     # tn
1301     if optimization.find('t_n') >= 0:
1302         fit_params.add('tn', value = tn_i, min=tn_LB, max=tn_UB,
1303             vary=True)
1304
1305     # ts
1306     if optimization.find('t_s') >= 0:
1307         fit_params.add('ts', value = ts_i, min=ts_LB, max=ts_UB,
1308             vary=True)
1309
1310     # tt
1311     if optimization.find('t_t') >= 0:
1312         fit_params.add('tt', value = tt_i, min=tt_LB, max=tt_UB,
1313             vary=True)
1314
1315     # FE
1316     if optimization.find('FE') >= 0:
1317         fit_params.add('FE', value = FE_i, min=FE_LB, max=FE_UB,
1318             vary=True)
1319
1320     # Set up minimization class
1321     minClass = lf.Minimizer(FEA_Residual, fit_params,
1322         fcn_kws={'data': data},
1323         max_nfev = maxFuncEval) # fcn_args=(x,),
1324
1325     # (Different methods can be used here) Uses an array
1326     # out = minClass.leastsq() # Levenberg-Marquardt
1327
1328     # single scalar value
1329     # out = minClass.scalar_minimize(method='Nelder-Mead', tol=tolVal)
1330
1331     # single scalar value (if the objective function returns an array,
1332     # the sum of the squares of the array will be used (L2Norm))
1333     out = minClass.scalar_minimize(method='Cobyla', tol=tolVal)
1334
1335     lf.report_fit(out) # modelpars=p_true, show_correl=True
1336
1337     # Write data to txt files
1338     writeOutputData(fileNameList)
1339
1340 else:
1341
1342     # Number of simulations to perform (Simulation Batch Total)
1343     SBT = []
1344     SBT.append(len(name))
1345     SBT.append(len(e1SeedArray))

```

```

1346 SBT.append(len(e2SeedArray))
1347 SBT.append(len(ptSeedArray))
1348 SBT.append(len(gSeedArray))
1349 SBT.append(len(v1SeedArray))
1350 SBT.append(len(v2SeedArray))
1351 SBT.append(len(rSeedArray))
1352 SBT.append(len(massScaleFactorArray))
1353 SBT.append(len(massScaleTimeArray))
1354 SBT.append(len(KnnArray))
1355 SBT.append(len(KssArray))
1356 SBT.append(len(KttArray))
1357 SBT.append(len(tnArray))
1358 SBT.append(len(tsArray))
1359 SBT.append(len(ttArray))
1360 SBT.append(len(FEArray))
1361
1362 ZipArray = []
1363 ZipArray.append(max(SBT)*name)
1364 ZipArray.append(max(SBT)*e1SeedArray)
1365 ZipArray.append(max(SBT)*e2SeedArray)
1366 ZipArray.append(max(SBT)*ptSeedArray)
1367 ZipArray.append(max(SBT)*gSeedArray)
1368 ZipArray.append(max(SBT)*v1SeedArray)
1369 ZipArray.append(max(SBT)*v2SeedArray)
1370 ZipArray.append(max(SBT)*rSeedArray)
1371 ZipArray.append(max(SBT)*massScaleFactorArray)
1372 ZipArray.append(max(SBT)*massScaleTimeArray)
1373 ZipArray.append(max(SBT)*KnnArray)
1374 ZipArray.append(max(SBT)*KssArray)
1375 ZipArray.append(max(SBT)*KttArray)
1376 ZipArray.append(max(SBT)*tnArray)
1377 ZipArray.append(max(SBT)*tsArray)
1378 ZipArray.append(max(SBT)*ttArray)
1379 ZipArray.append(max(SBT)*FEArray)
1380
1381 # Iterate over the different combinations of parameters
1382 # If varying one parameter, then use iter.product(items in list...)
1383 # If varying multiple parameters, use zip*max(SBT)*items in list...
1384
1385 for (namei,
1386      e1Seedi,
1387      e2Seedi,
1388      ptSeedi,
1389      gSeedi,
1390      v1Seedi,
1391      v2Seedi,
1392      rSeedi,
1393      massScaleFactori,
1394      massScaleTimeIncrementi,
1395      Knni,
1396      Kssi,
1397      Ktti,
1398      tni,
1399      tsi,
1400      tti,
1401      FEi) in zip(*ZipArray):
1402     tic() # Start time
1403     counter += 1

```

```

1404 print(counter, 'of ', max(*SBT))
1405
1406 namei = namei + '_{}'.format(counter)
1407
1408 # set the i'th value to the initial value (Updated in
1409 # optimization algorithm)
1410 RetinaYoungsModulus_i = RetinaYoungsModulus_0
1411 VitreousYoungsModulus_i = VitreousYoungsModulus_0
1412
1413 # Call the function
1414 # Runs jobs and saves file names
1415 funReturn = GenerateAbaqusModels()
1416 fileName = funReturn[0]
1417 maxSlopeSimulated = funReturn[1]
1418 maxForceSimulated = funReturn[2]
1419 SSmeanSimulated = funReturn[3]
1420 SSmedianSimulated = funReturn[4]
1421 retinaMaxUCRT = funReturn[5]
1422 vitreousMaxUCRT = funReturn[6]
1423 y_new_exp_disp = funReturn[7]
1424 y_new_sim_disp = funReturn[8]
1425
1426 # Error calculation
1427 errorDict = {} # Dictionary
1428 if objErr == 'Difference':
1429     errorDict['slope'] = (maxSlopeMeasured - maxSlopeSimulated) if
1430     ↪ slopeFlag == True else []
1431     errorDict['maxForce'] = (maxForceMeasured - maxForceSimulated) if
1432     ↪ maxForceFlag == True else []
1433     errorDict['ssForce'] = (SS_Measured - SSmeanSimulated) if
1434     ↪ ssForceFlag == True else []
1435 elif objErr == 'Ratio':
1436     errorDict['slope'] = (1 - maxSlopeMeasured / maxSlopeSimulated) if
1437     ↪ slopeFlag == True else []
1438     errorDict['maxForce'] = (1 - maxForceMeasured / maxForceSimulated) if
1439     ↪ maxForceFlag == True else []
1440     errorDict['ssForce'] = (1 - SS_Measured / SSmeanSimulated) if
1441     ↪ ssForceFlag == True else []
1442 elif objErr == 'Relative uncertainty':
1443     errorDict['slope'] = ((maxSlopeMeasured -
1444     ↪ maxSlopeSimulated)/maxSlopeMeasured) if slopeFlag == True else []
1445     errorDict['maxForce'] = ((maxForceMeasured -
1446     ↪ maxForceSimulated)/maxForceMeasured) if maxForceFlag == True else
1447     ↪ []
1448     errorDict['ssForce'] = ((SS_Measured -
1449     ↪ SSmedianSimulated)/SS_Measured) if ssForceFlag == True
1450     ↪ else []
1451 else:
1452     print('Error in MaxForceError')
1453     sys.exit()
1454
1455 # Error array values
1456 errorList = list(errorDict.values()) # convert to list
1457 errorList = [x for x in errorList if x] # get rid of empty values
1458
1459 # Error of the simulation
1460 L2Normi = np.sqrt(np.dot(errorList, errorList))

```

```

1451     # Calculate residual
1452     residual = y_new_exp_disp - y_new_sim_disp # residual
1453
1454     # Calculate L2Norm
1455     L2Norm = np.sqrt(np.dot(residual, residual))
1456
1457     # multipliers to be appended to the output file to show changes
1458     # in parameters
1459     aAM = [] # attributeArrayMultipliar
1460     aAM.append(time)
1461     aAM.append(e1Seedi[0])
1462     aAM.append(e2Seedi[0])
1463     aAM.append(ptSeedi[0])
1464     aAM.append(gSeedi[0])
1465     aAM.append(v1Seedi[0])
1466     aAM.append(v2Seedi[0])
1467     aAM.append(rSeedi[0])
1468     aAM.append(massScaleFactori[0])
1469     aAM.append(massScaleTimeIncrementi[0])
1470     aAM.append(RetinaYoungsModulus_i)
1471     aAM.append(VitreousYoungsModulus_i)
1472     aAM.append(Knni[0])
1473     aAM.append(Kssi[0])
1474     aAM.append(Ktti[0])
1475     aAM.append(DamageInitiation)
1476     aAM.append(tni[0])
1477     aAM.append(tsi[0])
1478     aAM.append(tti[0])
1479     aAM.append(DamageEvolution)
1480     aAM.append(FEi[0])
1481     aAM.append(optimization)
1482     aAM.append(tieInterface)
1483     aAM.append(L2Normi)
1484     aAM.append(objErr)
1485     aAM.append(maxSlopeSimulated)
1486     aAM.append(maxForceSimulated)
1487     aAM.append(SSmedianSimulated)
1488
1489     simulationTime = toc() # Determine run time
1490     # appends the fileName & File Attributes
1491     fileNameList.append([fileName, aAM,
1492                          simulationTime])
1493     print('{} Error calculation: '.format(objErr), errorList)
1494     print('L2 norm objective calculation', L2Normi)
1495     print('L2 Norm residual', L2Norm)
1496     print('Done')
1497
1498     # Write data to txt files
1499     writeOutputData(fileNameList)

```

## 1.6.2 Input Parameter Selection

Determine input parameters to the Abaqus model. The following script not only determines maximum and steady-state peel force, but also integrates the force-displacement



curve from the maximum force to the beginning of the steady-state peel as the failure energy input to the cohesive optimization routine.

</> **Script 13:** *Parameter selection script that determines the updated time at the maximum and steady-state peel force after linear extrapolation to the origin.* </>

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Jan 19 15:07:50 2021
4
5  @author: Kiffer2
6  """
7
8  import numpy as np
9  import pandas as pd
10 import os
11 import sys
12 import matplotlib.pyplot as plt
13 from matplotlib.pyplot import cm
14 import matplotlib.path_effects as pe
15 from matplotlib.patches import Polygon
16 plt.rcParams['figure.figsize'] = [16, 9]
17 from scipy import interpolate
18 import pdb
19
20
21 # # Define the location of the Abaqus Working Directory
22 # # specific folder path where this file is located
23 # pythonScriptPath = os.getcwd()
24 # abqWD, pythonFiles = os.path.split(pythonScriptPath) # split file path
25
26 # filePath = os.getcwd() # current working directory
27 # codePath, pythonFolder = os.path.split(filePath) # split file path
28 # HWPPath, codesFolder = os.path.split(codePath) # split file path
29
30 # expDataPath = 'experimentalData' # folder of data files
31 # dataPath = os.path.join(HWPPath, expDataPath) # Path to data files
32
33 def Least_Squares(x,y):
34     """
35     Calculate the slope and y-intercept using matrix math
36     x & y are the coordinates of points
37
38     parameters (X,Y) Data
39
40     Returns:
41         Curve fit data and parameters  $m \cdot x + b$ , R squared value
42     """
43     Z = np.ones((len(x),2))
44     Z[:,1] = x
45     # Calculate the matrix inverse for the constants of the regression
46     A = np.dot(np.linalg.inv(np.dot(Z.T,Z)),(np.dot(Z.T,y)))
47     linFit = x*A[1] + A[0]
48
49     # Stats
50     SS_tot = np.sum((y - np.mean(y))**2)

```

```

51     SS_res = np.sum((y - linFit)**2)
52     Rsqd = 1 - SS_res/SS_tot
53
54     return linFit, A, Rsqd
55
56 def myformat(x):
57     myexp = int(np.floor(np.log10(x)))
58     xout = x*10**(-myexp)
59     strout = '{:.4f}'.format(xout) + '\cdot10^{'+ '{}'.format(myexp) + '}'
60     return strout
61
62
63 # In[previous data]
64
65 def ReadRAWDataTrace(dataPath, abqWD, timeBeforePeak):
66     """
67     Inputs: dataPath - file path to raw data
68     abqWD: abaqus working directory
69     timeBeforePeak: number of seconds prior to the peak where data will
70                   be extrapolated to the origin for curve fitting
71     """
72
73     timeBeforePeak = timeBeforePeak*10 # Convert s --> cs (10 data points/sec)
74
75     # Eliminate the file extension
76     dataPathNoExt = dataPath.split('.txt')[0]
77
78     # Determine the specific file name
79     fileDir, dataCompare = os.path.split(dataPathNoExt)
80
81     """ Read in the csv file """
82     dfValsn = pd.read_csv(dataPath, sep="\t", nrows=29, header=None,
83                          names=['Var', 'Attribute'])
84
85     """ File Attributes """
86     HID = dfValsn['Attribute'][0]
87     HAGE = dfValsn['Attribute'][1]
88     HG = dfValsn['Attribute'][2]
89     HLR = dfValsn['Attribute'][3]
90     HR = dfValsn['Attribute'][4]
91     HSSi = float(dfValsn['Attribute'][12])
92     HSSf = float(dfValsn['Attribute'][13])
93     HTMax = float(dfValsn['Attribute'][14])
94     HDispMax = float(dfValsn['Attribute'][15])
95     HFMax = float(dfValsn['Attribute'][16]) # (mN)
96     HFSS = float(dfValsn['Attribute'][17])
97     # slope from 20 seconds prior to max force value
98     HSlope20 = float(dfValsn['Attribute'][20]) # (mN/m)
99
100     dfn = pd.read_csv(dataPath, sep="\t", header=30)
101     dfn.columns = ['Time', 'Extension', 'Force']
102     dfn_time = dfn.Time
103     dfn_extension = dfn.Extension # mm
104     dfn_force = dfn.Force*1e3 # N ---> mN
105
106     # SS Array
107     ssTimeArray = np.array([HSSi, HSSf])
108     ssValArray = np.array([HFSS, HFSS])

```

```

109
110 # slope calculation for 20 seconds prior to the max peel force
111 # (Experimental Data)
112 maxIndex = dfn_time[dfn_time == HTMax].index.values[0]
113
114 # to location of max force
115 # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
116 t_n = dfn_extension[maxIndex - timeBeforePeak:maxIndex]
117 # to location of max force
118 # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
119 y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
120 # Perform least squares and return
121 curveFit_n, Params_n, R_Squared_n = Least_Squares(t_n,y)
122
123 # Shift extension data so that the linear region is extrapolated through
124 # the origin
125 shift = abs(Params_n[0]/Params_n[1])*0
126 dfn_extension = dfn_extension - shift
127
128 # Now that the data has been shifted, recalculate the linear regression
129 # using the reduced data set
130
131 # to location of max force
132 # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
133 t_n = dfn_extension[maxIndex - timeBeforePeak:maxIndex]
134 # to location of max force
135 # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
136 y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
137 # Perform least squares and return
138 curveFit_n, Params_n, R_Squared_n = Least_Squares(t_n,y)
139
140 # # Slope of the curve up to the max force !!!(from the simulated data)!!!
141 # adjustDisp = min(dn, key=lambda x:abs(x - dfn_extension[maxIndex]))
142 # index = RF[dn == adjustDisp].index.values[0]
143 # simulationCriteria = index # Time before peak force for curve fitting
144 # # Array from 0 to location of max force
145 # x = dn[index - simulationCriteria:index]
146 # # Array from 0 to location of max force
147 # y = RF[index - simulationCriteria:index]
148 # # Perform least squares
149 # curveFit, Params, R_Squared = Least_Squares(x,y)
150
151 # # Updated force at specific max disp with adjusted value (Simulated data)
152 # specificTime = maxForceTime
153 # actualDisp = min(dn, key=lambda x:abs(x - dfn_extension[maxIndex]))
154 # force_at_Displacement = RF[dn == actualDisp].values[0]
155
156 # # Simulated max force
157 # simMaxForce = RF.max() # maximum simulated force value
158 # simMaxDisp = dn[RF == simMaxForce] # displacement at the max force value
159
160 # Max peel force displacement at max and steady state
161 dfn_max_Displacement = dfn_extension[dfn_time == HTMax]
162 # Didn't seem to work here
163 # dfn_ss_Displacement = np.array([dfn_extension[dfn_time == HSSi],
164 #                                dfn_extension[dfn_time == HSSf]]).flatten()
165 dfn_ss_Displacement = [dfn_extension[dfn_time == HSSi].values[0],
166                        dfn_extension[dfn_time == HSSf].values[0]]

```

```

167
168 # In[Simulated Trace]
169
170 # dataDirectory = 'D:\Downloads\experimentalData'
171
172 # fileName = ('output_Field_S25CohesiveXLVitDiff_CT250S11' +
173 #             'SFONS7RE1e_04VE5e_02opt.txt')
174
175 # df = pd.read_csv(os.path.join(dataDirectory, fileName),
176 #                  sep="\t", header=0)
177
178 # Header = [] # Header information for the dataframe
179 # Header.append('Frame') # h1
180 # Header.append('Time') # h2
181 # Header.append('RF_y_dot') # h3
182 # Header.append('RFx') # h4
183 # Header.append('RFy') # h5
184 # Header.append('RFz') # h6
185 # Header.append('Nodal_Force') # h7
186 # Header.append('Tab_Displacement') # h8
187 # Header.append('Bond_Displacement') # h9
188 # Header.append('Stress') # h10
189 # Header.append('AVG_CSMAXCRT') # h11
190 # Header.append('AVG_CSDMG') # h12
191 # df.columns = Header
192
193 # tt = df.Time
194 # RF = df.RF_y_dot*1000 # N to mN
195 # dn = df.Tab_Displacement*1000 # m
196
197 # In[Plots]
198
199 """ Plots """
200 # Plot the data trace to compare the simulated results with the force
201 # displacement curves
202 fig, ax = plt.subplots()
203 ax.plot(dfn_extension, dfn_force, '-', color='r', linewidth=1,
204         markersize=2, label = '{} Age: {}'.format(HID, HAGE),
205         alpha = 0.5)
206
207 if str(HFMax) == 'nan' and str(HSSi) == 'nan':
208     print('No max or steady state')
209     pass
210
211 if str(HFMax) != 'nan':
212     ax.plot(dfn_max_Displ, HFMax, '.', color='k', linewidth=1,
213             markersize=20,
214             label = 'Max Peel - {:.4f} (mN)'.format(HFMax),
215             path_effects=[pe.Stroke(linewidth=4, foreground='k'),
216                           pe.Normal()])
217     ax.plot(t_n, curveFit_n, '-', color='tab:blue', linewidth=2,
218             label=r'Curve fit Max - {}'.format(int(timeBeforePeak/10)) +
219             ' (s) y = {:.4f}x '.format(Params_n[1]) +
220             '+ {:.4f} (mN), '.format(Params_n[0]) +
221             '$r^2$ = {:.4f}'.format(R_Squared_n),
222             alpha = 1)
223
224 if str(HSSi) != 'nan':

```

```

225     ax.plot(dfn_ss_Displ, ssValArray, '-', color='c', linewidth=3,
226             markersize=2,
227             label = 'Steady State - {:.4f} (mN)'.format(HFSS),
228             path_effects=[pe.Stroke(linewidth=5,
229                                   foreground='k'),
230                           pe.Normal())])
231
232     # Make the shaded region for the entire integral
233     a = dfn_max_Displ.values[0] # dfn_ss_Displ[0]
234     b = dfn_ss_Displ[0] # dfn_ss_Displ[1]
235
236     # Make the shaded region include the square below
237     adjust = 0 # 0 or 1 to get rid of the small square
238
239     # Filter the data in between the bounds
240     dfn_ext_adjust = dfn_extension[(dfn_extension >= a) & (dfn_extension < b)]
241     dnf_force_adjust = dnf_force[(dfn_extension >= a) & (dfn_extension < b)]
242
243     verts = [(a, HFSS*adjust),
244              *zip(dfn_ext_adjust, dnf_force_adjust),
245              (b, HFSS*adjust)]
246     poly = Polygon(verts, facecolor='0.8', edgecolor='0.5')
247     ax.add_patch(poly)
248
249     # Integral area
250     Integral = np.trapz(dnf_force_adjust - HFSS*adjust, dfn_ext_adjust)
251
252     # Centroid for plotting
253     CentroidX = 1/Integral*(np.trapz(dfn_ext_adjust*(dnf_force_adjust -
254                                         HFSS*adjust),
255                                     dfn_ext_adjust))
256     CentroidY = 1/Integral*(np.trapz((dnf_force_adjust**2 -
257                                     (HFSS*adjust)**2*adjust)/2,
258                                     dfn_ext_adjust))
259
260     # ax.text(b, (HFSS + HFSS)/2, r'\int_a^b f(x)\mathrm{d}x=' +
261     #         myformat(Integral*1e-6) + '$ (J)', horizontalalignment='center',
262     #         fontsize=20)
263     # ax.plot([0.5*max(dfn_extension), CentroidX], [0.5*max(dfn_force),
264     #         CentroidY])
265
266     prop = dict(arrowstyle="->", head_width=0.4, head_length=0.8", shrinkA=0,
267                 shrinkB=0)
268     # ax.arrow(0.5*max(dfn_extension), 0.5*max(dfn_force),
269     #         CentroidX - 0.5*max(dfn_extension),
270     #         CentroidY - 0.5*max(dfn_force),
271     #         head_width=0.1, head_length=0.1)
272     ax.annotate("", xy=(CentroidX, CentroidY), xytext=(0.5*max(dfn_extension),
273                                                         0.5*max(dfn_force)),
274                arrowprops=prop)
275
276     ax.text(0.5*max(dfn_extension), 0.5*max(dfn_force),
277            r'\int_a^b f(x)\mathrm{d}x=' + myformat(Integral*1e-6) + '$ (J)',
278            horizontalalignment='center', fontsize=20)
279
280     ax.spines['right'].set_visible(False)
281     ax.spines['top'].set_visible(False)
282     ax.xaxis.set_ticks_position('bottom')

```

```

283 ax.set_xticks((a, b))
284 ax.set_xticklabels(('${}$.format(a), '{}$.format(b)))
285 ax.set_yticks((HFSS, HFMax))
286 ax.set_yticklabels(('${:.5}$.format(HFSS), '{}:.5}$.format(HFMax)))
287
288 ##### Plot Data #####
289 plt.axhline(0,color='black') # x = 0
290 plt.axvline(0,color='black') # y = 0
291 plt.ylabel('Force (mN)',fontsize=18)
292 plt.xlabel('Displacement (mm)',fontsize=18)
293 plt.title('Vitreous',fontsize=20)
294 plt.grid()
295 plt.legend(loc = 'best',fontsize = 'medium')
296 plt.savefig(os.path.join(abqWD, 'GcSelection.pdf'), dpi=300,
297             bbox_inches='tight')
298 # plt.show()
299 plt.close()
300
301
302
303
304 # """ Derivative of the data trace """
305 # fig, ax = plt.subplots()
306
307 # deriv = np.gradient(dfn_force, dfn_extension)
308
309 # ax.plot(dfn_extension, deriv)
310 # ax.set_ylim(-100, 100) # maxRFList
311 # plt.show()
312
313 # In[Time plot]
314
315 # slope calculation for n seconds prior to the max peel force
316 # (Experimental Data)
317 maxIndex = dfn_time[dfn_time == HTMax].index.values[0]
318
319 # to location of max force
320 # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
321 t_n = dfn_time[maxIndex - timeBeforePeak:maxIndex]
322 y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
323 # Perform least squares and return
324 curveFit_n, Params_n, R_Squared_n = LeastSquares(t_n, y)
325
326 # Shift extension data so that the linear region is extrapolated
327 # through the origin
328 shift_time = abs(Params_n[0]/Params_n[1])*1
329 if Params_n[0] > 0:
330     # shift time data for visual purposes
331     dfn_time_shift = dfn_time + shift_time
332     dfn_ss_time_shift = ssTimeArray + shift_time
333     HTMax_shift = HTMax + shift_time
334 else:
335     # shift time data for visual purposes
336     dfn_time_shift = dfn_time - shift_time
337     dfn_ss_time_shift = ssTimeArray - shift_time
338     HTMax_shift = HTMax - shift_time
339
340

```

```

341 # Curve fit the shifted displacement
342 maxIndex = dfn_time[dfn_time == HTMax].index.values[0]
343
344 # to location of max force
345 # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
346 t_n = dfn_time_shift[maxIndex - timeBeforePeak:maxIndex]
347 y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
348 # Perform least squares and return
349 curveFit_n, Params_n, R_Squared_n = Least_Squares(t_n, y)
350
351 # to location of max force
352 # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
353 x_n = dfn_extension[maxIndex - timeBeforePeak:maxIndex]
354 # Perform least squares
355 curveFit_n_disp, Params_n_disp, R_Squared_n_disp = Least_Squares(x_n, y)
356
357 # Shift extension data so that the linear region is extrapolated through
358 # the origin
359 shift_disp = abs(Params_n_disp[0]/Params_n_disp[1])*1
360 if Params_n[0] > 0:
361     dfn_extension_shift = dfn_extension + shift_disp
362     dfn_ss_Displacement = dfn_ss_Displacement + shift_disp
363 else:
364     dfn_extension_shift = dfn_extension - shift_disp
365     dfn_ss_Displacement = dfn_ss_Displacement - shift_disp
366
367 # to location of max force
368 # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
369 x_n = dfn_extension_shift[maxIndex - timeBeforePeak:maxIndex]
370 # Perform least squares
371 curveFit_n_disp, Params_n_disp, R_Squared_n_disp = Least_Squares(x_n, y)
372
373
374 Fmax_t_shift = dfn_time_shift[maxIndex]
375 fit_t = np.linspace(0, Fmax_t_shift, 200) # Selected value
376
377 # true max
378 # fit_t = np.linspace(0, dfn_time_shift[np.argmax(dfn_force)], 200)
379 Fmax_x_shift = dfn_extension_shift[maxIndex]
380
381 # true max
382 # fit_x = np.linspace(0, dfn_extension_shift[np.argmax(dfn_force)], 200)
383 fit_x = np.linspace(0, Fmax_x_shift, 200) # Selected value
384
385 def fit(params, x):
386     b, m = params
387     return m*x + b
388
389 fit_vals_y_time = fit(Params_n, fit_t)
390 fit_vals_y_force = fit(Params_n_disp, fit_x)
391
392 ''' Reaction force vs. time shifted '''
393 fig, ax = plt.subplots()
394 ax.plot(dfn_time_shift, dfn_force,
395         label=r'Data - {}'.format(dataCompare.split('.')[0]))
396 ax.plot(fit_t, fit_vals_y_time, '--', label=r'Assumed linear region')
397 ax.plot(Fmax_t_shift, dfn_force[maxIndex], 'o', markersize=10,
398         label=r'Time at peak = {:.4} (s)'.format(max(fit_t)))

```

```

399
400 ax.plot(dfn_ss_time_shift, ssValArray, '-', color='c', linewidth=3,
401         markersize=2, label = 'Steady State - {:.4f} (mN)'.format(HFSS),
402         path_effects=[pe.Stroke(linewidth=5, foreground='k'), pe.Normal()])
403
404 ax.plot([], [], 'w',
405         label='Start SS time = {:.4f} (s)'.format(min(dfn_ss_time_shift)))
406 ax.plot([], [], 'w',
407         label='End SS time = {:.4f} (s)'.format(max(dfn_ss_time_shift)))
408
409 plt.axhline(0,color='black')
410 plt.axvline(0,color='black')
411
412 plt.ylabel('Force (mN)',fontsize=18)
413 plt.xlabel('Time from extrapolated zero (s)',fontsize=18)
414 plt.legend(loc='best')
415 # plt.xlim([0, max(dfn_time_shift)])
416 plt.savefig(os.path.join(abqWD, 'SimulationTime.pdf'), dpi=300,
417            bbox_inches='tight')
418 # plt.show()
419 plt.close()
420
421 ''' Reaction force vs. displacement shifted '''
422 fig, ax = plt.subplots()
423 ax.plot(dfn_extension_shift, dfn_force,
424         label=r'Data - {}'.format(dataCompare.split('.')[0]))
425 ax.plot(fit_x, fit_vals_y_force, '--', label=r'Assumed linear region')
426 ax.plot(Fmax_x_shift, dfn_force[maxIndex], 'o', markersize=10,
427         label=r'Time at peak = {:.4} (s)'.format(max(fit_t)))
428
429 ax.plot(dfn_ss_Displacement_shift, ssValArray, '-', color='c', linewidth=3,
430         markersize=2, label = 'Steady State - {:.4f} (mN)'.format(HFSS),
431         path_effects=[pe.Stroke(linewidth=5, foreground='k'), pe.Normal()])
432
433 ax.plot([], [], 'w',
434         label='Start SS time = {:.4f} (s)'.format(min(dfn_ss_time_shift)))
435 ax.plot([], [], 'w',
436         label='End SS time = {:.4f} (s)'.format(max(dfn_ss_time_shift)))
437
438 plt.axhline(0, color='black')
439 plt.axvline(0, color='black')
440
441 plt.ylabel('Force (mN)',fontsize=18)
442 plt.xlabel('Displacement (mm)',fontsize=18)
443 plt.legend(loc='best')
444 # plt.xlim([0, max(dfn_time_shift)])
445 plt.savefig(os.path.join(abqWD, 'SimulationDisp.pdf'), dpi=300,
446            bbox_inches='tight')
447 # plt.show()
448 plt.close()
449
450 # In[Interpolated Experimental Data]
451
452 # create array from 0 max peel force (linear equation fit from above)
453 # populate a pandas dataframe
454 # merge the data frame with the data above from the peak force to the end
455 # use the interp1d fcn to interpolate between data
456 # pass the simulated data into the interpolation

```



```

457
458 # Time greater than the shift intersection point
459 t_exp = dfn_time_shift[dfn_time_shift >= 0]
460 x_exp = dfn_extension_shift[dfn_time_shift >= 0]
461 y_exp = dfn_force[dfn_time_shift >= 0]
462
463 # data frame with original data
464 dfdata = pd.DataFrame(np.array([t_exp, x_exp, y_exp]).T,
465                        columns=['t', 'x', 'y'])
466
467 # Select time beyond the max time to the end of the data
468 t_geq_max = dfn_time_shift[maxIndex:]
469 x_geq_max = dfn_extension_shift[maxIndex:]
470 y_geq_max = dfn_force[maxIndex:]
471
472 # dataframe of data points from the max value to the end
473 dfgmax = pd.DataFrame(np.array([t_geq_max, x_geq_max, y_geq_max]).T,
474                       columns=['t', 'x', 'y'])
475
476 # data frame of points from zero to the max value
477 linArray = np.array([fit_t, fit_x, fit_vals_y_time])
478 dfLin = pd.DataFrame(linArray.T, columns=['t', 'x', 'y'])
479
480 # create the new data frame of linear points up to the peak and all points
481 # beyond
482 dfNew = dfLin.append(dfgmax, ignore_index=True)
483
484 # # Interpolate the experimental data
485 # n_data_pts = 100
486 # Time at the peak (shifted)
487 # start_point_time = tt[RF.argmax()]# - shift
488 # Disp at the peak (shifted)
489 # start_point_disp = dn[RF.argmax()]# - shift_disp
490 # f_exp_time = interpolate.interp1d(dfNew['t'], dfNew['y'])
491 # f_exp_disp = interpolate.interp1d(dfNew['x'], dfNew['y'])
492 # t_new_exp = np.linspace(start_point_time, tt[tt.argmax()],
493 #                          n_data_pts) # (s)
494 # x_new_exp = np.linspace(start_point_disp, dn[tt.argmax()],
495 #                          n_data_pts) # (mm)
496 # y_new_exp_time = f_exp_time(t_new_exp) # Interpolate `interp1d`
497 # y_new_exp_disp = f_exp_disp(x_new_exp) # Interpolate `interp1d`
498
499 # In[Interpolated Simulated Trace]
500
501 # # Interpolate the simulated data
502 # f_sim_time = interpolate.interp1d(tt, RF)
503 # f_sim_disp = interpolate.interp1d(dn, RF)
504 # t_new_sim = np.linspace(start_point_time, tt[tt.argmax()],
505 #                          n_data_pts) # (s)
506 # x_new_sim = np.linspace(start_point_disp, dn[tt.argmax()],
507 #                          n_data_pts) # (mm)
508 # y_new_sim_time = f_sim_time(t_new_sim) # Interpolate `interp1d`
509 # y_new_sim_disp = f_sim_disp(x_new_sim) # Interpolate `interp1d`
510
511 # In[Plots]
512 # ''' Time curve '''
513 # fit, ax = plt.subplots()
514 # ax.plot()

```

```

515 # ax.plot(dfdata['t'], dfdata['y'], label='Original Shifted Data',
516 #         alpha = 0.5)
517 # ax.plot(dfNew['t'], dfNew['y'], label='Merged Data',
518 #         alpha = 0.5)
519 # ax.plot(t_new_exp, y_new_exp_time, '--',
520 #         label='Interp Experimental Data')
521 # ax.plot(tt, RF, label='Simulated Data')
522 # ax.plot(t_new_sim, y_new_sim_time, ':', label='Interp Simulated Data')
523 # ax.set_xlim([0, 300])
524 # ax.set_xlabel('Time (s)', fontsize=14)
525 # ax.set_ylabel('Force (N)', fontsize=14)
526 # ax.legend(loc='best', fontsize=14)
527 # ax.grid('on')
528 # plt.savefig(os.path.join(abqWD, 'interp1d_Time.pdf'), dpi=300,
529 #             bbox_inches='tight')
530 # plt.show()
531
532 # ''' Displacement curve '''
533 # fit, ax = plt.subplots()
534 # ax.plot()
535 # ax.plot(dfdata['x'], dfdata['y'], label='Original Shifted Data',
536 #         alpha = 0.5)
537 # ax.plot(dfNew['x'], dfNew['y'], label='Merged Data',
538 #         alpha = 0.5)
539 # ax.plot(x_new_exp, y_new_exp_disp, '--',
540 #         label='Interp Experimental Data')
541 # ax.plot(dn, RF, label='Simulated Data')
542 # ax.plot(x_new_sim, y_new_sim_disp, ':', label='Interp Simulated Data')
543 # ax.set_xlim([0, max(dn)])
544 # ax.set_xlabel('Displacement (mm)', fontsize=14)
545 # ax.set_ylabel('Force (N)', fontsize=14)
546 # ax.legend(loc='best', fontsize=14)
547 # ax.grid('on')
548 # plt.savefig(os.path.join(abqWD, 'interp1d_Disp.pdf'), dpi=300,
549 #             bbox_inches='tight')
550 # plt.show()
551
552 # ''' Displacement curve only showing interpolated data '''
553 # abs residual calculation
554 # residual = abs(y_new_exp_disp - y_new_sim_disp)
555 # L2Norm = np.dot(residual, residual)
556
557 # fit, ax = plt.subplots()
558 # ax.plot()
559 # ax.plot(x_new_exp, y_new_exp_disp, '-', label='Interp Experimental Data')
560 # ax.plot(x_new_sim, y_new_sim_disp, '-', label='Interp Simulated Data')
561 # ax.plot(x_new_sim, residual, ':',
562 #         label=r'Residual = $\\|\\|\\| exp - sim \\|\\|$', alpha = 0.8)
563 # ax.plot([], [], color='white',
564 #         label=r'$L^2$ norm = {:.4f}'.format(L2Norm))
565 # ax.axhline(color='k', linewidth=0.25)
566 # ax.set_xlim([0, max(x_new_exp)])
567 # ax.set_xlabel('Displacement (mm)', fontsize=14)
568 # ax.set_ylabel('Force (N)', fontsize=14)
569 # ax.legend(loc='best', fontsize=14)
570 # ax.grid('on')
571 # plt.savefig(os.path.join(abqWD, 'interp1d_Disp_clean.pdf'), dpi=300,
572 #             bbox_inches='tight')

```

```

573     # plt.show()
574
575     print('Output files have been printed to determine the appropriate ' +
576           'parameters for the simulation')
577
578     returnArray = [max(fit_t), max(dfn_ss_time_shift), HTMax_shift,
579                     Integral*1e-6]
580     return returnArray
581
582 if __name__ == '__main__':
583     # Run the function
584
585     # fileName = sys.argv[-2]
586     # savePath = sys.argv[-1]
587
588     ReadRAWDDataTrace(fileName, abqWD, timeBeforePeak)

```

### 1.6.3 Abaqus Python Script

</> **Script 14:** *Abaqus python script used to create the input file (.inp) and execute the simulation.* </>

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Thu Jan 28 21:51:32 2021
4
5  @author: Kiffer Creveling
6  """
7
8  """ abaqus cae -noGUI abaqusMacros.py """
9
10 # -*- coding: mbcs -*-
11 # Do not delete the following import lines
12 from abaqus import *
13 from abaqusConstants import *
14 import __main__
15
16 import section
17 import regionToolset
18 import displayGroupMdbToolset as dgm
19 import part
20 import material
21 import assembly
22 import step
23 import interaction
24 import load
25 import mesh
26 import optimization
27 import job
28 import sketch
29 import visualization
30 import xyPlot
31 import displayGroupOdbToolset as dgo
32 import connectorBehavior
33 import numpy as np

```

```

34 import os
35 import sys
36
37 # In[Non-symmetric model]
38 '''
39 Non-symmetric model
40 '''
41
42 # location of the folder
43 # specific folder path where this file is located # os.getcwd()
44 pythonScriptPath = os.path.abspath("file")
45 abqWD, pythonFiles = os.path.split(pythonScriptPath) # split file path
46
47 # StepFile = 'Adult Human Eye holder Assembly.STEP'
48 # StepFile = 'Adult Human Eye holder Assembly 2 Step.STEP'
49 # Constrained
50 # StepFile = 'Adult Human Eye holder Assembly Constrained Bottom.STEP'
51
52 # Trimmed to prevent element distortion on low elastic modulus curve fits
53 StepFile = ('Adult Human Eye holder Assembly Constrained Bottom Trimmed ' +
54            'Retina.STEP')
55
56 SolidWorksDir = 'SolidWorksStepFiles' # Folder name
57
58 # Combine folder directory
59 SolidWorksStepFile = os.path.join(SolidWorksDir, StepFile)
60
61 def ImportStepEyeConstrained():
62     """ Use with the constrained bottom STEP file """
63     step = mdb.openStep(os.path.join(abqWD, SolidWorksStepFile),
64                          scaleFromFile=OFF)
65
66     abqModel.PartFromGeometryFile(name='V', geometryFile=step, bodyNum=1,
67                                   combine=False, dimensionality=THREE_D,
68                                   type=DEFORMABLE_BODY)
69     abqModel.PartFromGeometryFile(name='E', geometryFile=step, bodyNum=2,
70                                   combine=False, dimensionality=THREE_D,
71                                   type=DISCRETE_RIGID_SURFACE)
72     abqModel.PartFromGeometryFile(name='R', geometryFile=step, bodyNum=3,
73                                   combine=False, dimensionality=THREE_D,
74                                   type=DEFORMABLE_BODY)
75     abqModel.PartFromGeometryFile(name='T', geometryFile=step, bodyNum=4,
76                                   combine=False, dimensionality=THREE_D,
77                                   type=DISCRETE_RIGID_SURFACE)
78     abqModel.PartFromGeometryFile(name='G', geometryFile=step, bodyNum=5,
79                                   combine=False, dimensionality=THREE_D,
80                                   type=DISCRETE_RIGID_SURFACE)
81
82
83 def Retina_Mat_Prop(RetinaProp):
84     retina_E = RetinaProp # Passed in young's modulus
85     Retina_Description = """
86 Actually used the value from Chen 2014
87 E = 11.12 KPa
88
89 -----
90 Density (kg/m^3)
91 1100 -----> Esposito_2013

```

```

92
93 """
94     abqModel.Material(name='Retina', description=Retina_Description)
95     abqModel.materials['Retina'].Density(table=((1100.0, ), ))
96     abqModel.materials['Retina'].Elastic(table=((retina_E, 0.49), ))
97
98     # Assign the section to the part
99     abqModel.HomogeneousSolidSection(name='Retina_Section', material='Retina',
100                                     thickness=None)
101
102 def Vitreous_Mat_Prop(vitreousProp):
103     vitreous_E = vitreousProp # Passed in young's modulus
104     Vitreous_Description = """
105 -----
106 Density (kg/m^3)
107 950 -----> Esposito_2013
108
109 -----
110
111 # Tram 2018 Viscoelasticity data
112 # 4 Term Prony (Tram Data # 5 HU2018-0074 OD 1 Pa)
113 (0.1486397420159951, 0.0, 331.4796231072498),
114 (0.12469207412616717, 0.0, 3.388868494747128),
115 (0.29059507092540404, 0.0, 15.59692349525066),
116 (0.1591569334281, 0.0, 69.85134248442381)
117 """
118     abqModel.Material(name='Vitreous', description=Vitreous_Description)
119     abqModel.materials['Vitreous'].Density(table=((950.0, ), ))
120     ''' Using Lin2020 Paper to relate SLSM curve fit parameters to physical
121     values. Prony 4 Term (Long term) initial guess 172.77874855377468
122     optimization of E'''
123     abqModel.materials['Vitreous'].Elastic(moduli=LONG_TERM,
124                                           table=((vitreous_E, 0.49), ))
125
126     # Prony 4 Term calculated from normalized data
127     abqModel.materials['Vitreous'].Viscoelastic(
128         domain=TIME, time=PRONY, table=(
129             # Tram Data # 5
130             (0.1486397420159951, 0.0, 331.4796231072498),
131             (0.12469207412616717, 0.0, 3.388868494747128),
132             (0.29059507092540404, 0.0, 15.59692349525066),
133             (0.1591569334281, 0.0, 69.85134248442381)))
134
135     # Assign the section to the part
136     abqModel.HomogeneousSolidSection(name='Vitreous_Section',
137                                     material='Vitreous', thickness=None)
138
139 def E_Features():
140     ''' Eye holder features '''
141     p = abqModel.parts['E']
142
143     # Remove shell
144     c = p.cells
145     p.RemoveCells(cellList = c[0:1])
146
147     # Reference point
148     p.ReferencePoint(point=(0.0, 0.0, 0.0))
149

```

```

150     # Add E-set to the reference point
151     r = p.referencePoints
152     refPoints=(r[3], )
153     p.Set(referencePoints=refPoints, name='E_RP_Set')
154
155     # Edge seed sets
156     e = p.edges
157     edges = e.getSequenceFromMask(mask=('[#400f000 #1402 ]', ), )
158     p.Set(edges=edges, name='E_Edge_Seed_Set')
159
160     edges = e.getSequenceFromMask(mask=('[#f1ff0fff #2838 ]', ), )
161     p.Set(edges=edges, name='E_Outside_Edge_Seed_Set')
162
163     # Surfaces
164     s = p.faces
165     side1Faces = s.getSequenceFromMask(mask=('[#1ffff ]', ), )
166     p.Surface(side1Faces=side1Faces, name='E_Surf')
167
168
169 def G_Features():
170     ''' Glue features '''
171     p = abqModel.parts['G']
172     c = p.cells
173
174     # Remove cells for rigid body
175     p.RemoveCells(cellList = c[0:1])
176
177     # Reference point
178     p.ReferencePoint(point=(9.799E-03, 5.657E-03, 2.54E-03))
179
180     # Define the reference point for the rigid body
181     r = p.referencePoints
182     refPoints=(r[3], )
183     p.Set(referencePoints=refPoints, name='G_RP_Set')
184
185     # # Create sets
186     f = p.faces
187     faces = f.getSequenceFromMask(mask=('[#3f ]', ), )
188     p.Set(faces=faces, name='G_Set')
189     faces = f.getSequenceFromMask(mask=('[#20 ]', ), )
190     p.Set(faces=faces, name='G_T_Set')
191     faces = f.getSequenceFromMask(mask=('[#1 ]', ), )
192     p.Set(faces=faces, name='G_R_Set')
193
194     # Create surfaces
195     s = p.faces
196     side1Faces = s.getSequenceFromMask(mask=('[#3f ]', ), )
197     p.Surface(side1Faces=side1Faces, name='G_Surf')
198     side1Faces = s.getSequenceFromMask(mask=('[#20 ]', ), )
199     p.Surface(side1Faces=side1Faces, name='G_T_Surf')
200     side1Faces = s.getSequenceFromMask(mask=('[#1 ]', ), )
201     p.Surface(side1Faces=side1Faces, name='G_R_Surf')
202
203
204 def T_Features():
205     ''' Plastic Tab features '''
206     p = abqModel.parts['T']
207     c = p.cells

```

```

208
209 # Remove cells for rigid body
210 p.RemoveCells(cellList = c[0:1])
211
212 # Reference point
213 p.ReferencePoint(point=(16.241E-03, 9.74E-03, 13.E-06))
214
215 # Define the reference point for the rigid body
216 r = p.referencePoints
217 refPoints=(r[3], )
218 p.Set(referencePoints=refPoints, name='T_RP_Set')
219
220 # Create sets
221 f = p.faces
222 faces = f.getSequenceFromMask(mask=('#ff '), , )
223 p.Set(faces=faces, name='T_Set')
224 f = p.faces
225 faces = f.getSequenceFromMask(mask=('#2 '), , )
226 p.Set(faces=faces, name='T_G_Set')
227
228 # Create surfaces
229 s = p.faces
230 side1Faces = s.getSequenceFromMask(mask=('#ff '), , )
231 p.Surface(side1Faces=side1Faces, name='T_Surf')
232 side1Faces = s.getSequenceFromMask(mask=('#2 '), , )
233 p.Surface(side1Faces=side1Faces, name='T_G_Surf')
234
235
236 def R_Features():
237     ''' Retina features '''
238     p = abqModel.parts['R']
239     c = p.cells
240     cells = c.getSequenceFromMask(mask=('#1 '), , )
241     p.Set(cells=cells, name='R_Set')
242
243     f = p.faces
244     faces = f.getSequenceFromMask(mask=('#3 '), , )
245     p.Set(faces=faces, name='R_G_Set')
246
247     faces = f.getSequenceFromMask(mask=('#4 '), , )
248     p.Set(faces=faces, name='R_V_Set')
249
250     s = p.faces
251     side1Faces = s.getSequenceFromMask(mask=('#ff '), , )
252     p.Surface(side1Faces=side1Faces, name='R_Surf')
253
254     side1Faces = s.getSequenceFromMask(mask=('#3 '), , )
255     p.Surface(side1Faces=side1Faces, name='R_G_Surf')
256
257     side1Faces = s.getSequenceFromMask(mask=('#4 '), , )
258     p.Surface(side1Faces=side1Faces, name='R_V_Surf_BOND')
259
260 # Assign section
261 region = p.sets['R_Set']
262 p.SectionAssignment(region=region, sectionName='Retina_Section',
263                     offset=0.0, offsetType=MIDDLE_SURFACE, offsetField='',
264                     thicknessAssignment=FROM_SECTION)
265

```

```

266
267
268 def PartitionRetinaOnVitreous():
269     ''' Vitreous features additional partitions for creating the surface for
270     bonding'''
271     p = abqModel.parts['V']
272
273     # Partition V along the width of the retina
274     p.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=-0.00254)
275     abqModel.parts['V'].features.changeKey(fromName='Datum plane-1',
276                                           toName='Retina_Width_Neg_Z')
277
278     p.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=0.00254)
279     abqModel.parts['V'].features.changeKey(fromName='Datum plane-1',
280                                           toName='Retina_Width_Pos_Z')
281
282     # Create a datum plnd along the z axis plane
283     p.DatumAxisByPrincipalAxis(principalAxis=ZAXIS)
284     p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset=0.0)
285
286     # Create the rotated datum planes
287     d = p.datums
288     p.DatumPlaneByRotation(plane=d[5], axis=d[4], angle=18.75)
289     p.DatumPlaneByRotation(plane=d[5], axis=d[4], angle=-18.75)
290
291     ''' Partition the surface of the retina on the vitreous '''
292     p = abqModel.parts['V']
293     c, d = p.cells, p.datums
294     pickedCells = c.getSequenceFromMask(mask=('[#440 ]', ), )
295     p.PartitionCellByDatumPlane(datumPlane=d[3], cells=pickedCells)
296     pickedCells = c.getSequenceFromMask(mask=('[#408 ]', ), )
297     p.PartitionCellByDatumPlane(datumPlane=d[2], cells=pickedCells)
298     pickedCells = c.getSequenceFromMask(mask=('[#22 ]', ), )
299     p.PartitionCellByDatumPlane(datumPlane=d[6], cells=pickedCells)
300     pickedCells = c.getSequenceFromMask(mask=('[#140 ]', ), )
301     p.PartitionCellByDatumPlane(datumPlane=d[7], cells=pickedCells)
302
303
304 def Vitreous_Features():
305     ''' Assign specific features to the vitreous '''
306     p = abqModel.parts['V']
307     c, f, s = p.cells, p.faces, p.faces
308
309     # Sets
310     cells = c.getSequenceFromMask(mask=('[#ffffff ]', ), )
311     p.Set(cells=cells, name='V_Set')
312     faces = f.getSequenceFromMask(mask=('[#5090 ]', ), )
313     p.Set(faces=faces, name='V_R_Set')
314
315     # Surfaces
316     side1Faces = s.getSequenceFromMask(mask=('[#1805090 #3 #ff0 ]', ), )
317     p.Surface(side1Faces=side1Faces, name='V_Surf')
318     side1Faces = s.getSequenceFromMask(mask=('[#5090 ]', ), )
319     p.Surface(side1Faces=side1Faces, name='V_R_Surf_BOND')
320
321     # Assign the section to the part
322     region = p.sets['V_Set']
323     p.SectionAssignment(region=region,

```



```

324         sectionName='Vitreous_Section',
325         offset=0.0,
326         offsetType=MIDDLE_SURFACE,
327         offsetField='',
328         thicknessAssignment=FROM_SECTION)
329
330
331 def V_Partition_XYZ_Axis():
332     ''' Partition the sphere along the x, y, z axis '''
333     p = abqModel.parts['V']
334     c, v, e, d = p.cells, p.vertices, p.edges, p.datums
335     pickedCells = c.getSequenceFromMask(mask=('#1 ', ), )
336     p.PartitionCellByPlaneThreePoints(point1=v[1],
337                                       point2=v[0],
338                                       point3=v[3],
339                                       cells=pickedCells)
340
341     pickedCells = c.getSequenceFromMask(mask=('#3 ', ), )
342     p.PartitionCellByPlaneThreePoints(point1=v[0],
343                                       point2=v[4],
344                                       point3=v[2],
345                                       cells=pickedCells)
346
347     pickedCells = c.getSequenceFromMask(mask=('#f ', ), )
348     p.PartitionCellByPlaneThreePoints(point1=v[5],
349                                       point2=v[2],
350                                       point3=v[4],
351                                       cells=pickedCells)
352
353
354 def V_Internal_Sphere():
355     sphereRadius = 0.008 # radius of the internal sphere for meshing
356
357     s1 = abqModel.ConstrainedSketch(name='__profile__', sheetSize=0.1)
358     g, v, d, c1 = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
359     s1.sketchOptions.setValues(decimalPlaces=3)
360     s1.setPrimaryObject(option=STANDALONE)
361     s1.ConstructionLine(point1=(0.0, -0.05), point2=(0.0, 0.05))
362     s1.FixedConstraint(entity=g[2])
363     s1.ArcByCenterEnds(center=(0.0, 0.0),
364                       point1=(0.0, sphereRadius),
365                       point2=(0.0, -sphereRadius),
366                       direction=CLOCKWISE)
367     s1.Line(point1=(0.0, sphereRadius),
368            point2=(0.0, -sphereRadius))
369     s1.VerticalConstraint(entity=g[4], addUndoState=False)
370     s1.PerpendicularConstraint(entity1=g[3], entity2=g[4], addUndoState=False)
371     p = abqModel.Part(name='V_internal',
372                      dimensionality=THREE_D,
373                      type=DEFORMABLE_BODY)
374     p = abqModel.parts['V_internal']
375     p.BaseSolidRevolve(sketch=s1, angle=360.0, flipRevolveDirection=OFF)
376     s1.unsetPrimaryObject()
377     p = abqModel.parts['V_internal']
378     del abqModel.sketches['__profile__']
379
380
381 def mergeV():

```

```

382     ''' Merge the internal sphere with the vitreous '''
383     a = abqModel.rootAssembly
384     a.InstanceFromBooleanMerge(name='V_Merge',
385                                instances=(a.instances['V-1'],
386                                           a.instances['V_internal-1'], ),
387                                keepIntersections=ON,
388                                originalInstances=DELETE,
389                                domain=GEOMETRY)
390
391     # Clean up file names after merge
392     del abqModel.parts['V']
393     del abqModel.parts['V_internal']
394
395     abqModel.parts.changeKey(fromName='V_Merge', toName='V')
396     a = abqModel.rootAssembly
397     a.regenerate()
398     abqModel.rootAssembly.features.changeKey(fromName='V_Merge-1',
399                                              toName='V-1')
400
401     a.regenerate()
402
403
404     def AssembleV_for_Merging():
405         a1 = abqModel.rootAssembly
406         a1.DatumCsysByDefault(CARTESIAN)
407         p = abqModel.parts['V']
408         a1.Instance(name='V-1', part=p, dependent=ON)
409         p = abqModel.parts['V_internal']
410         a1.Instance(name='V_internal-1', part=p, dependent=ON)
411
412
413     def E_Mesh(InsideSeed, OutsideSeed):
414         p = abqModel.parts['E']
415         e = p.edges
416         pickedEdges = e.getSequenceFromMask(mask=('[#400f000 #1402 ]', ), )
417         p.seedEdgeBySize(edges=pickedEdges,
418                          size=0.0005,
419                          deviationFactor=0.1,
420                          minSizeFactor=0.1,
421                          constraint=FINER)
422         pickedEdges = e.getSequenceFromMask(mask=('[#1ff0fff #2838 ]', ), )
423         p.seedEdgeBySize(edges=pickedEdges,
424                          size=0.00342673,
425                          deviationFactor=0.1,
426                          minSizeFactor=0.1,
427                          constraint=FINER)
428
429         # (unique node numbering)
430         p.setValues(startNodeLabel=1000000, startElemLabel=1000000)
431         p.generateMesh()
432
433     def G_Mesh(seed):
434         p = abqModel.parts['G']
435         p.seedPart(size=seed, deviationFactor=0.1, minSizeFactor=0.1)
436         f = p.faces
437         pickedRegions = f.getSequenceFromMask(mask=('[#3f ]', ), )
438         p.setMeshControls(regions=pickedRegions, elemShape=QUAD)
439         elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=EXPLICIT)

```

```

440     elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=EXPLICIT)
441     f = p.faces
442     faces = f.getSequenceFromMask(mask=('[#3f ]', ), )
443     pickedRegions = (faces, )
444     p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
445     # (unique node numbering)
446     p.setValues(startNodeLabel=2000000, startElemLabel=2000000)
447     p.generateMesh()
448
449
450 def T_Mesh(seed):
451     p = abqModel.parts['T']
452     p.seedPart(size=seed, deviationFactor=0.1, minSizeFactor=0.1)
453     f = p.faces
454     pickedRegions = f.getSequenceFromMask(mask=('[#ff ]', ), )
455     p.setMeshControls(regions=pickedRegions, elemShape=QUAD)
456     elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=EXPLICIT)
457     elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=EXPLICIT)
458     f = p.faces
459     faces = f.getSequenceFromMask(mask=('[#ff ]', ), )
460     pickedRegions = (faces, )
461     p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
462     # (unique node numbering)
463     p.setValues(startNodeLabel=3000000, startElemLabel=3000000)
464     p.generateMesh()
465
466
467 def R_Mesh(seed):
468     p = abqModel.parts['R']
469     p.seedPart(size=seed, deviationFactor=0.1, minSizeFactor=0.1)
470     c, e = p.cells, p.edges
471     pickedRegions = c.getSequenceFromMask(mask=('[#1 ]', ), )
472     p.setMeshControls(regions=pickedRegions,
473                       technique=SWEEP,
474                       algorithm=ADVANCING_FRONT)
475     p.setSweepPath(region=c[0], edge=e[10], sense=FORWARD)
476     elemType1 = mesh.ElemType(elemCode=C3D8R,
477                               elemLibrary=EXPLICIT,
478                               kinematicSplit=AVERAGE_STRAIN,
479                               secondOrderAccuracy=ON,
480                               hourglassControl=ENHANCED,
481                               distortionControl=ON,
482                               lengthRatio=0.100000001490116)
483     elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
484     elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT)
485     c = p.cells
486     cells = c.getSequenceFromMask(mask=('[#1 ]', ), )
487     pickedRegions = (cells, )
488     p.setElementType(regions=pickedRegions,
489                     elemTypes=(elemType1, elemType2, elemType3))
490     p.generateMesh()
491     # (unique node numbering)
492     p.setValues(startNodeLabel=4000000, startElemLabel=4000000)
493     p.generateMesh()
494
495
496 def VitreousMesh(v1Seed, v2Seed):
497     ''' Specity tetrahedral elements '''

```

```

498 p = abqModel.parts['V']
499 c = p.cells
500 pickedRegions = c.getSequenceFromMask(mask=['#86f800'], ), )
501 p.setMeshControls(regions=pickedRegions, elemShape=TET, technique=FREE)
502 elemType1 = mesh.ElemType(elemCode=C3D20R)
503 elemType2 = mesh.ElemType(elemCode=C3D15)
504 elemType3 = mesh.ElemType(elemCode=C3D10)
505 cells = c.getSequenceFromMask(mask=['#86f800'], ), )
506 pickedRegions = (cells, )
507 p.setElementType(regions=pickedRegions,
508                  elemTypes=(elemType1, elemType2, elemType3))
509
510 ''' Specify hexahedral elements '''
511 elemType1 = mesh.ElemType(elemCode=C3D8R, elemLibrary=EXPLICIT)
512 elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
513 elemType3 = mesh.ElemType(elemCode=C3D4,
514                           elemLibrary=EXPLICIT,
515                           secondOrderAccuracy=ON,
516                           distortionControl=ON,
517                           lengthRatio=0.100000001490116)
518 cells = c.getSequenceFromMask(mask=['#86f800'], ), )
519 pickedRegions = (cells, )
520 p.setElementType(regions=pickedRegions,
521                  elemTypes=(elemType1, elemType2, elemType3))
522
523 elemType1 = mesh.ElemType(elemCode=C3D8R,
524                           elemLibrary=EXPLICIT,
525                           kinematicSplit=AVERAGE_STRAIN,
526                           secondOrderAccuracy=ON,
527                           hourglassControl=ENHANCED,
528                           distortionControl=ON,
529                           lengthRatio=0.100000001490116)
530 elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
531 elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT)
532 cells = c.getSequenceFromMask(mask=['#7907ff'], ), )
533 pickedRegions = (cells, )
534 p.setElementType(regions=pickedRegions,
535                  elemTypes=(elemType1, elemType2, elemType3))
536
537 # Seed the entire part
538 p.seedPart(size=v2Seed, deviationFactor=0.1, minSizeFactor=0.1)
539
540 # Seed the retina interface
541 e = p.edges
542 pickedEdges = e.getSequenceFromMask(mask=['#ffffff #7fec0fff #80012'],
543                                     ), )
544 p.seedEdgeBySize(edges=pickedEdges,
545                  size=v1Seed,
546                  deviationFactor=0.1,
547                  minSizeFactor=0.1,
548                  constraint=FINER)
549
550 # Seed the bias edges
551 e = p.edges
552 pickedEdges1 = e.getSequenceFromMask(mask=['#0 #104000 #10001'], ), )
553 pickedEdges2 = e.getSequenceFromMask(mask=['#0 #80020000 #900000'], ), )
554 p.seedEdgeByBias(biasMethod=SINGLE,
555                  end1Edges=pickedEdges1,

```

```

556         end2Edges=pickedEdges2,
557         minSize=v1Seed,
558         maxSize=v2Seed,
559         constraint=FINER)
560
561     # (unique node numbering)
562     p.setValues(startNodeLabel=5000000, startElemLabel=5000000)
563     p.generateMesh()
564
565
566 def QuadraticTetVitreous():
567     # Vitreous
568     p = abqModel.parts['V']
569     c = p.cells
570     pickedRegions = c.getSequenceFromMask(mask=('[#9f ]', ), )
571     p.deleteMesh(regions=pickedRegions)
572     p.setMeshControls(regions=pickedRegions, elemShape=TET, technique=FREE)
573     elemType1 = mesh.ElemType(elemCode=UNKNOWN_HEX, elemLibrary=EXPLICIT)
574     elemType2 = mesh.ElemType(elemCode=UNKNOWN_WEDGE, elemLibrary=EXPLICIT)
575     elemType3 = mesh.ElemType(elemCode=C3D10M, elemLibrary=EXPLICIT)
576     cells = c.getSequenceFromMask(mask=('[#9f ]', ), )
577     pickedRegions = (cells, )
578     p.setElementType(regions=pickedRegions,
579                     elemTypes=(elemType1, elemType2, elemType3))
580     elemType1 = mesh.ElemType(elemCode=UNKNOWN_HEX, elemLibrary=EXPLICIT)
581     elemType2 = mesh.ElemType(elemCode=UNKNOWN_WEDGE, elemLibrary=EXPLICIT)
582     elemType3 = mesh.ElemType(elemCode=C3D10M,
583                               elemLibrary=EXPLICIT,
584                               secondOrderAccuracy=ON,
585                               distortionControl=ON,
586                               lengthRatio=0.100000001490116)
587     c = p.cells
588     p.setElementType(regions=pickedRegions,
589                     elemTypes=(elemType1, elemType2, elemType3))
590     p.generateMesh()
591
592
593 def QuadraticTetRetina():
594     # Retina
595     p = abqModel.parts['R']
596     c = p.cells
597     pickedRegions = c.getSequenceFromMask(mask=('[#1 ]', ), )
598     p.deleteMesh(regions=pickedRegions)
599     p.setMeshControls(regions=pickedRegions, elemShape=TET, technique=FREE)
600     elemType1 = mesh.ElemType(elemCode=UNKNOWN_HEX, elemLibrary=EXPLICIT)
601     elemType2 = mesh.ElemType(elemCode=UNKNOWN_WEDGE, elemLibrary=EXPLICIT)
602     elemType3 = mesh.ElemType(elemCode=C3D10M, elemLibrary=EXPLICIT)
603     c = p.cells
604     cells = c.getSequenceFromMask(mask=('[#1 ]', ), )
605     pickedRegions = (cells, )
606     p.setElementType(regions=pickedRegions,
607                     elemTypes=(elemType1, elemType2, elemType3))
608     elemType1 = mesh.ElemType(elemCode=UNKNOWN_HEX, elemLibrary=EXPLICIT)
609     elemType2 = mesh.ElemType(elemCode=UNKNOWN_WEDGE, elemLibrary=EXPLICIT)
610     elemType3 = mesh.ElemType(elemCode=C3D10M,
611                               elemLibrary=EXPLICIT,
612                               secondOrderAccuracy=ON,
613                               distortionControl=ON,

```

```

614         lengthRatio=0.100000001490116)
615     p.setElementType(regions=pickedRegions,
616                     elemTypes=(elemType1, elemType2, elemType3))
617     p.generateMesh()
618
619
620     def Assembly():
621         a1 = abqModel.rootAssembly
622         a1.DatumCsysByDefault(CARTESIAN)
623         p = abqModel.parts['E']
624         a1.Instance(name='E-1', part=p, dependent=ON)
625         p = abqModel.parts['G']
626         a1.Instance(name='G-1', part=p, dependent=ON)
627         p = abqModel.parts['R']
628         a1.Instance(name='R-1', part=p, dependent=ON)
629         p = abqModel.parts['T']
630         a1.Instance(name='T-1', part=p, dependent=ON)
631         p = abqModel.parts['V']
632         a1.Instance(name='V-1', part=p, dependent=ON)
633
634
635     def GravityStep(time, prevStep, scaleFactor, MSTI, stepName, descrip):
636         abqModel.ExplicitDynamicsStep(name=stepName,
637                                       previous=prevStep,
638                                       description=descrip,
639                                       timePeriod=time,
640                                       massScaling=((SEMI_AUTOMATIC,
641                                                     MODEL,
642                                                     AT_BEGINNING,
643                                                     scaleFactor,
644                                                     MSTI,
645                                                     BELOW_MIN, 0, 0, 0.0, 0.0, 0,
646                                                     None), ))
647
648
649     def Step(time, prevStep, scaleFactor, MSTI, stepName, descrip):
650         abqModel.ExplicitDynamicsStep(name=stepName,
651                                       previous=prevStep,
652                                       description=descrip,
653                                       timePeriod=time,
654                                       massScaling=((SEMI_AUTOMATIC,
655                                                     MODEL,
656                                                     AT_BEGINNING,
657                                                     scaleFactor,
658                                                     MSTI,
659                                                     BELOW_MIN, 0, 0, 0.0, 0.0,
660                                                     0, None), ),
661                                       nlgeom=ON)
662
663         # Mass Scale default
664         if MSTI == 0:
665             print('This will take a while...ABAQUS is deciding for us')
666             # Use zero value
667             abqModel.steps[stepName].setValues(massScaling=PREVIOUS_STEP)
668         else:
669             print('Mass Scale Time Increment has been defined')
670
671     def F_output(stepName):

```

```

672 FOutputInterval = 50 # Double the data points (Default is 20)
673
674 if damageInitiation == False and damageEvolution == False:
675     # Whole Model Fieldoutput (RF, U, NFORC)
676     abqModel.FieldOutputRequest(name='F-Output-1',
677                                 createStepName=stepName,
678                                 variables=('RF',
679                                          'U',
680                                          'NFORC'),
681                                 numIntervals=FOutputInterval)
682
683 elif damageInitiation == True and damageEvolution == False:
684     # Whole Model Fieldoutput (RF, U, NFORC)
685     abqModel.FieldOutputRequest(name='F-Output-1',
686                                 createStepName=stepName,
687                                 variables=('RF',
688                                          'U',
689                                          'NFORC',
690                                          'CSMAXSCRT'),
691                                 numIntervals=FOutputInterval)
692
693 elif damageInitiation == True and damageEvolution == True:
694     # Whole Model Fieldoutput (RF, U, NFORC)
695     abqModel.FieldOutputRequest(name='F-Output-1',
696                                 createStepName=stepName,
697                                 variables=('RF',
698                                          'U',
699                                          'NFORC',
700                                          'CSDMG',
701                                          'CSMAXSCRT'),
702                                 numIntervals=FOutputInterval)
703
704 # Set specific field output (Retina LE & S)
705 regionDef=abqModel.rootAssembly.allInstances['R-1'].sets['R_Set']
706 abqModel.FieldOutputRequest(name='Retina_LE_S',
707                             createStepName=stepName,
708                             variables=('LE',
709                                       'S'),
710                             numIntervals=FOutputInterval,
711                             region=regionDef,
712                             sectionPoints=DEFAULT,
713                             rebar=EXCLUDE)
714
715 # Set specific field output (Vitreous LE & S)
716 regionDef=abqModel.rootAssembly.allInstances['V-1'].sets['V_Set']
717 abqModel.FieldOutputRequest(name='Vitreous_LE_S',
718                             createStepName=stepName,
719                             variables=('LE',
720                                       'S'),
721                             numIntervals=FOutputInterval,
722                             region=regionDef,
723                             sectionPoints=DEFAULT,
724                             rebar=EXCLUDE)
725
726 # # Set specific field output (Rigid Body U & RF)
727 # regionDef=abqModel.rootAssembly.allInstances['G-1'].sets['G_RP_Set']
728 # abqModel.FieldOutputRequest(name='Glue_U_RF',

```

```

730     #                                     createStepName=stepName,
731     #                                     variables=('U',
732     #                                     'RF'),
733     #                                     numIntervals=FOutputInterval,
734     #                                     region=regionDef,
735     #                                     sectionPoints=DEFAULT,
736     #                                     rebar=EXCLUDE)
737
738
739 def H_output(stepName):
740     # Internal/Kinetic Energy
741     abqModel.HistoryOutputRequest(name='H-Output-1',
742                                   createStepName=stepName,
743                                   variables=('ALLIE',
744                                             'ALLKE'))
745
746     # # Define specific reaction force on the glue reference point
747     # a = abqModel.rootAssembly
748     # regionDef=a.allInstances['G-1'].sets['G_RP_Set']
749     # abqModel.HistoryOutputRequest(name='G_RP_Output_U_RF_RM',
750     #                               createStepName=stepName,
751     #                               variables=('U1', 'U2', 'U3',
752     #                                         'RF1', 'RF2', 'RF3',
753     #                                         'RM1', 'RM2', 'RM3'),
754     #                               region=regionDef, sectionPoints=DEFAULT,
755     #                               rebar=EXCLUDE)
756
757
758 def General_Contact(stepName, cIP):
759     # Rename the two variables
760     GC_IP = 'IntProp-GC' # Interaction property
761     GC = 'General_Contact' # General Contact name
762     # cIP = 'cohesive_IntProp' # cohesive interaction property name
763     abqModel.ContactProperty(GC_IP)
764
765     GC_IntProp = abqModel.interactionProperties[GC_IP] # simplify code
766
767     # if gravity == True:
768     #     # Gravity keeps the vitreous from energetically moving after peeling
769     GC_IntProp.TangentialBehavior(formulation=PENALTY,
770                                  directionality=ISOTROPIC,
771                                  slipRateDependency=OFF,
772                                  pressureDependency=OFF,
773                                  temperatureDependency=OFF,
774                                  dependencies=0,
775                                  table=((0.2, ), ),
776                                  shearStressLimit=None,
777                                  maximumElasticSlip=FRACTION,
778                                  fraction=0.005,
779                                  elasticSlipStiffness=None)
780
781     # else:
782     #     # Prevent the vitreous from sliding inside the eye holder
783     #     GC_IntProp.TangentialBehavior(formulation=ROUGH)
784
785     GC_IntProp.NormalBehavior(pressureOverclosure=HARD,
786                              allowSeparation=ON,
787                              constraintEnforcementMethod=DEFAULT)
788     abqModel.ContactExp(name=GC, createStepName=stepName)

```



```

788
789 GC_Int = abqModel.interactions[GC] # simplify code
790 GC_Int.includedPairs.setValuesInStep(stepName=stepName, useAllstar=ON)
791 GC_Int.contactPropertyAssignments.appendInStep(stepName=stepName,
792                                                  assignments=((GLOBAL,
793                                                         SELF,
794                                                         GC_IP),
795                                                         )
796                                                  )
797
798
799 def updateGeneralContact(stepName, Knn, Kss, Ktt, damageInitiation,
800                          tn, ts, tt, damageEvolution, FE):
801     ''' Specify the cohesive surface behavior between the retina and vitreous
802     during the step after the gravity step '''
803     # Simplify
804     GC = 'General_Contact'
805     cp = 'cohesivePeel'
806
807     abqModel.ContactProperty(cp)
808
809     CP_IP = abqModel.interactionProperties[cp]
810     CP_IP.TangentialBehavior(formulation=PENALTY,
811                             directionality=ISOTROPIC,
812                             slipRateDependency=OFF,
813                             pressureDependency=OFF,
814                             temperatureDependency=OFF,
815                             dependencies=0,
816                             table=((0.2, ), ),
817                             shearStressLimit=None,
818                             maximumElasticSlip=FRACTION,
819                             fraction=0.005,
820                             elasticSlipStiffness=None)
821
822     CP_IP.CohesiveBehavior(defaultPenalties=OFF,
823                             table=((Knn, Kss, Ktt), ))
824     # eligibility=INITIAL_NODES,
825
826     CP_IP.Damage(criterion=MAX_STRESS,
827                  initTable=((tn, ts, tt), ),
828                  useEvolution=ON,
829                  evolutionType=ENERGY,
830                  evolTable=((FE, ), ),
831                  useStabilization=ON,
832                  viscosityCoef=1e-05)
833
834     GCI = abqModel.interactions[GC]
835     if gravity == True:
836         GCI.contactPropertyAssignments.changeValuesInStep(stepName=stepName,
837                                                             index=1,
838                                                             value=cp)
839     else:
840         r11=abqModel.rootAssembly.instances['R-1'].surfaces['R_V_Surf_BOND']
841         r12=abqModel.rootAssembly.instances['V-1'].surfaces['V_R_Surf_BOND']
842         GCI.contactPropertyAssignments.appendInStep(stepName=stepName,
843                                                       assignments=((r11, r12,
844                                                                cp), ))
845

```

```

846
847 def smoothGravity():
848     abqModel.SmoothStepAmplitude(name='smoothGravity', timeSpan=STEP,
849         data=((0.0, 0.0), (100.0, 1.0)))
850     abqModel.loads['Gravity'].setValues(amplitude='smoothGravity',
851         distributionType=UNIFORM, field='')
852
853
854 def turnTieCohesive(stepName, cohTieName):
855     ''' Simulate the tie constraint with cohesive surface '''
856     GC = 'General_Contact'
857     CTG = cohTieName # Simplify
858     abqModel.ContactProperty(CTG)
859
860     # Simplify
861     CTG_IP = abqModel.interactionProperties[CTG]
862     GC_IP = abqModel.interactions[GC]
863
864     CTG_IP.CohesiveBehavior(eligibility=INITIAL_NODES)
865     r11=abqModel.rootAssembly.instances['R-1'].surfaces['R_V_Surf_BOND']
866     r12=abqModel.rootAssembly.instances['V-1'].surfaces['V_R_Surf_BOND']
867     GC_IP.contactPropertyAssignments.appendInStep(stepName=stepName,
868         assignments=((r11,
869             r12,
870             CTG), ))
871
872
873 def peelStepPostGravity(time, stepName, previousStep, descrip, scaleFactor,
874     MSTI):
875     ''' step after the gravity phase '''
876     abqModel.ExplicitDynamicsStep(name=stepName,
877         previous=previousStep,
878         description=descrip,
879         timePeriod=time,
880         massScaling=((SEMI_AUTOMATIC,
881             MODEL, AT_BEGINNING,
882             scaleFactor, MSTI, BELOW_MIN,
883             0, 0, 0.0, 0.0, 0, None), ))
884
885
886 def peelTestBCUpdate_With_Gravity(stepName):
887     ''' Update the boundary conditions in the second step, post gravity '''
888     abqModel.loads['Gravity'].setValuesInStep(stepName=stepName,
889         amplitude=FREED)
890     abqModel.boundaryConditions['EHR'].setValuesInStep(stepName=stepName,
891         vr3=-1.0)
892
893
894 def RG_Tie():
895     a = abqModel.rootAssembly
896     region1=a.instances['G-1'].surfaces['G_R_Surf']
897     a = abqModel.rootAssembly
898     region2=a.instances['R-1'].surfaces['R_G_Surf']
899     abqModel.Tie(name='RG',
900         master=region2,
901         slave=region1,
902         positionToleranceMethod=COMPUTED,
903         adjust=OFF,

```

```

904         tieRotations=ON,
905         constraintEnforcement=SURFACE_TO_SURFACE,
906         thickness=ON)
907
908
909 def Amp():
910     abqModel.SmoothStepAmplitude(name='TD_amp', timeSpan=STEP,
911                                   data=((0.0, 0.0),
912                                         (30.0, 2e-05))
913                                   )
914     abqModel.SmoothStepAmplitude(name='omega', timeSpan=STEP,
915                                   data=((0.0, 0.0),
916                                         (30.0, 0.000909174))
917                                   )
918
919
920 def EHR_BC_Fixed(stepName):
921     a = abqModel.rootAssembly
922     region = a.instances['E-1'].sets['E_RP_Set']
923     abqModel.VelocityBC(name='EHR', createStepName=stepName, region=region,
924                          v1=0.0, v2=0.0, v3=0.0, vr1=0.0, vr2=0.0, vr3=0.0,
925                          amplitude='omega', localCsys=None,
926                          distributionType=UNIFORM, fieldName='')
927
928
929 def EHR_BC(stepName):
930     a = abqModel.rootAssembly
931     region = a.instances['E-1'].sets['E_RP_Set']
932     abqModel.VelocityBC(name='EHR', createStepName=stepName, region=region,
933                          v1=0.0, v2=0.0, v3=0.0, vr1=0.0, vr2=0.0, vr3=-1.0,
934                          amplitude='omega', localCsys=None,
935                          distributionType=UNIFORM, fieldName='')
936
937 def GD_BC(stepName):
938     a = abqModel.rootAssembly
939     region = a.instances['G-1'].sets['G_RP_Set']
940     abqModel.VelocityBC(name='GD', createStepName=stepName,
941                          region=region, v1=0.866092, v2=0.499884, v3=0.0,
942                          vr1=0.0, vr2=0.0, vr3=0.0, amplitude='TD_amp',
943                          localCsys=None, distributionType=UNIFORM,
944                          fieldName='')
945
946
947 def TD_BC(stepName):
948     a = abqModel.rootAssembly
949     region = a.instances['T-1'].sets['T_RP_Set']
950     abqModel.VelocityBC(name='TD', createStepName=stepName, region=region,
951                          v1=0.866092, v2=0.499884, v3=0.0,
952                          vr1=0.0, vr2=0.0, vr3=0.0, amplitude='TD_amp',
953                          localCsys=None, distributionType=UNIFORM,
954                          fieldName='')
955
956
957 def Retina_Displacement_BC(stepName):
958     a = abqModel.rootAssembly
959     region = a.instances['R-1'].sets['R_G_Set']
960     abqModel.VelocityBC(name='R_Vel',
961                          createStepName=stepName,

```

```

962         region=region,
963         v1=0.866092,
964         v2=0.499884,
965         v3=UNSET,
966         vr1=UNSET,
967         vr2=UNSET,
968         vr3=UNSET,
969         amplitude='TD_amp',
970         localCsys=None,
971         distributionType=UNIFORM,
972         fieldName='')
973
974 def Gravity(stepName):
975     abqModel.Gravity(name='Gravity', createStepName=stepName, comp2=-9.81,
976                     distributionType=UNIFORM, field='')
977
978
979 def Write_Job(jobName, modelName, jobDescription):
980     mdb.Job(name=jobName,
981            model=modelName,
982            description=jobDescription,
983            type=ANALYSIS,
984            atTime=None,
985            waitMinutes=0,
986            waitHours=0,
987            queue=None,
988            memory=90,
989            memoryUnits=PERCENTAGE,
990            explicitPrecision=DOUBLE,
991            nodalOutputPrecision=SINGLE,
992            echoPrint=OFF,
993            modelPrint=OFF,
994            contactPrint=OFF,
995            historyPrint=OFF,
996            userSubroutine='',
997            scratch='',
998            resultsFormat=ODB,
999            parallelizationMethodExplicit=DOMAIN,
1000            numDomains=14,
1001            activateLoadBalancing=False,
1002            multiprocessingMode=DEFAULT,
1003            numCpus=14)
1004
1005
1006 def Save_INP(jobName):
1007     mdb.jobs[jobName].writeInput(consistencyChecking=OFF)
1008
1009
1010 def VR_Tie():
1011     a = abqModel.rootAssembly
1012     slaveSurf=a.instances['V-1'].surfaces['V_R_Surf_BOND']
1013     mastSurf=a.instances['R-1'].surfaces['R_V_Surf_BOND']
1014     abqModel.Tie(name='RV_Tie',
1015                 master=mastSurf,
1016                 slave=slaveSurf,
1017                 positionToleranceMethod=COMPUTED,
1018                 adjust=OFF,
1019                 tieRotations=ON,

```

```

1020         constraintEnforcement=SURFACE_TO_SURFACE,
1021         thickness=ON)
1022     return '_VR_Tie'
1023
1024
1025 def JobNameFile(modelName, fileNameAttributes, jobDescription):
1026     """
1027     Creates a txt file with the jobNames and all attributes associated with
1028     the model
1029     """
1030     fileName = modelName + fileNameAttributes
1031     outfile = open(fileName+'.txt', 'w')
1032     line = ('The file name indicates what parameters were used to define ' +
1033            'the model\n')
1034     outfile.write(line)
1035     line = '\n' + fileName + '\n'
1036     outfile.write(line)
1037     line = jobDescription
1038     outfile.write(line)
1039     outfile.close()
1040
1041
1042 def Submit_job(jobname):
1043     myJob = mdb.jobs[jobname]
1044     try:
1045         myJob.submit(consistencyChecking=OFF)
1046         myJob.waitForCompletion()
1047     except:
1048         print(str(datetime.datetime.now())+' stop by error!')
1049         pass
1050
1051
1052 def RemoveRigid(stepName):
1053     """ Remove the rigid bodies all together """
1054     a = abqModel.rootAssembly
1055     a.features['T-1'].suppress()
1056     a.features['G-1'].suppress()
1057     abqModel.fieldOutputRequests['Glue_U_RF'].suppress()
1058     # abqModel.historyOutputRequests['Contact_CP-R-G'].suppress()
1059     abqModel.historyOutputRequests['G_RP_Output_U_RF_RM'].suppress()
1060     # abqModel.interactions['CP-R-G'].suppress()
1061     # abqModel.constraints['RG'].suppress()
1062     abqModel.boundaryConditions['GD'].suppress()
1063     abqModel.boundaryConditions['TD'].suppress()
1064     r11=a.instances['E-1'].surfaces['E_Surf']
1065     r12=a.instances['T-1'].surfaces['T_Surf']
1066     r21=a.instances['E-1'].surfaces['E_Surf']
1067     r22=a.instances['G-1'].surfaces['G_Surf']
1068     r31=a.instances['G-1'].surfaces['G_Surf']
1069     r32=a.instances['T-1'].surfaces['T_Surf']
1070
1071     GC = 'General_Contact' # simplify
1072     GCI = abqModel.interactions[GC]
1073     GCI.excludedPairs.setValuesInStep(stepName=stepName,
1074                                       removePairs=((r11, r12),
1075                                                    (r21, r22),
1076                                                    (r31, r32)))
1077     region = a.instances['R-1'].sets['R_G_Set']

```

```

1078     abqModel.VelocityBC(name='R_Vel',
1079                          createStepName=stepName,
1080                          region=region,
1081                          v1=0.866092,
1082                          v2=0.499884,
1083                          v3=UNSET,
1084                          vr1=UNSET,
1085                          vr2=UNSET,
1086                          vr3=UNSET,
1087                          amplitude='TD_amp',
1088                          localCsys=None,
1089                          distributionType=UNIFORM,
1090                          fieldName='')
1091
1092
1093 def CohesiveSurface(Knn, Kss, Ktt, damageInitiation, tn, ts, tt,
1094                    damageEvolution, FE):
1095     coh_int_prop = 'cohesive_IntProp' # simplify
1096     abqModel.ContactProperty(coh_int_prop)
1097     C_IP = abqModel.interactionProperties[coh_int_prop] # simplify code
1098
1099     C_IP.TangentialBehavior(formulation=PENALTY,
1100                            directionality=ISOTROPIC,
1101                            slipRateDependency=OFF,
1102                            pressureDependency=OFF,
1103                            temperatureDependency=OFF,
1104                            dependencies=0,
1105                            table=((0.2, ), ),
1106                            shearStressLimit=None,
1107                            maximumElasticSlip=FRACTION,
1108                            fraction=0.005,
1109                            elasticSlipStiffness=None)
1110
1111     C_IP.CohesiveBehavior(defaultPenalties=OFF,
1112                           table=((Knn,
1113                                   Kss,
1114                                   Ktt), ))
1115
1116     if damageInitiation == True and damageEvolution == True:
1117         # If damage initiation and evolution are both turned on
1118         C_IP.Damage(criterion=MAX_STRESS,
1119                    initTable=((tn,
1120                                ts,
1121                                tt), ),
1122                    useEvolution=ON,
1123                    evolutionType=ENERGY,
1124                    softening=LINEAR,
1125                    evolTable=((FE, ), ),
1126                    useStabilization=ON,
1127                    viscosityCoef=1e-5) # was EXPONENTIAL, LINEAR
1128
1129     elif damageInitiation == True and damageEvolution == False:
1130         # If damage initiation is turned on but evolution is not
1131         C_IP.Damage(criterion=MAX_STRESS,
1132                    initTable=((tn,
1133                                ts,
1134                                tt), ),
1135                    useEvolution=OFF,

```

```

1136         useStabilization=OFF)
1137     else:
1138         print('No damage initiation or evolution')
1139
1140
1141 def FEA():
1142     """
1143     Function that generates FEA code to model vitreoretinal adhesion
1144
1145     # Steps are as follows:
1146         1 - Create new model database
1147         2 - Import SolidWorks STEP file (Includes all parts)
1148         3 - Material property definitions
1149         4 - Part features (Element & Node Sets & Reference Points ...)
1150         5 - Mesh parts (Specify seed size)
1151         6 - Assembly
1152         7 - Step (Dynamic Explicit with Mass Scaling)
1153         8 - Outputs (Field & History)
1154         9 - Contact (General Contact)
1155         10 - Contact pair (Retina/Vitreous - Cohesive Surface)
1156         11 - Tie Constraint (Retina - Glue)
1157         12 - Amplitude definition
1158         13 - BC's'
1159         14 - Submit Job :)
1160     """
1161
1162     # Import SolidWorks STEP file
1163     ImportStepEyeConstrained()
1164
1165     # Mat Props
1166     Retina_Mat_Prop(RetinaProp)
1167     Vitreous_Mat_Prop(VitreousProp)
1168
1169     # # Part Geometry/RPs/Sets/Surfaces
1170     E_Features()
1171     G_Features()
1172     T_Features()
1173     R_Features()
1174
1175     # Internal sphere to reduce mesh
1176     V_Partition_XYZ_Axis()
1177     V_Internal_Sphere()
1178     AssembleV_for_Merging()
1179     mergeV()
1180
1181     # Features on the vitreous
1182     PartitionRetinaOnVitreous()
1183     Vitreous_Features()
1184
1185     # Seed & Mesh parts
1186     E_Mesh(e1Seed, e2Seed) # Max/min
1187     G_Mesh(gSeed)
1188     T_Mesh(ptSeed)
1189     R_Mesh(rSeed)
1190     VitreousMesh(v1Seed, v2Seed)
1191
1192     # Assembly
1193     Assembly()

```

```

1194
1195 # Eliminate the glue and tab from the model
1196 a = abqModel.rootAssembly
1197 a.features['G-1'].suppress()
1198 a.features['T-1'].suppress()
1199
1200 # Gravity Step
1201 previousStep = 'Initial'
1202 if gravity == True:
1203     stepName = 'Gravity_Step'
1204     descrip = ('Applying gravity to the model and letting the ' +
1205               'vitreous and retina settle')
1206
1207     GravityStep(200, previousStep, scaleFactor, 0.03125, stepName, descrip)
1208     Gravity(stepName)
1209     smoothGravity()
1210
1211 # Interactions
1212 cohTieName = 'Cohesive_Gravity_Tie'
1213 General_Contact(stepName, cohTieName)
1214
1215 # Interaction properties
1216 turnTieCohesive(stepName, cohTieName)
1217
1218 # Zero movement boundary conditions
1219 Amp()
1220 EHR_BC_Fixed(stepName)
1221
1222 # Model outputs for gravity step
1223 F_output(stepName)
1224 H_output(stepName)
1225
1226 previousStep = stepName # Update the previous step to be gravity
1227 else:
1228     ''' General contact ''' # fix here if no gravity is specified
1229     peelCoh = 'Cohesive_Peel_Int'
1230     General_Contact(previousStep, peelCoh)
1231
1232 # Peel Step
1233 stepName = 'Peel_Test_Dynamic_Explicit'
1234 descrip = 'Peel the retina away from the vitreous (rotational peel test)'
1235 peelStepPostGravity(time, stepName, previousStep, descrip, scaleFactor,
1236                     MSTI)
1237
1238 updateGeneralContact(stepName, Knn, Kss, Ktt, damageInitiation,
1239                     tn, ts, tt, damageEvolution, FE)
1240
1241 if tieInterface == True:
1242     # Tie the interface together
1243     VR_sym_tie()
1244
1245 # Boundary Conditions
1246 Amp()
1247 if gravity == True:
1248     peelTestBCUpdate_With_Gravity(stepName)
1249 else:
1250     EHR_BC(stepName)
1251     # GD_BC(stepName) # Not used anymore

```



```

1252     # TD_BC(stepName) # Not used anymore
1253
1254     # Model Outputs
1255     F_output(stepName)
1256     H_output(stepName)
1257
1258     Retina_Dispatch_BC(stepName)
1259
1260     # Undo the spacing to pass in the job description
1261     global jobDescription
1262     # replace new lines, spaces, equal signs
1263     jobDescription = jobDescription.replace('NEWLINE', '\n')
1264     jobDescription = jobDescription.replace('TAB', '\t')
1265     jobDescription = jobDescription.replace('SPACE', ' ')
1266     jobDescription = jobDescription.replace('EQUALSSIGN', '=')
1267
1268     Write_Job(jobName, modelName, jobDescription)
1269     print('Job has been written')
1270     Save_INP(jobName)
1271     Submit_job(jobName)
1272     print('Job has been submitted')
1273     del mdb.models['Model-1']
1274
1275     # In[Symmetric Model]
1276     """
1277     Symmetry
1278     """
1279
1280     def VR_sym_tie():
1281         a = abqModel.rootAssembly
1282         mastSurf=a.instances['R-1'].surfaces['R_V_Surf_BOND']
1283         slaveSurf=a.instances['V-1'].surfaces['V_R_Surf_BOND']
1284         abqModel.Tie(name='VR_Tie',
1285                     master=mastSurf,
1286                     slave=slaveSurf,
1287                     positionToleranceMethod=COMPUTED,
1288                     adjust=OFF,
1289                     tieRotations=ON,
1290                     constraintEnforcement=SURFACE_TO_SURFACE,
1291                     thickness=ON)
1292
1293
1294     def E_sym_Constrained():
1295         p = abqModel.parts['E']
1296         c = p.cells
1297         pickedCells = c.getSequenceFromMask(mask=('[#1 ]', ), )
1298         v1, e1, d1 = p.vertices, p.edges, p.datums
1299         p.PartitionCellByPlaneThreePoints(cells=pickedCells,
1300                                         point1=p.InterestingPoint(edge=e1[4],
1301                                                                    rule=MIDDLE),
1302                                         point2=p.InterestingPoint(edge=e1[18],
1303                                                                    rule=MIDDLE),
1304                                         point3=p.InterestingPoint(edge=e1[7],
1305                                                                    rule=MIDDLE))
1306
1307         f = p.faces
1308         p.RemoveFaces(faceList = f[3:4]+f[5:6]+f[7:8]+f[9:12]+f[15:16]+f[17:20]+
1309                             f[21:22]+f[25:26]+f[27:28]+f[29:30], deleteCells=False)

```

```

1310 # Reference point
1311 p.ReferencePoint(point=(0.0, 0.0, 0.0))
1312
1313 r = p.referencePoints
1314 refPoints=(r[4], )
1315 p.Set(referencePoints=refPoints, name='E_RP_Set')
1316
1317 # Sets
1318 # Edge seeds
1319 e = p.edges
1320 edges = e.getSequenceFromMask(mask=('[#ffd03fd0 #131f ]', ), )
1321 p.Set(edges=edges, name='E_Outside_Edge_Seed_Set')
1322 edges = e.getSequenceFromMask(mask=('[#bc007 #c80 ]', ), )
1323 p.Set(edges=edges, name='E_Edge_Seed_Set')
1324
1325 # Define Surface
1326 s = p.faces
1327 side1Faces = s.getSequenceFromMask(mask=('[#1ffff ]', ), )
1328 p.Surface(side1Faces=side1Faces, name='E_Surf')
1329
1330 # Remove cells
1331 c = p.cells
1332 p.RemoveCells(cellList = c[0:1])
1333
1334 # redefine the E set now that the cells have been removed
1335 r = p.referencePoints
1336 refPoints=(r[4], )
1337 p.Set(referencePoints=refPoints, name='E_Set')
1338
1339
1340 def G_sym_Constrained():
1341
1342     p = abqModel.parts['G']
1343     c = p.cells
1344     pickedCells = c.getSequenceFromMask(mask=('[#1 ]', ), )
1345     v, e, d = p.vertices, p.edges, p.datums
1346     p.PartitionCellByPlaneThreePoints(cells=pickedCells,
1347                                       point1=p.InterestingPoint(edge=e[10],
1348                                                                    rule=MIDDLE),
1349                                       point2=p.InterestingPoint(edge=e[11],
1350                                                                    rule=MIDDLE),
1351                                       point3=p.InterestingPoint(edge=e[1],
1352                                                                    rule=MIDDLE))
1353
1354     f1 = p.faces
1355     p.RemoveFaces(faceList = f1[2:3]+f1[4:5]+f1[7:8]+f1[9:11],
1356                  deleteCells=False)
1357
1358     v1, e1, d1, n = p.vertices, p.edges, p.datums, p.nodes
1359     p.ReferencePoint(point=v1[2])
1360
1361     r = p.referencePoints
1362     refPoints=(r[4], )
1363     p.Set(referencePoints=refPoints, name='G_RP_Set')
1364
1365     f = p.faces
1366     faces = f.getSequenceFromMask(mask=('[#2 ]', ), )
1367     p.Set(faces=faces, name='G_T_Set')

```

```

1368
1369 faces = f.getSequenceFromMask(mask=('[#8 ]', ), )
1370 p.Set(faces=faces, name='G_R_Set')
1371
1372 s = p.faces
1373 side1Faces = s.getSequenceFromMask(mask=('[#3f ]', ), )
1374 p.Surface(side1Faces=side1Faces, name='G_Surf')
1375
1376 side1Faces = s.getSequenceFromMask(mask=('[#8 ]', ), )
1377 p.Surface(side1Faces=side1Faces, name='G_R_Surf')
1378
1379 side1Faces = s.getSequenceFromMask(mask=('[#2 ]', ), )
1380 p.Surface(side1Faces=side1Faces, name='G_T_Surf')
1381
1382 # Remove cells
1383 c1 = p.cells
1384 p.RemoveCells(cellList = c1[0:1])
1385
1386 # redefine the set to be the reference point
1387 r = p.referencePoints
1388 refPoints=(r[4], )
1389 p.Set(referencePoints=refPoints, name='G_Set')
1390
1391
1392 def T_sym_constrained():
1393     p = abqModel.parts['T']
1394     c = p.cells
1395     pickedCells = c.getSequenceFromMask(mask=('[#1 ]', ), )
1396     v, e, d = p.vertices, p.edges, p.datums
1397     p.PartitionCellByPlaneThreePoints(cells=pickedCells,
1398                                       point1=p.InterestingPoint(edge=e[11],
1399                                                                    rule=MIDDLE),
1400                                       point2=p.InterestingPoint(edge=e[7],
1401                                                                    rule=MIDDLE),
1402                                       point3=p.InterestingPoint(edge=e[5],
1403                                                                    rule=MIDDLE))
1404
1405     f = p.faces
1406     p.RemoveFaces(faceList = f[1:2]+f[4:5]+f[6:9]+f[11:13], deleteCells=False)
1407
1408     # reference point
1409     v1, e1, d1, n1 = p.vertices, p.edges, p.datums, p.nodes
1410     p.ReferencePoint(point=v1[3])
1411
1412     # Sets
1413     r = p.referencePoints
1414     refPoints=(r[4], )
1415     p.Set(referencePoints=refPoints, name='T_RP_Set')
1416
1417     f = p.faces
1418     faces = f.getSequenceFromMask(mask=('[#2 ]', ), )
1419     p.Set(faces=faces, name='T_G_Set')
1420
1421     # Surfaces
1422     s = p.faces
1423     side1Faces = s.getSequenceFromMask(mask=('[#ff ]', ), )
1424     p.Surface(side1Faces=side1Faces, name='T_Surf')
1425

```

```

1426 side1Faces = s.getSequenceFromMask(mask=('[#2 ]', ), )
1427 p.Surface(side1Faces=side1Faces, name='T_G_Surf')
1428
1429 c = p.cells
1430 p.RemoveCells(cellList = c[0:1])
1431
1432 # Redefine the set to be the reference point
1433 r = p.referencePoints
1434 refPoints=(r[4], )
1435 p.Set(referencePoints=refPoints, name='T_Set')
1436
1437
1438 def R_sym_constrained():
1439
1440     p = abqModel.parts['R']
1441     c = p.cells
1442     pickedCells = c.getSequenceFromMask(mask=('[#1 ]', ), )
1443     v1, e1, d1 = p.vertices, p.edges, p.datums
1444     p.PartitionCellByPlaneThreePoints(cells=pickedCells,
1445                                       point1=p.InterestingPoint(edge=e1[1],
1446                                                                    rule=MIDDLE),
1447                                       point2=p.InterestingPoint(edge=e1[6],
1448                                                                    rule=MIDDLE),
1449                                       point3=p.InterestingPoint(edge=e1[7],
1450                                                                    rule=MIDDLE))
1451
1452     f1 = p.faces
1453     p.RemoveFaces(faceList = f1[1:2]+f1[4:5]+f1[6:9]+f1[11:13],
1454                   deleteCells=False)
1455
1456     c = p.cells
1457     cells = c.getSequenceFromMask(mask=('[#1 ]', ), )
1458     p.Set(cells=cells, name='R_Set')
1459
1460     f = p.faces
1461     faces = f.getSequenceFromMask(mask=('[#6 ]', ), )
1462     p.Set(faces=faces, name='R_G_Set')
1463
1464     faces = f.getSequenceFromMask(mask=('[#10 ]', ), )
1465     p.Set(faces=faces, name='R_V_Set')
1466
1467     faces = f.getSequenceFromMask(mask=('[#1 ]', ), )
1468     p.Set(faces=faces, name='R_SYM_BC_SET')
1469
1470     s = p.faces
1471     side1Faces = s.getSequenceFromMask(mask=('[#ff ]', ), )
1472     p.Surface(side1Faces=side1Faces, name='R_Surf')
1473
1474     side1Faces = s.getSequenceFromMask(mask=('[#6 ]', ), )
1475     p.Surface(side1Faces=side1Faces, name='R_G_Surf')
1476
1477     side1Faces = s.getSequenceFromMask(mask=('[#10 ]', ), )
1478     p.Surface(side1Faces=side1Faces, name='R_V_Surf_BOND')
1479
1480     side1Faces = s.getSequenceFromMask(mask=('[#1 ]', ), )
1481     p.Surface(side1Faces=side1Faces, name='R_SYM_BC_SURF')
1482
1483     # Assign section

```

```

1484     region = p.sets['R_Set']
1485     p.SectionAssignment(region=region, sectionName='Retina_Section',
1486                        offset=0.0, offsetType=MIDDLE_SURFACE, offsetField='',
1487                        thicknessAssignment=FROM_SECTION)
1488
1489
1490 def V_partition_Sphere():
1491     p = abqModel.parts['V']
1492     c = p.cells
1493     pickedCells = c.getSequenceFromMask(mask=('#1 ', ), )
1494     v1, e1, d1 = p.vertices, p.edges, p.datums
1495     p.PartitionCellByPlaneThreePoints(point1=v1[1],
1496                                       point2=v1[0],
1497                                       point3=v1[3],
1498                                       cells=pickedCells)
1499
1500     pickedCells = c.getSequenceFromMask(mask=('#3 ', ), )
1501     v2, e, d2 = p.vertices, p.edges, p.datums
1502     p.PartitionCellByPlaneThreePoints(point1=v2[4],
1503                                       point2=v2[1],
1504                                       point3=v2[5],
1505                                       cells=pickedCells)
1506
1507     pickedCells = c.getSequenceFromMask(mask=('#f ', ), )
1508     v1, e1, d1 = p.vertices, p.edges, p.datums
1509     p.PartitionCellByPlaneThreePoints(point1=v1[2],
1510                                       point2=v1[5],
1511                                       point3=v1[3],
1512                                       cells=pickedCells)
1513
1514
1515 def Assembly_sym_constrain():
1516     a1 = abqModel.rootAssembly
1517     a1.DatumCsysByDefault(CARTESIAN)
1518     p = abqModel.parts['E']
1519     a1.Instance(name='E-1', part=p, dependent=ON)
1520     p = abqModel.parts['G']
1521     a1.Instance(name='G-1', part=p, dependent=ON)
1522     p = abqModel.parts['R']
1523     a1.Instance(name='R-1', part=p, dependent=ON)
1524     p = abqModel.parts['T']
1525     a1.Instance(name='T-1', part=p, dependent=ON)
1526     p = abqModel.parts['V']
1527     a1.Instance(name='V-1', part=p, dependent=ON)
1528     p = abqModel.parts['V_internal']
1529     a1.Instance(name='V_internal-1', part=p, dependent=ON)
1530
1531
1532 def mergeV_sym():
1533     # Merge the vitreous and the internal sphere
1534     a = abqModel.rootAssembly
1535     a.InstanceFromBooleanMerge(name='V_Merge',
1536                               instances=(a.instances['V-1'],
1537                                         a.instances['V_internal-1'], ),
1538                               keepIntersections=ON,
1539                               originalInstances=DELETE,
1540                               domain=GEOMETRY)
1541

```

```

1542 # Clean up file names after merge
1543 del abqModel.parts['V']
1544 del abqModel.parts['V_internal']
1545
1546 abqModel.parts.changeKey(fromName='V_Merge', toName='V')
1547 a = abqModel.rootAssembly
1548 a.regenerate()
1549 abqModel.rootAssembly.features.changeKey(fromName='V_Merge-1',
1550                                           toName='V-1')
1551
1552 p = abqModel.parts['V']
1553 f = p.faces
1554 p.RemoveFaces(faceList = f[0:3]+f[4:5]+f[8:9]+f[12:13]+f[15:16]+
1555                  f[19:21]+f[23:24]+f[26:27]+f[28:29]+f[32:36],
1556                  deleteCells=False)
1557
1558 a.regenerate()
1559
1560
1561 def V_sym_constrained():
1562     # Partition V along the width of the retina
1563     p = abqModel.parts['V']
1564     p.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=-0.00254)
1565     abqModel.parts['V'].features.changeKey(fromName='Datum plane-1',
1566                                           toName='Retina_Width')
1567
1568     c = p.cells
1569     pickedCells = c.getSequenceFromMask(mask=('#14 ', ), )
1570     d1 = p.datums
1571     p.PartitionCellByDatumPlane(datumPlane=d1[3], cells=pickedCells)
1572
1573     # # Left side of vitreous 22.5 degrees
1574     # # Right side of vitreous 18.875000
1575     p.DatumAxisByPrincipalAxis(principalAxis=ZAXIS)
1576     p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset=0.0)
1577
1578     d2 = p.datums
1579     p.DatumPlaneByRotation(plane=d2[6], axis=d2[5], angle=18.75)
1580
1581     d1 = p.datums
1582     ''' angle=-22.5 was the previous angle for the back side of the retina,
1583     because of extreme element deformation, a new model was Created'''
1584     # ang = -22.5 # Previous
1585     ang = -18.75*2 # Updated angle
1586     p.DatumPlaneByRotation(plane=d2[7], axis=d1[5], angle=ang)
1587     c = p.cells
1588     pickedCells = c.getSequenceFromMask(mask=('#2 ', ), )
1589     d2 = p.datums
1590     p.PartitionCellByDatumPlane(datumPlane=d2[7], cells=pickedCells)
1591
1592     pickedCells = c.getSequenceFromMask(mask=('#40 ', ), )
1593     d1 = p.datums
1594     p.PartitionCellByDatumPlane(datumPlane=d1[8], cells=pickedCells)
1595
1596     # Define sets
1597     c = p.cells
1598     cells = c.getSequenceFromMask(mask=('#fff ', ), )
1599     p.Set(cells=cells, name='V_Set')

```

```

1600 f = p.faces
1601 faces = f.getSequenceFromMask(mask=('[#8080 ]', ), )
1602 p.Set(faces=faces, name='V_R_Set')
1603
1604 # Symmetric BC
1605 faces = f.getSequenceFromMask(mask=('[#17000042 #6a ]', ), )
1606 p.Set(faces=faces, name='V_SYM_BC_SET')
1607
1608 # Surfaces
1609 s = p.faces
1610 side1Faces = s.getSequenceFromMask(mask=('[#1700a0ca #7ea ]', ), )
1611 p.Surface(side1Faces=side1Faces, name='V_Surf')
1612 side1Faces = s.getSequenceFromMask(mask=('[#8080 ]', ), )
1613 p.Surface(side1Faces=side1Faces, name='V_R_Surf_BOND')
1614
1615 # Symmetric BC
1616 side1Faces = s.getSequenceFromMask(mask=('[#17000042 #6a ]', ), )
1617 p.Surface(side1Faces=side1Faces, name='V_SYM_BC_SURF')
1618
1619 # Assign section
1620 region = p.sets['V_Set']
1621 p.SectionAssignment(region=region, sectionName='Vitreous_Section',
1622                     offset=0.0, offsetType=MIDDLE_SURFACE, offsetField='',
1623                     thicknessAssignment=FROM_SECTION)
1624
1625
1626 def E_sym_constrain_msh(e1Seed, e2Seed):
1627     p = abqModel.parts['E']
1628     e = p.edges
1629     pickedEdges = e.getSequenceFromMask(mask=('[#bc007 #c80 ]', ), )
1630     p.seedEdgeBySize(edges=pickedEdges,
1631                     size=e1Seed,
1632                     deviationFactor=0.1,
1633                     constraint=FINER)
1634     pickedEdges = e.getSequenceFromMask(mask=('[#ffd03fd0 #131f ]', ), )
1635     p.seedEdgeBySize(edges=pickedEdges,
1636                     size=e2Seed,
1637                     deviationFactor=0.1,
1638                     constraint=FINER)
1639     elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=EXPLICIT)
1640     elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=EXPLICIT)
1641     f = p.faces
1642     faces = f.getSequenceFromMask(mask=('[#1ffff ]', ), )
1643     pickedRegions = (faces, )
1644     p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
1645     # (unique node numbering)
1646     p.setValues(startNodeLabel=1000000, startElemLabel=1000000)
1647     p.generateMesh()
1648
1649
1650 def G_sym_constrain_msh(gSeed):
1651     p = abqModel.parts['G']
1652     p.seedPart(size=gSeed, deviationFactor=0.1, minSizeFactor=0.1)
1653     elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=EXPLICIT)
1654     elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=EXPLICIT)
1655     f = p.faces
1656     faces = f.getSequenceFromMask(mask=('[#3f ]', ), )
1657     pickedRegions = (faces, )

```

```

1658     p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
1659     # (unique node numbering)
1660     p.setValues(startNodeLabel=2000000, startElemLabel=2000000)
1661     p.generateMesh()
1662
1663
1664 def T_sym_constrain_msh(ptSeed):
1665     p = abqModel.parts['T']
1666     p.seedPart(size=ptSeed, deviationFactor=0.1, minSizeFactor=0.1)
1667     elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=EXPLICIT)
1668     elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=EXPLICIT)
1669     f = p.faces
1670     faces = f.getSequenceFromMask(mask=('#ff '), )
1671     pickedRegions = (faces, )
1672     p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
1673     # (unique node numbering)
1674     p.setValues(startNodeLabel=3000000, startElemLabel=3000000)
1675     p.generateMesh()
1676
1677
1678 def R_sym_constrain_msh(rSeed):
1679     p = abqModel.parts['R']
1680     e = p.edges
1681     pickedEdges = e.getSequenceFromMask(mask=('#3ffff '), )
1682     p.seedEdgeBySize(edges=pickedEdges,
1683                      size=rSeed,
1684                      deviationFactor=0.1,
1685                      constraint=FINER)
1686
1687     c = p.cells
1688     pickedRegions = c.getSequenceFromMask(mask=('#1 '), )
1689     p.setMeshControls(regions=pickedRegions, technique=SWEEP,
1690                      algorithm=ADVANCING_FRONT)
1691     c, e1 = p.cells, p.edges
1692     p.setSweepPath(region=c[0], edge=e1[3], sense=REVERSE)
1693     elemType1 = mesh.ElemType(elemCode=C3D8R,
1694                              elemLibrary=EXPLICIT,
1695                              kinematicSplit=AVERAGE_STRAIN,
1696                              secondOrderAccuracy=ON,
1697                              hourglassControl=ENHANCED,
1698                              distortionControl=ON,
1699                              lengthRatio=0.100000001490116)
1700     elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
1701     elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT)
1702     # c = p.cells
1703     cells = c.getSequenceFromMask(mask=('#1 '), )
1704     pickedRegions = (cells, )
1705     p.setElementType(regions=pickedRegions, elemTypes=(elemType1,
1706                                                         elemType2,
1707                                                         elemType3))
1708
1709     # (unique node numbering)
1710     p.setValues(startNodeLabel=4000000, startElemLabel=4000000)
1711     p.generateMesh()
1712
1713
1714 def VseedPart(v2Seed):
1715     ''' Seed the entire vitreous '''
1716     p = abqModel.parts['V']
1717     p.seedPart(size=v2Seed, deviationFactor=0.1, minSizeFactor=0.1)

```



```

1716
1717
1718 def V_SeedTop(v1Seed):
1719     ''' Seed the top of the vitreous where the retina is bonded '''
1720     p = abqModel.parts['V']
1721     e = p.edges
1722     pickedEdges = e.getSequenceFromMask(mask=('[#ffffff #f ]', ), )
1723     p.seedEdgeBySize(edges=pickedEdges,
1724                       size=v1Seed,
1725                       deviationFactor=0.1,
1726                       constraint=FINER)
1727
1728
1729 def vitreous_seed_bias(v1Seed, v2Seed):
1730     ''' Seed the outside edges of the vitreous leading up to the bonded
1731         interface with biased mesh to weight the attachment area '''
1732     p = abqModel.parts['V']
1733     e = p.edges
1734     pickedEdges1 = e.getSequenceFromMask(mask=('[#0 #100040 ]', ), )
1735     pickedEdges2 = e.getSequenceFromMask(mask=('[#0 #400000 ]', ), )
1736     p.seedEdgeByBias(biasMethod=SINGLE,
1737                     end1Edges=pickedEdges1,
1738                     end2Edges=pickedEdges2,
1739                     minSize=v1Seed,
1740                     maxSize=v2Seed,
1741                     constraint=FINER)
1742
1743
1744 def vitreous_Seed_Bottom_Bias(v1Seed, v2Seed):
1745     ''' Seed the bottom of the vitreous '''
1746     p = abqModel.parts['V']
1747     e = p.edges
1748     pickedEdges1 = e.getSequenceFromMask(mask=('[#0 #10000 ]', ), )
1749     pickedEdges2 = e.getSequenceFromMask(mask=('[#0 #1000 ]', ), )
1750     p.seedEdgeByBias(biasMethod=SINGLE,
1751                     end1Edges=pickedEdges1,
1752                     end2Edges=pickedEdges2,
1753                     minSize=v1Seed,
1754                     maxSize=v2Seed,
1755                     constraint=FINER)
1756
1757
1758 def vHex():
1759     ''' Hexahedral mesh definition for the vitreous '''
1760     p = abqModel.parts['V']
1761     elemType1 = mesh.ElemType(elemCode=C3D8R,
1762                               elemLibrary=EXPLICIT,
1763                               kinematicSplit=AVERAGE_STRAIN,
1764                               secondOrderAccuracy=ON,
1765                               hourglassControl=ENHANCED,
1766                               distortionControl=ON,
1767                               lengthRatio=0.100000001490116)
1768     elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
1769     elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT)
1770
1771     c = p.cells
1772     cells = c.getSequenceFromMask(mask=('[#4bf ]', ), )
1773     pickedRegions =(cells, )

```

```

1774     p.setElementType(regions=pickedRegions,
1775                       elemTypes=(elemType1, elemType2, elemType3))
1776
1777
1778 def vTet():
1779     ''' Tetrahedral mesh definition for the vitreous '''
1780     p = abqModel.parts['V']
1781     c = p.cells
1782     pickedRegions = c.getSequenceFromMask(mask=('[#b40 ]', ), )
1783     p.setMeshControls(regions=pickedRegions, elemShape=TET, technique=FREE)
1784     elemType1 = mesh.ElemType(elemCode=C3D20R)
1785     elemType2 = mesh.ElemType(elemCode=C3D15)
1786     elemType3 = mesh.ElemType(elemCode=C3D10)
1787
1788     c = p.cells
1789     cells = c.getSequenceFromMask(mask=('[#b40 ]', ), )
1790     pickedRegions = (cells, )
1791     p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2,
1792                                                         elemType3))
1793     elemType1 = mesh.ElemType(elemCode=C3D8R, elemLibrary=EXPLICIT)
1794     elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
1795     elemType3 = mesh.ElemType(elemCode=C3D4,
1796                               elemLibrary=EXPLICIT,
1797                               secondOrderAccuracy=ON,
1798                               distortionControl=ON,
1799                               lengthRatio=0.100000001490116)
1800
1801     cells = c.getSequenceFromMask(mask=('[#b40 ]', ), )
1802     pickedRegions = (cells, )
1803     p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2,
1804                                                         elemType3))
1805
1806
1807 def V_generate_mesh():
1808     ''' Mesh the vitreous '''
1809     p = abqModel.parts['V']
1810     # (unique node numbering)
1811     p.setValues(startNodeLabel=5000000, startElemLabel=5000000)
1812     p.generateMesh()
1813
1814
1815 def V_sym_constrain_msh(v1Seed, v2Seed):
1816     ''' Mesh the vitreous with the two different seed sizes
1817     Seed the part
1818     Seed the top
1819     bias the edge
1820     seed the bottom
1821     hexahedral elements
1822     tetrahedral elements
1823     generate mesh '''
1824     VseedPart(v2Seed)
1825     V_SeedTop(v1Seed)
1826     vitreous_seed_bias(v1Seed, v2Seed)
1827     vitreous_Seed_Bottom_Bias(v1Seed, v2Seed)
1828     vHex()
1829     vTet()
1830     V_generate_mesh()
1831

```

```

1832
1833
1834 def V_SYM_Constrain_BC(stepName):
1835     a = abqModel.rootAssembly
1836     f = a.instances['V-1'].faces
1837     faces = f.getSequenceFromMask(mask=('[#17000042 #6a ]', ), )
1838     region = a.Set(faces=faces, name='V_SYM_BC_SET')
1839     abqModel.ZsymmBC(name='V_sym',
1840                      createStepName=stepName,
1841                      region=region,
1842                      localCsys=None)
1843
1844
1845 def R_SYM_Constrain_BC(stepName):
1846     a = abqModel.rootAssembly
1847     f = a.instances['R-1'].faces
1848     faces = f.getSequenceFromMask(mask=('[#1 ]', ), )
1849     region = a.Set(faces=faces, name='R_SYM_BC_SET')
1850     abqModel.ZsymmBC(name='R_sym',
1851                      createStepName=stepName,
1852                      region=region,
1853                      localCsys=None)
1854
1855
1856 """ Write the FEA Code """
1857 def FEA_Symmetry():
1858     """
1859     Function that generates FEA code to model vitreoretinal adhesion
1860
1861     # Steps are as follows:
1862     1 - Create new model database
1863     2 - Import SolidWorks STEP file (Includes all parts)
1864     3 - Material property definitions
1865     4 - Part features (Element & Node Sets & Reference Points ...)
1866     5 - Mesh parts (Specify seed size)
1867     6 - Assembly
1868     7 - Step (Dynamic Explicit with Mass Scaling)
1869     8 - Outputs (Field & History)
1870     9 - Contact (General Contact)
1871     10 - Contact pair (Retina/Vitreous - Bonded Surface)
1872     11 - Tie Constraint (Retina - Glue)
1873     12 - Amplitude definition
1874     13 - BC's'
1875     14 - Submit Job :)
1876     """
1877
1878     # Import SolidWorks STEP file
1879     ImportStepEyeConstrained()
1880
1881     # Mat Props
1882     Retina_Mat_Prop(RetinaProp)
1883     Vitreous_Mat_Prop(VitreousProp)
1884
1885     """ Constrained vitreous """
1886     # Pat Geometry/RPs/Sets/Surfaces
1887     E_sym_Constrained()
1888     G_sym_Constrained()
1889     T_sym_constrained()

```

```

1890 R_sym_constrained()
1891
1892 # Define and then merge in the assembly to reduce computational time
1893 V_Internal_Sphere()
1894
1895 # # Assembly
1896 Assembly_sym_constrain()
1897
1898 # partition Vitreous x,y,z plane
1899 V_partition_Sphere()
1900
1901 # Merge V and V Int
1902 mergeV_sym()
1903
1904 # Update V sets
1905 V_sym_constrained()
1906
1907 # Mesh parts
1908 E_sym_constrain_msh(e1Seed, e2Seed)
1909 G_sym_constrain_msh(gSeed)
1910 T_sym_constrain_msh(ptSeed)
1911 V_sym_constrain_msh(v1Seed, v2Seed)
1912 R_sym_constrain_msh(rSeed)
1913
1914 # # Convert Hexahedral elements to quadratic tetrahedral elements
1915 # QuadraticTetVitreous()
1916 # QuadraticTetRetina()
1917
1918 # Eliminate the glue and tab from the model
1919 a = abqModel.rootAssembly
1920 a.features['G-1'].suppress()
1921 a.features['T-1'].suppress()
1922
1923 # Gravity Step
1924 previousStep = 'Initial'
1925 if gravity == True:
1926     stepName = 'Gravity_Step'
1927     descrip = ('Applying gravity to the model and letting the ' +
1928               'vitreous and retina settle')
1929
1930     GravityStep(200, previousStep, scaleFactor, 0.03125, stepName, descrip)
1931     Gravity(stepName)
1932     smoothGravity()
1933
1934 # Interactions
1935 cohTieName = 'Cohesive_Gravity_Tie'
1936 General_Contact(stepName, cohTieName)
1937
1938 # Interaction properties
1939 turnTieCohesive(stepName, cohTieName)
1940
1941 V_SYM_Constrain_BC(stepName)
1942 R_SYM_Constrain_BC(stepName)
1943
1944 # Zero movement boundary conditions
1945 Amp()
1946 EHR_BC_Fixed(stepName)
1947

```

```

1948     ## Model outputs for gravity step
1949     F_output(stepName)
1950     H_output(stepName)
1951
1952     previousStep = stepName # Update the previous step to be gravity
1953 else:
1954     ''' General contact ''' # fix here if no gravity is specified
1955     peelCoh = 'Cohesive_Peel_Int'
1956     General_Contact(previousStep, peelCoh)
1957
1958 ## Peel Step
1959 stepName = 'Peel_Test_Dynamic_Explicit'
1960 descrip = 'Peel the retina away from the vitreous (rotational peel test)'
1961 peelStepPostGravity(time, stepName, previousStep, descrip, scaleFactor,
1962                     MSTI)
1963
1964 updateGeneralContact(stepName, Knn, Kss, Ktt, damageInitiation,
1965                     tn, ts, tt, damageEvolution, FE)
1966
1967 if tieInterface == True:
1968     # Tie the interface together
1969     VR_sym_tie()
1970
1971 # Boundary Conditions
1972 Amp()
1973 if gravity == True:
1974     peelTestBCUpdate_With_Gravity(stepName)
1975
1976 else:
1977     EHR_BC(stepName)
1978     # GD_BC(stepName) # Not used anymore
1979     # TD_BC(stepName) # Not used anymore
1980     V_SYM_Constrain_BC(stepName)
1981     R_SYM_Constrain_BC(stepName)
1982
1983     # Model Outputs
1984     F_output(stepName)
1985     H_output(stepName)
1986
1987 Retina_Displacement_BC(stepName)
1988
1989 # Undo the spacing to pass in the job description
1990 global jobDescription
1991 # replace new lines, spaces, equal signs
1992 jobDescription = jobDescription.replace('NEWLINE', '\n')
1993 jobDescription = jobDescription.replace('TAB', '\t')
1994 jobDescription = jobDescription.replace('SPACE', ' ')
1995 jobDescription = jobDescription.replace('EQUALSSIGN', '=')
1996
1997 Write_Job(jobName, modelName, jobDescription)
1998 print('Job has been written')
1999 Save_INP(jobName)
2000 Submit_job(jobName)
2001 print('Job has been submitted')
2002 del mdb.models['Model-1']
2003
2004
2005 # In[Main import info]

```

```

2006
2007 if __name__ == '__main__':
2008     """ Run the following function """
2009
2010     # Print File of tests & attributes ran in order to make sure they are
2011     # being properly pass through
2012     print("\nWriting out the Argument Data...")
2013     filename = os.path.join(abqWD, 'FEAArgumentData' + '.txt')
2014     outfile = open(filename, 'w')
2015     outfile.write('sys.argv\n')
2016     outfile.write('\n'.join(sys.argv)) # write all arguments passed into abaqus
2017     outfile.close()
2018     print("\nDone!")
2019     print("\nThe output file will be named '{}'.format(filename) + """)
2020     print("\nIt will be in the same working directory as your Abaqus model\n")
2021
2022     # # Testing when importing into abaqus script
2023     # gravity = eval('True') # gravity
2024     # symmetry = eval('False') # symmetry
2025     # simplified= eval('True') # simplified model
2026     # modelName = 'T1Si' # model name
2027     # jobName = 'test' # file name/job name
2028     # time = float('100')
2029     # e1Seed = '[10,1,0.0009765625]'
2030     # e2Seed = '[8,1,0.00390625]'
2031     # ptSeed = '[6,1,0.015625]'
2032     # gSeed = '[7,1,0.0078125]'
2033     # v1Seed = '[10,1,0.0009765625]' #
2034     ↪ '[11.38,1,0.00037521366730664343]' #
2035     # v2Seed = '[8,1,0.00390625]'
2036     # rSeed = '[10,1,0.0009765625]' #
2037     ↪ '[11.3275,1,0.0003891192571059363]' #
2038     # scaleFactor = '[0,1,1]'
2039     # MSTI = '[4,1,0.0625]' # MassScaleTimeIncrement
2040     # RetinaProp = float('11120.0') # Young's modulus for retina
2041     # VitreousProp = float('100') # '69.56549028991259') # Young's modulus for
2042     ↪ vitreous 386.717932801091
2043     # KnnString = str([26.21216496521396,1,77740603.15760481]) #
2044     # KssString = str([27.992885300905385,1,267114916.34363237]) #
2045     # KttString = str([27.65583405906571,1,211463592.90645516])
2046     # damageInitiation = True # True/False
2047     # tnString = str([18.830816653206917,1,466273.4160693089])
2048     # tsString = str([17.49221225177773,1,184365.8917311695]) # Damage
2049     ↪ initiation
2050     # ttString = str([6.5328659715983814,1,92.59523006880973]) # Damage
2051     ↪ initiation
2052     # damageEvolution = True # True/false convert to bool
2053     # FEString = str([-0.9185766704351879,1,0.5290306923507394])
2054     # OptimizationStatus = True
2055     # tieInterface = True
2056     # jobDescription = """Test""" # 'Test MODEL Cube Script'
2057
2058     # Pass in arguments from previous file Strip the brackets from the strings
2059     gravity = eval(sys.argv[-29]) # gravity
2060     symmetry = eval(sys.argv[-28]) # symmetry
2061     simplified = eval(sys.argv[-27]) # simplified model
2062     modelName = sys.argv[-26] # model name

```

```

2059     jobName = sys.argv[-25] # file name/job name
2060     time = float(sys.argv[-24])
2061     e1Seed = sys.argv[-23]
2062     e2Seed = sys.argv[-22]
2063     ptSeed = sys.argv[-21]
2064     gSeed = sys.argv[-20]
2065     v1Seed = sys.argv[-19]
2066     v2Seed = sys.argv[-18]
2067     rSeed = sys.argv[-17]
2068     scaleFactor = sys.argv[-16]
2069     MSTI = sys.argv[-15] # MassScaleTimeIncrement
2070     RetinaProp = float(sys.argv[-14]) # Young's modulus for retina
2071     VitreousProp = float(sys.argv[-13]) # Young's modulus for vitreous
2072     KnnString = sys.argv[-12] # Cohesive behavior
2073     KssString = sys.argv[-11] # Cohesive behavior
2074     KttString = sys.argv[-10] # Cohesive behavior
2075     damageInitiation = eval(sys.argv[-9]) # True/false convert to bool
2076     tnString = sys.argv[-8] # Damage initiation
2077     tsString = sys.argv[-7] # Damage initiation
2078     ttString = sys.argv[-6] # Damage initiation
2079     damageEvolution = eval(sys.argv[-5]) # True/false convert to bool
2080     FEString = sys.argv[-4] # Fracture energy
2081     optimizationStatus = sys.argv[-3] # None/variables to be optimized
2082     tieInterface = eval(sys.argv[-2]) # True/false convert to bool
2083     jobDescription = sys.argv[-1] # String
2084
2085
2086     """ Convert the strings back to lists of floats """
2087     e1SeedStrip = str(e1Seed)[1:-1] # Strip the brackets from the string
2088     e1SeedList = [float(i) for i in e1SeedStrip.split(',')]
2089     e1Seed = e1SeedList[2] # value
2090
2091     e2SeedStrip = str(e2Seed)[1:-1] # Strip the brackets from the string
2092     e2SeedList = [float(i) for i in e2SeedStrip.split(',')]
2093     e2Seed = e2SeedList[2] # value
2094
2095     ptSeedStrip = str(ptSeed)[1:-1] # Strip the brackets from the string
2096     ptSeedList = [float(i) for i in ptSeedStrip.split(',')]
2097     ptSeed = ptSeedList[2] # value
2098
2099     gSeedStrip = str(gSeed)[1:-1] # Strip the brackets from the string
2100     gSeedList = [float(i) for i in gSeedStrip.split(',')]
2101     gSeed = gSeedList[2] # value
2102
2103     v1SeedStrip = str(v1Seed)[1:-1] # Strip the brackets from the string
2104     v1SeedList = [float(i) for i in v1SeedStrip.split(',')]
2105     v1Seed = v1SeedList[2] # value
2106
2107     v2SeedStrip = str(v2Seed)[1:-1] # Strip the brackets from the string
2108     v2SeedList = [float(i) for i in v2SeedStrip.split(',')]
2109     v2Seed = v2SeedList[2] # value
2110
2111     rSeedStrip = str(rSeed)[1:-1] # Strip the brackets from the string
2112     rSeedList = [float(i) for i in rSeedStrip.split(',')]
2113     rSeed = rSeedList[2] # value
2114
2115     # Strip the brackets from the string
2116     scaleFactorStrip = str(scaleFactor)[1:-1]

```

```

2117     scaleFactorList = [float(i) for i in scaleFactorStrip.split(',')]
2118     scaleFactor = scaleFactorList[2] # value
2119
2120     # Strip the brackets from the string
2121     # MassScaleTimeIncrement
2122     MSTIStrip = str(MSTI)[1:-1]
2123     MSTIList = [float(i) for i in MSTIStrip.split(',')]
2124     MSTI = MSTIList[2] # value
2125
2126     KnnStrip = str(KnnString)[1:-1] # Strip the brackets from the string
2127     KnnList = [float(i) for i in KnnStrip.split(',')]
2128     Knn = KnnList[2] # value
2129
2130     KssStrip = str(KssString)[1:-1] # Strip the brackets from the string
2131     KssList = [float(i) for i in KssStrip.split(',')]
2132     Kss = KssList[2] # value
2133
2134     KttStrip = str(KttString)[1:-1] # Strip the brackets from the string
2135     KttList = [float(i) for i in KttStrip.split(',')]
2136     Ktt = KttList[2] # value
2137
2138     tnStrip = str(tnString)[1:-1] # Strip the brackets from the string
2139     tnList = [float(i) for i in tnStrip.split(',')]
2140     tn = tnList[2] # value
2141
2142     tsStrip = str(tsString)[1:-1] # Strip the brackets from the string
2143     tsList = [float(i) for i in tsStrip.split(',')]
2144     ts = tsList[2] # value
2145
2146     ttStrip = str(ttString)[1:-1] # Strip the brackets from the string
2147     ttList = [float(i) for i in ttStrip.split(',')]
2148     tt = ttList[2] # value
2149
2150     FEStrip = str(FESString)[1:-1] # Strip the brackets from the string
2151     FEList = [float(i) for i in FEStrip.split(',')]
2152     FE = FEList[2] # value
2153
2154     """ Write the FEA Code """
2155     Mdb()
2156     modelDescription = ('Measure adhesion between the retina & vitreous of ' +
2157                        'the human eye')
2158     abqModel = mdb.Model(name=modelName,
2159                          description=modelDescription,
2160                          modelType=STANDARD_EXPLICIT,
2161                          copyInteractions=ON,
2162                          copyConstraints=ON)
2163
2164     if symmetry == True:
2165         print('FEA SYM model')
2166         FEA_Symmetry()
2167     else:
2168         print('FEA Non-SYM model')
2169         FEA()

```



## 1.6.4 Abaqus Extract Data Script

</> Script 15: Python script used to extract data from the output database file (.odb). </>

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Fri Jan 29 15:36:37 2021
4
5 @author: Kiffer Creveling
6 Instructions:
7     1) Save this script in a folder containing your ODB file
8     2) Open a command window and navigate to your directory containing this
9        script and your ODB file
10    3) Create a .bat file
11    3) Issue the command to call the script and extract data:
12        abaqus python -c "import BpT; BpT.data_extract('xxxxxx.odb')"
13 """
14 # *****
15 from odbAccess import *
16 import odbAccess as oa
17 from sys import argv, exit
18 from abaqusConstants import *
19 from textRepr import *
20 import pdb
21 import numpy as np
22 import os
23
24 """ Pass arguments into this script """
25 # Arguments from the previous script
26 script = sys.argv[0]
27 jobName = sys.argv[1]
28 gravity = eval(sys.argv[2]) # True/False
29 symmetry = eval(sys.argv[3]) # True/False
30 simplified = eval(sys.argv[4]) # True/False
31 DMGInitiation = eval(sys.argv[5]) # True/False
32 DMGEvolution = eval(sys.argv[6]) # True/False # not used in the extraction
33
34 def openOdb(jobName):
35     """
36     Function used to locate the .odb given a file name
37
38     Parameters
39     -----
40     jobName : Name of the ABAQUS .odb file
41
42     Returns
43     -----
44     odb : Abaqus output file
45     """
46     if jobName.endswith('.odb'):
47         odbFile = jobName
48         try:
49             odb=oa.openOdb(path=odbFile, readOnly=TRUE)
50             print("\nOpening the odb file... (.odb was specified)")
51             return odb
52         except:
53             print("ERROR: Unable to open the specified odb %s. Exiting."
```

```

54         % odbFile)
55     exit(0)
56
57 else:
58     odbFile = jobName + '.odb'
59     # Try opening the odb file
60     try:
61         odb=oa.openOdb(path=odbFile, readOnly=TRUE)
62         print("\nOpening the odb file... (Searching for .odb)")
63         return odb
64     except:
65         print("ERROR: Unable to open the specified odb %s. Exiting."
66               % odbFile)
67         exit(0)
68
69 def data_extract(jobName):
70     """
71     Function used to extract data from the .odb file
72
73     Parameters
74     -----
75     jobName : The name of ABAQUS .odb file
76
77     Returns
78     -----
79     Two files of data used for plotting
80     """
81
82     # due to symmetry multiply the values by 2
83     if symmetry == True:
84         mult = 2
85     else:
86         mult = 1
87
88     frames = []
89     try:
90         odb = openOdb(jobName)
91     except:
92         print(os.getcwd())
93         print("Looks like there is a problem with the job name or odb file")
94
95     theta = 30
96     LoadCellDirection = [np.cos(theta*np.pi/180), np.sin(theta*np.pi/180), 0]
97
98     """ Field Output data arrays """
99     RF = []
100
101     # vector components of the reaction force
102     RFx = []
103     RFy = []
104     RFz = []
105
106     U_top = [] # values to append
107     U_bot = [] # values to append
108     Nforc = []
109
110     # Used to calculate bond distance
111     R_bot = [] # bottom of retina

```

```

112 V_top = [] # top of vitreous
113 Bond_disp = [] # Bond separation distance
114
115 Stress = [] # Stress
116
117 CSDMG = [] # Damage variable for cohesive surfaces in general contact.
118 # Maximum stress-based damage initiation criterion for cohesive surfaces
119 # in general contact.
120 CSMAXSCRT = []
121
122 CSDMG_List = [] # values
123 CSMAXSCRT_List = [] # values
124
125 CSDMG_Nodes = [] # nodes
126 CSMAXSCRT_Nodes = [] # nodes
127
128 frames = [] # List of frames
129 time = [] # Time array
130
131 # Used for reaction force simplicity further in the code
132 # Temporary array used for iterating (Clears after each iteration)
133 temp = []
134 tempx = []
135 tempy = []
136 tempz = []
137
138 """ History Output data arrays """
139 Hist_Time = []
140 IE = []
141 KE = []
142
143 """ Loop over the field outputs """
144 # determines the step in the abaqus odb file (typically displacement)
145 step = odb.steps.keys()
146
147 if gravity != True:
148     disp_step = step[0] # Defines the step as a variable name
149 else:
150     # Step that includes the gravity kinetic energy settling
151     disp_step = step[1]
152
153 for frame, odbFrame in enumerate(odb.steps[disp_step].frames):
154     frames.append(frame) # Construct a list of all of the frames
155
156     """ Extract ODB fieldOutputs """
157     fieldOutput = odbFrame.fieldOutputs
158
159     # Print the time during the simulation
160     print(odbFrame.description)
161     time.append(odbFrame.frameValue)
162
163     """ Abaqus Instances (Parts) """
164     odbInstance = odb.rootAssembly.instances
165
166     if simplified == False:
167         # If Simp is not in the title
168         # Parts
169         E = odbInstance.keys(0)[0]

```

```

170         G = odbInstance.keys(0)[1]
171         R = odbInstance.keys(0)[2]
172         T = odbInstance.keys(0)[3]
173         V = odbInstance.keys(0)[4]
174
175     elif simplified == True:
176         # If simplification exists, omit the glue & tab
177         E = odbInstance.keys(0)[0]
178         R = odbInstance.keys(0)[1]
179         V = odbInstance.keys(0)[2]
180     else:
181         print('Error in part definitions')
182
183     """ Nodal displacements """
184     f0_U = fieldOutput['U'] # displacements
185
186     if simplified == False:
187         # If Simp is not in the title
188
189         # Glue
190         Displacements = f0_U.getSubset(region=odbInstance[G]
191                                         .nodeSets['G_RP_SET'])
192         # Loops over each node in the "SET" defined by the displacement
193         for Uyi in Displacements.values:
194             Uyi_vec = [Uyi.data[0], Uyi.data[1], Uyi.data[2]]
195             # Find the magnitude
196             # Creates a list of displacements in the "SET"
197             temp.append(np.dot(Uyi_vec, LoadCellDirection))
198
199             # Sums up the list of displacements from the "SET"
200             SU = np.sum(temp)
201             # Divide by the number of nodes in the set to get average
202             AvgU_top = SU/len(temp)
203             # Adds the total displacement to the U-array by summing across
204             # each step
205             U_top.append(AvgU_top)
206             temp = [] # Clear the array for the next iteration in the loop
207
208     elif simplified == True:
209         # If simplification exists, omit the values
210
211         Displacements = f0_U.getSubset(region=odbInstance[R]
212                                         .nodeSets['R_G_SET'])
213         # Loops over each node in the "SET" defined by the displacement
214         for Uyi in Displacements.values:
215             Uyi_vec = [Uyi.data[0], Uyi.data[1], Uyi.data[2]]
216             # Find the magnitude
217             # Creates a list of displacements in the "SET"
218             temp.append(np.dot(Uyi_vec, LoadCellDirection))
219
220             # Sums up the list of displacements from the "SET"
221             SU = np.sum(temp)
222             # Divide by the number of nodes in the set to get average
223             AvgU_top = SU/len(temp)
224             # Adds the total displacement to the U-array by summing across
225             # each step
226             U_top.append(AvgU_top)
227             temp = [] # Clear the array for the next iteration in the loop

```

```

228
229 else:
230     print('Error in nodal displacements')
231
232     """ Bond Distance """
233     Displacements = f0_U.getSubset(region=odbInstance[R]
234                                     .nodeSets['R_V_SET'])
235     # Loops over each node in the "SET" defined by the displacement
236     for Uyi in Displacements.values:
237         Uyi_vec = [Uyi.data[0], Uyi.data[1], Uyi.data[2]]
238         # Find the magnitude
239         # Creates a list of displacements in the "SET"
240         temp.append(np.dot(Uyi_vec, LoadCellDirection))
241
242     # Sums up the list of displacements from the "SET"
243     SU = np.sum(temp)
244     # Divide by the number of nodes in the set to get average
245     AvgR_bot = SU/len(temp)
246     # Adds the total displacement to the U-array by summing across
247     # each step
248     R_bot.append(AvgR_bot)
249     temp = [] # Clear the array for the next iteration in the loop
250
251     Displacements = f0_U.getSubset(region=odbInstance[V]
252                                     .nodeSets['V_R_SET'])
253     # Loops over each node in the "SET" defined by the displacement
254     for Uyi in Displacements.values:
255         Uyi_vec = [Uyi.data[0], Uyi.data[1], Uyi.data[2]]
256         # Find the magnitude
257         # Creates a list of displacements in the "SET"
258         temp.append(np.dot(Uyi_vec, LoadCellDirection))
259
260     # Sums up the list of displacements from the "SET"
261     SU = np.sum(temp)
262     # Divide by the number of nodes in the set to get average
263     AvgV_top = SU/len(temp)
264     # Adds the total displacement to the U-array by summing across
265     # each step
266     V_top.append(AvgV_top)
267     temp = [] # Clear the array for the next iteration in the loop
268
269     # average difference in nodal positions between the *bonded surfaces
270     Bond_disp.append(AvgR_bot - AvgV_top)
271
272     """ Cohesive Info """
273     if DMGInitiation == True:
274         # fieldObject_CSMA_XCRT
275         f0_CMS = fieldOutput['CSMA_XCRT General_Contact_Domain']
276
277         # Specify only the bonded interface
278         BONDED_Surface_R_CMS = f0_CMS.getSubset(region=odbInstance[R]
279                                                 .nodeSets['R_V_SET'])
280         BONDED_Surface_V_CMS = f0_CMS.getSubset(region=odbInstance[V]
281                                                 .nodeSets['V_R_SET'])
282
283         """ Contact initiation for cohesive surfaces """
284         # Retina-Vitreous cohesive initiation value
285

```

```

286     # Loop over all retina nodes
287     for CSMAXSCRT_i in BONDED_Surface_R_CMS.values:
288         temp.append(CSMAXSCRT_i.data) # nodal value
289         if frame == 0:
290             CSMAXSCRT_Nodes.append(CSMAXSCRT_i.nodeLabel)
291
292     # Loop over all vitreous nodes
293     for CSMAXSCRT_i in BONDED_Surface_V_CMS.values:
294         temp.append(CSMAXSCRT_i.data) # nodal value
295         if frame == 0:
296             CSMAXSCRT_Nodes.append(CSMAXSCRT_i.nodeLabel)
297
298     # Mean of the list of initiation values from the "SET"
299     Mean_CMS = np.mean(temp)
300     # append the list of nodal values
301     CSMAXSCRT_List.append(temp)
302     # Adds the average value to the array by summing across each step
303     CSMAXSCRT.append(Mean_CMS)
304     temp = [] # Clear the array for the next iteration in the loop
305
306 else:
307     print('No cohesive initiation info to update... ** Updating ' +
308           'with nans')
309     CSMAXSCRT_List.append(np.nan)
310     CSMAXSCRT.append(np.nan)
311     CSMAXSCRT_Nodes.append(np.nan)
312
313 if DMGEvolution == True:
314     # fieldObject_CSDMG
315     f0_CDG = fieldOutput['CSDMG General_Contact_Domain']
316
317     # Specify only the bonded interface
318     BONDED_Surface_R_CSDMG = f0_CDG.getSubset(region=odbInstance[R]
319                                                .nodeSets['R_V_SET'])
320     BONDED_Surface_V_CSDMG = f0_CDG.getSubset(region=odbInstance[V]
321                                                .nodeSets['V_R_SET'])
322
323     """ Contact damage for cohesive surfaces """
324     # Retina-Vitreous cohesive damage value
325
326     # Loop over all retina nodes
327     for CSDMG_i in BONDED_Surface_R_CSDMG.values:
328         temp.append(CSDMG_i.data)
329         if frame == 0:
330             CSDMG_Nodes.append(CSDMG_i.nodeLabel)
331
332     # Loop over all vitreous nodes
333     for CSDMG_i in BONDED_Surface_V_CSDMG.values:
334         temp.append(CSDMG_i.data)
335         if frame == 0:
336             CSDMG_Nodes.append(CSDMG_i.nodeLabel)
337
338     # Mean of the list of initiation values from the "SET"
339     Mean_CSDMG = np.mean(temp)
340     # append the list of nodal values
341     CSDMG_List.append(temp)
342     # Adds the average value to the array by summing across each step
343     CSDMG.append(Mean_CSDMG)

```

```

344     temp = [] # Clear the array for the next iteration in the loop
345 else:
346     print('No cohesive damage info to update... ** Updating with nans')
347     CSDMG_List.append(np.nan)
348     CSDMG.append(np.nan)
349     CSDMG_Nodes.append(np.nan)
350
351     """ Contact Node Lists """
352     R_V_SetNodeNames = []
353     V_R_SetNodeNames = []
354     for i, NodeLabeli in enumerate(odbiInstance[R]
355                                     .nodeSets['R_V_SET'].nodes):
356         R_V_SetNodeNames.append(NodeLabeli.label)
357
358     for i, NodeLabeli in enumerate(odbiInstance[V]
359                                     .nodeSets['V_R_SET'].nodes):
360         V_R_SetNodeNames.append(NodeLabeli.label)
361
362     """ Reaction forces """
363     f0_RF = fieldOutput['RF'] # reaction forces
364     if simplified == False:
365         # If Simp is not in the title
366
367         # Glue-Retina G_RP_Set Reaction forces
368         Reaction_Forces = f0_RF.getSubset(region=odbiInstance[G]
369                                           .nodeSets['G_RP_SET'])
370
371     elif simplified == True:
372
373         # Retina R_G_Set Reaction forces
374         Reaction_Forces = f0_RF.getSubset(region=odbiInstance[R]
375                                           .nodeSets['R_G_SET'])
376
377     else:
378         print('Error in RF output')
379
380     # Loops over each node in the "SET" defined by the reaction force
381     for RFi in Reaction_Forces.values:
382         RFxi = RFi.data[0]
383         RFyi = RFi.data[1]
384         RFzi = RFi.data[2]
385         RFi_vec = [RFxi, RFyi, RFzi]
386
387         # Find the component in the direction of the load cell
388         # Creates a list of reaction forces in the "SET"
389         temp.append(np.dot(RFi_vec, LoadCellDirection)*mult)
390         tempx.append(RFxi*mult) # X reaction forces along the R_G_SET
391         tempy.append(RFyi*mult) # Y reaction forces along the R_G_SET
392         tempz.append(RFzi*mult) # Z reaction forces along the R_G_SET
393
394     SRF = np.sum(temp) # Sums up the list of reaction forces from the "SET"
395     # Adds the total reaction force to the RF-array by summing across
396     # each step
397     RF.append(SRF)
398     temp = [] # Clear the array for the next iteration in the loop
399
400     SRFX = np.sum(tempx)
401     RFX.append(SRFX)

```

```

402 SRFY = np.sum(tempy)
403 RFy.append(SRFY)
404
405 SRFZ = np.sum(tempz)
406 RFz.append(SRFZ)
407
408
409 """ Nodal Forces """
410 ''' Forces at the nodes of an element from both the hourglass and the
411 regular deformation modes of that element (negative of the internal
412 forces in the global coordinate system). The specified position in
413 data and results file requests is ignored.'''
414
415 # Searches if the repository has the value
416 if fieldOutput.has_key('NFORC1') == 1:
417     f0_NFORC1 = fieldOutput['NFORC1'] # Normal force 1
418     f0_NFORC2 = fieldOutput['NFORC2'] # Normal force 2
419     f0_NFORC3 = fieldOutput['NFORC3'] # Normal force 3
420
421     # Retina nodal forces on the glue interface
422     nodeSet_R_G_SET = odbInstance[R].nodeSets['R_G_SET']
423     NF1 = f0_NFORC1.getSubset(region=nodeSet_R_G_SET)
424     NF2 = f0_NFORC2.getSubset(region=nodeSet_R_G_SET)
425     NF3 = f0_NFORC3.getSubset(region=nodeSet_R_G_SET)
426
427     # Loops over each node in the "SET" defined by the reaction force
428     for NFi in range(len(NF1.values)):
429         NFi_vec = [NF1.values[NFi].data,
430                   NF2.values[NFi].data,
431                   NF3.values[NFi].data]
432         NFi_veclabel = [NF1.values[NFi].nodeLabel,
433                        NF1.values[NFi].data,
434                        NF2.values[NFi].nodeLabel,
435                        NF2.values[NFi].data,
436                        NF3.values[NFi].nodeLabel,
437                        NF3.values[NFi].data]
438         # Find the component in the direction of the load cell
439         # Creates a list of reaction forces in the "SET"
440         temp.append(np.dot(NFi_vec, LoadCellDirection)*mult)
441
442     # Sums up the list of reaction forces from the "SET"
443     SNf = np.sum(temp)
444     # Adds the total reaction force to the RF-array by summing across
445     # each step (negative indicates the direction, which is opposite
446     # of tension when -1)
447     Nforc.append(SNf*-1)
448     temp = [] # Clear the array for the next iteration in the loop
449 else:
450     Nforc.append(0)
451     print('No NFORC... ** Updating with 0')
452
453 """ Stress """
454 f0_S = fieldOutput['S'] # stress
455 # Glue-Retina set-forces
456 # Loops over each node in the "SET" defined by the reaction force
457 for Si in f0_S.values:
458     stress_vec = [Si.data[0], Si.data[1], Si.data[2]]
459     # Append the component of stress in the load cell direction

```



```

460         Stress.append(np.dot(stress_vec, LoadCellDirection))
461
462     # In[History Output]
463     """ Loop over the history outputs """
464     # List all of the items in the dictionary
465     # odb.steps[disp_step].historyRegions.keys()
466     odbHistoryRegion = odb.steps[disp_step].historyRegions
467     odbHistAssem = 'Assembly ASSEMBLY'
468     Assembly = odbHistoryRegion[odbHistAssem]
469
470     # Energy output
471     ALLIE_KE = Assembly.historyOutputs.keys()[0]
472     Hist_ELEM = Assembly.historyOutputs.keys()[1]
473     Whole_Model_Energy = Assembly.historyOutputs
474     Internal_Energy = Whole_Model_Energy.keys()[0] # Internal energy
475     Kinetic_Energy = Whole_Model_Energy.keys()[1] # Kintic energy
476     for i,j in enumerate(Whole_Model_Energy[Internal_Energy].data):
477         Hist_Time.append(j[0]) # History Output Time Array
478         IE.append(j[1]) # Internal Energy
479         # Kinetic Energy
480         KE.append(Whole_Model_Energy[Kinetic_Energy].data[i][1])
481
482     # Glue Reference point
483     if simplified == False:
484         # If Simp is not in the title
485
486         odbHist_gRP = odbHistoryRegion.keys()[1]
487         gRP_Hist = odbHistoryRegion[odbHist_gRP]
488         gRP_Hist = gRP_Hist.historyOutputs
489         gRP_HistRF1 = gRP_Hist.keys()[0]
490         gRP_HistRF2 = gRP_Hist.keys()[1]
491         gRP_HistRF3 = gRP_Hist.keys()[2]
492         gRP_HistU1 = gRP_Hist.keys()[6]
493         gRP_HistU2 = gRP_Hist.keys()[7]
494         gRP_HistU3 = gRP_Hist.keys()[8]
495
496     elif simplified == True:
497         # If simplification, omit the tab and glue
498         print('Simplification')
499     else:
500         print('Error in simplification')
501
502     # In[Print Field Outputs]
503     """ Specify folder name where the files go... """
504     folderName = jobName
505     folder_sub_directory = 'Output'
506
507     """ Print the odbFieldOutput Data """
508     print("\nWriting out the load data...")
509     filename = os.path.join(folderName, folder_sub_directory,
510                             'output_Field_' + jobName + '.txt')
511     outfile = open(filename, 'w')
512
513     Header = [] # Header information for the dataframe
514     Header.append('frame')
515     Header.append('Time [s]')
516     Header.append('Reaction force dotted in y direction [N]')
517     Header.append('Reaction force X [N]')

```

```

518 Header.append('Reaction force Y [N]')
519 Header.append('Reaction force Z [N]')
520 Header.append('Sum Nodal Force [N]')
521 Header.append('Glue Displacements [m]')
522 Header.append('Bond Displacements [m]')
523 Header.append('Stress [Pa]')
524 Header.append('AVG CSMAXSCRT')
525 Header.append('AVG CSDMG')
526
527 lineWrite = '\t'.join(str(item) for item in Header)
528 outfile.write(lineWrite)
529
530 for i in frames:
531
532     lineNums = []
533     lineNums.append(time[i])
534     lineNums.append(RF[i])
535     lineNums.append(RFx[i])
536     lineNums.append(RFy[i])
537     lineNums.append(RFz[i])
538     lineNums.append(Nforc[i])
539     lineNums.append(U_top[i])
540     lineNums.append(Bond_disp[i])
541     lineNums.append(Stress[i])
542     lineNums.append(CSMAXSCRT[i])
543     lineNums.append(CSDMG[i])
544
545     # format the list to have a float with twenty decimal places
546     # Add floats
547     formatted_list = ['{: .20f}'.format(item) for item in lineNums]
548     line = '\n' + '{}\t'.format(i) + '\t'.join(str(item)
549                                                 for item in formatted_list)
550     outfile.write(line)
551
552 outfile.close()
553
554 print("\nDone!")
555 print("\nThe output file will be named {}".format(filename) + "")
556 print("\nIt will be in the same working directory as your Abaqus" +
557       " model\n")
558
559 # In[Print History Output]
560 """ Print the odbHistoryOutput Data """
561 print("\nWriting out the History Output data...")
562 filename = os.path.join(folderName, 'Output', 'output_History_' +
563                          jobName + '.txt')
564 outfile = open(filename, 'w')
565
566 Header = []
567 Header.append('frame')
568 Header.append('Time [s]')
569 Header.append('Internal Energy [J]')
570 Header.append('Kinetic Energy [J]')
571 lineWrite = '\t'.join(str(item) for item in Header)
572 outfile.write(lineWrite)
573
574 for i, j in enumerate(Hist_Time):
575     line = []

```

```

576     line.append('{}'.format(i)) # Integer for frame number
577     line.append('{:.10f}'.format(j))
578     line.append('{:.30f}'.format(IE[i]))
579     line.append('{:.30f}'.format(KE[i]))
580     lineWrite = '\n' + '\t'.join(str(item) for item in line)
581     outfile.write(lineWrite)
582
583     outfile.close()
584
585     print("\nDone!")
586     print("\nThe output file will be named '{}'.format(filename) + """)
587     print("\nIt will be in the same working directory as your Abaqus" +
588           " model\n")
589     # In[DMG Criteria]
590     if DMGInitiation == True:
591         """ Print the CSMAXSCRT Data """
592         print("\nWriting out the Field Output CSMAXSCRT data...")
593         filename = os.path.join(folderName, 'Output', 'CSMAXSCRT_' +
594                                 jobName + '.txt')
595         outfile = open(filename, 'w')
596         outfile.write('Time (s)\t' + '\t'.join(str(item)
597                                                for item in CSMAXSCRT_Nodes))
598         for i, j in enumerate(CSMAXSCRT_List):
599             outfile.write('\n')
600             tempList = [time[i]]
601             for k in list(j):
602                 tempList.append(k)
603             outfile.write('\t'.join(str(item) for item in tempList))
604         outfile.close()
605         print("\nDone!")
606         print("\nThe output file will be named '{}'.format(filename) + """)
607         print("\nIt will be in the same working directory as your Abaqus" +
608               " model\n")
609     if DMGEvolution == True:
610         """ Print the CSDMG Data """
611         print("\nWriting out the Field Output CSDMG data...")
612         filename = os.path.join(folderName, 'Output', 'CSDMG_' +
613                                 jobName + '.txt')
614         outfile = open(filename, 'w')
615         outfile.write('Time (s)\t' + '\t'.join(str(item)
616                                                for item in CSDMG_Nodes))
617         for i, j in enumerate(CSDMG_List):
618             outfile.write('\n')
619             tempList = [time[i]]
620             for k in list(j):
621                 tempList.append(k)
622             outfile.write('\t'.join(str(item) for item in tempList))
623         outfile.close()
624         print("\nDone!")
625         print("\nThe output file will be named '{}'.format(filename) + """)
626         print("\nIt will be in the same working directory as your Abaqus" +
627               " model\n")
628     return
629
630 # Run the function
631 data_extract(jobName)

```

## 1.6.5 Plotting Script

</> Script 16: Python script used to create plots for each simulation. </>

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Fri Jan 29 23:49:03 2021
4
5 @author: Kiffer Creveling
6 python3
7
8 """
9 # Packages & path folder
10 #from sys import argv, exit
11 #sys.path.append(r'F:\Abaqus Working Directory')
12 import pandas as pd
13 import matplotlib.pyplot as plt
14 from matplotlib.pyplot import cm
15 import matplotlib.path_effects as pe
16 import numpy as np
17 import os
18 import os.path
19 import sys
20 import pdb
21 plt.rcParams['figure.figsize'] = [16, 9]
22
23 def plot_Field_Output(fileName, dataDirectory, dataCompare,
24                      DMGInitiation, DMGEvolution):
25
26     """ Field Output Data """
27     df = pd.read_csv(os.path.join(dataDirectory, fileName),
28                     sep="\t", header=0)
29
30     Header = [] # Header information for the dataframe
31     Header.append('Frame')
32     Header.append('Time')
33     Header.append('RF_y_dot')
34     Header.append('RFx')
35     Header.append('RFy')
36     Header.append('RFz')
37     Header.append('Nodal_Force')
38     Header.append('Tab_Displacement')
39     Header.append('Bond_Displacement')
40     Header.append('Stress')
41     Header.append('AVG_CSMAXSCRT')
42     Header.append('AVG_CSDMG')
43
44     df.columns = Header
45
46     t = df.Time
47     RF = df.RF_y_dot*1e3 # Convert from N to mN
48     NF = df.Nodal_Force*1e3 # Convert from N to mN
49     TD = df.Tab_Displacement*1e3 # convert from N to mN
50     B = df.Bond_Displacement*1e3 # convert from N to mN
51     S = df.Stress
52     AVG_CSMAXSCRT = df.AVG_CSMAXSCRT
53     AVG_CSDMG = df.AVG_CSDMG
```

```

54
55 (figureName, ext) = os.path.splitext(fileName) # Split the file extension
56
57 """ Read in the csv file """
58 dfValsn = pd.read_csv(dataCompare, sep="\t", nrows=22, header=None,
59                        names=['Var', 'Attribute'])
60
61 """ File Attributes """
62 HID = dfValsn['Attribute'][0]
63 HAGE = dfValsn['Attribute'][1]
64 HG = dfValsn['Attribute'][2]
65 HLR = dfValsn['Attribute'][3]
66 HR = dfValsn['Attribute'][4]
67 HSSi = float(dfValsn['Attribute'][12])
68 HSSf = float(dfValsn['Attribute'][13])
69 HTMax = float(dfValsn['Attribute'][14])
70 HDispMax = float(dfValsn['Attribute'][15])
71 HFMax = float(dfValsn['Attribute'][16]) # (mN)
72 HFSS = float(dfValsn['Attribute'][17])
73 # (mN/m) slope from 20 seconds prior to max force value
74 HSlope20 = float(dfValsn['Attribute'][20])
75
76 dfn = pd.read_csv(os.path.join(dataCompare), sep="\t", header=30)
77 dfn.columns = ['Time', 'Extension', 'Force']
78 dfn_time = dfn.Time
79 dfn_extension = dfn.Extension
80 dfn_force = dfn.Force*1e3 # convert from N to mN
81
82 # SS Array
83 ssTimeArray = np.array([HSSi, HSSf])
84 ssValArray = np.array([HFSS, HFSS])
85
86 # Max peel force displacement at max and steady state
87 dfn_max_Displacement = dfn_extension[dfn_time == HTMax]
88 # .flatten()
89 dfn_ss_Displacement = np.array([dfn_extension[dfn_time == HSSi].values[0],
90                                dfn_extension[dfn_time == HSSf].values[0]])
91
92 # Plot the data trace to compare the simulated results with the force
93 # displacement curves
94 plt.plot(dfn_extension, dfn_force, '-', color='r', linewidth=1,
95          markersize=2, label = '{} , Age: {}'.format(HID, HAGE))
96 if str(HFMax) == 'nan' and str(HSSi) == 'nan':
97     print('No max or steady state')
98     pass
99
100 if str(HFMax) != 'nan':
101     plt.plot(dfn_max_Displacement, HFMax, '.', color='k', linewidth=1,
102              markersize=20, label='Max Peel - {:.4f} (mN)'.format(HFMax),
103              path_effects=[pe.Stroke(linewidth=4, foreground='k'),
104                             pe.Normal()])
105
106 if str(HSSi) != 'nan':
107     plt.plot(dfn_ss_Displacement, ssValArray, '-', color='c', linewidth=3,
108              markersize=2, label='Steady State - {:.4f} (mN)'.format(HFSS),
109              path_effects=[pe.Stroke(linewidth=5, foreground='k'),
110                             pe.Normal()])
111

```

```

112 """ Plots """
113 ##### Plot Data #####
114 plt.plot(TD, RF, '-', color='blue', linewidth=2, markersize=2,
115          label = r'Simulated Reaction force  $\Sigma F_{\text{Retina}}$ ')
116 plt.xlabel('Displacement (mm)', fontsize=18)
117 plt.ylabel('Force (mN)', fontsize=18)
118 plt.title('Vitreous', fontsize=20)
119 plt.grid()
120 plt.legend(loc = 'best', fontsize = 'medium')
121 plt.savefig(os.path.join(dataDirectory, 'Figures/' +
122                          figureName + '_RF_vs_Dis.pdf'),
123            dpi=300, bbox_inches='tight') # Save figure
124 plt.close()
125
126 # Plot the data trace to compare the simulated results
127 plt.plot(dfn_time, dfn_force, '-', color='r', linewidth=1, markersize=2,
128          label = '{} Age: {}'.format(HID, HAGE))
129 if str(HFMax) == 'nan' and str(HSSi) == 'nan':
130     print('No max or steady state')
131     pass
132
133 if str(HFMax) != 'nan':
134     plt.plot(HTMax, HFMax, '.', color='k', linewidth=1, markersize=20,
135              label = 'Max Peel - {:.4f} (mN)'.format(HFMax),
136              path_effects=[pe.Stroke(linewidth=4, foreground='k'),
137                            pe.Normal()])
138
139 if str(HSSi) != 'nan':
140     plt.plot(ssTimeArray, ssValArray, '-', color='c', linewidth=3,
141              markersize=2, label='Steady State - {:.4f} (mN)'.format(HFSS),
142              path_effects=[pe.Stroke(linewidth=5, foreground='k'),
143                            pe.Normal()])
144
145 """ Plots """
146 ##### Plot Data #####
147 plt.plot(t, RF, '-', color='blue', linewidth=2, markersize=2,
148          label = r'Simulated Reaction force  $\Sigma F_{\text{Retina}}$ ')
149 plt.xlabel('Time (sec)', fontsize=18)
150 plt.ylabel('Force (mN)', fontsize=18)
151 plt.title('Vitreous', fontsize=20)
152 plt.grid()
153 plt.legend(loc = 'best', fontsize = 'medium')
154 plt.savefig(os.path.join(dataDirectory, 'Figures/' +
155                          figureName + '_RF_vs_t.pdf'),
156            dpi=300, bbox_inches='tight') # Save figure
157 plt.close()
158
159 ##### Plot Data #####
160 plt.plot(t, NF, '-', color='blue', linewidth=2, markersize=2,
161          label = 'Reaction force NForce')
162 plt.xlabel('Time (sec)', fontsize=18)
163 plt.ylabel('Force (N)', fontsize=18)
164 plt.title('Vitreous', fontsize=20)
165 plt.grid()
166 plt.legend(loc = 'best', fontsize = 'medium')
167 plt.savefig(os.path.join(dataDirectory, 'Figures/' +
168                          figureName + '_NF_vs_t.pdf'),
169            dpi=300, bbox_inches='tight') # Save figure

```

```

170 plt.close()
171
172 ##### Plot Data #####
173 plt.plot(t, B, '-', color='blue', linewidth=2, markersize=2,
174          label = 'Bond - Disp')
175 plt.xlabel('Time (sec)', fontsize=18)
176 plt.ylabel('Bond Disp (mm)', fontsize=18)
177 plt.title('VR Interface', fontsize=20)
178 plt.grid()
179 plt.legend(loc = 'best', fontsize = 'medium')
180 plt.savefig(os.path.join(dataDirectory, 'Figures/' +
181                          figureName + '_B_vs_t.pdf'),
182            dpi=300, bbox_inches='tight') # Save figure
183 plt.close()
184
185 ##### Plot Data #####
186 plt.plot(t, B, '-', color='blue', linewidth=2, markersize=2,
187          label = 'Bond - Disp')
188 plt.plot(t, TD, '-.', color='red', linewidth=2, markersize=2,
189          label = 'Top - Disp')
190 plt.xlabel('Time (sec)', fontsize=18)
191 plt.ylabel('Bond Disp (mm)', fontsize=18)
192 plt.title('Vitreous', fontsize=20)
193 plt.grid()
194 plt.legend(loc = 'best', fontsize = 'medium')
195 plt.savefig(os.path.join(dataDirectory, 'Figures/' +
196                          figureName + '_disp_vs_t.pdf'),
197            dpi=300, bbox_inches='tight') # Save figure
198 plt.close()
199
200 ##### Plot Data #####
201 plt.plot(t, S, '-', color='blue', linewidth=2, markersize=2,
202          label = 'Stress')
203 plt.xlabel('Time (sec)', fontsize=18)
204 plt.ylabel('Stress (Pa)', fontsize=18)
205 plt.title('Vitreous', fontsize=20)
206 plt.grid()
207 plt.legend(loc = 'best', fontsize = 'medium')
208 plt.savefig(os.path.join(dataDirectory, 'Figures/' +
209                          figureName + '_Stress_vs_t.pdf'),
210            dpi=300, bbox_inches='tight') # Save figure
211 plt.close()
212
213 if DMGInitiation == True:
214     ##### Plot Data #####
215     plt.plot(t, AVG_CSMA_XSCRT, '-', color='blue', linewidth=2,
216             markersize=2, label = r'CSMA_XSCRT$_{AVG}$')
217     plt.xlabel('Time (sec)', fontsize=18)
218     plt.ylabel('Maximum Displacement Criterion Value', fontsize=18)
219     plt.title('Vitreous', fontsize=20)
220     plt.grid()
221     plt.legend(loc = 'best', fontsize = 'medium')
222     plt.savefig(os.path.join(dataDirectory, 'Figures/' +
223                             figureName + '_AVG_CSMA_XSCRT_vs_t.pdf'),
224               dpi=300, bbox_inches='tight') # Save figure
225     plt.close()
226
227 if DMGEvolution == True:

```

```

228 ##### Plot Data #####
229 plt.plot(t, AVG_CSDMG, '-', color='blue', linewidth=2,
230          markersize=2, label = r'CSDMG$_{AVG}$')
231 plt.xlabel('Time (sec)', fontsize=18)
232 plt.ylabel('Maximum Damage Value', fontsize=18)
233 plt.title('Vitreous', fontsize=20)
234 plt.grid()
235 plt.legend(loc = 'best', fontsize = 'medium')
236 plt.savefig(os.path.join(dataDirectory, 'Figures/' +
237                          fileName + '_AVG_CSDMG_vs_t.pdf'),
238            dpi=300, bbox_inches='tight') # Save figure
239 plt.close()
240
241
242 def plot_History_Output(fileName, dataDirectory):
243     """ History Output Data """
244     df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)
245     df.columns = ["Frame", "Time", "Internal_Energy", "Kinetic_Energy"]
246
247     t_h = df.Time
248     IE = df.Internal_Energy
249     KE = df.Kinetic_Energy
250
251     (figureName, ext) = os.path.splitext(fileName) # Split the file extension
252
253     """ Plots History Outputs """
254     ##### Plot Data #####
255     plt.plot(t_h, IE, '-', color='blue', linewidth=2, markersize=2,
256            label = 'Internal Energy')
257     plt.plot(t_h, KE, '-', color='red', linewidth=2, markersize=2,
258            label = 'Kinetic Energy')
259     plt.xlabel('Time (sec)', fontsize=18)
260     plt.ylabel('Energy (J)', fontsize=18)
261     plt.title('Energy', fontsize=20)
262     plt.grid()
263     plt.legend(loc = 'best', fontsize = 'medium')
264     plt.savefig(os.path.join(dataDirectory, 'Figures/' +
265                             figureName + '_Energy.pdf'),
266               dpi=300, bbox_inches='tight') # Save figure
267     plt.close()
268
269     ##### Plot Data #####
270     plt.semilogy(t_h, KE/IE, '-', color='blue', linewidth=2,
271                 markersize=2, label = r'Ratio $\frac{KE}{IE}$')
272     plt.semilogy(t_h, 0.1*np.ones(len(t_h)), '-', color='red',
273                 linewidth=2, markersize=2, label = '10%')
274     plt.xlabel('Time (sec)', fontsize=18)
275     plt.ylabel('Ratio of KE to IE', fontsize=18)
276     plt.title('Energy ratio', fontsize=20)
277     plt.grid()
278     plt.legend(loc = 'best', fontsize = 'medium')
279     plt.savefig(os.path.join(dataDirectory, 'Figures/' +
280                             figureName + '_Ratio_KE_IE.pdf'),
281               dpi=300, bbox_inches='tight') # Save figure
282     plt.close()
283
284     print("Plots will be in the figures folder")
285

```



```

286 def plot_CohesiveCSMAXSCRT_Output(fileName, dataDirectory):
287     """ CohesiveCSMAXSCRT Output Data """
288     df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)
289
290     t = df['Time (s)']
291
292     """
293     The incoming data has both the Retina and Vitreous nodes associated
294     with it. We need to split them apart and create plots for each data
295     set separately
296     """
297
298     # Filter data by the "name" of the node that begins with 1 i.e. '1000002'
299     # and create a new dataframe
300     dfR = df.loc[:, df.columns.str.startswith('4')] # Retina
301     dfV = df.loc[:, df.columns.str.startswith('5')] # Vitreous
302
303     """ Retina """
304
305     # determine the length of the number of bonded nodes
306     # linspace from 0 to 1 by the number of nodes for the y-position
307     # Loop over the number of bonded nodes and plot the y-th
308     # vs time with the color of the bond load on a single plot
309
310     fig1, ax1 = plt.subplots()
311     nRows = np.shape(dfR)[0]
312     nCols = np.shape(dfR)[1]
313     y = np.linspace(0, 1, nCols)
314     count = 0
315     for (colName, colData) in dfR.iteritems():
316         if colName.find('Time') == -1:
317             """ Plots CSMAXSCRT Outputs """
318             ##### Plot Data #####
319             sc = ax1.scatter(t, np.ones(nRows)*y[count], c=colData,
320                             cmap=cm.cool, s=5, edgecolors='none',
321                             vmin=0, vmax=1)
322             count += 1 # update the counter
323         else:
324             continue
325
326     # plt.gray() # turns image to grayscale
327     plt.colorbar(sc)
328     ax1.set_xlabel('Time (sec)', fontsize=18)
329     ax1.set_ylabel('Cohesive CSMAXSCRT', fontsize=18)
330     ax1.set_title('Retina CSMAXSCRT (Color indicates status)', fontsize=20)
331     (figureName, ext) = os.path.splitext(fileName) # Split the file extension
332     fig1.savefig(os.path.join(dataDirectory, 'Figures/' +
333                             figureName + '_CSMAXSCRT_vs_t_Retina.pdf'),
334                 dpi=300, bbox_inches='tight') # Save figure
335     plt.close()
336
337     """ Vitreous """
338
339     # determine the length of the number of bonded nodes
340     # linspace from 0 to 1 by the number of nodes for the y-position
341     # Loop over the number of bonded nodes and plot the y-th
342     # position vs time with the color of the bond load on a single plot
343

```

```

344 fig1, ax1 = plt.subplots()
345 nRows = np.shape(dfV)[0]
346 nCols = np.shape(dfV)[1] # - 1 # subtract the time column
347 y = np.linspace(0, 1, nCols)
348 count = 0
349 for (colName, colData) in dfV.iteritems():
350     if colName.find('Time') == -1:
351         """ Plots CSMAXSCRT Outputs """
352         ##### Plot Data #####
353         sc = ax1.scatter(t, np.ones(nRows)*y[count], c=colData,
354                         cmap=cm.cool, s=5, edgecolors='none',
355                         vmin=0, vmax=1)
356         count += 1 # update the counter
357     else:
358         continue
359
360     # plt.gray() # turns image to grayscale
361     plt.colorbar(sc)
362     ax1.set_xlabel('Time (sec)', fontsize=18)
363     ax1.set_ylabel('Cohesive CSMAXSCRT', fontsize=18)
364     ax1.set_title('Vitreous CSMAXSCRT (Color indicates status)', fontsize=20)
365     (figureName, ext) = os.path.splitext(fileName) # Split the file extension
366     fig1.savefig(os.path.join(dataDirectory, 'Figures/' +
367                               figureName + '_CSMAXSCRT_vs_t-Vitreous.pdf'),
368                 dpi=300, bbox_inches='tight') # Save figure
369     plt.close()
370
371     print("Plots will be in the figures folder")
372
373 def plot_CohesiveCSDMG_Output(fileName, dataDirectory):
374     """ CohesiveCSDMG Output Data """
375     df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)
376
377     t = df['Time (s)']
378
379     """
380     The incoming data has both the Retina and Vitreous nodes associated
381     with it. We need to split them apart and create plots for each data
382     set separately
383     """
384
385     # Filter data by the "name" of the node that begins with 1 i.e. '1000002'
386     dfR = df.loc[:, df.columns.str.startswith('4')] # Retina
387     dfV = df.loc[:, df.columns.str.startswith('5')] # Vitreous
388
389     """ Retina """
390
391     # determine the length of the number of bonded nodes
392     # linspace from 0 to 1 by the number of nodes for the y-position
393     # Loop over the number of bonded nodes and plot the y-th
394     # position vs time with the color of the bond load on a single plot
395
396     fig1, ax1 = plt.subplots()
397     nRows = np.shape(dfR)[0]
398     nCols = np.shape(dfR)[1] # - 1 # subtract the time column
399     y = np.linspace(0, 1, nCols)
400     count = 0
401     for (colName, colData) in dfR.iteritems():

```

```

402     if colName.find('Time') == -1:
403         """ Plots CohesiveCSDMG Outputs """
404         ##### Plot Data #####
405         sc = ax1.scatter(t, np.ones(nRows)*y[count], c=colData,
406                         cmap=cm.cool, s=5, edgecolors='none',
407                         vmin=0, vmax=1)
408         count += 1 # update the counter
409     else:
410         continue
411
412     # plt.gray() # turns image to grayscale
413     plt.colorbar(sc)
414     ax1.set_xlabel('Time (sec)', fontsize=18)
415     ax1.set_ylabel('Cohesive CSDMG', fontsize=18)
416     ax1.set_title('Retina CSDMG (Color indicates status)', fontsize=20)
417     (figureName, ext) = os.path.splitext(fileName) # Split the file extension
418     fig1.savefig(os.path.join(dataDirectory, 'Figures/' +
419                             figureName + '_CSDMG_vs_t_Retina.pdf'),
420                 dpi=300, bbox_inches='tight') # Save figure
421     plt.close()
422
423     """ Vitreous """
424
425     # determine the length of the number of bonded nodes
426     # linspace from 0 to 1 by the number of nodes for the y-position
427     # Loop over the number of bonded nodes and plot the y-th
428     # position vs time with the color of the bond load on a single plot
429
430     fig1, ax1 = plt.subplots()
431     nRows = np.shape(dfV)[0]
432     nCols = np.shape(dfV)[1] # - 1 # subtract the time column
433     y = np.linspace(0, 1, nCols)
434     count = 0
435     for (colName, colData) in dfV.iteritems():
436         if colName.find('Time') == -1:
437             """ Plots CohesiveCSDMG Outputs """
438             ##### Plot Data #####
439             sc = ax1.scatter(t, np.ones(nRows)*y[count], c=colData,
440                             cmap=cm.cool, s=5, edgecolors='none',
441                             vmin=0, vmax=1)
442             count += 1 # update the counter
443         else:
444             continue
445
446     # plt.gray() # turns image to grayscale
447     plt.colorbar(sc)
448     ax1.set_xlabel('Time (sec)', fontsize=18)
449     ax1.set_ylabel('Cohesive CSDMG', fontsize=18)
450     ax1.set_title('Vitreous CSDMG (Color indicates status)', fontsize=20)
451     (figureName, ext) = os.path.splitext(fileName) # Split the file extension
452     fig1.savefig(os.path.join(dataDirectory, 'Figures/' +
453                             figureName + '_CSDMG_vs_t_Vitreous.pdf'),
454                 dpi=300, bbox_inches='tight') # Save figure
455     plt.close()
456
457     print("Plots will be in the figures folder")
458
459 def PlotAbqData(fileName, dataDirectory, dataCompare,

```

```

460         DMGInitiation, DMGEvolution):
461
462     # """ Change directory to correct path """
463     # filePath = os.getcwd()
464     # data_directory = os.path.join(filePath, jobName)
465     # figures_directory = os.path.join(filePath, jobName, 'Figures')
466     # if not os.path.exists(figures_directory):
467     #     os.makedirs(figures_directory)
468
469     """ Call both functions to plot Field/History data """
470     field_files = [f for f in os.listdir(dataDirectory)
471                    if os.path.isfile(os.path.join(dataDirectory, f))
472                    and f.startswith('output_Field')]
473     for fname in field_files:
474         plot_Field_Output(fname, dataDirectory, dataCompare,
475                           DMGInitiation, DMGEvolution)
476
477     history_files = [f for f in os.listdir(dataDirectory)
478                     if os.path.isfile(os.path.join(dataDirectory, f))
479                     and f.startswith('output_History')]
480     for hname in history_files:
481         plot_History_Output(hname, dataDirectory)
482
483     if DMGInitiation == True:
484         CSMAXSCRT_files = [f for f in os.listdir(dataDirectory)
485                            if os.path.isfile(os.path.join(dataDirectory, f))
486                            and f.startswith('CSMAXSCRT')]
487         for CSMAXSCRTname in CSMAXSCRT_files:
488             plot_CohesiveCSMAXSCRT_Output(CSMAXSCRTname, dataDirectory)
489
490     if DMGEvolution == True:
491         CSDMG_files = [f for f in os.listdir(dataDirectory)
492                        if os.path.isfile(os.path.join(dataDirectory, f))
493                        and f.startswith('CSDMG')]
494         for CSDMGname in CSDMG_files:
495             plot_CohesiveCSDMG_Output(CSDMGname, dataDirectory)

```

## 1.6.6 Max Cohesive Stress/Damage Criteria Script

</> Script 17: Python script used to determine the max stress/damage criteria for the cohesive surface. </>

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Sat Jan 30 01:06:55 2021
4
5  @author: Kiffer Creveling
6
7  """
8
9  import pandas as pd
10 import matplotlib.pyplot as plt
11 import numpy as np
12 import os
13 import pdb

```

```

14 plt.rcParams['figure.figsize'] = [16, 9]
15
16 def MaxCohesiveCSMAXSCRT_Output(fileName, dataDirectory, maxForceTime,
17                                   dataCompare):
18
19     """ Read in the csv file """
20     dfValsn = pd.read_csv(os.path.join(dataCompare), sep="\t", nrows=29,
21                             header=None, names=['Var', 'Attribute'])
22
23     """ File Attributes """
24     HID = dfValsn['Attribute'][0]
25     HAGE = dfValsn['Attribute'][1]
26     HG = dfValsn['Attribute'][2]
27     HLR = dfValsn['Attribute'][3]
28     HR = dfValsn['Attribute'][4]
29     HSSi = float(dfValsn['Attribute'][12])
30     HSSf = float(dfValsn['Attribute'][13])
31     HTMax = float(dfValsn['Attribute'][14])
32     HDispMax = float(dfValsn['Attribute'][15])
33     HFMax = float(dfValsn['Attribute'][16]) # (mN)
34     HFSS = float(dfValsn['Attribute'][17])
35     # (mN/m) slope from 20 seconds prior to max force value
36     HSlope20 = float(dfValsn['Attribute'][20])
37
38     dfn = pd.read_csv(os.path.join(dataCompare), sep="\t", header=30)
39     dfn.columns = ['Time', 'Extension', 'Force']
40     dfn_time = dfn.Time
41     dfn_extension = dfn.Extension # mm
42     dfn_force = dfn.Force*1e3 # N ---> mN
43
44     """ CohesiveCSMAXSCRT Output Data """
45     df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)
46
47     t = df['Time (s)']
48
49     """ The incoming data has both the Retina and Vitreous nodes associated
50     with it. We need to split them apart and create plots for each data set
51     separately """
52
53     # Filter data by the "name" of the node that begins with 1 i.e. '1000002'
54     # and create a new dataframe
55     dfR = df.loc[:, df.columns.str.startswith('4')] # Retina
56     dfV = df.loc[:, df.columns.str.startswith('5')] # Vitreous
57
58     # Turns out this is unnecessary as the time value interferes with the max
59     # # Add time to dfR & dfV
60     # dfR.insert(loc=0, column='Time', value=t)
61     # dfV.insert(loc=0, column='Time', value=t)
62
63     # Max value at specific time
64     specificTime = maxForceTime
65
66     # Value in the data frame that is closest to the specified time
67     actualTime = min(t, key=lambda x:abs(x - specificTime))
68
69     index = t[t == actualTime].index.values[0] # index
70
71     dfRSelect = dfR[t < actualTime] # Selection of the data frame

```

```

72     dfVSelect = dfV[t < actualTime] # Selection of the data frameSelect
73
74     # determine max value in the dataframe
75     retinaMaxUCRT = dfRSelect.max().max()
76     vitreousMaxUCRT = dfVSelect.max().max()
77
78     # return values
79     return retinaMaxUCRT, vitreousMaxUCRT
80
81
82 def CSMAXSCRTAbqData(fileName, dataDirectory, maxForceTime, dataCompare):
83
84     # """ Change directory to correct path """
85     # filePath = os.getcwd()
86     # data_directory = os.path.join(filePath, jobName)
87     # figures_directory = os.path.join(filePath, jobName, 'Figures')
88     # if not os.path.exists(figures_directory):
89     #     os.makedirs(figures_directory)
90
91     """ Call both functions to plot Field/History data """
92     global maxCohesiveCSMUCRT
93
94     CSMAXSCRT_files = [f for f in os.listdir(dataDirectory)
95                        if os.path.isfile(os.path.join(dataDirectory, f))
96                        and f.startswith('CSMAXSCRT')]
97     for CSMAXSCRTname in CSMAXSCRT_files:
98         maxCohesiveCSMUCRT = MaxCohesiveCSMAXSCRT_Output(CSMAXSCRTname,
99                                                         dataDirectory,
100                                                         maxForceTime,
101                                                         dataCompare)
102     return maxCohesiveCSMUCRT

```

## 1.6.7 Residual Script For Optimization

</> **Script 18:** Python script used to calculate the residual for the objective function </>  
used in the optimization routine.

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Sat Nov 7 17:27:47 2020
4
5 @author: Kiffer2
6
7 """
8 import numpy as np
9 import pandas as pd
10 from scipy import interpolate
11 import matplotlib.pyplot as plt
12 from matplotlib.pyplot import cm
13 import matplotlib.patheffects as pe
14 import os
15 import os.path
16 import sys
17 import pdb
18

```

```

19 def Least_Squares(x, y):
20     """
21     Calculate the slope and y-intercept using matrix math
22     x & y are the coordinates of points
23
24     parameters (X,Y) Data
25
26     Returns:
27         Curve fit data and parameters  $m*x + b$ , R squared value
28     """
29     Z = np.ones((len(x),2))
30     Z[:,1] = x
31     # Calculate the matrix inverse for the constants of the regression
32     A = np.dot(np.linalg.inv(np.dot(Z.T,Z)),(np.dot(Z.T,y)))
33     linFit = x*A[1] + A[0]
34
35     # Stats
36     SS_tot = np.sum((y - np.mean(y))**2)
37     SS_res = np.sum((y - linFit)**2)
38     Rsqd = 1 - SS_res/SS_tot
39
40     return linFit, A, Rsqd
41
42
43 def residualFcn(fileName, dataDirectory, maxForceTime, dataCompare, objErr,
44                 slopeFlag, maxForceFlag, ssForceFlag, timeBeforePeak):
45     """
46     Parameters
47     -----
48     fileName : Output txt file with the odb data
49     dataDirectory : Location of the output file
50
51     Returns
52     -----
53     Maximum force from the txt file
54     """
55
56     # In[Simulated data]
57     df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)
58
59     Header = [] # Header information for the dataframe
60     Header.append('Frame') # h1
61     Header.append('Time') # h2
62     Header.append('RF_y_dot') # h3
63     Header.append('RFx') # h4
64     Header.append('RFy') # h5
65     Header.append('RFz') # h6
66     Header.append('Nodal_Force') # h7
67     Header.append('Tab_Displacement') # h8
68     Header.append('Bond_Displacement') # h9
69     Header.append('Stress') # h10
70     Header.append('AVG_CSMAXSCRT') # h11
71     Header.append('AVG_CSDMG') # h12
72     df.columns = Header
73
74     tt = df.Time
75     RF = df.RF_y_dot*1000 # N to mN
76     dn = df.Tab_Displacement*1000 # m

```

```

77
78 # maybe try to output the maximum force at a specific time
79 specificTime = maxForceTime
80 actualTime = min(df['Time'], key=lambda x:abs(x - specificTime))
81 force_at_time = RF[df['Time'] == actualTime].values[0]
82
83 # In[Experimental data]
84 """ Read in the csv file """
85 dfValsn = pd.read_csv(os.path.join(dataCompare), sep="\t", nrows=29,
86                       header=None, names=['Var', 'Attribute'])
87
88 """ File Attributes """
89 HID = dfValsn['Attribute'][0]
90 HAGE = dfValsn['Attribute'][1]
91 HG = dfValsn['Attribute'][2]
92 HLR = dfValsn['Attribute'][3]
93 HR = dfValsn['Attribute'][4]
94 HSSi = float(dfValsn['Attribute'][12])
95 HSSf = float(dfValsn['Attribute'][13])
96 HTMax = float(dfValsn['Attribute'][14])
97 HDispMax = float(dfValsn['Attribute'][15])
98 HFMax = float(dfValsn['Attribute'][16]) # (mN)
99 HFSS = float(dfValsn['Attribute'][17]) # (mN)
100 # slope from 20 seconds prior to max force value
101 HSlope20 = float(dfValsn['Attribute'][20]) # (mN/m)
102
103 dfn = pd.read_csv(os.path.join(dataCompare), sep="\t", header=30)
104 dfn.columns = ['Time', 'Extension', 'Force']
105 dfn_time = dfn.Time
106 dfn_extension = dfn.Extension # mm
107 dfn_force = dfn.Force*1e3 # N ---> mN
108
109 # if fileName.find('sym') >= 0:
110 #     # divide all data trace values by 2
111 #     dfn_force = dfn_force/2
112 #     HFMax = HFMax/2
113 #     HFSS = HFSS/2
114
115 # SS Array
116 ssTimeArray = np.array([HSSi, HSSf])
117 ssValArray = np.array([HFSS, HFSS])
118
119 # In[Experimental data isolate linear region up to peak]
120
121 # slope calculation for 20 seconds prior to the max peel force
122 # (Experimental Data)
123 maxIndex = dfn_time[dfn_time == HTMax].index.values[0]
124
125 # Convert to data array length
126 timeBeforePeak = timeBeforePeak*10
127
128 # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec) to location of max
129 # force
130 x_n = dfn_extension[maxIndex - timeBeforePeak:maxIndex]
131 y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
132 # Perform least squares
133 curveFit_n, Params_n, R_Squared_n = Least_Squares(x_n, y)

```



```

134 # Shift extension data so that the linear region is extrapolated
135 # through the origin
136 shift_disp = abs(Params_n[0]/Params_n[1])
137 if Params_n[0] > 0:
138     dfn_extension_shift = dfn_extension + shift_disp
139
140     if min(dfn_extension_shift) > 0:
141         # Add zero to prevent mishaps with interpolation
142         dfn_extension_shift = [0] + dfn_extension_shift
143 else:
144     dfn_extension_shift = dfn_extension - shift_disp
145
146 # Now that the data has been shifted, recalculate the linear regression
147 # using the reduced data set
148 # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec) to location of max
    ↪ force
149 x_n = dfn_extension_shift[maxIndex - timeBeforePeak:maxIndex]
150 # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec) to location of max
    ↪ force
151 y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
152 # Perform least squares
153 curveFit_n, Params_n, R_Squared_n = Least_Squares(x_n,y)
154
155 # Slope of the curve up to the max force !!!(from the simulated data)!!!
156 # find the closest simulated displacement to the experimental
157 # max displacement
158 # adjustDisp = min(dn, key=lambda x:abs(x - dfn_extension[maxIndex]))
159 # index = RF[dn == adjustDisp].index.values[0] # index determination
160 # Index where the max reaction force is in the array
161 simMaxIndex = RF.idxmax()
162 simMaxForce = RF.max() # maximum simulated force value
163 simMaxDisp = dn[RF == simMaxForce] # displacement at the max force value
164
165 # If the max index is the second data point add one to it (Difficulty in
166 # selecting the pandas series value) to select the first two values in the
167 # pandas array it needs to be RF[0:2] instead of RF[0:1] but the index
168 # value of the max force is 1. Try to fix this issue
169 if simMaxIndex == 1:
170     simMaxIndex += 1
171
172 x = dn[0:simMaxIndex] # Array from 0 to location of max force/n
173 y = RF[0:simMaxIndex] # Array from 0 to location of max force/n
174 # Perform least squares
175 curveFit, Params, R_Squared = Least_Squares(x,y)
176
177 # Updated force at specific max disp with adjusted value (Simulated data)
178 specificTime = maxForceTime
179 actualDisp = min(dn, key=lambda x:abs(x - dfn_extension_shift[maxIndex]))
180 force_at_Displacement = RF[dn == actualDisp].values[0]
181
182 # Max peel force displacement at max and steady state
183 dfn_max_Displacement = dfn_extension_shift[dfn_time == HTMax]
184 dfn_ss_Displacement = [dfn_extension_shift[dfn_time == HSSi].values[0],
185                        dfn_extension_shift[dfn_time == HSSf].values[0]] # flatten()
186
187 """ Simulated Steady State calculation """
188 if simMaxIndex == len(RF):
189     simMaxGreaterIndex = len(RF) - 1

```

```

190 else:
191     # return the mean and median of the points after the peak force value
192     # This will always round down
193     simMaxGreaterIndex = int(simMaxIndex + (len(RF) - simMaxIndex)*(31/64))
194
195     # Steady state values from the max force index half way to the end
196     # Force values after the peak force
197     RF_SteadyState = RF[simMaxGreaterIndex:]
198     # Displacement values after the peak force
199     dn_SteadyState = dn[simMaxGreaterIndex:]
200
201     SSMean = np.mean(RF_SteadyState) # Mean
202     SSMedian = np.median(RF_SteadyState) # Median
203
204     # In[Plots]
205     """ Plots """
206     # Plot the experimental, simulated, and curve fit data
207
208     # Split the file extension
209     (figureName, ext) = os.path.splitext(fileName)
210
211     # Plot the data trace to compare the simulated results with the force
212     # displacement curves
213     plt.plot(dfn_extension_shift, dfn_force, '-', color='r', linewidth=1,
214             markersize=2, label = '{} , Age: {}'.format(HID, HAGE),
215             alpha = 0.5)
216
217     if str(HFMax) == 'nan' and str(HSSi) == 'nan':
218         print('No max or steady state')
219         pass
220
221     if str(HFMax) != 'nan':
222         plt.plot(dfn_max_Dis, HFMax, '.', color='k', linewidth=1,
223             markersize=20,
224             label = 'Max Peel - {:.4f} (mN)'.format(HFMax),
225             path_effects=[pe.Stroke(linewidth=4, foreground='k'),
226                 pe.Normal()])
227         plt.plot(x_n, curveFit_n, '-', color='tab:blue', linewidth=2,
228             label=r'Curve fit Max - {} (s)'.format(timeBeforePeak/10) +
229             'y = {:.4f}x + '.format(Params_n[1]) +
230             '{:.4f} (mN), '.format(Params_n[0]) +
231             '$r^2$ = {:.4f}'.format(R_Squared_n), alpha = 1)
232
233     if str(HSSi) != 'nan':
234         plt.plot(dfn_ss_Dis, ssValArray, '-', color='c', linewidth=3,
235             markersize=2,
236             label = 'Steady State - {:.4f} (mN)'.format(HFSS),
237             path_effects=[pe.Stroke(linewidth=5, foreground='k'),
238                 pe.Normal()])
239
240     # Plot the simulated data
241     plt.plot(dn, RF, '-', color='blue', linewidth=2, markersize=2,
242         label = r'Simulated Reaction force $\Sigma F_{Retina}$')
243     plt.plot(x, curveFit, '-', color='tab:green', linewidth=2, markersize=2,
244         label = 'y = {:.4f}x + '.format(Params[1]) +
245         '{:.4f} (mN), '.format(Params[0]) +
246         '$r^2$ = {:.4f}'.format(R_Squared))
247     plt.plot(simMaxDisp, simMaxForce, '.', color='tab:red', linewidth=1,

```

```

248         markersize = 20,
249         label = 'Simulated maximum Force {:.4f} (mN)'.format(simMaxForce))
250 plt.plot(dn_SteadyState, np.ones(len(RF_SteadyState))*SSMean, '-',
251         color='tab:gray', label = 'Simulated steady state force ' +
252         '{:.4f} (mN)'.format(np.mean(RF_SteadyState)))
253
254     # In[Error Calculation]
255     # error between slope, force, and steady-state value
256
257     maxSlopeMeasured = Params_n[1] # Experimental slope
258     maxSlopeSimulated = Params[1] # Simulated slope
259     maxForceMeasured = HFMax # Experimental max force
260     maxForceSimulated = simMaxForce # Simulated max force
261     SS_Measured = HFSS # Experimental SS force
262     SSmeanSimulated = SSMean # Simulated SS force (mean)
263     SSmedianSimulated = SSMedian # Simulated SS force (median)
264
265     # Error calculation
266     errorDict = {} # Dictionary
267     if objErr == 'Difference':
268         errorDict['slope'] = (maxSlopeMeasured - maxSlopeSimulated) if slopeFlag
269         → == True else []
270         errorDict['maxForce'] = (maxForceMeasured - maxForceSimulated) if
271         → maxForceFlag == True else []
272         errorDict['ssForce'] = (SS_Measured - SSmeanSimulated) if ssForceFlag
273         → == True else []
274     elif objErr == 'Ratio':
275         errorDict['slope'] = (1 - maxSlopeMeasured / maxSlopeSimulated) if
276         → slopeFlag == True else []
277         errorDict['maxForce'] = (1 - maxForceMeasured / maxForceSimulated) if
278         → maxForceFlag == True else []
279         errorDict['ssForce'] = (1 - SS_Measured / SSmeanSimulated) if
280         → ssForceFlag == True else []
281     elif objErr == 'Relative uncertainty':
282         errorDict['slope'] = ((maxSlopeMeasured -
283         → maxSlopeSimulated)/maxSlopeMeasured) if slopeFlag == True else []
284         errorDict['maxForce'] = ((maxForceMeasured -
285         → maxForceSimulated)/maxForceMeasured) if maxForceFlag == True else []
286         errorDict['ssForce'] = ((SS_Measured - SSmedianSimulated)/SS_Measured)
287         → if ssForceFlag == True else []
288     else:
289         print('Error in MaxForceError')
290         sys.exit()
291
292     # Error array values
293     errorList = list(errorDict.values()) # convert to list
294     errorList = [x for x in errorList if x] # get rid of empty values
295
296     # String for the error array
297     errorString = ', '.join('{:.4f}'.format(i) for i in errorList)
298
299     plt.plot([dfn_max_Displacement, simMaxDisp], [HFMax, simMaxForce], '--',
300             linewidth = 1, color = 'magenta', label = r'Difference ' +
301             'between simulated & experiment max force: ' +
302             '{:.4f}'.format(HFMax - np.max(RF)))
303
304     # Plot the different conditions if they are to be compared
305     if slopeFlag == True:

```

```

297     plt.plot([], [], 'white', label = r'{} '.format(objErr) +
298             'between slopes is: ' +
299             '{:.4f}'.format(errorDict['slope']))
300
301     if maxForceFlag == True:
302         plt.plot([], [], 'white', label = r'{} '.format(objErr) +
303             'between max force is: ' +
304             '{:.4f}'.format(errorDict['maxForce']))
305
306     if ssForceFlag == True:
307         plt.plot([], [], 'white', label = r'{} '.format(objErr) +
308             'between steady state is: ' +
309             '{:.4f}'.format(errorDict['ssForce']))
310
311     plt.plot([], [], 'white',
312             label = r'Objective error array: [' + errorString + ']')
313     plt.plot([], [], 'white', label = r'Error  $L^2$  Norm: ' +
314             '{:.4f}'.format(np.sqrt(np.dot(errorList, errorList))))
315
316     ##### Plot Data #####
317     plt.axhline(0, color='black')
318     plt.axvline(0, color='black')
319     plt.ylabel('Force (mN)',fontSize=18)
320     plt.xlabel('Distance (mm)',fontSize=18)
321     plt.title('Simulation vs. Experimental Data Trace',fontSize=20)
322     plt.grid()
323     plt.legend(loc = 'best',fontSize = 'medium')
324     plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
325                             '_SlopeCompare.pdf'), dpi=300,
326               bbox_inches='tight')
327     plt.close()
328
329     # In[Calculate interpolated Experimental and Simulated data]
330
331     # slope calculation for 20 seconds prior to the max peel force
332     # (Experimental Data)
333     maxIndex = dfn_time[dfn_time == HTMax].index.values[0]
334     # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec) to location of max
335     ↪ force
336     t_n = dfn_time[maxIndex - timeBeforePeak:maxIndex]
337     y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
338     # Perform least squares and return
339     curveFit_n, Params_n_time, R_Squared_n = Least_Squares(t_n, y)
340
341     # Shift extension data so that the linear region is extrapolated
342     # through the origin
343     shift_time = abs(Params_n_time[0]/Params_n_time[1])
344
345     # shift time data for visual purposes
346     if Params_n_time[0] > 0:
347         dfn_time_shift = dfn_time + shift_time
348
349         if min(dfn_time_shift) > 0:
350             # Add zero to prevent mishaps with interpolation
351             dfn_time_shift = [0] + dfn_time_shift
352     else:
353         dfn_time_shift = dfn_time - shift_time

```

```

354 # x array for the linear region leading up to the peak force
355 Fmax_t_shift = dfn_time_shift[maxIndex]
356 fit_t = np.linspace(0, Fmax_t_shift, 200) # Selected value
357 # fit_t = np.linspace(0, dfn_time_shift[np.argmax(dfn_force)], 200) # true max
358 Fmax_x_shift = dfn_extension_shift[maxIndex]
359 # fit_x = np.linspace(0, dfn_extension_shift[np.argmax(dfn_force)], 200) # true
    ↪ max
360 fit_x = np.linspace(0, Fmax_x_shift, 200) # Selected value
361
362 # create the linear region leading up to the peak force
363 def fit(params, x):
364     b, m = params
365     return m*x + b
366 fit_vals_y_time = fit(Params_n_time, fit_t)
367 fit_vals_y_force = fit(Params_n, fit_x)
368
369 # Trim the shifted experimental data to be greater than zero
370 t_exp = dfn_time_shift[dfn_time_shift >= 0]
371 x_exp = dfn_extension_shift[dfn_time_shift >= 0]
372 y_exp = dfn_force[dfn_time_shift >= 0]
373
374 # data frame with original data only shifted
375 dfdata = pd.DataFrame(np.array([t_exp, x_exp, y_exp]).T,
376                        columns=['t', 'x', 'y'])
377
378 # Select time beyond the max time to the end of the data
379 t_geq_max = dfn_time_shift[maxIndex:]
380 x_geq_max = dfn_extension_shift[maxIndex:]
381 y_geq_max = dfn_force[maxIndex:]
382
383 # dataframe of data points from the max value to the end
384 dfgmax = pd.DataFrame(np.array([t_geq_max, x_geq_max, y_geq_max]).T,
385                        columns=['t', 'x', 'y'])
386
387 # data frame of points from zero to the max value
388 linArray = np.array([fit_t, fit_x, fit_vals_y_force])
389 dfLin = pd.DataFrame(linArray.T, columns=['t', 'x', 'y'])
390
391 # create the new data frame of linear points up to the peak and all points
392 # beyond
393 dfNew = dfLin.append(dfgmax, ignore_index=True)
394
395 # Interpolate the experimental data
396 n_data_pts = 100
397 start_point_time = tt[RF.argmax()] # Time at the peak (simulated)
398 start_point_disp = dn[RF.argmax()] # Disp at the peak (simulated)
399 f_exp_time = interpolate.interp1d(dfNew['t'], dfNew['y'])
400 f_exp_disp = interpolate.interp1d(dfNew['x'], dfNew['y'])
401 t_new_exp = np.linspace(start_point_time, tt[tt.argmax()],
402                          n_data_pts) # (s)
403 x_new_exp = np.linspace(start_point_disp, dn[tt.argmax()],
404                          n_data_pts) # (mm)
405 y_new_exp_time = f_exp_time(t_new_exp) # Interpolate `interp1d`
406 y_new_exp_disp = f_exp_disp(x_new_exp) # Interpolate `interp1d`
407
408 # In[Interpolated Simulated Trace]
409
410 # Interpolate the simulated data

```

```

411 f_sim_time = interpolate.interp1d(tt, RF)
412 f_sim_disp = interpolate.interp1d(dn, RF)
413 t_new_sim = np.linspace(start_point_time, tt[tt.argmax()],
414                          n_data_pts) # (s)
415 x_new_sim = np.linspace(start_point_disp, dn[tt.argmax()],
416                          n_data_pts) # (mm)
417 y_new_sim_time = f_sim_time(t_new_sim) # Interpolate `interp1d`
418 y_new_sim_disp = f_sim_disp(x_new_sim) # Interpolate `interp1d`
419
420 # In[Plots]
421 ''' Time curve '''
422 fit, ax = plt.subplots()
423 ax.plot()
424 ax.plot(dfdata['t'], dfdata['y'], label='Original Shifted Data',
425         alpha = 0.5)
426 ax.plot(dfNew['t'], dfNew['y'], label='Merged Data',
427         alpha = 0.5)
428 ax.plot(t_new_exp, y_new_exp_time, '--', label='Interp Experimental Data')
429 ax.plot(tt, RF, label='Simulated Data')
430 ax.plot(t_new_sim, y_new_sim_time, ':', label='Interp Simulated Data')
431 ax.axhline(color='k')
432 ax.set_xlim([0, 300])
433 ax.set_xlabel('Time (s)', fontsize=14)
434 ax.set_ylabel('Force (N)', fontsize=14)
435 ax.legend(loc='best', fontsize=14)
436 ax.grid('on')
437 plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
438                          '_Interp_Time.pdf'), dpi=300,
439            bbox_inches='tight')
440 plt.close()
441
442 ''' Displacement curve '''
443 fit, ax = plt.subplots()
444 ax.plot()
445 ax.plot(dfdata['x'], dfdata['y'], label='Original Shifted Data',
446         alpha = 0.5)
447 ax.plot(dfNew['x'], dfNew['y'], label='Merged Data',
448         alpha = 0.5)
449 ax.plot(x_new_exp, y_new_exp_disp, '--', label='Interp Experimental Data')
450 ax.plot(dn, RF, label='Simulated Data')
451 ax.plot(x_new_sim, y_new_sim_disp, ':', label='Interp Simulated Data')
452 ax.axhline(color='k')
453 ax.set_xlim([0, max(dn)])
454 ax.set_xlabel('Displacement (mm)', fontsize=14)
455 ax.set_ylabel('Force (N)', fontsize=14)
456 ax.legend(loc='best', fontsize=14)
457 ax.grid('on')
458 plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
459                          '_Interp_Disp.pdf'), dpi=300,
460            bbox_inches='tight')
461 plt.close()
462
463 ''' Displacement curve only showing interpolated data '''
464 residual = y_new_exp_disp - y_new_sim_disp # residual calculation
465 L2Norm = np.sqrt(np.dot(residual, residual))
466
467 fit, ax = plt.subplots()
468 ax.plot()

```

```

469 ax.plot(x_new_exp, y_new_exp_disp, '-', label='Interp Experimental Data')
470 ax.plot(x_new_sim, y_new_sim_disp, '-', label='Interp Simulated Data')
471 ax.plot(x_new_sim, residual, ':', label=r'Residual = $(exp - sim)$',
472         alpha = 0.8)
473 ax.plot([], [], color='white', label=r'$L^2$ norm = {:.4f}'.format(L2Norm))
474 ax.axhline(color='k', linewidth=0.25)
475 ax.set_xlim([0, max(x_new_exp)])
476 ax.set_xlabel('Displacement (mm)', fontsize=14)
477 ax.set_ylabel('Force (N)', fontsize=14)
478 ax.legend(loc='best', fontsize=14)
479 ax.grid('on')
480 plt.savefig(os.path.join(dataDirectory, 'Figures/' + figureName +
481                             '_Interp_Dispclean.pdf'), dpi=300,
482             bbox_inches='tight')
483 plt.close()
484
485 returnList = [Params[1], simMaxForce, SSMean, SSMedian, y_new_exp_disp,
486              y_new_sim_disp]
487 return returnList
488
489 # In[Function that calls the nested function to compute the residual]
490 def findResidual(fileName, dataDirectory, maxForceTime, dataCompare, objErr,
491                  slopeFlag, maxForceFlag, ssForceFlag, timeBeforePeak):
492     """
493     Parameters
494     -----
495     fileName : Output txt file with the odb data
496     dataDirectory : Location of the output file
497
498     Returns
499     -----
500     maximumForce : Maximum force from the txt file
501     """
502
503     global residual
504     """ Call function to return max displacement """
505     ModelParamsFile = [f for f in os.listdir(dataDirectory)
506                        if os.path.isfile(os.path.join(dataDirectory, f))
507                        and f.startswith('output_Field')]
508     for mpFile in ModelParamsFile:
509         residual = residualFcn(mpFile, dataDirectory, maxForceTime,
510                               dataCompare, objErr, slopeFlag, maxForceFlag,
511                               ssForceFlag, timeBeforePeak)
512
513     return residual

```

## 1.6.8 Move Simulation Files To A Single Folder

</> **Script 19:** Python script used to move all of the output Abaqus files to a separate folder for better organization during optimization batch runs. </>

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Fri Jun 19 16:02:44 2020
4

```

```

5  @author: Kiffer Creveling
6  """
7
8  # importing os module
9  import os
10 import glob
11 import shutil
12
13 def MoveAbqFiles(fileName, folderDirectory, abqWD):
14
15     # """ Change directory to correct path """
16
17     # dataDirectory = os.path.join(abqWD, fileName)
18     # if not os.path.exists(dataDirectory):
19     #     os.makedirs(dataDirectory)
20
21     # List of files in the ABQ working directory with the same name as the
22     # 'fileName'
23     fileList = glob.glob('{}.*'.format(os.path.join(abqWD, fileName)))
24     for i in fileList:
25         if i == folderDirectory:
26             # Skip the file with the exact same name (i.e. Folder name...)
27             continue
28         source = os.path.join(abqWD,i)
29         destination = os.path.join(folderDirectory)
30         # copy (since shutil.move wouldn't overwrite)
31         dest = shutil.copy(source, destination)
32         os.remove(source) # remove the source file
33
34     return print('Files moved = :)')

```

## 1.7 Simulation Summary

### 1.7.1 Data Compilation

</> **Script 20:** Python script used to compile simulation data results to make the summary Table 1.1. </>

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Sun Apr 18 12:12:57 2021
4
5  @author: Kiffer2
6  """
7
8  import os
9  from pathlib import Path
10 import shutil
11 import pandas as pd
12 import numpy as np
13 import pdb
14
15 cwd = os.getcwd() # Get current working directory
16

```



```

17 OP = 'Results'
18
19 # In[Elastic Modulus]
20 ElastModResults = os.path.join(cwd, OP, 'ElasticModulusPlots')
21
22 elasticModFileName = '*optTIE_SlopeCompare.pdf'
23
24 # Search the folder directory for the file name that matches
25 for path in Path(cwd).rglob(elasticModFileName):
26
27     filePathList = os.path.normpath(path).split(os.path.sep)
28
29     # Look in the "Finished" folder
30     if 'Finished' in filePathList:
31
32         # Data trace
33         dataTrace = filePathList[4] # Specific data trace
34
35         print(dataTrace, path.name)
36
37         # New file name (NFN)
38         NFN = dataTrace + '.pdf'
39
40         # Copy search results to the destination folder
41         try:
42             NP = os.path.join(cwd, ElastModResults) # New path
43
44             # Create folder if it doesn't exist
45             os.makedirs(NP, exist_ok=True)
46
47             shutil.copy(path, os.path.join(NP, NFN)) # move files
48         except shutil.SameFileError:
49             pass
50
51
52
53
54 # In[Elastic Modulus Attributes]
55 ElastModAttr = os.path.join(cwd, OP, 'ElasticModulusAttr')
56
57 initialName = 'output'
58 elasticModFileAttr = '*optTIE.txt'
59
60 # Search the folder directory for the file name that matches
61 for path in Path(cwd).rglob(elasticModFileAttr):
62
63
64     # Search for the input parameters
65     if path.name.find(initialName) < 0:
66
67
68         filePathList = os.path.normpath(path).split(os.path.sep)
69
70         # Look in the "Finished" folder
71         if 'Finished' in filePathList:
72
73             # Data trace
74             dataTrace = filePathList[4] # Specific data trace

```

```

75
76     print(dataTrace, path.name)
77
78     # New file name (NFN)
79     NFN = dataTrace + '.txt'
80
81     # Copy search results to the destination folder
82     try:
83         NP = os.path.join(cwd, ElastModAttr) # New path
84
85         # Create folder if it doesn't exist
86         os.makedirs(NP, exist_ok=True)
87
88         shutil.copy(path, os.path.join(NP, NFN)) # move files
89     except shutil.SameFileError:
90         pass
91
92
93     # In[Elastic Modulus Convergence]
94     ElastModConv = os.path.join(cwd, OP, 'ElasticModulusConvergence')
95
96     ElastModConvAttr = 'FEAAttributes.txt'
97
98     # Search the folder directory for the file name that matches
99     for path in Path(cwd).rglob(ElastModConvAttr):
100
101         # Split file path to a list
102         filePathList = os.path.normpath(path).split(os.path.sep)
103
104         # Look in the "Finished" folder
105         if 'Finished' in filePathList:
106
107             if filePathList[6].find('optTIE') > 0:
108
109                 # Data trace
110                 dataTrace = filePathList[4] # Specific data trace
111
112                 print(dataTrace, path.name)
113
114                 # New file name (NFN)
115                 NFN = dataTrace + '.txt'
116
117                 # Load Data
118
119                 # Add names to file because 'SimSlope', 'SimMax', 'SimSS'
120                 # were missing
121                 names = ['FileName', 'Time', 'E1', 'E2', 'PT', 'G', 'V1', 'V2',
122                         'R', 'F', 'MS', 'RE', 'VE', 'Knn', 'Kss', 'Ktt',
123                         'DamageInitiation', 'tn', 'ts', 'tt', 'DamageEvolution',
124                         'FE', 'Optimization', 'TIE', 'errorListL2Norm',
125                         'ObjectiveFunction', 'SimSlope', 'SimMax', 'SimSS',
126                         'simTime']
127                 df = pd.read_csv(path, names=names, sep='\t', header=0)
128
129                 # Save Dat
130                 df.to_csv(path, sep='\t', index=False, na_rep='nan')
131
132                 # Copy search results to the destination folder

```

```

133         try:
134             NP = os.path.join(cwd, ElastModConv) # New path
135
136             # Create folder if it doesn't exist
137             os.makedirs(NP, exist_ok=True)
138
139             shutil.copy(path, os.path.join(NP, NFN)) # move files
140
141             df.to_csv(os.path.join(NP, NFN), sep='\t', index=False,
142                      na_rep='nan')
143         except shutil.SameFileError:
144             pass
145
146     # In[Cohesive Behavior Plots]
147     CohesiveResults = os.path.join(cwd, OP, 'CohesiveBehaviorPlots')
148
149     CohesiveFileName = '*opt_SlopeCompare.pdf'
150
151     # Search the folder directory for the file name that matches
152     for path in Path(cwd).rglob(CohesiveFileName):
153
154         filePathList = os.path.normpath(path).split(os.path.sep)
155
156         # Look in the "Finished" folder
157         if 'Finished' in filePathList:
158
159             # Data trace
160             dataTrace = filePathList[4] # Specific data trace
161
162             print(dataTrace, path.name)
163
164             # New file name (NFN)
165             NFN = dataTrace + '.pdf'
166
167             # Copy search results to the destination folder
168             try:
169                 NP = os.path.join(cwd, CohesiveResults) # New path
170
171                 # Create folder if it doesn't exist
172                 os.makedirs(NP, exist_ok=True)
173
174                 shutil.copy(path, os.path.join(NP, NFN)) # move files
175             except shutil.SameFileError:
176                 pass
177
178
179     # In[Cohesive Behavior Attributes]
180     CohesiveAttr = os.path.join(cwd, OP, 'CohesiveBehaviorAttr')
181
182     initialName = '_T3_C'
183     elasticModFileAttr = '*opt.txt'
184
185     # Search the folder directory for the file name that matches
186     for path in Path(cwd).rglob(elasticModFileAttr):
187
188         # Search for the input parameters
189         if path.name.find(initialName) < 0:
190

```

```

191     filePathList = os.path.normpath(path).split(os.path.sep)
192
193     # Look in the "Finished" folder
194     if 'Finished' in filePathList:
195
196         # Data trace
197         dataTrace = filePathList[4] # Specific data trace
198
199         print(dataTrace, path.name)
200
201         # New file name (NFN)
202         NFN = dataTrace + '.txt'
203
204         # Copy search results to the destination folder
205         try:
206             NP = os.path.join(cwd, CohesiveAttr) # New path
207
208             # Create folder if it doesn't exist
209             os.makedirs(NP, exist_ok=True)
210
211             shutil.copy(path, os.path.join(NP, NFN)) # move files
212         except shutil.SameFileError:
213             pass
214
215
216     # In[Cohesive Behavior Convergence]
217     CohConv = os.path.join(cwd, OP, 'CohesiveBehaviorConvergence')
218
219     CohConvAttr = 'FEAAttributes.txt'
220
221     # Search the folder directory for the file name that matches
222     for path in Path(cwd).rglob(CohConvAttr):
223
224         # Split file path to a list
225         filePathList = os.path.normpath(path).split(os.path.sep)
226
227         # Look in the "Finished" folder
228         if 'Finished' in filePathList:
229
230             if filePathList[5].find('optTIE') == -1:
231
232                 # Data trace
233                 dataTrace = filePathList[4] # Specific data trace
234
235                 print(dataTrace, path.name)
236
237                 # New file name (NFN)
238                 NFN = dataTrace + '.txt'
239
240                 # Load Data
241
242                 # Add names to file because 'SimSlope', 'SimMax', 'SimSS'
243                 # were missing
244                 names = ['FileName', 'Time', 'E1', 'E2', 'PT', 'G', 'V1', 'V2',
245                         'R', 'F', 'MS', 'RE', 'VE', 'Knn', 'Kss', 'Ktt',
246                         'DamageInitiation', 'tn', 'ts', 'tt', 'DamageEvolution',
247                         'FE', 'Optimization', 'TIE', 'errorListL2Norm',
248                         'ObjectiveFunction', 'SimSlope', 'SimMax', 'SimSS',

```

```

249         'simTime']
250     df = pd.read_csv(path, names=names, sep='\t', header=0)
251
252     # Save Dat
253     df.to_csv(path, sep='\t', index=False, na_rep='nan')
254
255
256
257     # Copy search results to the destination folder
258     try:
259         NP = os.path.join(cwd, CohConv) # New path
260
261         # Create folder if it doesn't exist
262         os.makedirs(NP, exist_ok=True)
263
264         shutil.copy(path, os.path.join(NP, NFN)) # move files
265
266         df.to_csv(os.path.join(CohConv, NFN), sep='\t', index=False,
267                   na_rep='nan')
268     except shutil.SameFileError:
269         pass
270
271     # In[Fracture Energy Integral]
272
273     FEInt = os.path.join(cwd, OP, 'FractureEnergyIntegrals')
274
275     elasticModFileAttr = 'GcSelection.pdf'
276
277     # Search the folder directory for the file name that matches
278     for path in Path(cwd).rglob(elasticModFileAttr):
279
280         filePathList = os.path.normpath(path).split(os.path.sep)
281
282         # Look in the "Finished" folder
283         if 'Finished' in filePathList:
284
285             # Data trace
286             dataTrace = filePathList[4] # Specific data trace
287
288             print(dataTrace, path.name)
289
290             # New file name (NFN)
291             NFN = dataTrace + '.pdf'
292
293             # Copy search results to the destination folder
294             try:
295                 NP = os.path.join(cwd, FEInt) # New path
296
297                 # Create folder if it doesn't exist
298                 os.makedirs(NP, exist_ok=True)
299
300                 shutil.copy(path, os.path.join(NP, NFN)) # move files
301             except shutil.SameFileError:
302                 pass
303
304
305     # In[YouTube video links]
306

```

```

307 YouTube = os.path.join(cwd, OP, 'YouTube')
308
309 YouTubeFile = 'YouTubeLink.txt'
310
311 # Search the folder directory for the file name that matches
312 for path in Path(cwd).rglob(YouTubeFile):
313
314     filePathList = os.path.normpath(path).split(os.path.sep)
315
316     # Look in the "Finished" folder
317     if 'Finished' in filePathList:
318
319         # Data trace
320         dataTrace = filePathList[4] # Specific data trace
321
322         print(dataTrace, path.name)
323
324         # New file name (NFN)
325         NFN = dataTrace + '.txt'
326
327         # Copy search results to the destination folder
328         try:
329             NP = os.path.join(cwd, YouTube) # New path
330
331             # Create folder if it doesn't exist
332             os.makedirs(NP, exist_ok=True)
333
334             shutil.copy(path, os.path.join(NP, NFN)) # move files
335         except shutil.SameFileError:
336             pass

```

## 1.7.2 Simulation Table

Simulation optimization summary results are in Table 1.1.

## 1.8 Data Analytics

### 1.8.1 Statistics

</> **Script 21:** Post simulation python script that analyzes simulation to show parameter trends and statistical analyses. </>

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Apr 18 17:25:44 2021
4
5 @author: Kiffer
6 """
7
8 import pandas as pd
9 import numpy as np
10 import seaborn as sns
11 from statannot import add_stat_annotation

```

Table 1.1: Computational simulation optimization results.

Trace #	Age (Yrs.)	L/R	Region Eq/Po	$Exp_{\max}$ (mN)	$Exp_{SS}$ (mN)	$Sim_{\max}$ (mN)	$Sim_{SS}$ (mN)	$L^2$ Norm (mN)	Video link
1	30	R	Po.	1.674	1.317	1.750	1.409	0.129	<a href="#">1</a>
3	30	L	Po.	4.680	0.689	2.932	0.546	1.753	<a href="#">3</a>
4	30	L	Eq.	8.620	3.179	7.047	4.111	1.819	<a href="#">4</a>
5	34	R	Eq.	8.950	2.889	7.961	4.114	2.273	<a href="#">5</a>
6	34	R	Po.	3.800	0.559	3.034	1.778	1.407	<a href="#">6</a>
7	34	L	Po.	2.703	1.000	1.652	0.983	1.051	<a href="#">7</a>
8	60	L	Eq.	14.605	2.234	13.956	3.512	1.635	<a href="#">8</a>
9	60	L	Po.	4.560	2.846	4.197	2.898	0.367	<a href="#">9</a>
10	44	L	Po.	4.144	1.931	3.432	2.203	0.757	<a href="#">10</a>
11	44	L	Eq.	8.641	1.869	8.307	2.141	0.481	<a href="#">11</a>
12	44	R	Eq.	3.222	0.783	3.009	1.660	0.925	<a href="#">12</a>
13	44	R	Po.	5.300	1.399	4.607	3.239	1.993	<a href="#">13</a>
15	57	L	Eq.	16.230	2.121	15.575	2.739	0.932	<a href="#">15</a>
18	42	R	Po.	9.740	1.551	9.584	0.936	0.626	<a href="#">18</a>
20	42	L	Eq.	9.656	1.590	9.081	7.077	5.611	<a href="#">20</a>
23	47	R	Eq.	4.547	1.233	3.743	1.470	0.836	<a href="#">23</a>
24	47	L	Po.	3.830	2.762	3.366	2.884	0.484	<a href="#">24</a>
25	47	L	Eq.	11.136	1.424	10.510	10.510	1.097	<a href="#">25</a>
26	70	R	Eq.	4.740	1.432	4.831	1.511	0.107	<a href="#">26</a>
27	70	L	Po.	3.317	2.634	3.292	2.717	0.087	<a href="#">27</a>
28	72	R	Eq.	1.985	0.466	1.341	0.777	0.704	<a href="#">28</a>
30	72	L	Po.	2.283	0.863	1.774	0.614	0.606	<a href="#">30</a>
31	56	R	Po.	5.549	1.367	5.282	2.684	1.359	<a href="#">31</a>
32	56	L	Eq.	3.031	1.725	2.883	2.488	0.758	<a href="#">32</a>
34	70	R	Eq.	3.359	1.190	3.067	1.573	0.470	<a href="#">34</a>
35	70	R	Po.	5.969	1.912	5.537	2.257	0.527	<a href="#">35</a>
36	70	L	Eq.	4.838	1.754	4.030	2.618	1.175	<a href="#">36</a>
37	78	R	Eq.	5.010	0.416	3.852	1.128	1.349	<a href="#">37</a>
38	78	R	Po.	1.550	0.246	1.355	0.451	0.292	<a href="#">38</a>
39	78	L	Po.	2.467	0.687	2.883	0.676	0.429	<a href="#">39</a>
40	78	L	Eq.	9.080	1.097	9.896	1.717	1.258	<a href="#">40</a>
41	57	L	Eq.	4.275	0.569	3.356	0.253	1.066	<a href="#">41</a>
42	57	L	Po.	2.357	1.353	2.187	2.187	0.852	<a href="#">42</a>
43	57	R	Po.	6.058	0.638	5.727	5.727	0.706	<a href="#">43</a>
46	56	R	Po.	4.507	0.349	3.216	3.216	1.543	<a href="#">46</a>
47	63	R	Eq.	2.971	0.353	2.547	0.414	0.425	<a href="#">47</a>
48	63	R	Po.	3.026	1.081	2.491	1.017	0.535	<a href="#">48</a>
49	63	L	Po.	2.130	1.233	2.216	1.915	0.680	<a href="#">49</a>
50	69	L	Eq.	3.840	0.972	2.952	1.284	0.941	<a href="#">50</a>
51	69	L	Po.	2.810	1.122	2.232	1.775	0.881	<a href="#">51</a>
52	69	R	Eq.	4.637	0.759	3.568	0.570	1.074	<a href="#">52</a>

```

12 import matplotlib.pyplot as plt
13 plt.rcParams['figure.figsize'] = [16, 10]
14 from scipy import stats
15 import statsmodels.api as sm
16 from statsmodels.formula.api import ols
17 import pdb
18 import os
19 import glob
20 import re
21
22 cwd = os.getcwd()
23
24 # In[Peel test data]]
25
26 fp = os.path.join(cwd, 'Results', 'ExpPeelDataCompare')
27
28 # Grab all files that have "trace_"
29 fileList = glob.glob(os.path.join(fp, "trace_*"))
30
31 finalAttr = []
32
33 for i,j in enumerate(fileList):
34
35     # Load each data set
36     df = pd.read_csv(j, sep = '\t', nrows=29, names=['Variable', 'Value'])
37     df = df.set_index('Variable').T # Transpose
38
39     filePathList = os.path.normpath(j).split(os.path.sep)
40     DataTrace = filePathList[-1]
41
42     # Add the trace name to the dataframe
43     df['DataTrace'] = DataTrace
44
45     # Specific trace number from the string
46     result = re.search('Trace_(.*)_Instron_Data.txt', DataTrace)
47
48     # convert the string to an integer for sorting
49     df['DataTrace#'] = int(result.group(1))
50
51     finalAttr.append(df.tail(1).values.tolist())
52
53 # Compress list
54 finalAttr = [item for sublist in finalAttr for item in sublist]
55
56 # Create the new dataframe with the names from the previous data
57 df = pd.DataFrame(finalAttr, columns=df.columns.values)
58
59 # Experimental Data (ed)
60 ed = df.sort_values('DataTrace#').reset_index(drop=True)
61
62 #--- Save Data ---#
63 outputFileDirectory = os.path.join('Results', 'OutputFiles') # Folder
64
65 # Make folder if it doesn't exist
66 os.makedirs(outputFileDirectory, exist_ok=True)
67
68 # new File
69 outputFile = os.path.join(outputFileDirectory, 'PeelDataSummary.txt')

```



```

70
71 print('New file:', outputFile)
72
73 # Save results
74 ed.to_csv(outputFile, sep='\t', index=False, na_rep='nan')
75
76
77 # In[simulation results]
78 # Headers
79 names = ['FileName', 'Time', 'E1', 'E2', 'PT', 'G', 'V1', 'V2', 'R',
80          'F', 'MS', 'RE', 'VE', 'Knn', 'Kss', 'Ktt',
81          'DamageInitiation', 'tn', 'ts', 'tt', 'DamageEvolution',
82          'FE', 'Optimization', 'TIE', 'errorListL2Norm',
83          'ObjectiveFunction', 'SimSlope', 'SimMax', 'SimSS', 'simTime']
84
85 SF = os.path.join('Results', 'StatisticsFigures')
86
87 # Create folder if it doesn't exist
88 os.makedirs(SF, exist_ok=True)
89
90 # In[Elastic Modulus Convergente]
91
92 fp = os.path.join(cwd, 'Results', 'ElasticModulusConvergence')
93
94 fileList = glob.glob(os.path.join(fp, "sample*.txt"))
95
96 finalAttr = []
97 ElasticSummary = {} # Dictionary to look at each optimization routine
98
99 for i in fileList:
100
101     # Load each data set
102     df = pd.read_csv(i, sep = '\t')
103     finalAttr.append(df.tail(1).values.tolist())
104
105     # Append each data set to a single dictionary
106     ElasticSummary[i] = df
107
108 # Compress list
109 finalAttr = [item for sublist in finalAttr for item in sublist]
110
111 # Create the new dataframe
112 df = pd.DataFrame(finalAttr, columns=names)
113
114 #--- Save Data ---#
115 outputFileDirectory = os.path.join('Results', 'OutputFiles') # Folder
116
117 # Make folder if it doesn't exist
118 os.makedirs(outputFileDirectory, exist_ok=True)
119
120 # new File
121 outputFile = os.path.join(outputFileDirectory, 'ElasticConvergenceSummary.txt')
122
123 print('New file:', outputFile)
124
125 # Save results
126 df.to_csv(outputFile, sep='\t', index=False, na_rep='nan')
127

```

```

128
129 Ev = df['VE'] # Elastic modulus
130
131 # In[Plots]
132
133 standardError = 68 # Used for confidence intervals
134
135 sns.set_theme(context='paper', style='darkgrid', palette="Paired",
136               font_scale=2)
137 sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
138                             "axes.labelsize":12})
139 custom_style = {'axes.facecolor': 'white',
140                 'axes.edgecolor': 'black',
141                 'axes.grid': False,
142                 'axes.axisbelow': True,
143                 'axes.labelcolor': 'black',
144                 'figure.facecolor': 'white',
145                 'grid.color': '.8',
146                 'grid.linestyle': '-',
147                 'text.color': 'black',
148                 'xtick.color': 'black',
149                 'ytick.color': 'black',
150                 'xtick.direction': 'out',
151                 'ytick.direction': 'out',
152                 'lines.solid_capstyle': 'round',
153                 'patch.edgecolor': 'w',
154                 'patch.force_edgecolor': True,
155                 'image.cmap': 'rocket',
156                 'font.family': ['sans-serif'],
157                 'font.sans-serif': ['Arial', 'DejaVu Sans', 'Liberation Sans',
158                                     'Bitstream Vera Sans', 'sans-serif'],
159                 'xtick.bottom': True,
160                 'xtick.top': False,
161                 'ytick.left': True,
162                 'ytick.right': False,
163                 'axes.spines.left': True,
164                 'axes.spines.bottom': True,
165                 'axes.spines.right': False,
166                 'axes.spines.top': False,
167                 'xtick.labelsize' : 16,
168                 'ytick.labelsize' : 16,
169                 'legend.title_fontsize' : 20}
170
171 # White background with ticks and black border lines, Turns grid off
172 ax = sns.set_style(rc=custom_style)
173
174 # In[Functions]
175
176 # fcn for plotting
177 def yfit(x):
178     return slope*x + intercept
179
180 # In[Pivot table info]
181
182 pvtOut = {'count', np.median, np.mean, np.std} # pivot table outputs
183
184 # In[Plot simplifications]
185

```

```

186 R = 'Region'
187 Eq = 'Equator'
188 Po = 'Posterior'
189 AG = 'AgeGroup'
190 A60 = 'Age60'
191 Aleq60 = r'Age $\leq$ 60'
192 Ag60 = 'Age $>$ 60'
193 A = 'Age'
194
195 # Units
196 MPF = 'Maximum Peel Force (mN)'
197 SSPF = 'Steady-State Peel Force (mN)'
198 KDEUnit = r'Kernel Density Estimation'
199 ElasticUnit = r'Elastic Modulus (Pa)'
200 CohBehUnit = r'Cohesive Behavior (Pa)'
201 CohDMGUnit = r'Cohesive Damage Initiation (Pa)'
202 FEUnit = r'Fracture Energy (J)'
203
204 A_yrs = 'Age (yr.)'
205 A_G = 'Age Group (yr.)'
206
207
208 # In[Cohesive Behavior Convergence]
209
210 fp = os.path.join(cwd, 'Results', 'CohesiveBehaviorConvergence')
211
212 fileList = glob.glob(os.path.join(fp, "sample*.txt"))
213
214 finalAttr = []
215
216 for i,j in enumerate(fileList):
217
218     # Load each data set
219     df = pd.read_csv(j, sep = '\t')
220
221
222     filePathList = os.path.normpath(j).split(os.path.sep)
223     fileName = filePathList[-1]
224
225     # Add the filename
226     df['SimulationFileName'] = fileName
227
228     # Specific trace number from the string
229     result = re.search('Sample#(.*)\.txt', fileName)
230
231     # convert the string to an integer for sorting
232     df['DataTrace#'] = int(result.group(1))
233
234     # Grab the final row (converged results)
235     finalAttr.append(df.tail(1).values.tolist())
236
237 # Compress list
238 finalAttr = [item for sublist in finalAttr for item in sublist]
239
240 # Create the new dataframe
241 df = pd.DataFrame(finalAttr, columns=df.columns.values)
242
243 # Simulation Data (sd)

```

```

244 sd = df.sort_values('DataTrace#').reset_index(drop=True)
245
246 ### Save Data ###
247 outputFileDirectory = os.path.join('Results', 'OutputFiles') # Folder
248
249 # Make folder if it doesn't exist
250 os.makedirs(outputFileDirectory, exist_ok=True)
251
252 # new File
253 outputFile = os.path.join(outputFileDirectory, 'CohesiveBehaviorSummary.txt')
254
255 print('New file:', outputFile)
256
257 # Save results
258 sd.to_csv(outputFile, sep='\t', index=False, na_rep='nan')
259
260 Ev = sd['VE'] # Elastic modulus
261
262 # Cohesive Behaviour
263 sd['Knn'] = 2**sd['Knn'] # Elastic modulus
264 sd['Kss'] = 2**sd['Kss'] # Elastic modulus
265 sd['Ktt'] = 2**sd['Ktt'] # Elastic modulus
266
267 # Damage Initiation
268 sd['tn'] = 2**sd['tn'] # Normal 1
269 sd['ts'] = 2**sd['ts'] # Shear 1
270 sd['tt'] = 2**sd['tt'] # Shear 2
271
272 # Fracture energy
273 sd['FE'] = 2**sd['FE'] # Elastic modulus
274
275 # Outputs
276 errList = sd['errorListL2Norm']
277
278 SS = sd['SimSlope']
279 SM = sd['SimMax']
280 Ss = sd['SimSS']
281
282 # In[YouTube Links]
283
284 fp = os.path.join(cwd, 'Results', 'YouTube')
285
286 fileList = glob.glob(os.path.join(fp, "sample*.txt"))
287
288 finalAttr = []
289
290 for i,j in enumerate(fileList):
291
292     # Load each data set
293     df = pd.read_csv(j, names=['Link'])
294
295     filePathList = os.path.normpath(j).split(os.path.sep)
296     fileName = filePathList[-1]
297
298     # Add the filename
299     df['SimulationFileName'] = fileName
300
301     # Specific trace number from the string

```

```

302     result = re.search('Sample#(.*)\.txt', fileName)
303
304     tNum = int(result.group(1)) # Trace number
305
306     # convert the string to an integer for sorting
307     df['DataTrace#'] = tNum
308
309     # Hyperlink for LaTeX
310     df['HyperLink'] = ('\href{' + '{}'.format(df['Link'][0]) +
311                       '{}{' + '{}'.format(tNum) + '}')
312
313     # Grab the final row (converged results)
314     finalAttr.append(df.tail(1).values.tolist())
315
316 # Compress list
317 finalAttr = [item for sublist in finalAttr for item in sublist]
318
319 # Create the new dataframe
320 df = pd.DataFrame(finalAttr, columns=df.columns.values)
321
322 # YouTube (yt)
323 yt = df.sort_values('DataTrace#').reset_index(drop=True)
324
325 #--- Save Data ---#
326 outputFileDirectory = os.path.join('Results', 'OutputFiles') # Folder
327
328 # Make folder if it doesn't exist
329 os.makedirs(outputFileDirectory, exist_ok=True)
330
331 # new File
332 outputFile = os.path.join(outputFileDirectory, 'YouTube.txt')
333
334 print('New file:', outputFile)
335
336 # In[Cohesive Behavior Convergence]
337
338 fp = os.path.join(cwd, 'Results', 'CohesiveBehaviorConvergence')
339
340 fileList = glob.glob(os.path.join(fp, "sample*.txt"))
341
342 finalAttr = []
343
344 CohesiveSummary = {}
345
346 for i,j in enumerate(fileList):
347
348     # Load each data set
349     df = pd.read_csv(j, sep = '\t')
350
351
352     filePathList = os.path.normpath(j).split(os.path.sep)
353     fileName = filePathList[-1]
354
355     # Add the filename
356     df['SimulationFileName'] = fileName
357
358     # Specific trace number from the string
359     result = re.search('Sample#(.*)\.txt', fileName)

```

```

360
361     # convert the string to an integer for sorting
362     df['DataTrace#'] = int(result.group(1))
363
364     # Grab the final row (converged results)
365     finalAttr.append(df.tail(1).values.tolist())
366
367     # Append each data set to a single dictionary
368     CohesiveSummary[i] = df
369
370 # Compress list
371 finalAttr = [item for sublist in finalAttr for item in sublist]
372
373 # Create the new dataframe
374 df = pd.DataFrame(finalAttr, columns=df.columns.values)
375
376 # Simulation Data (sd)
377 sd = df.sort_values('DataTrace#').reset_index(drop=True)
378
379 #--- Save Data ---#
380 outputFileDirectory = os.path.join('Results', 'OutputFiles') # Folder
381
382 # Make folder if it doesn't exist
383 os.makedirs(outputFileDirectory, exist_ok=True)
384
385 # new File
386 outputFile = os.path.join(outputFileDirectory, 'CohesiveBehaviorSummary.txt')
387
388 print('New file:', outputFile)
389
390 # In[Merge data sets (Experimental & Simulation)]
391
392 # Merge experimental and simulation data
393 md1 = pd.merge(sd, ed, on='DataTrace#')
394 md = pd.merge(md1, yt, on='DataTrace#')
395
396 # Simplifications
397 R = 'Region'
398 Eq = 'Equator'
399 Po = 'Posterior'
400 A60 = 'Age60'
401 Aleq60 = r'Age $\leq$ 60'
402 Ag60 = r'Age $>$ 60'
403 A = 'Age'
404
405 # Redo some columns for plotting
406 md[A] = md['Human Age:']
407 md[R] = md['Human Region:']
408
409 # Break age groups into bins
410 bins = [0, 60, 90]
411 labelsAge60 = [Aleq60, Ag60]
412
413 # Properly update parameters
414 # Cohesive Behaviour
415 md['Knn'] = 2*md['Knn'] # Elastic modulus
416 md['Kss'] = 2*md['Kss'] # Elastic modulus
417 md['Ktt'] = 2*md['Ktt'] # Elastic modulus

```

```

418
419 # Damage Initiation
420 md['tn'] = 2*md['tn'] # Normal 1
421 md['ts'] = 2*md['ts'] # Shear 1
422 md['tt'] = 2*md['tt'] # Shear 2
423
424 # Fracture energy
425 md['FE'] = 2*md['FE'] # Elastic modulus
426
427 # Create binned AgeGroups
428 md[A60] = pd.cut(md[A].astype(int), bins, labels=labelsAge60, right=True)
429
430 # Convert Strings to floats/integers
431 md['EV'] = pd.to_numeric(md['VE'], downcast="float")
432
433 md['Trace'] = pd.to_numeric(md['DataTrace#'], downcast="integer")
434 md['$Exp_\max$'] = pd.to_numeric(md['FMax (mN):'], downcast="float")
435 md['$Exp_{SS}$'] = pd.to_numeric(md['FSS (mN):'], downcast="float")
436 md['$Sim_\max$'] = pd.to_numeric(md['SimMax'], downcast="float")
437 md['$Sim_{SS}$'] = pd.to_numeric(md['SimSS'], downcast="float")
438 md['$L^2$ Norm'] = pd.to_numeric(md['errorListL2Norm'], downcast="float")
439 md[A] = pd.to_numeric(md[A], downcast="integer")
440
441 # Simplify for later
442 md['L/R'] = np.where(md['Human Left/Right:'] == 'Left', 'L', 'R')
443 md['Region'] = np.where(md['Region'] == 'Equator', 'Eq.', 'Po.')
444
445 #--- Save Data ---#
446 outputFileDirectory = os.path.join('Results', 'OutputFiles') # Folder
447
448 # Make folder if it doesn't exist
449 os.makedirs(outputFileDirectory, exist_ok=True)
450
451 # new File
452 outputFile = os.path.join(outputFileDirectory, 'ExpSimSummary.txt')
453
454 print('New file:', outputFile)
455
456 # Save results
457 md.to_csv(outputFile, sep='\t', index=False, na_rep='nan')
458
459 # Create specific LaTeX table
460
461 # Add the index groups and convert NaN's to "-"'s
462 tabColumns = ['Trace',
463              A,
464              'L/R',
465              'Region',
466              '$Exp_\max$',
467              '$Exp_{SS}$',
468              '$Sim_\max$',
469              '$Sim_{SS}$',
470              '$L^2$ Norm',
471              'HyperLink']
472
473 print(md.to_latex(index=False, columns=tabColumns, na_rep='-', escape=False,
474                  float_format="{:0.3f}".format))
475

```

```

476 # In[Add full name to region after the table was created]
477
478 # md[R] = np.where(md[R] == 'Eq.', 'Equator', 'Posterior') # Difficult to use
479 md.loc[md[R] == 'Eq.', R] = Eq
480 md.loc[md[R] == 'Po.', R] = Po
481
482 # In[Smart Plot]
483
484 def boxPlotBlackBorder(ax):
485     # iterate over boxes in the plot to make each line black
486     for i, box in enumerate(ax.artists):
487         box.set_edgecolor('black')
488         # box.set_facecolor('white')
489
490     # iterate over whiskers and median lines
491     for j in range(6*i, 6*(i+1)):
492         ax.lines[j].set_color('black')
493
494 def smartPlot(data=None, x=None, y=None, hue=None, hue_order=None,
495              addBoxPair=None, ci=None, errcolor=None, capsize=None,
496              plot=None, test=None, sigLoc=None, text_format=None,
497              line_offset=None, line_offset_to_box=None, line_height=None,
498              fontsize=None, legLoc=None, verbose=None, yAxi=None,
499              xlabel=None, ylabel=None, legendTitle=None, figName=None,
500              folderName=None, dataPoints=None, stats=None):
501
502     # barplot
503     scale = 1.6
504     base = 10
505     f, ax = plt.subplots(figsize=(base*scale, base))
506
507     if plot == 'barplot':
508         ax = sns.barplot(data=data, x=x, y=y, hue=hue, hue_order=hue_order,
509                          ci=ci, errcolor=errcolor, capsize=capsize)
510
511     elif plot == 'boxplot':
512         ax = sns.boxplot(data=data, x=x, y=y, hue=hue, hue_order=hue_order)
513
514     # Statistical test for differences
515     x_grps = list(data[x].unique()) # List of groups
516     if hue != None:
517         # Create combinations to compare
518         box_pairs_1 = [((x_grps_i, hue_order[0]),
519                        (x_grps_i, hue_order[1]))
520                        for x_grps_i in x_grps]
521         box_pairs = box_pairs_1
522
523     if addBoxPair != None:
524         # Additional box pairs
525         box_pairs = box_pairs_1 + addBoxPair
526
527     elif hue_order != None:
528         box_pairs = [(hue_order[0], hue_order[1])]
529
530     if yAxi != None:
531         ax.set_yscale("log")
532
533     if stats != None:

```



```

534     #Stats results and significant differences (SR)
535     SR = add_stat_annotation(ax, plot=plot, data=data, x=x, y=y, hue=hue,
536                             hue_order=hue_order, box_pairs=box_pairs,
537                             test=test, loc=sigLoc,
538                             text_format=text_format, verbose=verbose,
539                             comparisons_correction=None,
540                             line_offset=line_offset,
541                             line_offset_to_box=line_offset_to_box,
542                             line_height= line_height,
543                             fontsize=fontsize) # 'bonferroni'
544
545     if plot == 'boxplot':
546         boxPlotBlackBorder(ax) # Make borders black
547
548     if dataPoints == True:
549         # Add data points to the box plot
550         sns.stripplot(data=data, x=x, y=y, hue=hue, hue_order=hue_order,
551                      color='.5', size=5, linewidth=1, dodge=True)
552
553         # gather plot attributes for legends
554         handles, labels = ax.get_legend_handles_labels()
555
556         if hue != None:
557             l = plt.legend(handles[0:2], labels[0:2], title=legendTitle,
558                           fontsize=18)
559
560     else:
561         if hue != None:
562             ax.legend(loc=legLoc, fontsize=18).set_title(legendTitle)
563
564     if hue != None and hue_order != None:
565         # for legend title
566         plt.setp(ax.get_legend().get_title(), fontsize=18)
567
568     ax.set_xlabel(xlabel, fontsize=18)
569     ax.set_ylabel(ylabel, fontsize=18)
570
571     # Adjust fonts, because it doesn't seem to work
572     for tick in ax.xaxis.get_major_ticks():
573         tick.label.set_fontsize(16)
574
575     for tick in ax.yaxis.get_major_ticks():
576         tick.label.set_fontsize(16)
577
578     ax = sns.despine() # takes the lines off on the right and top of the graph
579
580     if folderName != None:
581         # If a new folder name is given, put the files there
582
583         # New file path
584         NP = os.path.join(SF, folderName)
585
586         # Create folder if it doesn't exist
587         os.makedirs(NP, exist_ok=True)
588
589     else:
590         # Put the file in the same folder
591         NP = SF

```

```

592
593     f.savefig(os.path.join(NP, '{}.pdf'.format(figName)),
594               bbox_inches='tight')
595     plt.close()
596
597 def twoWayAnova(data=None, var=None, A=None, B=None, fileName=None,
598                filePath=None):
599     """
600     Two-Way Anova
601     Parameters
602     -----
603     data : dataframe
604     var : continuous variable
605     A : Effect #1
606     B : Effect #2
607     fileName : Filename
608     filePath : filepath
609     """
610     model = ols(f'{var} ~ {A} + {B} + {A}:{B}', data=data).fit()
611
612     res = sm.stats.anova_lm(model, typ=2)
613
614     print(80*'- ', 2*'\n', 'Two-way ANOVA\n', res, 2*'\n')
615
616     NP = os.path.join(SF, filePath)
617
618     # Create folder if it doesn't exist
619     os.makedirs(NP, exist_ok=True)
620
621     f = open(os.path.join(NP, f'{fileName}.txt'), "w")
622     f.write(var + '\twith effects: ' + A + ' and ' + B)
623     f.write('\n')
624     f.write(res.to_string())
625     f.close()
626
627 # In[Elastic modulus group plots by region and age +/- 60]
628 EV = 'EV'
629 Ev = [EV]
630 Folder = 'ElasticModulus'
631
632 # Additional group
633 addBoxPair1 = [((Aleq60, Eq), (Age_Group_i, Eq))
634                for Age_Group_i in list(md[A60].unique())]
635
636 addBoxPair2 = [((Aleq60, Eq), (Ag60, Po))]
637
638 addBoxPair = addBoxPair1 #+ addBoxPair2
639
640 for i in Ev:
641     pivotEv = pd.pivot_table(md, values=i, index=[A60, R],
642                               aggfunc=pvtOut)
643
644     print('pivotEv')
645     print(pivotEv)
646     # Add the index groups and convert NaN's to "-"'s
647     print(pivotEv.to_latex(index=True, na_rep='-', escape=False,
648                             float_format="{:0.3f}".format))
649

```

```

650 # Barplot
651 smartPlot(data=md, x=A60, y=i, hue=R, hue_order=[Eq, Po], ci=68,
652           errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
653           sigLoc='outside', text_format='star', line_offset=0.015,
654           line_offset_to_box=0.0, line_height=0.015, fontsize=16,
655           legLoc='best', verbose=2, yAxi=None,
656           xlabel=A_G, ylabel=ElasticUnit, legendTitle=R,
657           figName='RegionAge_BarPlot', folderName=Folder,
658           addBoxPair=addBoxPair, stats=None)
659
660 # Boxplot
661 smartPlot(data=md, x=A60, y=i, hue=R, hue_order=[Eq, Po], plot='boxplot',
662           test='t-test_ind', sigLoc='outside', text_format='star',
663           line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
664           fontsize=16, legLoc='best', verbose=2, yAxi=None,
665           xlabel=A_G, ylabel=ElasticUnit, addBoxPair=addBoxPair,
666           legendTitle=R, figName='RegionAge_BoxPlot', folderName=Folder,
667           stats=None)
668
669 # Boxplot with data
670 smartPlot(data=md, x=A60, y=i, hue=R, hue_order=[Eq, Po], plot='boxplot',
671           test='t-test_ind', sigLoc='outside', text_format='star',
672           line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
673           fontsize=16, legLoc='best', verbose=2, yAxi=None,
674           xlabel=A_G, ylabel=ElasticUnit, addBoxPair=addBoxPair,
675           legendTitle=R,
676           figName='RegionAge_BoxPlotWithData', folderName=Folder,
677           dataPoints=True, stats=None)
678
679 twoWayAnova(data=md, var='VE', A='Age', B='Region',
680             fileName='Age_Region_2wayAnova',
681             filePath=Folder)
682
683 twoWayAnova(data=md, var='VE', A=A60, B=R,
684             fileName='AgeGroup_Region_2wayAnova',
685             filePath=Folder)
686
687 # In[Elastic modulus group plots by Age Group +/- 60]
688 EV = 'EV'
689 Ev = [EV]
690 Folder = 'ElasticModulus'
691
692 # Additional group
693 addBoxPair1 = [(A60, Ag60)]
694
695 addBoxPair = addBoxPair1
696
697 # Create a matching column for repeated measures
698 md['MatchingID'] = md['Human ID:'].map(str) + md['Human Region:'].map(str)
699
700 # matched_pairs student's t-test
701
702 # dfMP = md[md.duplicated(['MatchingID'], keep=False)]
703 # f, p = stats.ttest_rel(dfMP[Ev][dfMP[R] == Eq],
704 #                        dfMP[Ev][dfMP[R] == Po])
705
706 data = md # dfMP
707

```

```

708 for i in Ev:
709     pivotEv = pd.pivot_table(md, values=i, index=[A60, R],
710                               aggfunc=pvtOut)
711
712     print('pivotEv')
713     print(pivotEv)
714     # Add the index groups and convert NaN's to "-"'s
715     print(pivotEv.to_latex(index=True, na_rep='-', escape=False,
716                             float_format="{:0.3f}".format))
717
718     # Barplot
719     smartPlot(data=data, x=A60, y=i, hue=None, hue_order=[Aleq60, Ag60], ci=68,
720               errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
721               sigLoc='outside', text_format='star', line_offset=0.015,
722               line_offset_to_box=0.0, line_height=0.015, fontsize=16,
723               legLoc='best', verbose=2, yAxis=None,
724               xlabel=A_G, ylabel=ElasticUnit, legendTitle=None,
725               figName='Age_BarPlot', folderName=Folder,
726               addBoxPair=addBoxPair, stats=True)
727
728     # Boxplot
729     smartPlot(data=data, x=A60, y=i, hue=None, hue_order=[Aleq60, Ag60],
730               plot='boxplot',
731               test='t-test_ind', sigLoc='outside', text_format='star',
732               line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
733               fontsize=16, legLoc='best', verbose=2, yAxis=None,
734               xlabel=A_G, ylabel=ElasticUnit, addBoxPair=addBoxPair,
735               legendTitle=R, figName='Age_BoxPlot', folderName=Folder,
736               stats=True)
737
738     # Boxplot with data
739     smartPlot(data=data, x=A60, y=i, hue=None, hue_order=[Aleq60, Ag60],
740               plot='boxplot',
741               test='t-test_ind', sigLoc='outside', text_format='star',
742               line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
743               fontsize=16, legLoc='best', verbose=2, yAxis=None,
744               xlabel=A_G, ylabel=ElasticUnit, addBoxPair=addBoxPair,
745               legendTitle=R,
746               figName='Age_BoxPlotWithData', folderName=Folder,
747               dataPoints=True, stats=True)
748
749 # In[Elastic modulus group plots by region]
750 EV = 'EV'
751 Ev = [EV]
752 Folder = 'ElasticModulus'
753
754 # Additional group
755 addBoxPair1 = [(Eq, Po)]
756
757 addBoxPair = addBoxPair1
758
759 for i in Ev:
760     pivotEv = pd.pivot_table(md, values=i, index=[A60, R],
761                               aggfunc=pvtOut)
762
763     print('pivotEv')
764     print(pivotEv)
765     # Add the index groups and convert NaN's to "-"'s

```

```

766     print(pivotEv.to_latex(index=True, na_rep='-', escape=False,
767                           float_format="{:0.3f}".format))
768
769     # Barplot
770     smartPlot(data=md, x=R, y=i, hue=None, hue_order=[Eq, Po], ci=68,
771              errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
772              sigLoc='outside', text_format='star', line_offset=0.015,
773              line_offset_to_box=0.0, line_height=0.015, fontsize=16,
774              legLoc='best', verbose=2, yAxi=None,
775              xlabel=A_G, ylabel=ElasticUnit, legendTitle=None,
776              figName='Age_BarPlot', folderName=Folder,
777              addBoxPair=addBoxPair, stats=True)
778
779     # Boxplot
780     smartPlot(data=md, x=R, y=i, hue=None, hue_order=[Eq, Po], plot='boxplot',
781              test='t-test_ind', sigLoc='outside', text_format='star',
782              line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
783              fontsize=16, legLoc='best', verbose=2, yAxi=None,
784              xlabel=A_G, ylabel=ElasticUnit, addBoxPair=addBoxPair,
785              legendTitle=R, figName='Age_BoxPlot', folderName=Folder,
786              stats=True)
787
788     # Boxplot with data
789     smartPlot(data=md, x=R, y=i, hue=None, hue_order=[Eq, Po], plot='boxplot',
790              test='t-test_ind', sigLoc='outside', text_format='star',
791              line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
792              fontsize=16, legLoc='best', verbose=2, yAxi=None,
793              xlabel=A_G, ylabel=ElasticUnit, addBoxPair=addBoxPair,
794              legendTitle=R,
795              figName='Age_BoxPlotWithData', folderName=Folder,
796              dataPoints=True, stats=True)
797
798     # In[Elastic Modulus Regression by Age in both Regions]
799
800     Folder = 'ElasticModulus'
801
802     # Linear regression
803     f, ax = plt.subplots()
804     sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
805                                "axes.labelsize":12})
806     ax = sns.lmplot(data=md, x=A, y=EV, hue=R, markers=["o", "x"],
807                    legend_out=False, fit_reg=True, height=5, aspect=1.6,
808                    palette="Set1", truncate=True, ci=95, line_kws={'lw':0})
809
810     ax.set(xlabel=A_yrs, ylabel=ElasticUnit)
811
812     ax = ax.axes[0][0] # Convert faceted grid to matplotlib fig
813
814     # Remove all NaN's from the data for regressions
815     # remove nans from ILM thickness & Max
816     df_no_Nan = md.dropna(subset=[A, EV])
817
818     # linear regressions for fitting
819     x = df_no_Nan[A][df_no_Nan[R] == Eq]
820     # Convert to N
821     y = df_no_Nan[EV][df_no_Nan[R] == Eq]
822
823     x_plot = np.linspace(min(x), max(x), 100)

```

```

824
825 slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
826 ax.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
827 ax.text(62, yfit(62) + 5, r'$r={:.4f}$'.format(r_value1), color='r',
828         horizontalalignment='left', fontsize=8, weight='semibold') # r value
829
830 print('Values for correlation between ' +
831       'Elastic Modulus and Age in the Equator\n',
832       'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value1))
833
834 # linear regressions for fitting
835 x = df_no_Nan[A][df_no_Nan[R] == Po]
836 y = df_no_Nan[EV][df_no_Nan[R] == Po]
837
838 x_plot = np.linspace(min(x), max(x), 100)
839 slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
840 ax.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
841 ax.text(62, yfit(62) + 5, r'$r={:.4f}$'.format(r_value2), color='b',
842         horizontalalignment='left', fontsize=8, weight='semibold') # r value
843
844 print('Values for correlation between ' +
845       'Elastic Modulus and Age in the Equator\n',
846       'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value2))
847
848 # Axis limits
849 ax.set(ylim=(0, 500))
850 ax.set(xlim=(29, 80))
851
852 # New path
853 NP = os.path.join(SF, Folder)
854
855 # Create folder if it doesn't exist
856 os.makedirs(NP, exist_ok=True)
857
858 plt.savefig(os.path.join(NP, 'Regression_Age_by_Region.pdf'),
859            bbox_inches='tight')
860 plt.close()
861
862
863 # In[Elastic Modulus Regression by Max Peel Force in both regions]
864
865 Fmax = 'SimMax'
866 Folder = 'ElasticModulus'
867
868 # Linear regression
869 f, ax = plt.subplots()
870 sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
871                             "axes.labelsize":12})
872 ax = sns.lmplot(data=md, x=EV, y=Fmax, hue=R, hue_order= [Eq, Po],
873                markers=["o", "x"], legend_out=False, fit_reg=True,
874                height=5, aspect=1.6, palette="Set1", truncate=True, ci=95,
875                line_kws={'lw':0})
876
877 ax.set(xlabel=ElasticUnit, ylabel=MPF)
878
879 ax = ax.axes[0][0] # Convert faceted grid to matplotlib fig
880
881 # Remove all NaN's from the data for regressions

```

```

882 # remove nans from ILM thickness & Max
883 df_no_Nan = md.dropna(subset=[Fmax, EV])
884
885 # linear regressions for fitting
886 x = df_no_Nan[EV][df_no_Nan[R] == Eq]
887 # Convert to N
888 y = df_no_Nan[Fmax][df_no_Nan[R] == Eq]
889
890 x_plot = np.linspace(min(x), max(x), 100)
891
892 slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
893 ax.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
894 ax.text(400, yfit(400) - 1, r'$r={:.4f}$'.format(r_value1), color='r',
895         horizontalalignment='left', fontsize=8, weight='semibold') # r value
896
897 print('Values for correlation between ' +
898       'Elastic Modulus and Max Force in the Equator\n',
899       'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value1))
900
901 # linear regressions for fitting
902 x = df_no_Nan[EV][df_no_Nan[R] == Po]
903 y = df_no_Nan[Fmax][df_no_Nan[R] == Po]
904
905 x_plot = np.linspace(min(x), max(x), 100)
906 slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
907 ax.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
908 ax.text(220, yfit(220) - 1, r'$r={:.4f}$'.format(r_value2), color='b',
909         horizontalalignment='left', fontsize=8, weight='semibold') # r value
910
911 print('Values for correlation between ' +
912       'Elastic Modulus and Max Force in the Equator\n',
913       'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value2))
914
915 # Axis limits
916 ax.set(ylim=(0, 18))
917 ax.set(xlim=(0, 500))
918
919 # New path
920 NP = os.path.join(SF, Folder)
921
922 # Create folder if it doesn't exist
923 os.makedirs(NP, exist_ok=True)
924
925 plt.savefig(os.path.join(NP, 'Regression_MaxForce_by_Region.pdf'),
926            bbox_inches='tight')
927 plt.close()
928
929
930 # In[Elastic Modulus Regression by Steady-State Peel Force in both regions]
931
932 FSS = 'SimSS'
933
934 # Linear regression
935 f, ax = plt.subplots()
936 sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
937                             "axes.labelsize":12})
938 ax = sns.lmplot(data=md, x=EV, y=FSS, hue=R, hue_order=[Eq, Po],
939                markers=["o", "x"], legend_out=False, fit_reg=True,

```

```

940         height=5, aspect=1.6, palette="Set1", truncate=True, ci=95,
941         line_kws={'lw':0})
942
943 ax.set(xlabel=ElasticUnit, ylabel=SSPF)
944
945 ax = ax.axes[0][0] # Convert faceted grid to matplotlib fig
946
947 # Remove all NaN's from the data for regressions
948 # remove nans from ILM thickness & Max
949 df_no_Nan = md.dropna(subset=[FSS, EV])
950
951 # linear regressions for fitting
952 x = df_no_Nan[EV][df_no_Nan[R] == Eq]
953 # Convert to N
954 y = df_no_Nan[FSS][df_no_Nan[R] == Eq]
955
956 x_plot = np.linspace(min(x), max(x), 100)
957
958 slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
959 ax.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
960 ax.text(400, yfit(400) - 1, r'$r={:.4f}$'.format(r_value1), color='r',
961         horizontalalignment='left', fontsize=8, weight='semibold') # r value
962
963 print('Values for correlation between ' +
964       'Elastic Modulus and SS Force in the Equator\n',
965       'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value1))
966
967 # linear regressions for fitting
968 x = df_no_Nan[EV][df_no_Nan[R] == Po]
969 y = df_no_Nan[FSS][df_no_Nan[R] == Po]
970
971 x_plot = np.linspace(min(x), max(x), 100)
972 slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
973 ax.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
974 ax.text(220, yfit(220) - 1, r'$r={:.4f}$'.format(r_value2), color='b',
975         horizontalalignment='left', fontsize=8, weight='semibold') # r value
976
977 print('Values for correlation between ' +
978       'Elastic Modulus and SS Force in the Equator\n',
979       'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value2))
980
981 # Axis limits
982 ax.set(ylim=(0, 12))
983 ax.set(xlim=(0, 500))
984
985 # New path
986 NP = os.path.join(SF, 'ElasticModulus')
987
988 # Create folder if it doesn't exist
989 os.makedirs(NP, exist_ok=True)
990
991 plt.savefig(os.path.join(NP, 'Regression_SSForce_by_Region.pdf'),
992            bbox_inches='tight')
993 plt.close()
994
995 # In[Cohesive parameter group plots]
996
997 Knn = 'Knn'

```



```

998 Kss = 'Kss'
999 Ktt = 'Ktt'
1000
1001 Folder = 'CohesiveBehavior'
1002
1003 # Filter data (Brittany)
1004 dfKnn = md[md[Knn] > 25e6]
1005 dfKss = md[md[Kss] > 100e6]
1006 dfKtt = md[md[Ktt] > 150e6]
1007
1008 dfFilt = {Knn: dfKnn,
1009           Kss: dfKss,
1010           Ktt: dfKtt}
1011
1012 for key, val in dfFilt.items():
1013     pivotCohBeh = pd.pivot_table(val, values=key, index=[A60, R],
1014                                   aggfunc=pvtOut)
1015
1016     print('pivotCohBeh')
1017     print(pivotCohBeh)
1018     # Add the index groups and convert NaN's to "-"'s
1019     print(pivotCohBeh.to_latex(index=True, na_rep='-', escape=False,
1020                                float_format="{:0.3f}".format))
1021
1022     # Barplot
1023     smartPlot(data=val, x=A60, y=key, hue=R, hue_order=[Eq, Po], ci=68,
1024               errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
1025               sigLoc='outside', text_format='star', line_offset=0.015,
1026               line_offset_to_box=0.0, line_height=0.015, fontsize='small',
1027               legLoc='best', verbose=2, yAxis='log',
1028               xlabel=A_G, ylabel=CohBehUnit, legendTitle=R,
1029               figName=f'Region_BarPlot_{key}', folderName=Folder)
1030
1031     # Boxplot
1032     smartPlot(data=val, x=A60, y=key, hue=R, hue_order=[Eq, Po],
1033               plot='boxplot',
1034               test='t-test_ind', sigLoc='outside', text_format='star',
1035               line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
1036               fontsize='small', legLoc='best', verbose=2, yAxis='log',
1037               xlabel=A_G, ylabel=CohBehUnit,
1038               legendTitle=R, figName=f'Region_BoxPlot_{key}',
1039               folderName=Folder)
1040
1041     # Boxplot with data
1042     smartPlot(data=val, x=A60, y=key, hue=R, hue_order=[Eq, Po],
1043               plot='boxplot',
1044               test='t-test_ind', sigLoc='outside', text_format='star',
1045               line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
1046               fontsize='small', legLoc='best', verbose=2, yAxis='log',
1047               xlabel=A_G, ylabel=CohBehUnit,
1048               legendTitle=R,
1049               figName=f'Region_BoxPlotWithData_{key}', folderName=Folder,
1050               dataPoints=True)
1051
1052 # In[Kss Regression by Max Peel Force in both regions]
1053
1054 Fmax = 'SimMax'
1055 Folder = 'CohesiveBehavior'

```

```

1056
1057 # Linear regression
1058 f, ax = plt.subplots()
1059 sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
1060                             "axes.labelsize":12})
1061 ax = sns.lmplot(data=dfKnn, x='Kss', y=Fmax, hue=R, hue_order= [Eq, Po],
1062                 markers=["o", "x"], legend_out=False, fit_reg=True,
1063                 height=5, aspect=1.6, palette="Set1", truncate=True, ci=95,
1064                 line_kws={'lw':0})
1065
1066 ax.set(xscale="log")
1067
1068 ax.set(xlabel=CohBehUnit, ylabel=MPF)
1069
1070 ax = ax.axes[0][0] # Convert faceted grid to matplotlib fig
1071
1072 # Remove all NaN's from the data for regressions
1073 # remove nans from ILM thickness & Max
1074 df_no_Nan = dfKnn.dropna(subset=[Fmax, 'Kss'])
1075
1076 # linear regressions for fitting
1077 x = df_no_Nan['Kss'][df_no_Nan[R] == Eq]
1078 # Convert to N
1079 y = df_no_Nan[Fmax][df_no_Nan[R] == Eq]
1080
1081 x_plot = np.linspace(min(x), max(x), 100)
1082
1083 slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
1084 ax.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
1085 ax.text(5e7, yfit(5e7) + 1, r'$r={:.4f}$'.format(r_value1), color='r',
1086         horizontalalignment='left', fontsize=8, weight='semibold') # r value
1087
1088 print('Values for correlation between ' +
1089       'Kss and Max Force in the Equator\n',
1090       'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value1))
1091
1092 # linear regressions for fitting
1093 x = df_no_Nan['Kss'][df_no_Nan[R] == Po]
1094 y = df_no_Nan[Fmax][df_no_Nan[R] == Po]
1095
1096 x_plot = np.linspace(min(x), max(x), 100)
1097 slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
1098 ax.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
1099 ax.text(6e7, yfit(6e7) + 1, r'$r={:.4f}$'.format(r_value2), color='b',
1100         horizontalalignment='left', fontsize=8, weight='semibold') # r value
1101
1102 print('Values for correlation between ' +
1103       'Kss and Max Force in the Equator\n',
1104       'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value2))
1105
1106 # Axis limits
1107 # ax.set(ylim=(0, 18))
1108 ax.set(xlim=(3e7, 9e7))
1109
1110 # New path
1111 NP = os.path.join(SF, Folder)
1112
1113 # Create folder if it doesn't exist

```

```

1114 os.makedirs(NP, exist_ok=True)
1115
1116 plt.savefig(os.path.join(NP, 'Kss_vs_MaxForce_by_Region.pdf'),
1117             bbox_inches='tight')
1118 plt.close()
1119
1120 # In[Kss Regression by Steady State Peel Force in both regions]
1121
1122 FSS = 'SimSS'
1123 Folder = 'CohesiveBehavior'
1124
1125 # Linear regression
1126 f, ax = plt.subplots()
1127 sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
1128                             "axes.labelsize":12})
1129 ax = sns.lmplot(data=dfKnn, x='Kss', y=FSS, hue=R, hue_order= [Eq, Po],
1130                 markers=["o", "x"], legend_out=False, fit_reg=True,
1131                 height=5, aspect=1.6, palette="Set1", truncate=True, ci=95,
1132                 line_kws={'lw':0})
1133
1134 ax.set(xscale="log")
1135
1136 ax.set(xlabel=CohBehUnit, ylabel=SSPF)
1137
1138 ax = ax.axes[0][0] # Convert faceted grid to matplotlib fig
1139
1140 # Remove all NaN's from the data for regressions
1141 # remove nans from ILM thickness & Max
1142 df_no_Nan = dfKnn.dropna(subset=[FSS, 'Kss'])
1143
1144 # linear regressions for fitting
1145 x = df_no_Nan['Kss'][df_no_Nan[R] == Eq]
1146 # Convert to N
1147 y = df_no_Nan[FSS][df_no_Nan[R] == Eq]
1148
1149 x_plot = np.linspace(min(x), max(x), 100)
1150
1151 slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
1152 ax.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
1153 ax.text(5e7, yfit(5e7) + 1, r'$r={:.4f}$'.format(r_value1), color='r',
1154         horizontalalignment='left', fontsize=8, weight='semibold') # r value
1155
1156 print('Values for correlation between ' +
1157       'Kss and Steady State Peel Force in the Equator\n',
1158       'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value1))
1159
1160 # linear regressions for fitting
1161 x = df_no_Nan['Kss'][df_no_Nan[R] == Po]
1162 y = df_no_Nan[FSS][df_no_Nan[R] == Po]
1163
1164 x_plot = np.linspace(min(x), max(x), 100)
1165 slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
1166 ax.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
1167 ax.text(6e7, yfit(6e7) + 1, r'$r={:.4f}$'.format(r_value2), color='b',
1168         horizontalalignment='left', fontsize=8, weight='semibold') # r value
1169
1170 print('Values for correlation between ' +
1171       'Kss and Steady State Peel Force in the Equator\n',

```

```

1172         'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value2))
1173
1174     # Axis limits
1175     # ax.set(ylim=(0, 18))
1176     ax.set(xlim=(3e7, 9e7))
1177
1178     # New path
1179     NP = os.path.join(SF, Folder)
1180
1181     # Create folder if it doesn't exist
1182     os.makedirs(NP, exist_ok=True)
1183
1184     plt.savefig(os.path.join(NP, 'Kss_vs_SSForce_by_Region.pdf'),
1185                 bbox_inches='tight')
1186     plt.close()
1187
1188     # In[Cohesive Damage Initiation parameter group plots]
1189
1190     tn = 'tn'
1191     ts = 'ts'
1192     tt = 'tt'
1193
1194     Folder = 'CohesiveDamage'
1195
1196     # Filter data (Brittany)
1197     dftn = md[md[tn] < 3000]
1198     dfts = md[md[ts] < 3000]
1199     dftt = md[md[tt] < 3000]
1200
1201     dfFilt = {tn: dftn,
1202               ts: dfts,
1203               tt: dftt}
1204
1205     for key, val in dfFilt.items():
1206         pivotCohDMG = pd.pivot_table(val, values=key, index=[A60, R],
1207                                     aggfunc=pvtOut)
1208
1209         print('pivotCohDMG')
1210         print(pivotCohDMG)
1211         # Add the index groups and convert NaN's to "-"'s
1212         print(pivotCohDMG.to_latex(index=True, na_rep='-', escape=False,
1213                                   float_format="{:0.3f}".format))
1214
1215         # Barplot
1216         smartPlot(data=val, x=A60, y=key, hue=R, hue_order=[Eq, Po], ci=68,
1217                  errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
1218                  sigLoc='outside', text_format='star', line_offset=0.015,
1219                  line_offset_to_box=0.0, line_height=0.015, fontsize='small',
1220                  legLoc='best', verbose=2, yAxis=None,
1221                  xlabel=A_G, ylabel=CohDMGUnit, legendTitle=R,
1222                  figName=f'Region_BarPlot_{key}', folderName=Folder)
1223
1224         # Boxplot
1225         smartPlot(data=val, x=A60, y=key, hue=R, hue_order=[Eq, Po],
1226                  plot='boxplot', test='t-test_ind', sigLoc='outside',
1227                  text_format='star', line_offset=0.015, line_offset_to_box=0.0,
1228                  line_height=0.015,
1229                  fontsize='small', legLoc='best', verbose=2, yAxis=None,

```

```

1230         xlabel=A_G, ylabel=CohDMGUnit,
1231         legendTitle=R, figName=f'Region_BoxPlot_{key}', folderName=Folder)
1232
1233     # Boxplot with data
1234     smartPlot(data=val, x=A60, y=key, hue=R, hue_order=[Eq, Po],
1235               plot='boxplot', test='t-test_ind', sigLoc='outside',
1236               text_format='star', line_offset=0.015, line_offset_to_box=0.0,
1237               line_height=0.015,
1238               fontsize='small', legLoc='best', verbose=2, yAxis=None,
1239               xlabel=A_G, ylabel=CohDMGUnit,
1240               legendTitle=R,
1241               figName=f'Region_BoxPlotWithData_{key}', folderName=Folder,
1242               dataPoints=True)
1243
1244     # In[ts Regression by Max Peel Force in both regions]
1245
1246
1247     mod = sm.OLS(df['VE'], df[Fmax])
1248     res = mod.fit()
1249     print(80*'- ', 2*'\n', 'Correlation between Age & E\n', res.summary())
1250
1251     text_file = open('Correlation.txt', "w")
1252     text_file.write(res.summary().as_text())
1253     text_file.close()
1254
1255
1256     stats.ttest_ind(df['VE'], df[Fmax])
1257
1258
1259     Fmax = 'SimMax'
1260     Folder = 'CohesiveDamage'
1261
1262     # Linear regression
1263     f, ax = plt.subplots()
1264     sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
1265                                "axes.labelsize":12})
1266     ax = sns.lmplot(data=dfts, x=ts, y=Fmax, hue=R, hue_order= [Eq, Po],
1267                    markers=["o", "x"], legend_out=False, fit_reg=True,
1268                    height=5, aspect=1.6, palette="Set1", truncate=True, ci=95,
1269                    line_kws={'lw':0})
1270
1271     # ax.set(xscale="log")
1272
1273     ax.set(xlabel=CohDMGUnit, ylabel=MPF)
1274
1275     ax = ax.axes[0][0] # Convert faceted grid to matplotlib fig
1276
1277     # Remove all NaN's from the data for regressions
1278     # remove nans from ILM thickness & Max
1279     df_no_Nan = dfts.dropna(subset=[Fmax, ts])
1280
1281     # linear regressions for fitting
1282     x = df_no_Nan[ts][df_no_Nan[R] == Eq]
1283     # Convert to N
1284     y = df_no_Nan[Fmax][df_no_Nan[R] == Eq]
1285
1286     x_plot = np.linspace(min(x), max(x), 100)
1287

```

```

1288 xt = 3*10**2 # Location of text
1289
1290 slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
1291 ax.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
1292 plt.text(xt, yfit(xt) + 1, r'$r={:.4f}$'.format(r_value1), color='r',
1293          horizontalalignment='left', fontsize=8, weight='semibold') # r value
1294
1295 print('Values for correlation between ' +
1296       'ts and Max Force in the Equator\n',
1297       'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value1))
1298
1299 # linear regressions for fitting
1300 x = df_no_Nan[ts][df_no_Nan[R] == Po]
1301 y = df_no_Nan[Fmax][df_no_Nan[R] == Po]
1302
1303 xt = 8*10**2 # Location of text
1304
1305 x_plot = np.linspace(min(x), max(x), 100)
1306 slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
1307 ax.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
1308 plt.text(xt, yfit(xt) + 1, r'$r={:.4f}$'.format(r_value2), color='b',
1309          horizontalalignment='left', fontsize=8, weight='semibold') # r value
1310
1311 print('Values for correlation between ' +
1312       'ts and Max Force in the Equator\n',
1313       'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value2))
1314
1315 # Axis limits
1316 # ax.set(ylim=(0, 18))
1317 # ax.set(xlim=(3e7, 9e7))
1318 ax.set(xlim=(1.8*10**2, 1.3*10**3))
1319
1320 # New path
1321 NP = os.path.join(SF, Folder)
1322
1323 # Create folder if it doesn't exist
1324 os.makedirs(NP, exist_ok=True)
1325
1326 plt.savefig(os.path.join(NP, 'ts_vs_MaxForce_by_Region.pdf'),
1327            bbox_inches='tight')
1328 plt.close()
1329
1330 # In[tt Regression by Max Peel Force in both regions]
1331
1332 Fmax = 'SimMax'
1333 Folder = 'CohesiveDamage'
1334
1335 # Linear regression
1336 f, ax = plt.subplots()
1337 sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
1338                             "axes.labelsize":12})
1339 ax = sns.lmplot(data=dftt, x=tt, y=Fmax, hue=R, hue_order= [Eq, Po],
1340                markers=["o", "x"], legend_out=False, fit_reg=True,
1341                height=5, aspect=1.6, palette="Set1", truncate=True, ci=95,
1342                line_kws={'lw':0})
1343
1344 # ax.set(xscale="log")
1345

```

```

1346 ax.set(xlabel=CohDMGUnit, ylabel=MPF)
1347
1348 ax = ax.axes[0][0] # Convert faceted grid to matplotlib fig
1349
1350 # Remove all NaN's from the data for regressions
1351 # remove nans from ILM thickness & Max
1352 df_no_Nan = dftt.dropna(subset=[Fmax, tt])
1353
1354 # linear regressions for fitting
1355 x = df_no_Nan[tt][df_no_Nan[R] == Eq]
1356 # Convert to N
1357 y = df_no_Nan[Fmax][df_no_Nan[R] == Eq]
1358
1359 x_plot = np.linspace(min(x), max(x), 100)
1360
1361 xt = 2*10**2 # Location of text
1362
1363 slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
1364 ax.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
1365 ax.text(xt, yfit(xt) + 2, r'$r={:.4f}$'.format(r_value1), color='r',
1366         horizontalalignment='left', fontsize=8, weight='semibold') # r value
1367
1368 print('Values for correlation between ' +
1369       'tt and Max Force in the Equator\n',
1370       'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value1))
1371
1372 # linear regressions for fitting
1373 x = df_no_Nan[tt][df_no_Nan[R] == Po]
1374 y = df_no_Nan[Fmax][df_no_Nan[R] == Po]
1375
1376 xt = 1.2*10**3 # Location of text
1377
1378 x_plot = np.linspace(min(x), max(x), 100)
1379 slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
1380 ax.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
1381 ax.text(xt, yfit(xt) + 1, r'$r={:.4f}$'.format(r_value2), color='b',
1382         horizontalalignment='left', fontsize=8, weight='semibold') # r value
1383
1384 print('Values for correlation between ' +
1385       'tt and Max Force in the Equator\n',
1386       'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value2))
1387
1388 # Axis limits
1389 ax.set(ylim=(0, None))
1390 ax.set(xlim=(1.*10**1, 2*10**3))
1391
1392 # New path
1393 NP = os.path.join(SF, Folder)
1394
1395 # Create folder if it doesn't exist
1396 os.makedirs(NP, exist_ok=True)
1397
1398 plt.savefig(os.path.join(NP, 'tt_vs_MaxForce_by_Region.pdf'),
1399            bbox_inches='tight')
1400 plt.close()
1401
1402 # In[tt Regression by Age regions]
1403

```

```

1404 Folder = 'CohesiveDamage'
1405
1406 Aleq63 = r'Age $\leq$ 63'
1407 Ag63 = 'Age $>$ 63'
1408
1409 bins = [0, 63, 90]
1410 labelsAge63 = [Aleq63, Ag63]
1411
1412 # Create binned AgeGroups
1413 A63 = 'A63'
1414 md[A63] = pd.cut(md[A], bins, labels=labelsAge63, right=True)
1415
1416 dftt = md[md[tt] < 3000]
1417
1418 # Linear regression
1419 f, ax = plt.subplots()
1420 # sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
1421 #                               "axes.labelsize":12})
1422
1423 # scatter plot
1424 # Averages for the +/- 63 age group
1425 y_less_63 = dftt[tt][(dftt[A] <= 63)].dropna()
1426 y_greater_63 = dftt[tt][(dftt[A] > 63)].dropna()
1427
1428 f, ax = plt.subplots()
1429 ax = sns.scatterplot(data=dftt, x=A, y=tt, hue=A63, style=A63,
1430                     hue_order=None,
1431                     palette='Set1', s=200, legend=True)
1432
1433 ax.get_legend_handles_labels()[0][0]._sizes = [200.]
1434 ax.get_legend_handles_labels()[0][1]._sizes = [200.]
1435
1436 legend = ax.legend(loc='best', fontsize=18, title=A_G)
1437
1438 plt.setp(legend.get_title(), fontsize=18)
1439
1440 # Axis labels
1441 ax.set_xlabel(A_yrs, fontsize=18)
1442 ax.set_ylabel(CohDMGUnit, fontsize=18)
1443
1444
1445 x_plot_less_63 = np.linspace(30, 63, 100)
1446 x_plot_greater_63 = np.linspace(63, 80, 100)
1447
1448 # Plot averages
1449 plt.plot(x_plot_less_63, np.mean(y_less_63)*np.ones(len(x_plot_less_63)),
1450         '-.', color='r', linewidth=3) # , label=r'Age $\leq$ 60 AVG')
1451
1452 ax.text(np.mean(x_plot_less_63)*1.1, np.mean(y_less_63) + 50,
1453         r'Average', color='r', horizontalalignment='left',
1454         fontsize=18, weight='semibold')
1455
1456 plt.plot(x_plot_greater_63,
1457         np.mean(y_greater_63)*np.ones(len(x_plot_greater_63)), '-.',
1458         color='b', linewidth=3) # , label=r'Age $>$ 60 AVG')
1459
1460 ax.text(np.mean(x_plot_greater_63)*0.9, np.mean(y_greater_63) - 75,
1461         r'Average', color='b', horizontalalignment='left',

```



```

1462         fontsize=18, weight='semibold')
1463
1464 ax.tick_params(axis='x', labelsiz=14)
1465 ax.tick_params(axis='y', labelsiz=14)
1466
1467 # New path
1468 NP = os.path.join(SF, Folder)
1469
1470 # Create folder if it doesn't exist
1471 os.makedirs(NP, exist_ok=True)
1472
1473 plt.savefig(os.path.join(NP, 'tt_vs_Age.pdf'), bbox_inches='tight')
1474
1475 plt.close()
1476 # In[Fracture Energy group plots]
1477
1478 FE = 'FE'
1479
1480 Folder = 'FractureEnergy'
1481
1482 # Filter data (Brittany)
1483 dfFE = md[md[FE] < 0.0009]
1484
1485 dfFilt = {FE: dfFE}
1486
1487 for key, val in dfFilt.items():
1488
1489     pivotFE = pd.pivot_table(val, values=key, index=[A60, R],
1490                               aggfunc=pvtOut)
1491
1492     print('pivotFE')
1493     print(pivotFE)
1494     # Add the index groups and convert NaN's to "-"'s
1495     print(pivotFE.to_latex(index=True, na_rep='-', escape=False,
1496                             float_format="{:0.3f}".format))
1497
1498     # Barplot
1499     smartPlot(data=val, x=A60, y=key, hue=R, hue_order=[Eq, Po], ci=68,
1500               errcolor='black', capsiz=.2, plot='barplot', test='t-test_ind',
1501               sigLoc='outside', text_format='star', line_offset=0.015,
1502               line_offset_to_box=0.0, line_height=0.015, fontsize='small',
1503               legLoc='best', verbose=2, yAxi='log',
1504               xlabel=A_G, ylabel=FEUnit, legendTitle=R,
1505               figName='Region_BarPlot', folderName=Folder)
1506
1507     # Boxplot
1508     smartPlot(data=val, x=A60, y=key, hue=R, hue_order=[Eq, Po], plot='boxplot',
1509               test='t-test_ind', sigLoc='outside', text_format='star',
1510               line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
1511               fontsize='small', legLoc='best', verbose=2, yAxi='log',
1512               xlabel=A_G, ylabel=FEUnit,
1513               legendTitle=R, figName='Region_BoxPlot', folderName=Folder)
1514
1515     # Boxplot with data
1516     smartPlot(data=val, x=A60, y=key, hue=R, hue_order=[Eq, Po], plot='boxplot',
1517               test='t-test_ind', sigLoc='outside', text_format='star',
1518               line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
1519               fontsize='small', legLoc='best', verbose=2, yAxi='log',

```

```

1520         xlabel=A_G, ylabel=FEUnit,
1521         legendTitle=R,
1522         figName='Region_BoxPlotWithData', folderName=Folder,
1523         dataPoints=True)
1524
1525
1526 # In[Summary Convergence Table Elastic]
1527
1528 # value summary
1529 simList = []
1530
1531 for key, val in ElasticSummary.items():
1532     d = val[1:] # Subset - skip first row
1533     d['simTime'] = pd.to_numeric(d['simTime'], downcast="float")
1534
1535     L = len(d.index)
1536     s = np.sum(d['simTime'])
1537     avg = s/L
1538
1539     simList.append([L, s, avg])
1540
1541 simDF = pd.DataFrame(simList, columns=['N', 'TotalTime', 'AVGTime'])
1542
1543 print(np.mean(simDF['N']),
1544       np.mean(simDF['TotalTime']),
1545       np.mean(simDF['AVGTime']))
1546
1547 # In[Summary Convergence Table Cohesive]
1548
1549 # value summary
1550 simList = []
1551
1552 for key, val in CohesiveSummary.items():
1553     d = val[1:] # Subset - skip first row
1554     d['simTime'] = pd.to_numeric(d['simTime'], downcast="float")
1555
1556     L = len(d.index)
1557     s = np.sum(d['simTime'])
1558     avg = s/L
1559
1560     simList.append([L, s, avg])
1561
1562 simDF = pd.DataFrame(simList, columns=['N', 'TotalTime', 'AVGTime'])
1563
1564 print(np.mean(simDF['N']),
1565       np.mean(simDF['TotalTime']),
1566       np.mean(simDF['AVGTime']))

```

## 1.8.2 Visualization Distributions

</> **Script 22:** Post simulation python script creates simulation result distributions. </>

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Wed Apr 28 23:59:51 2021
4

```

```

5  @author: Kiffer
6  """
7
8  import pandas as pd
9  import numpy as np
10 import seaborn as sns
11 from statannot import add_stat_annotation
12 import matplotlib.pyplot as plt
13 plt.rcParams['figure.figsize'] = [16, 10]
14 from scipy import stats
15 import pdb
16 import os
17 import glob
18 import re
19 from scipy import stats
20
21 cwd = os.getcwd()
22
23 SF = os.path.join('Results', 'StatisticsFigures')
24
25 # Create folder if it doesn't exist
26 os.makedirs(SF, exist_ok=True)
27
28 # In[KDE plot function]
29
30 def KDEplot(data=None, x=None, hue=None, hue_order=None,
31             Regions=None, figName=None, legendTitle=None, legendLoc=None,
32             xlabel=None, ylabel=None, bw_adjust=None, alpha=None,
33             initGuess=None, constraints=None, folderName=None,
34             optLegendLoc=None, bounds=None):
35     colors = [plt.cm.tab10.colors[i:i + 2] for i in
36               range(0, len(data[R].unique()) * 2, 2)]
37     hatches = ['', '/////']
38
39     f, ax = plt.subplots(figsize=(9.6, 6))
40     sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
41                                "axes.labelsize":12})
42     handles = []
43     for region, palette in zip(Regions, colors):
44
45         # Data subset
46         dataSubset = data[(data[R] == region)]
47
48         # KDE plot
49         ax = sns.kdeplot(data=dataSubset, x=x, hue=hue,
50                          hue_order=hue_order, multiple='stack', fill=True,
51                          palette=palette, ax=ax, log_scale=True,
52                          alpha=alpha, bw_adjust=bw_adjust)
53         # pdb.set_trace()
54         for h, age, hatch in zip(ax.legend_.legendHandles, hue_order,
55                                 hatches):
56             h.set_label(f'{region}, {age}')
57             h.set_hatch(hatch)
58             handles.append(h)
59
60     extra = []
61     if initGuess != None:

```

```

63     ax.axvline(x = initGuess, color='black', linestyle='-',
64                linewidth=1, label=r'Initial Guess')
65     extra.append(0)
66
67     if bounds != None and bounds == True:
68         # Only add bounds if "True"
69         if constraints != None:
70             ax.axvline(x = constraints[0], color='black', linestyle=':',
71                        linewidth=1, label=r'Lower Bound', ymax=0.4)
72             extra.append(1)
73             ax.axvline(x = constraints[1], color='black', linestyle='--',
74                        linewidth=1, label=r'Upper Bound', ymax=0.4)
75             extra.append(2)
76
77     ax.legend_.remove() # remove the automatic legends
78     ax.set(xlabel=xlabel, ylabel=ylabel)
79     for collection, hatch in zip(ax.collections[:: -1],
80                                hatches * len(Regions)):
81         collection.set_hatch(hatch)
82
83     # Add bounds
84     if initGuess != None and optLegendLoc !=None:
85         lines = f.gca().get_lines()
86         # pdb.set_trace()
87         legend2 = ax.legend([lines[i] for i in extra],
88                             [lines[i].get_label() for i in extra],
89                             prop={"size":10}, loc=optLegendLoc,
90                             title='Optimization')
91         ax.add_artist(legend2)
92
93     legend1 = ax.legend(handles=handles, loc=legendLoc).set_title(legendTitle)
94
95     # This doesn't work using the method so the bounds need to be plotted
96     # before the custom legend with handles
97     # plt.gca().add_artist(legend1)
98
99     if folderName != None:
100         # If a new folder name is given, put the files there
101
102         # New file path
103         NP = os.path.join(SF, folderName)
104
105         # Create folder if it doesn't exist
106         os.makedirs(NP, exist_ok=True)
107
108     else:
109         # Put the file in the same folder
110         NP = SF
111
112     f.savefig(os.path.join(NP, f'{figName}_{x}.pdf'),
113              bbox_inches='tight')
114
115     plt.close(f)
116
117     return f, ax
118
119
120 # In[Plot simplifications]

```

```

121
122 R = 'Region'
123 Eq = 'Equator'
124 Po = 'Posterior'
125 AG = 'AgeGroup'
126 A60 = 'Age60'
127 Aleq60 = r'Age $\leq$ 60'
128 Ag60 = 'Age $>$ 60'
129 A = 'Age'
130
131 # Units
132 MPF = 'Maximum Peel Force (mN)'
133 SSPF = 'Steady-State Peel Force (mN)'
134 KDEUnit = r'Kernel Density Estimation'
135 ElasticUnit = r'Elastic Modulus (Pa)'
136 CohBehUnit = r'Cohesive Behavior (Pa)'
137 CohDMGUnit = r'Cohesive Damage Initiation (Pa)'
138 FEUnit = r'Fracture Energy (J)'
139
140 A_yrs = 'Age (yr.)'
141 A_G = 'Age Group (yr.)'
142
143 # In[Load data]
144
145 data = os.path.join(cwd, 'Results', 'OutputFiles', 'ExpSimSummary.txt')
146 df = pd.read_csv(data, sep = '\t')
147
148 df.loc[df[R] == 'Eq.', R] = Eq
149 df.loc[df[R] == 'Po.', R] = Po
150
151 # In[Elastic Modulus distribution]
152
153 Folder = 'ElasticModulus'
154
155 # Normal distribution plots
156 f, ax = plt.subplots(figsize=(9.6, 6))
157 sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
158                             "axes.labelsize":12})
159 ax = sns.kdeplot(data=df, x='EV', fill=True, legend=False, palette='Paired',
160                  cut=0, bw_adjust=0.5)
161
162 ax.set(xlabel=ElasticUnit, ylabel=KDEUnit)
163
164 # New path
165 NP = os.path.join(SF, Folder)
166
167 # Make folder if it doesn't exist
168 os.makedirs(NP, exist_ok=True)
169
170 plt.savefig(os.path.join(NP, 'Distribution_AllData.pdf'),
171             bbox_inches='tight')
172
173 # In[Distributions of cohesive parameters]
174
175 alpha = 0.4
176 bw_adjust = 0.8 # Normal distribution smoothing (smaller is less smooth)
177 figName = 'Optimization'
178 F1 = 'ElasticModulus'

```

```

179 F2 = 'CohesiveBehavior'
180 F3 = 'CohesiveDamage'
181 F4 = 'FractureEnergy'
182
183 # Bandwidth selector
184 bwE = 1.06*np.std(df['EV'])*np.count_nonzero(df['EV'])**(-1/5)
185 bwE = 0.9*min(np.std(df['EV']),
186               stats.iqr(df['EV'])*np.count_nonzero(df['EV'])**(-1/5))
187
188 bwKnn = 0.9*min(np.std(df['Knn']),
189                 stats.iqr(df['Knn'])*np.count_nonzero(df['Knn'])**(-1/5))
190
191 # Elastic Modulus
192 f, ax = KDEplot(data=df, x='EV', hue=A60, hue_order=[Aleq60, Ag60],
193                 Regions=[Eq, Po], figName=figName, folderName=F1,
194                 legendLoc='best', legendTitle=R,
195                 xlabel=ElasticUnit, ylabel=KDEUnit,
196                 bw_adjust=bw_adjust, alpha=alpha,
197                 initGuess=172, constraints=[50, 2100],
198                 optLegendLoc='center right', bounds=False)
199
200 # Cohesive Behavior
201 f, ax = KDEplot(data=df, x='Knn', hue=A60, hue_order=[Aleq60, Ag60],
202                 Regions=[Eq, Po], figName=figName, folderName=F2,
203                 legendLoc='best', legendTitle=R,
204                 xlabel=CohBehUnit, ylabel=KDEUnit,
205                 bw_adjust=bw_adjust, alpha=alpha,
206                 initGuess=2**20.872765304828103, constraints=[2**10, 2**28],
207                 optLegendLoc='center right', bounds=False)
208
209 f, ax = KDEplot(data=df, x='Kss', hue=A60, hue_order=[Aleq60, Ag60],
210                 Regions=[Eq, Po], figName=figName, folderName=F2,
211                 legendLoc='best', legendTitle=R,
212                 xlabel=CohBehUnit, ylabel=KDEUnit,
213                 bw_adjust=bw_adjust, alpha=alpha,
214                 initGuess=2**26.094732037712763, constraints=[2**10, 2**28],
215                 optLegendLoc='center right', bounds=False)
216
217 f, ax = KDEplot(data=df, x='Ktt', hue=A60, hue_order=[Aleq60, Ag60],
218                 Regions=[Eq, Po], figName=figName, folderName=F2,
219                 legendLoc='best', legendTitle=R,
220                 xlabel=CohBehUnit, ylabel=KDEUnit,
221                 bw_adjust=bw_adjust, alpha=alpha,
222                 initGuess=2**26.20110650892766, constraints=[2**10, 2**28],
223                 optLegendLoc='center right', bounds=False)
224
225 # Damage Initiation
226 f, ax = KDEplot(data=df, x='tn', hue=A60, hue_order=[Aleq60, Ag60],
227                 Regions=[Eq, Po], figName=figName, folderName=F3,
228                 legendLoc='best', legendTitle=R,
229                 xlabel=CohDMGUnit, ylabel=KDEUnit,
230                 bw_adjust=bw_adjust, alpha=alpha,
231                 initGuess=2**9.712181223168551, constraints=[2**3, 2**20],
232                 optLegendLoc='center right', bounds=False)
233
234 f, ax = KDEplot(data=df, x='ts', hue=A60, hue_order=[Aleq60, Ag60],
235                 Regions=[Eq, Po], figName=figName, folderName=F3,
236                 legendLoc='best', legendTitle=R,

```

```

237         xlabel=CohDMGUnit, ylabel=KDEUnit,
238         bw_adjust=bw_adjust, alpha=alpha,
239         initGuess=2**9.931687876075074, constraints=[2**3, 2**20],
240         optLegendLoc='center right', bounds=False)
241
242 f, ax = KDEplot(data=df, x='tt', hue=A60, hue_order=[A1eq60, Ag60],
243                 Regions=[Eq, Po], figName=figName, folderName=F3,
244                 legendLoc='best', legendTitle=R,
245                 xlabel=CohDMGUnit, ylabel=KDEUnit,
246                 bw_adjust=bw_adjust, alpha=alpha,
247                 initGuess=2**9.022372079206395, constraints=[2**3, 2**20],
248                 optLegendLoc='center right', bounds=False)
249
250 # Fracture Energy
251 f, ax = KDEplot(data=df, x='FE', hue=A60, hue_order=[A1eq60, Ag60],
252                 Regions=[Eq, Po], figName=figName, folderName=F4,
253                 legendLoc='best', legendTitle=R,
254                 xlabel=FEUnit, ylabel=KDEUnit,
255                 bw_adjust=bw_adjust, alpha=alpha,
256                 initGuess=3.738925970000001e-6, constraints=[2**-30, 2**0],
257                 optLegendLoc='center right', bounds=False)
258
259 # In[Stack overflow]
260
261 # fig, axs = plt.subplots(ncols=3, figsize=(15, 3), sharex=True, sharey=True)
262 # f, axs = plt.subplots()
263 # colors = [plt.cm.tab20.colors[i:i + 2] for i in range(0,
264 #             ↳ len(df_CohDmg['Region'].unique()) * 2, 2)]
265 # hatches = ['', '//']
266 # for ax, coh_dmg in zip(axs, ['tn', 'ts', 'tt']):
267 #     handles = []
268 #     # for region, palette in zip([Eq, Po], colors):
269 #     #     sns.kdeplot(data=df_CohDmg[(df_CohDmg['CohDmg'] == coh_dmg) &
270 #     #                               (df_CohDmg['Region'] == region)],
271 #     #                 x='value', hue='Age60', hue_order=[A1eq60, Ag60],
272 #     #                 multiple='stack', palette=palette, ax=ax, log_scale=True,)
273 #     #     for h, age, hatch in zip(ax.legend_.legendHandles, [A1eq60, Ag60],
274 #     #             ↳ hatches):
275 #     #         h.set_label(f'{region}, {age}')
276 #     #         h.set_hatch(hatch)
277 #     #         handles.append(h)
278 #     # ax.legend_.remove() # remove the automatic legends
279 #     # ax.set_title(f'CohDmg={coh_dmg}')
280 #     # for collection, hatch in zip(ax.collections[:-1], hatches * len([Eq, Po])):
281 #     #     collection.set_hatch(hatch)
282
283 # ax.legend(handles=handles, loc='best')
284 # plt.tight_layout()
285
286 # fig.savefig(os.path.join(SF, f'StackOverflow_{coh_dmg}.pdf'),
287 #             ↳ bbox_inches='tight')
288
289 # In[Successful KDF plot example]
290 # colors = [plt.cm.tab20.colors[i:i + 2] for i in
291 #             ↳ range(0, len(df_CohDmg['Region'].unique()) * 2, 2)]
292 # hatches = ['', '//']
293
294 # for i in ['tn', 'ts', 'tt']:

```

```

293 # f, ax = plt.subplots(figsize=(8, 5))
294 # sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
295 #                             "axes.labelsize":12})
296 #
297 # handles = []
298 # for region, palette in zip([Eq, Po], colors):
299 #     ax = sns.kdeplot(data=df_CohDmg[(df_CohDmg['CohDmg'] == i) &
300 #                                     (df_CohDmg['Region'] == region)],
301 #                      x='value', hue='Age60', hue_order=[A1eq60, Ag60],
302 #                      multiple='stack', palette=palette, ax=ax,
303 #                      log_scale=True, alpha=0.9, bw_adjust=0.5)
304 #     for h, age, hatch in zip(ax.legend_.legendHandles, [A1eq60, Ag60],
305 #                             hatches):
306 #         h.set_label(f'{region}, {age}')
307 #         h.set_hatch(hatch)
308 #         handles.append(h)
309 # ax.legend_.remove() # remove the automatic legends
310 # ax.set(xlabel='Elastic Modulus [Pa]',
311 #        ylabel='Kernel Density Estimation')
312 # for collection, hatch in zip(ax.collections[::1],
313 #                               hatches * len([Eq, Po])):
314 #     collection.set_hatch(hatch)
315 #
316 # ax.legend(handles=handles, loc='best')
317 #
318 # f.savefig(os.path.join(SF, f'StackOverflow_{i}.pdf'),
319 #           bbox_inches='tight')

```