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.* </>

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Nov 20 14:20:21 2020

@author: Kiffer
"""

import lmfit as lf # lmfit
import numpy as np # numpy
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [16, 9]
import pandas as pd
import os
import sys
import pdb
filePath = os.getcwd() # Location of Python script

dataSets = ['Jami', 'Lee', 'Kashani', 'Polymer']
dataSet = 1
dataSet = dataSets[dataSet]

def PronyR2(y, fit):
    # R squared calculation
    SS_tot = np.sum((y - np.mean(y))**2)
    SS_res = np.sum((y - fit)**2)
    Rsqd = 1 - SS_res/SS_tot
    return Rsqd

# In[Jami Data for example]
if dataSet == 'Jami':
    #############  Jami Creep Shear Test Data Curve Fit######################
    dataName = 'Jami 2014'
    Jami_data = pd.read_csv('Jami/Jami_Shear_Data.txt', sep="\t", header=0)
    Jami_data.columns = ["Time", "NormShearCreep"]
    Jami_data['NormRelaxData'] = 1/Jami_data.NormShearCreep

    # Invert and compute the raw data
    Raw_data_0 = 0.0214671 # first shear point in raw data
    Jami_data['CreepData'] = Jami_data.NormShearCreep*Raw_data_0
    Jami_data['RelaxData'] = 1/Jami_data['CreepData']
```

```

    # Convert data to array
    t = Jami_data.Time
    # data = Jami_data.NormShearCreep
    # data = Jami_data.NormRelaxData # Normalized relaxation data
    data = Jami_data.CreepData
    # data = Jami_data.RelaxData

    # Units for plotting
    CreepUnits = r'$\left(\mathrm{Pa}\right)$'
    RelaxUnits = r'$\left(\frac{1}{\mathrm{Pa}}\right)$'
    HorizontalUnits = r'Time (s)'

# In[Lee Digitized Data]

if dataSet == 'Lee':
    # Read data from Lee1992 Viscoelastic material properties
    # Digitized from http://getdata-graph-digitizer.com/
    dataName = 'Lee 1992'
    Lee_data = pd.read_csv('Lee1992_DigitizedData.txt', sep="\t", header=3)
    Lee_data.columns = ["Time", "Compliance"]

    t = Lee_data.Time

    # Conversion is dyne/cm^2 to Pa is multiply by 0.1
    data = Lee_data.Compliance*0.1

    # Units for plotting
    CreepUnits = r'$\left(\mathrm{Pa}\right)$'
    RelaxUnits = r'$\left(\frac{1}{\mathrm{Pa}}\right)$'
    HorizontalUnits = r'Time (s)'

# In[Kashani_2011]
if dataSet == 'Kashani':
    # Porcine eyes
    t = np.linspace(1, 600, 1000)

    def modelEqn(t,J1,J2,T1,T2,eta_m):
        """
        Parameters
        ----------
        J1 : TYPE
            DESCRIPTION.
        J2 : TYPE
            DESCRIPTION.
        T1 : TYPE
            DESCRIPTION.
        T2 : TYPE
            DESCRIPTION.
        eta_m : TYPE
            DESCRIPTION.

        Returns
        -------
        Compliance
        """
        return J1*(1 - np.exp(-t/T1)) + J2*(1 - np.exp(-t/T2)) + t/eta_m

```

```
    J1,J2,T1,T2,eta_m = 1.36, 2.64, 1.77, 1.36, 1332.0
    J_0 = modelEqn(1,J1,J2,T1,T2,eta_m) # Initial value
    data = 1/(modelEqn(t,J1,J2,T1,T2,eta_m)/J_0) # invert to define relaxation
    # I'm not sure how this equation works

    # Read data from Lee1992 Viscoelastic material properties
    # Digitized from http://getdata-graph-digitizer.com/
    dataName = 'Kashani 2011'
    Kashani_data = pd.read_csv('KashaniCreepData.csv', sep=",", header=1)
    Kashani_data.columns = ["Time1", "Compliance1",
                            "Time2", "Compliance2",
                            "Time3", "Compliance3"]
    t = Kashani_data.Time1.dropna() # get rid of NaN
    Compliance = Kashani_data.Compliance1.dropna() # Pa # get rid of NaN
    t = Kashani_data.Time2.dropna() # get rid of NaN
    Compliance = Kashani_data.Compliance2.dropna() # Pa # get rid of NaN
    # t = Kashani_data.Time3.dropna() # get rid of NaN
    # Compliance = Kashani_data.Compliance3.dropna() # Pa # get rid of NaN

    data = Compliance

    # Units for plotting
    CreepUnits = r'$\left(\mathrm{Pa}\right)$'
    RelaxUnits = r'$\left(\frac{1}{\mathrm{Pa}}\right)$'
    HorizontalUnits = r'Time (s)'

# In[polymer data]

if dataSet == 'Polymer':
    ###############  Creep Shear Test Data Curve Fit########################
    dataName = 'Polymer 2019'
    subFolder = 'Polymer'
    # df = pd.read_csv(os.path.join(filePath, subFolder,
    #                               'Dogbone1_Test1.csv'),
    #                  sep=",", header=5)
    df = pd.read_csv(os.path.join(filePath, subFolder,
                                  'Dogbone_1_StressRelaxation_2.csv'),
                     sep=",", header=5)
    # df = pd.read_csv(os.path.join(filePath, subFolder,
    #                               'Dogbone1_StressRelaxtion_3.csv'),
    #                  sep=",", header=5)
    df.columns = ["Time", "Extension","Load"]

    # Convert dataframe to array
    time = np.asarray(df['Time'].tolist())
    extension = np.asarray(df['Extension'].tolist())
    load = np.asarray(df['Load'].tolist())

    # Specimen properties
    width = 12/1000.0
    thickness = 5.3/1000.0

    # calculations
    stress = load/(width*thickness)
    strain = extension/100

    # Determine where the stress begins to decrease from the max point in
    # the array
```

```
    StressRelax=stress[np.argmax(stress):-1];
    TimeRelax=time[np.argmax(stress):-1]-time[np.argmax(stress)]

    data = 1/StressRelax
    t = TimeRelax

    # Units for plotting
    CreepUnits = r'$\left(\mathrm{Pa}\right)$'
    RelaxUnits = r'$\left(\frac{1}{\mathrm{Pa}}\right)$'
    HorizontalUnits = r'Displacement (x)'

# In[Lmfit]

def residual(pars, t, data=None):
    """
    Parameters
    ----------
    pars : Ee and E1 and tau2 for Visco terms for the Standard Linear Solid
            Model from Lin2020
    t : time array being passed in
    data : Data to be passed
            through to compare to model data.  The default is None.

    Returns
    -------
    If no data is supplied, the return is the new model for plotting the final
    curve

    If data is supplied it will calculate the error between the actual
    and known data
    """

    Ee = pars['Ee'].value # Instantaneous modulus
    E1 = pars['E1'].value # Instantaneous modulus
    tau2 = pars['tau2'].value # Time constant

    # Standard Linear Solid Model (SLSM)
    model = 1/Ee*(1 - E1/(E1 + Ee)*np.exp(-t/tau2))

    if data is None:
        return model
    return model - data

def dresidual(pars, t, data=None):
    """
    Derivative of the function to return the jacobian for faster optimization
    Parameters
    ----------
    pars : Ee and E1 and tau2 for Visco terms for the Standard Linear Solid
            Model from Lin2020
    t : time array being passed in
    data : Normalized relaxation data (1/creep compliance) to be passed
            through to compare to model data.  The default is None.
    Returns
    -------
    The jacobian (partial derivatives with respect to unknown variables)
    """

```

```python
    Ee = pars['Ee'].value # Instantaneous modulus
    E1 = pars['E1'].value # Instantaneous modulus
    tau2 = pars['tau2'].value # Time constant

    jac = []

    dCdEe = ((E1*Ee + (E1 + Ee)*(E1 - (E1 + Ee)*np.exp(t/tau2)))*
             np.exp(-t/tau2)/(Ee**2*(E1 + Ee)**2))
    dCdE1 = -np.exp(-t/tau2)/(E1 + Ee)**2
    dCdtau2 = -E1*t*np.exp(-t/tau2)/(Ee*tau2**2*(E1 + Ee))

    jac.append(dCdEe)
    jac.append(dCdE1)
    jac.append(dCdtau2)
    return np.asarray(jac)

def SLSM(jac=None):
    """
    Parameters
    ----------
    Add parameters to be fit using the SLS model

    Returns
    -------
    out : Model output
    t : Model time output
    fit : Model fit output
    """

    # Specify parameters
    fit_params = lf.Parameters() # intialize the class for parameters
    fit_params.add('Ee', value = 1, min=0) # Instantaneous shear modulus
    fit_params.add('E1', value = 1, min=0) # Total change in modulus
    fit_params.add('tau2', value = 1) # Time constant

    # Set up minimization class to be able to pass derivative in (Jacobian)
    minClass = lf.Minimizer(residual, fit_params, fcn_args=(t,),
                            fcn_kws={'data': data})
    if jac is None:
        # No jacobian
        out = minClass.leastsq()
    else:
        # Yes jacobian
        out = minClass.leastsq(Dfun=dresidual, col_deriv=1)
    fit = residual(out.params, t) # run the model to fit the data

    lf.report_fit(out) # modelpars=p_true,  show_correl=True
    print('\n\n\n')
    return out, t, fit

# In[Prony series curve fit]
""" Information for running the curve-fit algorithm """
nu = 0.49
jac = True # Jacobian (if None, then don't include.  If True, do include)

if jac is None:
    print('No Jacobian')
elif jac is True:
```

```
    print('Jacobian')
elif jac is False:
    print('jac needs to be "None" or "True"')
    sys.exit()

p = {} # empty dictionary
tfit = {} # empty dictionary
f = {} # empty dictionary

# run the curve fit
p['p'], tfit['tfit'], f['f'] = SLSM(jac)

# In[Plot data]

color_map = plt.cm.tab10

# Plot Relaxation
plt.plot(t, data, 'o', label=dataName + ' Data', linewidth=2, markersize=5,
         color=color_map.colors[0])

rsqrd = PronyR2(data, f['f'])

plt.plot(tfit['tfit'], f['f'],
         label='LMFIT 2-Term Standard Linear Solid Model' if jac is None else
         'LMFIT 2-Term Standard Linear Solid Model with Jacobian, ' +
         f'$r^2={rsqrd:.5}$', linewidth=2, color=color_map.colors[1])
plt.xlabel(HorizontalUnits, fontsize=18)
plt.ylabel(r'Creep Response ' + CreepUnits, fontsize=18)
plt.legend(loc = 'best', fontsize=14)
plt.grid(True, which='both', alpha=0.5)
plt.savefig("Figures/1LmFitSLSMCreep.pdf" if jac is None else
            "Figures/1LmFitSLSMCreepJac.pdf", bbox_inches='tight')
plt.show()

# Plot Compliance
plt.plot(t, 1/data, '.', label=dataName + ' Data')
plt.plot(tfit['tfit'], 1/f['f'],
         label='LMFIT Standard Linear Solid Model' if jac is None else
         'LMFIT Standard Linear Solid Model with Jacobian')
plt.xlabel(HorizontalUnits, fontsize=18)
plt.ylabel(r'Relaxation Response ' + RelaxUnits, fontsize=18)
plt.legend(loc = 'best', fontsize=14)
plt.grid(True, which='both')
plt.savefig("Figures/2LmFitSLSMRelax.pdf" if jac is None else
            "Figures/2LmFitSLSMRelaxJac.pdf", bbox_inches='tight')
plt.show()

# LogLog (To show different regions (Elastic, Retardation, Viscous))
plt.loglog(t, data, '.', label=dataName + ' Data')
plt.loglog(tfit['tfit'], f['f'],
           label='LMFIT Standard Linear Solid Model' if jac is None else
           'LMFIT Standard Linear Solid Model with Jacobian')
plt.xlabel(HorizontalUnits, fontsize=18)
plt.ylabel(r'Creep Response ' + CreepUnits, fontsize=18)
plt.legend(loc = 'best', fontsize=14)
plt.grid(True, which='both')
plt.savefig("Figures/3LmFitSLSMCreepLogLog.pdf" if jac is None else
            "Figures/3LmFitSLSMCreepJacLogLog.pdf", bbox_inches='tight')
```

```
plt.show()

# Plot Compliance
plt.loglog(t, 1/data, '.', label=dataName + ' Data')
plt.plot(tfit['tfit'], 1/f['f'],
         label='LMFIT Standard Linear Solid Model' if jac is None else
         'LMFIT Standard Linear Solid Model with Jacobian')
plt.xlabel(HorizontalUnits, fontsize=18)
plt.ylabel(r'Relaxation Response ' + RelaxUnits, fontsize=18)
plt.legend(loc = 'best', fontsize=14)
plt.grid(True, which='both')
plt.savefig("Figures/4LmFitSLSMRelaxLogLog.pdf" if jac is None else
            "Figures/4mFitSLSMRelaxJacLogLog.pdf", bbox_inches='tight')
plt.show()

# In[Extract Data]
# Extract data from LMFIT report

Ee = p['p'].params['Ee'].value
E1 = p['p'].params['E1'].value
Tau2 = p['p'].params['tau2'].value

# Write vitreous creep compliance data to a txt file for abaqus importing
file1 = open('Vitreous_SLSM_Constants_LMFIT.txt' if jac is None else
             'Vitreous_SLSM_Constants_LMFIT_Jac.txt' ,"w")
str1 = ("Equation is in the form:  C(t) = 1/Go*(1 - G1/(G1" +
        " + Go)*np.exp(-t/tau2)) # Standard Linear Solid Model (SLSM)")
str2 = ('Standard Linear Solid Model paper Lin2020 Figure 3, ' +
        'equation 2 used in optimization')
str3 = 'Data set = ' + dataName
str4 = '' if jac is None else 'Jacobian was used to converge'
file1.write('\n'.join([str1, str2, str3, str4]) + '\n')

Eo = E1 + Ee # Instantaneous modulus

nu = 0.49
G_to_E_conversion = 2*(1+nu)

# Write to a txt file
file1.write('\n' + 79*'=')
file1.write('\nCalculated long term modulus (Goo) is: ' + str(Ee))
file1.write("\nCalculated modulus (G1) is: " + str(E1))
file1.write("\nCalculated instantaneous modulus (Go) is: " + str(Eo))
file1.write("\nCalculated time constant (Tau2) is: " + str(Tau2))
file1.write('\n' + 79*'.')
file1.write('\n E = G*2*(1+nu), where nu = {}'.format(nu))
file1.write('\nCalculated long term modulus (Eoo) is: ' +
            str(Ee*G_to_E_conversion))
file1.write("\nCalculated modulus (E1) is: " + str(E1*G_to_E_conversion))
file1.write("\nCalculated instantaneous modulus (Eo) is: " +
            str(Eo*G_to_E_conversion))
file1.write("\n")
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.* </>

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 31 17:48:07 2020

@author: Kiffer
"""
import lmfit as lf # lmfit
import numpy as np # numpy
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [16, 9]
import pandas as pd
import os
import sys
import pdb
filePath = os.getcwd() # Location of Python script

dataSets = ['Jami', 'Lee', 'Kashani', 'Polymer', 'Tram']
dataSet = 4
dataSet = dataSets[dataSet]

def PronyR2(y, fit):
    # R squared calculation
    SS_tot = np.sum((y - np.mean(y))**2)
    SS_res = np.sum((y - fit)**2)
    Rsqd = 1 - SS_res/SS_tot
    return Rsqd

# In[Jami Data for example]
if dataSet == 'Jami':
    #############  Jami Creep Shear Test Data Curve Fit######################
    dataName = 'Jami 2014'
    Jami_data = pd.read_csv('Jami/Jami_Shear_Data.txt', sep="\t", header=0)
    Jami_data.columns = ["Time", "NormShearCreep"]
    Jami_data['NormRelaxData'] = 1/Jami_data.NormShearCreep

    # Invert and compute the raw data
    Raw_data_0 = 0.0214671 # first shear point in raw data
    Jami_data['CreepData'] = Jami_data.NormShearCreep*Raw_data_0
    Jami_data['RelaxData'] = 1/Jami_data['CreepData']

    # Convert data to array
    t = Jami_data.Time
    # data = Jami_data.NormShearCreep
    data = Jami_data.NormRelaxData # Normalized relaxation data
    # data = Jami_data.CreepData
    # data = Jami_data.RelaxData

    # Units for plotting
    CreepUnits = r'$\left(\mathrm{Pa}\right)$'
    RelaxUnits = r'$\left(\frac{1}{\mathrm{Pa}}\right)$'
    HorizontalUnits = r'Time (s)'

# In[Lee Digitized Data]
```

```
if dataSet == 'Lee':
    # Read data from Lee1992 Viscoelastic material properties
    # Digitized from http://getdata-graph-digitizer.com/
    dataName = 'Lee 1992'
    Lee_data = pd.read_csv('Lee1992_DigitizedData.txt', sep="\t", header=3)
    Lee_data.columns = ["Time", "CreepDyne"]
    # Conversion is dyne/cm^2 to Pa is multiply by 0.1
    Lee_data["CreepPa"] = Lee_data.CreepDyne*0.1
    Lee_data["RelaxPa"] = 1/Lee_data.CreepPa
    Lee_data["CreepNormalized"] = Lee_data.CreepPa/Lee_data.CreepPa[0]
    Lee_data["RelaxNormalized"] = Lee_data.RelaxPa/Lee_data.RelaxPa[0]

    t = Lee_data.Time
    data = Lee_data.RelaxNormalized

    # Units for plotting
    CreepUnits = r'$\left(\mathrm{Pa}\right)$'
    RelaxUnits = r'$\left(\frac{1}{\mathrm{Pa}}\right)$'
    HorizontalUnits = r'Time (s)'

# In[Kashani_2011]
if dataSet == 'Kashani':
    # Porcine eyes
    t = np.linspace(1, 600, 1000)

    def modelEqn(t,J1,J2,T1,T2,eta_m):
        """
        Parameters
        ----------
        J1 : TYPE
            DESCRIPTION.
        J2 : TYPE
            DESCRIPTION.
        T1 : TYPE
            DESCRIPTION.
        T2 : TYPE
            DESCRIPTION.
        eta_m : TYPE
            DESCRIPTION.

        Returns
        -------
        Compliance
        """
        return J1*(1 - np.exp(-t/T1)) + J2*(1 - np.exp(-t/T2)) + t/eta_m

    J1,J2,T1,T2,eta_m = 1.36, 2.64, 1.77, 1.36, 1332.0
    J_0 = modelEqn(1,J1,J2,T1,T2,eta_m) # Initial value
    data = 1/(modelEqn(t,J1,J2,T1,T2,eta_m)/J_0) # invert to define relaxation
    # I'm not sure how this equation works

    # Read data from Lee1992 Viscoelastic material properties
    # Digitized from http://getdata-graph-digitizer.com/
    dataName = 'Kashani 2011'
    Kashani_data = pd.read_csv('KashaniCreepData.csv', sep=",", header=1)
    Kashani_data.columns = ["Time1", "Creep1",
                            "Time2", "Creep2",
```

```
                                "Time3", "Creep3"]
    # t = Kashani_data.Time1.dropna() # get rid of NaN
    # Compliance = Kashani_data.Creep1.dropna() # Pa # get rid of NaN
    t = Kashani_data.Time2.dropna() # get rid of NaN
    Creep = Kashani_data.Creep2.dropna() # Pa # get rid of NaN
    # t = Kashani_data.Time3.dropna() # get rid of NaN
    # Compliance = Kashani_data.Creep3.dropna() # Pa # get rid of NaN

    CreepNorm = Creep/Creep[0]
    NormRelaxation = 1/CreepNorm
    data = NormRelaxation

    # Units for plotting
    CreepUnits = r'$\left(\mathrm{Pa}\right)$'
    RelaxUnits = r'$\left(\frac{1}{\mathrm{Pa}}\right)$'
    HorizontalUnits = r'Time (s)'

# In[polymer data]

if dataSet == 'Polymer':
    ###############  Creep Shear Test Data Curve Fit########################
    dataName = 'Polymer 2019'
    subFolder = 'Polymer'
    # df = pd.read_csv(os.path.join(filePath, subFolder,
    #                                'Dogbone1_Test1.csv'),
    #                  sep=",", header=5)
    df = pd.read_csv(os.path.join(filePath, subFolder,
                                'Dogbone_1_StressRelaxation_2.csv'),
                     sep=",", header=5)
    # df = pd.read_csv(os.path.join(filePath, subFolder,
    #                                'Dogbone1_StressRelaxtion_3.csv'),
    #                  sep=",", header=5)
    df.columns = ["Time", "Extension","Load"]

    # Convert dataframe to array
    time = np.asarray(df['Time'].tolist())
    extension = np.asarray(df['Extension'].tolist())
    load = np.asarray(df['Load'].tolist())

    # Specimen properties
    width = 12/1000.0
    thickness = 5.3/1000.0

    # calculations
    stress = load/(width*thickness)
    strain = extension/100

    # Determine where the stress begins to decrease from the max point
    # in the array
    StressRelax=stress[np.argmax(stress):-1];
    TimeRelax=time[np.argmax(stress):-1]-time[np.argmax(stress)]

    data = StressRelax#/StressRelax[0] # Normalized stress relaxation
    t = TimeRelax

    # Units for plotting
    CreepUnits = r'$\left(\mathrm{Pa}\right)$'
    RelaxUnits = r'$\left(\frac{1}{\mathrm{Pa}}\right)$'
```

```python
    HorizontalUnits = r'Displacement (x)'

# In[Tram]

if dataSet == 'Tram':
    dataName = 'Tram'
    ExcelPath = 'Tram_2018_Creep_Data.xlsx'

    df = pd.read_excel(os.path.join(filePath, ExcelPath), sheet_name=None)

    path = 'Tram'

    # Folder for general figures to be stored
    TramFigures = os.path.join(path, 'Figures')
    if not os.path.exists(TramFigures):
        os.makedirs(TramFigures)

    for i,j in enumerate(df.keys()):
        if j == 'HU 0764 OS 1 Pa': # 9
        # if j == 'HU2018-0074 OD 1 Pa': # 5
        # if j =='HU2018-0125 OS 1 Pa': # 4
        # if j == 'HU2018-0125 OD 1 Pa': # 3

            # Specific file name (adds the iteration number for organization)
            specificName = str(i) + '_' + j

            # Make directory for each data trace and associated images
            specificPath = os.path.join(path, '{}_'.format(i) + j)
            if not os.path.exists(specificPath):
                os.makedirs(specificPath)

            sheeti = df[j] #.dropna() # Eliminate rows with NA

            time = sheeti.iloc[1:-1,0].reset_index(drop=True)
            creep = sheeti.iloc[1:-1,1].reset_index(drop=True)

            time_constantStress = 6
            # Shift values past region of ramp stress
            creep = creep[time >= time_constantStress].reset_index(drop=True)
            # Shift values past region of ramp stress
            time = time[time >= time_constantStress].reset_index(drop=True)

            # Convert pandas series to numpy arrays
            timeArray = time.to_numpy(dtype='float')
            creepArray = creep.to_numpy(dtype='float')

            # Get rid of nan values from the data trace
            timeArrayRemoveNans = timeArray[np.logical_not(np.isnan(creepArray))]
            creepArrayRemoveNans = creepArray[np.logical_not(np.isnan(creepArray))]

            # Start time at 0
            timeArrayRemoveNans = timeArrayRemoveNans - timeArrayRemoveNans[0]

    t = timeArrayRemoveNans
    creepData = creepArrayRemoveNans
    creepNorm = creepData/creepData[0]
    data = 1/creepNorm

```

```
# In[Lmfit]

def residual(pars, t, data=None):
    """
    Parameters
    ----------
    pars : g_k and Tau_k for Prony N'th order terms
            The final parameter is the sum of the terms that needs to be less
            than 1 for realistic thermodynamic properties
    t : time array being passed in
    data : Normalized relaxation data (1/creep compliance) to be passed
            through to compare to model data.  The default is None.

    Returns
    -------
    If no data is supplied, the return is the new model for plotting the final
    curve

    If data is supplied it will calculate the error between the actual
    and known data
    """

    # Extract g_k and tau_k from the pars class variable
    g_k = []
    tau_k = []
    for key, value in pars.items():
        if key.find('g') >= 0:
            g_k.append(value.value)
        elif key.find('T') >= 0:
            tau_k.append(value.value)

    if NormalizedData is True:
        model = 1 # Normalized so this begins at 1
    else:
        G0 = pars['G0'].value # Instantaneous modulus
        model = G0
    for i in range(len(g_k)):
        model -= g_k[i]*(1 - np.exp(-t/tau_k[i])) # Loop over prony terms

    if data is None:
        return model
    return model - data

def dresidual(pars, t, data=None):
    """
    Derivative of the function to return the jacobian for faster optimization
    Parameters
    ----------
    pars : g_k and Tau_k for Prony N'th order terms
            The final parameter is the sum of the terms that needs to be less
            than 1 for realistic thermodynamic properties
    t : time array being passed in
    data : Normalized relaxation data (1/creep compliance) to be passed
            through to compare to model data.  The default is None.
    Returns
    -------
    The jacobian (partial derivatives with respect to unknown variables)
    """
```

```
    # Extract g_k and tau_k from the pars class variable
    g_k = []
    tau_k = []
    for key, value in pars.items():
        if key.find('g') >= 0:
            g_k.append(value.value)
        elif key.find('T') >= 0:
            tau_k.append(value.value)
    jac = []
    if NormalizedData is not True:
        jac.append(np.ones(len(t))) # derivative of G(t) with respect to G0
    for i in range(len(g_k)):
        jac.append(-1 + np.exp(-t/tau_k[i]))
        jac.append(g_k[i]*t*np.exp(-t/tau_k[i])/tau_k[i]**2)
    return np.asarray(jac)


def PronyN(N, jac=None):
    """
    Parameters
    ----------
    N : Number of parameters in the Prony Series fit

    sumG ensures that the values for the individual springs divided by the G0
    value sum to a value less than 1.

    Returns
    -------
    out : Model output
    t_fit : Model time output
    fit : Model fit output
    """
    # Specify paramters bounds with a for loop for N terms
    fit_params = lf.Parameters() # intialize the class for parameters
    if NormalizedData is True:
        # Used when normalized data
        fit_params.add('G0', value = 1, vary=False)
    else:
        # Instantaneous shear modulus
        fit_params.add('G0', value = 1, min=0, max=G_0_UpperLimit)
    sumG = ''
    for i in range(N):
        if i == N-1:
            # append g_k values for the constraint eqn
            sumG = sumG + 'g_{}/G0'.format(i + 1)
        else:
            # append g_k values for the constraint eqn
            sumG = sumG + 'g_{}/G0 + '.format(i + 1)

        # If Normalized data the bounds of the values are [0,1],
        # otherwise [0,infinity]
        if NormalizedData is True:
            # Used when normalized
            fit_params.add('g_{}'.format(i + 1), value=0.1/N,
                           min=0.0, max=1.0)
            fit_params.add('Tau_{}'.format(i + 1), value=1, min=0.0)
        else:
            # Used when not normalized (1/N)
```

```
            fit_params.add('g_{}'.format(i + 1), value=1, min=0.0)
            # Polymer 0.00001*G_0_UpperLimit
            fit_params.add('Tau_{}'.format(i + 1), value=1, min=0.0)

    # comment this out if you want to relax the requirement for the
    # sum of coefficients
    if Constraint is True:
        # Constraint eqn
        fit_params.add('sumG', min=0, max=1, expr=sumG, vary=True)

    # Set up minimization class to be able to pass derivative in (Jacobian)
    minClass = lf.Minimizer(residual, fit_params, fcn_args=(t,),
                            fcn_kws={'data': data})
    if jac is None:
        # No jacobian
        out = minClass.leastsq()
    else:
        # Yes jacobian
        out = minClass.leastsq(Dfun=dresidual, col_deriv=1)
    # t_fit = np.linspace(0,max(t), 1000)
    fit = residual(out.params, t) # t_fit

    lf.report_fit(out) # modelpars=p_true,  show_correl=True
    print('\n\n\n')
    return out, t, fit # t_fit

# In[Prony series curve fit]
""" Information for running the curve-fit algorithm """
pronyTerms = [1,2,3,4] # number of prony series terms to be plotted [List]
# Upper limit on the instantaneous shear modulus
G_0_UpperLimit = 1000 # 1000000
NormalizedData = True # Normalized data
# enforce the constraint where the sum of G_k's can't be more than 1
Constraint = True
nu = 0.49
jac = True # Jacobian (if None, then don't include.  If True, do include)
if jac is None:
    print('No Jacobian')
elif jac is True:
    print('Jacobian')
elif jac is False:
    print('jac needs to be "None" or "True"')
    sys.exit()

print('Upper limit for instantaneous shear modulus is', G_0_UpperLimit)

p = {} # empty dictionary
tfit = {} # empty dictionary
f = {} # empty dictionary

# Loop over the number of prony terms to calculate the curve fit paramters
for i in pronyTerms:
    A, B, C = PronyN(i, jac)
    p['p{}'.format(i)] = A
    tfit['tfit{}'.format(i)] = B
    f['f{}'.format(i)] = C

# In[Plot data]
```

```python
E_0 = 1#1840
v_0 = 0.49
G_0 = E_0/(2*(1+v_0))
G_0 = 49.3075445
def Prony2(t,a,b,c,d):
    return G_0*(1 - a*(1 - np.exp(-t/b)) - c*(1 - np.exp(-t/d)))
ABQtime = np.linspace(0, max(t), 300)
ABQfit_Norm = Prony2(0, 0.70134, 2.96389e-2, 0.19334, 0.47088)
ABQfit = Prony2(ABQtime, 0.70134, 2.96389e-2, 0.19334, 0.47088)

# Plot Relaxation
# plt.plot(ABQtime, ABQfit, 'b-', label='ABAQUS')
plt.plot(t, data, '.', label=dataName + ' Data')
for i in pronyTerms:
    plt.plot(tfit['tfit{}'.format(i)], f['f{}'.format(i)],
             label='LMFIT {} Prony terms'.format(i) if jac is None else
             'LMFIT {} Prony terms with Jacobian'.format(i))
# plt.ylim(0.01,0.02)
plt.xlabel('Time (s)',fontsize=18)
plt.ylabel(r'Relaxation Modulus',fontsize=18) # Normalized
plt.title('Viscoelastic Response',fontsize=20)
plt.legend(loc = 'best', fontsize=14)
plt.grid(True, which='both')
plt.savefig("Figures/1LmFitRelax.pdf" if jac is None else
            "Figures/1LmFitRelaxJac.pdf", bbox_inches='tight')
plt.show()


# Plot Compliance
# plt.plot(ABQtime, 1/ABQfit, 'b-', label='ABAQUS')

color_map = plt.cm.tab10

plt.plot(t, 1/data, ':o', label=dataName + ' Data', linewidth=2,
         markersize=5, color=color_map.colors[0])

# Plot only the 4 term fit instead of all fits
for i in pronyTerms[-1:]:
    rsqrd_i = PronyR2(1/data, 1/f[f'f{i}'])
    plt.plot(tfit[f'tfit{i}'], 1/f[f'f{i}'],
             label=f'LMFIT {i} Prony terms' if jac is None else
             f'LMFIT {i} Prony terms with Jacobian, $r^2={rsqrd_i:.5}$',
             linewidth=2, color=color_map.colors[1])

# plt.ylim(50,100)
plt.xlabel('Time (s)',fontsize=18)
plt.ylabel(r'Normalized Creep Compliance',fontsize=18)
# plt.title('Viscoelastic Response',fontsize=20)
plt.legend(loc = 'best', fontsize=14)
plt.grid(True, which='both', alpha=0.5)
plt.savefig("Figures/2LmFitCompliance.pdf" if jac is None else
            "Figures/2LmFitComplianceJac.pdf", bbox_inches='tight')
plt.show()

# LogLog (To show different regions (Elastic, Retardation, Viscous))
plt.loglog(t, data, '.', label=dataName + ' Data')
for i in pronyTerms:
    plt.loglog(tfit['tfit{}'.format(i)], f['f{}'.format(i)],
```

```python
                label='LMFIT {} Prony terms'.format(i) if jac is None else
                'LMFIT {} Prony terms with Jacobian'.format(i))
plt.xlabel('Time (s)',fontsize=18)
plt.ylabel(r'Relaxation Modulus',fontsize=18)
plt.title('Viscoelastic Response',fontsize=20)
plt.legend(loc = 'best', fontsize=14)
plt.grid(True, which='both')
plt.savefig("Figures/3LmFitRelaxLogLog.pdf" if jac is None else
            "Figures/3LmFitRelaxJacLogLog.pdf", bbox_inches='tight')
plt.show()

# Plot Compliance
plt.loglog(t, 1/data, '.', label=dataName + ' Data')
for i in pronyTerms:
    plt.loglog(tfit['tfit{}'.format(i)], 1/f['f{}'.format(i)],
                label='LMFIT {} Prony terms'.format(i) if jac is None else
                'LMFIT {} Prony terms with Jacobian'.format(i))
plt.xlabel('Time (s)',fontsize=18)
plt.ylabel(r'Creep Compliance',fontsize=18)
plt.title('Viscoelastic Response',fontsize=20)
plt.legend(loc = 'best', fontsize=14)
plt.grid(True, which='both')
plt.savefig("Figures/4LmFitComplianceLogLog.pdf" if jac is None else
            "Figures/4mFitComplianceJacLogLog.pdf", bbox_inches='tight')
plt.show()

# In[Normalized Plots]
# if NormalizedData == True:
#     # Plot Relaxation
#     data0 = data[0]
#     plt.plot(t, data/data0, '.', label=dataName + ' Data')
#     for i in pronyTerms:
#         ti = tfit['tfit{}'.format(i)] # Time
#         fi = f['f{}'.format(i)] # Curve fit data
#         fi0 = fi[0] # Normalization by the first data point data0#
        # plt.plot(ti, fi/fi0,
        #          label='LMFIT {} Prony terms'.format(i) if jac is None else
        #           'LMFIT {} Prony terms with Jacobian'.format(i))
#     # plt.ylim(0.01,0.02)
#     plt.xlabel('Time (s)',fontsize=18)
#     plt.ylabel(r'Normalized Relaxation Modulus',fontsize=18) # Normalized
#     plt.title('Viscoelastic Response',fontsize=20)
#     plt.legend(loc = 'best', fontsize=14)
#     plt.grid(True, which='both')
    # plt.savefig("Figures/5NormLmFitRelax.pdf" if jac is None else
    #             "Figures/5NormLmFitRelaxJac.pdf", bbox_inches='tight')
#     plt.show()

#     # Plot Compliance
#     plt.plot(t, 1/(data/data0), '.', label=dataName + ' Data')
#     for i in pronyTerms:
#         ti = tfit['tfit{}'.format(i)] # Time
#         fi = f['f{}'.format(i)] # Curve fit data
#         fi0 = fi[0] # Normalization by the first data point data0#
        # plt.plot(ti, 1/(fi/fi0),
        #          label='LMFIT {} Prony terms'.format(i) if jac is None else
        #          'LMFIT {} Prony terms with Jacobian'.format(i))
#     # plt.ylim(50,100)
```

```
#     plt.xlabel('Time (s)',fontsize=18)
#     plt.ylabel(r'Normalized Creep Compliance',fontsize=18)
#     plt.title('Viscoelastic Response',fontsize=20)
#     plt.legend(loc = 'best', fontsize=14)
#     plt.grid(True, which='both')
    # plt.savefig("Figures/6NormLmFitCompliance.pdf" if jac is None else
    #             "Figures/6NormLmFitComplianceJac.pdf", bbox_inches='tight')
#     plt.show()

#     # LogLog (To show different regions (Elastic, Retardation, Viscous))
#     plt.loglog(t, data/data0, '.', label=dataName + ' Data')
#     for i in pronyTerms:
#         ti = tfit['tfit{}'.format(i)] # Time
#         fi = f['f{}'.format(i)] # Curve fit data
#         fi0 = fi[0] # Normalization by the first data point data0#
        # plt.loglog(ti, fi,
        #            label='LMFIT {} Prony terms'.format(i) if jac is None else
        #            LMFIT {} Prony terms with Jacobian'.format(i))
#     plt.xlabel('Time (s)',fontsize=18)
#     plt.ylabel(r'Normalized Relaxation Modulus',fontsize=18)
#     plt.title('Viscoelastic Response',fontsize=20)
#     plt.legend(loc = 'best', fontsize=14)
#     plt.grid(True, which='both')
    # plt.savefig("Figures/7NormLmFitRelaxLogLog.pdf" if jac is None else
    #             "Figures/7NormLmFitRelaxJacLogLog.pdf",
    #             bbox_inches='tight')
#     plt.show()

#     # Plot Compliance
#     plt.loglog(t, 1/(data/data0), '.', label=dataName + ' Data')
#     for i in pronyTerms:
#         ti = tfit['tfit{}'.format(i)] # Time
#         fi = f['f{}'.format(i)] # Curve fit data
#         fi0 = fi[0] # Normalization by the first data point data0#
        # plt.loglog(ti, 1/(fi/fi0),
        #            label='LMFIT {} Prony terms'.format(i) if jac is None else
        #            'LMFIT {} Prony terms with Jacobian'.format(i))
#     plt.xlabel('Time (s)',fontsize=18)
#     plt.ylabel(r'Normalized Creep Compliance',fontsize=18)
#     plt.title('Viscoelastic Response',fontsize=20)
#     plt.legend(loc = 'best', fontsize=14)
#     plt.grid(True, which='both')
    # plt.savefig("Figures/8NormLmFitComplianceLogLog.pdf" if jac is None else
    #             "Figures/8NormLmFitComplianceJacLogLog.pdf",
    #             bbox_inches='tight')
#     plt.show()


# In[Extract Data]
# Extract data from LMFIT report
optParams = {} # empty dictionary
# Loop over pronyTerms to extract the g_k and tau_k values for each N'th
# order fit
for i in pronyTerms:
    g_k = [] # Shear modulus per Prony element
    tau_k = [] # Time constant
    G0_k = [] # Instantaneous shear modulus
    for key, value in p['p{}'.format(i)].params.items():
```

```
            if key.find('g') >= 0:
                g_k.append(value.value)
            elif key.find('T') >= 0:
                tau_k.append(value.value)
            elif key.find('G0') >= 0:
                G0_k.append(value.value)
        optParams['P{}'.format(i)] = g_k
        optParams['T{}'.format(i)] = tau_k
        optParams['G0{}'.format(i)] = G0_k

# Write vitreous creep compliance data to a txt file for abaqus importing
file1 = open('Vitreous_Prony_Constants_LMFIT.txt' if jac is None else
             'Vitreous_Prony_Constants_LMFIT_Jac.txt' ,"w")
str1 = ("Equation is in the form:  G(t) = G_0*(1 - " +
        "SUM_i^N(g_k^P*(1 - exp(-t/tau_k))))")
str2 = '\t'.join(["Prony_#", "g_k^P", "k_i", "tau_k"])
str3 = 'Data set = ' + dataName
str4 = '' if jac is None else 'Jacobian was used to converge'
str5 = '' if Constraint is False else ("enforce the constraint where the " +
                                       "sum of G_k/G_0's can't be more than 1")
str6 = '' if NormalizedData is True else ("Upper limit for G_0 is" +
                                          " {}".format(G_0_UpperLimit))
file1.write('\n'.join([str1, str2 ,str3, str4, str5, str6]) + '\n')

# Loop over pronyTerm results to write to a txt file
for i in pronyTerms:
    g_k = optParams['P{}'.format(i)] # shear modulus of prony term
    tau_k = optParams['T{}'.format(i)] # Time constant
    G_0 = optParams['G0{}'.format(i)][0] # Instantaneous shear modulus
    file1.write('\n' + 79*'=')
    file1.write('\n' + 22*' ' + 'Prony series order ' + str(len(g_k)))
    file1.write('\nCalculated instantaneous shear modulus (G_0) is: ' +
                str(G_0))
    file1.write('\nKnown nu = {}'.format(nu))
    file1.write("\nCalculated instantaneous Young's modulus (E_0) is: " +
                str(G_0*2*(1 + nu)) + '\n')
    file1.write('\nNormalized coefficients\n')
    for m in range(len(g_k)):
        file1.write(''.join(['(' + str(g_k[m]),', 0.0, ',
                             str(tau_k[m]) + '),']) + '\n')
    print(sum(g_k),"g_k")
    file1.write("\n")
    # file1.write('Normalized coefficients (g_k/G_0)\n')
    # for m in range(len(g_k)):
        # file1.write(''.join(['(' + str(g_k[m]/G_0),', 0.0, ',
        #                      str(tau_k[m]) + '),']) + '\n')
    # print(sum(g_k),"g_k")
    # file1.write("\n")
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.*

```
# -*- coding: utf-8 -*-
"""
Created on Mon Aug 24 15:45:38 2020

@author: Kiffer2
"""

import pandas as pd
import sqlite3
import glob
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from matplotlib.pyplot import cm
import matplotlib.patheffects as pe
plt.rcParams['figure.figsize'] = [16, 9]
import pdb


def Least_Squares(x,y):
    """
    Calculate the slope and y-intercept using matrix math
    x & y are the coordinates of points

    parameters (X,Y) Data

    Returns:
        Curve fit data and parameters m*x + b
    """
    Z = np.ones((len(x),2))
    Z[:,1] = x
    # Calculate the matrix inverse for the constants of the regression
    A = np.dot(np.linalg.inv(np.dot(Z.T,Z)),(np.dot(Z.T,y)))
    linFit = x*A[1] + A[0]

    # Stats
    SS_tot = np.sum((y - np.mean(y))**2)
    SS_res = np.sum((y - linFit)**2)
    Rsqd = 1 - SS_res/SS_tot

    return linFit, A, Rsqd

os.chdir('F:/Abaqus Working Directory/PeterComp/HumanData')

filePath = os.getcwd()

ExcelName = 'HumanData.xlsx'

ExcelPath = os.path.join(filePath, ExcelName)

conn = sqlite3.connect('HumanData.db')
c = conn.cursor()

c.execute('DROP TABLE IF EXISTS HumanData')
```

```
c.execute('''CREATE TABLE 'HumanData'(
    'HumanID' TEXT,
    'HumanAGE' REAL,
    'HumanGender' TEXT,
    'HumanLeftRight' TEXT,
    'HumanRegion' TEXT,
    'PostMortemHrs_Min' REAL,
    'DateOfDeath' timestamp,
    'TimeOfDeath' timestamp,
    'EnucleationDate' timestamp,
    'EnucleationTime' timestamp,
    'DateOfTesting' timestamp,
    'TimeOfTesting' timestamp,
    'DiameterPostAnt' REAl,
    'DiameterNasTemp' REAL,
    'SSi' REAL,
    'SSf' REAL,
    'TFMax' REAL,
    'DispMax' REAL,
    'FMax' REAL,
    'FSS' REAL,
    'Slope10' REAL,
    'Rsqrd10' REAL,
    'Slope20' REAL,
    'Rsqrd20' REAL,
    'Slope30' REAL,
    'Rsqrd30' REAL,
    'Slope0' REAL,
    'Rsqrd0' REAL,
    'PeelVideoName' TEXT,
    'PeelVideoHyperlink' TEXT,
    'VideoComments' TEXT,
    'LightMicroscopyImages' TEXT)
''')

df = pd.read_excel(ExcelPath, sheet_name=None)

""" Put values into a dictionary of dataframes """
HumanID = {}
HumanAge = {}
HumanGender = {}
HumanLeftRight = {}
HumanRegion = {}
PostMortemHrs_Min = {}
DateOfDeath = {}
TimeOfDeath = {}
EnucleationDate = {}
EnucleationTime = {}
DateOfTesting = {}
TimeOfTesting = {}
DiameterPostAnt = {}
DiameterNasTemp = {}
SSi = {}
SSf = {}
FMax = {}
FSS = {}
TFMax = {}
DispMax = {}
```

```
Slope10 = {}
Rsqrd10 = {}
Slope20 = {}
Rsqrd20 = {}
Slope30 = {}
Rsqrd30 = {}
Slope0 = {}
Rsqrd0 = {}
PeelVideoName = {}
PeelVideoHyperlink = {}
VideoComments = {}
LightMicroscopyImages = {}
time = {}
extension = {}
force = {}
for i in df.keys():
    if len(i) <= 2: # Only look at data traces ... [Row,Col]
        print(i)
        HumanID['{}'.format(i)] =  df['{}'.format(i)].iloc[0,1]
        HumanAge['{}'.format(i)] =  df['{}'.format(i)].iloc[1,1]
        HumanGender['{}'.format(i)] =  df['{}'.format(i)].iloc[2,1]
        HumanLeftRight['{}'.format(i)] =  df['{}'.format(i)].iloc[3,1]
        HumanRegion['{}'.format(i)] =  df['{}'.format(i)].iloc[4,1]
        PostMortemHrs_Min['{}'.format(i)] =  df['{}'.format(i)].iloc[5,1]
        DateOfDeath['{}'.format(i)] =  df['{}'.format(i)].iloc[6,1]
        TimeOfDeath['{}'.format(i)] =  df['{}'.format(i)].iloc[7,1]
        EnucleationDate['{}'.format(i)] =  df['{}'.format(i)].iloc[8,1]
        EnucleationTime['{}'.format(i)] =  df['{}'.format(i)].iloc[9,1]
        DateOfTesting['{}'.format(i)] =  df['{}'.format(i)].iloc[10,1]
        TimeOfTesting['{}'.format(i)] =  df['{}'.format(i)].iloc[11,1]
        DiameterPostAnt['{}'.format(i)] =  df['{}'.format(i)].iloc[12,1]
        DiameterNasTemp['{}'.format(i)] =  df['{}'.format(i)].iloc[13,1]
        SSi['{}'.format(i)] =  df['{}'.format(i)].iloc[18,6]
        SSf['{}'.format(i)] =  df['{}'.format(i)].iloc[19,6]
        FMax['{}'.format(i)] =  df['{}'.format(i)].iloc[22,6]
        FSS['{}'.format(i)] =  df['{}'.format(i)].iloc[21,6]
        PeelVideoName['{}'.format(i)] =  df['{}'.format(i)].iloc[26,6]
        PeelVideoHyperlink['{}'.format(i)] =  df['{}'.format(i)].iloc[27,6]
        VideoComments['{}'.format(i)] =  df['{}'.format(i)].iloc[28,6]
        LightMicroscopyImages['{}'.format(i)] =  df['{}'.format(i)].iloc[29,6]

        """ Data Traces """
        time['{}'.format(i)] = pd.to_numeric(df['{}'.format(i)].iloc[17:-1,0].reset_index(drop=True))
        extension['{}'.format(i)] = pd.to_numeric(df['{}'.format(i)].iloc[17:-1,1].reset_index(drop=True))
        force['{}'.format(i)] = pd.to_numeric(df['{}'.format(i)].iloc[17:-1,2].reset_index(drop=True))

        if str(FMax['{}'.format(i)]) != 'nan':
            # Time at max force
            TFMaxi = time['{}'.format(i)][force['{}'.format(i)].loc[lambda x: x == FMax['{}'.format(i)]].index.values[0]]
            TFMax['{}'.format(i)] = TFMaxi

            # Slope calculation 10 seconds before max peel force
```

```python
            maxIndex = force['{}'.format(i)].loc[lambda x: x == FMax['{}'.format(i)]].index.values[0] # Location in the array for the max force
            x = extension['{}'.format(i)][maxIndex-100:maxIndex] # Array from MaxIndex-100 (10 sec) to location of max force
            y = force['{}'.format(i)][maxIndex-100:maxIndex] # Array from MaxIndex-100 (10 sec) to location of max force
            curveFit, Params, Rsqrd = Least_Squares(x,y) # Perform least squares and return
            Slope10['{}'.format(i)] = Params[1]
            Rsqrd10['{}'.format(i)] = Rsqrd

            # Slope calculation 20 seconds before max peel force
            x = extension['{}'.format(i)][maxIndex-200:maxIndex] # Array from MaxIndex-200 (20 sec) to location of max force
            y = force['{}'.format(i)][maxIndex-200:maxIndex] # Array from MaxIndex-200 (20 sec) to location of max force
            curveFit, Params, Rsqrd = Least_Squares(x,y) # Perform least squares and return
            Slope20['{}'.format(i)] = Params[1]
            Rsqrd20['{}'.format(i)] = Rsqrd

            # Slope calculation 30 seconds before max peel force
            x = extension['{}'.format(i)][maxIndex-300:maxIndex] # Array from MaxIndex-300 (30 sec) to location of max force
            y = force['{}'.format(i)][maxIndex-300:maxIndex] # Array from MaxIndex-300 (30 sec) to location of max force
            curveFit, Params, Rsqrd = Least_Squares(x,y) # Perform least squares and return
            Slope30['{}'.format(i)] = Params[1]
            Rsqrd30['{}'.format(i)] = Rsqrd

            # Slope calculation from zero to max peel force
            x = extension['{}'.format(i)][0:maxIndex] # Array from 0 to location of max force
            y = force['{}'.format(i)][0:maxIndex] # Array from 0 to location of max force
            curveFit, Params, Rsqrd = Least_Squares(x,y) # Perform least squares and return
            Slope0['{}'.format(i)] = Params[1]
            Rsqrd0['{}'.format(i)] = Rsqrd

            # Displacement at max force
            DispMax['{}'.format(i)] = extension['{}'.format(i)][time['{}'.format(i)] == TFMaxi].values[0]

        else:
            TFMaxi = np.nan
            TFMax['{}'.format(i)] = TFMaxi
            Slope10['{}'.format(i)] = np.nan
            Rsqrd10['{}'.format(i)] = np.nan
            Slope20['{}'.format(i)] = np.nan
            Rsqrd20['{}'.format(i)] = np.nan
            Slope30['{}'.format(i)] = np.nan
            Rsqrd30['{}'.format(i)] = np.nan
            Slope0['{}'.format(i)] = np.nan
            Rsqrd0['{}'.format(i)] = np.nan
            DispMax['{}'.format(i)] = np.nan
```

```

        """ Add data to SQL database """
        HumanIDi = HumanID['{}'.format(i)]
        HumanAgei = HumanAge['{}'.format(i)]
        HumanGenderi = HumanGender['{}'.format(i)]
        HumanLeftRighti = HumanLeftRight['{}'.format(i)]
        HumanRegioni = HumanRegion['{}'.format(i)]
        PostMortemHrs_Mini = PostMortemHrs_Min['{}'.format(i)]
        DateOfDeathi = DateOfDeath['{}'.format(i)]
        TimeOfDeathi = TimeOfDeath['{}'.format(i)]
        EnucleationDatei = EnucleationDate['{}'.format(i)]
        EnucleationTimei = EnucleationTime['{}'.format(i)]
        DateOfTestingi = DateOfTesting['{}'.format(i)]
        TimeOfTestingi = TimeOfTesting['{}'.format(i)]
        DiameterPostAnti = DiameterPostAnt['{}'.format(i)]
        DiameterNasTempi = DiameterNasTemp['{}'.format(i)]
        SSii = SSi['{}'.format(i)]
        SSfi = SSf['{}'.format(i)]
        TFMaxi = TFMax['{}'.format(i)]
        DispMaxi = DispMax['{}'.format(i)]
        FMaxi = FMax['{}'.format(i)]
        FSSi = FSS['{}'.format(i)]
        Slope10i = Slope10['{}'.format(i)]
        Rsqrd10i = Rsqrd10['{}'.format(i)]
        Slope20i = Slope20['{}'.format(i)]
        Rsqrd20i = Rsqrd20['{}'.format(i)]
        Slope30i = Slope30['{}'.format(i)]
        Rsqrd30i = Rsqrd30['{}'.format(i)]
        Slope0i = Slope0['{}'.format(i)]
        Rsqrd0i = Rsqrd0['{}'.format(i)]
        PeelVideoNamei = PeelVideoName['{}'.format(i)]
        PeelVideoHyperlinki = PeelVideoHyperlink['{}'.format(i)]
        VideoCommentsi = VideoComments['{}'.format(i)]
        LightMicroscopyImagesi = LightMicroscopyImages['{}'.format(i)]

        # add to sql database
        # cur.execute(''' INSERT INTO HumanData(HumanID) VALUES (?)''',(HumanIDi))
        # conn.commit()
        c.execute(''' INSERT INTO HumanData(
                        HumanID,
                        HumanAge,
                        HumanGender,
                        HumanLeftRight,
                        HumanRegion,
                        PostMortemHrs_Min,
                        DateOfDeath,
                        TimeOfDeath,
                        EnucleationDate,
                        EnucleationTime,
                        DateOfTesting,
                        TimeOfTesting,
                        DiameterPostAnt,
                        DiameterNasTemp,
                        SSi,
                        SSf,
                        TFMax,
                        DispMax,
                        FMax,
```

```
                        FSS,
                        Slope10,
                        Rsqrd10,
                        Slope20,
                        Rsqrd20,
                        Slope30,
                        Rsqrd30,
                        Slope0,
                        Rsqrd0,
                        PeelVideoName,
                        PeelVideoHyperlink,
                        VideoComments,
                        LightMicroscopyImages
                        ) VALUES
↪  (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
                    ''',(HumanIDi,
                        HumanAgei,
                        HumanGenderi,
                        HumanLeftRighti,
                        HumanRegioni,
                        str(PostMortemHrs_Mini),
                        str(DateOfDeathi),
                        str(TimeOfDeathi),
                        str(EnucleationDatei),
                        str(EnucleationTimei),
                        str(DateOfTestingi),
                        str(TimeOfTestingi),
                        DiameterPostAnti,
                        DiameterNasTempi,
                        SSii,
                        SSfi,
                        TFMaxi,
                        DispMaxi,
                        FMaxi,
                        FSSi,
                        Slope10i,
                        Rsqrd10i,
                        Slope20i,
                        Rsqrd20i,
                        Slope30i,
                        Rsqrd30i,
                        Slope0i,
                        Rsqrd0i,
                        PeelVideoNamei,
                        PeelVideoHyperlinki,
                        VideoCommentsi,
                        LightMicroscopyImagesi))
        conn.commit()


# In[1] Plot individual data traces in the Figures folder as well as the
# age group/region folder

# Make the new folder to put figure
folderPath = os.path.join(filePath, 'Figures')
if not os.path.exists(folderPath):
    os.makedirs(folderPath)

```

```python
""" Age Groups """
g1e = [] # 30 - 39 yrs Equator
g1p = [] # 30 - 39 yrs Posterior
g2e = [] # 40 - 49 yrs Equator
g2p = [] # 40 - 49 yrs Posterior
g3e = [] # 50 - 59 yrs Equator
g3p = [] # 50 - 59 yrs Posterior
g4e = [] # 60 - 69 yrs Equator
g4p = [] # 60 - 69 yrs Posterior
g5e = [] # 70 - 79 yrs Equator
g5p = [] # 70 - 79 yrs Posterior
g6e = [] # 80 - 89 yrs Equator
g6p = [] # 80 - 89 yrs Posterior

# specific for paper 3
g8e = [] # 30 - 59 yrs Equator
g8p = [] # 30 - 59 yrs Posterior
g9e = [] # 60 - 89 yrs Equator
g9p = [] # 60 - 89 yrs Posterior

""" Plot Data """
for i in df.keys():
    if len(i) <= 2: # Only look at data traces ... [Row,Col]
        print(i)

        """ Data Traces """
        time['{}'.format(i)] =
        ↪  pd.to_numeric(df['{}'.format(i)].iloc[17:-1,0].reset_index(drop=True))
        extension['{}'.format(i)] =
        ↪  pd.to_numeric(df['{}'.format(i)].iloc[17:-1,1].reset_index(drop=True))
        force['{}'.format(i)] =
        ↪  pd.to_numeric(df['{}'.format(i)].iloc[17:-1,2].reset_index(drop=True))

        """ call data from pandas dataframe """
        HumanIDi = HumanID['{}'.format(i)]
        HumanAgei = HumanAge['{}'.format(i)]
        HumanGenderi = HumanGender['{}'.format(i)]
        HumanLeftRighti = HumanLeftRight['{}'.format(i)]
        HumanRegioni = HumanRegion['{}'.format(i)]
        PostMortemHrs_Mini = PostMortemHrs_Min['{}'.format(i)]
        DateOfDeathi = DateOfDeath['{}'.format(i)]
        TimeOfDeathi = TimeOfDeath['{}'.format(i)]
        EnucleationDatei = EnucleationDate['{}'.format(i)]
        EnucleationTimei = EnucleationTime['{}'.format(i)]
        DateOfTestingi = DateOfTesting['{}'.format(i)]
        TimeOfTestingi = TimeOfTesting['{}'.format(i)]
        DiameterPostAnti = DiameterPostAnt['{}'.format(i)]
        DiameterNasTempi = DiameterNasTemp['{}'.format(i)]
        SSii = SSi['{}'.format(i)]
        SSfi = SSf['{}'.format(i)]
        TMaxi = TFMax['{}'.format(i)]
        DispMaxi = DispMax['{}'.format(i)]
        FMaxi = FMax['{}'.format(i)]
        FSSi = FSS['{}'.format(i)]
        Slope10i = Slope10['{}'.format(i)]
        Rsqrd10i = Rsqrd10['{}'.format(i)]
        Slope20i = Slope20['{}'.format(i)]
        Rsqrd20i = Rsqrd20['{}'.format(i)]
```

```
        Slope30i = Slope30['{}'.format(i)]
        Rsqrd30i = Rsqrd30['{}'.format(i)]
        Slope0i = Slope0['{}'.format(i)]
        Rsqrd0i = Rsqrd0['{}'.format(i)]
        PeelVideoNamei = PeelVideoName['{}'.format(i)]
        PeelVideoHyperlinki = PeelVideoHyperlink['{}'.format(i)]
        VideoCommentsi = VideoComments['{}'.format(i)]
        LightMicroscopyImagesi = LightMicroscopyImages['{}'.format(i)]

        if HumanAgei >= 30 and HumanAgei < 40:
            if HumanRegioni == 'Equator':
                g1e.append(i)
                folder = '30_39_Equator'
            elif HumanRegioni == 'Posterior':
                g1p.append(i)
                folder = '30_39_Posterior'
        elif HumanAgei >= 40 and HumanAgei < 50:
            if HumanRegioni == 'Equator':
                g2e.append(i)
                folder = '40_49_Equator'
            elif HumanRegioni == 'Posterior':
                g2p.append(i)
                folder = '40_49_Posterior'
        elif HumanAgei >= 50 and HumanAgei < 60:
            if HumanRegioni == 'Equator':
                g3e.append(i)
                folder = '50_59_Equator'
            elif HumanRegioni == 'Posterior':
                g3p.append(i)
                folder = '50_59_Posterior'
        elif HumanAgei >= 60 and HumanAgei < 70:
            if HumanRegioni == 'Equator':
                g4e.append(i)
                folder = '60_69_Equator'
            elif HumanRegioni == 'Posterior':
                g4p.append(i)
                folder = '60_69_Posterior'
        elif HumanAgei >= 70 and HumanAgei < 80:
            if HumanRegioni == 'Equator':
                g5e.append(i)
                folder = '70_79_Equator'
            elif HumanRegioni == 'Posterior':
                g5p.append(i)
                folder = '70_79_Posterior'
        elif HumanAgei >= 80 and HumanAgei < 90:
            if HumanRegioni == 'Equator':
                g6e.append(i)
                folder = '80_89_Equator'
            elif HumanRegioni == 'Posterior':
                g6p.append(i)
                folder = '80_89_Posterior'

        # Category for the age group/region for paper 3
        if HumanAgei >= 30 and HumanAgei < 60:
            if HumanRegioni == 'Equator':
                g8e.append(i)
                folder2 = '30_59_Equator'
            elif HumanRegioni == 'Posterior':
```

```
                g8p.append(i)
                folder2 = '30_59_Posterior'
        elif HumanAgei >= 60 and HumanAgei < 90:
            if HumanRegioni == 'Equator':
                g9e.append(i)
                folder2 = '60_89_Equator'
            elif HumanRegioni == 'Posterior':
                g9p.append(i)
                folder2 = '60_89_Posterior'


        # Make the new folder to put figure
        folderPath = os.path.join(filePath, folder)
        if not os.path.exists(folderPath):
            os.makedirs(folderPath)

        # specific for paper 3
        folderPath2 = os.path.join(filePath, folder2)
        if not os.path.exists(folderPath2):
            os.makedirs(folderPath2)

        """ Plot force vs time and force vs disp """
        # Calculate gradient
        n = i
        gn = np.gradient(extension['{}'.format(n)],time['{}'.format(n)])

        tn = time['{}'.format(n)]
        fn = force['{}'.format(n)]
        dn = extension['{}'.format(n)]

        ssin = SSi['{}'.format(n)]
        ssfn = SSf['{}'.format(n)]

        # Check if steady state values exist, if they do create time/force array
        if str(ssin) == 'nan':
            timeLocSSIn = np.nan
            timeLocSSFn = np.nan

            avgFnVal = np.nan

            sstnArray = np.array([ssin, ssfn])
            ssfnArray = np.array([avgFnVal, avgFnVal])
        else:
            # Location for steady state start/stop in the time array
            timeLocSSIn = tn.loc[lambda x: x == ssin].index.values[0]
            timeLocSSFn = tn.loc[lambda x: x == ssfn].index.values[0]

            # Average force value in between the steady state times
            avgFnVal = np.sum(fn[timeLocSSIn:timeLocSSFn])/len(range(timeLocSSIn,timeLocSSFn))

            sstnArray = np.array([ssin, ssfn]) # Steady-state time array
            ssfnArray = np.array([avgFnVal, avgFnVal]) # Steady-state force array
            ssdnArray = np.array([dn[timeLocSSIn], dn[timeLocSSFn]]) # Steady-state displacement array

        FMaxi = FMax['{}'.format(n)]
        if str(FMaxi) == 'nan':
```

```
            FmaxTimeLoc = np.nan
            tFmax = np.nan
            dFmax = np.nan # location in displacement where F = max

        else:
            FmaxTimeLoc = fn.loc[lambda x: x == FMaxi].index.values[0]
            tFmax = tn[FmaxTimeLoc]
            dFmax = dn[FmaxTimeLoc] # location in displacement where F = max

            # slope calculation for 10 seconds prior to the max peel force
            maxIndex = force['{}'.format(i)].loc[lambda x: x == FMax['{}'.format(i)]].index.values[0] # Location in the array for the max force
            x10 = extension['{}'.format(i)][maxIndex-100:maxIndex] # Array from maxIndex - 100 (10 sec) to location of max force
            y = force['{}'.format(i)][maxIndex-100:maxIndex] # Array from maxIndex - 100 (10 sec) to location of max force
            curveFit10, Params10, Rsqrd10val = Least_Squares(x10,y) # Perform least squares and return

            # slope calculation for 20 seconds prior to the max peel force
            x20 = extension['{}'.format(i)][maxIndex-200:maxIndex] # Array from maxIndex - 200 (20 sec) to location of max force
            y = force['{}'.format(i)][maxIndex-200:maxIndex] # Array from maxIndex - 200 (20 sec) to location of max force
            curveFit20, Params20, Rsqrd20val = Least_Squares(x20,y) # Perform least squares and return

            # slope calculation for 30 seconds prior to the max peel force
            x30 = extension['{}'.format(i)][maxIndex-300:maxIndex] # Array from maxIndex - 300 (30 sec) to location of max force
            y = force['{}'.format(i)][maxIndex-300:maxIndex] # Array from maxIndex - 300 (30 sec) to location of max force
            curveFit30, Params30, Rsqrd30val = Least_Squares(x30,y) # Perform least squares and return

            # Slope calculation from zero to max peel force
            x0 = extension['{}'.format(i)][0:maxIndex] # Array from 0 to location of max force
            y = force['{}'.format(i)][0:maxIndex] # Array from 0 to location of max force
            curveFit0, Params0, Rsqrd0val = Least_Squares(x0,y) # Perform least squares and return

        FSSi = avgFnVal

        """
        Plot force vs time
        """
        fign, axn = plt.subplots()

        # Plot data trace and Max & Steady-State vs time
        axn.plot(tn, fn*1e3,'-', color='k', linewidth=1, markersize=2, label = 'Trace #{}'.format(n))
        if str(FMaxi) == 'nan' and str(ssin) == 'nan':
            pass # used to be 'continue' but an error showed up

        if str(FMaxi) != 'nan':
```

```python
            axn.plot(tFmax, FMaxi*1e3,'.', color='r', linewidth=1, markersize=20, label = 'Max Peel - {:.4f} (mN)'.format(FMaxi*1e3))

        if str(ssin) != 'nan':
            axn.plot(sstnArray, ssfnArray*1e3,'-', color='c', linewidth=3, markersize=2, label = 'Steady State - {:.4f} (mN)'.format(FSSi*1e3))

        axn.axhline(y=0, color='k')
        axn.set_xlabel('Time (sec)',fontsize=18)

        if str(FMaxi) == 'nan':
            axn.set_ylim(-0.5, fn.max()*1e3 + 1)
        else:
            axn.set_ylim(-0.5, FMaxi*1e3 + 1)
        axn.set_ylabel('Force (mN)',fontsize=18)
        axn.set_title(HumanIDi + ', ' + HumanGenderi + ', ' + 'Age: ' + str(HumanAgei) + ', ' + HumanLeftRighti + ', ' + HumanRegioni, fontsize=20)
        axn.grid(True, which='both')
        axn.legend(loc='best', prop={"size":12})
        fign.savefig(os.path.join(filePath,'Figures/' + 'Trace_{}'.format(str(n)) + '_F_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
        fign.savefig(os.path.join(filePath, folder, 'Trace_{}'.format(str(n)) + '_F_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
        fign.savefig(os.path.join(filePath, folder2, 'Trace_{}'.format(str(n)) + '_F_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
        plt.close()

        """
        Plot force vs displacement with slope
        """
        fign, axn = plt.subplots()

        # Plot data trace and Max & Steady-State vs displacement
        axn.plot(dn, fn*1e3,'-', color='k', linewidth=1, markersize=2, label = 'Trace #{}'.format(n), alpha = 0.3)

        if str(FMaxi) == 'nan' and str(ssin) == 'nan':
            pass # used to be 'continue' but an error showed up

        if str(FMaxi) != 'nan':
            axn.plot(dFmax, FMaxi*1e3,'.', color='r', linewidth=1, markersize=20, label = 'Max Peel - {:.4f} (mN)'.format(FMaxi*1e3))
            axn.plot(x0, curveFit0*1e3, '-', color='tab:blue', linewidth=1, label=r'Curve fit 0 (s) y = {:.4f}x + {:.4f} (mN), $r^2$ = {:.4f}'.format(Params0[1]*1e3, Params0[0]*1e3, Rsqrd0val))
            axn.plot(x30, curveFit30*1e3, '-', color='tab:orange', linewidth=2, label=r'Curve fit Max - 30 (s) y = {:.4f}x + {:.4f} (mN), $r^2$ = {:.4f}'.format(Params30[1]*1e3, Params30[0]*1e3, Rsqrd30val))
            axn.plot(x20, curveFit20*1e3, '-', color='tab:purple', linewidth=3, label=r'Curve fit Max - 20 (s) y = {:.4f}x + {:.4f} (mN), $r^2$ = {:.4f}'.format(Params20[1]*1e3, Params20[0]*1e3, Rsqrd20val))
            axn.plot(x10, curveFit10*1e3, '-', color='tab:green', linewidth=4, label=r'Curve fit Max - 10 (s) y = {:.4f}x + {:.4f} (mN), $r^2$ = {:.4f}'.format(Params10[1]*1e3, Params10[0]*1e3, Rsqrd10val))

        if str(ssin) != 'nan':
```

```
            axn.plot(ssdnArray, ssfnArray*1e3,'-', color='c', linewidth=3,
             ↪ markersize=2, label = 'Steady State - {:.4f} (mN)'.format(FSSi*1e3))

        axn.axhline(y=0, color='k')
        axn.set_xlabel('Disp (mm)',fontsize=18)

        if str(FMaxi) == 'nan':
            axn.set_ylim(-0.5, fn.max()*1e3 + 1)
        else:
            axn.set_ylim(-0.5, FMaxi*1e3 + 1)

        axn.set_ylabel('Force (mN)',fontsize=18)
        axn.set_title(HumanIDi + ', ' + HumanGenderi + ', ' + 'Age: ' +
         ↪ str(HumanAgei) + ', ' + HumanLeftRighti + ', ' + HumanRegioni,
         ↪ fontsize=20)
        axn.grid(True, which='both')
        axn.legend(loc='best', prop={"size":12})
        fign.savefig(os.path.join(filePath,'Figures/' + 'Trace_{}'.format(str(n)) +
         ↪ '_F_vs_disp.png'), dpi=300, bbox_inches='tight') # Save figure
        fign.savefig(os.path.join(filePath, folder, 'Trace_{}'.format(str(n)) +
         ↪ '_F_vs_disp.png'), dpi=300, bbox_inches='tight') # Save figure
        fign.savefig(os.path.join(filePath, folder2, 'Trace_{}'.format(str(n)) +
         ↪ '_F_vs_disp.png'), dpi=300, bbox_inches='tight') # Save figure
        plt.close()

        # Write the txt file with the force, extension, time data to the folder
        """ Print the Instron Data """
        print("\nWriting out the Instron data...")
        filename = os.path.join(filePath, folder,
         ↪ 'Trace_{}_Instron_Data'.format(str(n)) + '.txt')
        filename2 = os.path.join(filePath, folder2,
         ↪ 'Trace_{}_Instron_Data'.format(str(n)) + '.txt')
        outfile = open(filename,'w')
        outfile2 = open(filename2,'w')
        DataFile = ['Human ID:\t{}'.format(HumanIDi),
                    'Human Age:\t{}'.format(HumanAgei),
                    'Human Gender:\t{}'.format(HumanGenderi),
                    'Human Left/Right:\t{}'.format(HumanLeftRighti),
                    'Human Region:\t{}'.format(HumanRegioni),
                    'Post Mortem Hrs_Min:\t{}'.format(PostMortemHrs_Mini),
                    'Date of Death:\t{}'.format(DateOfDeathi),
                    'Time of Death:\t{}'.format(TimeOfDeathi),
                    'Enucleation Date:\t{}'.format(DateOfTestingi),
                    'Enucleation Time:\t{}'.format(TimeOfTestingi),
                    'Diameter Posterior Anterior (in):\t{}'.format(DiameterPostAnti),
                    'Diameter Nasal Temporal (in):\t{}'.format(DiameterNasTempi),
                    'SSi (s):\t{}'.format(SSii),
                    'SSf (s):\t{}'.format(SSfi),
                    'Time Max (s):\t{}'.format(TMaxi),
                    'Disp Max (mm):\t{}'.format(DispMaxi),
                    'FMax (mN):\t{}'.format(FMaxi*1e3),
                    'FSS (mN):\t{}'.format(FSSi*1e3),
                    'Slope 10 seconds before max to max force value
                     ↪ (mN/m):\t{}'.format(Slope10i*1e3),
                    'R^2 for linear regression 10 seconds before
                     ↪ max:\t{}'.format(Rsqrd10i),
                    'Slope 20 seconds before max to max force value
                     ↪ (mN/m):\t{}'.format(Slope20i*1e3),
```

```
                    'R^2 for linear regression 20 seconds before max:\t{}'.format(Rsqrd20i),
                    'Slope 30 seconds before max to max force value (mN/m):\t{}'.format(Slope30i*1e3),
                    'R^2 for linear regression 30 seconds before max:\t{}'.format(Rsqrd30i),
                    'Slope from 0 to max force value (mN/m):\t{}'.format(Slope0i*1e3),
                    'R^2 for linear regression from 0 to max:\t{}'.format(Rsqrd0i),
                    'Peel Video Name:\t{}'.format(PeelVideoNamei),
                    'Peel Video Hyperlink:\t{}'.format(PeelVideoHyperlinki),
                    'Video Comments:\t{}'.format(VideoCommentsi),
                    'Light Microscopy Images:\t{}'.format(LightMicroscopyImagesi),
                    '\n',
                    'Time (s)\tExtension (mm)\tForce (N)']

        HeaderWrite = '\n'.join(item for item in DataFile)
        outfile.write(HeaderWrite)
        outfile2.write(HeaderWrite)
        for i,j in enumerate(tn):
            line = '\n%f\t%f\t%f' % (j, dn[i], fn[i])
            outfile.write(line)
            outfile2.write(line)
        outfile.close()
        outfile2.close()
        print("\nDone!")
        print("\nThe output file will be named '{}".format(filename) + "'")

""" Plot the extension rate for last test """
fig2, ax2 = plt.subplots()
ax2.plot(tn, gn,'-', color='k', linewidth=1, markersize=2, label = '1')
ax2.set_xlim(0,2.5)
ax2.set_xlabel('Time (sec)',fontsize=18)
ax2.set_ylim(0,0.045)
ax2.set_ylabel('Velocity (mm/s)',fontsize=18)
ax2.set_title('Data Trace', fontsize=20)
ax2.grid(True, which='both')
lines = fig2.gca().get_lines()
show = [0]
legend1 = ax2.legend([lines[i] for i in show],[lines[i].get_label() for i in show], prop={"size":12}, loc='best', title = 'Trace')
ax2.add_artist(legend1)
plt.show()
plt.close()


# In[2]

""" Plot each age group data on top of one another """
os.chdir(filePath)
fileNames = next(os.walk('.'))[1]
print(fileNames)

for i in fileNames:
    if i == 'Figures' or i == 'StatisticsFigures':
        # skip these two folders
        continue

```

```
    elif i != '30_59_Equator' and i != '30_59_Posterior' and i != '60_89_Equator' and i != '60_89_Posterior':
        print(i, 'Age decade')
        subPath = os.path.join(filePath, i)
        (folderName, directory) = os.path.split(subPath)
        os.chdir(subPath)
        subTxtFiles = [x for x in glob.glob('*.txt')] # Posterior/Equator
        show = []
        maxVals = []

        color1 = iter(cm.Set1(np.linspace(0,1,len(subTxtFiles))))
        color2 = iter(cm.Set1(np.linspace(0,1,len(subTxtFiles))))
        color3 = iter(cm.Set1(np.linspace(0,1,len(subTxtFiles))))
        fign, axn = plt.subplots()
        for j,k in enumerate(subTxtFiles):
            c1 = next(color1)
            c2 = next(color2)
            c3 = next(color3)
            """ Read in the csv file """
            dfValsn = pd.read_csv(os.path.join(subPath, k), sep="\t", nrows=29, header=None, names=['Var','Attribute'])

            """ File Attributes """
            HID = dfValsn['Attribute'][0] # ID
            HAGE = dfValsn['Attribute'][1] # Age
            HG = dfValsn['Attribute'][2] # Gender
            HLR = dfValsn['Attribute'][3] # Left/Right
            HR = dfValsn['Attribute'][4] # Region
            HSSi = float(dfValsn['Attribute'][12]) # Steady state start time
            HSSf = float(dfValsn['Attribute'][13]) # Steady state final time
            HTMax = float(dfValsn['Attribute'][14]) # Time @ max force
            HFMax = float(dfValsn['Attribute'][16]) # Value at max force
            HFSS = float(dfValsn['Attribute'][17]) # Value at steady state

            dfn = pd.read_csv(os.path.join(subPath, k), sep="\t", header=30)
            dfn.columns = ['Time', 'Extension', 'Force']
            tn = dfn.Time
            dn = dfn.Extension
            force = dfn.Force

            # SS Array
            ssTimeArray = np.array([HSSi, HSSf])
            ssValArray = np.array([HFSS, HFSS])

            axn.plot(tn, force*1e3,'-', color=c1, linewidth=1, markersize=2, label = '{}, {}, Age: {}'.format(HID, HLR, HAGE), alpha = 1)
            if str(HFMax) == 'nan' and str(HSSi) == 'nan':
                continue

            if str(HFMax) != 'nan':
                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()])

            if str(HSSi) != 'nan':
```

```
                axn.plot(ssTimeArray, ssValArray,'-', color=c3, linewidth=3, markersize=2, label = 'Steady State - {:.4f} (mN)'.format(HFSS), path_effects=[pe.Stroke(linewidth=5, foreground='k'), pe.Normal()])

            # append to the list for plot identification
            show.append(fign.gca().get_lines())

            # append the max value
            maxVals.append(HFMax)

        path, subFolder = os.path.split(subPath) # Extract the folder name
        axn.axhline(y=0, color='k')
        axn.set_xlabel('Time (sec)',fontsize=18)
        axn.set_ylim(-0.5, max(maxVals) + 1)
        axn.set_ylabel('Force (mN)',fontsize=18)
        axn.set_title(subFolder, fontsize=20)
        axn.grid(True, which='both')
        for i,j in enumerate(show):
            graphLine = []
            step = len(j)
            if i == 0:
                for k in range(0, len(j), 1):
                    graphLine.append(k)
            elif i > 0:
                for k in range(len(show[i-1]), len(j), 1):
                    graphLine.append(k)
            linesn = fign.gca().get_lines()
            legendn = axn.legend([linesn[i] for i in graphLine],[linesn[i].get_label() for i in graphLine], prop={"size":10}, loc=i+1, title = 'Data')
            axn.add_artist(legendn)
        plt.show()
        fign.savefig(os.path.join(subPath, '{}_All'.format(directory) + '_F_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
        plt.close()

    elif i == '30_59_Equator' or i == '30_59_Posterior' or i == '60_89_Equator' or i == '60_89_Posterior':
        # Plot the groups for paper 3 but don't include the max and steady state value (All legend items are in a single legend)
        print(i)
        subPath = os.path.join(filePath, i)
        (folderName, directory) = os.path.split(subPath)
        os.chdir(subPath)
        subTxtFiles = [x for x in glob.glob('*.txt')] # Posterior/Equator
        show = []
        maxVals = []

        color1 = iter(cm.rainbow(np.linspace(0,1,len(subTxtFiles))))
        color2 = iter(cm.rainbow(np.linspace(0,1,len(subTxtFiles))))
        color3 = iter(cm.rainbow(np.linspace(0,1,len(subTxtFiles))))
        fign, axn = plt.subplots()

        for j,k in enumerate(subTxtFiles):
            c1 = next(color1)
            c2 = next(color2)
            c3 = next(color3)
```

```
            """ Read in the csv file """
            dfValsn = pd.read_csv(os.path.join(subPath, k), sep="\t", nrows=29, header=None, names=['Var','Attribute'])

            """ File Attributes """
            HID = dfValsn['Attribute'][0] # ID
            HAGE = dfValsn['Attribute'][1] # Age
            HG = dfValsn['Attribute'][2] # Gender
            HLR = dfValsn['Attribute'][3] # Left/Right
            HR = dfValsn['Attribute'][4] # Region
            HSSi = float(dfValsn['Attribute'][12]) # Steady state start time
            HSSf = float(dfValsn['Attribute'][13]) # Steady state final time
            HTMax = float(dfValsn['Attribute'][14]) # Time @ max force
            HDispMax = float(dfValsn['Attribute'][15]) # Disp @ max force
            HFMax = float(dfValsn['Attribute'][16]) # Value at max force
            HFSS = float(dfValsn['Attribute'][17]) # Value at steady state

            dfn = pd.read_csv(os.path.join(subPath, k), sep="\t", header=30)
            dfn.columns = ['Time', 'Extension', 'Force']
            tn = dfn.Time
            dn = dfn.Extension # mm
            force = dfn.Force*1e3 # N ---> mN

            # SS Array
            ssTimeArray = np.array([HSSi, HSSf])
            ssValArray = np.array([HFSS, HFSS])
            ssDispArray = np.array([dn[tn == HSSi].values[0] if HSSi is not np.nan else np.nan,
                                    dn[tn == HSSf].values[0] if HSSi is not np.nan else np.nan])

            axn.plot(dn, force,'-', color=c1, linewidth=1, markersize=2, label = '{}, {}, Age: {}'.format(HID, HLR, HAGE), alpha=0.3)

            if str(HFMax) != 'nan':
                maxIndex = force[tn == HTMax].index.values[0] # Location in the array for the max force

                # slope calculation for 10 seconds prior to the max peel force
                x10 = dn[maxIndex-100:maxIndex] # Array from maxIndex - 100 (10 sec) to location of max force
                y = force[maxIndex-100:maxIndex] # Array from maxIndex - 100 (10 sec) to location of max force
                curveFit10, Params10, Rsqrd10 = Least_Squares(x10,y) # Perform least squares and return

                # slope calculation for 20 seconds prior to the max peel force
                x20 = dn[maxIndex-200:maxIndex] # Array from maxIndex - 200 (20 sec) to location of max force
                y = force[maxIndex-200:maxIndex] # Array from maxIndex - 200 (20 sec) to location of max force
                curveFit20, Params20, Rsqrd20 = Least_Squares(x20,y) # Perform least squares and return

                # slope calculation for 30 seconds prior to the max peel force
                x30 = dn[maxIndex-300:maxIndex] # Array from maxIndex - 300 (30 sec) to location of max force
```

```python
                y = force[maxIndex-300:maxIndex] # Array from maxIndex - 300 (30 sec) to location of max force
                curveFit30, Params30, Rsqrd30 = Least_Squares(x30,y) # Perform least squares and return

                # Slope calculation from zero to max peel force
                x0 = dn[0:maxIndex] # Array from 0 to location of max force
                y = force[0:maxIndex] # Array from 0 to location of max force
                curveFit0, Params0, Rsqrd0 = Least_Squares(x0,y) # Perform least squares and return

                # 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)
                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)
                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)
                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)

                # # Plot the max force value
                # 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()]) #

            # if str(HSSi) != 'nan':
            #     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)

            # append to the list for plot identification
            show.append(fign.gca().get_lines())

            # append the max value
            maxVals.append(HFMax)

        path, subFolder = os.path.split(subPath) # Extract the folder name
        axn.axhline(y=0, color='k')
        axn.set_xlabel('Displacement (mm)',fontsize=18)
        axn.set_xlim(0, 9) # focus in on just ramp up
        axn.set_ylim(-0.5, max(maxVals) + 1)
        axn.set_ylabel('Force (mN)',fontsize=18)
        axn.set_title(subFolder, fontsize=20)
        # axn.grid(True, which='both')

        # where some data has already been plotted to ax
        handles, labels = axn.get_legend_handles_labels()

        # Manually add items to the legend
        fit_0 = mpatches.Patch(color='black', label=r'Curvefit (0 - $Time_{Max})$')
        fit_30 = mpatches.Patch(color='green', label=r'Curvefit (30 s before $Time_{Max})$')
```

```
        fit_20 = mpatches.Patch(color='blue', label=r'Curvefit (20 s before
        ↪  $Time_{Max})$')
        fit_10 = mpatches.Patch(color='red', label=r'Curvefit (10 s before
        ↪  $Time_{Max})$')

        # handles is a list, so append manual patch
        # handles.append(fit_0)
        handles.append(fit_30)
        handles.append(fit_20)
        handles.append(fit_10)

        axn.legend(handles=handles, loc='best')

        plt.show()
        fign.savefig(os.path.join(subPath, '{}_All'.format(directory) +
        ↪  '_F_vs_t.png'), dpi=300, bbox_inches='tight') # Save figure
        plt.close()

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:

$$\mathbf{r} = \begin{cases} \langle m,\ \max(\mathbf{RF}) \rangle_{Exp} - \langle m,\ \max(\mathbf{RF}) \rangle_{Sim} & \text{, Modulus} \quad (1.1) \\ \langle \max(\mathbf{RF}),\ \overline{\mathbf{SS}} \rangle_{Exp} - \langle \max(\mathbf{RF}),\ \overline{\mathbf{SS}} \rangle_{Sim} & \text{, Cohesive} \quad (1.2) \end{cases}$$

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.* </>

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 19 15:22:26 2020

@author: Kiffer2

This Python script does the following

    1) Select input parameters
    2) Generates the filename/description
    3) Calls Abaqus to create the .inp file w/ attributes & runs the job
    4) Creats a folder with the filename
    5) Extracts data from the Abaqus.odb file and creates two output files
↪   (Field/Hist)
    6) Plots the data
    7) Moves all files that have the same filename

"""

import os
import sys
import numpy as np
import pandas as pd
import itertools as it # iteration tools (product fcn)
from scipy import *
import scipy.optimize as opt
import lmfit as lf # lmfit
import pdb
import subprocess
import pprint

# Define the location of the Abaqus Working Directory
# specific folder path where this file is located
pythonScriptPath = os.getcwd()
abqWD, pythonFiles = os.path.split(pythonScriptPath) # split file path

# pythonScriptCreateINP_Run_ABQ (pS_ABQ)
pS_ABQ = os.path.join(pythonFiles, 'Bond_T1_EyeModel_Generate_Abaqus.py')
# pythonScriptExtract (pSE)
pSE = os.path.join(pythonFiles, 'Bond_T1_EyeModel_DataExtract.py')

# In[Job Info]

optE_V = False
optBondParams = False
optBondParams_no_db = True
sweep = False

if optE_V == True:
    optimization = 'E_V'

    """ Tie interface """
    tieInterface = True

    """ Objective Function Flags """
```

```
    slopeFlag = True
    maxForceFlag = True
    ssForceFlag = False # Only used for damage

    """ Traction separation """
    BondStatus = False # If "False" then a tied interface will be used
    PDFMStatus = False # If "False" then no post damage will occur


if optBondParams == True:
    optimization = 'FNFSdbufnufs'

    """ Tie interface """
    tieInterface = False

    """ Objective Function Flags """
    slopeFlag = False
    maxForceFlag = True
    ssForceFlag = True # Only used for damage

    """ Traction separation """
    BondStatus = True # If "False" then a tied interface will be used
    PDFMStatus = True # If "False" then no post damage will occur


if optBondParams_no_db == True:
    optimization = 'FNFSufnufs'

    """ Tie interface """
    tieInterface = False

    """ Objective Function Flags """
    slopeFlag = False
    maxForceFlag = True
    ssForceFlag = True # Only used for damage

    """ Traction separation """
    BondStatus = True # If "False" then a tied interface will be used
    PDFMStatus = True # If "False" then no post damage will occur


if sweep == True:
    optimization = None

    """ Tie interface """
    tieInterface = False

    """ Objective Function Flags """
    slopeFlag = False
    maxForceFlag = True
    ssForceFlag = True # Only used for damage

    """ Traction separation """
    BondStatus = True # If "False" then a tied interface will be used
    PDFMStatus = True # If "False" then no post damage will occur


""" Objective Function Error Formulation """
```

```python
objFunErr = []
objFunErr.append('Difference') # Experimental - Simulated
objFunErr.append('Ratio') # Experimental/Simulated
# (Experimental - Simulated)/Experimental
objFunErr.append('Relative uncertainty')
# Change to specific optimization parameter.  If 'None', no optimization
objErr = objFunErr[0]
print('Objective function error formulation = ', objErr)

# Calculation for error
ErrorCalculation = []
ErrorCalculation.append('two-point method') # Slope, Peak force, SS Force
ErrorCalculation.append('data-trace method') # interpolated array

errorMethod = ErrorCalculation[0]
print('Error method calculation = ', errorMethod)

''' Symmetry '''
# Split model in half and multiply output by 2
symmetry = False # *bond doesn't allow symmetric BCs

''' Simplified '''
# Remove the rigid body on the plastic tab and glue
simplified = True

''' Gravity '''
# Turn gravity on/off
gravity = False # Keep off until model is updated

# In[Comparison Data Trace]
compareDataFolder = 'PeelDataCompare'
specificDataTrace = 'Trace_51_Instron_Data.txt' # Data trace number
timeBeforePeak = 20 # Default is 20 seconds
dataCompare = os.path.join(abqWD,compareDataFolder,specificDataTrace)
dfValsn = pd.read_csv(dataCompare, sep="\t", nrows=29, header=None,
                      names=['Var', 'Attribute'])

""" File Attributes """
HID =            dfValsn['Attribute'][0]
HAGE =           dfValsn['Attribute'][1]
HG =             dfValsn['Attribute'][2]
HLR =            dfValsn['Attribute'][3]
HR =             dfValsn['Attribute'][4]
HSSi =     float(dfValsn['Attribute'][12])
HSSf =     float(dfValsn['Attribute'][13])
HTMax =    float(dfValsn['Attribute'][14])
HDispMax = float(dfValsn['Attribute'][15])
HFMax =    float(dfValsn['Attribute'][16]) # (mN)
HFSS =     float(dfValsn['Attribute'][17])
# (mN/m) slope from 20 seconds prior to max force value
HSlope20 = float(dfValsn['Attribute'][20])

dfn = pd.read_csv(os.path.join(dataCompare), sep="\t", header=30)
dfn.columns = ['Time', 'Extension', 'Force']
tn = dfn.Time
dn = dfn.Extension
df = dfn.Force # (N)

```

```
maxForceMeasured = HFMax # Value from data trace
maxSlopeMeasured = HSlope20 # slope from 20 seconds prior to max force value
SS_Measured = HFSS # simulated steady state force

# In[Functions]

if BondStatus == False and PDFMStatus == True:
    print('Unable to have PDFM without *Bond')
    sys.exit()

""" Tic Toc to determine runtime """
def tic():
    #Homemade version of matlab tic and toc functions
    import time
    global startTime_for_tictoc
    startTime_for_tictoc = time.time()

def toc():
    import time
    if 'startTime_for_tictoc' in globals():
        print("Elapsed time is " + str(time.time() - startTime_for_tictoc) +
              " seconds.")
        timeDiff = time.time() - startTime_for_tictoc
        return timeDiff
    else:
        print("Toc: start time not set")

try:
    os.environ.pop('PYTHONIOENCODING')
except KeyError:
    pass

# Import modules that plot/move all abq files to the new foldername
from ParameterSelection import ReadRAWDataTrace
from Bond_T1_Data_Plot import PlotAbqData
from Move_ABQ_Files_To_Folder import MoveAbqFiles
from Bond_T1_Residual import findResidual

newLine = '\n' + 77*'-' + '\n'

def jobAttributes():
    """
    Input: parameters used to create the filename and job description

    Output: namei, fileName, JobDescription
    """

    # Build the fileName
    fi = [] # initialize array
    fi.append(      '{}'.format(namei))
    fi.append(       'g') if gravity == True else ''
    fi.append(     'sym') if symmetry == True else ''
    fi.append(     't{}'.format(time))
    fi.append(    'E1{}'.format(e1Seedi[0]))
    fi.append(    'E2{}'.format(e2Seedi[0]))

    if simplified == False:
        fi.append(    'PT{}'.format(ptSeedi[0]))
```

```
        fi.append(    'G{}'.format(gSeedi[0]))

    fi.append(    'V1{}'.format(v1Seedi[0]))
    fi.append(    'V2{}'.format(v2Seedi[0]))
    fi.append(     'R{}'.format(rSeedi[0]))
    fi.append(     'F{}'.format(massScaleFactori[0]))
    fi.append(    'MS{}'.format(massScaleTimeIncrementi[0]))
    fi.append('RE{:.0e}'.format(RetinaYoungsModulus_i))

    if optimization is not None:
        if optimization.find('E_V') == -1:
            fi.append('VE{:.0e}'.format(VitreousYoungsModulus_i))

    # If True, then damage initation, If optimization, get rid of the title
    # (Not an integer anymore)
    if BondStatus == True and optimization is None:
        fi.append(    'FN{}'.format(int(FNi[0])))
        fi.append(    'FS{}'.format(int(FSi[0])))

    # If True, then damage evolution, If optimization, get rid of the title
    # (Not an integer anymore)
    if ((BondStatus == True) and (PDFMStatus == True) and
        (optimization is None)):
        fi.append(    'db{}'.format(int(dbi[0])))
        fi.append(    'ufn{}'.format(int(ufni[0])))
        fi.append(    'ufs{}'.format(int(ufsi[0])))

    # .format(optimization))  optimization flag (I.e. RE, VE, FN, FS,
    # ufn, or none)
    fi.append(     'opt') if optimization is not None else ''
    fi.append(     'TIE') if tieInterface == True else ''

    if sweep == True:
        # get rid of all attributes because a sweep is taking place
        fi = fi[0]

    """ Build file name and description """
    fileName = ''.join(item for item in fi)
    # fix header so no decimals, math show up in title
    fileName = fileName.replace('+', '_').replace('-', '_').replace('.', '_')
    jobNameString = 'Job Name - {}'.format(fileName)

    # used for simplification of script
    # Large value
    multStrL = ('\n\tgeometric multiplier = 2**{}, \n\tbase value = {}, ' +
                '\n\tmodel value = {}')
    # Small value
    multStrS = ('\n\tgeometric multiplier = 0.5**{}, \n\tbase value = {}, ' +
                '\n\tmodel value = {}')

    # Build the model description
    si = [] # initialize array
    si.append(newLine)
    si.append('({}) = model name'.format(namei))
    si.append(jobNameString)
    si.append('(g) - Gravity') if gravity == True else si.append('No Gravity')
    # update name in list
    si.append('(sym) SYMMETRIC model (XY) Plane') if symmetry == True else ''
```

```
    # update name in list
    si.append('(t) Simulated time {} (s)'.format(time))

    # Eye Holder
    si.append(('(E1) Eye holder outside edge seed size (Max) (SINGLE BIAS):  '
               + multStrS + ' (m)').format(*e1Seedi))
    si.append(('(E2) Eye holder inside edge seed size (Min):  ' + multStrS +
               ' (m)').format(*e2Seedi))

    # If simplified is in the title, get rid of glue and platic tab
    if simplified == False:
        si.append(('(PT) Plastic tab seed size:  ' + multStrS +
                   ' (m)').format(*ptSeedi))
        si.append(('(G) Glue seed size:  ' + multStrS +
                   ' (m)').format(*gSeedi))

    # Vitreous
    si.append(('(V1) Vitreous seed size max (side edge seed set)-' +
               '(SINGLE BIAS):  ' + multStrS + ' (m)').format(*v1Seedi))
    si.append(('(V2) Vitreous seed size min (top edge in contact with ' +
               'retina):  ' + multStrS + ' (m)').format(*v2Seedi))

    # Retina
    si.append(('(R) Retina seed size:  ' + multStrS + ' (m)').format(*rSeedi))

    # Mass scale factor
    si.append(('(F) Mass scale factor:  ' + multStrL +
               '').format(*massScaleFactori))

    # Mass scale time increment
    si.append(('(MS) Mass scale time increment:  ' + multStrS +
               ' (s)').format(*massScaleTimeIncrementi))

    # Material properties (Young's Modulus)
    si.append("(RE) Retina Young's Modulus:  model value = {} (Pa)"
              .format(RetinaYoungsModulus_i))
    si.append("(VE) Vitreous Young's Modulus:  model value = {} (Pa)"
              .format(VitreousYoungsModulus_i))

    # If True, then damage initation
    if BondStatus == True:
        si.append(('(FN) FN:  ' + multStrL + ' (N)').format(*FNi))
        si.append(('(FS) FS:  ' + multStrL + ' (N)').format(*FSi))

    # If True, then damage evolution
    if BondStatus == True and PDFMStatus == True:
        si.append(('(db) Bead size:  ' + multStrL +
                   ' (m**2)').format(*dbi))
        si.append(('(ufn) Normal displacement:  ' + multStrL +
                   ' (m)').format(*ufni))
        si.append(('(ufs) Shear displacement:  ' + multStrL +
                   ' (m)').format(*ufsi))

    # Optimization
    if optimization is not None:
        si.append('Optimization of {}'.format(optimization))
        si.append('Objective function error formulation is the ' +
                  '{} calculation'.format(objErr))
```

```python
        si.append('Objective error calculation is the {}'.format(errorMethod))

    if optimization == None:
        si.append('Parametric sweep')
        si.append('Objective function error formulation is the ' +
                  '{} calculation'.format(objErr))
        si.append('Objective error calculation is the {}'.format(errorMethod))

    # Tied interface
    if tieInterface == True:
        si.append('Tied interface between the Retina and the Vitreous')

    # Data trace being compared for optimization
    si.append('The data trace being compared is:  {}'
              .format(specificDataTrace))

    si.append(newLine)

    # Job description
    jobDescription =  '\n'.join(item for item in si)

    print(newLine)
    print(fileName)
    print(newLine)
    print(jobDescription)

    # Write a .txt file with the file attributes
    outfile = open(os.path.join(abqWD, fileName +'.txt'),'w')
    line = ('The file name indicates what parameters were used to define ' +
            'the model\n')
    outfile.write(line)
    line = '\n' + jobDescription + '\n'
    outfile.write(line)
    outfile.close()
    print(outfile)
    return namei, fileName, jobDescription

def GenerateAbaqusModels():
    """
    Function used to call Command Line (Windows Batch file)

    Parameters
    ----------
    fileName : abaqus job with paramters

    """
    # ----------------------- Step 2 -----------------------#
    # Generates the filename/description
    modelName, fileName, jobDescription = jobAttributes()

    # Strip job description from spaces and new lines
    # replace new lines, spaces, equal signs
    jobDescription = jobDescription.replace(' ', 'SPACE')
    jobDescription = jobDescription.replace('\n', 'NEWLINE')
    jobDescription = jobDescription.replace('\t', 'TAB')
    jobDescription = jobDescription.replace('=', 'EQUALSSIGN')

    print(newLine)
```

```

    # ----------------------- Step 3 ----------------------#
    # Calls Abaqus to create the job with the filename just created and
    # run the job

    # Strip spaces and make strings
    ABQ = []
    ABQ.append(pS_ABQ) # python 2.7 script

    # gravity
    ABQ.append(','.join([i.strip(' ') for i in str(gravity).split(',')]))

    # symmetry
    ABQ.append(','.join([i.strip(' ') for i in str(symmetry).split(',')]))

    # Simplified model
    ABQ.append(','.join([i.strip(' ') for i in str(simplified).split(',')]))

    ABQ.append(modelName) # model name
    ABQ.append(fileName) # file name

    # time
    ABQ.append(','.join([i.strip(' ') for i in str(time).split(',')]))

    # eye holder seed size 1
    ABQ.append(','.join([i.strip(' ') for i in str(e1Seedi).split(',')]))

    # eye holder seed size 2
    ABQ.append(','.join([i.strip(' ') for i in str(e2Seedi).split(',')]))

    # plastic tab seed size
    ABQ.append(','.join([i.strip(' ') for i in str(ptSeedi).split(',')]))

    # glue seed size
    ABQ.append(','.join([i.strip(' ') for i in str(gSeedi).split(',')]))

    # vitreous seed 1 size
    ABQ.append(','.join([i.strip(' ') for i in str(v1Seedi).split(',')]))

    # vitreous seed 2 size
    ABQ.append(','.join([i.strip(' ') for i in str(v2Seedi).split(',')]))

    # retina seed size
    ABQ.append(','.join([i.strip(' ') for i in str(rSeedi).split(',')]))

    # mass scale factor
    ABQ.append(','.join([i.strip(' ') for i in
                         str(massScaleFactori).split(',')]))

    # mass scale time
    ABQ.append(','.join([i.strip(' ') for i in
                         str(massScaleTimeIncrementi).split(',')]))

    # Retina Young's Modulus
    ABQ.append(','.join([i.strip(' ') for i in
                         str(RetinaYoungsModulus_i).split(',')]))

    # Vitreous Young's Modulus
```

```
    ABQ.append(','.join([i.strip(' ') for i in
                         str(VitreousYoungsModulus_i).split(',')]))

    # BondStatus
    ABQ.append(','.join([i.strip(' ') for i in
                         str(BondStatus).split(',')]))
    ABQ.append(','.join([i.strip(' ') for i in str(FNi).split(',')])) # FN
    ABQ.append(','.join([i.strip(' ') for i in str(FSi).split(',')])) # FS

    # PDFMStatus
    ABQ.append(','.join([i.strip(' ') for i in
                         str(PDFMStatus).split(',')]))
    ABQ.append(','.join([i.strip(' ') for i in str(dbi).split(',')])) # db
    ABQ.append(','.join([i.strip(' ') for i in str(ufni).split(',')])) # ufn
    ABQ.append(','.join([i.strip(' ') for i in str(ufsi).split(',')])) # ufs

    # Optimization None/optimized parameters
    ABQ.append(','.join([i.strip(' ') for i in str(optimization).split(',')]))

    # Tied interface
    ABQ.append(','.join([i.strip(' ') for i in str(tieInterface).split(',')]))

    # Model description
    ABQ.append(jobDescription)

    ABQ_parse_string = 'abaqus cae noGUI={} --' + (len(ABQ)-1)*' {}'

    # # Used for debugging, comment out to copy/paste output to cmd window
    # # # to check and see if it works
    # print(ABQ_parse_string.format(*ABQ))
    # pdb.set_trace()

    cmd = subprocess.Popen(ABQ_parse_string.format(*ABQ),
                           cwd=r'{}'.format(abqWD), stdin=subprocess.PIPE,
                           stdout=subprocess.PIPE, stderr=subprocess.PIPE,
                           shell=True).communicate()[0]

    print(newLine)
    print('Abaqus has generated the .inp and executed the job')

    # ----------------------- Step 4 -----------------------#
    # Creates a folder with the filename
    folderDirectory = os.path.join(abqWD, fileName)
    if not os.path.exists(folderDirectory):
        os.makedirs(folderDirectory)
    dataDirectory = os.path.join(folderDirectory, 'Output')
    if not os.path.exists(dataDirectory):
        os.makedirs(dataDirectory)
    figuresDirectory = os.path.join(dataDirectory, 'Figures')
    if not os.path.exists(figuresDirectory):
        os.makedirs(figuresDirectory)
    print(newLine)
    print('New file location:\n{} \n'.format(folderDirectory))

    # ----------------------- Step 5 -----------------------#
    """
    Extracts data from the Abaqus.odb file and creates two output files
    (Field/Hist).  Create the name to be parsed into ABQ from the command
```

```
    line through a subprocess
    """
    ABQ = []
    ABQ.append(pSE)
    ABQ.append(fileName)
    ABQ.append(gravity)
    ABQ.append(symmetry)
    ABQ.append(simplified)
    ABQ.append(BondStatus)
    ABQ.append(PDFMStatus)

    ABQ_parse_string = 'abaqus python' + len(ABQ)*' {}'

    # # # Used for debugging, comment out to copy/paste output to cmd window
    # # # to check and see if it works
    # print(ABQ_parse_string.format(*ABQ))
    # pdb.set_trace()

    cmd = subprocess.Popen(ABQ_parse_string.format(*ABQ),
                            cwd=r'{}'.format(abqWD), stdin=subprocess.PIPE,
                            stdout=subprocess.PIPE, stderr=subprocess.PIPE,
                            shell=True).communicate()[0]

    print(newLine)
    print('Abaqus has extracted Field/History output:  ' +
          '\n{} \n'.format(dataDirectory))

    # ----------------------- Step 6 -----------------------#
    # Plot data and store it in the variable name folder under "Figures"
    print(fileName)
    print(dataDirectory)
    PlotAbqData(fileName, dataDirectory, dataCompare, BondStatus,
                PDFMStatus)
    print(newLine)
    print('New data plots:\n{} \n'.format(figuresDirectory))

    # ----------------------- Step 7 -----------------------#
    # Move all abaqus files to the folder with the same name
    MoveAbqFiles(fileName, folderDirectory, abqWD)
    print(newLine)
    print('Files have been moved to:  \n{} \n'.format(dataDirectory))

    # ------------------- Step 8 (Error for minimization) -------------------#
    maxForceTime = 100 # s
    # slope is (mN/m)
    residVals = findResidual(fileName, dataDirectory, maxForceTime,
                             dataCompare, objErr, slopeFlag, maxForceFlag,
                             ssForceFlag)
    # Unpack
    slopeSimulated =    residVals[0]
    maxForceSimulated = residVals[1]
    SSmeanSimulated =   residVals[2]
    SSmedianSimulated = residVals[3]
    y_new_exp_disp =    residVals[4]
    y_new_sim_disp =    residVals[5]

    # (return slope, force, and maxucrt @ specified displacement)
    fcnReturn = []
```

```python
    fcnReturn.append(fileName)
    fcnReturn.append(slopeSimulated)
    fcnReturn.append(maxForceSimulated)
    fcnReturn.append(SSmeanSimulated)
    fcnReturn.append(SSmedianSimulated)
    fcnReturn.append(y_new_exp_disp)
    fcnReturn.append(y_new_sim_disp)
    return fcnReturn


def writeOutputData(fileNameList):
    print("\nWriting out the Reaction Force data...")
    filename = os.path.join(abqWD, 'FEAAttributes' + '.txt')
    outfile = open(filename,'w')
    sep = '\t'
    Header = [] # List of items for the Header
    Header.append('FileName')
    Header.append('Time')
    Header.append('E1')
    Header.append('E2')
    Header.append('PT')
    Header.append('G')
    Header.append('V1')
    Header.append('V2')
    Header.append('R')
    Header.append('F')
    Header.append('MS')
    Header.append('RE')
    Header.append('VE')
    Header.append('FN')
    Header.append('FS')
    Header.append('db')
    Header.append('UFN')
    Header.append('UFS')
    Header.append('BondStatus')
    Header.append('PDFMStatus')
    Header.append('Optimization')
    Header.append('TIE')
    Header.append('errorListL2Norm')
    Header.append('ObjectiveFunction')
    Header.append('maxSlopeSimulated')
    Header.append('maxForceSimulated')
    Header.append('SSmeanSimulated')
    Header.append('simTime')
    line = sep.join(item for item in Header)
    outfile.write(line)
    outfile.write('\n')
    outfile.write('\t'.join(str(item) for item in attributeArray_0))
    for i in list(fileNameList):
        outfile.write('\n')
        tempList = [str(i[0])] # filename
        for j in list(i[1]):
            tempList.append(str(j)) # file attributes
        tempList.append(str(i[2])) # sim time
        outfile.write('\t'.join(str(item) for item in tempList))
    outfile.close()
    print("\nDone!")
    print("\nThe output file will be named '{}".format(filename) + "'")
```

```
    print("\nIt will be in the same working directory as your Abaqus model\n")

    # Print File of tests ran in order
    print("\nWriting out the Reaction Force data...")
    filename = os.path.join(abqWD, 'FEAFileList' + '.txt')
    outfile = open(filename,'w')
    line = 'FileName'
    outfile.write(line)
    for i in list(fileNameList):
        line = '\n%s' % (i[0])
        outfile.write(line)
    outfile.close()
    print("\nDone!")
    print("\nThe output file will be named '{}".format(filename) + "'")
    print("\nIt will be in the same working directory as your Abaqus model\n")



if __name__ == '__main__':
    # Run the function

    # ----------------------- Step 1 -----------------------#
    # T1
    name = ['T1']

    paramSelect = ReadRAWDataTrace(dataCompare, abqWD, timeBeforePeak)

    t0, t1, tshift, fe = paramSelect # Unpack variables

    if t0 > tshift:
        # If the t1 value is greater than tshfit, use tshift for
        # the simulation time
        # Shouldn't have to do this as this issue has been handeled
        t0 = tshift
        print('updated the time to be the shift value')

    # Determine which time to use (Max value or steady state)
    if optE_V == True:
        time = int(t0)

    elif optBondParams == True or sweep == True:
        time = int(t1)

    elif optBondParams_no_db == True or sweep == True:
        time = int(t1)

    # Select input parameters

    ''' First round of optimized material properties '''
    VitreousYoungsModulus_0 = 399.4216617623035
    FNValOpt = -8
    FSValOpt = -8
    dbValOpt = -25
    ufnValOpt = -8
    ufsValOpt = -8

    """ Retina Young's Modulus """
    RetinaYoungsModulus_0 = 11120.0 # Pa Optimized with the vitreous
```

```python

    """ Eye holder inside edge """
    e1Seed_0 = 1 # Base seed
    e1SeedArray = [] # Array of multipliers
    n = 12 # number of increments
    for i in range(11, n):
        # Decrease mesh seed by a factor of 2
        e1SeedArray.append([i, e1Seed_0, e1Seed_0*(0.5)**i])

    """ Eye holder outside edge """
    # This will most likely never get smaller (saves computational time)
    e2Seed_0 = 1 # Base seed
    e2SeedArray = []
    n = 7 # number of increments
    for i in range(6, n):
        # Decrease mesh seed by a factor of 2
        e2SeedArray.append([i, e2Seed_0, e2Seed_0*(0.5)**i])

    """ Plastic tab """
    ptSeed_0 = 1 # Plastic tab seed size
    ptSeedArray = [] # Array of multipliers
    n = 7 # number of increments
    for i in range(6, n):
        # Decrease mesh seed by a factor of 2
        ptSeedArray.append([i, ptSeed_0, ptSeed_0*(0.5)**i])

    """ Glue """
    gSeed_0 = 1 # Glue seed size
    gSeedArray = [] # Array of multipliers
    n = 8 # number of increments
    for i in range(7, n):
        # Decrease mesh seed by a factor of 2
        gSeedArray.append([i, gSeed_0, gSeed_0*(0.5)**i])

    """ Vitreous """
    # smaller seed size
    v1Seed_0 = 1 # Vitreous (max seed size)
    v1SeedArray = [] # Array of multipliers
    # n = 11 # number of increments
    # for i in range(10, n):
    #     # Decrease mesh seed by a factor of 2
    #     v1SeedArray.append([i, v1Seed_0, v1Seed_0*(0.5)**i])

    # Comment out when parameters have been optimized
    v1ValOpt = 11.38 # 10
    v1SeedArray.append([v1ValOpt, v1Seed_0, v1Seed_0*(0.5)**v1ValOpt])

    # larger seed size (should be factor of 4 times smaller ## 2 numbers)
    v2Seed_0 = 1 # Vitreous (min seed size)
    v2SeedArray = [] # Array of multipliers
    n = 9 # number of increments
    for i in range(8, n):
        # Decrease mesh seed by a factor of 2
        v2SeedArray.append([i, v2Seed_0, v2Seed_0*(0.5)**i])

    """ Retina """
    rSeed_0 = 1 # Base seed
    rSeedArray = [] # Array of multipliers
```

```
    # n = 11 # number of increments
    # for i in range(10, n):
    #     # Decrease mesh seed by a factor of 2
    #     rSeedArray.append([i, rSeed_0, rSeed_0*(0.5)**i])

    rValOpt = 11.3275 # 10 #
    rSeedArray.append([rValOpt, rSeed_0, rSeed_0*(0.5)**rValOpt])


    """ mass scale factor """
    massScaleFactor_0 = 1
    massScaleFactorArray = [] # Array of multipliers
    n = 1 # number of increments
    for i in range(0, n):
        # Increase by a factor of 2
        massScaleFactorArray.append([i, massScaleFactor_0,
                                     massScaleFactor_0*2**i])

    """ mass scale time increment """
    massScaleTimeIncrement_0 = 1
    massScaleTimeArray = [] # multiplier and value
    n = 8 # number of increments
    for i in range(7, n):
        # Decrease by a factor of 2
        massScaleTimeArray.append([i, massScaleTimeIncrement_0,
                                   massScaleTimeIncrement_0*(0.5)**i])

    if massScaleTimeIncrement_0 == 0:
        print('No Mass Scaling... This will take a while...ABAQUS is ' +
              'deciding for us')

    """ FN """
    FN_0 = 1
    FNArray = [] # Array of multipliers
    # n = 10 # 10 works when using max stress criteria
    # for i in range(9, n):
    #     # Increase by a factor of 2
    #     FNArray.append([i, FN_0, FN_0*(2)**i])

    # Comment out when parameters have been optimized
    FNArray.append([FNValOpt, FN_0, FN_0*(2)**FNValOpt])

    """ FS """
    FS_0 = 1
    FSArray = [] # Array of multipliers
    # n = 10
    # for i in range(9, n):
    #     # Increase by a factor of 2
    #     FSArray.append([i, FS_0, FS_0*(2)**i])

    # Comment out when parameters have been optimized
    FSArray.append([FSValOpt, FS_0, FS_0*(2)**FSValOpt])

    """ db """
    db_0 = 1
    dbArray = [] # Array of multipliers
    # n = 10
    # for i in range(9, n):
```

```python
    #     # Increase by a factor of 2
    #     dbArray.append([i, db_0, db_0*(2)**i])

    # Comment out when parameters have been optimized
    dbArray.append([dbValOpt, db_0, db_0*(2)**dbValOpt])

    """ ufn """
    ufn_0 = 1
    ufnArray = [] # Array of multipliers
    # n = -8
    # for i in range(-9, n):
    #     # Increase by a factor of 2
    #     ufnArray.append([i, ufn_0, ufn_0*(2)**i])

    ufnArray.append([ufnValOpt, ufn_0, ufn_0*(2)**ufnValOpt])

    """ ufs """
    ufs_0 = 1
    ufsArray = [] # Array of multipliers
    # n = -8
    # for i in range(-9, n):
    #     # Increase by a factor of 2
    #     ufsArray.append([i, ufs_0, ufs_0*(2)**i])

    ufsArray.append([ufsValOpt, ufs_0, ufs_0*(2)**ufsValOpt])

    errorList = np.nan # initial error

    """ Attribute Array Initial Values """
    attributeArray_0 = []
    attributeArray_0.append('BaseVals')
    attributeArray_0.append(time)
    attributeArray_0.append(e1Seed_0)
    attributeArray_0.append(e2Seed_0)
    attributeArray_0.append(ptSeed_0)
    attributeArray_0.append(gSeed_0)
    attributeArray_0.append(v1Seed_0)
    attributeArray_0.append(v2Seed_0)
    attributeArray_0.append(rSeed_0)
    attributeArray_0.append(massScaleFactor_0)
    attributeArray_0.append(massScaleTimeIncrement_0)
    attributeArray_0.append(RetinaYoungsModulus_0)
    attributeArray_0.append(VitreousYoungsModulus_0)
    attributeArray_0.append(BondStatus)
    attributeArray_0.append(FN_0)
    attributeArray_0.append(FS_0)
    attributeArray_0.append(db_0)
    attributeArray_0.append(PDFMStatus)
    attributeArray_0.append(ufn_0)
    attributeArray_0.append(ufs_0)
    attributeArray_0.append(optimization)
    attributeArray_0.append(tieInterface)
    attributeArray_0.append(errorList)
    attributeArray_0.append(objErr)
    attributeArray_0.append('simTime')


    fileNameList = [] # List of files
```

```
    counter = 0

    if optimization is not None:
        """ If the optimization variable is not "None" then optimize the
        specific variable beins passed through """

        name = name[0]

        # BondStatus = True # interested in bonding

        # # post damage failure model (If False, ignore db, ufs, and ufn,
        # # otherwise include them)
        # pdfm = False

        # Optimization method
        # optName = 'NM' # Nelder-mead
        # optName = 'P' # Powell
        optName = 'C' # COBYLA
        # optName = 'L' # LBFGSB
        # optName = 'T' # Truncated Newton
        # optName = 'S' # SLSQP
        # optName - 'TC' # Trust-Constr

        name0 = '_'.join([name, optName]) # used for optimization

        def FEA_Residual(pars, data=None):
            # Global variables
            global counter
            global name
            global name0
            global fileNameList
            global time
            global namei
            global e1Seedi
            global e2Seedi
            global ptSeedi
            global gSeedi
            global v1Seedi
            global v2Seedi
            global rSeedi
            global massScaleFactori
            global massScaleTimeIncrementi
            global RetinaYoungsModulus_i
            global VitreousYoungsModulus_i
            global FNi
            global FSi
            global dbi
            global ufni
            global ufsi

            # Parameters used for optimization
            global errorList

            print('Iteration # ', counter)

            tic() # Start time

            e1Seedi = e1SeedArray[0] # Default array
```

```
            e2Seedi = e2SeedArray[0] # Default array
            ptSeedi = ptSeedArray[0] # Default array
            gSeedi = gSeedArray[0] # Default array
            v1Seedi = v1SeedArray[0] # Default array
            v2Seedi = v2SeedArray[0] # Default array
            rSeedi = rSeedArray[0] # Default array
            massScaleFactori = massScaleFactorArray[0] # Default array
            massScaleTimeIncrementi = massScaleTimeArray[0] # Default array
            RetinaYoungsModulus_i = RetinaYoungsModulus_0 # Default value
            VitreousYoungsModulus_i = VitreousYoungsModulus_0 # Default value
            FNi = FNArray[0] # Default array
            FSi = FSArray[0] # Default array
            dbi = dbArray[0] # Default array
            ufni = ufnArray[0] # Default array
            ufsi = ufsArray[0] # Default array

            # Extract the unknown parameters from the pars class variable
            # Determine the multiplier for the title
            for key, value in pars.items():

                if key.find('ER') >= 0:
                    """ Retina Young's Modulus """
                    val = value.value
                    RetinaYoungsModulus_i = val

                elif key.find('EV') >= 0:
                    """ Vitreous Young's Modulus """
                    val = value.value
                    VitreousYoungsModulus_i = val

                elif key.find('FN') >= 0:
                    """ Fn """
                    val = value.value
                    mult = np.log(val)/np.log(2) # multiplier
                    FNi = [mult, FN_0, val]

                elif key.find('FS') >= 0:
                    """ FS """
                    val = value.value
                    mult = np.log(val)/np.log(2) # multiplier
                    FSi = [mult, FS_0, val]

                elif key.find('db') >= 0:
                    """ db """
                    val = value.value
                    mult = np.log(val)/np.log(2) # multiplier
                    dbi = [mult, db_0, val]

                elif key.find('ufn') >= 0:
                    """ ufn """
                    val = value.value
                    mult = np.log(val)/np.log(2) # multiplier
                    ufni = [mult, ufn_0, val]

                elif key.find('ufs') >= 0:
                    """ ufs """
                    val = value.value
                    mult = np.log(val)/np.log(2) # multiplier
```

```
                    ufsi = [mult, ufs_0, val]

            # Keep track of simulation results by unique names with the count
            # number.  Comment out the second part to save file space if you
            # are not interested in saving every single simulation
            namei = name0 #+ str(counter)

            # Error of the simulation
            L2Normi = np.sqrt(np.dot(errorList, errorList))

            # multipliers to be appended to the output file to show changes
            # in parameters
            aAM = [] # attributeArrayMultipliar
            aAM.append(time)
            aAM.append(e1Seedi[0])
            aAM.append(e2Seedi[0])
            aAM.append(ptSeedi[0])
            aAM.append(gSeedi[0])
            aAM.append(v1Seedi[0])
            aAM.append(v2Seedi[0])
            aAM.append(rSeedi[0])
            aAM.append(massScaleFactori[0])
            aAM.append(massScaleTimeIncrementi[0])
            aAM.append(RetinaYoungsModulus_i)
            aAM.append(VitreousYoungsModulus_i)
            aAM.append(BondStatus)
            aAM.append(FNi[0])
            aAM.append(FSi[0])
            aAM.append(PDFMStatus)
            aAM.append(dbi[0])
            aAM.append(ufni[0])
            aAM.append(ufsi[0])
            aAM.append(optimization)
            aAM.append(tieInterface)
            aAM.append(L2Normi)
            aAM.append(objErr)

            # Call the function
            # Runs jobs and saves file names
            funReturn = GenerateAbaqusModels()
            fileName =           funReturn[0]
            maxSlopeSimulated =  funReturn[1]
            maxForceSimulated =  funReturn[2]
            SSmeanSimulated =    funReturn[3]
            SSmedianSimulated =  funReturn[4]
            y_new_exp_disp =     funReturn[5]
            y_new_sim_disp =     funReturn[6]

            # add the simulated outputs to the data file
            aAM.append(maxSlopeSimulated)
            aAM.append(maxForceSimulated)
            aAM.append(SSmedianSimulated)

            # Determine the measure of error used for optimization
            # Let the data trace being passed in act as the comparison
            maxSlopeMeasured, maxForceMeasured = data

           # Error calculation
```

```
            errorDict = {} # Dictionary
            if objErr == 'Difference':
                errorDict['slope']    = (maxSlopeMeasured - maxSlopeSimulated) if slopeFlag == True else []
                errorDict['maxForce'] = (maxForceMeasured - maxForceSimulated) if maxForceFlag == True else []
                errorDict['ssForce']  = (SS_Measured - SSmeanSimulated)        if ssForceFlag == True else []
            elif objErr == 'Ratio':
                errorDict['slope']    = (1 - maxSlopeMeasured / maxSlopeSimulated) if slopeFlag == True else []
                errorDict['maxForce'] = (1 - maxForceMeasured / maxForceSimulated) if maxForceFlag == True else []
                errorDict['ssForce']  = (1 - SS_Measured / SSmeanSimulated)        if ssForceFlag == True else []
            elif objErr == 'Relative uncertainty':
                errorDict['slope']    = ((maxSlopeMeasured - maxSlopeSimulated)/maxSlopeMeasured) if slopeFlag == True else []
                errorDict['maxForce'] = ((maxForceMeasured - maxForceSimulated)/maxForceMeasured) if maxForceFlag == True else []
                errorDict['ssForce']  = ((SS_Measured - SSmedianSimulated)/SS_Measured)           if ssForceFlag == True else []
            else:
                print('Error in MaxForceError')
                sys.exit()

            # Error array values
            errorList = list(errorDict.values()) # convert to list
            errorList = [x for x in errorList if x] # get rid of empty values
            L2Normi = np.sqrt(np.dot(errorList, errorList))

            # Calculate residual
            residual = y_new_exp_disp - y_new_sim_disp # residual

            # Calculate L2Norm
            L2Norm = np.sqrt(np.dot(residual, residual))

            simulationTime = toc() # Determine run time
            # apends the fileName & File Attributes
            fileNameList.append([fileName, aAM,
                                 simulationTime])
            print('{} Error calculation: '.format(objErr), errorList)
            print('L2 norm objective calculation', L2Normi)
            print('L2 Norm residual', L2Norm)
            print('Done\n\n\n')
            counter += 1

            # Determine which calculation is going to be used for optimization
            if errorMethod == 'two-point method':
                FEA_Residual = errorList
            elif errorMethod == 'data-trace method':
                FEA_Residual = residual

            return FEA_Residual

        maxFuncEval = 200
```

```python
        tolVal = 1e-4

        # Use the data variable to input the max slope and force from the
        # known data trace
        data = [maxSlopeMeasured, maxForceMeasured]

        # Initial, Upper, and Lower bounds for parameters

        # Young's Modulus - Retina
        ER_i = 5000 # Pa
        ER_LB = 50 # Pa
        ER_UB = 11000 # Pa

        # Young's Modulus - Vitreous
        EV_i = 172 # Pa (Prony series calculation)
        # EV_i = 500 # Pa (Trying higher initial guess)
        EV_LB = 50 # Pa
        EV_UB = 2100 # Pa

        # Traction-Separation Behavior
        FN_i = 2**-6.083194128688112 # Force [N]
        FN_LB = 2**-12 # Force [N]
        FN_UB = 2**-5 # Force [N]

        FS_i = 2**-5.372267011053392 # Force [N]
        FS_LB = 2**-12 # Force [N]
        FS_UB = 2**-5 # Force [N]

        db_i = 2**-25 # Area [m**2]
        db_LB = 2**-30 # Area [m**2]
        db_UB = 2**-20 # Area [m**2]

        # Damage Initiation Behavior
        ufn_i = 2**-6.470108340925128 # Disp [m]
        ufn_LB = 2**-12 # Disp [m]
        ufn_UB = 2**-5 # Disp [m]

        ufs_i = 2**-6.470108340925128 # -11.169201472977056 # Disp [m]
        ufs_LB = 2**-20 # Disp [m]
        ufs_UB = 2**-5 # Disp [m]

        # Specify parameters
        fit_params = lf.Parameters() # intialize the class for parameters

        # Retina young's modulus
        if optimization.find('E_R') >= 0:
            fit_params.add('ER', value = ER_i, min=ER_LB, max=ER_UB, vary=True)

        # Vitreous Young's Modulus
        if optimization.find('E_V') >= 0:
            fit_params.add('EV', value = EV_i, min=EV_LB, max=EV_UB, vary=True)

        # parameter for making the retina stiffer than the vitreous
        if optimization.find('E_R') >= 0 and optimization.find('E_V') >= 0:
            fit_params.add('StiffDelta', value = 0.01, min=0, vary=True)
            # Constraint to allow vitreous to be not as stiff as the retina
            fit_params.add('stiffnessConstraint', expr = 'EV - StiffDelta')

```

```
        # FN
        if optimization.find('FN') >= 0:
            fit_params.add('FN', value = FN_i, min=FN_LB, max=FN_UB,
                           vary=True)

        # FS
        if optimization.find('FS') >= 0:
            fit_params.add('FS', value = FS_i, min=FS_LB, max=FS_UB,
                           vary=True)

        # db
        if optimization.find('db') >= 0:
            fit_params.add('db', value = db_i, min=db_LB, max=db_UB,
                           vary=True)

        # ufn
        if optimization.find('ufn') >= 0:
            fit_params.add('ufn', value = ufn_i, min=ufn_LB, max=ufn_UB,
                           vary=True)

        # ufs
        if optimization.find('ufs') >= 0:
            fit_params.add('ufs', value = ufs_i, min=ufs_LB, max=ufs_UB,
                           vary=True)

        # Set up minimization class
        minClass = lf.Minimizer(FEA_Residual, fit_params,
                                fcn_kws={'data': data},
                                max_nfev = maxFuncEval) # fcn_args=(x,),

        # (Different methods can be used here) Uses an array
        # out = minClass.leastsq() # Levenberg-Marquardt

        # single scalar value
        # out = minClass.scalar_minimize(method='Nelder-Mead', tol=tolVal)

        # single scalar value (if the objective function returns an array,
        # the sum of the squares of the array will be used (L2Norm))
        out = minClass.scalar_minimize(method='Cobyla', tol=tolVal)

        lf.report_fit(out) # modelpars=p_true,  show_correl=True

        # Write data to txt files
        writeOutputData(fileNameList)

    else:

        # Number of simulations to perform (Simulation Batch Total)
        SBT = []
        SBT.append(len(name))
        SBT.append(len(e1SeedArray))
        SBT.append(len(e2SeedArray))
        SBT.append(len(ptSeedArray))
        SBT.append(len(gSeedArray))
        SBT.append(len(v1SeedArray))
        SBT.append(len(v2SeedArray))
        SBT.append(len(rSeedArray))
        SBT.append(len(massScaleFactorArray))
```

```
        SBT.append(len(massScaleTimeArray))
        SBT.append(len(FNArray))
        SBT.append(len(FSArray))
        SBT.append(len(dbArray))
        SBT.append(len(ufnArray))
        SBT.append(len(ufsArray))

        ZipArray = []
        ZipArray.append(max(SBT)*name)
        ZipArray.append(max(SBT)*e1SeedArray)
        ZipArray.append(max(SBT)*e2SeedArray)
        ZipArray.append(max(SBT)*ptSeedArray)
        ZipArray.append(max(SBT)*gSeedArray)
        ZipArray.append(max(SBT)*v1SeedArray)
        ZipArray.append(max(SBT)*v2SeedArray)
        ZipArray.append(max(SBT)*rSeedArray)
        ZipArray.append(max(SBT)*massScaleFactorArray)
        ZipArray.append(max(SBT)*massScaleTimeArray)
        ZipArray.append(max(SBT)*FNArray)
        ZipArray.append(max(SBT)*FSArray)
        ZipArray.append(max(SBT)*dbArray)
        ZipArray.append(max(SBT)*ufnArray)
        ZipArray.append(max(SBT)*ufsArray)

        # Iterate over the different combinations of parameters
        # If varying one parameter, then use iter.product(items in list...)
        # If varying multiple parameters, use zip*max(SBT)*items in list...)

        for (namei,
             e1Seedi,
             e2Seedi,
             ptSeedi,
             gSeedi,
             v1Seedi,
             v2Seedi,
             rSeedi,
             massScaleFactori,
             massScaleTimeIncrementi,
             FNi,
             FSi,
             dbi,
             ufni,
             ufsi) in zip(*ZipArray):

            tic() # Start time
            counter += 1
            print(counter, 'of ', max(*SBT))

            # set the i'th value to the initial value (Updated in
            # optimization algorithm)
            RetinaYoungsModulus_i = RetinaYoungsModulus_0
            VitreousYoungsModulus_i = VitreousYoungsModulus_0

            # Call the function
            # Runs jobs and saves file names
            funReturn = GenerateAbaqusModels()
            fileName =           funReturn[0]
            maxSlopeSimulated =  funReturn[1]
```

```
            maxForceSimulated =  funReturn[2]
            SSmeanSimulated =    funReturn[3]
            SSmedianSimulated =  funReturn[4]
            y_new_exp_disp =     funReturn[5]
            y_new_sim_disp =     funReturn[6]

            # Error calculation
            errorDict = {} # Dictionary
            if objErr == 'Difference':
                errorDict['slope']    = (maxSlopeMeasured - maxSlopeSimulated) if slopeFlag == True else []
                errorDict['maxForce'] = (maxForceMeasured - maxForceSimulated) if maxForceFlag == True else []
                errorDict['ssForce']  = (SS_Measured - SSmeanSimulated)        if ssForceFlag == True else []
            elif objErr == 'Ratio':
                errorDict['slope']    = (1 - maxSlopeMeasured / maxSlopeSimulated) if slopeFlag == True else []
                errorDict['maxForce'] = (1 - maxForceMeasured / maxForceSimulated) if maxForceFlag == True else []
                errorDict['ssForce']  = (1 - SS_Measured / SSmeanSimulated)        if ssForceFlag == True else []
            elif objErr == 'Relative uncertainty':
                errorDict['slope']    = ((maxSlopeMeasured - maxSlopeSimulated)/maxSlopeMeasured) if slopeFlag == True else []
                errorDict['maxForce'] = ((maxForceMeasured - maxForceSimulated)/maxForceMeasured) if maxForceFlag == True else []
                errorDict['ssForce']  = ((SS_Measured - SSmedianSimulated)/SS_Measured)           if ssForceFlag == True else []
            else:
                print('Error in MaxForceError')
                sys.exit()

            # Error array values
            errorList = list(errorDict.values()) # convert to list
            errorList = [x for x in errorList if x] # get rid of empty values

            # Error of the simulation
            L2Normi = np.sqrt(np.dot(errorList, errorList))

            # Calculate residual
            residual = y_new_exp_disp - y_new_sim_disp # residual

            # Calculate L2Norm
            L2Norm = np.sqrt(np.dot(residual, residual))

            # multipliers to be appended to the output file to show changes
            # in parameters
            aAM = [] # attributeArrayMultipliar
            aAM.append(time)
            aAM.append(e1Seedi[0])
            aAM.append(e2Seedi[0])
            aAM.append(ptSeedi[0])
            aAM.append(gSeedi[0])
            aAM.append(v1Seedi[0])
            aAM.append(v2Seedi[0])
```

```python
            aAM.append(rSeedi[0])
            aAM.append(massScaleFactori[0])
            aAM.append(massScaleTimeIncrementi[0])
            aAM.append(RetinaYoungsModulus_i)
            aAM.append(VitreousYoungsModulus_i)
            aAM.append(BondStatus)
            aAM.append(FNi[0])
            aAM.append(FSi[0])
            aAM.append(PDFMStatus)
            aAM.append(dbi[0])
            aAM.append(ufni[0])
            aAM.append(ufsi[0])
            aAM.append(optimization)
            aAM.append(tieInterface)
            aAM.append(L2Normi)
            aAM.append(objErr)
            aAM.append(maxSlopeSimulated)
            aAM.append(maxForceSimulated)
            aAM.append(SSmedianSimulated)

            simulationTime = toc() # Determine run time
            # apends the fileName & File Attributes
            fileNameList.append([fileName, aAM,
                                 simulationTime])
            print('{} Error calculation: '.format(objErr), errorList)
            print('L2 norm objective calculation', L2Normi)
            print('L2 Norm residual', L2Norm)
            print('Done')

        # Write data to txt files
        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.* </>

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Jan 19 15:07:50 2021

@author: Kiffer2
"""

import numpy as np
import pandas as pd
import os
```

```
import sys
import matplotlib.pyplot as plt
from matplotlib.pyplot import cm
import matplotlib.patheffects as pe
from matplotlib.patches import Polygon
plt.rcParams['figure.figsize'] = [16, 9]
from scipy import interpolate
import pdb


# # Define the location of the Abaqus Working Directory
# # specific folder path where this file is located
# pythonScriptPath = os.getcwd()
# abqWD, pythonFiles = os.path.split(pythonScriptPath) # split file path

# filePath = os.getcwd() # current working directory
# codePath, pythonFolder = os.path.split(filePath) # split file path
# HWPath, codesFolder = os.path.split(codePath) # split file path

# expDataPath = 'experimentalData' # folder of data files
# dataPath = os.path.join(HWPath, expDataPath) # Path to data files

def Least_Squares(x,y):
    """
    Calculate the slope and y-intercept using matrix math
    x & y are the coordinates of points

    parameters (X,Y) Data

    Returns:
        Curve fit data and parameters m*x + b, R squared value
    """
    Z = np.ones((len(x),2))
    Z[:,1] = x
    # Calculate the matrix inverse for the constants of the regression
    A = np.dot(np.linalg.inv(np.dot(Z.T,Z)),(np.dot(Z.T,y)))
    linFit = x*A[1] + A[0]

    # Stats
    SS_tot = np.sum((y - np.mean(y))**2)
    SS_res = np.sum((y - linFit)**2)
    Rsqd = 1 - SS_res/SS_tot

    return linFit, A, Rsqd

def myformat(x):
    myexp = int(np.floor(np.log10(x)))
    xout = x*10**(-myexp)
    strout = '{:.4f}'.format(xout) + '\cdot10^{' + '{}'.format(myexp) + '}'
    return strout


# In[previous data]

def ReadRAWDataTrace(dataPath, abqWD, timeBeforePeak):
    """
    Inputs: dataPath - file path to raw data
    abqWD: abaqus working directory
```

```
    timeBeforePeak: number of seconds prior to the peak where data will
                    be extrapolated to the origin for curve fitting
    """

    timeBeforePeak = timeBeforePeak*10 # Convert s --> cs (10 data points/sec)

    # Eliminate the file extension
    dataPathNoExt = dataPath.split('.txt')[0]

    # Determine the specific file name
    fileDir, dataCompare = os.path.split(dataPathNoExt)

    """ Read in the csv file """
    dfValsn = pd.read_csv(dataPath, sep="\t", nrows=29, header=None,
                          names=['Var', 'Attribute'])

    """ File Attributes """
    HID =             dfValsn['Attribute'][0]
    HAGE =            dfValsn['Attribute'][1]
    HG =              dfValsn['Attribute'][2]
    HLR =             dfValsn['Attribute'][3]
    HR =              dfValsn['Attribute'][4]
    HSSi =      float(dfValsn['Attribute'][12])
    HSSf =      float(dfValsn['Attribute'][13])
    HTMax =     float(dfValsn['Attribute'][14])
    HDispMax = float(dfValsn['Attribute'][15])
    HFMax =     float(dfValsn['Attribute'][16]) # (mN)
    HFSS =      float(dfValsn['Attribute'][17])
    # slope from 20 seconds prior to max force value
    HSlope20 = float(dfValsn['Attribute'][20]) # (mN/m)

    dfn = pd.read_csv(dataPath, sep="\t", header=30)
    dfn.columns = ['Time', 'Extension', 'Force']
    dfn_time = dfn.Time
    dfn_extension = dfn.Extension # mm
    dfn_force = dfn.Force*1e3 # N ---> mN

    # SS Array
    ssTimeArray = np.array([HSSi, HSSf])
    ssValArray = np.array([HFSS, HFSS])

    # slope calculation for 20 seconds prior to the max peel force
    # (Experimental Data)
    maxIndex = dfn_time[dfn_time == HTMax].index.values[0]

    # to location of max force
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
    t_n = dfn_extension[maxIndex - timeBeforePeak:maxIndex]
    # to location of max force
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
    y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
    # Perform least squares and return
    curveFit_n, Params_n, R_Squared_n = Least_Squares(t_n,y)

    # Shift extension data so that the linear region is extrapolated through
    # the origin
    shift = abs(Params_n[0]/Params_n[1])*0
    dfn_extension = dfn_extension - shift
```

```python

    # Now that the data has been shifted, recalculate the linear regression
    # using the reduced data set

    # to location of max force
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
    t_n = dfn_extension[maxIndex - timeBeforePeak:maxIndex]
    # to location of max force
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
    y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
    # Perform least squares and return
    curveFit_n, Params_n, R_Squared_n = Least_Squares(t_n,y)

    # # Slope of the curve up to the max force !!!(from the simulated data)!!!
    # adjustDisp = min(dn, key=lambda x:abs(x - dfn_extension[maxIndex]))
    # index = RF[dn == adjustDisp].index.values[0]
    # simulationCriteria = index # Time before peak force for curve fitting
    # # Array from 0 to location of max force
    # x = dn[index - simulationCriteria:index]
    # # Array from 0 to location of max force
    # y = RF[index - simulationCriteria:index]
    # # Perform least squares
    # curveFit, Params, R_Squared = Least_Squares(x,y)

    # # Updated force at specific max disp with adjusted value (Simulated data)
    # specificTime = maxForceTime
    # actualDisp = min(dn, key=lambda x:abs(x - dfn_extension[maxIndex]))
    # force_at_Disp = RF[dn == actualDisp].values[0]

    # # Simulated max force
    # simMaxForce = RF.max() # maximum simulated force value
    # simMaxDisp = dn[RF == simMaxForce] # displacement at the max force value

    # Max peel force displacement at max and steady state
    dfn_max_Disp = dfn_extension[dfn_time == HTMax]
    # Didn't seem to work here
    # dfn_ss_Disp = np.array([dfn_extension[dfn_time == HSSi],
    #                         dfn_extension[dfn_time == HSSf]]).flatten()
    dfn_ss_Disp = [dfn_extension[dfn_time == HSSi].values[0],
                   dfn_extension[dfn_time == HSSf].values[0]]

    # In[Simulated Trace]

    # dataDirectory = 'D:\Downloads\experimentalData'

    # fileName = ('output_Field_S25CohesiveXLVitDiff_CT250S11' +
    #             'SF0MS7RE1e_04VE5e_02opt.txt')

    # df = pd.read_csv(os.path.join(dataDirectory, fileName),
    #                  sep="\t", header=0)

    # Header = [] # Header information for the dataframe
    # Header.append('Frame') #                   h1
    # Header.append('Time') #                    h2
    # Header.append('RF_y_dot') #                h3
    # Header.append('RFx') #                     h4
    # Header.append('RFy') #                     h5
    # Header.append('RFz') #                     h6
```

```python
    # Header.append('Nodal_Force') #             h7
    # Header.append('Tab_Displacement') #        h8
    # Header.append('Bond_Displacement') #       h9
    # Header.append('Stress') #                  h10
    # Header.append('AVG_CSMAXSCRT') #           h11
    # Header.append('AVG_CSDMG') #               h12
    # df.columns = Header

    # tt = df.Time
    # RF = df.RF_y_dot*1000 # N to mN
    # dn = df.Tab_Displacement*1000 # m

    # In[Plots]

    """ Plots """
    # Plot the data trace to compare the simulated results with the force
    # displacement curves
    fig, ax = plt.subplots()
    ax.plot(dfn_extension, dfn_force,'-', color='r', linewidth=1,
            markersize=2, label = '{}, Age: {}'.format(HID, HAGE),
            alpha = 0.5)

    if str(HFMax) == 'nan' and str(HSSi) == 'nan':
        print('No max or steady state')
        pass

    if str(HFMax) != 'nan':
        ax.plot(dfn_max_Disp, HFMax,'.', color='k', linewidth=1,
                markersize=20,
                label = 'Max Peel - {:.4f} (mN)'.format(HFMax),
                path_effects=[pe.Stroke(linewidth=4, foreground='k'),
                              pe.Normal()])
        ax.plot(t_n, curveFit_n, '-', color='tab:blue', linewidth=2,
                label=r'Curve fit Max - {}'.format(int(timeBeforePeak/10)) +
                ' (s) y = {:.4f}x '.format(Params_n[1]) +
                '+ {:.4f} (mN), '.format(Params_n[0]) +
                '$r^2$ = {:.4f}'.format(R_Squared_n),
                alpha = 1)

    if str(HSSi) != 'nan':
        ax.plot(dfn_ss_Disp, ssValArray,'-', color='c', linewidth=3,
                markersize=2,
                label = 'Steady State - {:.4f} (mN)'.format(HFSS),
                path_effects=[pe.Stroke(linewidth=5,
                                        foreground='k'),
                              pe.Normal()])

    # Make the shaded region for the entire integral
    a = dfn_max_Disp.values[0] # dfn_ss_Disp[0]
    b = dfn_ss_Disp[0] # dfn_ss_Disp[1]

    # Make the shaded region include the square below
    adjust = 0 # 0 or 1 to get rid of the small square

    # Filter the data in between the bounds
    dfn_ext_adjust = dfn_extension[(dfn_extension >= a) & (dfn_extension < b)]
    dnf_force_adjust = dfn_force[(dfn_extension >= a) & (dfn_extension < b)]
```

```
    verts = [(a, HFSS*adjust),
             *zip(dfn_ext_adjust, dnf_force_adjust),
             (b, HFSS*adjust)]
    poly = Polygon(verts, facecolor='0.8', edgecolor='0.5')
    ax.add_patch(poly)

    # Integral area
    Integral = np.trapz(dnf_force_adjust - HFSS*adjust, dfn_ext_adjust)

    # Centroid for plotting
    CentroidX = 1/Integral*(np.trapz(dfn_ext_adjust*(dnf_force_adjust -
                                                      HFSS*adjust),
                                     dfn_ext_adjust))
    CentroidY = 1/Integral*(np.trapz((dnf_force_adjust**2 -
                                      (HFSS*adjust)**2*adjust)/2,
                                     dfn_ext_adjust))

    # ax.text(b, (HFMax + HFSS)/2, r'$\int_a^b f(x)\mathrm{d}x=' +
    #         myformat(Integral*1e-6) + '$ (J)', horizontalalignment='center',
    #         fontsize=20)
    # ax.plot([0.5*max(dfn_extension), CentroidX], [0.5*max(dfn_force),
    #                                               CentroidY])

    prop = dict(arrowstyle="-|>,head_width=0.4, head_length=0.8", shrinkA=0,
                shrinkB=0)
    # ax.arrow(0.5*max(dfn_extension), 0.5*max(dfn_force),
    #          CentroidX - 0.5*max(dfn_extension),
    #          CentroidY - 0.5*max(dfn_force),
    #          head_width=0.1, head_length=0.1)
    ax.annotate("", xy=(CentroidX, CentroidY), xytext=(0.5*max(dfn_extension),
                                                       0.5*max(dfn_force)),
                arrowprops=prop)

    ax.text(0.5*max(dfn_extension), 0.52*max(dfn_force),
            r'$\int_a^b f(x)\mathrm{d}x=' + myformat(Integral*1e-6) + '$ (J)',
            horizontalalignment='center', fontsize=20)

    ax.spines['right'].set_visible(False)
    ax.spines['top'].set_visible(False)
    ax.xaxis.set_ticks_position('bottom')

    ax.set_xticks((a, b))
    ax.set_xticklabels(('${}$'.format(a), '${}$'.format(b)))
    ax.set_yticks((HFSS, HFMax))
    ax.set_yticklabels(('${:.5}$'.format(HFSS), '${:.5}$'.format(HFMax)))

    ################# Plot Data ########################
    plt.axhline(0,color='black') # x = 0
    plt.axvline(0,color='black') # y = 0
    plt.ylabel('Force (mN)',fontsize=18)
    plt.xlabel('Displacement (mm)',fontsize=18)
    plt.title('Vitreous',fontsize=20)
    plt.grid()
    plt.legend(loc = 'best',fontsize = 'medium')
    plt.savefig(os.path.join(abqWD, 'GcSelection.pdf'), dpi=300,
                bbox_inches='tight')
    # plt.show()
    plt.close()
```

```python



    # """ Derivative of the data trace """
    # fig, ax = plt.subplots()

    # deriv = np.gradient(dfn_force, dfn_extension)

    # ax.plot(dfn_extension, deriv)
    # ax.set_ylim(-100, 100) # maxRFList
    # plt.show()

    # In[Time plot]

    # slope calculation for n seconds prior to the max peel force
    # (Experimental Data)
    maxIndex = dfn_time[dfn_time == HTMax].index.values[0]

    # to location of max force
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
    t_n = dfn_time[maxIndex - timeBeforePeak:maxIndex]
    y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
    # Perform least squares and return
    curveFit_n, Params_n, R_Squared_n = Least_Squares(t_n, y)

    # Shift extension data so that the linear region is extrapolated
    # through the origin
    shift_time = abs(Params_n[0]/Params_n[1])*1
    if Params_n[0] > 0:
        # shift time data for visual purposes
        dfn_time_shift = dfn_time + shift_time
        dfn_ss_time_shift = ssTimeArray + shift_time
        HTMax_shift = HTMax + shift_time
    else:
        # shift time data for visual purposes
        dfn_time_shift = dfn_time - shift_time
        dfn_ss_time_shift = ssTimeArray - shift_time
        HTMax_shift = HTMax - shift_time


    # Curve fit the shifted displacement
    maxIndex = dfn_time[dfn_time == HTMax].index.values[0]

    # to location of max force
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
    t_n = dfn_time_shift[maxIndex - timeBeforePeak:maxIndex]
    y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
    # Perform least squares and return
    curveFit_n, Params_n, R_Squared_n = Least_Squares(t_n, y)

    # to location of max force
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
    x_n = dfn_extension[maxIndex - timeBeforePeak:maxIndex]
    # Perform least squares
    curveFit_n_disp, Params_n_disp, R_Squared_n_disp = Least_Squares(x_n, y)

    # Shift extension data so that the linear region is extrapolated through
    # the origin
```

```
    shift_disp = abs(Params_n_disp[0]/Params_n_disp[1])*1
    if Params_n[0] > 0:
        dfn_extension_shift = dfn_extension + shift_disp
        dfn_ss_Disp_shift = dfn_ss_Disp + shift_disp
    else:
        dfn_extension_shift = dfn_extension - shift_disp
        dfn_ss_Disp_shift = dfn_ss_Disp - shift_disp

    # to location of max force
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
    x_n = dfn_extension_shift[maxIndex - timeBeforePeak:maxIndex]
    # Perform least squares
    curveFit_n_disp, Params_n_disp, R_Squared_n_disp = Least_Squares(x_n, y)


    Fmax_t_shift = dfn_time_shift[maxIndex]
    fit_t = np.linspace(0, Fmax_t_shift, 200) # Selected value

    # true max
    # fit_t = np.linspace(0, dfn_time_shift[np.argmax(dfn_force)], 200)
    Fmax_x_shift = dfn_extension_shift[maxIndex]

    # true max
    # fit_x = np.linspace(0, dfn_extension_shift[np.argmax(dfn_force)], 200)
    fit_x = np.linspace(0, Fmax_x_shift, 200) # Selected value

    def fit(params, x):
        b, m = params
        return m*x + b

    fit_vals_y_time = fit(Params_n, fit_t)
    fit_vals_y_force = fit(Params_n_disp, fit_x)

    ''' Reaction force vs. time shifted '''
    fig, ax = plt.subplots()
    ax.plot(dfn_time_shift, dfn_force,
            label=r'Data - {}'.format(dataCompare.split('.')[0]))
    ax.plot(fit_t, fit_vals_y_time, '--', label=r'Assumed linear region')
    ax.plot(Fmax_t_shift, dfn_force[maxIndex], 'o', markersize=10,
            label=r'Time at peak = {:.4} (s)'.format(max(fit_t)))

    ax.plot(dfn_ss_time_shift, ssValArray,'-', color='c', linewidth=3,
            markersize=2, label = 'Steady State - {:.4f} (mN)'.format(HFSS),
            path_effects=[pe.Stroke(linewidth=5, foreground='k'), pe.Normal()])

    ax.plot([], [], 'w',
            label='Start SS time = {:.4f} (s)'.format(min(dfn_ss_time_shift)))
    ax.plot([], [], 'w',
            label='End SS time = {:.4f} (s)'.format(max(dfn_ss_time_shift)))

    plt.axhline(0,color='black')
    plt.axvline(0,color='black')

    plt.ylabel('Force (mN)',fontsize=18)
    plt.xlabel('Time from extrapolated zero (s)',fontsize=18)
    plt.legend(loc='best')
    # plt.xlim([0, max(dfn_time_shift)])
    plt.savefig(os.path.join(abqWD, 'SimulationTime.pdf'), dpi=300,
```

```
                bbox_inches='tight')
    # plt.show()
    plt.close()

    ''' Reaction force vs. displacement shifted '''
    fig, ax = plt.subplots()
    ax.plot(dfn_extension_shift, dfn_force,
            label=r'Data - {}'.format(dataCompare.split('.')[0]))
    ax.plot(fit_x, fit_vals_y_force, '--', label=r'Assumed linear region')
    ax.plot(Fmax_x_shift, dfn_force[maxIndex], 'o', markersize=10,
            label=r'Time at peak = {:.4} (s)'.format(max(fit_t)))

    ax.plot(dfn_ss_Disp_shift, ssValArray,'-', color='c', linewidth=3,
            markersize=2, label = 'Steady State - {:.4f} (mN)'.format(HFSS),
            path_effects=[pe.Stroke(linewidth=5, foreground='k'), pe.Normal()])

    ax.plot([], [], 'w',
            label='Start SS time = {:.4f} (s)'.format(min(dfn_ss_time_shift)))
    ax.plot([], [], 'w',
            label='End SS time = {:.4f} (s)'.format(max(dfn_ss_time_shift)))

    plt.axhline(0, color='black')
    plt.axvline(0, color='black')

    plt.ylabel('Force (mN)',fontsize=18)
    plt.xlabel('Displacement (mm)',fontsize=18)
    plt.legend(loc='best')
    # plt.xlim([0, max(dfn_time_shift)])
    plt.savefig(os.path.join(abqWD, 'SimulationDisp.pdf'), dpi=300,
                bbox_inches='tight')
    # plt.show()
    plt.close()

    # In[Interpolated Experimental Data]

    # create array from 0 max peel force (linear equation fit from above)
    # populate a pandas dataframe
    # merge the data frame with the data above from the peak force to the end
    # use the interp1d fcn to interpolate between data
    # pass the simulated data into the interpolation

    # Time greater than the shift intersection point
    t_exp = dfn_time_shift[dfn_time_shift >= 0]
    x_exp = dfn_extension_shift[dfn_time_shift >= 0]
    y_exp = dfn_force[dfn_time_shift >= 0]

    # data frame with original data
    dfdata = pd.DataFrame(np.array([t_exp, x_exp, y_exp]).T,
                          columns=['t', 'x', 'y'])

    # Select time beyond the max time to the end of the data
    t_geq_max = dfn_time_shift[maxIndex:]
    x_geq_max = dfn_extension_shift[maxIndex:]
    y_geq_max = dfn_force[maxIndex:]

    # dataframe of data points from the max value to the end
    dfgmax = pd.DataFrame(np.array([t_geq_max, x_geq_max, y_geq_max]).T,
                          columns=['t', 'x', 'y'])
```

```
    # data frame of points from zero to the max value
    linArray = np.array([fit_t, fit_x, fit_vals_y_time])
    dfLin = pd.DataFrame(linArray.T, columns=['t', 'x', 'y'])

    # create the new data frame of linear points up to the peak and all points
    # beyond
    dfNew = dfLin.append(dfgmax, ignore_index=True)

    # # Interpolate the experimental data
    # n_data_pts = 100
    # Time at the peak (shifted)
    # start_point_time = tt[RF.argmax()]# - shift
    # Disp at the peak (shifted)
    # start_point_disp = dn[RF.argmax()]# - shift_disp
    # f_exp_time = interpolate.interp1d(dfNew['t'], dfNew['y'])
    # f_exp_disp = interpolate.interp1d(dfNew['x'], dfNew['y'])
    # t_new_exp = np.linspace(start_point_time, tt[tt.argmax()],
    #                         n_data_pts) # (s)
    # x_new_exp = np.linspace(start_point_disp, dn[tt.argmax()],
    #                         n_data_pts) # (mm)
    # y_new_exp_time = f_exp_time(t_new_exp) # Interpolate `interp1d`
    # y_new_exp_disp = f_exp_disp(x_new_exp) # Interpolate `interp1d`

    # In[Interpolated Simulated Trace]

    # # Interpolate the simulated data
    # f_sim_time = interpolate.interp1d(tt, RF)
    # f_sim_disp = interpolate.interp1d(dn, RF)
    # t_new_sim = np.linspace(start_point_time, tt[tt.argmax()],
    #                         n_data_pts) # (s)
    # x_new_sim = np.linspace(start_point_disp, dn[tt.argmax()],
    #                         n_data_pts) # (mm)
    # y_new_sim_time = f_sim_time(t_new_sim) # Interpolate `interp1d`
    # y_new_sim_disp = f_sim_disp(x_new_sim) # Interpolate `interp1d`

    # In[Plots]
    # ''' Time curve '''
    # fit, ax = plt.subplots()
    # ax.plot()
    # ax.plot(dfdata['t'], dfdata['y'], label='Original Shifted Data',
    #         alpha = 0.5)
    # ax.plot(dfNew['t'], dfNew['y'], label='Merged Data',
    #         alpha = 0.5)
    # ax.plot(t_new_exp, y_new_exp_time, '--',
    #         label='Interp Experimental Data')
    # ax.plot(tt, RF, label='Simulated Data')
    # ax.plot(t_new_sim, y_new_sim_time, ':', label='Interp Simulated Data')
    # ax.set_xlim([0, 300])
    # ax.set_xlabel('Time (s)', fontsize=14)
    # ax.set_ylabel('Force (N)', fontsize=14)
    # ax.legend(loc='best', fontsize=14)
    # ax.grid('on')
    # plt.savefig(os.path.join(abqWD, 'interp1d_Time.pdf'), dpi=300,
    #             bbox_inches='tight')
    # plt.show()

    # ''' Displacement curve '''
```

```python
    # fit, ax = plt.subplots()
    # ax.plot()
    # ax.plot(dfdata['x'], dfdata['y'], label='Original Shifted Data',
    #         alpha = 0.5)
    # ax.plot(dfNew['x'], dfNew['y'], label='Merged Data',
    #         alpha = 0.5)
    # ax.plot(x_new_exp, y_new_exp_disp, '--',
    #         label='Interp Experimental Data')
    # ax.plot(dn, RF, label='Simulated Data')
    # ax.plot(x_new_sim, y_new_sim_disp, ':', label='Interp Simulated Data')
    # ax.set_xlim([0, max(dn)])
    # ax.set_xlabel('Displacement (mm)', fontsize=14)
    # ax.set_ylabel('Force (N)', fontsize=14)
    # ax.legend(loc='best', fontsize=14)
    # ax.grid('on')
    # plt.savefig(os.path.join(abqWD, 'interp1d_Disp.pdf'), dpi=300,
    #             bbox_inches='tight')
    # plt.show()

    # ''' Displacement curve only showing interpolated data '''
    # abs residual calculation
    # residual = abs(y_new_exp_disp - y_new_sim_disp)
    # L2Norm = np.dot(residual, residual)

    # fit, ax = plt.subplots()
    # ax.plot()
    # ax.plot(x_new_exp, y_new_exp_disp, '-', label='Interp Experimental Data')
    # ax.plot(x_new_sim, y_new_sim_disp, '-', label='Interp Simulated Data')
    # ax.plot(x_new_sim, residual, ':',
    #         label=r'Residual = $\|\| exp - sim \|\|$', alpha = 0.8)
    # ax.plot([], [], color='white',
    #         label=r'$L^2$ norm = {:.4f}'.format(L2Norm))
    # ax.axhline(color='k', linewidth=0.25)
    # ax.set_xlim([0, max(x_new_exp)])
    # ax.set_xlabel('Displacement (mm)', fontsize=14)
    # ax.set_ylabel('Force (N)', fontsize=14)
    # ax.legend(loc='best', fontsize=14)
    # ax.grid('on')
    # plt.savefig(os.path.join(abqWD, 'interp1d_Disp_clean.pdf'), dpi=300,
    #             bbox_inches='tight')
    # plt.show()

    print('Output files have been printed to determine the appropriate ' +
          'parameters for the simulation')

    returnArray = [max(fit_t), max(dfn_ss_time_shift), HTMax_shift,
                   Integral*1e-6]
    return returnArray

if __name__ == '__main__':
    # Run the function

    # fileName = sys.argv[-2]
    # savePath = sys.argv[-1]

    ReadRAWDataTrace(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.* </>

```
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 18 22:12:05 2020

@author: Kiffer2
"""

"""  abaqus cae -noGUI abaqusMacros.py """

# -*- coding: mbcs -*-
# Do not delete the following import lines
from abaqus import *
from abaqusConstants import *
import __main__

import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import optimization
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
import numpy as np
import os
import sys

# location of the folder
# specific folder path where this file is located # os.getcwd()
pythonScriptPath = os.path.abspath("file")
abqWD, pythonFiles = os.path.split(pythonScriptPath) # split file path

# StepFile = 'Adult Human Eye holder Assembly.STEP'
# StepFile = 'Adult Human Eye holder Assembly 2 Step.STEP'
# Constrained
# StepFile = 'Adult Human Eye holder Assembly Constrained Bottom.STEP'

# Trimmed to prevent element distortion on low elastic modulus curve fits
StepFile = ('Adult Human Eye holder Assembly Constrained Bottom Trimmed ' +
            'Retina.STEP')

SolidWorksDir = 'SolidWorksStepFiles' # Folder name

```

```
# Combine folder directory
SolidWorksStepFile = os.path.join(SolidWorksDir, StepFile)

# In[Non-symmetric model]

def ImportStepEyeConstrained():
    """ Use with the constrained bottom STEP file"""
    step = mdb.openStep(os.path.join(abqWD, SolidWorksStepFile),
                        scaleFromFile=OFF)

    abqModel.PartFromGeometryFile(name='V', geometryFile=step, bodyNum=1,
                                  combine=False, dimensionality=THREE_D,
                                  type=DEFORMABLE_BODY)
    abqModel.PartFromGeometryFile(name='E', geometryFile=step, bodyNum=2,
                                  combine=False, dimensionality=THREE_D,
                                  type=DISCRETE_RIGID_SURFACE)
    abqModel.PartFromGeometryFile(name='R', geometryFile=step, bodyNum=3,
                                  combine=False, dimensionality=THREE_D,
                                  type=DEFORMABLE_BODY)
    abqModel.PartFromGeometryFile(name='T', geometryFile=step, bodyNum=4,
                                  combine=False, dimensionality=THREE_D,
                                  type=DISCRETE_RIGID_SURFACE)
    abqModel.PartFromGeometryFile(name='G', geometryFile=step, bodyNum=5,
                                  combine=False, dimensionality=THREE_D,
                                  type=DISCRETE_RIGID_SURFACE)


def Retina_Mat_Prop(RetinaProp):
    retina_E = RetinaProp # Passed in young's modulus
    Retina_Description = """
Actually used the value from Chen 2014
E = 11.12 KPa

------------------------------------------------
Density (kg/m^3)
1100 --------> Esposito_2013

"""
    abqModel.Material(name='Retina', description=Retina_Description)
    abqModel.materials['Retina'].Density(table=((1100.0, ), ))
    abqModel.materials['Retina'].Elastic(table=((retina_E, 0.49), ))

    # Assign the section to the part
    abqModel.HomogeneousSolidSection(name='Retina_Section', material='Retina',
                                     thickness=None)

def Vitreous_Mat_Prop(vitreousProp):
    vitreous_E = vitreousProp # Passed in young's modulus
    Vitreous_Description = """
------------------------------------------------
Density (kg/m^3)
950   ------------> Esposito_2013

------------------------------------------------

# Tram 2018 Viscoelasticity data
# 4 Term Prony (Tram Data # 5 HU2018-0074 OD 1 Pa)
(0.1486397420159951, 0.0, 331.4796231072498),
```

```python
(0.12469207412616717, 0.0, 3.388868494747128),
(0.29059507092540404, 0.0, 15.59692349525066),
(0.1591569334281, 0.0, 69.85134248442381)
"""
    abqModel.Material(name='Vitreous', description=Vitreous_Description)
    abqModel.materials['Vitreous'].Density(table=((950.0, ), ))
    ''' Using Lin2020 Paper to relate SLSM curve fit parameters to physical
    values.  Prony 4 Term (Long term) initial guess 172.77874855377468
    optimization of E'''
    abqModel.materials['Vitreous'].Elastic(moduli=LONG_TERM,
                                            table=((vitreous_E, 0.49), ))
    # Prony 4 Term calculated from normalized data
    abqModel.materials['Vitreous'].Viscoelastic(
            domain=TIME, time=PRONY, table=(
            # Tram Data # 5
            (0.1486397420159951, 0.0, 331.4796231072498),
            (0.12469207412616717, 0.0, 3.388868494747128),
            (0.29059507092540404, 0.0, 15.59692349525066),
            (0.1591569334281, 0.0, 69.85134248442381)))

    # Assign the section to the part
    abqModel.HomogeneousSolidSection(name='Vitreous_Section',
                                      material='Vitreous', thickness=None)

def E_Features():
    ''' Eye holder features '''
    p = abqModel.parts['E']

    # Remove shell
    c = p.cells
    p.RemoveCells(cellList = c[0:1])

    # Reference point
    p.ReferencePoint(point=(0.0, 0.0, 0.0))

    # Add E-set to the reference point
    r = p.referencePoints
    refPoints=(r[3], )
    p.Set(referencePoints=refPoints, name='E_RP_Set')

    # Edge seed sets
    e = p.edges
    edges = e.getSequenceFromMask(mask=('[#400f000 #1402 ]', ), )
    p.Set(edges=edges, name='E_Edge_Seed_Set')

    edges = e.getSequenceFromMask(mask=('[#f1ff0fff #2838 ]', ), )
    p.Set(edges=edges, name='E_Outside_Edge_Seed_Set')

    # Surfaces
    s = p.faces
    side1Faces = s.getSequenceFromMask(mask=('[#1ffff ]', ), )
    p.Surface(side1Faces=side1Faces, name='E_Surf')


def G_Features():
    ''' Glue features '''
    p = abqModel.parts['G']
    c = p.cells
```

```python

    # Remeove cells for rigid body
    p.RemoveCells(cellList = c[0:1])

    # Reference point
    p.ReferencePoint(point=(9.799E-03, 5.657E-03, 2.54E-03))

    # Define the reference point for the rigid body
    r = p.referencePoints
    refPoints=(r[3], )
    p.Set(referencePoints=refPoints, name='G_RP_Set')

    # # Create sets
    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#3f ]', ), )
    p.Set(faces=faces, name='G_Set')
    faces = f.getSequenceFromMask(mask=('[#20 ]', ), )
    p.Set(faces=faces, name='G_T_Set')
    faces = f.getSequenceFromMask(mask=('[#1 ]', ), )
    p.Set(faces=faces, name='G_R_Set')

    # Create surfaces
    s = p.faces
    side1Faces = s.getSequenceFromMask(mask=('[#3f ]', ), )
    p.Surface(side1Faces=side1Faces, name='G_Surf')
    side1Faces = s.getSequenceFromMask(mask=('[#20 ]', ), )
    p.Surface(side1Faces=side1Faces, name='G_T_Surf')
    side1Faces = s.getSequenceFromMask(mask=('[#1 ]', ), )
    p.Surface(side1Faces=side1Faces, name='G_R_Surf')


def T_Features():
    ''' Plastic Tab features '''
    p = abqModel.parts['T']
    c = p.cells

    # Remeove cells for rigid body
    p.RemoveCells(cellList = c[0:1])

    # Reference point
    p.ReferencePoint(point=(16.241E-03, 9.74E-03, 13.E-06))

    # Define the reference point for the rigid body
    r = p.referencePoints
    refPoints=(r[3], )
    p.Set(referencePoints=refPoints, name='T_RP_Set')

    # Create sets
    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#ff ]', ), )
    p.Set(faces=faces, name='T_Set')
    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#2 ]', ), )
    p.Set(faces=faces, name='T_G_Set')

    # Create surfaces
    s = p.faces
    side1Faces = s.getSequenceFromMask(mask=('[#ff ]', ), )
```

```python
    p.Surface(side1Faces=side1Faces, name='T_Surf')
    side1Faces = s.getSequenceFromMask(mask=('[#2 ]', ), )
    p.Surface(side1Faces=side1Faces, name='T_G_Surf')


def R_Features():
    ''' Retina features '''
    p = abqModel.parts['R']
    c = p.cells
    cells = c.getSequenceFromMask(mask=('[#1 ]', ), )
    p.Set(cells=cells, name='R_Set')

    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#3 ]', ), )
    p.Set(faces=faces, name='R_G_Set')

    faces = f.getSequenceFromMask(mask=('[#4 ]', ), )
    p.Set(faces=faces, name='R_V_Set')

    s = p.faces
    side1Faces = s.getSequenceFromMask(mask=('[#ff ]', ), )
    p.Surface(side1Faces=side1Faces, name='R_Surf')

    side1Faces = s.getSequenceFromMask(mask=('[#3 ]', ), )
    p.Surface(side1Faces=side1Faces, name='R_G_Surf')

    side1Faces = s.getSequenceFromMask(mask=('[#4 ]', ), )
    p.Surface(side1Faces=side1Faces, name='R_V_Surf_BOND')

    # Assign section
    region = p.sets['R_Set']
    p.SectionAssignment(region=region, sectionName='Retina_Section',
                        offset=0.0, offsetType=MIDDLE_SURFACE, offsetField='',
                        thicknessAssignment=FROM_SECTION)



def PartitionRetinaOnVitreous():
    ''' Vitreous features additional partitions for creating the surface for
    bonding'''
    p = abqModel.parts['V']

    # Partition V along the width of the retina
    p.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=-0.00254)
    abqModel.parts['V'].features.changeKey(fromName='Datum plane-1',
                                           toName='Retina_Width_Neg_Z')

    p.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=0.00254)
    abqModel.parts['V'].features.changeKey(fromName='Datum plane-1',
                                           toName='Retina_Width_Pos_Z')

    # Create a datum plnd along the z axis plane
    p.DatumAxisByPrincipalAxis(principalAxis=ZAXIS)
    p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset=0.0)

    # Create the rotated datum planes
    d = p.datums
    p.DatumPlaneByRotation(plane=d[5], axis=d[4], angle=18.75)
```

```python
    p.DatumPlaneByRotation(plane=d[5], axis=d[4], angle=-18.75)

    ''' Partition the surface of the retina on the vitreous '''
    p = abqModel.parts['V']
    c, d = p.cells, p.datums
    pickedCells = c.getSequenceFromMask(mask=('[#440 ]', ), )
    p.PartitionCellByDatumPlane(datumPlane=d[3], cells=pickedCells)
    pickedCells = c.getSequenceFromMask(mask=('[#408 ]', ), )
    p.PartitionCellByDatumPlane(datumPlane=d[2], cells=pickedCells)
    pickedCells = c.getSequenceFromMask(mask=('[#22 ]', ), )
    p.PartitionCellByDatumPlane(datumPlane=d[6], cells=pickedCells)
    pickedCells = c.getSequenceFromMask(mask=('[#140 ]', ), )
    p.PartitionCellByDatumPlane(datumPlane=d[7], cells=pickedCells)


def Vitreous_Features():
    ''' Assign specific features to the vitreous '''
    p = abqModel.parts['V']
    c, f, s = p.cells, p.faces, p.faces

    # Sets
    cells = c.getSequenceFromMask(mask=('[#ffffff ]', ), )
    p.Set(cells=cells, name='V_Set')
    faces = f.getSequenceFromMask(mask=('[#5090 ]', ), )
    p.Set(faces=faces, name='V_R_Set')

    # Surfaces
    side1Faces = s.getSequenceFromMask(mask=('[#1805090 #3 #ff0 ]', ), )
    p.Surface(side1Faces=side1Faces, name='V_Surf')
    side1Faces = s.getSequenceFromMask(mask=('[#5090 ]', ), )
    p.Surface(side1Faces=side1Faces, name='V_R_Surf_BOND')

    # Assign the section to the part
    region = p.sets['V_Set']
    p.SectionAssignment(region=region,
                        sectionName='Vitreous_Section',
                        offset=0.0,
                        offsetType=MIDDLE_SURFACE,
                        offsetField='',
                        thicknessAssignment=FROM_SECTION)


def V_Partition_XYZ_Axis():
    ''' Partition the sphere along the x, y, z axis '''
    p = abqModel.parts['V']
    c, v, e, d = p.cells, p.vertices, p.edges, p.datums
    pickedCells = c.getSequenceFromMask(mask=('[#1 ]', ), )
    p.PartitionCellByPlaneThreePoints(point1=v[1],
                                      point2=v[0],
                                      point3=v[3],
                                      cells=pickedCells)

    pickedCells = c.getSequenceFromMask(mask=('[#3 ]', ), )
    p.PartitionCellByPlaneThreePoints(point1=v[0],
                                      point2=v[4],
                                      point3=v[2],
                                      cells=pickedCells)

```

```
    pickedCells = c.getSequenceFromMask(mask=('[#f ]', ), )
    p.PartitionCellByPlaneThreePoints(point1=v[5],
                                      point2=v[2],
                                      point3=v[4],
                                      cells=pickedCells)


def V_Internal_Sphere():
    sphereRadius = 0.008 # radius of the internal sphere for meshing

    s1 = abqModel.ConstrainedSketch(name='__profile__', sheetSize=0.1)
    g, v, d, c1 = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
    s1.sketchOptions.setValues(decimalPlaces=3)
    s1.setPrimaryObject(option=STANDALONE)
    s1.ConstructionLine(point1=(0.0, -0.05), point2=(0.0, 0.05))
    s1.FixedConstraint(entity=g[2])
    s1.ArcByCenterEnds(center=(0.0, 0.0),
                       point1=(0.0, sphereRadius),
                       point2=(0.0, -sphereRadius),
                       direction=CLOCKWISE)
    s1.Line(point1=(0.0, sphereRadius),
            point2=(0.0, -sphereRadius))
    s1.VerticalConstraint(entity=g[4], addUndoState=False)
    s1.PerpendicularConstraint(entity1=g[3], entity2=g[4], addUndoState=False)
    p = abqModel.Part(name='V_internal',
                      dimensionality=THREE_D,
                      type=DEFORMABLE_BODY)
    p = abqModel.parts['V_internal']
    p.BaseSolidRevolve(sketch=s1, angle=360.0, flipRevolveDirection=OFF)
    s1.unsetPrimaryObject()
    p = abqModel.parts['V_internal']
    del abqModel.sketches['__profile__']


def mergeV():
    ''' Merge the internal sphere with the vitreous '''
    a = abqModel.rootAssembly
    a.InstanceFromBooleanMerge(name='V_Merge',
                               instances=(a.instances['V-1'],
                                          a.instances['V_internal-1'], ),
                               keepIntersections=ON,
                               originalInstances=DELETE,
                               domain=GEOMETRY)

    # Clean up file names after merge
    del abqModel.parts['V']
    del abqModel.parts['V_internal']

    abqModel.parts.changeKey(fromName='V_Merge', toName='V')
    a = abqModel.rootAssembly
    a.regenerate()
    abqModel.rootAssembly.features.changeKey(fromName='V_Merge-1',
                                             toName='V-1')

    a.regenerate()


def AssembleV_for_Merging():
```

```
    a1 = abqModel.rootAssembly
    a1.DatumCsysByDefault(CARTESIAN)
    p = abqModel.parts['V']
    a1.Instance(name='V-1', part=p, dependent=ON)
    p = abqModel.parts['V_internal']
    a1.Instance(name='V_internal-1', part=p, dependent=ON)


def E_Mesh(InsideSeed, OutsideSeed):
    p = abqModel.parts['E']
    e = p.edges
    pickedEdges = e.getSequenceFromMask(mask=('[#400f000 #1402 ]', ), )
    p.seedEdgeBySize(edges=pickedEdges,
                     size=0.0005,
                     deviationFactor=0.1,
                     minSizeFactor=0.1,
                     constraint=FINER)
    pickedEdges = e.getSequenceFromMask(mask=('[#f1ff0fff #2838 ]', ), )
    p.seedEdgeBySize(edges=pickedEdges,
                     size=0.00342673,
                     deviationFactor=0.1,
                     minSizeFactor=0.1,
                     constraint=FINER)
    # (unique node numbering)
    p.setValues(startNodeLabel=1000000, startElemLabel=1000000)
    p.generateMesh()


def G_Mesh(seed):
    p = abqModel.parts['G']
    p.seedPart(size=seed, deviationFactor=0.1, minSizeFactor=0.1)
    f = p.faces
    pickedRegions = f.getSequenceFromMask(mask=('[#3f ]', ), )
    p.setMeshControls(regions=pickedRegions, elemShape=QUAD)
    elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=EXPLICIT)
    elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=EXPLICIT)
    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#3f ]', ), )
    pickedRegions =(faces, )
    p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
    # (unique node numbering)
    p.setValues(startNodeLabel=2000000, startElemLabel=2000000)
    p.generateMesh()


def T_Mesh(seed):
    p = abqModel.parts['T']
    p.seedPart(size=seed, deviationFactor=0.1, minSizeFactor=0.1)
    f = p.faces
    pickedRegions = f.getSequenceFromMask(mask=('[#ff ]', ), )
    p.setMeshControls(regions=pickedRegions, elemShape=QUAD)
    elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=EXPLICIT)
    elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=EXPLICIT)
    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#ff ]', ), )
    pickedRegions =(faces, )
    p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
    # (unique node numbering)
```

```
    p.setValues(startNodeLabel=3000000, startElemLabel=3000000)
    p.generateMesh()


def R_Mesh(seed):
    p = abqModel.parts['R']
    p.seedPart(size=seed, deviationFactor=0.1, minSizeFactor=0.1)
    c, e = p.cells, p.edges
    pickedRegions = c.getSequenceFromMask(mask=('[#1 ]', ), )
    p.setMeshControls(regions=pickedRegions,
                      technique=SWEEP,
                      algorithm=ADVANCING_FRONT)
    p.setSweepPath(region=c[0], edge=e[10], sense=FORWARD)
    elemType1 = mesh.ElemType(elemCode=C3D8R,
                              elemLibrary=EXPLICIT,
                              kinematicSplit=AVERAGE_STRAIN,
                              secondOrderAccuracy=ON,
                              hourglassControl=ENHANCED,
                              distortionControl=ON,
                              lengthRatio=0.100000001490116)
    elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
    elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT)
    c = p.cells
    cells = c.getSequenceFromMask(mask=('[#1 ]', ), )
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions,
                     elemTypes=(elemType1, elemType2, elemType3))
    p.generateMesh()
    # (unique node numbering)
    p.setValues(startNodeLabel=4000000, startElemLabel=4000000)
    p.generateMesh()


def VitreousMesh(v1Seed, v2Seed):
    ''' Specity tetrahedral elements '''
    p = abqModel.parts['V']
    c = p.cells
    pickedRegions = c.getSequenceFromMask(mask=('[#86f800 ]', ), )
    p.setMeshControls(regions=pickedRegions, elemShape=TET, technique=FREE)
    elemType1 = mesh.ElemType(elemCode=C3D20R)
    elemType2 = mesh.ElemType(elemCode=C3D15)
    elemType3 = mesh.ElemType(elemCode=C3D10)
    cells = c.getSequenceFromMask(mask=('[#86f800 ]', ), )
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions,
                     elemTypes=(elemType1, elemType2, elemType3))

    ''' Specify hexahedral elements '''
    elemType1 = mesh.ElemType(elemCode=C3D8R, elemLibrary=EXPLICIT)
    elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
    elemType3 = mesh.ElemType(elemCode=C3D4,
                              elemLibrary=EXPLICIT,
                              secondOrderAccuracy=ON,
                              distortionControl=ON,
                              lengthRatio=0.100000001490116)
    cells = c.getSequenceFromMask(mask=('[#86f800 ]', ), )
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions,
```

```
                     elemTypes=(elemType1, elemType2, elemType3))

    elemType1 = mesh.ElemType(elemCode=C3D8R,
                              elemLibrary=EXPLICIT,
                              kinematicSplit=AVERAGE_STRAIN,
                              secondOrderAccuracy=ON,
                              hourglassControl=ENHANCED,
                              distortionControl=ON,
                              lengthRatio=0.100000001490116)
    elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
    elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT)
    cells = c.getSequenceFromMask(mask=('[#7907ff ]', ), )
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions,
                     elemTypes=(elemType1, elemType2, elemType3))

    # Seed the entire part
    p.seedPart(size=v2Seed, deviationFactor=0.1, minSizeFactor=0.1)

    # Seed the retina interface
    e = p.edges
    pickedEdges = e.getSequenceFromMask(mask=('[#ffffffff #7fec0fff #80012 ]',
                                              ), )
    p.seedEdgeBySize(edges=pickedEdges,
                     size=v1Seed,
                     deviationFactor=0.1,
                     minSizeFactor=0.1,
                     constraint=FINER)

    # Seed the bias edges
    e = p.edges
    pickedEdges1 = e.getSequenceFromMask(mask=('[#0 #104000 #10001 ]', ), )
    pickedEdges2 = e.getSequenceFromMask(mask=('[#0 #80020000 #900000 ]', ), )
    p.seedEdgeByBias(biasMethod=SINGLE,
                     end1Edges=pickedEdges1,
                     end2Edges=pickedEdges2,
                     minSize=v1Seed,
                     maxSize=v2Seed,
                     constraint=FINER)

    # (unique node numbering)
    p.setValues(startNodeLabel=5000000, startElemLabel=5000000)
    p.generateMesh()


def QuadraticTetVitreous():
    # Vitreous
    p = abqModel.parts['V']
    c = p.cells
    pickedRegions = c.getSequenceFromMask(mask=('[#9f ]', ), )
    p.deleteMesh(regions=pickedRegions)
    p.setMeshControls(regions=pickedRegions, elemShape=TET, technique=FREE)
    elemType1 = mesh.ElemType(elemCode=UNKNOWN_HEX, elemLibrary=EXPLICIT)
    elemType2 = mesh.ElemType(elemCode=UNKNOWN_WEDGE, elemLibrary=EXPLICIT)
    elemType3 = mesh.ElemType(elemCode=C3D10M, elemLibrary=EXPLICIT)
    cells = c.getSequenceFromMask(mask=('[#9f ]', ), )
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions,
```

```
                     elemTypes=(elemType1, elemType2, elemType3))
    elemType1 = mesh.ElemType(elemCode=UNKNOWN_HEX, elemLibrary=EXPLICIT)
    elemType2 = mesh.ElemType(elemCode=UNKNOWN_WEDGE, elemLibrary=EXPLICIT)
    elemType3 = mesh.ElemType(elemCode=C3D10M,
                              elemLibrary=EXPLICIT,
                              secondOrderAccuracy=ON,
                              distortionControl=ON,
                              lengthRatio=0.100000001490116)
    c = p.cells
    p.setElementType(regions=pickedRegions,
                     elemTypes=(elemType1, elemType2, elemType3))
    p.generateMesh()


def QuadraticTetRetina():
    # Retina
    p = abqModel.parts['R']
    c = p.cells
    pickedRegions = c.getSequenceFromMask(mask=('[#1 ]', ), )
    p.deleteMesh(regions=pickedRegions)
    p.setMeshControls(regions=pickedRegions, elemShape=TET, technique=FREE)
    elemType1 = mesh.ElemType(elemCode=UNKNOWN_HEX, elemLibrary=EXPLICIT)
    elemType2 = mesh.ElemType(elemCode=UNKNOWN_WEDGE, elemLibrary=EXPLICIT)
    elemType3 = mesh.ElemType(elemCode=C3D10M, elemLibrary=EXPLICIT)
    c = p.cells
    cells = c.getSequenceFromMask(mask=('[#1 ]', ), )
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions,
                      elemTypes=(elemType1, elemType2, elemType3))
    elemType1 = mesh.ElemType(elemCode=UNKNOWN_HEX, elemLibrary=EXPLICIT)
    elemType2 = mesh.ElemType(elemCode=UNKNOWN_WEDGE, elemLibrary=EXPLICIT)
    elemType3 = mesh.ElemType(elemCode=C3D10M,
                              elemLibrary=EXPLICIT,
                              secondOrderAccuracy=ON,
                              distortionControl=ON,
                              lengthRatio=0.100000001490116)
    p.setElementType(regions=pickedRegions,
                      elemTypes=(elemType1, elemType2, elemType3))
    p.generateMesh()


def Assembly():
    a1 = abqModel.rootAssembly
    a1.DatumCsysByDefault(CARTESIAN)
    p = abqModel.parts['E']
    a1.Instance(name='E-1', part=p, dependent=ON)
    p = abqModel.parts['G']
    a1.Instance(name='G-1', part=p, dependent=ON)
    p = abqModel.parts['R']
    a1.Instance(name='R-1', part=p, dependent=ON)
    p = abqModel.parts['T']
    a1.Instance(name='T-1', part=p, dependent=ON)
    p = abqModel.parts['V']
    a1.Instance(name='V-1', part=p, dependent=ON)


def GravityStep(time, prevStep, scaleFactor, MSTI, stepName, descrip):
    abqModel.ExplicitDynamicsStep(name=stepName,
```

```
                                  previous=prevStep,
                                  description=descrip,
                                  timePeriod=time,
                                  massScaling=((SEMI_AUTOMATIC,
                                                MODEL,
                                                AT_BEGINNING,
                                                scaleFactor,
                                                MSTI,
                                                BELOW_MIN, 0, 0, 0.0, 0.0, 0,
                                                None), ),
                                  nlgeom=ON)


def General_Contact(stepName, cIP):
    # Rename the two variables
    GC_IP = 'IntProp-GC' # Interaction property
    GC = 'General_Contact' # General Contact name
    # cIP = 'cohesive_IntProp' # cohesive interaction property name
    abqModel.ContactProperty(GC_IP)

    GC_IntProp = abqModel.interactionProperties[GC_IP] # simplify code

    # if gravity == True:
    # Gravity keeps the vitreous from energetically moving after peeling
    GC_IntProp.TangentialBehavior(formulation=PENALTY,
                                  directionality=ISOTROPIC,
                                  slipRateDependency=OFF,
                                  pressureDependency=OFF,
                                  temperatureDependency=OFF,
                                  dependencies=0,
                                  table=((0.2, ), ),
                                  shearStressLimit=None,
                                  maximumElasticSlip=FRACTION,
                                  fraction=0.005,
                                  elasticSlipStiffness=None)
    # else:
    #     # Prevent the vitreous from sliding inside the eye holder
    #     GC_IntProp.TangentialBehavior(formulation=ROUGH)

    GC_IntProp.NormalBehavior(pressureOverclosure=HARD,
                              allowSeparation=ON,
                              constraintEnforcementMethod=DEFAULT)
    abqModel.ContactExp(name=GC, createStepName=stepName)

    GC_Int = abqModel.interactions[GC] # simplify code
    GC_Int.includedPairs.setValuesInStep(stepName=stepName, useAllstar=ON)
    GC_Int.contactPropertyAssignments.appendInStep(stepName=stepName,
                                                   assignments=((GLOBAL,
                                                                 SELF,
                                                                 GC_IP),
                                                                )
                                                   )


def updateGeneralContact(stepName, Knn, Kss, Ktt, damageInitiation,
                         tn, ts, tt, damageEvolution, FE):
    ''' Specify the cohesive surface behavior between the retina and vitreous
    during the step after the gravity step '''
```

```
    # Simplify
    GC = 'General_Contact'
    cp = 'cohesivePeel'

    abqModel.ContactProperty(cp)

    CP_IP = abqModel.interactionProperties[cp]
    CP_IP.TangentialBehavior(formulation=PENALTY,
                             directionality=ISOTROPIC,
                             slipRateDependency=OFF,
                             pressureDependency=OFF,
                             temperatureDependency=OFF,
                             dependencies=0,
                             table=((0.2, ), ),
                             shearStressLimit=None,
                             maximumElasticSlip=FRACTION,
                             fraction=0.005,
                             elasticSlipStiffness=None)

    CP_IP.CohesiveBehavior(defaultPenalties=OFF,
                           table=((Knn, Kss, Ktt), ))
    # eligibility=INITIAL_NODES,

    CP_IP.Damage(criterion=MAX_STRESS,
                 initTable=((tn, ts, tt), ),
                 useEvolution=ON,
                 evolutionType=ENERGY,
                 evolTable=((FE, ), ),
                 useStabilization=ON,
                 viscosityCoef=1e-05)

    GCI = abqModel.interactions[GC]
    if gravity == True:
        GCI.contactPropertyAssignments.changeValuesInStep(stepName=stepName,
                                                          index=1,
                                                          value=cp)
    else:
        r11=abqModel.rootAssembly.instances['R-1'].surfaces['R_V_Surf_BOND']
        r12=abqModel.rootAssembly.instances['V-1'].surfaces['V_R_Surf_BOND']
        GCI.contactPropertyAssignments.appendInStep(stepName=stepName,
                                                    assignments=((r11, r12,
                                                                  cp), ))


def smoothGravity():
    abqModel.SmoothStepAmplitude(name='smoothGravity', timeSpan=STEP,
        data=((0.0, 0.0), (100.0, 1.0)))
    abqModel.loads['Gravity'].setValues(amplitude='smoothGravity',
        distributionType=UNIFORM, field='')


def turnTieCohesive(stepName, cohTieName):
    ''' Simulate the tie constraint with cohesive surface '''
    GC = 'General_Contact'
    CTG = cohTieName # Simplify
    abqModel.ContactProperty(CTG)

    # Simplify
```

```python
    CTG_IP = abqModel.interactionProperties[CTG]
    GC_IP = abqModel.interactions[GC]

    CTG_IP.CohesiveBehavior(eligibility=INITIAL_NODES)
    r11=abqModel.rootAssembly.instances['R-1'].surfaces['R_V_Surf_BOND']
    r12=abqModel.rootAssembly.instances['V-1'].surfaces['V_R_Surf_BOND']
    GC_IP.contactPropertyAssignments.appendInStep(stepName=stepName,
                                              assignments=((r11,
                                                            r12,
                                                            CTG), ))


def peelStepPostGravity(time, stepName, previousStep, descrip, scaleFactor,
                        MSTI):
    ''' step after the gravity phase '''
    abqModel.ExplicitDynamicsStep(name=stepName,
                                  previous=previousStep,
                                  description=descrip,
                                  timePeriod=time,
                                  massScaling=((SEMI_AUTOMATIC,
                                                MODEL, AT_BEGINNING,
                                                scaleFactor, MSTI, BELOW_MIN,
                                                0, 0, 0.0, 0.0, 0, None), ))


def F_output(stepName):
    FOutputInterval = 50 # Double the data points (Default is 20)
    # Whole Model Fieldoutput (RF, U, NFORC)
    abqModel.FieldOutputRequest(name='F-Output-1',
                                createStepName=stepName,
                                variables=('RF',
                                           'U',
                                           'NFORC'),
                                numIntervals=FOutputInterval)

    # Set specific field output (Retina LE & S)
    regionDef=abqModel.rootAssembly.allInstances['R-1'].sets['R_Set']
    abqModel.FieldOutputRequest(name='Retina_LE_S',
                                createStepName=stepName,
                                variables=('LE',
                                           'S'),
                                numIntervals=FOutputInterval,
                                region=regionDef,
                                sectionPoints=DEFAULT,
                                rebar=EXCLUDE)

    # Set specific field output (Vitreous LE & S)
    regionDef=abqModel.rootAssembly.allInstances['V-1'].sets['V_Set']
    abqModel.FieldOutputRequest(name='Vitreous_LE_S',
                                createStepName=stepName,
                                variables=('LE',
                                           'S'),
                                numIntervals=FOutputInterval,
                                region=regionDef,
                                sectionPoints=DEFAULT,
                                rebar=EXCLUDE)

    # # Set specific field output (Rigid Body U & RF)
```

```python
    # regionDef=abqModel.rootAssembly.allInstances['G-1'].sets['G_RP_Set']
    # abqModel.FieldOutputRequest(name='Glue_U_RF',
    #                             createStepName=stepName,
    #                             variables=('U', 'RF'),
    #                             numIntervals=FOutputInterval,
    #                             region=regionDef,
    #                             sectionPoints=DEFAULT,
    #                             rebar=EXCLUDE)

    # # Contact Pair field output (CP - CSTRESS, CDISP, CFORCE)
    # abqModel.FieldOutputRequest(name='CF_Output_R_V',
    #                             createStepName=stepName,
    #                             variables=('CSTRESS', 'CDISP', 'CFORCE'),
    #                             numIntervals=FOutputInterval,
    #                             interactions=('CP-R-V', ),
    #                             sectionPoints=DEFAULT,
    #                             rebar=EXCLUDE)


    # # Contact Pair (Bond) field output (CP - CSTRESS, CDISP, CFORCE)
    # abqModel.FieldOutputRequest(name='CF_Output_BOND',
    #                             createStepName=stepName,
    #                             variables=('CSTRESS', 'CDISP', 'CFORCE'),
    #                             interactions=('CP_BOND', ),
    #                             sectionPoints=DEFAULT,
    #                             rebar=EXCLUDE)


def H_output(stepName):
    # Internal/Kinetic Energy
    abqModel.HistoryOutputRequest(name='H-Output-1',
                                createStepName=stepName,
                                variables=('ALLIE',
                                           'ALLKE'))

    # # Define specific reaction force on the glue reference point
    # a = abqModel.rootAssembly
    # regionDef=a.allInstances['G-1'].sets['G_RP_Set']
    # abqModel.HistoryOutputRequest(name='G_RP_Output_U_RF_RM',
    #                               createStepName=stepName,
    #                               variables=('U1', 'U2', 'U3',
    #                                          'RF1', 'RF2', 'RF3',
    #                                          'RM1', 'RM2', 'RM3'),
    #                               region=regionDef,
    #                               sectionPoints=DEFAULT,
    #                               rebar=EXCLUDE)

    # # Define specific CFN between the vitreous and retina
    # regionDef=a.allInstances['V-1'].sets['V_R_Set']
    # abqModel.HistoryOutputRequest(name='Contact-VR-Hist',
    #                               createStepName=stepName,
    #                               variables=('CFN1', 'CFN2', 'CFN3', 'CAREA'),
    #                               region=regionDef,
    #                               sectionPoints=DEFAULT,
    #                               rebar=EXCLUDE)

    # # Define specific CFN between the vitreous and retina
    # abqModel.HistoryOutputRequest(name='Contact_CP-R-V',
```

```
    #                              createStepName=stepName,
    #                              variables=('CFN1', 'CFN2', 'CFN3', 'CAREA'),
    #                              interactions=('CP-R-V', ),
    #                              sectionPoints=DEFAULT,
    #                              rebar=EXCLUDE)

    # # Define specific CFN between the retina and glue
    # abqModel.HistoryOutputRequest(name='Contact_CP-R-G',
    #                              createStepName=stepName,
    #                              variables=('CFN1', 'CFN2', 'CFN3', 'CAREA'),
    #                              interactions=('CP-R-G', ),
    #                              sectionPoints=DEFAULT,
    #                              rebar=EXCLUDE)


def CP_RV():
    abqModel.ContactProperty('CP_R_V_Int_Prop')
    a = abqModel.rootAssembly
    s1 = a.instances['R-1'].faces
    side1Faces1 = s1.getSequenceFromMask(mask=('[#4 ]', ), )
    a.Surface(side1Faces=side1Faces1, name='CP-R_V')
    region1=a.surfaces['CP-R_V']
    s1 = a.instances['V-1'].faces
    side1Faces1 = s1.getSequenceFromMask(mask=('[#5090 ]', ), )
    a.Surface(side1Faces=side1Faces1, name='CP-V_R')
    region2=a.surfaces['CP-V_R']
    abqModel.SurfaceToSurfaceContactExp(name='CP-R-V',
                                        createStepName='Initial',
                                        master=region1,
                                        slave=region2,
                                        sliding=FINITE,
                                        interactionProperty='CP_R_V_Int_Prop',
                                        weightingFactorType=SPECIFIED,
                                        weightingFactor=1.0,
                                        initialClearance=OMIT,
                                        datumAxis=None,
                                        clearanceRegion=None)


def Amp():
    abqModel.SmoothStepAmplitude(name='TD_amp',
                                 timeSpan=STEP,
                                 data=((0.0, 0.0),
                                       (30.0, 2e-05))
                                 )
    abqModel.SmoothStepAmplitude(name='omega',
                                 timeSpan=STEP,
                                 data=((0.0, 0.0),
                                       (30.0, 0.000909174))
                                 )


def EHR_BC_Fixed(stepName):
    a = abqModel.rootAssembly
    region = a.instances['E-1'].sets['E_RP_Set']
    abqModel.VelocityBC(name='EHR', createStepName=stepName, region=region,
                        v1=0.0, v2=0.0, v3=0.0, vr1=0.0, vr2=0.0, vr3=0.0,
                        amplitude='omega', localCsys=None,
```

```
                        distributionType=UNIFORM, fieldName='')


def EHR_BC(stepName):
    a = abqModel.rootAssembly
    region = a.instances['E-1'].sets['E_RP_Set']
    abqModel.VelocityBC(name='EHR',
                        createStepName=stepName,
                        region=region,
                        v1=0.0, v2=0.0, v3=0.0,
                        vr1=0.0, vr2=0.0, vr3=-1.0,
                        amplitude='omega',
                        localCsys=None,
                        distributionType=UNIFORM,
                        fieldName='')


def Retina_Disp_BC(stepName):
    a = abqModel.rootAssembly
    region = a.instances['R-1'].sets['R_G_Set']
    abqModel.VelocityBC(name='R_Vel',
                        createStepName=stepName,
                        region=region,
                        v1=0.866092,
                        v2=0.499884,
                        v3=UNSET,
                        vr1=UNSET,
                        vr2=UNSET,
                        vr3=UNSET,
                        amplitude='TD_amp',
                        localCsys=None,
                        distributionType=UNIFORM,
                        fieldName='')


def Write_Job(jobName, modelName, jobDescription):
    mdb.Job(name=jobName,
            model=modelName,
            description=jobDescription,
            type=ANALYSIS,
            atTime=None,
            waitMinutes=0,
            waitHours=0,
            queue=None,
            memory=90,
            memoryUnits=PERCENTAGE,
            explicitPrecision=DOUBLE,
            nodalOutputPrecision=SINGLE,
            echoPrint=OFF,
            modelPrint=OFF,
            contactPrint=OFF,
            historyPrint=OFF,
            userSubroutine='',
            scratch='',
            resultsFormat=ODB,
            parallelizationMethodExplicit=DOMAIN,
            numDomains=14,
            activateLoadBalancing=False,
```

```
            multiprocessingMode=DEFAULT,
            numCpus=14)


def Save_INP(jobName):
    mdb.jobs[jobName].writeInput(consistencyChecking=OFF)


def VR_Tie():
    a = abqModel.rootAssembly
    slaveSurf=a.instances['V-1'].surfaces['V_R_Surf_BOND']
    mastSurf=a.instances['R-1'].surfaces['R_V_Surf_BOND']
    abqModel.Tie(name='RV_Tie',
                 master=mastSurf,
                 slave=slaveSurf,
                 positionToleranceMethod=COMPUTED,
                 adjust=OFF,
                 tieRotations=ON,
                 constraintEnforcement=SURFACE_TO_SURFACE,
                 thickness=ON)
    return '_VR_Tie'


def keywordBlockR_G_SET_NodeNum():
    # Work here because the abaqus default is to use Nset Generate
    # Think about this method.  Used to be V_R_Set
    a = abqModel
    modelkwb = a.keywordBlock
    assembly = a.rootAssembly

    # Synch edits to modelkwb with those made in the model. We don't need
    # access to *nodes and *elements as they would appear in the inp file,
    # so set the storeNodesAndElements arg to False.
    modelkwb.synchVersions(storeNodesAndElements=False)

    # Search the modelkwb for the desired insertion point.  If it is found, we
    # break the loop, storing the line number, and then write our keywords
    # using the insert method (which actually inserts just below the specified
    # line number, fyi).
    line_num = 0
    for n, line in enumerate(modelkwb.sieBlocks):
        if line.find('*Nset, nset=R_G_Set') >= 0:
            line_num = n
            break
    if line_num:
        line = line.replace('\n', ',') # replaces the new line with commas

        if line.find('generate') == -1:
            if line[-1] == ',':
                # if the node list ends with a comma
                # split up the string into ints
                nList = [int(i) for i in line[20:-1].split(',')]
            else:
                # split up the string into ints
                nList = [int(i) for i in line[20:].split(',')]

            nodeNum = len(nList) # count the number of nodes
        else:
```

```
            # use the equation in ABQ documentation to determine the number
            # of nodes in the set
            # nNodes = (n2-n1)/increment
            # split up the string into ints
            nList = [int(i) for i in line[30:].split(',')]
            n1 = nList[0] # first node
            n2 = nList[1] # last node
            increment = nList[2]
            nodeNum = (n2 - n1)/increment + 1
        print(nodeNum, 'nodes in nset=R_G_Set')
    else:
        e = ("Error: R_G_Set was not found in the Model KeywordBlock to " +
             "determine the number of nodes in the nodeSet.")
        raise Exception(" ".join(e))
    return nodeNum


def keywordBlockBond(MSTI, scaleFactor, FN, FS, db, ufn, ufs):
    a = abqModel
    modelkwb = a.keywordBlock
    assembly = a.rootAssembly

    #if assembly.isOutOfDate:
    #    assembly.regenerate()

    # Synch edits to modelkwb with those made in the model. We don't need
    # access to *nodes and *elements as they would appear in the inp file,
    # so set the storeNodesAndElements arg to False.
    modelkwb.synchVersions(storeNodesAndElements=False)

    # Search the modelkwb for the desired insertion point.  If it is found, we
    # break the loop, storing the line number, and then write our keywords
    # using the insert method (which actually inserts just below the specified
    # line number, fyi).

    # The abaqus keyword "*BOND" needs to be defined after the "*STEP" keyword
    # in the .inp file.  "Search for the "*Fixed Mass Scaling ..." term to
    # add text specific to the bond
    keywordSearch = ('*Fixed Mass Scaling, ' +
                     'dt={}, type=below min, '.format(MSTI) +
                     'factor={:.0f}.'.format(scaleFactor))
    line_num = 0
    for n, line in enumerate(modelkwb.sieBlocks):
        if line == keywordSearch:
            line_num = n
            break
    if line_num:
        kwds = ('*Contact Pair, interaction=Bond_Int_Prop, mechanical ' +
                'constraint=KINEMATIC, weight=0., ' +
                'cpset=CP_Bond\nV-1.V_R_SURF_BOND, ' +
                'R-1.R_V_SURF_BOND\n*Surface Interaction, ' +
                'name=Bond_Int_Prop\n' +
                '*BOND\nV-1.V_R_SET, {}'.format(FN) +
                ', {}'.format(FS) +
                ', {}'.format(db) +
                ', '+', {}'.format(ufn) +
                ', {}'.format(ufs))
        modelkwb.insert(position=line_num, text=kwds)
```

```python
    else:
        e = ("Error: Mass Scaling was not found in the Model KeywordBlock.")
        raise Exception(" ".join(e))


def keywordBlockBondNPDFM(FN, FS):
    a = abqModel
    modelkwb = a.keywordBlock
    assembly = a.rootAssembly

    #if assembly.isOutOfDate:
    #    assembly.regenerate()

    # Synch edits to modelkwb with those made in the model. We don't need
    # access to *nodes and *elements as they would appear in the inp file,
    # so set the storeNodesAndElements arg to False.
    modelkwb.synchVersions(storeNodesAndElements=False)

    # Search the modelkwb for the desired insertion point.  If it is found, we
    # break the loop, storing the line number, and then write our keywords
    # using the insert method (which actually inserts just below the specified
    # line number, fyi).
    line_num = 0
    for n, line in enumerate(modelkwb.sieBlocks):
        # Use this line because CP from defined in code doesn't seem to work..
        if line == '*Contact Property Assignment\n ,  , IntProp-GC':
            line_num = n
            break
    if line_num:
        kwds = ('*Contact Pair, interaction=Bond_Int_Prop, mechanical ' +
                'constraint=KINEMATIC, weight=0., ' +
                'cpset=CP_Bond\nV-1.V_R_SURF_BOND, ' +
                'R-1.R_V_SURF_BOND\n*Surface Interaction, ' +
                'name=Bond_Int_Prop\n' +
                '*BOND\nV-1.V_R_SET, {}'.format(FN) +
                ', {}'.format(FS) +
                ', , ')
        modelkwb.insert(position=line_num, text=kwds)
    else:
        e = ("Error: Mass Scaling was not found in the Model KeywordBlock.")
        raise Exception(" ".join(e))


def keywordBlockBondHistOutput():
    a = abqModel
    modelkwb = a.keywordBlock
    assembly = a.rootAssembly

    # Synch edits to modelkwb with those made in the model. We don't need
    # access to *nodes and *elements as they would appear in the inp file,
    # so set the storeNodesAndElements arg to False.
    modelkwb.synchVersions(storeNodesAndElements=False)

    # Search the modelkwb for the desired insertion point.  If it is found, we
    # break the loop, storing the line number, and then write our keywords
    # using the insert method (which actually inserts just below the specified
    # line number, fyi).
    line_num = 0
```

```
    for n, line in enumerate(modelkwb.sieBlocks):
        #print(n,line)
        if line == "*Energy Output\nALLIE, ALLKE":
            line_num = n
            break
    if line_num:
        kwds = '*Contact Output, Nset=V-1.V_R_SET\nBONDSTAT, BONDLOAD'
        modelkwb.insert(position=line_num, text=kwds)
    else:
        e = ("Error: Bond Output was not found in the Model KeywordBlock.")
        raise Exception(" ".join(e))


# Added 9/11/2020
def keywordBlockBondFieldOutput():
    a = abqModel
    modelkwb = a.keywordBlock
    assembly = a.rootAssembly

    modelkwb.synchVersions(storeNodesAndElements=False)
    # Add history output for the contact
    line_num = 0
    for n, line in enumerate(modelkwb.sieBlocks):
        if line == '*Element Output, directions=YES\nNFORC, ':
            line_num = n
            break
    if line_num:
        kwds = ('** FIELD OUTPUT: CF_Output_R_V\n\n**\n*Contact Output, ' +
                'cpset=CP_Bond\nCDISP, CFORCE, CSTRESS')
        modelkwb.insert(position=line_num, text=kwds)
    else:
        e = ("Error: Bond Output was not found in the Model KeywordBlock.")
        raise Exception(" ".join(e))


def Submit_job(jobname):
    myJob = mdb.jobs[jobname]
    try:
        myJob.submit(consistencyChecking=OFF)
        myJob.waitForCompletion()
    except:
        print(str(datetime.datetime.now())+' stop by error!')
        pass


def FEA():
    """
    Function that generates FEA code to model vitreoretinal adhesion

    # Steps are as follows:
        1 - Create new model database
        2 - Import SolidWorks STEP file (Includes all parts)
        3 - Material property definitions
        4 - Part features (Element & Node Sets & Reference Points ...)
        5 - Mesh parts (Specify seed size)
        6 - Assembly
        7 - Step (Dynamic Explicit with Mass Scaling)
        8 - Outputs (Field & History)
```

```
        9 - Contact (General Contact)
        10 - Contact pair (Retina/Vitreous - Bonded Surface)
        11 - Tie Constraint (Retina - Glue)
        12 - Amplitude definition
        13 - BC's'
        14 - Submit Job  :)
    """

    # Import SolidWorks STEP file
    ImportStepEyeConstrained()

    # Mat Props
    Retina_Mat_Prop(RetinaProp)
    Vitreous_Mat_Prop(VitreousProp)

    # # Part Geometry/RPs/Sets/Surfaces
    E_Features()
    G_Features()
    T_Features()
    R_Features()

    # Internal sphere to reduce mesh
    V_Partition_XYZ_Axis()
    V_Internal_Sphere()
    AssembleV_for_Merging()
    mergeV()

    # Features on the vitreous
    PartitionRetinaOnVitreous()
    Vitreous_Features()

    # Seed & Mesh parts
    E_Mesh(e1Seed, e2Seed) # Max/min
    G_Mesh(gSeed)
    T_Mesh(ptSeed)
    R_Mesh(rSeed)
    VitreousMesh(v1Seed, v2Seed)

    # Assembly
    Assembly()

    # Eliminate the glue and tab from the model
    a = abqModel.rootAssembly
    a.features['G-1'].suppress()
    a.features['T-1'].suppress()

    # Gravity Step
    previousStep = 'Initial'
    if gravity == True:
        stepName = 'Gravity_Step'
        descrip = ('Applying gravity to the model and letting the ' +
                   'vitreous and retina settle')

        GravityStep(200, previousStep, scaleFactor, 0.03125, stepName, descrip)
        Gravity(stepName)
        smoothGravity()

        # Interactions
```

```
        cohTieName = 'Cohesive_Gravity_Tie'
        General_Contact(stepName, cohTieName)

        # Interaction properties
        turnTieCohesive(stepName, cohTieName)

        # Zero movement boundary conditions
        Amp()
        EHR_BC_Fixed(stepName)

        # # Model outputs for gravity step
        F_output(stepName)
        H_output(stepName)

        previousStep = stepName # Update the previous step to be gravity
    else:
        ''' General contact ''' # fix here if no gravity is specified
        peelCoh = 'Cohesive_Peel_Int'
        General_Contact(previousStep, peelCoh)


    # # Peel Step
    stepName = 'Peel_Test_Dynamic_Explicit'
    descrip = 'Peel the retina away from the vitreous (rotational peel test)'
    peelStepPostGravity(time, stepName, previousStep, descrip, scaleFactor,
                        MSTI)

    # Interactions
    # CP_RV() # comment out?

    # Boundary conditions
    Amp()
    EHR_BC(stepName)
    Retina_Disp_BC(stepName)

    # Model Outputs
    F_output(stepName)
    H_output(stepName)

    if tieInterface == True:
        """ If The VR interface is tied """
        VR_Tie()
    else:
        """ If bonding is true, then add the *Bond info to the .inp file """
          # determine number of nodes in the nodeSet
        nodeNum = keywordBlockR_G_SET_NodeNum()

        # Bonding
        # divide the bond tension load by the number of nodes in the set
        FN_Norm = FN/nodeNum
        # divide the bond shear load by the number of nodes in the set
        FS_Norm = FS/nodeNum

        if PDFMStatus == True:
            # Include post damage failure model
            # FN, FS, SpotWeldRadius, ufn, ufs
            keywordBlockBond(MSTI, scaleFactor, FN_Norm,
                             FS_Norm, db, ufn, ufs)
```

```
        else:
            # Do not include post damage failure model
            keywordBlockBondNPDFM(FN_Norm, FS_Norm)

        keywordBlockBondHistOutput()
        keywordBlockBondFieldOutput()

    # Undo the spacing to pass in the job description
    global jobDescription
    # replace new lines, spaces, equal signs
    jobDescription = jobDescription.replace('NEWLINE', '\n')
    jobDescription = jobDescription.replace('TAB', '\t')
    jobDescription = jobDescription.replace('SPACE', ' ')
    jobDescription = jobDescription.replace('EQUALSSIGN', '=')

    Write_Job(jobName, modelName, jobDescription)
    print('Job has been written')
    Save_INP(jobName)
    Submit_job(jobName)
    print('Job has been submitted')
    del mdb.models['Model-1']


# In[Main import info]


if __name__ == '__main__':
    """ Run the following function """

    # Print File of tests & attributes ran in order to make sure they are
    # being properly pass through
    print("\nWriting out the Argument Data...")
    filename = os.path.join(abqWD, 'FEAArgumentData' + '.txt')
    outfile = open(filename,'w')
    outfile.write('sys.argv\n')
    outfile.write('\n'.join(sys.argv)) # write all arguments passed into abaqus
    outfile.close()
    print("\nDone!")
    print("\nThe output file will be named '{}".format(filename) + "'")
    print("\nIt will be in the same working directory as your Abaqus model\n")

    # # Testing when importing into abaqus script
    # gravity =              eval('False') # gravity
    # symmetry =             eval('False') # symmetry
    # simplified =           eval('True') # simplified model (not used anymore)
    # modelName =                 'ABQScript' # model name
    # jobName =                   'jobNameTest' # file name/job name
    # time =                float('50')
    # e1Seed =                    '[10,1,0.0009765625]'
    # e2Seed =                    '[8,1,0.00390625]'
    # ptSeed =                    '[6,1,0.015625]'
    # gSeed =                     '[7,1,0.0078125]'
    # v1Seed =                    '[10,1,0.0009765625]' #
    ↪  '[11.38,1,0.00037521366730664343]' #
    # v2Seed =                    '[8,1,0.00390625]'
    # rSeed =                     '[10,1,0.0009765625]'
    # scaleFactor =               '[0,1,1]'
```

```
    # MSTI =                      '[4,1,0.0625]' # MassScaleTimeIncrement
    # RetinaProp =          float('11120.0') # Young's modulus for retina
    # VitreousProp =        float('400') # Young's modulus for vitreous
    # BondStatus =           eval('True')
    # FN =                        '[-5,1,0.03125]'
    # FS =                        '[-5,1,0.03125]'
    # PDFMStatus =           eval('True') # True/False - convert to bool
    # db =                        '[-5,1,0.03125]'
    # ufn =                       '[-5,1,0.03125]'
    # ufs =                       '[-5,1,0.03125]'
    # OptimizationStatus =        'None' # None/variables to be optimized
    # tieInterface =         eval('False') # True/False - convert to bool
    # jobDescription =            """Test Model via Abaqus Run Script"""

    # Pass in arguments from previous file Strip the brackets from the strings
    gravity =                   sys.argv[-27]
    symmetry =                  sys.argv[-26]
    simplified =                sys.argv[-25]
    modelName =                 sys.argv[-24] # model name
    jobName =                   sys.argv[-23] # file name/job name
    time =                float(sys.argv[-22])
    e1Seed =                    sys.argv[-21]
    e2Seed =                    sys.argv[-20]
    ptSeed =                    sys.argv[-19]
    gSeed =                     sys.argv[-18]
    v1Seed =                    sys.argv[-17]
    v2Seed =                    sys.argv[-16]
    rSeed =                     sys.argv[-15]
    scaleFactor =               sys.argv[-14]
    MSTI =                      sys.argv[-13]
    RetinaProp =          float(sys.argv[-12]) # Young's modulus for retina
    VitreousProp =        float(sys.argv[-11]) # Young's modulus for vitreous
    BondStatus =           eval(sys.argv[-10])
    FN =                        sys.argv[-9]
    FS =                        sys.argv[-8]
    PDFMStatus =           eval(sys.argv[-7]) # True/False - convert to bool
    db =                        sys.argv[-6]
    ufn =                       sys.argv[-5]
    ufs =                       sys.argv[-4]
    OptimizationStatus =        sys.argv[-3] # None/variables to be optimized
    tieInterface =              sys.argv[-2]
    jobDescription =            sys.argv[-1] # String

    """ Convert the strings back to lists of floats """
    e1SeedStrip = str(e1Seed)[1:-1] # Strip the brackets from the string
    e1SeedList = [float(i) for i in e1SeedStrip.split(',')]
    e1Seed = e1SeedList[2] # value

    e2SeedStrip = str(e2Seed)[1:-1] # Strip the brackets from the string
    e2SeedList = [float(i) for i in e2SeedStrip.split(',')]
    e2Seed = e2SeedList[2] # value

    ptSeedStrip = str(ptSeed)[1:-1] # Strip the brackets from the string
    ptSeedList = [float(i) for i in ptSeedStrip.split(',')]
    ptSeed = ptSeedList[2] # value

    gSeedStrip = str(gSeed)[1:-1] # Strip the brackets from the string
    gSeedList = [float(i) for i in gSeedStrip.split(',')]
```

```
    gSeed = gSeedList[2] # value

    v1SeedStrip = str(v1Seed)[1:-1] # Strip the brackets from the string
    v1SeedList = [float(i) for i in v1SeedStrip.split(',')]
    v1Seed = v1SeedList[2] # value

    v2SeedStrip = str(v2Seed)[1:-1] # Strip the brackets from the string
    v2SeedList = [float(i) for i in v2SeedStrip.split(',')]
    v2Seed = v2SeedList[2] # value

    rSeedStrip = str(rSeed)[1:-1] # Strip the brackets from the string
    rSeedList = [float(i) for i in rSeedStrip.split(',')]
    rSeed = rSeedList[2] # value

    # Strip the brackets from the string
    scaleFactorStrip = str(scaleFactor)[1:-1]
    scaleFactorList = [float(i) for i in scaleFactorStrip.split(',')]
    scaleFactor = scaleFactorList[2] # value

    # Strip the brackets from the string
    # MassScaleTimeIncrement
    MSTIStrip = str(MSTI)[1:-1]
    MSTIList = [float(i) for i in MSTIStrip.split(',')]
    MSTI = MSTIList[2] # value

    # Strip the brackets from the string
    FNStrip = str(FN)[1:-1]
    FNList = [float(i) for i in FNStrip.split(',')]
    FN = FNList[2] # value

    # Strip the brackets from the string
    FSStrip = str(FS)[1:-1]
    FSList = [float(i) for i in FSStrip.split(',')]
    FS = FSList[2] # value

    dbStrip = str(db)[1:-1] # Strip the brackets from the string
    dbList = [float(i) for i in dbStrip.split(',')]
    db = dbList[2] # value

    ufnStrip = str(ufn)[1:-1] # Strip the brackets from the string
    ufnList = [float(i) for i in ufnStrip.split(',')]
    ufn = ufnList[2] # value

    ufsStrip = str(ufs)[1:-1] # Strip the brackets from the string
    ufsList = [float(i) for i in ufsStrip.split(',')]
    ufs = ufsList[2] # value

    """ Write the FEA Code """
    Mdb()
    modelDescription = ('Measure adhesion between the retina & vitreous of ' +
                        'the human eye using the Abaqus *Bond technique')
    abqModel = mdb.Model(name=modelName,
                         description=modelDescription,
                         modelType=STANDARD_EXPLICIT,
                         copyInteractions=ON,
                         copyConstraints=ON)

    # Call the function
```

```
    FEA()
```

1.5.4 Abaqus Extract Data Script

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

```
"""
Created on Wed Jun 17 16:48:49 2020

@author: Kiffer Creveling
Instructions:
    1) Save this script in a folder containing your ODB file
    2) Open a command window and navigate to your directory containing this script
    ↪ and your ODB file
    3) Create a .bat file
    3) Issue the command to call the script and extract data:
        abaqus python -c "import BpT; BpT.data_extract('xxxxxxx.odb')"
"""
# **************************
from odbAccess import *
import odbAccess as oa
from sys import argv, exit
from abaqusConstants import *
from textRepr import *
import pdb
import numpy as np
import os

""" Pass arguments into this script """
script =             sys.argv[0]
jobName =            sys.argv[1]
gravity =       eval(sys.argv[2]) # True/False
symmetry =      eval(sys.argv[3]) # True/False
simplified =    eval(sys.argv[4]) # True/False
BondStatus =    eval(sys.argv[5]) # True/False
PDFMStatus =    eval(sys.argv[6]) # True/False # not used in the extraction

def openOdb(jobName):
    """
    Function used to locate the .odb given a file name

    Parameters
    ----------
    jobName : Name of the ABAQUS .odb file

    Returns
    -------
    odb : Abaqus output file
    """
    if jobName.endswith('.odb'):
        odbFile = jobName
        try:
            odb=oa.openOdb(path=odbFile, readOnly=TRUE)
            print("\nOpening the odb file... (.odb was specified)")
            return odb
        except:
```

```
            print("ERROR: Unable to open the specified odb %s.  Exiting."
                  % odbFile)
            exit(0)

    else:
        odbFile = jobName + '.odb'
        # Try opening the odb file
        try:
            odb=oa.openOdb(path=odbFile, readOnly=TRUE)
            print("\nOpening the odb file... (Searching for .odb)")
            return odb
        except:
            print("ERROR: Unable to open the specified odb %s.  Exiting."
                  % odbFile)
            exit(0)

def data_extract(jobName):
    """
    Function used to extract data from the .odb file

    Parameters
    ----------
    jobName : The name of ABAQUS .odb file

    Returns
    -------
    Two files of data used for plotting
    """

    # due to symmetry multiply the values by 2
    if symmetry == True:
        mult = 2
    else:
        mult = 1

    frames = []
    try:
        odb = openOdb(jobName)
    except:
        print(os.getcwd())
        print("Looks like there is a problem with the job name or odb file")

    theta = 30
    LoadCellDirection = [np.cos(theta*np.pi/180), np.sin(theta*np.pi/180), 0]

    """ Field Output data arrays """
    RF = []

    # vector components of the reaction force
    RFx = []
    RFy = []
    RFz = []

    U_top = [] # values to append
    U_bot = [] # values to append
    Nforc = []

    # Used to calculate bond distance
```

```
    R_bot = [] # bottom of retina
    V_top = [] # top of vitreous
    Bond_disp = [] # Bond separation distance

    CnormF_RV = []
    CnormF_VR = []
    Cpress_RV = []
    Cpress_VR = []
    Cshear1_RV = []
    Cshear1_VR = []
    Cshear2_RV = []
    Cshear2_VR = []
    CshearF_RV = []
    CshearF_VR = []

    # Cpress_RV = []
    # Cpress_VR = []
    Cpress_RV_AVG = []
    Cpress_VR_AVG = []
    frames = [] # List of frames
    time = [] # Time array

    # Used for reaction force simplicity further in the code
    temp = [] # Temporary array used for iterating (Clears after each iteration)
    tempx = []
    tempy = []
    tempz = []

    # List variables for exporting data
    CnormF_RV_List = []
    CnormF_VR_List = []
    Cpress_RV_List = []
    Cpress_VR_List = []
    Cshear1_RV_List = []
    Cshear1_VR_List = []
    Cshear2_RV_List = []
    Cshear2_VR_List = []
    CshearF_RV_List = []
    CshearF_VR_List = []

    """ History Output data arrays """
    Hist_Time = []
    IE = []
    KE = []
    CAreaCP_RG = []
    CAreaCP_GR = []
    CAreaCP_RV = []
    CAreaCP_VR = []
    CFNCP_RG = []
    CFNCP_GR = []
    CFNCP_RV = []
    CFNCP_VR = []
    Glue_RP_RF = []

    """ Loop over the field outputs"""
    step = odb.steps.keys() # determines the step in the abaqus odb file (typically
    ↪  displacement)
    disp_step = step[0] # Defines the step as a variable name
```

```
    for frame, odbFrame in enumerate(odb.steps[disp_step].frames):
        frames.append(frame) # Construct a list of all of the frames

        """ Extract ODB fieldOutputs """
        fieldOutput = odbFrame.fieldOutputs

        # Print the time during the simulation
        print(odbFrame.description)
        time.append(odbFrame.frameValue)

        """ Abaqus Instances (Parts) """
        odbInstance = odb.rootAssembly.instances

        if simplified == False:
            # If Simp is not in the title
            part_E = odbInstance.keys(0)[0]
            part_G = odbInstance.keys(0)[1]
            part_R = odbInstance.keys(0)[2]
            part_T = odbInstance.keys(0)[3]
            part_V = odbInstance.keys(0)[4]

        elif simplified == True:
            # If simplification exists, omit the glue & tab
            part_E = odbInstance.keys(0)[0]
            part_R = odbInstance.keys(0)[1]
            part_V = odbInstance.keys(0)[2]
        else:
            print('Error in part definitions')

        """ Nodal displacements """
        fieldObject_U = fieldOutput['U'] # displacements

        if simplified == False:
            # If Simp is not in the title

            # Glue
            Displacements =
            ↪  fieldObject_U.getSubset(region=odbInstance[part_G].nodeSets['G_RP_SET'])
            for Uyi in Displacements.values: # Loops over each node in the "SET"
            ↪  defined by the displacement
                Uyi_vec = [Uyi.data[0], Uyi.data[1], Uyi.data[2]]
                # Find the magnitude
                temp.append(np.dot(Uyi_vec, LoadCellDirection)) # Creates a list of
                ↪  displacements in the "SET"

            SU = np.sum(temp) # Sums up the list of displacements from the "SET"
            AvgU_top = SU/len(temp) # Divide by the number of nodes in the set to get
            ↪  average
            U_top.append(AvgU_top) # Adds the total displacement to the U-array by
            ↪  summing across each step
            temp = [] # Clear the array for the next iteration in the loop

        elif simplified == True:
            # If simplification exists, omit the values

            Displacements =
            ↪  fieldObject_U.getSubset(region=odbInstance[part_R].nodeSets['R_G_SET'])
```

```
            for Uyi in Displacements.values: # Loops over each node in the "SET" defined by the displacement
                Uyi_vec = [Uyi.data[0], Uyi.data[1], Uyi.data[2]]
                # Find the magnitude
                temp.append(np.dot(Uyi_vec, LoadCellDirection)) # Creates a list of displacements in the "SET"

            SU = np.sum(temp) # Sums up the list of displacements from the "SET"
            AvgU_top = SU/len(temp) # Divide by the number of nodes in the set to get average
            U_top.append(AvgU_top) # Adds the total displacement to the U-array by summing across each step
            temp = [] # Clear the array for the next iteration in the loop

        else:
            print('Error in nodal displacements')

        """ Bond Distance """
        Displacements = fieldObject_U.getSubset(region=odbInstance[part_R].nodeSets['R_V_SET'])
        for Uyi in Displacements.values: # Loops over each node in the "SET" defined by the displacement
            Uyi_vec = [Uyi.data[0], Uyi.data[1], Uyi.data[2]]
            # Find the magnitude
            temp.append(np.dot(Uyi_vec, LoadCellDirection)) # Creates a list of displacements in the "SET"

        SU = np.sum(temp) # Sums up the list of displacements from the "SET"
        AvgR_bot = SU/len(temp) # Divide by the number of nodes in the set to get average
        R_bot.append(AvgR_bot) # Adds the total displacement to the U-array by summing across each step
        temp = [] # Clear the array for the next iteration in the loop

        Displacements = fieldObject_U.getSubset(region=odbInstance[part_V].nodeSets['V_R_SET'])
        for Uyi in Displacements.values: # Loops over each node in the "SET" defined by the displacement
            Uyi_vec = [Uyi.data[0], Uyi.data[1], Uyi.data[2]]
            # Find the magnitude
            temp.append(np.dot(Uyi_vec, LoadCellDirection)) # Creates a list of displacements in the "SET"

        SU = np.sum(temp) # Sums up the list of displacements from the "SET"
        AvgV_top = SU/len(temp) # Divide by the number of nodes in the set to get average
        V_top.append(AvgV_top) # Adds the total displacement to the U-array by summing across each step
        temp = [] # Clear the array for the next iteration in the loop

        # average difference in nodal positions between the *bonded surfaces
        Bond_disp.append(AvgR_bot - AvgV_top)

        if BondStatus == True:
        # if fieldOutput.has_key('CNORMF ASSEMBLY_R-1_R_V_SURF_BOND/ASSEMBLY_V-1_V_R_SURF_BOND') == 1: # if the repository has the item
```

```
            # '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

            # Contact Force
            fieldObject_CNORMF_RV = fieldOutput.keys()[0]
            fieldObject_CNORMF_VR = fieldOutput.keys()[1]

            # Contact Stress
            fieldObject_CPRESS_RV = fieldOutput.keys()[2]
            fieldObject_CPRESS_VR = fieldOutput.keys()[3]

            # Contact Shear1
            fieldObject_CSHEAR1_RV = fieldOutput.keys()[4]
            fieldObject_CSHEAR1_VR = fieldOutput.keys()[5]

            # Contact Shear2
            fieldObject_CSHEAR2_RV = fieldOutput.keys()[6]
            fieldObject_CSHEAR2_VR = fieldOutput.keys()[7]

            # Contact ShearF
            fieldObject_CSHEARF_RV = fieldOutput.keys()[8]
            fieldObject_CSHEARF_VR = fieldOutput.keys()[9]

            """ Contact Force (Retina-Vitreous) """
            # Retina-Vitreous contact stress
            for CnF_RV_i in fieldOutput[fieldObject_CNORMF_RV].values:
                temp.append(CnF_RV_i.data*mult)

            S_CnF_RV = np.sum(temp) # Sums up the list of stress from the "SET"
            CnormF_RV_List.append(temp) # append the list of nodal values
            CnormF_RV.append(S_CnF_RV) # Adds the total stress to the stress-array by summing across each step
            temp = [] # Clear the array for the next iteration in the loop

            """ Contact Force (Vitreous-Retina) """
            # Retina-Vitreous contact stress
            for CnF_VR_i in fieldOutput[fieldObject_CNORMF_VR].values:
                temp.append(CnF_RV_i.data*mult)

            S_CnF_VR = np.sum(temp) # Sums up the list of stress from the "SET"
            CnormF_VR_List.append(temp) # append the list of nodal values
            CnormF_VR.append(S_CnF_VR) # Adds the total stress to the stress-array by summing across each step
            temp = [] # Clear the array for the next iteration in the loop

            """ Contact Stress (Retina-Vitreous) """
            # Retina-Vitreous contact stress
            for CP_RV_i in fieldOutput[fieldObject_CPRESS_RV].values:
                temp.append(CP_RV_i.data*mult)

            S_CP_RV = np.sum(temp) # Sums up the list of stress from the "SET"
            Cpress_RV_List.append(temp)
            Cpress_RV.append(S_CP_RV) # Adds the total stress to the stress-array by summing across each step
            Cpress_RV_AVG.append(S_CP_RV/len(fieldOutput[fieldObject_CPRESS_RV].values))
            temp = [] # Clear the array for the next iteration in the loop
```

```
            """ Contact Stress (Vitreous-Retina) """
            # Retina-Vitreous contact stress
            for CP_VR_i in fieldOutput[fieldObject_CPRESS_VR].values:
                temp.append(CP_VR_i.data*mult)

            S_CP_VR = np.sum(temp) # Sums up the list of stress from the "SET"
            Cpress_VR_List.append(temp)
            Cpress_VR.append(S_CP_VR) # Adds the total stress to the stress-array by summing across each step
            Cpress_VR_AVG.append(S_CP_VR/len(fieldOutput[fieldObject_CPRESS_VR].values))
            temp = [] # Clear the array for the next iteration in the loop

            """ Contact Shear1 (Retina-Vitreous) """
            # Retina-Vitreous contact stress
            for Cs1_RV_i in fieldOutput[fieldObject_CSHEAR1_RV].values:
                temp.append(Cs1_RV_i.data*mult)

            S_Cs1_RV = np.sum(temp) # Sums up the list of stress from the "SET"
            Cshear1_RV_List.append(temp) # append the list of nodal values
            Cshear1_RV.append(S_Cs1_RV) # Adds the total stress to the stress-array by summing across each step
            temp = [] # Clear the array for the next iteration in the loop

            """ Contact Shear1 (Vitreous-Retina) """
            # Retina-Vitreous contact stress
            for Cs1_VR_i in fieldOutput[fieldObject_CSHEAR1_VR].values:
                temp.append(Cs1_VR_i.data*mult)

            S_Cs1_VR = np.sum(temp) # Sums up the list of stress from the "SET"
            Cshear1_VR_List.append(temp) # append the list of nodal values
            Cshear1_VR.append(S_Cs1_VR) # Adds the total stress to the stress-array by summing across each step
            temp = [] # Clear the array for the next iteration in the loop

            """ Contact Shear2 (Retina-Vitreous) """
            # Retina-Vitreous contact stress
            for Cs2_RV_i in fieldOutput[fieldObject_CSHEAR2_RV].values:
                temp.append(Cs2_RV_i.data*mult)

            S_Cs2_RV = np.sum(temp) # Sums up the list of stress from the "SET"
            Cshear2_RV_List.append(temp) # append the list of nodal values
            Cshear2_RV.append(S_Cs2_RV) # Adds the total stress to the stress-array by summing across each step
            temp = [] # Clear the array for the next iteration in the loop

            """ Contact Shear2 (Vitreous-Retina) """
            # Retina-Vitreous contact stress
            for Cs2_VR_i in fieldOutput[fieldObject_CSHEAR2_VR].values:
                temp.append(Cs2_VR_i.data*mult)

            S_Cs2_VR = np.sum(temp) # Sums up the list of stress from the "SET"
            Cshear2_VR_List.append(temp) # append the list of nodal values
            Cshear2_VR.append(S_Cs2_VR) # Adds the total stress to the stress-array by summing across each step
            temp = [] # Clear the array for the next iteration in the loop
```

```
            """ Contact ShearF (Retina-Vitreous) """
            # Retina-Vitreous contact stress
            for CsF_RV_i in fieldOutput[fieldObject_CSHEARF_RV].values:
                temp.append(CsF_RV_i.data*mult)

            S_CsF_RV = np.sum(temp) # Sums up the list of stress from the "SET"
            CshearF_RV_List.append(temp) # append the list of nodal values
            CshearF_RV.append(S_CsF_RV) # Adds the total stress to the stress-array by summing across each step
            temp = [] # Clear the array for the next iteration in the loop

            """ Contact ShearF (Vitreous-Retina) """
            # Retina-Vitreous contact stress
            for CsF_VR_i in fieldOutput[fieldObject_CSHEARF_VR].values:
                temp.append(CsF_VR_i.data*mult)

            S_CsF_VR = np.sum(temp) # Sums up the list of stress from the "SET"
            CshearF_VR_List.append(temp) # append the list of nodal values
            CshearF_VR.append(S_CsF_VR) # Adds the total stress to the stress-array by summing across each step
            temp = [] # Clear the array for the next iteration in the loop

        else:
            # append nans if not available
            CnormF_RV.append(np.nan)
            CnormF_VR.append(np.nan)
            Cpress_RV.append(np.nan)
            Cpress_RV_AVG.append(np.nan)
            Cpress_VR.append(np.nan)
            Cpress_VR_AVG.append(np.nan)
            Cshear1_RV.append(np.nan)
            Cshear1_VR.append(np.nan)
            Cshear2_RV.append(np.nan)
            Cshear2_VR.append(np.nan)
            CshearF_RV.append(np.nan)
            CshearF_VR.append(np.nan)

            CnormF_RV_List.append([np.nan, np.nan, np.nan])
            CnormF_VR_List.append([np.nan, np.nan, np.nan])
            Cpress_RV_List.append([np.nan, np.nan, np.nan])
            Cpress_VR_List.append([np.nan, np.nan, np.nan])
            Cshear1_RV_List.append([np.nan, np.nan, np.nan])
            Cshear1_VR_List.append([np.nan, np.nan, np.nan])
            Cshear2_RV_List.append([np.nan, np.nan, np.nan])
            Cshear2_VR_List.append([np.nan, np.nan, np.nan])
            CshearF_RV_List.append([np.nan, np.nan, np.nan])
            CshearF_VR_List.append([np.nan, np.nan, np.nan])

            print('No CPRESS... ** Updating with NANs')

        """ Contact Node Lists """
        R_V_SetNodeNames = []
        V_R_SetNodeNames = []
        for i, NodeLabeli in enumerate(odbInstance[part_R].nodeSets['R_V_SET'].nodes):
            R_V_SetNodeNames.append(NodeLabeli.label)

```

```
        for i, NodeLabeli in enumerate(odbInstance[part_V].nodeSets['V_R_SET'].nodes):
            V_R_SetNodeNames.append(NodeLabeli.label)

        """ Reaction forces """
        # fieldObject_RF = fieldOutput['RF'] # reaction forces
        # # Glue-Retina RP set-forces
        # Reaction_Forces = fieldObject_RF.getSubset(region=odbInstance[part_G].nodeSets['G_RP_SET'])
        # for RFi in Reaction_Forces.values: # Loops over each node in the "SET" defined by the reaction force
        #     RFi_vec = [RFi.data[0], RFi.data[1], RFi.data[2]]
        #     # Find the component in the direction of the load cell
        #     temp.append(np.dot(RFi_vec, LoadCellDirection)) # Creates a list of reaction forces in the "SET"

        # SRF = np.sum(temp) # Sums up the list of reaction forces from the "SET"
        # RF.append(SRF) # Adds the total reaction force to the RF-array by summing across each step
        # temp = [] # Clear the array for the next iteration in the loop

        fieldObject_RF = fieldOutput['RF'] # reaction forces
        if simplified == False:
            # If Simp is not in the title

            # Glue-Retina G_RP_Set Reaction forces
            Reaction_Forces = fieldObject_RF.getSubset(region=odbInstance[part_G].nodeSets['G_RP_SET'])

        elif simplified == True:

            # Retina R_G_Set Reaction forces
            Reaction_Forces = fieldObject_RF.getSubset(region=odbInstance[part_R].nodeSets['R_G_SET'])

        else:
            print('Error in RF output')

        for RFi in Reaction_Forces.values: # Loops over each node in the "SET" defined by the reaction force
            RFxi = RFi.data[0]
            RFyi = RFi.data[1]
            RFzi = RFi.data[2]
            RFi_vec = [RFxi, RFyi, RFzi]

            # Find the component in the direction of the load cell
            temp.append(np.dot(RFi_vec, LoadCellDirection)*mult) # Creates a list of reaction forces in the "SET"
            tempx.append(RFxi) # X reaction forces along the R_G_SET
            tempy.append(RFyi) # Y reaction forces along the R_G_SET
            tempz.append(RFzi) # Z reaction forces along the R_G_SET

        SRF = np.sum(temp) # Sums up the list of reaction forces from the "SET"
        RF.append(SRF) # Adds the total reaction force to the RF-array by summing across each step
        temp = [] # Clear the array for the next iteration in the loop

        SRFX = np.sum(tempx)
```

```
        RFx.append(SRFX)

        SRFY = np.sum(tempy)
        RFy.append(SRFY)

        SRFZ = np.sum(tempz)
        RFz.append(SRFZ)

        """ Nodal Forces """
        # Forces at the nodes of an element from both the hourglass and the regular
        # deformation modes of that element (negative of the internal forces in
        # the global coordinate system). The specified position in data and results
        # file requests is ignored.

        if fieldOutput.has_key('NFORC1') == 1: # Searches if the repository has the value
            fieldObject_NFORC1 = fieldOutput['NFORC1'] # Normal force 1
            fieldObject_NFORC2 = fieldOutput['NFORC2'] # Normal force 2
            fieldObject_NFORC3 = fieldOutput['NFORC3'] # Normal force 3

            # Retina nodal forces on the glue interface
            nodeSet_R_G_SET = odbInstance[part_R].nodeSets['R_G_SET']
            NF1 = fieldObject_NFORC1.getSubset(region=nodeSet_R_G_SET)
            NF2 = fieldObject_NFORC2.getSubset(region=nodeSet_R_G_SET)
            NF3 = fieldObject_NFORC3.getSubset(region=nodeSet_R_G_SET)

            for NFi in range(len(NF1.values)): # Loops over each node in the "SET" defined by the reaction force
                NFi_vec = [NF1.values[NFi].data, NF2.values[NFi].data, NF3.values[NFi].data]
                NFi_veclabel = [NF1.values[NFi].nodeLabel, NF1.values[NFi].data, NF2.values[NFi].nodeLabel, NF2.values[NFi].data, NF3.values[NFi].nodeLabel, NF3.values[NFi].data]
                # Find the component in the direction of the load cell
                temp.append(np.dot(NFi_vec, LoadCellDirection)*mult) # Creates a list of reaction forces in the "SET"

            SNf = np.sum(temp) # Sums up the list of reaction forces from the "SET"
            Nforc.append(SNf*-1) # Adds the total reaction force to the RF-array by summing across each step (negative indicates the direction, which is opposite of tension when -1)
            temp = [] # Clear the array for the next iteration in the loop
        else:
            Nforc.append(0)
            print('No NFORC... ** Updating with 0')

    """ Loop over the history outputs"""
    # odb.steps[disp_step].historyRegions.keys() List all of the items in the dictionary
    odbHistoryRegion = odb.steps[disp_step].historyRegions
    odbHistAssem = 'Assembly ASSEMBLY'
    Assembly = odbHistoryRegion[odbHistAssem]

    # Energy output
    ALLIE_KE = Assembly.historyOutputs.keys()[0]
    Hist_ELEM = Assembly.historyOutputs.keys()[1]
    Whole_Model_Energy = Assembly.historyOutputs
    Internal_Energy = Whole_Model_Energy.keys()[0] # Internal energy
```

```python
    Kinetic_Energy = Whole_Model_Energy.keys()[1] # Kintic energy
    for i, j in enumerate(Whole_Model_Energy[Internal_Energy].data):
        Hist_Time.append(j[0]) # History Output Time Array
        IE.append(j[1]) # Internal Energy
        KE.append(Whole_Model_Energy[Kinetic_Energy].data[i][1]) # Kinetic Energy

    if BondStatus == True:
    # if jobName.find('VRTie') == -1:
        # Figure out how to extract these for each node in the connected set
        """ Bond Loads """
        # This is an array of bond load per node
        V_R_SetNodeLength = len(odbInstance[part_V].nodeSets['V_R_SET'].nodes) # length of the V_R_Set node list
        BondNodeNames = odbHistoryRegion.keys()[-V_R_SetNodeLength:]

        # array of bond status and bond load
        BondStat = []
        BondLoad = []

        # used for iterating and clearing
        temp1 = []
        temp2 = []

        # loop over the length of the BondStat/BondLoad list
        for m,n in enumerate(odbHistoryRegion[BondNodeNames[0]].historyOutputs['BONDSTAT'].data):
            #  loop over the length of the bond node list and append each time step
            for i,BondNodeNames_i in enumerate(BondNodeNames):
                temp1.append(odbHistoryRegion[BondNodeNames_i].historyOutputs['BONDSTAT'].data[m][1])
                temp2.append(odbHistoryRegion[BondNodeNames_i].historyOutputs['BONDLOAD'].data[m][1]*mu

            # build the arrays for BondStat/BondLoad
            BondStat.append(temp1)
            BondLoad.append(temp2)

            # clear the arrays
            temp1 = []
            temp2 = []
        else:
            print('No bonding,VR interface is tied')

    # # Contact
    # odbHistElementSetPIBATCH = odbHistoryRegion.keys()[1] # ElementSet PIBATCH
    # elementSetPIBATCH = odbHistoryRegion[odbHistElementSetPIBATCH]
    # eC = elementSetPIBATCH.historyOutputs # Element contact

    # if jobName.find('Si') == -1:
    #     # If Simp is not in the title
    #     cAreaCP_RG = eC.keys()[0]
    #     cAreaCP_RV = eC.keys()[1]
    #     cAreaCP_GR = eC.keys()[2]
    #     cAreaCP_VR = eC.keys()[3]
    #     CFN1CP_RG = eC.keys()[4]
    #     CFN1CP_RV = eC.keys()[5]
    #     CFN1CP_GR = eC.keys()[6]
```

```
    #     CFN1CP_VR = eC.keys()[7]
    #     CFN2CP_RG = eC.keys()[8]
    #     CFN2CP_RV = eC.keys()[9]
    #     CFN2CP_GR = eC.keys()[10]
    #     CFN2CP_VR = eC.keys()[11]
    #     CFN3CP_RG = eC.keys()[12]
    #     CFN3CP_RV = eC.keys()[13]
    #     CFN3CP_GR = eC.keys()[14]
    #     CFN3CP_VR = eC.keys()[15]

    # elif jobName.find('Si') >= 0:
    #     # If simplification omit the tab and glue
    #     cAreaCP_RV = eC.keys()[0]
    #     cAreaCP_VR = eC.keys()[1]
    #     CFN1CP_RV = eC.keys()[2]
    #     CFN1CP_VR = eC.keys()[3]
    #     CFN2CP_RV = eC.keys()[4]
    #     CFN2CP_VR = eC.keys()[5]
    #     CFN3CP_RV = eC.keys()[6]
    #     CFN3CP_VR = eC.keys()[7]
    # else:
    #     print('Error in Hist Output Names')
    #Bond_Nodes = energyHistRegion.historyOutputs.keys()[2:-1]

    # Glue Reference point
    if simplified == False:
        # If Simp is not in the title

        odbHist_gRP = odbHistoryRegion.keys()[1]
        gRP_Hist = odbHistoryRegion[odbHist_gRP]
        gRP_Hist = gRP_Hist.historyOutputs
        gRP_HistRF1 = gRP_Hist.keys()[0]
        gRP_HistRF2 = gRP_Hist.keys()[1]
        gRP_HistRF3 = gRP_Hist.keys()[2]
        gRP_HistU1 = gRP_Hist.keys()[6]
        gRP_HistU2 = gRP_Hist.keys()[7]
        gRP_HistU3 = gRP_Hist.keys()[8]

    elif simplified == True:
        # If simplification, omit the tab and glue
        print('Simplification')
    else:
        print('Error in simplification')

    # for i,j in enumerate(Internal_Energy.data):
    #     Hist_Time.append(j[0]) # History Output Time Array

    #     # Energy array
    #     IE.append(j[1]*mult) # Internal Energy
    #     KE.append(Kinetic_Energy.data[i][1]*mult) # Kinetic Energy

    #     if jobName.find('Si') == -1:
    #         # If Simp is not in the title

    #         # Contact area arrays, not sure if these need to be multiplied by 2 (Check .ODB)
    #         CAreaCP_RG.append(eC[cAreaCP_RG].data[i][1])
    #         CAreaCP_GR.append(eC[cAreaCP_GR].data[i][1])
```

```
#         CAreaCP_RV.append(eC[cAreaCP_RV].data[i][1])
#         CAreaCP_VR.append(eC[cAreaCP_VR].data[i][1])

#         # Create a vector for CP RG
#         CFNCP_RG_vec = [eC[CFN1CP_RG].data[i][1], eC[CFN2CP_RG].data[i][1],
↪ eC[CFN3CP_RG].data[i][1]]
#         # Find the component in the direction of the load cell
#         CFNCP_RG.append(np.dot(CFNCP_RG_vec, LoadCellDirection)*mult)

#         # Create a vector for CP GR
#         CFNCP_GR_vec = [eC[CFN1CP_GR].data[i][1], eC[CFN2CP_GR].data[i][1],
↪ eC[CFN3CP_GR].data[i][1]]
#         # Find the component in the direction of the load cell
#         CFNCP_GR.append(np.dot(CFNCP_GR_vec, LoadCellDirection)*mult)

#         # Create a vector for CP RV
#         CFNCP_RV_vec = [eC[CFN1CP_RV].data[i][1], eC[CFN2CP_RV].data[i][1],
↪ eC[CFN3CP_RV].data[i][1]]
#         # Find the component in the direction of the load cell
#         CFNCP_RV.append(np.dot(CFNCP_RV_vec, LoadCellDirection)*mult)

#         # Create a vector for CP VR
#         CFNCP_VR_vec = [eC[CFN1CP_VR].data[i][1], eC[CFN2CP_VR].data[i][1],
↪ eC[CFN3CP_VR].data[i][1]]
#         # Find the component in the direction of the load cell
#         CFNCP_VR.append(np.dot(CFNCP_VR_vec, LoadCellDirection)*mult)

#         # Create a vector for the Glue Reference point
#         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]]
#         # Find the component in the direction of the load cell
#         Glue_RP_RF.append(np.dot(Glue_RP_RF_vec, LoadCellDirection)*mult)

#     elif jobName.find('Si') >= 0:
#         # Contact area arrays
#         CAreaCP_RG.append(np.nan)
#         CAreaCP_GR.append(np.nan)
#         CAreaCP_RV.append(eC[cAreaCP_RV].data[i][1])
#         CAreaCP_VR.append(eC[cAreaCP_VR].data[i][1])

#         # Create a vector for CP RG
#         CFNCP_RG_vec = [np.nan, np.nan, np.nan]
#         # Find the component in the direction of the load cell
#         CFNCP_RG.append(np.dot(CFNCP_RG_vec, LoadCellDirection)*mult)

#         # Create a vector for CP GR
#         CFNCP_GR_vec = [np.nan, np.nan, np.nan]
#         # Find the component in the direction of the load cell
#         CFNCP_GR.append(np.dot(CFNCP_GR_vec, LoadCellDirection)*mult)

#         # Create a vector for CP RV
#         CFNCP_RV_vec = [eC[CFN1CP_RV].data[i][1], eC[CFN2CP_RV].data[i][1],
↪ eC[CFN3CP_RV].data[i][1]]
#         # Find the component in the direction of the load cell
#         CFNCP_RV.append(np.dot(CFNCP_RV_vec, LoadCellDirection)*mult)

#         # Create a vector for CP VR
```

```
#         CFNCP_VR_vec = [eC[CFN1CP_VR].data[i][1], eC[CFN2CP_VR].data[i][1],
    ↪  eC[CFN3CP_VR].data[i][1]]
#         # Find the component in the direction of the load cell
#         CFNCP_VR.append(np.dot(CFNCP_VR_vec, LoadCellDirection)*mult)

#         # Create a vector for the Glue Reference point
#         Glue_RP_RF_vec = [np.nan, np.nan, np.nan]
#         # Find the component in the direction of the load cell
#         Glue_RP_RF.append(np.dot(Glue_RP_RF_vec, LoadCellDirection)*mult)
#     else:
#         print('Error in hist output data with simplification')

# if jobName.find('VRTie') == -1: # if VRTie is not in the title
#     """ Bond Loads """
#     # This is an array of bond load per node
#     V_R_SetNodeLength = len(odbInstance[part_V].nodeSets['V_R_SET'].nodes) #
    ↪  length of the V_R_Set node list
#     BondNodeNames = odbHistoryRegion.keys()[-V_R_SetNodeLength:]

#     # array of bond status and bond load
#     BondStat = []
#     BondLoad = []

#     # used for iterating and clearing
#     temp1 = []
#     temp2 = []

#     # loop over the length of the BondStat/BondLoad list
#     for m,n in
    ↪  enumerate(odbHistoryRegion[BondNodeNames[0]].historyOutputs['BONDSTAT'].data):
#         #  loop over the length of the bond node list and append each time step
#         for i,BondNodeNames_i in enumerate(BondNodeNames):
#
    ↪  temp1.append(odbHistoryRegion[BondNodeNames_i].historyOutputs['BONDSTAT'].data[n][1])
#
    ↪  temp2.append(odbHistoryRegion[BondNodeNames_i].historyOutputs['BONDLOAD'].data[n][1])

#         # build the arrays for BondStat/BondLoad
#         BondStat.append(temp1)
#         BondLoad.append(temp2)

#         # clear the arrays
#         temp1 = []
#         temp2 = []
#     else:
#         print('No bond because the VR interface is tied')

""" Specify folder name where the files go..."""
folderName = jobName
folder_sub_directory = 'Output'

""" Print the odbFieldOutput Data """
print("\nWriting out the load data...")
filename = os.path.join(folderName, folder_sub_directory, 'output_Field_'
                        + jobName + '.txt')
outfile = open(filename,'w')

Header = [] # Header information for the dataframe
```

```python
    Header.append('frame')
    Header.append('Time [s]')
    Header.append('Reaction force dotted in y direction [N]')
    Header.append('Reaction force X [N]')
    Header.append('Reaction force Y [N]')
    Header.append('Reaction force Z [N]')
    Header.append('Sum NForc (Glue-Retina Set) [N]')
    Header.append('CnormF_RV [N]')
    Header.append('CnormF_VR [N]')
    Header.append('Cpress_RV [Pa]')
    Header.append('Cpress_VR [Pa]')
    Header.append('AVG_Cpress_RV_AVG [Pa]')
    Header.append('AVG_Cpress_VR_AVG [Pa]')
    Header.append('Cshear1_RV [Pa]')
    Header.append('Cshear1_VR [Pa]')
    Header.append('Cshear2_RV [Pa]')
    Header.append('Cshear2_VR [Pa]')
    Header.append('CshearF_RV [Pa]')
    Header.append('CshearF_VR [Pa]')
    Header.append('Glue Displacements [m]')
    Header.append('Bond Displacements [m]')
    lineWrite = '\t'.join(str(item) for item in Header)
    outfile.write(lineWrite)

    for i in frames:

        lineNums = []
        lineNums.append(time[i])
        lineNums.append(RF[i])
        lineNums.append(RFx[i])
        lineNums.append(RFy[i])
        lineNums.append(RFz[i])
        lineNums.append(Nforc[i])
        lineNums.append(CnormF_RV[i])
        lineNums.append(CnormF_VR[i])
        lineNums.append(Cpress_RV[i])
        lineNums.append(Cpress_VR[i])
        lineNums.append(Cpress_RV_AVG[i])
        lineNums.append(Cpress_VR_AVG[i])
        lineNums.append(Cshear1_RV[i])
        lineNums.append(Cshear1_VR[i])
        lineNums.append(Cshear2_RV[i])
        lineNums.append(Cshear2_VR[i])
        lineNums.append(CshearF_RV[i])
        lineNums.append(CshearF_VR[i])
        lineNums.append(U_top[i])
        lineNums.append(Bond_disp[i])

        # format the list to have a float with twenty decimal places
        # Add floats
        formatted_list = ['{:.20f}'.format(item) for item in lineNums]
        line = '\n' + '{}\t'.format(i) + '\t'.join(str(item) for item in
                                                    formatted_list)
        outfile.write(line)

    outfile.close()

    print("\nDone!")
```

```
    print("\nThe output file will be named '{}".format(filename) + "'")
    print("\nIt will be in the same working directory as your Abaqus model\n")

    """ Print the odbHistoryOutput Data """
    print("\nWriting out the History Output data...")
    filename = os.path.join(folderName, 'Output', 'output_History_' +
                            jobName + '.txt')
    outfile = open(filename, 'w')

    Header = []
    Header.append('frame')
    Header.append('Time [s]')
    Header.append('Internal Energy [J]')
    Header.append('Kinetic Energy [J]')
    lineWrite = '\t'.join(str(item) for item in Header)
    outfile.write(lineWrite)

    for i, j in enumerate(Hist_Time):
        line = []
        line.append('{}'.format(i)) # Integer for frame number
        line.append('{:.10f}'.format(j))
        line.append('{:.30f}'.format(IE[i]))
        line.append('{:.30f}'.format(KE[i]))
        lineWrite = '\n' + '\t'.join(str(item) for item in line)
        outfile.write(lineWrite)

    outfile.close()

    print("\nDone!")
    print("\nThe output file will be named '{}".format(filename) + "'")
    print("\nIt will be in the same working directory as your Abaqus model\n")

    if BondStatus == True:
    # if jobName.find('VRTie') == -1: # if VRTie is not in the jobName
        """ Cube Info Plots """
        """ Print the odbHistoryOutput BondStat Data """
        print("\nWriting out the History Output Bond data...")
        filename = os.path.join(folderName, 'Output', 'BONDSTAT_' + jobName + '.txt')
        outfile = open(filename,'w')
        outfile.write('Time (s)\t' + '\t'.join(str(item) for item in BondNodeNames))
        for i,j in enumerate(BondStat):
            outfile.write('\n')
            tempList = [Hist_Time[i]]
            for k in list(j):
                tempList.append(k)
            outfile.write('\t'.join(str(item) for item in tempList))
        outfile.close()
        print("\nDone!")
        print("\nThe output file will be named '{}".format(filename) + "'")
        print("\nIt will be in the same working directory as your Abaqus model\n")

        """ Print the odbHistoryOutput BondLoad Data """
        print("\nWriting out the History Output Bond data...")
        filename = os.path.join(folderName, 'Output', 'BONDLOAD_' + jobName + '.txt')
        outfile = open(filename,'w')
        outfile.write('Time (s)\t' + '\t'.join(str(item) for item in BondNodeNames))
        for i,j in enumerate(BondLoad):
            outfile.write('\n')
```

```
            tempList = [Hist_Time[i]]
            for k in list(j):
                tempList.append(k)
            outfile.write('\t'.join(str(item) for item in tempList))
        outfile.close()
        print("\nDone!")
        print("\nThe output file will be named '{}".format(filename) + "'")
        print("\nIt will be in the same working directory as your Abaqus model\n")

        """ Print the odbFieldOutput CnormF_RV Data """
        print("\nWriting out the Field Output CnormF_RV data...")
        filename = os.path.join(folderName, 'Output', 'CnormF_RV_' + jobName + '.txt')
        outfile = open(filename,'w')
        xyz = ['X','Y','Z']
        header = []
        for R_V_SetXYZi in list(R_V_SetNodeNames):
            for m in range(3):
                header.append('R-' + str(R_V_SetXYZi) + xyz[m])
        outfile.write('Time (s)\t' + '\t'.join(item for item in header))
        for i,Nodei in enumerate(CnormF_RV_List):
            outfile.write('\n')
            tempList = [time[i]]
            for XYZ in list(Nodei):
                for XYZi in list(XYZ):
                    tempList.append(XYZi)
            outfile.write('\t'.join(str(item) for item in tempList))
        outfile.close()
        print("\nDone!")
        print("\nThe output file will be named '{}".format(filename) + "'")
        print("\nIt will be in the same working directory as your Abaqus model\n")

        """ Print the odbFieldOutput CnormF_VR Data """
        print("\nWriting out the Field Output CnormF_VR data...")
        filename = os.path.join(folderName, 'Output', 'CnormF_VR_' + jobName + '.txt')
        outfile = open(filename,'w')
        xyz = ['X','Y','Z']
        header = []
        for V_R_SetXYZi in list(V_R_SetNodeNames):
            for m in range(3):
                header.append('V-' + str(V_R_SetXYZi) + xyz[m])
        outfile.write('Time (s)\t' + '\t'.join(item for item in header))
        for i,Nodei in enumerate(CnormF_VR_List):
            outfile.write('\n')
            tempList = [time[i]]
            for XYZ in list(Nodei):
                for XYZi in list(XYZ):
                    tempList.append(XYZi)
            outfile.write('\t'.join(str(item) for item in tempList))
        outfile.close()
        print("\nDone!")
        print("\nThe output file will be named '{}".format(filename) + "'")
        print("\nIt will be in the same working directory as your Abaqus model\n")

        """ Print the odbFieldOutput Cpress_RV Data """
        print("\nWriting out the Field Output Cpress_RV data...")
```

```python
        filename = os.path.join(folderName, 'Output', 'Cpress_RV_' + jobName + '.txt')
        outfile = open(filename,'w')
        header = []
        for R_V_SetXYZi in list(R_V_SetNodeNames):
            header.append('R-' + str(R_V_SetXYZi))
        outfile.write('Time (s)\t' + '\t'.join(item for item in header))
        for i,Nodei in enumerate(Cpress_RV_List):
            outfile.write('\n')
            tempList = [time[i]]
            for ni in list(Nodei):
                tempList.append(ni)
            outfile.write('\t'.join(str(item) for item in tempList))
        outfile.close()
        print("\nDone!")
        print("\nThe output file will be named '{}".format(filename) + "'")
        print("\nIt will be in the same working directory as your Abaqus model\n")

        """ Print the odbFieldOutput Cpress_VR Data """
        print("\nWriting out the Field Output Cpress_VR data...")
        filename = os.path.join(folderName, 'Output', 'Cpress_VR_' + jobName + '.txt')
        outfile = open(filename,'w')
        header = []
        for V_R_SetXYZi in list(V_R_SetNodeNames):
            header.append('V-' + str(V_R_SetXYZi))
        outfile.write('Time (s)\t' + '\t'.join(item for item in header))
        for i,Nodei in enumerate(Cshear1_RV_List):
            outfile.write('\n')
            tempList = [time[i]]
            for ni in list(Nodei):
                tempList.append(ni)
            outfile.write('\t'.join(str(item) for item in tempList))
        outfile.close()
        print("\nDone!")
        print("\nThe output file will be named '{}".format(filename) + "'")
        print("\nIt will be in the same working directory as your Abaqus model\n")

        """ Print the odbFieldOutput Cshear1_RV Data """
        print("\nWriting out the Field Output Cshear1_RV data...")
        filename = os.path.join(folderName, 'Output', 'Cshear1_RV_' + jobName + '.txt')
        outfile = open(filename,'w')
        header = []
        for R_V_SetXYZi in list(R_V_SetNodeNames):
            header.append('R-' + str(R_V_SetXYZi))
        outfile.write('Time (s)\t' + '\t'.join(item for item in header))
        for i,Nodei in enumerate(Cshear1_RV_List):
            outfile.write('\n')
            tempList = [time[i]]
            for ni in list(Nodei):
                tempList.append(ni)
            outfile.write('\t'.join(str(item) for item in tempList))
        outfile.close()
        print("\nDone!")
        print("\nThe output file will be named '{}".format(filename) + "'")
        print("\nIt will be in the same working directory as your Abaqus model\n")
```

```
        """ Print the odbFieldOutput Cshear1_VR Data """
        print("\nWriting out the Field Output Cshear1_VR data...")
        filename = os.path.join(folderName, 'Output', 'Cshear1_VR_' + jobName + '.txt')
        outfile = open(filename,'w')
        header = []
        for V_R_SetXYZi in list(V_R_SetNodeNames):
            header.append('V-' + str(V_R_SetXYZi))
        outfile.write('Time (s)\t' + '\t'.join(item for item in header))
        for i,Nodei in enumerate(Cshear1_VR_List):
            outfile.write('\n')
            tempList = [time[i]]
            for ni in list(Nodei):
                tempList.append(ni)
            outfile.write('\t'.join(str(item) for item in tempList))
        outfile.close()
        print("\nDone!")
        print("\nThe output file will be named '{}".format(filename) + "'")
        print("\nIt will be in the same working directory as your Abaqus model\n")

        """ Print the odbFieldOutput Cshear2_RV Data """
        print("\nWriting out the Field Output Cshear2_RV data...")
        filename = os.path.join(folderName, 'Output', 'Cshear2_RV_' + jobName + '.txt')
        outfile = open(filename,'w')
        header = []
        for R_V_SetXYZi in list(R_V_SetNodeNames):
            header.append('R-' + str(R_V_SetXYZi))
        outfile.write('Time (s)\t' + '\t'.join(item for item in header))
        for i,Nodei in enumerate(Cshear2_RV_List):
            outfile.write('\n')
            tempList = [time[i]]
            for ni in list(Nodei):
                tempList.append(ni)
            outfile.write('\t'.join(str(item) for item in tempList))
        outfile.close()
        print("\nDone!")
        print("\nThe output file will be named '{}".format(filename) + "'")
        print("\nIt will be in the same working directory as your Abaqus model\n")

        """ Print the odbFieldOutput Cshear2_VR Data """
        print("\nWriting out the Field Output Cshear2_VR data...")
        filename = os.path.join(folderName, 'Output', 'Cshear2_VR_' + jobName + '.txt')
        outfile = open(filename,'w')
        header = []
        for V_R_SetXYZi in list(V_R_SetNodeNames):
            header.append('V-' + str(V_R_SetXYZi))
        outfile.write('Time (s)\t' + '\t'.join(item for item in header))
        for i,Nodei in enumerate(Cshear2_VR_List):
            outfile.write('\n')
            tempList = [time[i]]
            for ni in list(Nodei):
                tempList.append(ni)
            outfile.write('\t'.join(str(item) for item in tempList))
        outfile.close()
        print("\nDone!")
        print("\nThe output file will be named '{}".format(filename) + "'")
```

```
        print("\nIt will be in the same working directory as your Abaqus model\n")

        """ Print the odbFieldOutput CshearF_RV Data """
        print("\nWriting out the Field Output CshearF_RV data...")
        filename = os.path.join(folderName, 'Output', 'CshearF_RV_' + jobName + '.txt')
        outfile = open(filename,'w')
        xyz = ['X','Y','Z']
        header = []
        for R_V_SetXYZi in list(R_V_SetNodeNames):
            for m in range(3):
                header.append('R-' + str(R_V_SetXYZi) + xyz[m])
        outfile.write('Time (s)\t' + '\t'.join(item for item in header))
        for i,Nodei in enumerate(CshearF_RV_List):
            outfile.write('\n')
            tempList = [time[i]]
            for XYZ in list(Nodei):
                for XYZi in list(XYZ):
                    tempList.append(XYZi)
            outfile.write('\t'.join(str(item) for item in tempList))
        outfile.close()
        print("\nDone!")
        print("\nThe output file will be named '{}".format(filename) + "'")
        print("\nIt will be in the same working directory as your Abaqus model\n")

        """ Print the odbFieldOutput CshearF_VR Data """
        print("\nWriting out the Field Output CshearF_VR data...")
        filename = os.path.join(folderName, 'Output', 'CshearF_VR_' + jobName + '.txt')
        outfile = open(filename,'w')
        xyz = ['X','Y','Z']
        header = []
        for V_R_SetXYZi in list(V_R_SetNodeNames):
            for m in range(3):
                header.append('V-' + str(V_R_SetXYZi) + xyz[m])
        outfile.write('Time (s)\t' + '\t'.join(item for item in header))
        for i,Nodei in enumerate(CshearF_VR_List):
            outfile.write('\n')
            tempList = [time[i]]
            for XYZ in list(Nodei):
                for XYZi in list(XYZ):
                    tempList.append(XYZi)
            outfile.write('\t'.join(str(item) for item in tempList))
        outfile.close()
        print("\nDone!")
        print("\nThe output file will be named '{}".format(filename) + "'")
        print("\nIt will be in the same working directory as your Abaqus model\n")

    else:
        print('No bonding because the VR Interface is tied')
    return

# Run the function
data_extract(jobName)
```

1.5.5 Plotting Script

</> **Script 8:** *Python script used to create plots for each simulation.* </>

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 03 11:58:40 2020

@author: Kiffer Creveling
python3
"""
# Packages & path folder
#from sys import argv, exit
#sys.path.append(r'F:\Abaqus Working Directory')
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import cm
import matplotlib.patheffects as pe
import numpy as np
import os
import os.path
import sys
import pdb
plt.rcParams['figure.figsize'] = [16, 9]

# jobName = sys.argv[1] # Extract the jobName from the previous script

#if __name__ == '__main__':
def plot_Field_Output(fileName, dataDirectory, dataCompare, BondStatus, PDFMStatus):

    """ Field Output Data """
    df = pd.read_csv(os.path.join(dataDirectory,fileName), sep="\t", header=0)

    Header = [] # Header information for the dataframe
    Header.append('Frame')
    Header.append('Time')
    Header.append('RF_y_dot')
    Header.append('RFx')
    Header.append('RFy')
    Header.append('RFz')
    Header.append('Nodal_Force')
    Header.append('CnormF_RV')
    Header.append('CnormF_VR')
    Header.append('Cpress_RV')
    Header.append('Cpress_VR')
    Header.append('AVG_Cpress_RV_AVG')
    Header.append('AVG_Cpress_VR_AVG')
    Header.append('Cshear1_RV')
    Header.append('Cshear1_VR')
    Header.append('Cshear2_RV')
    Header.append('Cshear2_VR')
    Header.append('CshearF_RV')
    Header.append('CshearF_VR')
    Header.append('Glue_Displacements')
    Header.append('Bond_Displacements')

    df.columns = Header
```

```

    t = df.Time
    RF = df.RF_y_dot*1e3 # Convert from N to mN
    NF = df.Nodal_Force*1e3 # Convert from N to mN
    CnF_RV = df.CnormF_RV*1e3 # Convert from N to mN
    CnF_VR = df.CnormF_VR*1e3 # Convert from N to mN
    Cp_RV = df.Cpress_RV
    Cp_VR = df.Cpress_VR
    AVG_Cp_RV = df.AVG_Cpress_RV_AVG
    AVG_Cp_VR = df.AVG_Cpress_VR_AVG
    Cs1_RV = df.Cshear1_RV*1e3 # Convert from N to mN
    Cs1_VR = df.Cshear1_VR*1e3 # Convert from N to mN
    Cs2_RV = df.Cshear2_RV*1e3 # Convert from N to mN
    Cs2_VR = df.Cshear2_VR*1e3 # Convert from N to mN
    CsF_RV = df.CshearF_RV*1e3 # Convert from N to mN
    CsF_VR = df.CshearF_VR*1e3 # Convert from N to mN
    TD = df.Glue_Displacements*1e3 # Convert from m to mm
    BD = df.Bond_Displacements*1e3 # Convert from m to mm

    tabArea = 0.00002247 # m^2 (Used in the hand calc - not used anymore)

    (figureName, ext) = os.path.splitext(fileName) # Split the file extension

    """ Read in the csv file """
    dfValsn = pd.read_csv(dataCompare, sep="\t", nrows=22, header=None,
    ↪  names=['Var','Attribute'])

    """ File Attributes """
    HID =            dfValsn['Attribute'][0]
    HAGE =           dfValsn['Attribute'][1]
    HG =             dfValsn['Attribute'][2]
    HLR =            dfValsn['Attribute'][3]
    HR =             dfValsn['Attribute'][4]
    HSSi =     float(dfValsn['Attribute'][12])
    HSSf =     float(dfValsn['Attribute'][13])
    HTMax =    float(dfValsn['Attribute'][14])
    HDispMax = float(dfValsn['Attribute'][15])
    HFMax =    float(dfValsn['Attribute'][16]) # (mN)
    HFSS =     float(dfValsn['Attribute'][17])
    HSlope20 = float(dfValsn['Attribute'][20]) # (mN/m) slope from 20 seconds prior
    ↪  to max force value

    dfn = pd.read_csv(os.path.join(dataCompare), sep="\t", header=30)
    dfn.columns = ['Time', 'Extension', 'Force']
    dfn_time = dfn.Time
    dfn_extension = dfn.Extension # mm
    dfn_force = dfn.Force*1e3 # convert from N to mN

    # SS Array
    ssTimeArray = np.array([HSSi, HSSf])
    ssValArray = np.array([HFSS, HFSS])

    # Max peel force displacement at max and steady state
    dfn_max_Disp = dfn_extension[dfn_time == HTMax]
    dfn_ss_Disp = np.array([dfn_extension[dfn_time == HSSi].values[0],
    ↪  dfn_extension[dfn_time == HSSf].values[0]])

```

```python
    # Plot the data trace to compare the simulated results with the force displacement curves
    plt.plot(dfn_extension, dfn_force,'-', color='r', linewidth=1, markersize=2, label = '{}, Age: {}'.format(HID, HAGE))
    if str(HFMax) == 'nan' and str(HSSi) == 'nan':
        print('No max or steady state')
        pass

    if str(HFMax) != 'nan':
        plt.plot(dfn_max_Disp, HFMax,'.', color='k', linewidth=1, markersize=20, label = 'Max Peel - {:.4f} (mN)'.format(HFMax), path_effects=[pe.Stroke(linewidth=4, foreground='k'), pe.Normal()])

    if str(HSSi) != 'nan':
        plt.plot(dfn_ss_Disp, ssValArray,'-', color='c', linewidth=3, markersize=2, label = 'Steady State - {:.4f} (mN)'.format(HFSS), path_effects=[pe.Stroke(linewidth=5, foreground='k'), pe.Normal()])

    """ Plots """
    ################## Plot Data #########################
    plt.plot(TD, RF,'-',color='blue',linewidth=2,markersize=2,label = r'Simulated Reaction force $\Sigma F_{Retina}$')
    plt.xlabel('Displacement (mm)',fontsize=18)
    plt.ylabel('Force (mN)',fontsize=18)
    plt.title('Vitreous',fontsize=20)
    plt.grid()
    plt.legend(loc = 'best',fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName + '_RF_vs_Disp.png'),dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    # Plot the data trace to compare the simulated results
    plt.plot(dfn_time, dfn_force,'-', color='r', linewidth=1, markersize=2, label = '{}, Age: {}'.format(HID, HAGE))
    if str(HFMax) == 'nan' and str(HSSi) == 'nan':
        print('No max or steady state')
        pass

    if str(HFMax) != 'nan':
        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()])

    if str(HSSi) != 'nan':
        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()])

    """ Plots """
    ################## Plot Data #########################
    plt.plot(t, RF,'-',color='blue',linewidth=2,markersize=2,label = r'Simulated Reaction force $\Sigma F_{Retina}$')
    plt.xlabel('Time (sec)',fontsize=18)
    plt.ylabel('Force (mN)',fontsize=18)
    plt.title('Vitreous',fontsize=20)
    plt.grid()
    plt.legend(loc = 'best',fontsize = 'medium')
```

```
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
    ↪  '_RF_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    """ Sum Nodal Force Reaction force """
    ################## Plot Data #########################
    plt.plot(t, NF,'-',color='blue',linewidth=2,markersize=2,label = 'Reaction
    ↪  force')
    plt.xlabel('Time (sec)',fontsize=18)
    plt.ylabel('Force (mN)',fontsize=18)
    plt.title('Vitreous',fontsize=20)
    plt.grid()
    plt.legend(loc = 'best',fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
    ↪  '_NForce_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    """ Compare sum RF vs Nforce """
    ################## Plot Data #########################
    plt.plot(t, RF,'-',color='blue',linewidth=2,markersize=2,label = 'RF')
    plt.plot(t, NF,':',color='red',linewidth=2,markersize=2,label = 'NFORC')
    plt.xlabel('Time (sec)',fontsize=18)
    plt.ylabel('Force (mN)',fontsize=18)
    plt.title('Vitreous',fontsize=20)
    plt.grid()
    plt.legend(loc = 'best',fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
    ↪  '_RF_vs_NForce_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    """ Plot bond disp """
    ################## Plot Data #########################
    plt.plot(t, BD,'-',color='blue',linewidth=2,markersize=2,label = 'Bond - Disp')
    plt.plot(t, TD,'-',color='red',linewidth=2,markersize=2,label = 'Top - Disp')
    plt.xlabel('Time (sec)',fontsize=18)
    plt.ylabel('Bond Disp (mm)',fontsize=18)
    plt.title('Vitreous',fontsize=20)
    plt.grid()
    plt.legend(loc = 'best',fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
    ↪  '_disp_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    """ Contact Nforce """
    ################## Plot Data #########################
    plt.plot(t, CnF_RV,'-',color='red',linewidth=2,markersize=2,label =
    ↪  r'CnormF$_{RV}$')
    plt.plot(t, CnF_VR,':',color='blue',linewidth=2,markersize=2,label =
    ↪  r'CnormF$_{VR}$')
    plt.xlabel('Time (sec)',fontsize=18)
    plt.ylabel('Force (mN)',fontsize=18)
    plt.title('Contact Normal Force',fontsize=20)
    plt.grid()
    plt.legend(loc = 'best',fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
    ↪  'CnormF_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
    plt.close()

```

```
    """ Contact Cpress """
    ################## Plot Data #########################
    plt.plot(t, Cp_RV,'-',color='red',linewidth=2,markersize=2,label = r'Cpress$_{RV}$')
    plt.plot(t, Cp_VR,':',color='blue',linewidth=2,markersize=2,label = r'Cpress$_{VR}$')
    plt.xlabel('Time (sec)',fontsize=18)
    plt.ylabel('Pressure (Pa)',fontsize=18)
    plt.title('Contact pressure',fontsize=20)
    plt.grid()
    plt.legend(loc = 'best',fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName + 'Cpress_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    """ Contact AVG_Cpress """
    ################## Plot Data #########################
    plt.plot(t, AVG_Cp_RV,'-',color='red',linewidth=2,markersize=2,label = r'AVG Cpress$_{RV}$')
    plt.plot(t, AVG_Cp_VR,':',color='blue',linewidth=2,markersize=2,label = r'AVG Cpress$_{VR}$')
    plt.xlabel('Time (sec)',fontsize=18)
    plt.ylabel('Pressure (Pa)',fontsize=18)
    plt.title('Contact pressure Average',fontsize=20)
    plt.grid()
    plt.legend(loc = 'best',fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName + 'AVG_Cpress_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    """ Contact Cshear1 """
    ################## Plot Data #########################
    plt.plot(t, Cs1_RV,'-',color='red',linewidth=2,markersize=2,label = r'Cshear$_{RV}^1$')
    plt.plot(t, Cs1_VR,':',color='blue',linewidth=2,markersize=2,label = r'Cshear$_{VR}^1$')
    plt.xlabel('Time (sec)',fontsize=18)
    plt.ylabel('Shear Force (mN)',fontsize=18)
    plt.title('Contact shear 1 force',fontsize=20)
    plt.grid()
    plt.legend(loc = 'best',fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName + 'Cshear1_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    """ Contact Cshear2 """
    ################## Plot Data #########################
    plt.plot(t, Cs2_RV,'-',color='red',linewidth=2,markersize=2,label = r'Cshear$_{RV}^2$')
    plt.plot(t, Cs2_VR,':',color='blue',linewidth=2,markersize=2,label = r'Cshear$_{VR}^2$')
    plt.xlabel('Time (sec)',fontsize=18)
    plt.ylabel('Shear Force (mN)',fontsize=18)
    plt.title('Contact shear 2 force',fontsize=20)
    plt.grid()
    plt.legend(loc = 'best',fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName + 'Cshear2_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
```

```python
    plt.close()

    """ Contact CshearF """
    ################## Plot Data #########################
    plt.plot(t, CsF_RV,'-',color='red',linewidth=2,markersize=2,label = r'Cshear$_{RV}^F$')
    plt.plot(t, CsF_VR,':',color='blue',linewidth=2,markersize=2,label = r'Cshear$_{VR}^F$')
    plt.xlabel('Time (sec)',fontsize=18)
    plt.ylabel('Shear Force (mN)',fontsize=18)
    plt.title('Contact shear F force',fontsize=20)
    plt.grid()
    plt.legend(loc = 'best',fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName + 'CshearF_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
    plt.close()

def plot_History_Output(fileName, dataDirectory):
    """ History Output Data """
    df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)
    df.columns = ["Frame","Time","Internal_Energy","Kinetic_Energy"]

    t_h = df.Time
    IE = df.Internal_Energy
    KE = df.Kinetic_Energy

    (figureName, ext) = os.path.splitext(fileName) # Split the file extension

    """ Plots History Outputs """
    ################## Plot Data #########################
    plt.plot(t_h, IE,'-',color='blue',linewidth=2,markersize=2,label = 'Internal Energy')
    plt.plot(t_h, KE,'-',color='red',linewidth=2,markersize=2,label = 'Kinetic Energy')
    plt.xlabel('Time (sec)',fontsize=18)
    plt.ylabel('Energy (J)',fontsize=18)
    plt.title('Energy',fontsize=20)
    plt.grid()
    plt.legend(loc = 'best',fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName + '_Energy_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    ################## Plot Data #########################
    plt.semilogy(t_h, KE/IE,'-',color='blue',linewidth=2,markersize=2,label = r'Ratio $\frac{KE}{IE}$')
    plt.semilogy(t_h, 0.1*np.ones(len(t_h)),'-',color='red',linewidth=2,markersize=2,label = '10%')
    plt.xlabel('Time (sec)',fontsize=18)
    plt.ylabel('Ratio of KE to IE',fontsize=18)
    plt.title('Energy ratio',fontsize=20)
    plt.grid()
    plt.legend(loc = 'best',fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName + '_Ratio_KE_IE_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    # ################## Plot Data #########################
```

```
    # plt.plot(t_h, gRP_RF,'-',color='blue',linewidth=2,markersize=2,label = r'G$_{RP}$')
    # plt.xlabel('Time (sec)',fontsize=18)
    # plt.ylabel('Reaction Force (mN)',fontsize=18)
    # plt.title('Glue Reference Point History Output',fontsize=20)
    # plt.grid()
    # plt.legend(loc = 'best',fontsize = 'medium')
    # plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName + '_Glue_RP_RF.png'),dpi=300, bbox_inches='tight') # Save figure
    # plt.close()

    # ################# Plot Data ########################
    # plt.plot(t_h, CFNCP_RG*1e3,'-',color='red',linewidth=2,markersize=2,label = r'CFNCP$_{RG}$')
    # plt.plot(t_h, CFNCP_GR*1e3,':',color='blue',linewidth=2,markersize=2,label = r'CFNCP$_{GR}$')
    # plt.xlabel('Time (sec)',fontsize=18)
    # plt.ylabel('Reaction Force (mN)',fontsize=18)
    # plt.title('Contact Force CFNCP_RG/GR History Output',fontsize=20)
    # plt.grid()
    # plt.legend(loc = 'best',fontsize = 'medium')
    # plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName + '_CFNCP_RG_GR_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
    # plt.close()

    # ################# Plot Data ########################
    # plt.plot(t_h, CFNCP_RV*1e3,'-',color='red',linewidth=2,markersize=2,label = r'CFNCP$_{RV}$')
    # plt.plot(t_h, CFNCP_VR*1e3,':',color='blue',linewidth=2,markersize=2,label = r'CFNCP$_{VR}$')
    # plt.xlabel('Time (sec)',fontsize=18)
    # plt.ylabel('Reaction Force (mN)',fontsize=18)
    # plt.title('Contact Force CFNCP_RV/GR History Output',fontsize=20)
    # plt.grid()
    # plt.legend(loc = 'best',fontsize = 'medium')
    # plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName + '_CFNCP_RV_VR_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
    # plt.close()

    # ################# Plot Data ########################
    # plt.plot(t_h, CAreaCP_RG,'-',color='red',linewidth=2,markersize=2,label = r'CAreaCP$_{RG}$')
    # plt.plot(t_h, CAreaCP_GR,':',color='blue',linewidth=2,markersize=2,label = r'CAreaCP$_{GR}$')
    # plt.xlabel('Time (sec)',fontsize=18)
    # plt.ylabel(r'CArea ($m^2$)',fontsize=18)
    # plt.title('Contact Area CAreaCP_RG/GR History Output',fontsize=20)
    # plt.grid()
    # plt.legend(loc = 'best',fontsize = 'medium')
    # plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName + 'CAreaCP_RG_GR_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
    # plt.close()

    # ################# Plot Data ########################
    # plt.plot(t_h, CAreaCP_RV*1e3,'-',color='red',linewidth=2,markersize=2,label = r'CAreaCP$_{RV}$')
    # plt.plot(t_h, CAreaCP_VR*1e3,':',color='blue',linewidth=2,markersize=2,label = r'CAreaCP$_{VR}$')
```

```python
    # plt.xlabel('Time (sec)',fontsize=18)
    # plt.ylabel(r'CArea ($m^2$)',fontsize=18)
    # plt.title('Contact Area CAreaCP_RV/VR History Output',fontsize=20)
    # plt.grid()
    # plt.legend(loc = 'best',fontsize = 'medium')
    # plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
    ↪  'CAreaCP_RV_VR_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
    # plt.close()

    print("Plots will be in the figures folder")

def plotFieldHist(FieldfileName, HistfileName, dataDirectory, dataCompare, BondStatus,
↪  PDFMStatus):
    """ Field Output Data """
    df1 = pd.read_csv(os.path.join(dataDirectory,FieldfileName), sep="\t", header=0)

    Header = [] # Header information for the dataframe
    Header.append('Frame')
    Header.append('Time')
    Header.append('RF_y_dot')
    Header.append('RFx')
    Header.append('RFy')
    Header.append('RFz')
    Header.append('Nodal_Force')
    Header.append('CnormF_RV')
    Header.append('CnormF_VR')
    Header.append('Cpress_RV')
    Header.append('Cpress_VR')
    Header.append('AVG_Cpress_RV_AVG')
    Header.append('AVG_Cpress_VR_AVG')
    Header.append('Cshear1_RV')
    Header.append('Cshear1_VR')
    Header.append('Cshear2_RV')
    Header.append('Cshear2_VR')
    Header.append('CshearF_RV')
    Header.append('CshearF_VR')
    Header.append('Glue_Displacements')
    Header.append('Bond_Displacements')

    df1.columns = Header

    t = df1.Time
    RF = df1.RF_y_dot*1e3 # Convert from N to mN
    NF = df1.Nodal_Force*1e3 # Convert from N to mN
    CnF_RV = df1.CnormF_RV
    CnF_VR = df1.CnormF_VR
    Cp_RV = df1.Cpress_RV
    Cp_VR = df1.Cpress_VR
    AVG_Cp_RV = df1.AVG_Cpress_RV_AVG
    AVG_Cp_VR = df1.AVG_Cpress_VR_AVG
    Cs1_RV = df1.Cshear1_RV
    Cs1_VR = df1.Cshear1_VR
    Cs2_RV = df1.Cshear2_RV
    Cs2_VR = df1.Cshear2_VR
    CsF_RV = df1.CshearF_RV
    CsF_VR = df1.CshearF_VR
    TD = df1.Glue_Displacements*1e3 # Convert from m to mm
    BD = df1.Bond_Displacements*1e3 # Convert from m to mm
```

```python

    (figureName, ext) = os.path.splitext(FieldfileName) # Split the file extension

    """ History Output Data """
    df2 = pd.read_csv(os.path.join(dataDirectory, HistfileName), sep="\t", header=0, index_col=False)

    df2.columns = ["Frame","Time","Internal_Energy","Kinetic_Energy"]

    t_h = df2.Time
    IE = df2.Internal_Energy
    KE = df2.Kinetic_Energy

    (figureName, ext) = os.path.splitext(HistfileName) # Split the file extension

    # ################## Plot Data #########################
    # plt.plot(t, RF,'-',color='blue',linewidth=2,markersize=2,label = r'Field $G^{RP}$ RF')
    # plt.plot(t, NF,'--',color='red',linewidth=2,markersize=2,label = 'Field NFORC R')
    # plt.plot(t_h, gRP_RF,':',color='black',linewidth=2,markersize=2,label = r'Hist $G^{RP}$ RF')
    # plt.xlabel('Time (sec)',fontsize=18)
    # plt.ylabel('Force (mN)',fontsize=18)
    # plt.title('Reaction Force Compare Field to History Outputs',fontsize=20)
    # plt.grid()
    # plt.legend(loc = 'best',fontsize = 'medium')
    # plt.savefig(os.path.join(dataDirectory,'Figures/' + 'CompareFieldtoHist_RF.png'),dpi=300, bbox_inches='tight') # Save figure
    # plt.close()

def plot_BondStat_Output(fileName, dataDirectory):
    """ BondStat Output Data """
    df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)

    t = df['Time (s)']

    # determine the length of the number of bonded nodes
    # linspace from 0 to 1 by the number of nodes for the y-position
    # Loop over the number of bonded nodes and plot the y-th position vs time with the color of the bond load on a single plot

    fig1, ax1 = plt.subplots()
    nRows  = np.shape(df)[0]
    nCols = np.shape(df)[1] - 1 # subtract the time column
    y = np.linspace(0,1,nCols)
    count = 0
    for (colName, colData) in df.iteritems():
        if colName.find('Time') == -1:
            """ Plots History Outputs """
            ################# Plot Data ########################
            sc = ax1.scatter(t, np.ones(nRows)*y[count], c=colData, cmap=cm.cool, s=5, edgecolors='none', vmin=0, vmax=1)
            count += 1 # update the counter
        else:
            continue

    # plt.gray() # turns image to grayscale
```

```python
    plt.colorbar(sc)
    ax1.set_xlabel('Time (sec)',fontsize=18)
    ax1.set_ylabel('Bonded Nodes',fontsize=18)
    ax1.set_title('BONDSTAT (Color indicates status)',fontsize=20)
    (figureName, ext) = os.path.splitext(fileName) # Split the file extension
    fig1.savefig(os.path.join(dataDirectory,'Figures/' + figureName + '_BONDSTAT_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    print("Plots will be in the figures folder")

def plot_BondLoad_Output(fileName, dataDirectory):
    """ BondLoad Output Data """
    df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)

    t = df['Time (s)']

    # determine the length of the number of bonded nodes
    # linspace from 0 to 1 by the number of nodes for the y-position
    # Loop over the number of bonded nodes and plot the y-th position vs time with the color of the bond load on a single plot

    fig1, ax1 = plt.subplots()
    nRows  = np.shape(df)[0]
    nCols = np.shape(df)[1] - 1 # subtract the time column
    y = np.linspace(0,1,nCols)
    count = 0
    for (colName, colData) in df.iteritems():
        if colName.find('Time') == -1:
            """ Plots History Outputs """
            ################# Plot Data #######################
            sc = ax1.scatter(t, np.ones(nRows)*y[count], c=colData, cmap=cm.cool, s=5, edgecolors='none', vmin=0, vmax=1)
            count += 1 # update the counter
        else:
            continue

    # plt.gray() # turns image to grayscale
    plt.colorbar(sc)
    ax1.set_xlabel('Time (sec)',fontsize=18)
    ax1.set_ylabel('Bonded Nodes',fontsize=18)
    ax1.set_title('BONDLOAD (Color indicates status)',fontsize=20)
    (figureName, ext) = os.path.splitext(fileName) # Split the file extension
    fig1.savefig(os.path.join(dataDirectory,'Figures/' + figureName + '_BONDLOAD_vs_t.png'),dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    print("Plots will be in the figures folder")

def PlotAbqData(fileName, dataDirectory, dataCompare, BondStatus, PDFMStatus):

    # """ Change directory to correct path """
    # filePath = os.getcwd()
    # data_directory = os.path.join(filePath,jobName)
    # figures_directory = os.path.join(filePath,jobName,'Figures')
    # if not os.path.exists(figures_directory):
    #     os.makedirs(figures_directory)

```

```python
    """ Call both functions to plot Field/History data """
    field_files = [f for f in os.listdir(dataDirectory) if os.path.isfile(os.path.join(dataDirectory, f)) and f.startswith('output_Field')]
    for fname in field_files:
        plot_Field_Output(fname, dataDirectory, dataCompare, BondStatus, PDFMStatus)

    history_files = [f for f in os.listdir(dataDirectory) if os.path.isfile(os.path.join(dataDirectory, f)) and f.startswith('output_History')]
    for hname in history_files:
        plot_History_Output(hname, dataDirectory)

    for fname in field_files:
        plotFieldHist(fname, hname, dataDirectory, dataCompare, BondStatus, PDFMStatus)

    if BondStatus == True:
        BONDSTAT_files = [f for f in os.listdir(dataDirectory) if os.path.isfile(os.path.join(dataDirectory, f)) and f.startswith('BONDSTAT')]
        for bname in BONDSTAT_files:
            plot_BondStat_Output(bname, dataDirectory)

        BONDLOAD_files = [f for f in os.listdir(dataDirectory) if os.path.isfile(os.path.join(dataDirectory, f)) and f.startswith('BONDLOAD')]
        for bname in BONDLOAD_files:
            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.*

```python
# -*- coding: utf-8 -*-
"""
Created on Sat Nov  7 17:27:47 2020

@author: Kiffer2

"""
import numpy as np
import pandas as pd
from scipy import interpolate
import matplotlib.pyplot as plt
from matplotlib.pyplot import cm
import matplotlib.patheffects as pe
import os
import os.path
import sys
import pdb

def Least_Squares(x, y):
    """
```

```
    Calculate the slope and y-intercept using matrix math
    x & y are the coordinates of points

    parameters (X,Y) Data

    Returns:
        Curve fit data and parameters m*x + b, R squared value
    """
    Z = np.ones((len(x),2))
    Z[:,1] = x
    # Calculate the matrix inverse for the constants of the regression
    A = np.dot(np.linalg.inv(np.dot(Z.T,Z)),(np.dot(Z.T,y)))
    linFit = x*A[1] + A[0]

    # Stats
    SS_tot = np.sum((y - np.mean(y))**2)
    SS_res = np.sum((y - linFit)**2)
    Rsqd = 1 - SS_res/SS_tot

    return linFit, A, Rsqd


def residualFcn(fileName, dataDirectory, maxForceTime, dataCompare, objErr,
                slopeFlag, maxForceFlag, ssForceFlag, timeBeforePeak):
    """
    Parameters
    ----------
    fileName : Output txt file with the odb data
    dataDirectory : Location of the output file

    Returns
    -------
    Maximum force from the txt file
    """

    # In[Simulated data]
    df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)

    Header = [] # Header information for the dataframe
    Header.append('Frame')
    Header.append('Time')
    Header.append('RF_y_dot')
    Header.append('RFx')
    Header.append('RFy')
    Header.append('RFz')
    Header.append('Nodal_Force')
    Header.append('CnormF_RV')
    Header.append('CnormF_VR')
    Header.append('Cpress_RV')
    Header.append('Cpress_VR')
    Header.append('AVG_Cpress_RV_AVG')
    Header.append('AVG_Cpress_VR_AVG')
    Header.append('Cshear1_RV')
    Header.append('Cshear1_VR')
    Header.append('Cshear2_RV')
    Header.append('Cshear2_VR')
    Header.append('CshearF_RV')
    Header.append('CshearF_VR')
```

```
    Header.append('Retina_Glue_Top')
    Header.append('Bond_Displacements')

    df.columns = Header

    tt = df.Time
    RF = df.RF_y_dot*1e3 # Convert from N to mN
    NF = df.Nodal_Force*1e3 # Convert from N to mN
    CnF_RV = df.CnormF_RV*1e3 # Convert from N to mN
    CnF_VR = df.CnormF_VR*1e3 # Convert from N to mN
    Cp_RV = df.Cpress_RV
    Cp_VR = df.Cpress_VR
    AVG_Cp_RV = df.AVG_Cpress_RV_AVG
    AVG_Cp_VR = df.AVG_Cpress_VR_AVG
    Cs1_RV = df.Cshear1_RV*1e3 # Convert from N to mN
    Cs1_VR = df.Cshear1_VR*1e3 # Convert from N to mN
    Cs2_RV = df.Cshear2_RV*1e3 # Convert from N to mN
    Cs2_VR = df.Cshear2_VR*1e3 # Convert from N to mN
    CsF_RV = df.CshearF_RV*1e3 # Convert from N to mN
    CsF_VR = df.CshearF_VR*1e3 # Convert from N to mN
    dn = df.Retina_Glue_Top*1e3 # Convert from m to mm
    BD = df.Bond_Displacements*1e3 # Convert from m to mm

    # maybe try to output the maximum force at a specific time
    specificTime = maxForceTime
    actualTime = min(df['Time'], key=lambda x:abs(x - specificTime))
    force_at_time = RF[df['Time'] == actualTime].values[0]

    # In[Experimental data]
    """ Read in the csv file """
    dfValsn = pd.read_csv(os.path.join(dataCompare), sep="\t", nrows=29,
                          header=None, names=['Var', 'Attribute'])

    """ File Attributes """
    HID =           dfValsn['Attribute'][0]
    HAGE =          dfValsn['Attribute'][1]
    HG =            dfValsn['Attribute'][2]
    HLR =           dfValsn['Attribute'][3]
    HR =            dfValsn['Attribute'][4]
    HSSi =     float(dfValsn['Attribute'][12])
    HSSf =     float(dfValsn['Attribute'][13])
    HTMax =    float(dfValsn['Attribute'][14])
    HDispMax = float(dfValsn['Attribute'][15])
    HFMax =    float(dfValsn['Attribute'][16]) # (mN)
    HFSS =     float(dfValsn['Attribute'][17]) # (mN)
    # slope from 20 seconds prior to max force value
    HSlope20 = float(dfValsn['Attribute'][20]) # (mN/m)

    dfn = pd.read_csv(os.path.join(dataCompare), sep="\t", header=30)
    dfn.columns = ['Time', 'Extension', 'Force']
    dfn_time = dfn.Time
    dfn_extension = dfn.Extension # mm
    dfn_force = dfn.Force*1e3 # N ---> mN

    # if fileName.find('sym') >= 0:
    #     # divide all data trace values by 2
    #     dfn_force = dfn_force/2
    #     HFMax = HFMax/2
```

```python
    #     HFSS = HFSS/2

    # SS Array
    ssTimeArray = np.array([HSSi, HSSf])
    ssValArray = np.array([HFSS, HFSS])

# In[Experimental data isolate linear region up to peak]

    # slope calculation for 20 seconds prior to the max peel force
    # (Experimental Data)
    maxIndex = dfn_time[dfn_time == HTMax].index.values[0]

    # Convert to data array length
    timeBeforePeak = timeBeforePeak*10

    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec) to location of max
    ↪  force
    x_n = dfn_extension[maxIndex - timeBeforePeak:maxIndex]
    y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
    # Perform least squares
    curveFit_n, Params_n, R_Squared_n = Least_Squares(x_n, y)

    # Shift extension data so that the linear region is extrapolated
    # through the origin
    shift_disp = abs(Params_n[0]/Params_n[1])
    if Params_n[0] > 0:
        dfn_extension_shift = dfn_extension + shift_disp

        if min(dfn_extension_shift) > 0:
            # Add zero to prevent mishaps with interpolation
            dfn_extension_shift = [0] + dfn_extension_shift
    else:
        dfn_extension_shift = dfn_extension - shift_disp

    # Now that the data has been shifted, recalculate the linear regression
    # using the reduced data set
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec) to location of max
    ↪  force
    x_n = dfn_extension_shift[maxIndex - timeBeforePeak:maxIndex]
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec) to location of max
    ↪  force
    y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
    # Perform least squares
    curveFit_n, Params_n, R_Squared_n = Least_Squares(x_n,y)

    # Slope of the curve up to the max force !!!(from the simulated data)!!!
    # find the closest simulated displacement to the experimental
    # max displacement
    # adjustDisp = min(dn, key=lambda x:abs(x - dfn_extension[maxIndex]))
    # index = RF[dn == adjustDisp].index.values[0] # index determination
    # Index where the max reaction force is in the array
    simMaxIndex = RF.idxmax()
    simMaxForce = RF.max() # maximum simulated force value
    simMaxDisp = dn[RF == simMaxForce] # displacement at the max force value

    # If the max index is the second data point add one to it (Difficulty in
    # selecting the pandas series value) to select the fist two values in the
    # pandas array it needs to be RF[0:2] instead of RF[0:1] but the index
```

```python
    # value of the max force is 1.  Try to fix this issue
    if simMaxIndex == 1:
        simMaxIndex += 1

    x = dn[0:simMaxIndex] # Array from 0 to location of max force/n
    y = RF[0:simMaxIndex] # Array from 0 to location of max force/n
    # Perform least squares
    curveFit, Params, R_Squared = Least_Squares(x,y)

    # Updated force at specific max disp with adjusted value (Simulated data)
    specificTime = maxForceTime
    actualDisp = min(dn, key=lambda x:abs(x - dfn_extension_shift[maxIndex]))
    force_at_Disp = RF[dn == actualDisp].values[0]

    # Max peel force displacement at max and steady state
    dfn_max_Disp = dfn_extension_shift[dfn_time == HTMax]
    dfn_ss_Disp = [dfn_extension_shift[dfn_time == HSSi].values[0],
                   dfn_extension_shift[dfn_time == HSSf].values[0]] # flatten()

    """ Simulated Steady State calculation """
    if simMaxIndex == len(RF):
        simMaxGreaterIndex = len(RF) - 1
    else:
        # return the mean and median of the points after the peak force value
        # This will always round down
        simMaxGreaterIndex = int(simMaxIndex + (len(RF) - simMaxIndex)*(31/64))

    # Steady state values from the max force index half way to the end
    # Force values after the peak force
    RF_SteadyState = RF[simMaxGreaterIndex:]
    # Displacement values after the peak force
    dn_SteadyState = dn[simMaxGreaterIndex:]

    SSMean = np.mean(RF_SteadyState) # Mean
    SSMedian = np.median(RF_SteadyState) # Median

    # In[Plots]
    """ Plots """
    # Plot the experimental, simulated, and curve fit data

    # Split the file extension
    (figureName, ext) = os.path.splitext(fileName)

    # Plot the data trace to compare the simulated results with the force
    # displacement curves
    plt.plot(dfn_extension_shift, dfn_force,'-', color='r', linewidth=1,
             markersize=2, label = '{}, Age: {}'.format(HID, HAGE),
             alpha = 0.5)

    if str(HFMax) == 'nan' and str(HSSi) == 'nan':
        print('No max or steady state')
        pass

    if str(HFMax) != 'nan':
        plt.plot(dfn_max_Disp, HFMax,'.', color='k', linewidth=1,
                 markersize=20,
                 label = 'Max Peel - {:.4f} (mN)'.format(HFMax),
                 path_effects=[pe.Stroke(linewidth=4, foreground='k'),
```

```
                               pe.Normal()])
        plt.plot(x_n, curveFit_n, '-', color='tab:blue', linewidth=2,
                 label=r'Curve fit Max - {} (s) '.format(timeBeforePeak/10) +
                 'y = {:.4f}x + '.format(Params_n[1]) +
                 '{:.4f} (mN), '.format(Params_n[0]) +
                 '$r^2$ = {:.4f}'.format(R_Squared_n), alpha = 1)

    if str(HSSi) != 'nan':
        plt.plot(dfn_ss_Disp, ssValArray,'-', color='c', linewidth=3,
                 markersize=2,
                 label = 'Steady State - {:.4f} (mN)'.format(HFSS),
                 path_effects=[pe.Stroke(linewidth=5, foreground='k'),
                               pe.Normal()])

    # Plot the simulated data
    plt.plot(dn, RF,'-', color='blue', linewidth=2, markersize=2,
             label = r'Simulated Reaction force $\Sigma F_{Retina}$')
    plt.plot(x, curveFit,'-', color='tab:green', linewidth=2, markersize=2,
             label= 'y = {:.4f}x + '.format(Params[1]) +
             '{:.4f} (mN), '.format(Params[0]) +
             '$r^2$ = {:.4f}'.format(R_Squared))
    plt.plot(simMaxDisp, simMaxForce, '.', color='tab:red', linewidth=1,
             markersize = 20,
             label = 'Simulated maximum Force {:.4f} (mN)'.format(simMaxForce))
    plt.plot(dn_SteadyState, np.ones(len(RF_SteadyState))*SSMean, '-',
             color='tab:gray', label = 'Simulated steady state force ' +
             '{:.4f} (mN)'.format(np.mean(RF_SteadyState)))

    # In[Error Calculation]
    # error between slope, force, and steady-state value

    maxSlopeMeasured = Params_n[1] # Experimental slope
    maxSlopeSimulated = Params[1] # Simulated slope
    maxForceMeasured = HFMax # Experimental max force
    maxForceSimulated = simMaxForce # Simulated max force
    SS_Measured = HFSS # Experimental SS force
    SSmeanSimulated = SSMean # Simulated SS force (mean)
    SSmedianSimulated = SSMedian # Simulated SS force (median)

    # Error calculation
    errorDict = {} # Dictionary
    if objErr == 'Difference':
        errorDict['slope']    = (maxSlopeMeasured - maxSlopeSimulated) if slopeFlag == True else []
        errorDict['maxForce'] = (maxForceMeasured - maxForceSimulated) if maxForceFlag == True else []
        errorDict['ssForce']  = (SS_Measured - SSmeanSimulated)        if ssForceFlag == True else []
    elif objErr == 'Ratio':
        errorDict['slope']    = (1 - maxSlopeMeasured / maxSlopeSimulated) if slopeFlag == True else []
        errorDict['maxForce'] = (1 - maxForceMeasured / maxForceSimulated) if maxForceFlag == True else []
        errorDict['ssForce']  = (1 - SS_Measured / SSmeanSimulated)        if ssForceFlag == True else []
    elif objErr == 'Relative uncertainty':
        errorDict['slope']    = ((maxSlopeMeasured - maxSlopeSimulated)/maxSlopeMeasured) if slopeFlag == True else []
```

```
        errorDict['maxForce'] = ((maxForceMeasured - maxForceSimulated)/maxForceMeasured) if maxForceFlag == True else []
        errorDict['ssForce']  = ((SS_Measured - SSmedianSimulated)/SS_Measured) if ssForceFlag == True else []
    else:
        print('Error in MaxForceError')
        sys.exit()

    # Error array values
    errorList = list(errorDict.values()) # convert to list
    errorList = [x for x in errorList if x] # get rid of empty values

    # String for the error array
    errorString = ', '.join('{:.4}'.format(i) for i in errorList)

    plt.plot([dfn_max_Disp, simMaxDisp], [HFMax, simMaxForce], '--',
             linewidth = 1, color = 'magenta', label = r'Difference ' +
             'between simulated & experiment max force: ' +
             '{:.4f}'.format(HFMax - np.max(RF)))

    # Plot the different conditions if they are to be compared
    if slopeFlag == True:
        plt.plot([], [], 'white', label = r'{} '.format(objErr) +
                 'between slopes is:  ' +
                 '{:.4f}'.format(errorDict['slope']))

    if maxForceFlag == True:
        plt.plot([], [], 'white', label = r'{} '.format(objErr) +
                 'between max force is:  ' +
                 '{:.4f}'.format(errorDict['maxForce']))

    if ssForceFlag == True:
        plt.plot([], [], 'white', label = r'{} '.format(objErr) +
                 'between steady state is:  ' +
                 '{:.4f}'.format(errorDict['ssForce']))

    plt.plot([], [], 'white',
             label = r'Objective error array:  [' + errorString + ']')
    plt.plot([], [], 'white', label = r'Error $L^2$ Norm: ' +
             '{:.4f}'.format(np.sqrt(np.dot(errorList, errorList))))

    ################# Plot Data ########################
    plt.axhline(0, color='black')
    plt.axvline(0, color='black')
    plt.ylabel('Force (mN)',fontsize=18)
    plt.xlabel('Distance (mm)',fontsize=18)
    plt.title('Simulation vs. Experimental Data Trace',fontsize=20)
    plt.grid()
    plt.legend(loc = 'best',fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
                             '_SlopeCompare.pdf'), dpi=300,
                bbox_inches='tight')
    plt.close()

    # In[Calculate interpolated Experimental and Simulated data]

    # slope calculation for 20 seconds prior to the max peel force
    # (Experimental Data)
```

```
    maxIndex = dfn_time[dfn_time == HTMax].index.values[0]
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec) to location of max
    ↪ force
    t_n = dfn_time[maxIndex - timeBeforePeak:maxIndex]
    y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
    # Perform least squares and return
    curveFit_n, Params_n_time, R_Squared_n = Least_Squares(t_n, y)

    # Shift extension data so that the linear region is extrapolated
    # through the origin
    shift_time = abs(Params_n_time[0]/Params_n_time[1])

    # shift time data for visual purposes
    if Params_n_time[0] > 0:
        dfn_time_shift = dfn_time + shift_time

        if min(dfn_time_shift) > 0:
            # Add zero to prevent mishaps with interpolation
            dfn_time_shift = [0] + dfn_time_shift
    else:
        dfn_time_shift = dfn_time - shift_time

    # x array for the linear region leading up to the peak force
    Fmax_t_shift = dfn_time_shift[maxIndex]
    fit_t = np.linspace(0, Fmax_t_shift, 200) # Selected value
    # fit_t = np.linspace(0, dfn_time_shift[np.argmax(dfn_force)], 200) # true max
    Fmax_x_shift = dfn_extension_shift[maxIndex]
    # fit_x = np.linspace(0, dfn_extension_shift[np.argmax(dfn_force)], 200) # true
    ↪ max
    fit_x = np.linspace(0, Fmax_x_shift, 200) # Selected value

    # create the linear region leading up to the peak force
    def fit(params, x):
        b, m = params
        return m*x + b
    fit_vals_y_time = fit(Params_n_time, fit_t)
    fit_vals_y_force = fit(Params_n, fit_x)

    # Trim the shifted experimental data to be greater than zero
    t_exp = dfn_time_shift[dfn_time_shift >= 0]
    x_exp = dfn_extension_shift[dfn_time_shift >= 0]
    y_exp = dfn_force[dfn_time_shift >= 0]

    # data frame with original data only shifted
    dfdata = pd.DataFrame(np.array([t_exp, x_exp, y_exp]).T,
                          columns=['t', 'x', 'y'])

    # Select time beyond the max time to the end of the data
    t_geq_max = dfn_time_shift[maxIndex:]
    x_geq_max = dfn_extension_shift[maxIndex:]
    y_geq_max = dfn_force[maxIndex:]

    # dataframe of data points from the max value to the end
    dfgmax = pd.DataFrame(np.array([t_geq_max, x_geq_max, y_geq_max]).T,
                          columns=['t', 'x', 'y'])

    # data frame of points from zero to the max value
    linArray = np.array([fit_t, fit_x, fit_vals_y_force])
```

```python
    dfLin = pd.DataFrame(linArray.T, columns=['t', 'x', 'y'])

    # create the new data frame of linear points up to the peak and all points
    # beyond
    dfNew = dfLin.append(dfgmax, ignore_index=True)

    # Interpolate the experimental data
    n_data_pts = 100
    start_point_time = tt[RF.argmax()] # Time at the peak (simulated)
    start_point_disp = dn[RF.argmax()] # Disp at the peak (simulated)
    f_exp_time = interpolate.interp1d(dfNew['t'], dfNew['y'])
    f_exp_disp = interpolate.interp1d(dfNew['x'], dfNew['y'])
    t_new_exp = np.linspace(start_point_time, tt[tt.argmax()],
                            n_data_pts) # (s)
    x_new_exp = np.linspace(start_point_disp, dn[tt.argmax()],
                            n_data_pts) # (mm)
    y_new_exp_time = f_exp_time(t_new_exp) # Interpolate `interp1d`
    y_new_exp_disp = f_exp_disp(x_new_exp) # Interpolate `interp1d`

    # In[Interpolated Simulated Trace]

    # Interpolate the simulated data
    f_sim_time = interpolate.interp1d(tt, RF)
    f_sim_disp = interpolate.interp1d(dn, RF)
    t_new_sim = np.linspace(start_point_time, tt[tt.argmax()],
                            n_data_pts) # (s)
    x_new_sim = np.linspace(start_point_disp, dn[tt.argmax()],
                            n_data_pts) # (mm)
    y_new_sim_time = f_sim_time(t_new_sim) # Interpolate `interp1d`
    y_new_sim_disp = f_sim_disp(x_new_sim) # Interpolate `interp1d`

    # In[Plots]
    ''' Time curve '''
    fit, ax = plt.subplots()
    ax.plot()
    ax.plot(dfdata['t'], dfdata['y'], label='Original Shifted Data',
            alpha = 0.5)
    ax.plot(dfNew['t'], dfNew['y'], label='Merged Data',
            alpha = 0.5)
    ax.plot(t_new_exp, y_new_exp_time, '--', label='Interp Experimental Data')
    ax.plot(tt, RF, label='Simulated Data')
    ax.plot(t_new_sim, y_new_sim_time, ':', label='Interp Simulated Data')
    ax.axhline(color='k')
    ax.set_xlim([0, 300])
    ax.set_xlabel('Time (s)', fontsize=14)
    ax.set_ylabel('Force (N)', fontsize=14)
    ax.legend(loc='best', fontsize=14)
    ax.grid('on')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
                             '_Interp_Time.pdf'), dpi=300,
                bbox_inches='tight')
    plt.close()

    ''' Displacement curve '''
    fit, ax = plt.subplots()
    ax.plot()
    ax.plot(dfdata['x'], dfdata['y'], label='Original Shifted Data',
            alpha = 0.5)
```

```
    ax.plot(dfNew['x'], dfNew['y'], label='Merged Data',
            alpha = 0.5)
    ax.plot(x_new_exp, y_new_exp_disp, '--', label='Interp Experimental Data')
    ax.plot(dn, RF, label='Simulated Data')
    ax.plot(x_new_sim, y_new_sim_disp, ':', label='Interp Simulated Data')
    ax.axhline(color='k')
    ax.set_xlim([0, max(dn)])
    ax.set_xlabel('Displacement (mm)', fontsize=14)
    ax.set_ylabel('Force (N)', fontsize=14)
    ax.legend(loc='best', fontsize=14)
    ax.grid('on')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
                             '_Interp_Disp.pdf'), dpi=300,
                bbox_inches='tight')
    plt.close()

    ''' Displacement curve only showing interpolated data '''
    residual = y_new_exp_disp - y_new_sim_disp # residual calculation
    L2Norm = np.sqrt(np.dot(residual, residual))

    fit, ax = plt.subplots()
    ax.plot()
    ax.plot(x_new_exp, y_new_exp_disp, '-', label='Interp Experimental Data')
    ax.plot(x_new_sim, y_new_sim_disp, '-', label='Interp Simulated Data')
    ax.plot(x_new_sim, residual, ':', label=r'Residual = $(exp - sim)$',
            alpha = 0.8)
    ax.plot([], [], color='white', label=r'$L^2$ norm = {:.4f}'.format(L2Norm))
    ax.axhline(color='k', linewidth=0.25)
    ax.set_xlim([0, max(x_new_exp)])
    ax.set_xlabel('Displacement (mm)', fontsize=14)
    ax.set_ylabel('Force (N)', fontsize=14)
    ax.legend(loc='best', fontsize=14)
    ax.grid('on')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
                             '_Interp_Disp_Clean.pdf'), dpi=300,
                bbox_inches='tight')
    plt.close()

    returnList = [Params[1], simMaxForce, SSMean, SSMedian, y_new_exp_disp,
                  y_new_sim_disp]
    return returnList

# In[Function that calls the nested function to compute the residual]
def findResidual(fileName, dataDirectory, maxForceTime, dataCompare, objErr,
                 slopeFlag, maxForceFlag, ssForceFlag, timeBeforePeak):
    """
    Parameters
    ----------
    fileName : Output txt file with the odb data
    dataDirectory : Location of the output file

    Returns
    -------
    maximumForce : Maximum force from the txt file
    """

    global residual
    """ Call function to return max displacement """
```

```
    ModelParamsFile = [f for f in os.listdir(dataDirectory)
                       if os.path.isfile(os.path.join(dataDirectory, f))
                       and f.startswith('output_Field')]
    for mpFile in ModelParamsFile:
        residual = residualFcn(mpFile, dataDirectory, maxForceTime,
                               dataCompare, objErr, slopeFlag, maxForceFlag,
                               ssForceFlag, timeBeforePeak)

    return residual
```

1.5.7 Max Force Script

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

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jan 17 23:56:35 2021

@author: Kiffer2
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import cm
import matplotlib.patheffects as pe
import os
import os.path
import sys
import pdb

def Least_Squares(x,y):
    """
    Calculate the slope and y-intercept using matrix math
    x & y are the coordinates of points

    parameters (X,Y) Data

    Returns:
        Curve fit data and parameters m*x + b, R squared value
    """
    Z = np.ones((len(x),2))
    Z[:,1] = x
    A = np.dot(np.linalg.inv(np.dot(Z.T,Z)),(np.dot(Z.T,y))) # Calculate the matrix
    ↪  inverse for the constants of the regression
    linFit = x*A[1] + A[0]

    # Stats
    SS_tot = np.sum((y - np.mean(y))**2)
    SS_res = np.sum((y - linFit)**2)
    Rsqd = 1 - SS_res/SS_tot

    return linFit, A, Rsqd

def maxForce(fileName, dataDirectory, maxForceTime, dataCompare):
    """
```

```python
    Parameters
    ----------
    fileName : Output txt file with the odb data
    dataDirectory : Location of the output file

    Returns
    -------
    Maximum force from the txt file
    """
    df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)

    Header = [] # Header information for the dataframe
    Header.append('Frame')
    Header.append('Time')
    Header.append('RF_y_dot')
    Header.append('RFx')
    Header.append('RFy')
    Header.append('RFz')
    Header.append('Nodal_Force')
    Header.append('CnormF_RV')
    Header.append('CnormF_VR')
    Header.append('Cpress_RV')
    Header.append('Cpress_VR')
    Header.append('AVG_Cpress_RV_AVG')
    Header.append('AVG_Cpress_VR_AVG')
    Header.append('Cshear1_RV')
    Header.append('Cshear1_VR')
    Header.append('Cshear2_RV')
    Header.append('Cshear2_VR')
    Header.append('CshearF_RV')
    Header.append('CshearF_VR')
    Header.append('Glue_Displacements')
    Header.append('Bond_Displacements')

    df.columns = Header

    RF = df.RF_y_dot*1e3 # N to mN
    dn = df.Glue_Displacements*1e3 # m to mm

    # maybe try to output the maximum force at a specific time
    specificTime = maxForceTime
    actualTime = min(df['Time'], key=lambda x:abs(x - specificTime))
    force_at_time = RF[df['Time'] == actualTime].values[0]

    # Plot the experimental, simulated, and curve fit data

    (figureName, ext) = os.path.splitext(fileName) # Split the file extension

    """ Read in the csv file """
    dfValsn = pd.read_csv(os.path.join(dataCompare), sep="\t", nrows=29, header=None,
    ↪  names=['Var', 'Attribute'])

    """ File Attributes """
    HID =            dfValsn['Attribute'][0]
    HAGE =           dfValsn['Attribute'][1]
    HG =             dfValsn['Attribute'][2]
    HLR =            dfValsn['Attribute'][3]
    HR =             dfValsn['Attribute'][4]
```

```python
    HSSi =     float(dfValsn['Attribute'][12])
    HSSf =     float(dfValsn['Attribute'][13])
    HTMax =    float(dfValsn['Attribute'][14])
    HDispMax = float(dfValsn['Attribute'][15])
    HFMax =    float(dfValsn['Attribute'][16]) # (mN)
    HFSS =     float(dfValsn['Attribute'][17])
    HSlope20 = float(dfValsn['Attribute'][20]) # (mN/m) slope from 20 seconds prior to max force value

    dfn = pd.read_csv(os.path.join(dataCompare), sep="\t", header=30)
    dfn.columns = ['Time', 'Extension', 'Force']
    dfn_time = dfn.Time
    dfn_extension = dfn.Extension # mm
    dfn_force = dfn.Force*1e3 # N ---> mN

    # SS Array
    ssTimeArray = np.array([HSSi, HSSf])
    ssValArray = np.array([HFSS, HFSS])

    # slope calculation for 20 seconds prior to the max peel force (Experimental Data)
    maxIndex = dfn_time[dfn_time == HTMax].index.values[0]
    x20 = dfn_extension[maxIndex-200:maxIndex] # Array from maxIndex - 200 (20 sec) to location of max force
    y = dfn_force[maxIndex-200:maxIndex] # Array from maxIndex - 200 (20 sec) to location of max force
    curveFit20, Params20, R_Squared20 = Least_Squares(x20,y) # Perform least squares and return

    # Shift extension data so that the linear region is extrapolated through the origin
    shift = abs(Params20[0]/Params20[1])
    dfn_extension = dfn_extension - shift

    # Now that the data has been shifted, recalculate the linear regression using the reduced data set
    x20 = dfn_extension[maxIndex-200:maxIndex] # Array from maxIndex - 200 (20 sec) to location of max force
    y = dfn_force[maxIndex-200:maxIndex] # Array from maxIndex - 200 (20 sec) to location of max force
    curveFit20, Params20, R_Squared20 = Least_Squares(x20,y) # Perform least squares and return

    # Slope of the curve up to the max force !!!(from the simulated data)!!!
    adjustDisp = min(dn, key=lambda x:abs(x - dfn_extension[maxIndex]))
    index = RF[dn == adjustDisp].index.values[0]
    simulationCriteria = index # Time before peak force for curve fitting
    x = dn[index - simulationCriteria:index] # Array from 0 to location of max force
    y = RF[index - simulationCriteria:index] # Array from 0 to location of max force
    curveFit, Params, R_Squared = Least_Squares(x,y) # Perform least squares and return

    # Updated force at specific max disp with adjusted value (Simulated data)
    specificTime = maxForceTime
    actualDisp = min(dn, key=lambda x:abs(x - dfn_extension[maxIndex]))
    force_at_Disp = RF[dn == actualDisp].values[0]

    # Simulated max force
```

```python
    simMaxForce = RF.max() # maximum simulated force value
    simMaxDisp = dn[RF == simMaxForce] # displacement at the max force value

    # Max peel force displacement at max and steady state
    dfn_max_Disp = dfn_extension[dfn_time == HTMax]
    # dfn_ss_Disp = np.array([dfn_extension[dfn_time == HSSi], dfn_extension[dfn_time == HSSf]]).flatten() # Didn't seem to work here
    dfn_ss_Disp = [dfn_extension[dfn_time == HSSi].values[0], dfn_extension[dfn_time == HSSf].values[0]]

    """ Plots """
    # Plot the data trace to compare the simulated results with the force displacement curves
    plt.plot(dfn_extension, dfn_force,'-', color='r', linewidth=1, markersize=2, label = '{}, Age: {}'.format(HID, HAGE), alpha = 0.5)

    if str(HFMax) == 'nan' and str(HSSi) == 'nan':
        print('No max or steady state')
        pass

    if str(HFMax) != 'nan':
        plt.plot(dfn_max_Disp, HFMax,'.', color='k', linewidth=1, markersize=20, label = 'Max Peel - {:.4f} (mN)'.format(HFMax), path_effects=[pe.Stroke(linewidth=4, foreground='k'), pe.Normal()])
        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)

    if str(HSSi) != 'nan':
        plt.plot(dfn_ss_Disp, ssValArray,'-', color='c', linewidth=3, markersize=2, label = 'Steady State - {:.4f} (mN)'.format(HFSS), path_effects=[pe.Stroke(linewidth=5, foreground='k'), pe.Normal()])

    plt.plot(dn, RF,'-',color='blue',linewidth=2,markersize=2,label = r'Simulated Reaction force $\Sigma F_{Retina}$')
    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))
    plt.plot(actualDisp, force_at_Disp, '.', color='tab:orange', linewidth=1, markersize = 20, label = 'Force at max disp {:.4f} (mN)'.format(force_at_Disp))
    plt.plot(simMaxDisp, simMaxForce, '.', color='tab:red', linewidth=1, markersize = 20, label = 'Simulated maximum Force {:.4f} (mN)'.format(simMaxForce))

    # error between slope and force value
    plt.plot([actualDisp, dfn_max_Disp], [force_at_Disp, HFMax], '--', linewidth = 1, color = 'magenta', label = r'ABS difference between force @ peak values is: {:.4f}'.format(abs(HFMax - force_at_Disp)))
    plt.plot([], [], 'white', label = r'ABS difference between slopes is: {:.4f}'.format(abs(Params20[1] - Params[1])))
    plt.plot([], [], 'white', label = r'ABS ratio between slopes is: {:.4f}'.format(abs(Params20[1] / Params[1])))

    ################## Plot Data ##########################
    plt.axhline(0,color='black') # x = 0
    plt.axvline(0,color='black') # y = 0    plt.xlabel('Displacement (mm)',fontsize=18)
    plt.ylabel('Force (mN)',fontsize=18)
```

```python
    plt.title('Vitreous',fontsize=20)
    plt.grid()
    plt.legend(loc = 'best',fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName + '_SlopeCompare.png'),dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    return Params[1], force_at_Disp, np.max(RF) # Slope, force @ specified time, max force

def findMaxForce(fileName, dataDirectory, maxForceTime, dataCompare):
    """
    Parameters
    ----------
    fileName : Output txt file with the odb data
    dataDirectory : Location of the output file

    Returns
    -------
    maximumForce : Maximum force from the txt file
    """

    global maximumForce
    """ Call function to return max displacement """
    ModelParamsFile = [f for f in os.listdir(dataDirectory) if os.path.isfile(os.path.join(dataDirectory, f)) and f.startswith('output_Field')]
    for mpFile in ModelParamsFile:
        maximumForce = maxForce(mpFile, dataDirectory, maxForceTime, dataCompare)

    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.*

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Jun 19 16:02:44 2020

@author: Kiffer Creveling
"""

# importing os module
import os
import glob
import shutil

def MoveAbqFiles(fileName, folderDirectory, abqWD):

    # """ Change directory to correct path """

    # dataDirectory = os.path.join(abqWD, fileName)
    # if not os.path.exists(dataDirectory):
```

```python
    #     os.makedirs(dataDirectory)

    # List of files in the ABQ working directory with the same name as the
    # 'fileName''
    fileList = glob.glob('{}.*'.format(os.path.join(abqWD, fileName)))
    for i in fileList:
        if i == folderDirectory:
            # Skip the file with the exact same name (i.e. Folder name...)
            continue
        source = os.path.join(abqWD,i)
        destination = os.path.join(folderDirectory)
        # copy (since shutil.move wouldn't overwrite)
        dest = shutil.copy(source, destination)
        os.remove(source) # remove the source file

    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.*

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Jan 28 22:28:58 2021

@author: Kiffer Creveling

This Python script does the following

    1) Select input parameters
    2) Generates the filename/description
    3) Calls Abaqus to create the .inp file w/ attributes & runs the job
    4) Creats a folder with the filename
    5) Extracts data from the Abaqus.odb file and creates two output files
        (Field/Hist)
    6) Plots the data
    7) Moves all files that have the same filename

"""

import os
import sys
import numpy as np
```

```python
import pandas as pd
# import itertools as it # iteration tools (product fcn)
# from scipy import *
# import scipy.optimize as opt
import lmfit as lf
import pdb
import subprocess
import pprint

# Define the location of the Abaqus Working Directory
# specific folder path where this file is located
pythonScriptPath = os.getcwd()
abqWD, pythonFiles = os.path.split(pythonScriptPath) # split file path

# pythonScriptCreateINP_Run_ABQ (pS_ABQ)
pS_ABQ = os.path.join(pythonFiles, 'Cohesive_T3_EyeModel_Generate_Abaqus.py')
# pythonScriptExtract (pSE)
pSE = os.path.join(pythonFiles, 'Cohesive_T3_EyeModel_DataExtract.py')

# In[Job Info]

optE_V = True
optKsTsFE = False
sweep = False

if optE_V == True:
    optimization = 'E_V'

    """ Optimization of the vitreous using a tied interface """
    # If "True" then abaqus uses a tied interface between the nodes
    tieInterface = True

    """ Objective Function Flags """
    slopeFlag = True
    maxForceFlag = True
    ssForceFlag = False # Only used for damage

    """ Traction separation """
    DamageInitiation = False # If "False" then do not include damage initation
    DamageEvolution = False # If "False" then do not include damage evolution


if optKsTsFE == True:
    optimization = 'K_nnK_ssK_ttt_nt_st_tFE'

    """ Optimization of the vitreous using a tied interface """
    # If "True" then abaqus uses a tied interface between the nodes
    tieInterface = False

    """ Objective Function Flags """
    slopeFlag = False
    maxForceFlag = True
    ssForceFlag = True # Only used for damage

    """ Traction separation """
    DamageInitiation = True # If "False" then do not include damage initation
    DamageEvolution = True # If "False" then do not include damage evolution

```

```

if sweep == True:
    optimization = None

    """ Parametric sweep of the vitreous using a tied interface """
    # If "True" then abaqus uses a tied interface between the nodes
    tieInterface = False

    """ Objective Function Flags """
    slopeFlag = False
    maxForceFlag = True
    ssForceFlag = True # Only used for damage

    """ Traction separation """
    DamageInitiation = True # If "False" then do not include damage initation
    DamageEvolution = True # If "False" then do not include damage evolution


# # optimization info
# optList = []
# optList.append(None)
# optList.append('E_R')
# optList.append('E_V')
# optList.append('E_RE_V') # Retina and Vitreous Young's Modulus
# optList.append('K_nnK_ssK_tt') # Traction Separation Paramters
# Vitreous Young's Modulus and Traction Separation Parameters
# optList.append('E_VK_nnK_ssK_tt')
# optList.append('t_nt_st_t') # Damage initiation parameters
# optList.append('FE') # Damage evolution parameters
# optList.append('t_nt_st_tFE') # Damage initiation and evolution parameteres
# optList.append('K_nnK_ssK_ttt_nt_st_tFE') # All cohesive parameters
# All parameters except for retina young's modulus
# optList.append('E_VK_nnK_ssK_ttt_nt_st_tFE')

# Change to specific optimization parameter.  If 'None', no optimization
# optimization = optList[2]
# print('Optimization parameters = ', optimization)

# """ Optimization of the vitreous using a tied interface """
# # If "True" then abaqus uses a tied interface between the nodes
# tieInterface = False

# """ Objective Function Flags """
# slopeFlag = False
# maxForceFlag = True
# ssForceFlag = True # Only used for damage

# """ Traction separation """
# DamageInitiation = True # If "False" then do not include damage initation
# DamageEvolution = True # If "False" then do not include damage evolution

""" Objective Function Error Formulation """
objFunErr = []
objFunErr.append('Difference') # Experimental - Simulated
objFunErr.append('Ratio') # Experimental/Simulated
# (Experimental - Simulated)/Experimental
objFunErr.append('Relative uncertainty')
# Change to specific optimization parameter.  If 'None', no optimization
```

```
objErr = objFunErr[0]
print('Objective function error formulation = ', objErr)

# Calculation for error
ErrorCalculation = []
ErrorCalculation.append('two-point method') # Slope, Peak force, SS Force
ErrorCalculation.append('data-trace method') # interpolated array

errorMethod = ErrorCalculation[0]
print('Error method calculation = ', errorMethod)

''' Symmetry '''
# Split model in half and multiply output by 2
symmetry = True

''' Simplified '''
# Remove the rigid body on the plastic tab and glue
simplified = True

''' Gravity '''
# Turn gravity on/off
gravity = False # Keep off until model is updated

# In[Comparison Data Trace]
compareDataFolder = 'PeelDataCompare'
specificDataTrace = 'Trace_45_Instron_Data.txt' # Data trace number
timeBeforePeak = 40 # Default is 20 seconds
dataCompare = os.path.join(abqWD,compareDataFolder,specificDataTrace)
dfValsn = pd.read_csv(dataCompare, sep="\t", nrows=29, header=None,
                      names=['Var', 'Attribute'])

""" File Attributes """
HID =            dfValsn['Attribute'][0]
HAGE =           dfValsn['Attribute'][1]
HG =             dfValsn['Attribute'][2]
HLR =            dfValsn['Attribute'][3]
HR =             dfValsn['Attribute'][4]
HSSi =     float(dfValsn['Attribute'][12])
HSSf =     float(dfValsn['Attribute'][13])
HTMax =    float(dfValsn['Attribute'][14])
HDispMax = float(dfValsn['Attribute'][15])
HFMax =    float(dfValsn['Attribute'][16]) # (mN)
HFSS =     float(dfValsn['Attribute'][17])
# (mN/m) slope from 20 seconds prior to max force value
HSlope20 = float(dfValsn['Attribute'][20])

dfn = pd.read_csv(os.path.join(dataCompare), sep="\t", header=30)
dfn.columns = ['Time', 'Extension', 'Force']
tn = dfn.Time
dn = dfn.Extension
df = dfn.Force # (N)

maxForceMeasured = HFMax # Value from data trace
maxSlopeMeasured = HSlope20 # slope from 20 seconds prior to max force value
SS_Measured = HFSS # simulated steady state force

# In[Functions]

```

```
if DamageInitiation == False and DamageEvolution == True:
    print('Unable to have DamageEvolution without DamageInitiation')
    sys.exit()

""" Tic Toc to determine runtime """
def tic():
    #Homemade version of matlab tic and toc functions
    import time
    global startTime_for_tictoc
    startTime_for_tictoc = time.time()

def toc():
    import time
    if 'startTime_for_tictoc' in globals():
        print("Elapsed time is " + str(time.time() - startTime_for_tictoc) +
              " seconds.")
        timeDiff = time.time() - startTime_for_tictoc
        return timeDiff
    else:
        print("Toc: start time not set")

try:
    os.environ.pop('PYTHONIOENCODING')
except KeyError:
    pass

# Import modules that plot/move all abq files to the new foldername
from ParameterSelection import ReadRAWDataTrace
from Cohesive_T3_Data_Plot import PlotAbqData
from Cohesive_T3_Residual import findResidual
from Cohesive_T3_CSMAXSCRT_MaxValue import CSMAXSCRTAbqData
from Move_ABQ_Files_To_Folder import MoveAbqFiles

newLine = '\n' + 77*'-' + '\n'

def jobAttributes():
    """
    Input: parameters used to create the filename and job description

    Output: namei, fileName, JobDescription
    """

    # Build the fileName
    fi = [] # initialize array
    fi.append(      '{}'.format(namei))
    fi.append(       'g') if gravity == True else ''
    fi.append(     'sym') if symmetry == True else ''
    fi.append(     't{}'.format(time))
    fi.append(    'E1{}'.format(e1Seedi[0]))
    fi.append(    'E2{}'.format(e2Seedi[0]))

    if simplified == False:
        fi.append(    'PT{}'.format(ptSeedi[0]))
        fi.append(     'G{}'.format(gSeedi[0]))

    fi.append(    'V1{}'.format(v1Seedi[0]))
    fi.append(    'V2{}'.format(v2Seedi[0]))
    fi.append(     'R{}'.format(rSeedi[0]))
```

```python
    fi.append(     'F{}'.format(massScaleFactori[0]))
    fi.append(    'MS{}'.format(massScaleTimeIncrementi[0]))
    fi.append('RE{:.0e}'.format(RetinaYoungsModulus_i))

    if optimization is not None:
        if optimization.find('E_V') == -1:
            fi.append('VE{:.0e}'.format(VitreousYoungsModulus_i))

    # If optimization, get rid of the title (Not an integer anymore)
    if optimization is None:
        fi.append(    'Kn{}'.format(int(Knni[0])))
        fi.append(    'Ks{}'.format(int(Kssi[0])))
        fi.append(    'Kt{}'.format(int(Ktti[0])))

    # If True, then damage initation, If optimization, get rid of the title
    # (Not an integer anymore)
    if DamageInitiation == True and optimization is None:
        fi.append(    'tn{}'.format(int(tni[0])))
        fi.append(    'ts{}'.format(int(tsi[0])))
        fi.append(    'tt{}'.format(int(tti[0])))

    # If True, then damage evolution, If optimization, get rid of the title
    # (Not an integer anymore)
    if ((DamageInitiation == True) and (DamageEvolution == True) and
        (optimization is None)):
        fi.append(    'FE{}'.format(int(FEi[0])))

    # .format(optimization))  optimization flag (I.e. RE, VE, Knn, Kss,
    # tn, or none)
    fi.append(     'opt') if optimization is not None else ''
    fi.append(     'TIE') if tieInterface == True else ''

    if sweep == True:
        # get rid of all attributes because a sweep is taking place
        fi = fi[0]

    """ Build file name and description """
    fileName = ''.join(item for item in fi)
    # fix header so no decimals, math show up in title
    fileName = fileName.replace('+', '_').replace('-', '_').replace('.', '_')
    jobNameString = 'Job Name - {}'.format(fileName)

    # used for simplification of script
    # Large value
    multStrL = ('\n\tgeometric multiplier = 2**{}, \n\tbase value = {}, ' +
                '\n\tmodel value = {}')
    # Small value
    multStrS = ('\n\tgeometric multiplier = 0.5**{}, \n\tbase value = {}, ' +
                '\n\tmodel value = {}')

    # Build the model description
    si = [] # initialize array
    si.append(newLine)
    si.append('({}) = model name'.format(namei))
    si.append(jobNameString)
    si.append('(g) - Gravity') if gravity == True else si.append('No Gravity')
    # update name in list
    si.append('(sym) SYMMETRIC model (XY) Plane') if symmetry == True else ''
```

```
    # update name in list
    si.append('(t) Simulated time {} (s)'.format(time))

    # Eye Holder
    si.append(('(E1) Eye holder outside edge seed size (Max) (SINGLE BIAS):  '
               + multStrS + ' (m)').format(*e1Seedi))
    si.append(('(E2) Eye holder inside edge seed size (Min):  ' + multStrS +
               ' (m)').format(*e2Seedi))

    # If simplified is in the title, get rid of glue and platic tab
    if simplified == False:
        si.append(('(PT) Plastic tab seed size:  ' + multStrS +
                   ' (m)').format(*ptSeedi))
        si.append(('(G) Glue seed size:  ' + multStrS + ' (m)').format(*gSeedi))

    # Vitreous
    si.append(('(V1) Vitreous seed size max (side edge seed set)-' +
               '(SINGLE BIAS):  ' + multStrS + ' (m)').format(*v1Seedi))
    si.append(('(V2) Vitreous seed size min (top edge in contact with ' +
               'retina):  ' + multStrS + ' (m)').format(*v2Seedi))

    # Retina
    si.append(('(R) Retina seed size:  ' + multStrS + ' (m)').format(*rSeedi))

    # Mass scale factor
    si.append(('(F) Mass scale factor:  ' + multStrL +
               '').format(*massScaleFactori))

    # Mass scale time increment
    si.append(('(MS) Mass scale time increment:  ' + multStrS +
               ' (s)').format(*massScaleTimeIncrementi))

    # Material properties (Young's Modulus)
    si.append("(RE) Retina Young's Modulus:  model value = {} (Pa)"
              .format(RetinaYoungsModulus_i))
    si.append("(VE) Vitreous Young's Modulus:  model value = {} (Pa)"
              .format(VitreousYoungsModulus_i))

    # Cohesive traction parameters
    if tieInterface == False:
        si.append(('(Kn) Knn:  ' + multStrL + ' (Pa)').format(*Knni))
        si.append(('(Ks) Kss:  ' + multStrL + ' (Pa)').format(*Kssi))
        si.append(('(Kt) Ktt:  ' + multStrL + ' (Pa)').format(*Ktti))

    # If True, then damage initation
    if DamageInitiation == True:
        si.append(('(tn) tn:  ' + multStrL + ' (Pa)').format(*tni))
        si.append(('(ts) ts:  ' + multStrL + ' (Pa)').format(*tsi))
        si.append(('(tt) tt:  ' + multStrL + ' (Pa)').format(*tti))

    # If True, then damage evolution
    if DamageInitiation == True and DamageEvolution == True:
        si.append(('(FE) Fracture energy:  ' + multStrL + ' (J)').format(*FEi))

    # Optimization
    if optimization is not None:
        si.append('Optimization of {}'.format(optimization))
        si.append('Objective function error formulation is the ' +
```

```
                    '{} calculation'.format(objErr))
        si.append('Objective error calculation is the {}'.format(errorMethod))

    if optimization == None:
        si.append('Parametric sweep')
        si.append('Objective function error formulation is the ' +
                    '{} calculation'.format(objErr))
        si.append('Objective error calculation is the {}'.format(errorMethod))

    # Tied interface
    if tieInterface == True:
        si.append('Tied interface between the Retina and the Vitreous')

    # Data trace being compared for optimization
    si.append('The data trace being compared is:  {}'
              .format(specificDataTrace))

    # Time shift info as it is a new capability
    si.append('The time prior to the peak force time event used for ' +
              'determining the linear region ' +
              'was extended ({}) '.format(timeBeforePeak) +
              'seconds before the actual peak')

    si.append(newLine)

    # Job description
    jobDescription =  '\n'.join(item for item in si)

    print(newLine)
    print(fileName)
    print(newLine)
    print(jobDescription)

    # Write a .txt file with the file attributes
    outfile = open(os.path.join(abqWD, fileName +'.txt'),'w')
    line = ('The file name indicates what parameters were used to define ' +
            'the model\n')
    outfile.write(line)
    line = '\n' + jobDescription + '\n'
    outfile.write(line)
    outfile.close()
    print(outfile)
    return namei, fileName, jobDescription


def GenerateAbaqusModels():
    """
    Function used to call Command Line (Windows Batch file)

    Parameters
    ----------
    fileName : abaqus job with paramters

    """
    # ----------------------- Step 2 -----------------------#
    # Generates the filename/description
    modelName, fileName, jobDescription = jobAttributes()

```

```
    # Strip job description from spaces and new lines
    # replace new lines, spaces, equal signs
    jobDescription = jobDescription.replace(' ', 'SPACE')
    jobDescription = jobDescription.replace('\n', 'NEWLINE')
    jobDescription = jobDescription.replace('\t', 'TAB')
    jobDescription = jobDescription.replace('=', 'EQUALSSIGN')

    print(newLine)

    # ---------------------- Step 3 ----------------------#
    # Calls Abaqus to create the job with the filename just created and
    # run the job

    # Strip spaces and make strings
    ABQ = []
    ABQ.append(pS_ABQ) # python 2.7 script

    # gravity
    ABQ.append(','.join([i.strip(' ') for i in str(gravity).split(',')]))

    # symmetry
    ABQ.append(','.join([i.strip(' ') for i in str(symmetry).split(',')]))

    # Simplified model
    ABQ.append(','.join([i.strip(' ') for i in str(simplified).split(',')]))

    ABQ.append(modelName) # model name
    ABQ.append(fileName) # file name

    # time
    ABQ.append(','.join([i.strip(' ') for i in str(time).split(',')]))

    # eye holder seed size 1
    ABQ.append(','.join([i.strip(' ') for i in str(e1Seedi).split(',')]))

    # eye holder seed size 2
    ABQ.append(','.join([i.strip(' ') for i in str(e2Seedi).split(',')]))

    # plastic tab seed size
    ABQ.append(','.join([i.strip(' ') for i in str(ptSeedi).split(',')]))

    # glue seed size
    ABQ.append(','.join([i.strip(' ') for i in str(gSeedi).split(',')]))

    # vitreous seed 1 size
    ABQ.append(','.join([i.strip(' ') for i in str(v1Seedi).split(',')]))

    # vitreous seed 2 size
    ABQ.append(','.join([i.strip(' ') for i in str(v2Seedi).split(',')]))

    # retina seed size
    ABQ.append(','.join([i.strip(' ') for i in str(rSeedi).split(',')]))

    # mass scale factor
    ABQ.append(','.join([i.strip(' ') for i in
                        str(massScaleFactori).split(',')]))

    # mass scale time
```

```python
    ABQ.append(','.join([i.strip(' ') for i in
                        str(massScaleTimeIncrementi).split(',')]))

    # Retina Young's Modulus
    ABQ.append(','.join([i.strip(' ') for i in
                        str(RetinaYoungsModulus_i).split(',')]))

    # Vitreous Young's Modulus
    ABQ.append(','.join([i.strip(' ') for i in
                        str(VitreousYoungsModulus_i).split(',')]))

    # Cohesive behavior
    ABQ.append(','.join([i.strip(' ') for i in str(Knni).split(',')])) # Knn
    ABQ.append(','.join([i.strip(' ') for i in str(Kssi).split(',')])) # Kss
    ABQ.append(','.join([i.strip(' ') for i in str(Ktti).split(',')])) # Ktt

    # DamageInitiation
    ABQ.append(','.join([i.strip(' ') for i in
                        str(DamageInitiation).split(',')]))
    ABQ.append(','.join([i.strip(' ') for i in str(tni).split(',')])) # tn
    ABQ.append(','.join([i.strip(' ') for i in str(tsi).split(',')])) # ts
    ABQ.append(','.join([i.strip(' ') for i in str(tti).split(',')])) # tt

    # DamageEvolution
    ABQ.append(','.join([i.strip(' ') for i in
                        str(DamageEvolution).split(',')]))
    ABQ.append(','.join([i.strip(' ') for i in str(FEi).split(',')])) # FE

    # Optimization None/optimized parameters
    ABQ.append(','.join([i.strip(' ') for i in str(optimization).split(',')]))

    # Tied interface
    ABQ.append(','.join([i.strip(' ') for i in str(tieInterface).split(',')]))

    # Model description
    ABQ.append(jobDescription)

    ABQ_parse_string = 'abaqus cae noGUI={} --' + (len(ABQ)-1)*' {}'

    # # Used for debugging, comment out to copy/paste output to cmd window
    # # to check and see if it works
    # print(ABQ_parse_string.format(*ABQ))
    # pdb.set_trace()

    cmd = subprocess.Popen(ABQ_parse_string.format(*ABQ),
                           cwd=r'{}'.format(abqWD), stdin=subprocess.PIPE,
                           stdout=subprocess.PIPE, stderr=subprocess.PIPE,
                           shell=True).communicate()[0]

    print(newLine)
    print('Abaqus has generated the .inp and executed the job')

    # ----------------------- Step 4 -----------------------#
    # Creates a folder with the filename
    folderDirectory = os.path.join(abqWD, fileName)
    if not os.path.exists(folderDirectory):
        os.makedirs(folderDirectory)
    dataDirectory = os.path.join(folderDirectory, 'Output')
```

```python
    if not os.path.exists(dataDirectory):
        os.makedirs(dataDirectory)
    figuresDirectory = os.path.join(dataDirectory, 'Figures')
    if not os.path.exists(figuresDirectory):
        os.makedirs(figuresDirectory)
    print(newLine)
    print('New file location:\n{} \n'.format(folderDirectory))

    # ----------------------- Step 5 ----------------------#
    """
    Extracts data from the Abaqus.odb file and creates two output files
    (Field/Hist).  Create the name to be parsed into ABQ from the command
    line through a subprocess
    """
    ABQ = []
    ABQ.append(pSE)
    ABQ.append(fileName)
    ABQ.append(gravity)
    ABQ.append(symmetry)
    ABQ.append(simplified)
    ABQ.append(DamageInitiation)
    ABQ.append(DamageEvolution)

    ABQ_parse_string = 'abaqus python' + len(ABQ)*' {}'

    # # # # Used for debugging, comment out to copy/paste output to cmd window
    # # # # to check and see if it works
    # print(ABQ_parse_string.format(*ABQ))
    # pdb.set_trace()

    cmd = subprocess.Popen(ABQ_parse_string.format(*ABQ),
                           cwd=r'{}'.format(abqWD), stdin=subprocess.PIPE,
                           stdout=subprocess.PIPE, stderr=subprocess.PIPE,
                           shell=True).communicate()[0]
    print(newLine)
    print('Abaqus has extracted Field/History output:  ' +
          '\n{} \n'.format(dataDirectory))

    # ----------------------- Step 6 ----------------------#
    # Plot data and store it in the variable name folder under "Figures"
    print(fileName)
    print(dataDirectory)
    PlotAbqData(fileName, dataDirectory, dataCompare, DamageInitiation,
                DamageEvolution)
    print(newLine)
    print('New data plots:\n{} \n'.format(figuresDirectory))

    # ----------------------- Step 7 ----------------------#
    # Move all abaqus files to the folder with the same name
    MoveAbqFiles(fileName, folderDirectory, abqWD)
    print(newLine)
    print('Files have been moved to:  \n{} \n'.format(dataDirectory))

    # ------------------- Step 8 (Error for minimization) -------------------#
    maxForceTime = 100 # s
    # slope is (mN/m)
    residVals = findResidual(fileName, dataDirectory, maxForceTime,
                             dataCompare, objErr, slopeFlag, maxForceFlag,
```

```
                              ssForceFlag, timeBeforePeak)
    # Unpack
    slopeSimulated =    residVals[0]
    maxForceSimulated = residVals[1]
    SSmeanSimulated =   residVals[2]
    SSmedianSimulated = residVals[3]
    y_new_exp_disp =    residVals[4]
    y_new_sim_disp =    residVals[5]

    # ------------Step 9 (Damage Initation for minimization) -------------#
    if DamageInitiation == True:
        retinaMaxUCRT, vitreousMaxUCRT = CSMAXSCRTAbqData(fileName,
                                                          dataDirectory,
                                                          maxForceTime,
                                                          dataCompare)
    else:
        retinaMaxUCRT = np.nan
        vitreousMaxUCRT = np.nan

    # (return slope, force, and maxucrt @ specified displacement)
    fcnReturn = []
    fcnReturn.append(fileName)
    fcnReturn.append(slopeSimulated)
    fcnReturn.append(maxForceSimulated)
    fcnReturn.append(SSmeanSimulated)
    fcnReturn.append(SSmedianSimulated)
    fcnReturn.append(retinaMaxUCRT)
    fcnReturn.append(vitreousMaxUCRT)
    fcnReturn.append(y_new_exp_disp)
    fcnReturn.append(y_new_sim_disp)
    return fcnReturn


def writeOutputData(fileNameList):
    print("\nWriting out the Reaction Force data...")
    filename = os.path.join(abqWD, 'FEAAttributes' + '.txt')
    outfile = open(filename,'w')
    sep = '\t'
    Header = [] # List of items for the header
    Header.append('FileName')
    Header.append('Time')
    Header.append('E1')
    Header.append('E2')
    Header.append('PT')
    Header.append('G')
    Header.append('V1')
    Header.append('V2')
    Header.append('R')
    Header.append('F')
    Header.append('MS')
    Header.append('RE')
    Header.append('VE')
    Header.append('Knn')
    Header.append('Kss')
    Header.append('Ktt')
    Header.append('DamageInitiation')
    Header.append('tn')
    Header.append('ts')
```

```
    Header.append('tt')
    Header.append('DamageEvolution')
    Header.append('FE')
    Header.append('Optimization')
    Header.append('TIE')
    Header.append('errorListL2Norm')
    Header.append('ObjectiveFunction')
    Header.append('simTime')
    line = sep.join(item for item in Header)
    outfile.write(line)
    outfile.write('\n')
    outfile.write('\t'.join(str(item) for item in attributeArray_0))
    for i in list(fileNameList):
        outfile.write('\n')
        tempList = [str(i[0])] # filename
        for j in list(i[1]):
            tempList.append(str(j)) # file attributes
        tempList.append(str(i[2])) # sim time
        outfile.write('\t'.join(str(item) for item in tempList))
    outfile.close()
    print("\nDone!")
    print("\nThe output file will be named '{}".format(filename) + "'")
    print("\nIt will be in the same working directory as your Abaqus model\n")

    # Print File of tests ran in order
    print("\nWriting out the Reaction Force data...")
    filename = os.path.join(abqWD, 'FEAFileList' + '.txt')
    outfile = open(filename,'w')
    line = 'FileName'
    outfile.write(line)
    for i in list(fileNameList):
        line = '\n%s' % (i[0])
        outfile.write(line)
    outfile.close()
    print("\nDone!")
    print("\nThe output file will be named '{}".format(filename) + "'")
    print("\nIt will be in the same working directory as your Abaqus model\n")



if __name__ == '__main__':
    # Run the function

    # ----------------------- Step 1 -----------------------#
    # T3
    name = ['T3']

    paramSelect = ReadRAWDataTrace(dataCompare, abqWD, timeBeforePeak)

    t0, t1, tshift, fe = paramSelect # Unpack variables

    if t0 > tshift:
        # If the t1 value is greater than tshfit, use tshift for
        # the simulation time
        # Shouldn't have to do this as this issue has been handeled
        t0 = tshift
        print('updated the time to be the shift value')

```

```
    # Determine which time to use (Max value or steady state)
    if optE_V == True:
        time = int(t0)
        FEValOpt = fe

    elif optKsTsFE == True or sweep == True:
        time = int(t1)
        FEValOpt = fe

    # Select input parameters
    # time = 97 # Simulation parameter time S25 shifted
    # time = 250 # Simulation parameter time

    ''' Optimized results using the updated optimization routine 2/11/21
    using the larger vitreous strip model with first looking at the tied
    interface between the vitreous and retina '''
    VitreousYoungsModulus_0 = 524.265652
    KnnValOpt = 26.312450336667535
    KssValOpt = 25.620054908304496
    KttValOpt = 27.028398378844223
    tnValOpt = 18.51999887865916
    tsValOpt = 17.98861859153288
    ttValOpt = 10.906247748496245
    # FEValOpt = -9.427062078905504

    """ Vitreous Young's Modulus """

    VitreousYoungsModulus_0 = 50.03617188307464 # optimized using Tie

    """ Retina Young's Modulus """
    RetinaYoungsModulus_0 = 11120.0 # Pa Optimized with the vitreous

    """ Eye holder inside edge """
    e1Seed_0 = 1 # Base seed
    e1SeedArray = [] # Array of multipliers
    n = 11 # number of increments
    for i in range(10, n):
        # Decrease mesh seed by a factor of 2
        e1SeedArray.append([i, e1Seed_0, e1Seed_0*(0.5)**i])

    """ Eye holder outside edge """
    # This will most likely never get smaller (saves computational time)
    e2Seed_0 = 1 # Base seed
    e2SeedArray = []
    n = 9 # number of increments
    for i in range(8, n):
        # Decrease mesh seed by a factor of 2
        e2SeedArray.append([i, e2Seed_0, e2Seed_0*(0.5)**i])

    """ Plastic tab """
    ptSeed_0 = 1 # Plastic tab seed size
    ptSeedArray = [] # Array of multipliers
    n = 7 # number of increments
    for i in range(6, n):
        # Decrease mesh seed by a factor of 2
        ptSeedArray.append([i, ptSeed_0, ptSeed_0*(0.5)**i])

    """ Glue """
```

```
    gSeed_0 = 1 # Glue seed size
    gSeedArray = [] # Array of multipliers
    n = 8 # number of increments
    for i in range(7, n):
        # Decrease mesh seed by a factor of 2
        gSeedArray.append([i, gSeed_0, gSeed_0*(0.5)**i])

    """ Vitreous """
    # smaller seed size
    v1Seed_0 = 1 # Vitreous (max seed size)
    v1SeedArray = [] # Array of multipliers
    # n = 30 # number of increments
    # for i in np.linspace(10, 12, n): # range(10, n):
    #     # Decrease mesh seed by a factor of 2
    #     v1SeedArray.append([i, v1Seed_0, v1Seed_0*(0.5)**i])

    # Comment out when parameters have been optimized
    v1ValOpt = 11.38 # (convergence value)
    v1SeedArray.append([v1ValOpt, v1Seed_0, v1Seed_0*(0.5)**v1ValOpt])

    # larger seed size (should be factor of 4 times smaller ## 2 numbers)
    v2Seed_0 = 1 # Vitreous (min seed size)
    v2SeedArray = [] # Array of multipliers
    # n = 9 # number of increments
    # for i in range(8, n):
    #     # Decrease mesh seed by a factor of 2
    #     v2SeedArray.append([i, v2Seed_0, v2Seed_0*(0.5)**i])

    # Comment out when parameters have been optimized
    v2ValOpt = 8
    v2SeedArray.append([v2ValOpt, v2Seed_0, v2Seed_0*(0.5)**v2ValOpt])

    """ Retina """
    rSeed_0 = 1 # Base seed
    rSeedArray = [] # Array of multipliers
    # n = 30 # number of increments
    # for i in np.linspace(10, 13.5, n): # range(10, n):
    #     # Decrease mesh seed by a factor of 2
    #     rSeedArray.append([i, rSeed_0, rSeed_0*(0.5)**i])

    rValOpt = 11.3275 # (convergence value)
    rSeedArray.append([rValOpt, rSeed_0, rSeed_0*(0.5)**rValOpt])

    """ mass scale factor """
    massScaleFactor_0 = 1
    massScaleFactorArray = [] # Array of multipliers
    n = 1 # number of increments
    for i in range(0, n):
        # Increase by a factor of 2
        massScaleFactorArray.append([i, massScaleFactor_0,
                                     massScaleFactor_0*2**i])

    """ mass scale time increment """
    massScaleTimeIncrement_0 = 1
    massScaleTimeArray = [] # multiplier and value
    n = 8 # number of increments
    for i in range(7, n):
        # Decrease by a factor of 2
```

```
        massScaleTimeArray.append([i, massScaleTimeIncrement_0,
                                   massScaleTimeIncrement_0*(0.5)**i])

    if massScaleTimeIncrement_0 == 0:
        print('No Mass Scaling... This will take a while...ABAQUS is ' +
              'deciding for us')

    """ Knn """
    Knn_0 = 1
    KnnArray = [] # Array of multipliers
    # n = 31 # number of increments # 23 works when R = 2e3, and V = 736 Pa
    # for i in range(30, n):
    #     # Increase by a factor of 2
    #     KnnArray.append([i, Knn_0, Knn_0*(2)**i])

    # Comment out when parameters have been optimized
    KnnArray.append([KnnValOpt, Knn_0, Knn_0*(2)**KnnValOpt])

    """ Kss """
    Kss_0 = 1
    KssArray = [] # Array of multipliers
    # n = 31
    # for i in range(30, n):
    #     # Increase by a factor of 2
    #     KssArray.append([i, Kss_0, Kss_0*(2)**i])

    # Comment out when parameters have been optimized
    KssArray.append([KssValOpt, Kss_0, Kss_0*(2)**KssValOpt])

    """ Ktt """
    Ktt_0 = 1
    KttArray = [] # Array of multipliers
    # n = 31
    # for i in range(30, n):
    #     # Increase by a factor of 2
    #     KttArray.append([i, Ktt_0, Ktt_0*(2)**i])

    # Comment out when parameters have been optimized
    KttArray.append([KttValOpt, Ktt_0, Ktt_0*(2)**KttValOpt])

    """ tn """
    tn_0 = 1
    tnArray = [] # Array of multipliers
    # n = 10 # 10 works when using max stress criteria
    # for i in range(9, n):
    #     # Increase by a factor of 2
    #     tnArray.append([i, tn_0, tn_0*(2)**i])

    # Comment out when parameters have been optimized
    tnArray.append([tnValOpt, tn_0, tn_0*(2)**tnValOpt])

    """ ts """
    ts_0 = 1
    tsArray = [] # Array of multipliers
    # n = 10
    # for i in range(9, n):
    #     # Increase by a factor of 2
    #     tsArray.append([i, ts_0, ts_0*(2)**i])
```

```

    # Comment out when parameters have been optimized
    tsArray.append([tsValOpt, tn_0, tn_0*(2)**tsValOpt])

    """ tt """
    tt_0 = 1
    ttArray = [] # Array of multipliers
    # n = 10
    # for i in range(9, n):
    #     # Increase by a factor of 2
    #     ttArray.append([i, tt_0, tt_0*(2)**i])

    # Comment out when parameters have been optimized
    ttArray.append([ttValOpt, tn_0, tn_0*(2)**ttValOpt])

    """ FE """
    FE_0 = 1
    FEArray = [] # Array of multipliers
    # n = -8
    # for i in range(-9, n):
    #     # Increase by a factor of 2
    #     FEArray.append([i, FE_0, FE_0*(2)**i])

    FEArray.append([FEValOpt, FE_0, FE_0*(2)**FEValOpt])

    errorList = np.nan # initial error
    slopeList = np.nan # Initial slope
    FmaxList = np.nan # Initial max peel force
    FSSList = np.nan # Initial steady-state peel force

    """ Attribute Array Initial Values """
    attributeArray_0 = []
    attributeArray_0.append('BaseVals')
    attributeArray_0.append(time)
    attributeArray_0.append(e1Seed_0)
    attributeArray_0.append(e2Seed_0)
    attributeArray_0.append(ptSeed_0)
    attributeArray_0.append(gSeed_0)
    attributeArray_0.append(v1Seed_0)
    attributeArray_0.append(v2Seed_0)
    attributeArray_0.append(rSeed_0)
    attributeArray_0.append(massScaleFactor_0)
    attributeArray_0.append(massScaleTimeIncrement_0)
    attributeArray_0.append(RetinaYoungsModulus_0)
    attributeArray_0.append(VitreousYoungsModulus_0)
    attributeArray_0.append(Knn_0)
    attributeArray_0.append(Kss_0)
    attributeArray_0.append(Ktt_0)
    attributeArray_0.append(DamageInitiation)
    attributeArray_0.append(tn_0)
    attributeArray_0.append(ts_0)
    attributeArray_0.append(tt_0)
    attributeArray_0.append(DamageEvolution)
    attributeArray_0.append(FE_0)
    attributeArray_0.append(optimization)
    attributeArray_0.append(tieInterface)
    attributeArray_0.append(errorList)
    attributeArray_0.append(objErr)
```

```
    attributeArray_0.append(slopeList)
    attributeArray_0.append(FmaxList)
    attributeArray_0.append(FSSList)
    attributeArray_0.append('simTime')


    fileNameList = [] # List of files
    counter = 0

    if optimization is not None:
        """ If the optimization variable is not "None" then optimize the
        specific variable beins passed through """

        name = name[0]

        # BondStatus = True # interested in bonding

        # # post damage failure model (If False, ignore Ktt, ts, and tn,
        # # otherwise include them)
        # pdfm = False

        # Optimization method
        # optName = 'NM' # Nelder-mead
        # optName = 'P' # Powell
        optName = 'C' # COBYLA
        # optName = 'L' # LBFGSB
        # optName = 'T' # Truncated Newton
        # optName = 'S' # SLSQP
        # optName - 'TC' # Trust-Constr

        name0 = '_'.join([name, optName]) # used for optimization

        def FEA_Residual(pars, data=None):
            # Global variables
            global counter
            global name
            global name0
            global fileNameList
            global time
            global namei
            global e1Seedi
            global e2Seedi
            global ptSeedi
            global gSeedi
            global v1Seedi
            global v2Seedi
            global rSeedi
            global massScaleFactori
            global massScaleTimeIncrementi
            global RetinaYoungsModulus_i
            global VitreousYoungsModulus_i
            global Knni
            global Kssi
            global Ktti
            global tni
            global tsi
            global tti
            global FEi
```

```
        # Parameters used for optimization
        global errorList

        print('Iteration # ', counter)

        tic() # Start time

        e1Seedi = e1SeedArray[0] # Default array
        e2Seedi = e2SeedArray[0] # Default array
        ptSeedi = ptSeedArray[0] # Default array
        gSeedi = gSeedArray[0] # Default array
        v1Seedi = v1SeedArray[0] # Default array
        v2Seedi = v2SeedArray[0] # Default array
        rSeedi = rSeedArray[0] # Default array
        massScaleFactori = massScaleFactorArray[0] # Default array
        massScaleTimeIncrementi = massScaleTimeArray[0] # Default array
        RetinaYoungsModulus_i = RetinaYoungsModulus_0 # Default value
        VitreousYoungsModulus_i = VitreousYoungsModulus_0 # Default value
        Knni = KnnArray[0] # Default array
        Kssi = KssArray[0] # Default array
        Ktti = KttArray[0] # Default array
        tni = tnArray[0] # Default array
        tsi = tsArray[0] # Default array
        tti = ttArray[0] # Default array
        FEi = FEArray[0] # Default array

        # Extract the unknown parameters from the pars class variable
        # Determine the multiplier for the title
        for key, value in pars.items():

            if key.find('ER') >= 0:
                """ Retina Young's Modulus """
                val = value.value
                RetinaYoungsModulus_i = val

            elif key.find('EV') >= 0:
                """ Vitreous Young's Modulus """
                val = value.value
                VitreousYoungsModulus_i = val

            elif key.find('Knn') >= 0:
                """ Knn """
                val = value.value
                mult = np.log(val)/np.log(2) # multiplier
                Knni = [mult, Knn_0, val]

            elif key.find('Kss') >= 0:
                """ Kss """
                val = value.value
                mult = np.log(val)/np.log(2) # multiplier
                Kssi = [mult, Kss_0, val]

            elif key.find('Ktt') >= 0:
                """ Ktt """
                val = value.value
                mult = np.log(val)/np.log(2) # multiplier
                Ktti = [mult, Ktt_0, val]
```

```python

                elif key.find('tn') >= 0:
                    """ tn """
                    val = value.value
                    mult = np.log(val)/np.log(2) # multiplier
                    tni = [mult, tn_0, val]

                elif key.find('ts') >= 0:
                    """ ts """
                    val = value.value
                    mult = np.log(val)/np.log(2) # multiplier
                    tsi = [mult, ts_0, val]

                elif key.find('tt') >= 0:
                    """ tt """
                    val = value.value
                    mult = np.log(val)/np.log(2) # multiplier
                    tti = [mult, tt_0, val]

                elif key.find('FE') >= 0:
                    """ FE """
                    val = value.value
                    mult = np.log(val)/np.log(2) # multiplier
                    FEi = [mult, FE_0, val]

            # Keep track of simulation results by unique names with the count
            # number.  Comment out the second part to save file space if you
            # are not interested in saving every single simulation
            namei = name0 #+ str(counter)

            # Error of the simulation
            L2Normi = np.sqrt(np.dot(errorList, errorList))

            # multipliers to be appended to the output file to show changes
            # in parameters
            aAM = [] # attributeArrayMultipliar
            aAM.append(time)
            aAM.append(e1Seedi[0])
            aAM.append(e2Seedi[0])
            aAM.append(ptSeedi[0])
            aAM.append(gSeedi[0])
            aAM.append(v1Seedi[0])
            aAM.append(v2Seedi[0])
            aAM.append(rSeedi[0])
            aAM.append(massScaleFactori[0])
            aAM.append(massScaleTimeIncrementi[0])
            aAM.append(RetinaYoungsModulus_i)
            aAM.append(VitreousYoungsModulus_i)
            aAM.append(Knni[0])
            aAM.append(Kssi[0])
            aAM.append(Ktti[0])
            aAM.append(DamageInitiation)
            aAM.append(tni[0])
            aAM.append(tsi[0])
            aAM.append(tti[0])
            aAM.append(DamageEvolution)
            aAM.append(FEi[0])
            aAM.append(optimization)
```

```
            aAM.append(tieInterface)
            aAM.append(L2Normi)
            aAM.append(objErr)

            # Call the function
            # Runs jobs and saves file names
            funReturn = GenerateAbaqusModels()
            fileName =           funReturn[0]
            maxSlopeSimulated =  funReturn[1]
            maxForceSimulated =  funReturn[2]
            SSmeanSimulated =    funReturn[3]
            SSmedianSimulated =  funReturn[4]
            retinaMaxUCRT =      funReturn[5]
            vitreousMaxUCRT =    funReturn[6]
            y_new_exp_disp =     funReturn[7]
            y_new_sim_disp =     funReturn[8]

            # add the simulated outputs to the data file
            aAM.append(maxSlopeSimulated)
            aAM.append(maxForceSimulated)
            aAM.append(SSmedianSimulated)

            # Determine the measure of error used for optimization
            # Let the data trace being passed in act as the comparison
            maxSlopeMeasured, maxForceMeasured = data

           # Error calculation
            errorDict = {} # Dictionary
            if objErr == 'Difference':
                errorDict['slope']    = (maxSlopeMeasured - maxSlopeSimulated) if slopeFlag == True else []
                errorDict['maxForce'] = (maxForceMeasured - maxForceSimulated) if maxForceFlag == True else []
                errorDict['ssForce']  = (SS_Measured - SSmeanSimulated)        if ssForceFlag == True else []
            elif objErr == 'Ratio':
                errorDict['slope']    = (1 - maxSlopeMeasured / maxSlopeSimulated) if slopeFlag == True else []
                errorDict['maxForce'] = (1 - maxForceMeasured / maxForceSimulated) if maxForceFlag == True else []
                errorDict['ssForce']  = (1 - SS_Measured / SSmeanSimulated)        if ssForceFlag == True else []
            elif objErr == 'Relative uncertainty':
                errorDict['slope']    = ((maxSlopeMeasured - maxSlopeSimulated)/maxSlopeMeasured) if slopeFlag == True else []
                errorDict['maxForce'] = ((maxForceMeasured - maxForceSimulated)/maxForceMeasured) if maxForceFlag == True else []
                errorDict['ssForce']  = ((SS_Measured - SSmedianSimulated)/SS_Measured)           if ssForceFlag == True else []
            else:
                print('Error in MaxForceError')
                sys.exit()

            # Error array values
            errorList = list(errorDict.values()) # convert to list
            errorList = [x for x in errorList if x] # get rid of empty values
```

```
            L2Normi = np.sqrt(np.dot(errorList, errorList))

            # Calculate residual
            residual = y_new_exp_disp - y_new_sim_disp # residual

            # Calculate L2Norm
            L2Norm = np.sqrt(np.dot(residual, residual))

            simulationTime = toc() # Determine run time
            # apends the fileName & File Attributes
            fileNameList.append([fileName, aAM,
                                 simulationTime])
            print('{} Error calculation: '.format(objErr), errorList)
            print('L2 norm objective calculation', L2Normi)
            print('L2 Norm residual', L2Norm)
            print('Done\n\n\n')
            counter += 1

            # Determine which calculation is going to be used for optimization
            if errorMethod == 'two-point method':
                FEA_Residual = errorList
            elif errorMethod == 'data-trace method':
                FEA_Residual = residual

            return FEA_Residual

        maxFuncEval = 200
        tolVal = 1e-4

        # Use the data variable to input the max slope and force from the
        # known data trace
        data = [maxSlopeMeasured, maxForceMeasured]

        # Initial, Upper, and Lower bounds for parameters

        # Young's Modulus - Retina
        ER_i = 5000 # Pa
        ER_LB = 50 # Pa
        ER_UB = 11000 # Pa

        # Young's Modulus - Vitreous
        EV_i = 172 # Pa (Prony series calculation)
        # EV_i = 500 # Pa (Trying higher initial guess)
        EV_LB = 50 # Pa
        EV_UB = 2100 # Pa
        # EV_UB = 400 # Pa (Lowering the upper bound)

        # Traction-Separation Behavior
        # Knn_i = 2**18 # Stress [Pa]
        # Knn_i = 2**26.32642676301851 # better optimized guess
        Knn_i = 2**20.872765304828103 # Stress [Pa] # different guess Low E_v
        # Knn_LB = 2**22 # Stress [Pa]
        Knn_LB = 2**10 # Stress [Pa] # Try lowering the bound
        Knn_UB = 2**28 # Stress [Pa]

        # Kss_i = 2**18 # Stress [Pa]
        # Kss_i = 2**27.387981486684094 # better optimized guess
        Kss_i = 2**26.094732037712763 # different guess Low E_v
```

```
        # Kss_LB = 2**22 # Stress [Pa]
        Kss_LB = 2**10 # Stress [Pa] # Try lowering the bound
        Kss_UB = 2**28 # Stress [Pa]

        # Ktt_i = 2**18 # Stress [Pa]
        # Ktt_i = 2**27.88464867824286 # better optimized guess
        Ktt_i = 2**26.20110650892766 # different guess Low E_v
        # Ktt_LB = 2**22 # Stress [Pa]
        Ktt_LB = 2**10 # Stress [Pa] # Try lowering the bound
        Ktt_UB = 2**28# # Stress [Pa]

        # Damage Initiation Behavior
        # tn_i = 2**9 # Stress [Pa]
        # tn_i = 2**18.51999887865916 # better optimized guess
        tn_i = 2**9.712181223168551 # different guess Low E_v
        tn_LB = 2**3 # Stress [Pa]
        tn_UB = 2**20 # Stress [Pa] 11 before

        # ts_i = 2**9 # Stress [Pa]
        # ts_i = 2**17.98861859153288 # better optimized guess
        ts_i = 2**9.931687876075074 # different guess Low E_v
        ts_LB = 2**3 # Stress [Pa]
        ts_UB = 2**20 # Stress [Pa] 11 before

        # tt_i = 2**9 # Stress [Pa]
        # tt_i = 2**10.906247748496245 # better optimized guess
        tt_i = 2**9.022372079206395 # different guess Low E_v
        tt_LB = 2**3 # Stress [Pa]
        tt_UB = 2**15 # Stress [Pa] 11 before

        # Damage Evolution Behavior
        # FE_i = 3.738925970000001e-6 # Energy [J]  ~ -18.028944662923816 # S25
        FE_i = 1.929e-6 # Energy [J]  S47
        # FE_i = 2**-9.427062078905504 # better optimized guess
        FE_LB = 2**-30 # 2**0 # Energy [J]
        # FE_UB = 2**-8 # Energy [J] # small bounds on energy
        FE_UB = 2**0 # Energy [J] # increase bounds

        # Specify parameters
        fit_params = lf.Parameters() # intialize the class for parameters

        # Retina young's modulus
        if optimization.find('E_R') >= 0:
            fit_params.add('ER', value = ER_i, min=ER_LB, max=ER_UB, vary=True)

        # Vitreous Young's Modulus
        if optimization.find('E_V') >= 0:
            fit_params.add('EV', value = EV_i, min=EV_LB, max=EV_UB, vary=True)

        # parameter for making the retina stiffer than the vitreous
        if optimization.find('E_R') >= 0 and optimization.find('E_V') >= 0:
            fit_params.add('StiffDelta', value = 0.01, min=0, vary=True)
            # Constraint to allow vitreous to be not as stiff as the retina
            fit_params.add('stiffnessConstraint', expr = 'EV - StiffDelta')

        # Knn
        if optimization.find('K_nn') >= 0:
            fit_params.add('Knn', value = Knn_i, min=Knn_LB, max=Knn_UB,
```

```
                            vary=True)

        # Kss
        if optimization.find('K_ss') >= 0:
            fit_params.add('Kss', value = Kss_i, min=Kss_LB, max=Kss_UB,
                            vary=True)

        # Ktt
        if optimization.find('K_tt') >= 0:
            fit_params.add('Ktt', value = Ktt_i, min=Ktt_LB, max=Ktt_UB,
                            vary=True)

        # tn
        if optimization.find('t_n') >= 0:
            fit_params.add('tn', value = tn_i, min=tn_LB, max=tn_UB,
                            vary=True)

        # ts
        if optimization.find('t_s') >= 0:
            fit_params.add('ts', value = ts_i, min=ts_LB, max=ts_UB,
                            vary=True)

        # tt
        if optimization.find('t_t') >= 0:
            fit_params.add('tt', value = tt_i, min=tt_LB, max=tt_UB,
                            vary=True)

        # FE
        if optimization.find('FE') >= 0:
            fit_params.add('FE',  value = FE_i, min=FE_LB, max=FE_UB,
                            vary=True)

        # Set up minimization class
        minClass = lf.Minimizer(FEA_Residual, fit_params,
                                fcn_kws={'data': data},
                                max_nfev = maxFuncEval) # fcn_args=(x,),

        # (Different methods can be used here) Uses an array
        # out = minClass.leastsq() # Levenberg-Marquardt

        # single scalar value
        # out = minClass.scalar_minimize(method='Nelder-Mead', tol=tolVal)

        # single scalar value (if the objective function returns an array,
        # the sum of the squares of the array will be used (L2Norm))
        out = minClass.scalar_minimize(method='Cobyla', tol=tolVal)

        lf.report_fit(out) # modelpars=p_true,  show_correl=True

        # Write data to txt files
        writeOutputData(fileNameList)

    else:

        # Number of simulations to perform (Simulation Batch Total)
        SBT = []
        SBT.append(len(name))
        SBT.append(len(e1SeedArray))
```

```
        SBT.append(len(e2SeedArray))
        SBT.append(len(ptSeedArray))
        SBT.append(len(gSeedArray))
        SBT.append(len(v1SeedArray))
        SBT.append(len(v2SeedArray))
        SBT.append(len(rSeedArray))
        SBT.append(len(massScaleFactorArray))
        SBT.append(len(massScaleTimeArray))
        SBT.append(len(KnnArray))
        SBT.append(len(KssArray))
        SBT.append(len(KttArray))
        SBT.append(len(tnArray))
        SBT.append(len(tsArray))
        SBT.append(len(ttArray))
        SBT.append(len(FEArray))

        ZipArray = []
        ZipArray.append(max(SBT)*name)
        ZipArray.append(max(SBT)*e1SeedArray)
        ZipArray.append(max(SBT)*e2SeedArray)
        ZipArray.append(max(SBT)*ptSeedArray)
        ZipArray.append(max(SBT)*gSeedArray)
        ZipArray.append(max(SBT)*v1SeedArray)
        ZipArray.append(max(SBT)*v2SeedArray)
        ZipArray.append(max(SBT)*rSeedArray)
        ZipArray.append(max(SBT)*massScaleFactorArray)
        ZipArray.append(max(SBT)*massScaleTimeArray)
        ZipArray.append(max(SBT)*KnnArray)
        ZipArray.append(max(SBT)*KssArray)
        ZipArray.append(max(SBT)*KttArray)
        ZipArray.append(max(SBT)*tnArray)
        ZipArray.append(max(SBT)*tsArray)
        ZipArray.append(max(SBT)*ttArray)
        ZipArray.append(max(SBT)*FEArray)

        # Iterate over the different combinations of parameters
        # If varying one parameter, then use iter.product(items in list...)
        # If varying multiple parameters, use zip*max(SBT)*items in list...)

        for (namei,
             e1Seedi,
             e2Seedi,
             ptSeedi,
             gSeedi,
             v1Seedi,
             v2Seedi,
             rSeedi,
             massScaleFactori,
             massScaleTimeIncrementi,
             Knni,
             Kssi,
             Ktti,
             tni,
             tsi,
             tti,
             FEi) in zip(*ZipArray):
            tic() # Start time
            counter += 1
```

```python
            print(counter, 'of ', max(*SBT))

            namei = namei + '_{}'.format(counter)

            # set the i'th value to the initial value (Updated in
            # optimization algorithm)
            RetinaYoungsModulus_i = RetinaYoungsModulus_0
            VitreousYoungsModulus_i = VitreousYoungsModulus_0

            # Call the function
            # Runs jobs and saves file names
            funReturn = GenerateAbaqusModels()
            fileName =           funReturn[0]
            maxSlopeSimulated =  funReturn[1]
            maxForceSimulated =  funReturn[2]
            SSmeanSimulated =    funReturn[3]
            SSmedianSimulated =  funReturn[4]
            retinaMaxUCRT =      funReturn[5]
            vitreousMaxUCRT =    funReturn[6]
            y_new_exp_disp =     funReturn[7]
            y_new_sim_disp =     funReturn[8]

            # Error calculation
            errorDict = {} # Dictionary
            if objErr == 'Difference':
                errorDict['slope']    = (maxSlopeMeasured - maxSlopeSimulated) if slopeFlag == True else []
                errorDict['maxForce'] = (maxForceMeasured - maxForceSimulated) if maxForceFlag == True else []
                errorDict['ssForce']  = (SS_Measured - SSmeanSimulated)        if ssForceFlag == True else []
            elif objErr == 'Ratio':
                errorDict['slope']    = (1 - maxSlopeMeasured / maxSlopeSimulated) if slopeFlag == True else []
                errorDict['maxForce'] = (1 - maxForceMeasured / maxForceSimulated) if maxForceFlag == True else []
                errorDict['ssForce']  = (1 - SS_Measured / SSmeanSimulated)        if ssForceFlag == True else []
            elif objErr == 'Relative uncertainty':
                errorDict['slope']    = ((maxSlopeMeasured - maxSlopeSimulated)/maxSlopeMeasured) if slopeFlag == True else []
                errorDict['maxForce'] = ((maxForceMeasured - maxForceSimulated)/maxForceMeasured) if maxForceFlag == True else []
                errorDict['ssForce']  = ((SS_Measured - SSmedianSimulated)/SS_Measured)           if ssForceFlag == True else []
            else:
                print('Error in MaxForceError')
                sys.exit()

            # Error array values
            errorList = list(errorDict.values()) # convert to list
            errorList = [x for x in errorList if x] # get rid of empty values

            # Error of the simulation
            L2Normi = np.sqrt(np.dot(errorList, errorList))
```

```
            # Calculate residual
            residual = y_new_exp_disp - y_new_sim_disp # residual

            # Calculate L2Norm
            L2Norm = np.sqrt(np.dot(residual, residual))

            # multipliers to be appended to the output file to show changes
            # in parameters
            aAM = [] # attributeArrayMultipliar
            aAM.append(time)
            aAM.append(e1Seedi[0])
            aAM.append(e2Seedi[0])
            aAM.append(ptSeedi[0])
            aAM.append(gSeedi[0])
            aAM.append(v1Seedi[0])
            aAM.append(v2Seedi[0])
            aAM.append(rSeedi[0])
            aAM.append(massScaleFactori[0])
            aAM.append(massScaleTimeIncrementi[0])
            aAM.append(RetinaYoungsModulus_i)
            aAM.append(VitreousYoungsModulus_i)
            aAM.append(Knni[0])
            aAM.append(Kssi[0])
            aAM.append(Ktti[0])
            aAM.append(DamageInitiation)
            aAM.append(tni[0])
            aAM.append(tsi[0])
            aAM.append(tti[0])
            aAM.append(DamageEvolution)
            aAM.append(FEi[0])
            aAM.append(optimization)
            aAM.append(tieInterface)
            aAM.append(L2Normi)
            aAM.append(objErr)
            aAM.append(maxSlopeSimulated)
            aAM.append(maxForceSimulated)
            aAM.append(SSmedianSimulated)

            simulationTime = toc() # Determine run time
            # apends the fileName & File Attributes
            fileNameList.append([fileName, aAM,
                                 simulationTime])
            print('{} Error calculation: '.format(objErr), errorList)
            print('L2 norm objective calculation', L2Normi)
            print('L2 Norm residual', L2Norm)
            print('Done')

        # Write data to txt files
        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.*

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jan 19 15:07:50 2021

@author: Kiffer2
"""

import numpy as np
import pandas as pd
import os
import sys
import matplotlib.pyplot as plt
from matplotlib.pyplot import cm
import matplotlib.patheffects as pe
from matplotlib.patches import Polygon
plt.rcParams['figure.figsize'] = [16, 9]
from scipy import interpolate
import pdb


# # Define the location of the Abaqus Working Directory
# # specific folder path where this file is located
# pythonScriptPath = os.getcwd()
# abqWD, pythonFiles = os.path.split(pythonScriptPath) # split file path

# filePath = os.getcwd() # current working directory
# codePath, pythonFolder = os.path.split(filePath) # split file path
# HWPath, codesFolder = os.path.split(codePath) # split file path

# expDataPath = 'experimentalData' # folder of data files
# dataPath = os.path.join(HWPath, expDataPath) # Path to data files

def Least_Squares(x,y):
    """
    Calculate the slope and y-intercept using matrix math
    x & y are the coordinates of points

    parameters (X,Y) Data

    Returns:
        Curve fit data and parameters m*x + b, R squared value
    """
    Z = np.ones((len(x),2))
    Z[:,1] = x
    # Calculate the matrix inverse for the constants of the regression
    A = np.dot(np.linalg.inv(np.dot(Z.T,Z)),(np.dot(Z.T,y)))
    linFit = x*A[1] + A[0]

    # Stats
    SS_tot = np.sum((y - np.mean(y))**2)
```

```
    SS_res = np.sum((y - linFit)**2)
    Rsqd = 1 - SS_res/SS_tot

    return linFit, A, Rsqd

def myformat(x):
    myexp = int(np.floor(np.log10(x)))
    xout = x*10**(-myexp)
    strout = '{:.4f}'.format(xout) + '\cdot10^{' + '{}'.format(myexp) + '}'
    return strout


# In[previous data]

def ReadRAWDataTrace(dataPath, abqWD, timeBeforePeak):
    """
    Inputs: dataPath - file path to raw data
    abqWD: abaqus working directory
    timeBeforePeak: number of seconds prior to the peak where data will
                    be extrapolated to the origin for curve fitting
    """

    timeBeforePeak = timeBeforePeak*10 # Convert s --> cs (10 data points/sec)

    # Eliminate the file extension
    dataPathNoExt = dataPath.split('.txt')[0]

    # Determine the specific file name
    fileDir, dataCompare = os.path.split(dataPathNoExt)

    """ Read in the csv file """
    dfValsn = pd.read_csv(dataPath, sep="\t", nrows=29, header=None,
                          names=['Var', 'Attribute'])

    """ File Attributes """
    HID =           dfValsn['Attribute'][0]
    HAGE =          dfValsn['Attribute'][1]
    HG =            dfValsn['Attribute'][2]
    HLR =           dfValsn['Attribute'][3]
    HR =            dfValsn['Attribute'][4]
    HSSi =     float(dfValsn['Attribute'][12])
    HSSf =     float(dfValsn['Attribute'][13])
    HTMax =    float(dfValsn['Attribute'][14])
    HDispMax = float(dfValsn['Attribute'][15])
    HFMax =    float(dfValsn['Attribute'][16]) # (mN)
    HFSS =     float(dfValsn['Attribute'][17])
    # slope from 20 seconds prior to max force value
    HSlope20 = float(dfValsn['Attribute'][20]) # (mN/m)

    dfn = pd.read_csv(dataPath, sep="\t", header=30)
    dfn.columns = ['Time', 'Extension', 'Force']
    dfn_time = dfn.Time
    dfn_extension = dfn.Extension # mm
    dfn_force = dfn.Force*1e3 # N ---> mN

    # SS Array
    ssTimeArray = np.array([HSSi, HSSf])
    ssValArray = np.array([HFSS, HFSS])
```

```

    # slope calculation for 20 seconds prior to the max peel force
    # (Experimental Data)
    maxIndex = dfn_time[dfn_time == HTMax].index.values[0]

    # to location of max force
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
    t_n = dfn_extension[maxIndex - timeBeforePeak:maxIndex]
    # to location of max force
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
    y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
    # Perform least squares and return
    curveFit_n, Params_n, R_Squared_n = Least_Squares(t_n,y)

    # Shift extension data so that the linear region is extrapolated through
    # the origin
    shift = abs(Params_n[0]/Params_n[1])*0
    dfn_extension = dfn_extension - shift

    # Now that the data has been shifted, recalculate the linear regression
    # using the reduced data set

    # to location of max force
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
    t_n = dfn_extension[maxIndex - timeBeforePeak:maxIndex]
    # to location of max force
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
    y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
    # Perform least squares and return
    curveFit_n, Params_n, R_Squared_n = Least_Squares(t_n,y)

    # # Slope of the curve up to the max force !!!(from the simulated data)!!!
    # adjustDisp = min(dn, key=lambda x:abs(x - dfn_extension[maxIndex]))
    # index = RF[dn == adjustDisp].index.values[0]
    # simulationCriteria = index # Time before peak force for curve fitting
    # # Array from 0 to location of max force
    # x = dn[index - simulationCriteria:index]
    # # Array from 0 to location of max force
    # y = RF[index - simulationCriteria:index]
    # # Perform least squares
    # curveFit, Params, R_Squared = Least_Squares(x,y)

    # # Updated force at specific max disp with adjusted value (Simulated data)
    # specificTime = maxForceTime
    # actualDisp = min(dn, key=lambda x:abs(x - dfn_extension[maxIndex]))
    # force_at_Disp = RF[dn == actualDisp].values[0]

    # # Simulated max force
    # simMaxForce = RF.max() # maximum simulated force value
    # simMaxDisp = dn[RF == simMaxForce] # displacement at the max force value

    # Max peel force displacement at max and steady state
    dfn_max_Disp = dfn_extension[dfn_time == HTMax]
    # Didn't seem to work here
    # dfn_ss_Disp = np.array([dfn_extension[dfn_time == HSSi],
    #                          dfn_extension[dfn_time == HSSf]]).flatten()
    dfn_ss_Disp = [dfn_extension[dfn_time == HSSi].values[0],
                   dfn_extension[dfn_time == HSSf].values[0]]
```

```python

# In[Simulated Trace]

# dataDirectory = 'D:\Downloads\experimentalData'

# fileName = ('output_Field_S25CohesiveXLVitDiff_CT250S11' +
#             'SF0MS7RE1e_04VE5e_02opt.txt')

# df = pd.read_csv(os.path.join(dataDirectory, fileName),
#                  sep="\t", header=0)

# Header = [] # Header information for the dataframe
# Header.append('Frame') #                    h1
# Header.append('Time') #                     h2
# Header.append('RF_y_dot') #                 h3
# Header.append('RFx') #                      h4
# Header.append('RFy') #                      h5
# Header.append('RFz') #                      h6
# Header.append('Nodal_Force') #              h7
# Header.append('Tab_Displacement') #         h8
# Header.append('Bond_Displacement') #        h9
# Header.append('Stress') #                   h10
# Header.append('AVG_CSMAXSCRT') #            h11
# Header.append('AVG_CSDMG') #                h12
# df.columns = Header

# tt = df.Time
# RF = df.RF_y_dot*1000 # N to mN
# dn = df.Tab_Displacement*1000 # m

# In[Plots]

""" Plots """
# Plot the data trace to compare the simulated results with the force
# displacement curves
fig, ax = plt.subplots()
ax.plot(dfn_extension, dfn_force,'-', color='r', linewidth=1,
        markersize=2, label = '{}, Age: {}'.format(HID, HAGE),
        alpha = 0.5)

if str(HFMax) == 'nan' and str(HSSi) == 'nan':
    print('No max or steady state')
    pass

if str(HFMax) != 'nan':
    ax.plot(dfn_max_Disp, HFMax,'.', color='k', linewidth=1,
            markersize=20,
            label = 'Max Peel - {:.4f} (mN)'.format(HFMax),
            path_effects=[pe.Stroke(linewidth=4, foreground='k'),
                          pe.Normal()])
    ax.plot(t_n, curveFit_n, '-', color='tab:blue', linewidth=2,
            label=r'Curve fit Max - {}'.format(int(timeBeforePeak/10)) +
            ' (s) y = {:.4f}x '.format(Params_n[1]) +
            '+ {:.4f} (mN), '.format(Params_n[0]) +
            '$r^2$ = {:.4f}'.format(R_Squared_n),
            alpha = 1)

if str(HSSi) != 'nan':
```

```
        ax.plot(dfn_ss_Disp, ssValArray,'-', color='c', linewidth=3,
                markersize=2,
                label = 'Steady State - {:.4f} (mN)'.format(HFSS),
                path_effects=[pe.Stroke(linewidth=5,
                                        foreground='k'),
                              pe.Normal()])

    # Make the shaded region for the entire integral
    a = dfn_max_Disp.values[0] # dfn_ss_Disp[0]
    b = dfn_ss_Disp[0] # dfn_ss_Disp[1]

    # Make the shaded region include the square below
    adjust = 0 # 0 or 1 to get rid of the small square

    # Filter the data in between the bounds
    dfn_ext_adjust = dfn_extension[(dfn_extension >= a) & (dfn_extension < b)]
    dnf_force_adjust = dfn_force[(dfn_extension >= a) & (dfn_extension < b)]

    verts = [(a, HFSS*adjust),
             *zip(dfn_ext_adjust, dnf_force_adjust),
             (b, HFSS*adjust)]
    poly = Polygon(verts, facecolor='0.8', edgecolor='0.5')
    ax.add_patch(poly)

    # Integral area
    Integral = np.trapz(dnf_force_adjust - HFSS*adjust, dfn_ext_adjust)

    # Centroid for plotting
    CentroidX = 1/Integral*(np.trapz(dfn_ext_adjust*(dnf_force_adjust -
                                                     HFSS*adjust),
                                     dfn_ext_adjust))
    CentroidY = 1/Integral*(np.trapz((dnf_force_adjust**2 -
                                      (HFSS*adjust)**2*adjust)/2,
                                     dfn_ext_adjust))

    # ax.text(b, (HFMax + HFSS)/2, r'$\int_a^b f(x)\mathrm{d}x=' +
    #         myformat(Integral*1e-6) + '$ (J)', horizontalalignment='center',
    #         fontsize=20)
    # ax.plot([0.5*max(dfn_extension), CentroidX], [0.5*max(dfn_force),
    #                                                CentroidY])

    prop = dict(arrowstyle="-|>,head_width=0.4, head_length=0.8", shrinkA=0,
                shrinkB=0)
    # ax.arrow(0.5*max(dfn_extension), 0.5*max(dfn_force),
    #          CentroidX - 0.5*max(dfn_extension),
    #          CentroidY - 0.5*max(dfn_force),
    #          head_width=0.1, head_length=0.1)
    ax.annotate("", xy=(CentroidX, CentroidY), xytext=(0.5*max(dfn_extension),
                                                       0.5*max(dfn_force)),
                arrowprops=prop)

    ax.text(0.5*max(dfn_extension), 0.52*max(dfn_force),
            r'$\int_a^b f(x)\mathrm{d}x=' + myformat(Integral*1e-6) + '$ (J)',
            horizontalalignment='center', fontsize=20)

    ax.spines['right'].set_visible(False)
    ax.spines['top'].set_visible(False)
    ax.xaxis.set_ticks_position('bottom')
```

```python

    ax.set_xticks((a, b))
    ax.set_xticklabels(('${}$'.format(a), '${}$'.format(b)))
    ax.set_yticks((HFSS, HFMax))
    ax.set_yticklabels(('${:.5}$'.format(HFSS), '${:.5}$'.format(HFMax)))

    ################## Plot Data #########################
    plt.axhline(0,color='black') # x = 0
    plt.axvline(0,color='black') # y = 0
    plt.ylabel('Force (mN)',fontsize=18)
    plt.xlabel('Displacement (mm)',fontsize=18)
    plt.title('Vitreous',fontsize=20)
    plt.grid()
    plt.legend(loc = 'best',fontsize = 'medium')
    plt.savefig(os.path.join(abqWD, 'GcSelection.pdf'), dpi=300,
                bbox_inches='tight')
    # plt.show()
    plt.close()



    # """ Derivative of the data trace """
    # fig, ax = plt.subplots()

    # deriv = np.gradient(dfn_force, dfn_extension)

    # ax.plot(dfn_extension, deriv)
    # ax.set_ylim(-100, 100) # maxRFList
    # plt.show()

    # In[Time plot]

    # slope calculation for n seconds prior to the max peel force
    # (Experimental Data)
    maxIndex = dfn_time[dfn_time == HTMax].index.values[0]

    # to location of max force
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
    t_n = dfn_time[maxIndex - timeBeforePeak:maxIndex]
    y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
    # Perform least squares and return
    curveFit_n, Params_n, R_Squared_n = Least_Squares(t_n, y)

    # Shift extension data so that the linear region is extrapolated
    # through the origin
    shift_time = abs(Params_n[0]/Params_n[1])*1
    if Params_n[0] > 0:
        # shift time data for visual purposes
        dfn_time_shift = dfn_time + shift_time
        dfn_ss_time_shift = ssTimeArray + shift_time
        HTMax_shift = HTMax + shift_time
    else:
        # shift time data for visual purposes
        dfn_time_shift = dfn_time - shift_time
        dfn_ss_time_shift = ssTimeArray - shift_time
        HTMax_shift = HTMax - shift_time


```

```python
    # Curve fit the shifted displacement
    maxIndex = dfn_time[dfn_time == HTMax].index.values[0]

    # to location of max force
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
    t_n = dfn_time_shift[maxIndex - timeBeforePeak:maxIndex]
    y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
    # Perform least squares and return
    curveFit_n, Params_n, R_Squared_n = Least_Squares(t_n, y)

    # to location of max force
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
    x_n = dfn_extension[maxIndex - timeBeforePeak:maxIndex]
    # Perform least squares
    curveFit_n_disp, Params_n_disp, R_Squared_n_disp = Least_Squares(x_n, y)

    # Shift extension data so that the linear region is extrapolated through
    # the origin
    shift_disp = abs(Params_n_disp[0]/Params_n_disp[1])*1
    if Params_n[0] > 0:
        dfn_extension_shift = dfn_extension + shift_disp
        dfn_ss_Disp_shift = dfn_ss_Disp + shift_disp
    else:
        dfn_extension_shift = dfn_extension - shift_disp
        dfn_ss_Disp_shift = dfn_ss_Disp - shift_disp

    # to location of max force
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec)
    x_n = dfn_extension_shift[maxIndex - timeBeforePeak:maxIndex]
    # Perform least squares
    curveFit_n_disp, Params_n_disp, R_Squared_n_disp = Least_Squares(x_n, y)


    Fmax_t_shift = dfn_time_shift[maxIndex]
    fit_t = np.linspace(0, Fmax_t_shift, 200) # Selected value

    # true max
    # fit_t = np.linspace(0, dfn_time_shift[np.argmax(dfn_force)], 200)
    Fmax_x_shift = dfn_extension_shift[maxIndex]

    # true max
    # fit_x = np.linspace(0, dfn_extension_shift[np.argmax(dfn_force)], 200)
    fit_x = np.linspace(0, Fmax_x_shift, 200) # Selected value

    def fit(params, x):
        b, m = params
        return m*x + b

    fit_vals_y_time = fit(Params_n, fit_t)
    fit_vals_y_force = fit(Params_n_disp, fit_x)

    ''' Reaction force vs. time shifted '''
    fig, ax = plt.subplots()
    ax.plot(dfn_time_shift, dfn_force,
            label=r'Data - {}'.format(dataCompare.split('.')[0]))
    ax.plot(fit_t, fit_vals_y_time, '--', label=r'Assumed linear region')
    ax.plot(Fmax_t_shift, dfn_force[maxIndex], 'o', markersize=10,
            label=r'Time at peak = {:.4} (s)'.format(max(fit_t)))
```

```python

    ax.plot(dfn_ss_time_shift, ssValArray,'-', color='c', linewidth=3,
            markersize=2, label = 'Steady State - {:.4f} (mN)'.format(HFSS),
            path_effects=[pe.Stroke(linewidth=5, foreground='k'), pe.Normal()])

    ax.plot([], [], 'w',
            label='Start SS time = {:.4f} (s)'.format(min(dfn_ss_time_shift)))
    ax.plot([], [], 'w',
            label='End SS time = {:.4f} (s)'.format(max(dfn_ss_time_shift)))

    plt.axhline(0,color='black')
    plt.axvline(0,color='black')

    plt.ylabel('Force (mN)',fontsize=18)
    plt.xlabel('Time from extrapolated zero (s)',fontsize=18)
    plt.legend(loc='best')
    # plt.xlim([0, max(dfn_time_shift)])
    plt.savefig(os.path.join(abqWD, 'SimulationTime.pdf'), dpi=300,
                bbox_inches='tight')
    # plt.show()
    plt.close()

    ''' Reaction force vs. displacement shifted '''
    fig, ax = plt.subplots()
    ax.plot(dfn_extension_shift, dfn_force,
            label=r'Data - {}'.format(dataCompare.split('.')[0]))
    ax.plot(fit_x, fit_vals_y_force, '--', label=r'Assumed linear region')
    ax.plot(Fmax_x_shift, dfn_force[maxIndex], 'o', markersize=10,
            label=r'Time at peak = {:.4} (s)'.format(max(fit_t)))

    ax.plot(dfn_ss_Disp_shift, ssValArray,'-', color='c', linewidth=3,
            markersize=2, label = 'Steady State - {:.4f} (mN)'.format(HFSS),
            path_effects=[pe.Stroke(linewidth=5, foreground='k'), pe.Normal()])

    ax.plot([], [], 'w',
            label='Start SS time = {:.4f} (s)'.format(min(dfn_ss_time_shift)))
    ax.plot([], [], 'w',
            label='End SS time = {:.4f} (s)'.format(max(dfn_ss_time_shift)))

    plt.axhline(0, color='black')
    plt.axvline(0, color='black')

    plt.ylabel('Force (mN)',fontsize=18)
    plt.xlabel('Displacement (mm)',fontsize=18)
    plt.legend(loc='best')
    # plt.xlim([0, max(dfn_time_shift)])
    plt.savefig(os.path.join(abqWD, 'SimulationDisp.pdf'), dpi=300,
                bbox_inches='tight')
    # plt.show()
    plt.close()

    # In[Interpolated Experimental Data]

    # create array from 0 max peel force (linear equation fit from above)
    # populate a pandas dataframe
    # merge the data frame with the data above from the peak force to the end
    # use the interp1d fcn to interpolate between data
    # pass the simulated data into the interpolation
```

```python

    # Time greater than the shift intersection point
    t_exp = dfn_time_shift[dfn_time_shift >= 0]
    x_exp = dfn_extension_shift[dfn_time_shift >= 0]
    y_exp = dfn_force[dfn_time_shift >= 0]

    # data frame with original data
    dfdata = pd.DataFrame(np.array([t_exp, x_exp, y_exp]).T,
                          columns=['t', 'x', 'y'])

    # Select time beyond the max time to the end of the data
    t_geq_max = dfn_time_shift[maxIndex:]
    x_geq_max = dfn_extension_shift[maxIndex:]
    y_geq_max = dfn_force[maxIndex:]

    # dataframe of data points from the max value to the end
    dfgmax = pd.DataFrame(np.array([t_geq_max, x_geq_max, y_geq_max]).T,
                          columns=['t', 'x', 'y'])

    # data frame of points from zero to the max value
    linArray = np.array([fit_t, fit_x, fit_vals_y_time])
    dfLin = pd.DataFrame(linArray.T, columns=['t', 'x', 'y'])

    # create the new data frame of linear points up to the peak and all points
    # beyond
    dfNew = dfLin.append(dfgmax, ignore_index=True)

    # # Interpolate the experimental data
    # n_data_pts = 100
    # Time at the peak (shifted)
    # start_point_time = tt[RF.argmax()]# - shift
    # Disp at the peak (shifted)
    # start_point_disp = dn[RF.argmax()]# - shift_disp
    # f_exp_time = interpolate.interp1d(dfNew['t'], dfNew['y'])
    # f_exp_disp = interpolate.interp1d(dfNew['x'], dfNew['y'])
    # t_new_exp = np.linspace(start_point_time, tt[tt.argmax()],
    #                         n_data_pts) # (s)
    # x_new_exp = np.linspace(start_point_disp, dn[tt.argmax()],
    #                         n_data_pts) # (mm)
    # y_new_exp_time = f_exp_time(t_new_exp) # Interpolate `interp1d`
    # y_new_exp_disp = f_exp_disp(x_new_exp) # Interpolate `interp1d`

    # In[Interpolated Simulated Trace]

    # # Interpolate the simulated data
    # f_sim_time = interpolate.interp1d(tt, RF)
    # f_sim_disp = interpolate.interp1d(dn, RF)
    # t_new_sim = np.linspace(start_point_time, tt[tt.argmax()],
    #                         n_data_pts) # (s)
    # x_new_sim = np.linspace(start_point_disp, dn[tt.argmax()],
    #                         n_data_pts) # (mm)
    # y_new_sim_time = f_sim_time(t_new_sim) # Interpolate `interp1d`
    # y_new_sim_disp = f_sim_disp(x_new_sim) # Interpolate `interp1d`

    # In[Plots]
    # ''' Time curve '''
    # fit, ax = plt.subplots()
    # ax.plot()
```

```
    # ax.plot(dfdata['t'], dfdata['y'], label='Original Shifted Data',
    #         alpha = 0.5)
    # ax.plot(dfNew['t'], dfNew['y'], label='Merged Data',
    #         alpha = 0.5)
    # ax.plot(t_new_exp, y_new_exp_time, '--',
    #         label='Interp Experimental Data')
    # ax.plot(tt, RF, label='Simulated Data')
    # ax.plot(t_new_sim, y_new_sim_time, ':', label='Interp Simulated Data')
    # ax.set_xlim([0, 300])
    # ax.set_xlabel('Time (s)', fontsize=14)
    # ax.set_ylabel('Force (N)', fontsize=14)
    # ax.legend(loc='best', fontsize=14)
    # ax.grid('on')
    # plt.savefig(os.path.join(abqWD, 'interp1d_Time.pdf'), dpi=300,
    #             bbox_inches='tight')
    # plt.show()

    # ''' Displacement curve '''
    # fit, ax = plt.subplots()
    # ax.plot()
    # ax.plot(dfdata['x'], dfdata['y'], label='Original Shifted Data',
    #         alpha = 0.5)
    # ax.plot(dfNew['x'], dfNew['y'], label='Merged Data',
    #         alpha = 0.5)
    # ax.plot(x_new_exp, y_new_exp_disp, '--',
    #         label='Interp Experimental Data')
    # ax.plot(dn, RF, label='Simulated Data')
    # ax.plot(x_new_sim, y_new_sim_disp, ':', label='Interp Simulated Data')
    # ax.set_xlim([0, max(dn)])
    # ax.set_xlabel('Displacement (mm)', fontsize=14)
    # ax.set_ylabel('Force (N)', fontsize=14)
    # ax.legend(loc='best', fontsize=14)
    # ax.grid('on')
    # plt.savefig(os.path.join(abqWD, 'interp1d_Disp.pdf'), dpi=300,
    #             bbox_inches='tight')
    # plt.show()

    # ''' Displacement curve only showing interpolated data '''
    # abs residual calculation
    # residual = abs(y_new_exp_disp - y_new_sim_disp)
    # L2Norm = np.dot(residual, residual)

    # fit, ax = plt.subplots()
    # ax.plot()
    # ax.plot(x_new_exp, y_new_exp_disp, '-', label='Interp Experimental Data')
    # ax.plot(x_new_sim, y_new_sim_disp, '-', label='Interp Simulated Data')
    # ax.plot(x_new_sim, residual, ':',
    #         label=r'Residual = $\|\| exp - sim \|\|$', alpha = 0.8)
    # ax.plot([], [], color='white',
    #         label=r'$L^2$ norm = {:.4f}'.format(L2Norm))
    # ax.axhline(color='k', linewidth=0.25)
    # ax.set_xlim([0, max(x_new_exp)])
    # ax.set_xlabel('Displacement (mm)', fontsize=14)
    # ax.set_ylabel('Force (N)', fontsize=14)
    # ax.legend(loc='best', fontsize=14)
    # ax.grid('on')
    # plt.savefig(os.path.join(abqWD, 'interp1d_Disp_clean.pdf'), dpi=300,
    #             bbox_inches='tight')
```

```python
    # plt.show()

    print('Output files have been printed to determine the appropriate ' +
          'parameters for the simulation')

    returnArray = [max(fit_t), max(dfn_ss_time_shift), HTMax_shift,
                   Integral*1e-6]
    return returnArray

if __name__ == '__main__':
    # Run the function

    # fileName = sys.argv[-2]
    # savePath = sys.argv[-1]

    ReadRAWDataTrace(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.*

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Jan 28 21:51:32 2021

@author: Kiffer Creveling
"""

"""  abaqus cae -noGUI abaqusMacros.py """

# -*- coding: mbcs -*-
# Do not delete the following import lines
from abaqus import *
from abaqusConstants import *
import __main__

import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import optimization
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
import numpy as np
```

```python
import os
import sys

# In[Non-symmetric model]
'''
Non-symmetric model
'''

# location of the folder
# specific folder path where this file is located # os.getcwd()
pythonScriptPath = os.path.abspath("file")
abqWD, pythonFiles = os.path.split(pythonScriptPath) # split file path

# StepFile = 'Adult Human Eye holder Assembly.STEP'
# StepFile = 'Adult Human Eye holder Assembly 2 Step.STEP'
# Constrained
# StepFile = 'Adult Human Eye holder Assembly Constrained Bottom.STEP'

# Trimmed to prevent element distortion on low elastic modulus curve fits
StepFile = ('Adult Human Eye holder Assembly Constrained Bottom Trimmed ' +
            'Retina.STEP')

SolidWorksDir = 'SolidWorksStepFiles' # Folder name

# Combine folder directory
SolidWorksStepFile = os.path.join(SolidWorksDir, StepFile)

def ImportStepEyeConstrained():
    """ Use with the constrained bottom STEP file"""
    step = mdb.openStep(os.path.join(abqWD, SolidWorksStepFile),
                        scaleFromFile=OFF)

    abqModel.PartFromGeometryFile(name='V', geometryFile=step, bodyNum=1,
                                  combine=False, dimensionality=THREE_D,
                                  type=DEFORMABLE_BODY)
    abqModel.PartFromGeometryFile(name='E', geometryFile=step, bodyNum=2,
                                  combine=False, dimensionality=THREE_D,
                                  type=DISCRETE_RIGID_SURFACE)
    abqModel.PartFromGeometryFile(name='R', geometryFile=step, bodyNum=3,
                                  combine=False, dimensionality=THREE_D,
                                  type=DEFORMABLE_BODY)
    abqModel.PartFromGeometryFile(name='T', geometryFile=step, bodyNum=4,
                                  combine=False, dimensionality=THREE_D,
                                  type=DISCRETE_RIGID_SURFACE)
    abqModel.PartFromGeometryFile(name='G', geometryFile=step, bodyNum=5,
                                  combine=False, dimensionality=THREE_D,
                                  type=DISCRETE_RIGID_SURFACE)


def Retina_Mat_Prop(RetinaProp):
    retina_E = RetinaProp # Passed in young's modulus
    Retina_Description = """
Actually used the value from Chen 2014
E = 11.12 KPa

------------------------------------------------
Density (kg/m^3)
1100 --------> Esposito_2013
```

```

"""
    abqModel.Material(name='Retina', description=Retina_Description)
    abqModel.materials['Retina'].Density(table=((1100.0, ), ))
    abqModel.materials['Retina'].Elastic(table=((retina_E, 0.49), ))

    # Assign the section to the part
    abqModel.HomogeneousSolidSection(name='Retina_Section', material='Retina',
                                     thickness=None)

def Vitreous_Mat_Prop(vitreousProp):
    vitreous_E = vitreousProp # Passed in young's modulus
    Vitreous_Description = """
------------------------------------------------
Density (kg/m^3)
950   ------------> Esposito_2013

------------------------------------------------

# Tram 2018 Viscoelasticity data
# 4 Term Prony (Tram Data # 5 HU2018-0074 OD 1 Pa)
(0.1486397420159951, 0.0, 331.4796231072498),
(0.12469207412616717, 0.0, 3.388868494747128),
(0.29059507092540404, 0.0, 15.59692349525066),
(0.1591569334281, 0.0, 69.85134248442381)
"""
    abqModel.Material(name='Vitreous', description=Vitreous_Description)
    abqModel.materials['Vitreous'].Density(table=((950.0, ), ))
    ''' Using Lin2020 Paper to relate SLSM curve fit parameters to physical
    values.  Prony 4 Term (Long term) initial guess 172.77874855377468
    optimization of E'''
    abqModel.materials['Vitreous'].Elastic(moduli=LONG_TERM,
                                           table=((vitreous_E, 0.49), ))
    # Prony 4 Term calculated from normalized data
    abqModel.materials['Vitreous'].Viscoelastic(
            domain=TIME, time=PRONY, table=(
            # Tram Data # 5
            (0.1486397420159951, 0.0, 331.4796231072498),
            (0.12469207412616717, 0.0, 3.388868494747128),
            (0.29059507092540404, 0.0, 15.59692349525066),
            (0.1591569334281, 0.0, 69.85134248442381)))

    # Assign the section to the part
    abqModel.HomogeneousSolidSection(name='Vitreous_Section',
                                     material='Vitreous', thickness=None)


def E_Features():
    ''' Eye holder features '''
    p = abqModel.parts['E']

    # Remove shell
    c = p.cells
    p.RemoveCells(cellList = c[0:1])

    # Reference point
    p.ReferencePoint(point=(0.0, 0.0, 0.0))

```

```python
    # Add E-set to the reference point
    r = p.referencePoints
    refPoints=(r[3], )
    p.Set(referencePoints=refPoints, name='E_RP_Set')

    # Edge seed sets
    e = p.edges
    edges = e.getSequenceFromMask(mask=('[#400f000 #1402 ]', ), )
    p.Set(edges=edges, name='E_Edge_Seed_Set')

    edges = e.getSequenceFromMask(mask=('[#f1ff0fff #2838 ]', ), )
    p.Set(edges=edges, name='E_Outside_Edge_Seed_Set')

    # Surfaces
    s = p.faces
    side1Faces = s.getSequenceFromMask(mask=('[#1ffff ]', ), )
    p.Surface(side1Faces=side1Faces, name='E_Surf')


def G_Features():
    ''' Glue features '''
    p = abqModel.parts['G']
    c = p.cells

    # Remeove cells for rigid body
    p.RemoveCells(cellList = c[0:1])

    # Reference point
    p.ReferencePoint(point=(9.799E-03, 5.657E-03, 2.54E-03))

    # Define the reference point for the rigid body
    r = p.referencePoints
    refPoints=(r[3], )
    p.Set(referencePoints=refPoints, name='G_RP_Set')

    # # Create sets
    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#3f ]', ), )
    p.Set(faces=faces, name='G_Set')
    faces = f.getSequenceFromMask(mask=('[#20 ]', ), )
    p.Set(faces=faces, name='G_T_Set')
    faces = f.getSequenceFromMask(mask=('[#1 ]', ), )
    p.Set(faces=faces, name='G_R_Set')

    # Create surfaces
    s = p.faces
    side1Faces = s.getSequenceFromMask(mask=('[#3f ]', ), )
    p.Surface(side1Faces=side1Faces, name='G_Surf')
    side1Faces = s.getSequenceFromMask(mask=('[#20 ]', ), )
    p.Surface(side1Faces=side1Faces, name='G_T_Surf')
    side1Faces = s.getSequenceFromMask(mask=('[#1 ]', ), )
    p.Surface(side1Faces=side1Faces, name='G_R_Surf')


def T_Features():
    ''' Plastic Tab features '''
    p = abqModel.parts['T']
    c = p.cells
```

```python

    # Remeove cells for rigid body
    p.RemoveCells(cellList = c[0:1])

    # Reference point
    p.ReferencePoint(point=(16.241E-03, 9.74E-03, 13.E-06))

    # Define the reference point for the rigid body
    r = p.referencePoints
    refPoints=(r[3], )
    p.Set(referencePoints=refPoints, name='T_RP_Set')

    # Create sets
    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#ff ]', ), )
    p.Set(faces=faces, name='T_Set')
    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#2 ]', ), )
    p.Set(faces=faces, name='T_G_Set')

    # Create surfaces
    s = p.faces
    side1Faces = s.getSequenceFromMask(mask=('[#ff ]', ), )
    p.Surface(side1Faces=side1Faces, name='T_Surf')
    side1Faces = s.getSequenceFromMask(mask=('[#2 ]', ), )
    p.Surface(side1Faces=side1Faces, name='T_G_Surf')


def R_Features():
    ''' Retina features '''
    p = abqModel.parts['R']
    c = p.cells
    cells = c.getSequenceFromMask(mask=('[#1 ]', ), )
    p.Set(cells=cells, name='R_Set')

    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#3 ]', ), )
    p.Set(faces=faces, name='R_G_Set')

    faces = f.getSequenceFromMask(mask=('[#4 ]', ), )
    p.Set(faces=faces, name='R_V_Set')

    s = p.faces
    side1Faces = s.getSequenceFromMask(mask=('[#ff ]', ), )
    p.Surface(side1Faces=side1Faces, name='R_Surf')

    side1Faces = s.getSequenceFromMask(mask=('[#3 ]', ), )
    p.Surface(side1Faces=side1Faces, name='R_G_Surf')

    side1Faces = s.getSequenceFromMask(mask=('[#4 ]', ), )
    p.Surface(side1Faces=side1Faces, name='R_V_Surf_BOND')

    # Assign section
    region = p.sets['R_Set']
    p.SectionAssignment(region=region, sectionName='Retina_Section',
                        offset=0.0, offsetType=MIDDLE_SURFACE, offsetField='',
                        thicknessAssignment=FROM_SECTION)

```

```python


def PartitionRetinaOnVitreous():
    ''' Vitreous features additional partitions for creating the surface for
    bonding'''
    p = abqModel.parts['V']

    # Partition V along the width of the retina
    p.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=-0.00254)
    abqModel.parts['V'].features.changeKey(fromName='Datum plane-1',
                                           toName='Retina_Width_Neg_Z')

    p.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=0.00254)
    abqModel.parts['V'].features.changeKey(fromName='Datum plane-1',
                                           toName='Retina_Width_Pos_Z')

    # Create a datum plnd along the z axis plane
    p.DatumAxisByPrincipalAxis(principalAxis=ZAXIS)
    p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset=0.0)

    # Create the rotated datum planes
    d = p.datums
    p.DatumPlaneByRotation(plane=d[5], axis=d[4], angle=18.75)
    p.DatumPlaneByRotation(plane=d[5], axis=d[4], angle=-18.75)

    ''' Partition the surface of the retina on the vitreous '''
    p = abqModel.parts['V']
    c, d = p.cells, p.datums
    pickedCells = c.getSequenceFromMask(mask=('[#440 ]', ), )
    p.PartitionCellByDatumPlane(datumPlane=d[3], cells=pickedCells)
    pickedCells = c.getSequenceFromMask(mask=('[#408 ]', ), )
    p.PartitionCellByDatumPlane(datumPlane=d[2], cells=pickedCells)
    pickedCells = c.getSequenceFromMask(mask=('[#22 ]', ), )
    p.PartitionCellByDatumPlane(datumPlane=d[6], cells=pickedCells)
    pickedCells = c.getSequenceFromMask(mask=('[#140 ]', ), )
    p.PartitionCellByDatumPlane(datumPlane=d[7], cells=pickedCells)


def Vitreous_Features():
    ''' Assign specific features to the vitreous '''
    p = abqModel.parts['V']
    c, f, s = p.cells, p.faces, p.faces

    # Sets
    cells = c.getSequenceFromMask(mask=('[#ffffff ]', ), )
    p.Set(cells=cells, name='V_Set')
    faces = f.getSequenceFromMask(mask=('[#5090 ]', ), )
    p.Set(faces=faces, name='V_R_Set')

    # Surfaces
    side1Faces = s.getSequenceFromMask(mask=('[#1805090 #3 #ff0 ]', ), )
    p.Surface(side1Faces=side1Faces, name='V_Surf')
    side1Faces = s.getSequenceFromMask(mask=('[#5090 ]', ), )
    p.Surface(side1Faces=side1Faces, name='V_R_Surf_BOND')

    # Assign the section to the part
    region = p.sets['V_Set']
    p.SectionAssignment(region=region,
```

```
                        sectionName='Vitreous_Section',
                        offset=0.0,
                        offsetType=MIDDLE_SURFACE,
                        offsetField='',
                        thicknessAssignment=FROM_SECTION)


def V_Partition_XYZ_Axis():
    ''' Partition the sphere along the x, y, z axis '''
    p = abqModel.parts['V']
    c, v, e, d = p.cells, p.vertices, p.edges, p.datums
    pickedCells = c.getSequenceFromMask(mask=('[#1 ]', ), )
    p.PartitionCellByPlaneThreePoints(point1=v[1],
                                      point2=v[0],
                                      point3=v[3],
                                      cells=pickedCells)

    pickedCells = c.getSequenceFromMask(mask=('[#3 ]', ), )
    p.PartitionCellByPlaneThreePoints(point1=v[0],
                                      point2=v[4],
                                      point3=v[2],
                                      cells=pickedCells)

    pickedCells = c.getSequenceFromMask(mask=('[#f ]', ), )
    p.PartitionCellByPlaneThreePoints(point1=v[5],
                                      point2=v[2],
                                      point3=v[4],
                                      cells=pickedCells)


def V_Internal_Sphere():
    sphereRadius = 0.008 # radius of the internal sphere for meshing

    s1 = abqModel.ConstrainedSketch(name='__profile__', sheetSize=0.1)
    g, v, d, c1 = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
    s1.sketchOptions.setValues(decimalPlaces=3)
    s1.setPrimaryObject(option=STANDALONE)
    s1.ConstructionLine(point1=(0.0, -0.05), point2=(0.0, 0.05))
    s1.FixedConstraint(entity=g[2])
    s1.ArcByCenterEnds(center=(0.0, 0.0),
                        point1=(0.0, sphereRadius),
                        point2=(0.0, -sphereRadius),
                        direction=CLOCKWISE)
    s1.Line(point1=(0.0, sphereRadius),
            point2=(0.0, -sphereRadius))
    s1.VerticalConstraint(entity=g[4], addUndoState=False)
    s1.PerpendicularConstraint(entity1=g[3], entity2=g[4], addUndoState=False)
    p = abqModel.Part(name='V_internal',
                      dimensionality=THREE_D,
                      type=DEFORMABLE_BODY)
    p = abqModel.parts['V_internal']
    p.BaseSolidRevolve(sketch=s1, angle=360.0, flipRevolveDirection=OFF)
    s1.unsetPrimaryObject()
    p = abqModel.parts['V_internal']
    del abqModel.sketches['__profile__']


def mergeV():
```

```
    ''' Merge the internal sphere with the vitreous '''
    a = abqModel.rootAssembly
    a.InstanceFromBooleanMerge(name='V_Merge',
                               instances=(a.instances['V-1'],
                                          a.instances['V_internal-1'], ),
                               keepIntersections=ON,
                               originalInstances=DELETE,
                               domain=GEOMETRY)

    # Clean up file names after merge
    del abqModel.parts['V']
    del abqModel.parts['V_internal']

    abqModel.parts.changeKey(fromName='V_Merge', toName='V')
    a = abqModel.rootAssembly
    a.regenerate()
    abqModel.rootAssembly.features.changeKey(fromName='V_Merge-1',
                                              toName='V-1')

    a.regenerate()


def AssembleV_for_Merging():
    a1 = abqModel.rootAssembly
    a1.DatumCsysByDefault(CARTESIAN)
    p = abqModel.parts['V']
    a1.Instance(name='V-1', part=p, dependent=ON)
    p = abqModel.parts['V_internal']
    a1.Instance(name='V_internal-1', part=p, dependent=ON)


def E_Mesh(InsideSeed, OutsideSeed):
    p = abqModel.parts['E']
    e = p.edges
    pickedEdges = e.getSequenceFromMask(mask=('[#400f000 #1402 ]', ), )
    p.seedEdgeBySize(edges=pickedEdges,
                     size=0.0005,
                     deviationFactor=0.1,
                     minSizeFactor=0.1,
                     constraint=FINER)
    pickedEdges = e.getSequenceFromMask(mask=('[#f1ff0fff #2838 ]', ), )
    p.seedEdgeBySize(edges=pickedEdges,
                     size=0.00342673,
                     deviationFactor=0.1,
                     minSizeFactor=0.1,
                     constraint=FINER)
    # (unique node numbering)
    p.setValues(startNodeLabel=1000000, startElemLabel=1000000)
    p.generateMesh()


def G_Mesh(seed):
    p = abqModel.parts['G']
    p.seedPart(size=seed, deviationFactor=0.1, minSizeFactor=0.1)
    f = p.faces
    pickedRegions = f.getSequenceFromMask(mask=('[#3f ]', ), )
    p.setMeshControls(regions=pickedRegions, elemShape=QUAD)
    elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=EXPLICIT)
```

```python
    elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=EXPLICIT)
    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#3f ]', ), )
    pickedRegions =(faces, )
    p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
    # (unique node numbering)
    p.setValues(startNodeLabel=2000000, startElemLabel=2000000)
    p.generateMesh()


def T_Mesh(seed):
    p = abqModel.parts['T']
    p.seedPart(size=seed, deviationFactor=0.1, minSizeFactor=0.1)
    f = p.faces
    pickedRegions = f.getSequenceFromMask(mask=('[#ff ]', ), )
    p.setMeshControls(regions=pickedRegions, elemShape=QUAD)
    elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=EXPLICIT)
    elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=EXPLICIT)
    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#ff ]', ), )
    pickedRegions =(faces, )
    p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
    # (unique node numbering)
    p.setValues(startNodeLabel=3000000, startElemLabel=3000000)
    p.generateMesh()


def R_Mesh(seed):
    p = abqModel.parts['R']
    p.seedPart(size=seed, deviationFactor=0.1, minSizeFactor=0.1)
    c, e = p.cells, p.edges
    pickedRegions = c.getSequenceFromMask(mask=('[#1 ]', ), )
    p.setMeshControls(regions=pickedRegions,
                      technique=SWEEP,
                      algorithm=ADVANCING_FRONT)
    p.setSweepPath(region=c[0], edge=e[10], sense=FORWARD)
    elemType1 = mesh.ElemType(elemCode=C3D8R,
                              elemLibrary=EXPLICIT,
                              kinematicSplit=AVERAGE_STRAIN,
                              secondOrderAccuracy=ON,
                              hourglassControl=ENHANCED,
                              distortionControl=ON,
                              lengthRatio=0.100000001490116)
    elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
    elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT)
    c = p.cells
    cells = c.getSequenceFromMask(mask=('[#1 ]', ), )
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions,
                     elemTypes=(elemType1, elemType2, elemType3))
    p.generateMesh()
    # (unique node numbering)
    p.setValues(startNodeLabel=4000000, startElemLabel=4000000)
    p.generateMesh()


def VitreousMesh(v1Seed, v2Seed):
    ''' Specity tetrahedral elements '''
```

```
    p = abqModel.parts['V']
    c = p.cells
    pickedRegions = c.getSequenceFromMask(mask=('[#86f800 ]', ), )
    p.setMeshControls(regions=pickedRegions, elemShape=TET, technique=FREE)
    elemType1 = mesh.ElemType(elemCode=C3D20R)
    elemType2 = mesh.ElemType(elemCode=C3D15)
    elemType3 = mesh.ElemType(elemCode=C3D10)
    cells = c.getSequenceFromMask(mask=('[#86f800 ]', ), )
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions,
                     elemTypes=(elemType1, elemType2, elemType3))

    ''' Specify hexahedral elements '''
    elemType1 = mesh.ElemType(elemCode=C3D8R, elemLibrary=EXPLICIT)
    elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
    elemType3 = mesh.ElemType(elemCode=C3D4,
                              elemLibrary=EXPLICIT,
                              secondOrderAccuracy=ON,
                              distortionControl=ON,
                              lengthRatio=0.100000001490116)
    cells = c.getSequenceFromMask(mask=('[#86f800 ]', ), )
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions,
                     elemTypes=(elemType1, elemType2, elemType3))

    elemType1 = mesh.ElemType(elemCode=C3D8R,
                              elemLibrary=EXPLICIT,
                              kinematicSplit=AVERAGE_STRAIN,
                              secondOrderAccuracy=ON,
                              hourglassControl=ENHANCED,
                              distortionControl=ON,
                              lengthRatio=0.100000001490116)
    elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
    elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT)
    cells = c.getSequenceFromMask(mask=('[#7907ff ]', ), )
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions,
                     elemTypes=(elemType1, elemType2, elemType3))

    # Seed the entire part
    p.seedPart(size=v2Seed, deviationFactor=0.1, minSizeFactor=0.1)

    # Seed the retina interface
    e = p.edges
    pickedEdges = e.getSequenceFromMask(mask=('[#ffffffff #7fec0fff #80012 ]',
                                              ), )
    p.seedEdgeBySize(edges=pickedEdges,
                     size=v1Seed,
                     deviationFactor=0.1,
                     minSizeFactor=0.1,
                     constraint=FINER)

    # Seed the bias edges
    e = p.edges
    pickedEdges1 = e.getSequenceFromMask(mask=('[#0 #104000 #10001 ]', ), )
    pickedEdges2 = e.getSequenceFromMask(mask=('[#0 #80020000 #900000 ]', ), )
    p.seedEdgeByBias(biasMethod=SINGLE,
                     end1Edges=pickedEdges1,
```

```
                     end2Edges=pickedEdges2,
                     minSize=v1Seed,
                     maxSize=v2Seed,
                     constraint=FINER)

    # (unique node numbering)
    p.setValues(startNodeLabel=5000000, startElemLabel=5000000)
    p.generateMesh()


def QuadraticTetVitreous():
    # Vitreous
    p = abqModel.parts['V']
    c = p.cells
    pickedRegions = c.getSequenceFromMask(mask=('[#9f ]', ), )
    p.deleteMesh(regions=pickedRegions)
    p.setMeshControls(regions=pickedRegions, elemShape=TET, technique=FREE)
    elemType1 = mesh.ElemType(elemCode=UNKNOWN_HEX, elemLibrary=EXPLICIT)
    elemType2 = mesh.ElemType(elemCode=UNKNOWN_WEDGE, elemLibrary=EXPLICIT)
    elemType3 = mesh.ElemType(elemCode=C3D10M, elemLibrary=EXPLICIT)
    cells = c.getSequenceFromMask(mask=('[#9f ]', ), )
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions,
                     elemTypes=(elemType1, elemType2, elemType3))
    elemType1 = mesh.ElemType(elemCode=UNKNOWN_HEX, elemLibrary=EXPLICIT)
    elemType2 = mesh.ElemType(elemCode=UNKNOWN_WEDGE, elemLibrary=EXPLICIT)
    elemType3 = mesh.ElemType(elemCode=C3D10M,
                              elemLibrary=EXPLICIT,
                              secondOrderAccuracy=ON,
                              distortionControl=ON,
                              lengthRatio=0.100000001490116)
    c = p.cells
    p.setElementType(regions=pickedRegions,
                     elemTypes=(elemType1, elemType2, elemType3))
    p.generateMesh()


def QuadraticTetRetina():
    # Retina
    p = abqModel.parts['R']
    c = p.cells
    pickedRegions = c.getSequenceFromMask(mask=('[#1 ]', ), )
    p.deleteMesh(regions=pickedRegions)
    p.setMeshControls(regions=pickedRegions, elemShape=TET, technique=FREE)
    elemType1 = mesh.ElemType(elemCode=UNKNOWN_HEX, elemLibrary=EXPLICIT)
    elemType2 = mesh.ElemType(elemCode=UNKNOWN_WEDGE, elemLibrary=EXPLICIT)
    elemType3 = mesh.ElemType(elemCode=C3D10M, elemLibrary=EXPLICIT)
    c = p.cells
    cells = c.getSequenceFromMask(mask=('[#1 ]', ), )
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions,
                      elemTypes=(elemType1, elemType2, elemType3))
    elemType1 = mesh.ElemType(elemCode=UNKNOWN_HEX, elemLibrary=EXPLICIT)
    elemType2 = mesh.ElemType(elemCode=UNKNOWN_WEDGE, elemLibrary=EXPLICIT)
    elemType3 = mesh.ElemType(elemCode=C3D10M,
                              elemLibrary=EXPLICIT,
                              secondOrderAccuracy=ON,
                              distortionControl=ON,
```

```
                              lengthRatio=0.100000001490116)
    p.setElementType(regions=pickedRegions,
                     elemTypes=(elemType1, elemType2, elemType3))
    p.generateMesh()


def Assembly():
    a1 = abqModel.rootAssembly
    a1.DatumCsysByDefault(CARTESIAN)
    p = abqModel.parts['E']
    a1.Instance(name='E-1', part=p, dependent=ON)
    p = abqModel.parts['G']
    a1.Instance(name='G-1', part=p, dependent=ON)
    p = abqModel.parts['R']
    a1.Instance(name='R-1', part=p, dependent=ON)
    p = abqModel.parts['T']
    a1.Instance(name='T-1', part=p, dependent=ON)
    p = abqModel.parts['V']
    a1.Instance(name='V-1', part=p, dependent=ON)


def GravityStep(time, prevStep, scaleFactor, MSTI, stepName, descrip):
    abqModel.ExplicitDynamicsStep(name=stepName,
                                  previous=prevStep,
                                  description=descrip,
                                  timePeriod=time,
                                  massScaling=((SEMI_AUTOMATIC,
                                                MODEL,
                                                AT_BEGINNING,
                                                scaleFactor,
                                                MSTI,
                                                BELOW_MIN, 0, 0, 0.0, 0.0, 0,
                                                None), ))


def Step(time, prevStep, scaleFactor, MSTI, stepName, descrip):
    abqModel.ExplicitDynamicsStep(name=stepName,
                                  previous=prevStep,
                                  description=descrip,
                                  timePeriod=time,
                                  massScaling=((SEMI_AUTOMATIC,
                                                MODEL,
                                                AT_BEGINNING,
                                                scaleFactor,
                                                MSTI,
                                                BELOW_MIN, 0, 0, 0.0, 0.0,
                                                0, None), ),
                                  nlgeom=ON)
    # Mass Scale default
    if MSTI == 0:
        print('This will take a while...ABAQUS is deciding for us')
        # Use zero value
        abqModel.steps[stepName].setValues(massScaling=PREVIOUS_STEP)
    else:
        print('Mass Scale Time Increment has been defined')


def F_output(stepName):
```

```python
    FOutputInterval = 50 # Double the data points (Default is 20)

    if damageInitiation == False and damageEvolution == False:
        # Whole Model Fieldoutput (RF, U, NFORC)
        abqModel.FieldOutputRequest(name='F-Output-1',
                                    createStepName=stepName,
                                    variables=('RF',
                                               'U',
                                               'NFORC'),
                                    numIntervals=FOutputInterval)

    elif damageInitiation == True and damageEvolution == False:
        # Whole Model Fieldoutput (RF, U, NFORC)
        abqModel.FieldOutputRequest(name='F-Output-1',
                                    createStepName=stepName,
                                    variables=('RF',
                                               'U',
                                               'NFORC',
                                               'CSMAXSCRT'),
                                    numIntervals=FOutputInterval)

    elif damageInitiation == True and damageEvolution == True:
        # Whole Model Fieldoutput (RF, U, NFORC)
        abqModel.FieldOutputRequest(name='F-Output-1',
                                    createStepName=stepName,
                                    variables=('RF',
                                               'U',
                                               'NFORC',
                                               'CSDMG',
                                               'CSMAXSCRT'),
                                    numIntervals=FOutputInterval)

    # Set specific field output (Retina LE & S)
    regionDef=abqModel.rootAssembly.allInstances['R-1'].sets['R_Set']
    abqModel.FieldOutputRequest(name='Retina_LE_S',
                                createStepName=stepName,
                                variables=('LE',

                                           'S'),
                                numIntervals=FOutputInterval,
                                region=regionDef,
                                sectionPoints=DEFAULT,
                                rebar=EXCLUDE)

    # Set specific field output (Vitreous LE & S)
    regionDef=abqModel.rootAssembly.allInstances['V-1'].sets['V_Set']
    abqModel.FieldOutputRequest(name='Vitreous_LE_S',
                                createStepName=stepName,
                                variables=('LE',
                                           'S'),
                                numIntervals=FOutputInterval,
                                region=regionDef,
                                sectionPoints=DEFAULT,
                                rebar=EXCLUDE)

    # # Set specific field output (Rigid Body U & RF)
    # regionDef=abqModel.rootAssembly.allInstances['G-1'].sets['G_RP_Set']
    # abqModel.FieldOutputRequest(name='Glue_U_RF',
```

```
    #                             createStepName=stepName,
    #                             variables=('U',
    #                                        'RF'),
    #                             numIntervals=FOutputInterval,
    #                             region=regionDef,
    #                             sectionPoints=DEFAULT,
    #                             rebar=EXCLUDE)


def H_output(stepName):
    # Internal/Kinetic Energy
    abqModel.HistoryOutputRequest(name='H-Output-1',
                                  createStepName=stepName,
                                  variables=('ALLIE',
                                             'ALLKE'))

    # # Define specific reaction force on the glue reference point
    # a = abqModel.rootAssembly
    # regionDef=a.allInstances['G-1'].sets['G_RP_Set']
    # abqModel.HistoryOutputRequest(name='G_RP_Output_U_RF_RM',
    #                               createStepName=stepName,
    #                               variables=('U1', 'U2', 'U3',
    #                                          'RF1', 'RF2', 'RF3',
    #                                          'RM1', 'RM2', 'RM3'),
    #                               region=regionDef, sectionPoints=DEFAULT,
    #                               rebar=EXCLUDE)


def General_Contact(stepName, cIP):
    # Rename the two variables
    GC_IP = 'IntProp-GC' # Interaction property
    GC = 'General_Contact' # General Contact name
    # cIP = 'cohesive_IntProp' # cohesive interaction property name
    abqModel.ContactProperty(GC_IP)

    GC_IntProp = abqModel.interactionProperties[GC_IP] # simplify code

    # if gravity == True:
    #     # Gravity keeps the vitreous from energetically moving after peeling
    GC_IntProp.TangentialBehavior(formulation=PENALTY,
                                  directionality=ISOTROPIC,
                                  slipRateDependency=OFF,
                                  pressureDependency=OFF,
                                  temperatureDependency=OFF,
                                  dependencies=0,
                                  table=((0.2, ), ),
                                  shearStressLimit=None,
                                  maximumElasticSlip=FRACTION,
                                  fraction=0.005,
                                  elasticSlipStiffness=None)
    # else:
    #     # Prevent the vitreous from sliding inside the eye holder
    #     GC_IntProp.TangentialBehavior(formulation=ROUGH)

    GC_IntProp.NormalBehavior(pressureOverclosure=HARD,
                              allowSeparation=ON,
                              constraintEnforcementMethod=DEFAULT)
    abqModel.ContactExp(name=GC, createStepName=stepName)
```

```

    GC_Int = abqModel.interactions[GC] # simplify code
    GC_Int.includedPairs.setValuesInStep(stepName=stepName, useAllstar=ON)
    GC_Int.contactPropertyAssignments.appendInStep(stepName=stepName,
                                                assignments=((GLOBAL,
                                                              SELF,
                                                              GC_IP),
                                                             )
                                                )


def updateGeneralContact(stepName, Knn, Kss, Ktt, damageInitiation,
                         tn, ts, tt, damageEvolution, FE):
    ''' Specify the cohesive surface behavior between the retina and vitreous
    during the step after the gravity step '''
    # Simplify
    GC = 'General_Contact'
    cp = 'cohesivePeel'

    abqModel.ContactProperty(cp)

    CP_IP = abqModel.interactionProperties[cp]
    CP_IP.TangentialBehavior(formulation=PENALTY,
                             directionality=ISOTROPIC,
                             slipRateDependency=OFF,
                             pressureDependency=OFF,
                             temperatureDependency=OFF,
                             dependencies=0,
                             table=((0.2, ), ),
                             shearStressLimit=None,
                             maximumElasticSlip=FRACTION,
                             fraction=0.005,
                             elasticSlipStiffness=None)

    CP_IP.CohesiveBehavior(defaultPenalties=OFF,
                           table=((Knn, Kss, Ktt), ))
    # eligibility=INITIAL_NODES,

    CP_IP.Damage(criterion=MAX_STRESS,
                 initTable=((tn, ts, tt), ),
                 useEvolution=ON,
                 evolutionType=ENERGY,
                 evolTable=((FE, ), ),
                 useStabilization=ON,
                 viscosityCoef=1e-05)

    GCI = abqModel.interactions[GC]
    if gravity == True:
        GCI.contactPropertyAssignments.changeValuesInStep(stepName=stepName,
                                                          index=1,
                                                          value=cp)
    else:
        r11=abqModel.rootAssembly.instances['R-1'].surfaces['R_V_Surf_BOND']
        r12=abqModel.rootAssembly.instances['V-1'].surfaces['V_R_Surf_BOND']
        GCI.contactPropertyAssignments.appendInStep(stepName=stepName,
                                                    assignments=((r11, r12,
                                                                  cp), ))

```

```python

def smoothGravity():
    abqModel.SmoothStepAmplitude(name='smoothGravity', timeSpan=STEP,
        data=((0.0, 0.0), (100.0, 1.0)))
    abqModel.loads['Gravity'].setValues(amplitude='smoothGravity',
        distributionType=UNIFORM, field='')


def turnTieCohesive(stepName, cohTieName):
    ''' Simulate the tie constraint with cohesive surface '''
    GC = 'General_Contact'
    CTG = cohTieName # Simplify
    abqModel.ContactProperty(CTG)

    # Simplify
    CTG_IP = abqModel.interactionProperties[CTG]
    GC_IP = abqModel.interactions[GC]

    CTG_IP.CohesiveBehavior(eligibility=INITIAL_NODES)
    r11=abqModel.rootAssembly.instances['R-1'].surfaces['R_V_Surf_BOND']
    r12=abqModel.rootAssembly.instances['V-1'].surfaces['V_R_Surf_BOND']
    GC_IP.contactPropertyAssignments.appendInStep(stepName=stepName,
                                              assignments=((r11,
                                                            r12,
                                                            CTG), ))


def peelStepPostGravity(time, stepName, previousStep, descrip, scaleFactor,
                        MSTI):
    ''' step after the gravity phase '''
    abqModel.ExplicitDynamicsStep(name=stepName,
                                  previous=previousStep,
                                  description=descrip,
                                  timePeriod=time,
                                  massScaling=((SEMI_AUTOMATIC,
                                                MODEL, AT_BEGINNING,
                                                scaleFactor, MSTI, BELOW_MIN,
                                                0, 0, 0.0, 0.0, 0, None), ))


def peelTestBCUpdate_With_Gravity(stepName):
    ''' Update the boundary conditions in the second step, post gravity '''
    abqModel.loads['Gravity'].setValuesInStep(stepName=stepName,
                                              amplitude=FREED)
    abqModel.boundaryConditions['EHR'].setValuesInStep(stepName=stepName,
                                                       vr3=-1.0)


def RG_Tie():
    a = abqModel.rootAssembly
    region1=a.instances['G-1'].surfaces['G_R_Surf']
    a = abqModel.rootAssembly
    region2=a.instances['R-1'].surfaces['R_G_Surf']
    abqModel.Tie(name='RG',
                 master=region2,
                 slave=region1,
                 positionToleranceMethod=COMPUTED,
                 adjust=OFF,
```

```
                 tieRotations=ON,
                 constraintEnforcement=SURFACE_TO_SURFACE,
                 thickness=ON)


def Amp():
    abqModel.SmoothStepAmplitude(name='TD_amp', timeSpan=STEP,
                                data=((0.0, 0.0),
                                      (30.0, 2e-05))
                                )
    abqModel.SmoothStepAmplitude(name='omega', timeSpan=STEP,
                                data=((0.0, 0.0),
                                      (30.0, 0.000909174))
                                )


def EHR_BC_Fixed(stepName):
    a = abqModel.rootAssembly
    region = a.instances['E-1'].sets['E_RP_Set']
    abqModel.VelocityBC(name='EHR', createStepName=stepName, region=region,
                        v1=0.0, v2=0.0, v3=0.0, vr1=0.0, vr2=0.0, vr3=0.0,
                        amplitude='omega', localCsys=None,
                        distributionType=UNIFORM, fieldName='')


def EHR_BC(stepName):
    a = abqModel.rootAssembly
    region = a.instances['E-1'].sets['E_RP_Set']
    abqModel.VelocityBC(name='EHR', createStepName=stepName, region=region,
                        v1=0.0, v2=0.0, v3=0.0, vr1=0.0, vr2=0.0, vr3=-1.0,
                        amplitude='omega', localCsys=None,
                        distributionType=UNIFORM, fieldName='')

def GD_BC(stepName):
    a = abqModel.rootAssembly
    region = a.instances['G-1'].sets['G_RP_Set']
    abqModel.VelocityBC(name='GD', createStepName=stepName,
                        region=region, v1=0.866092, v2=0.499884, v3=0.0,
                        vr1=0.0, vr2=0.0, vr3=0.0, amplitude='TD_amp',
                        localCsys=None, distributionType=UNIFORM,
                        fieldName='')


def TD_BC(stepName):
    a = abqModel.rootAssembly
    region = a.instances['T-1'].sets['T_RP_Set']
    abqModel.VelocityBC(name='TD', createStepName=stepName, region=region,
                        v1=0.866092, v2=0.499884, v3=0.0,
                        vr1=0.0, vr2=0.0, vr3=0.0, amplitude='TD_amp',
                        localCsys=None, distributionType=UNIFORM,
                        fieldName='')


def Retina_Disp_BC(stepName):
    a = abqModel.rootAssembly
    region = a.instances['R-1'].sets['R_G_Set']
    abqModel.VelocityBC(name='R_Vel',
                        createStepName=stepName,
```

```
                        region=region,
                        v1=0.866092,
                        v2=0.499884,
                        v3=UNSET,
                        vr1=UNSET,
                        vr2=UNSET,
                        vr3=UNSET,
                        amplitude='TD_amp',
                        localCsys=None,
                        distributionType=UNIFORM,
                        fieldName='')

def Gravity(stepName):
    abqModel.Gravity(name='Gravity', createStepName=stepName, comp2=-9.81,
                     distributionType=UNIFORM, field='')


def Write_Job(jobName, modelName, jobDescription):
    mdb.Job(name=jobName,
            model=modelName,
            description=jobDescription,
            type=ANALYSIS,
            atTime=None,
            waitMinutes=0,
            waitHours=0,
            queue=None,
            memory=90,
            memoryUnits=PERCENTAGE,
            explicitPrecision=DOUBLE,
            nodalOutputPrecision=SINGLE,
            echoPrint=OFF,
            modelPrint=OFF,
            contactPrint=OFF,
            historyPrint=OFF,
            userSubroutine='',
            scratch='',
            resultsFormat=ODB,
            parallelizationMethodExplicit=DOMAIN,
            numDomains=14,
            activateLoadBalancing=False,
            multiprocessingMode=DEFAULT,
            numCpus=14)


def Save_INP(jobName):
    mdb.jobs[jobName].writeInput(consistencyChecking=OFF)


def VR_Tie():
    a = abqModel.rootAssembly
    slaveSurf=a.instances['V-1'].surfaces['V_R_Surf_BOND']
    mastSurf=a.instances['R-1'].surfaces['R_V_Surf_BOND']
    abqModel.Tie(name='RV_Tie',
                 master=mastSurf,
                 slave=slaveSurf,
                 positionToleranceMethod=COMPUTED,
                 adjust=OFF,
                 tieRotations=ON,
```

```python
                constraintEnforcement=SURFACE_TO_SURFACE,
                thickness=ON)
    return '_VR_Tie'


def JobNameFile(modelName, fileNameAttributes, jobDescription):
    """
    Creates a txt file with the jobNames and all attributes associated with
    the model
    """
    fileName = modelName + fileNameAttributes
    outfile = open(fileName+'.txt', 'w')
    line = ('The file name indicates what parameters were used to define ' +
            'the model\n')
    outfile.write(line)
    line = '\n' + fileName + '\n'
    outfile.write(line)
    line = jobDescription
    outfile.write(line)
    outfile.close()


def Submit_job(jobname):
    myJob = mdb.jobs[jobname]
    try:
        myJob.submit(consistencyChecking=OFF)
        myJob.waitForCompletion()
    except:
        print(str(datetime.datetime.now())+' stop by error!')
        pass


def RemoveRigid(stepName):
    """ Remove the rigid bodies all together """
    a = abqModel.rootAssembly
    a.features['T-1'].suppress()
    a.features['G-1'].suppress()
    abqModel.fieldOutputRequests['Glue_U_RF'].suppress()
    # abqModel.historyOutputRequests['Contact_CP-R-G'].suppress()
    abqModel.historyOutputRequests['G_RP_Output_U_RF_RM'].suppress()
    # abqModel.interactions['CP-R-G'].suppress()
    # abqModel.constraints['RG'].suppress()
    abqModel.boundaryConditions['GD'].suppress()
    abqModel.boundaryConditions['TD'].suppress()
    r11=a.instances['E-1'].surfaces['E_Surf']
    r12=a.instances['T-1'].surfaces['T_Surf']
    r21=a.instances['E-1'].surfaces['E_Surf']
    r22=a.instances['G-1'].surfaces['G_Surf']
    r31=a.instances['G-1'].surfaces['G_Surf']
    r32=a.instances['T-1'].surfaces['T_Surf']

    GC = 'General_Contact' # simplify
    GCI = abqModel.interactions[GC]
    GCI.excludedPairs.setValuesInStep(stepName=stepName,
                                      removePairs=((r11, r12),
                                                   (r21, r22),
                                                   (r31, r32)))
    region = a.instances['R-1'].sets['R_G_Set']
```

```
    abqModel.VelocityBC(name='R_Vel',
                        createStepName=stepName,
                        region=region,
                        v1=0.866092,
                        v2=0.499884,
                        v3=UNSET,
                        vr1=UNSET,
                        vr2=UNSET,
                        vr3=UNSET,
                        amplitude='TD_amp',
                        localCsys=None,
                        distributionType=UNIFORM,
                        fieldName='')


def CohesiveSurface(Knn, Kss, Ktt, damageInitiation, tn, ts, tt,
                    damageEvolution, FE):
    coh_int_prop = 'cohesive_IntProp' # simplify
    abqModel.ContactProperty(coh_int_prop)
    C_IP = abqModel.interactionProperties[coh_int_prop] # simplify code

    C_IP.TangentialBehavior(formulation=PENALTY,
                            directionality=ISOTROPIC,
                            slipRateDependency=OFF,
                            pressureDependency=OFF,
                            temperatureDependency=OFF,
                            dependencies=0,
                            table=((0.2, ), ),
                            shearStressLimit=None,
                            maximumElasticSlip=FRACTION,
                            fraction=0.005,
                            elasticSlipStiffness=None)

    C_IP.CohesiveBehavior(defaultPenalties=OFF,
                          table=((Knn,
                                  Kss,
                                  Ktt), ))

    if damageInitiation == True and damageEvolution == True:
        # If damage initiation and evolution are both turned on
        C_IP.Damage(criterion=MAX_STRESS,
                    initTable=((tn,
                                ts,
                                tt), ),
                    useEvolution=ON,
                    evolutionType=ENERGY,
                    softening=LINEAR,
                    evolTable=((FE, ), ),
                    useStabilization=ON,
                    viscosityCoef=1e-5) # was EXPONENTIAL, LINEAR

    elif damageInitiation == True and damageEvolution == False:
        # If damage initation is turned on but evolution is not
        C_IP.Damage(criterion=MAX_STRESS,
                    initTable=((tn,
                                ts,
                                tt), ),
                    useEvolution=OFF,
```

```
                    useStabilization=OFF)
    else:
        print('No damage initiation or evolution')


def FEA():
    """
    Function that generates FEA code to model vitreoretinal adhesion

    # Steps are as follows:
        1 - Create new model database
        2 - Import SolidWorks STEP file (Includes all parts)
        3 - Material property definitions
        4 - Part features (Element & Node Sets & Reference Points ...)
        5 - Mesh parts (Specify seed size)
        6 - Assembly
        7 - Step (Dynamic Explicit with Mass Scaling)
        8 - Outputs (Field & History)
        9 - Contact (General Contact)
        10 - Contact pair (Retina/Vitreous - Cohesive Surface)
        11 - Tie Constraint (Retina - Glue)
        12 - Amplitude definition
        13 - BC's'
        14 - Submit Job  :)
    """

    # Import SolidWorks STEP file
    ImportStepEyeConstrained()

    # Mat Props
    Retina_Mat_Prop(RetinaProp)
    Vitreous_Mat_Prop(VitreousProp)

    # # Part Geometry/RPs/Sets/Surfaces
    E_Features()
    G_Features()
    T_Features()
    R_Features()

    # Internal sphere to reduce mesh
    V_Partition_XYZ_Axis()
    V_Internal_Sphere()
    AssembleV_for_Merging()
    mergeV()

    # Features on the vitreous
    PartitionRetinaOnVitreous()
    Vitreous_Features()

    # Seed & Mesh parts
    E_Mesh(e1Seed, e2Seed) # Max/min
    G_Mesh(gSeed)
    T_Mesh(ptSeed)
    R_Mesh(rSeed)
    VitreousMesh(v1Seed, v2Seed)

    # Assembly
    Assembly()
```

```
    # Eliminate the glue and tab from the model
    a = abqModel.rootAssembly
    a.features['G-1'].suppress()
    a.features['T-1'].suppress()

    # Gravity Step
    previousStep = 'Initial'
    if gravity == True:
        stepName = 'Gravity_Step'
        descrip = ('Applying gravity to the model and letting the ' +
                   'vitreous and retina settle')

        GravityStep(200, previousStep, scaleFactor, 0.03125, stepName, descrip)
        Gravity(stepName)
        smoothGravity()

        # Interactions
        cohTieName = 'Cohesive_Gravity_Tie'
        General_Contact(stepName, cohTieName)

        # Interaction properties
        turnTieCohesive(stepName, cohTieName)

        # Zero movement boundary conditions
        Amp()
        EHR_BC_Fixed(stepName)

        # # Model outputs for gravity step
        F_output(stepName)
        H_output(stepName)

        previousStep = stepName # Update the previous step to be gravity
    else:
        ''' General contact ''' # fix here if no gravity is specified
        peelCoh = 'Cohesive_Peel_Int'
        General_Contact(previousStep, peelCoh)

    # # Peel Step
    stepName = 'Peel_Test_Dynamic_Explicit'
    descrip = 'Peel the retina away from the vitreous (rotational peel test)'
    peelStepPostGravity(time, stepName, previousStep, descrip, scaleFactor,
                        MSTI)

    updateGeneralContact(stepName, Knn, Kss, Ktt, damageInitiation,
                         tn, ts, tt, damageEvolution, FE)

    if tieInterface == True:
        # Tie the interface together
        VR_sym_tie()

    # Boundary Conditions
    Amp()
    if gravity == True:
        peelTestBCUpdate_With_Gravity(stepName)
    else:
        EHR_BC(stepName)
        # GD_BC(stepName) # Not used anymore
```

```
        # TD_BC(stepName) # Not used anymore

        # Model Outputs
        F_output(stepName)
        H_output(stepName)

    Retina_Disp_BC(stepName)

    # Undo the spacing to pass in the job description
    global jobDescription
    # replace new lines, spaces, equal signs
    jobDescription = jobDescription.replace('NEWLINE', '\n')
    jobDescription = jobDescription.replace('TAB', '\t')
    jobDescription = jobDescription.replace('SPACE', ' ')
    jobDescription = jobDescription.replace('EQUALSSIGN', '=')

    Write_Job(jobName, modelName, jobDescription)
    print('Job has been written')
    Save_INP(jobName)
    Submit_job(jobName)
    print('Job has been submitted')
    del mdb.models['Model-1']

# In[Symmetric Model]
"""
Symmetry
"""

def VR_sym_tie():
    a = abqModel.rootAssembly
    mastSurf=a.instances['R-1'].surfaces['R_V_Surf_BOND']
    slaveSurf=a.instances['V-1'].surfaces['V_R_Surf_BOND']
    abqModel.Tie(name='VR_Tie',
                 master=mastSurf,
                 slave=slaveSurf,
                 positionToleranceMethod=COMPUTED,
                 adjust=OFF,
                 tieRotations=ON,
                 constraintEnforcement=SURFACE_TO_SURFACE,
                 thickness=ON)


def E_sym_Constrained():
    p = abqModel.parts['E']
    c = p.cells
    pickedCells = c.getSequenceFromMask(mask=('[#1 ]', ), )
    v1, e1, d1 = p.vertices, p.edges, p.datums
    p.PartitionCellByPlaneThreePoints(cells=pickedCells,
                                      point1=p.InterestingPoint(edge=e1[4],
                                                                rule=MIDDLE),
                                      point2=p.InterestingPoint(edge=e1[18],
                                                                rule=MIDDLE),
                                      point3=p.InterestingPoint(edge=e1[7],
                                                                rule=MIDDLE))
    f = p.faces
    p.RemoveFaces(faceList = f[3:4]+f[5:6]+f[7:8]+f[9:12]+f[15:16]+f[17:20]+
                  f[21:22]+f[25:26]+f[27:28]+f[29:30], deleteCells=False)

```

```python
    # Reference point
    p.ReferencePoint(point=(0.0, 0.0, 0.0))

    r = p.referencePoints
    refPoints=(r[4], )
    p.Set(referencePoints=refPoints, name='E_RP_Set')

    # Sets
    # Edge seeds
    e = p.edges
    edges = e.getSequenceFromMask(mask=('[#ffd03fd0 #131f ]', ), )
    p.Set(edges=edges, name='E_Outside_Edge_Seed_Set')
    edges = e.getSequenceFromMask(mask=('[#bc007 #c80 ]', ), )
    p.Set(edges=edges, name='E_Edge_Seed_Set')

    # # Define Surface
    s = p.faces
    side1Faces = s.getSequenceFromMask(mask=('[#1ffff ]', ), )
    p.Surface(side1Faces=side1Faces, name='E_Surf')

    # Remove cells
    c = p.cells
    p.RemoveCells(cellList = c[0:1])

    # redefine the E set no w that the cells have been removed
    r = p.referencePoints
    refPoints=(r[4], )
    p.Set(referencePoints=refPoints, name='E_Set')


def G_sym_Constrained():

    p = abqModel.parts['G']
    c = p.cells
    pickedCells = c.getSequenceFromMask(mask=('[#1 ]', ), )
    v, e, d = p.vertices, p.edges, p.datums
    p.PartitionCellByPlaneThreePoints(cells=pickedCells,
                                      point1=p.InterestingPoint(edge=e[10],
                                                                rule=MIDDLE),
                                      point2=p.InterestingPoint(edge=e[11],
                                                                rule=MIDDLE),
                                      point3=p.InterestingPoint(edge=e[1],
                                                                rule=MIDDLE))

    f1 = p.faces
    p.RemoveFaces(faceList = f1[2:3]+f1[4:5]+f1[7:8]+f1[9:11],
                  deleteCells=False)

    v1, e1, d1, n = p.vertices, p.edges, p.datums, p.nodes
    p.ReferencePoint(point=v1[2])

    r = p.referencePoints
    refPoints=(r[4], )
    p.Set(referencePoints=refPoints, name='G_RP_Set')

    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#2 ]', ), )
    p.Set(faces=faces, name='G_T_Set')
```

```
    faces = f.getSequenceFromMask(mask=('[#8 ]', ), )
    p.Set(faces=faces, name='G_R_Set')

    s = p.faces
    side1Faces = s.getSequenceFromMask(mask=('[#3f ]', ), )
    p.Surface(side1Faces=side1Faces, name='G_Surf')

    side1Faces = s.getSequenceFromMask(mask=('[#8 ]', ), )
    p.Surface(side1Faces=side1Faces, name='G_R_Surf')

    side1Faces = s.getSequenceFromMask(mask=('[#2 ]', ), )
    p.Surface(side1Faces=side1Faces, name='G_T_Surf')

    # Remove cells
    c1 = p.cells
    p.RemoveCells(cellList = c1[0:1])

    # redefine the set to be the reference point
    r = p.referencePoints
    refPoints=(r[4], )
    p.Set(referencePoints=refPoints, name='G_Set')


def T_sym_constrained():
    p = abqModel.parts['T']
    c = p.cells
    pickedCells = c.getSequenceFromMask(mask=('[#1 ]', ), )
    v, e, d = p.vertices, p.edges, p.datums
    p.PartitionCellByPlaneThreePoints(cells=pickedCells,
                                  point1=p.InterestingPoint(edge=e[11],
                                                            rule=MIDDLE),
                                  point2=p.InterestingPoint(edge=e[7],
                                                            rule=MIDDLE),
                                  point3=p.InterestingPoint(edge=e[5],
                                                            rule=MIDDLE))

    f = p.faces
    p.RemoveFaces(faceList = f[1:2]+f[4:5]+f[6:9]+f[11:13], deleteCells=False)

    # reference point
    v1, e1, d1, n1 = p.vertices, p.edges, p.datums, p.nodes
    p.ReferencePoint(point=v1[3])

    # Sets
    r = p.referencePoints
    refPoints=(r[4], )
    p.Set(referencePoints=refPoints, name='T_RP_Set')

    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#2 ]', ), )
    p.Set(faces=faces, name='T_G_Set')

    # Surfaces
    s = p.faces
    side1Faces = s.getSequenceFromMask(mask=('[#ff ]', ), )
    p.Surface(side1Faces=side1Faces, name='T_Surf')

```

```
    side1Faces = s.getSequenceFromMask(mask=('[#2 ]', ), )
    p.Surface(side1Faces=side1Faces, name='T_G_Surf')

    c = p.cells
    p.RemoveCells(cellList = c[0:1])

    # Redefine the set to be the reference point
    r = p.referencePoints
    refPoints=(r[4], )
    p.Set(referencePoints=refPoints, name='T_Set')


def R_sym_constrained():

    p = abqModel.parts['R']
    c = p.cells
    pickedCells = c.getSequenceFromMask(mask=('[#1 ]', ), )
    v1, e1, d1 = p.vertices, p.edges, p.datums
    p.PartitionCellByPlaneThreePoints(cells=pickedCells,
                                      point1=p.InterestingPoint(edge=e1[1],
                                                                rule=MIDDLE),
                                      point2=p.InterestingPoint(edge=e1[6],
                                                               rule=MIDDLE),
                                      point3=p.InterestingPoint(edge=e1[7],
                                                                rule=MIDDLE))

    f1 = p.faces
    p.RemoveFaces(faceList = f1[1:2]+f1[4:5]+f1[6:9]+f1[11:13],
                  deleteCells=False)

    c = p.cells
    cells = c.getSequenceFromMask(mask=('[#1 ]', ), )
    p.Set(cells=cells, name='R_Set')

    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#6 ]', ), )
    p.Set(faces=faces, name='R_G_Set')

    faces = f.getSequenceFromMask(mask=('[#10 ]', ), )
    p.Set(faces=faces, name='R_V_Set')

    faces = f.getSequenceFromMask(mask=('[#1 ]', ), )
    p.Set(faces=faces, name='R_SYM_BC_SET')

    s = p.faces
    side1Faces = s.getSequenceFromMask(mask=('[#ff ]', ), )
    p.Surface(side1Faces=side1Faces, name='R_Surf')

    side1Faces = s.getSequenceFromMask(mask=('[#6 ]', ), )
    p.Surface(side1Faces=side1Faces, name='R_G_Surf')

    side1Faces = s.getSequenceFromMask(mask=('[#10 ]', ), )
    p.Surface(side1Faces=side1Faces, name='R_V_Surf_BOND')

    side1Faces = s.getSequenceFromMask(mask=('[#1 ]', ), )
    p.Surface(side1Faces=side1Faces, name='R_SYM_BC_SURF')

    # Assign section
```

```
    region = p.sets['R_Set']
    p.SectionAssignment(region=region, sectionName='Retina_Section',
                        offset=0.0, offsetType=MIDDLE_SURFACE, offsetField='',
                        thicknessAssignment=FROM_SECTION)


def V_partition_Sphere():
    p = abqModel.parts['V']
    c = p.cells
    pickedCells = c.getSequenceFromMask(mask=('[#1 ]', ), )
    v1, e1, d1 = p.vertices, p.edges, p.datums
    p.PartitionCellByPlaneThreePoints(point1=v1[1],
                                      point2=v1[0],
                                      point3=v1[3],
                                      cells=pickedCells)

    pickedCells = c.getSequenceFromMask(mask=('[#3 ]', ), )
    v2, e, d2 = p.vertices, p.edges, p.datums
    p.PartitionCellByPlaneThreePoints(point1=v2[4],
                                      point2=v2[1],
                                      point3=v2[5],
                                      cells=pickedCells)

    pickedCells = c.getSequenceFromMask(mask=('[#f ]', ), )
    v1, e1, d1 = p.vertices, p.edges, p.datums
    p.PartitionCellByPlaneThreePoints(point1=v1[2],
                                      point2=v1[5],
                                      point3=v1[3],
                                      cells=pickedCells)


def Assembly_sym_constrain():
    a1 = abqModel.rootAssembly
    a1.DatumCsysByDefault(CARTESIAN)
    p = abqModel.parts['E']
    a1.Instance(name='E-1', part=p, dependent=ON)
    p = abqModel.parts['G']
    a1.Instance(name='G-1', part=p, dependent=ON)
    p = abqModel.parts['R']
    a1.Instance(name='R-1', part=p, dependent=ON)
    p = abqModel.parts['T']
    a1.Instance(name='T-1', part=p, dependent=ON)
    p = abqModel.parts['V']
    a1.Instance(name='V-1', part=p, dependent=ON)
    p = abqModel.parts['V_internal']
    a1.Instance(name='V_internal-1', part=p, dependent=ON)


def mergeV_sym():
    # Merge the vitreous and the internal sphere
    a = abqModel.rootAssembly
    a.InstanceFromBooleanMerge(name='V_Merge',
                               instances=(a.instances['V-1'],
                                          a.instances['V_internal-1'], ),
                               keepIntersections=ON,
                               originalInstances=DELETE,
                               domain=GEOMETRY)

```

```python
    # Clean up file names after merge
    del abqModel.parts['V']
    del abqModel.parts['V_internal']

    abqModel.parts.changeKey(fromName='V_Merge', toName='V')
    a = abqModel.rootAssembly
    a.regenerate()
    abqModel.rootAssembly.features.changeKey(fromName='V_Merge-1',
                                             toName='V-1')

    p = abqModel.parts['V']
    f = p.faces
    p.RemoveFaces(faceList = f[0:3]+f[4:5]+f[8:9]+f[12:13]+f[15:16]+
                  f[19:21]+f[23:24]+f[26:27]+f[28:29]+f[32:36],
                  deleteCells=False)

    a.regenerate()


def V_sym_constrained():
    # Partition V along the width of the retina
    p = abqModel.parts['V']
    p.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=-0.00254)
    abqModel.parts['V'].features.changeKey(fromName='Datum plane-1',
                                           toName='Retina_Width')
    c = p.cells
    pickedCells = c.getSequenceFromMask(mask=('[#14 ]', ), )
    d1 = p.datums
    p.PartitionCellByDatumPlane(datumPlane=d1[3], cells=pickedCells)

    # # Left side of vitreous 22.5 degrees
    # # Right side of vitreous 18.875000
    p.DatumAxisByPrincipalAxis(principalAxis=ZAXIS)
    p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset=0.0)

    d2 = p.datums
    p.DatumPlaneByRotation(plane=d2[6], axis=d2[5], angle=18.75)

    d1 = p.datums
    ''' angle=-22.5 was the previous angle for the back side of the retina,
    because of extreme element deformation, a new model was Created'''
    # ang = -22.5 # Previous
    ang = -18.75*2 # Updated angle
    p.DatumPlaneByRotation(plane=d2[7], axis=d1[5], angle=ang)
    c = p.cells
    pickedCells = c.getSequenceFromMask(mask=('[#2 ]', ), )
    d2 = p.datums
    p.PartitionCellByDatumPlane(datumPlane=d2[7], cells=pickedCells)

    pickedCells = c.getSequenceFromMask(mask=('[#40 ]', ), )
    d1 = p.datums
    p.PartitionCellByDatumPlane(datumPlane=d1[8], cells=pickedCells)

    # Define sets
    c = p.cells
    cells = c.getSequenceFromMask(mask=('[#fff ]', ), )
    p.Set(cells=cells, name='V_Set')

```

```
    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#8080 ]', ), )
    p.Set(faces=faces, name='V_R_Set')

    # Symmetric BC
    faces = f.getSequenceFromMask(mask=('[#17000042 #6a ]', ), )
    p.Set(faces=faces, name='V_SYM_BC_SET')

    # Surfaces
    s = p.faces
    side1Faces = s.getSequenceFromMask(mask=('[#1700a0ca #7ea ]', ), )
    p.Surface(side1Faces=side1Faces, name='V_Surf')
    side1Faces = s.getSequenceFromMask(mask=('[#8080 ]', ), )
    p.Surface(side1Faces=side1Faces, name='V_R_Surf_BOND')

    # Symmetric BC
    side1Faces = s.getSequenceFromMask(mask=('[#17000042 #6a ]', ), )
    p.Surface(side1Faces=side1Faces, name='V_SYM_BC_SURF')

    # Assign section
    region = p.sets['V_Set']
    p.SectionAssignment(region=region, sectionName='Vitreous_Section',
                        offset=0.0, offsetType=MIDDLE_SURFACE, offsetField='',
                        thicknessAssignment=FROM_SECTION)


def E_sym_constrain_msh(e1Seed, e2Seed):
    p = abqModel.parts['E']
    e = p.edges
    pickedEdges = e.getSequenceFromMask(mask=('[#bc007 #c80 ]', ), )
    p.seedEdgeBySize(edges=pickedEdges,
                     size=e1Seed,
                     deviationFactor=0.1,
                     constraint=FINER)
    pickedEdges = e.getSequenceFromMask(mask=('[#ffd03fd0 #131f ]', ), )
    p.seedEdgeBySize(edges=pickedEdges,
                     size=e2Seed,
                     deviationFactor=0.1,
                     constraint=FINER)
    elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=EXPLICIT)
    elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=EXPLICIT)
    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#1ffff ]', ), )
    pickedRegions =(faces, )
    p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
    # (unique node numbering)
    p.setValues(startNodeLabel=1000000, startElemLabel=1000000)
    p.generateMesh()


def G_sym_constrain_msh(gSeed):
    p = abqModel.parts['G']
    p.seedPart(size=gSeed, deviationFactor=0.1, minSizeFactor=0.1)
    elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=EXPLICIT)
    elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=EXPLICIT)
    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#3f ]', ), )
    pickedRegions =(faces, )
```

```python
    p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
    # (unique node numbering)
    p.setValues(startNodeLabel=2000000, startElemLabel=2000000)
    p.generateMesh()


def T_sym_constrain_msh(ptSeed):
    p = abqModel.parts['T']
    p.seedPart(size=ptSeed, deviationFactor=0.1, minSizeFactor=0.1)
    elemType1 = mesh.ElemType(elemCode=R3D4, elemLibrary=EXPLICIT)
    elemType2 = mesh.ElemType(elemCode=R3D3, elemLibrary=EXPLICIT)
    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#ff ]', ), )
    pickedRegions =(faces, )
    p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
    # (unique node numbering)
    p.setValues(startNodeLabel=3000000, startElemLabel=3000000)
    p.generateMesh()


def R_sym_constrain_msh(rSeed):
    p = abqModel.parts['R']
    e = p.edges
    pickedEdges = e.getSequenceFromMask(mask=('[#3ffff ]', ), )
    p.seedEdgeBySize(edges=pickedEdges,
                     size=rSeed,
                     deviationFactor=0.1,
                     constraint=FINER)
    c = p.cells
    pickedRegions = c.getSequenceFromMask(mask=('[#1 ]', ), )
    p.setMeshControls(regions=pickedRegions, technique=SWEEP,
        algorithm=ADVANCING_FRONT)
    c, e1 = p.cells, p.edges
    p.setSweepPath(region=c[0], edge=e1[3], sense=REVERSE)
    elemType1 = mesh.ElemType(elemCode=C3D8R,
                              elemLibrary=EXPLICIT,
                              kinematicSplit=AVERAGE_STRAIN,
                              secondOrderAccuracy=ON,
                              hourglassControl=ENHANCED,
                              distortionControl=ON,
                              lengthRatio=0.100000001490116)
    elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
    elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT)
    # c = p.cells
    cells = c.getSequenceFromMask(mask=('[#1 ]', ), )
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions, elemTypes=(elemType1,
                                                       elemType2,
                                                       elemType3))
    # (unique node numbering)
    p.setValues(startNodeLabel=4000000, startElemLabel=4000000)
    p.generateMesh()


def VseedPart(v2Seed):
    ''' Seed the entire vitreous '''
    p = abqModel.parts['V']
    p.seedPart(size=v2Seed, deviationFactor=0.1, minSizeFactor=0.1)
```

```python


def V_SeedTop(v1Seed):
    ''' Seed the top of the vitreous where the retina is bonded '''
    p = abqModel.parts['V']
    e = p.edges
    pickedEdges = e.getSequenceFromMask(mask=('[#ffffffff #f ]', ), )
    p.seedEdgeBySize(edges=pickedEdges,
                     size=v1Seed,
                     deviationFactor=0.1,
                     constraint=FINER)


def vitreous_seed_bias(v1Seed, v2Seed):
    ''' Seed the outside edges of the vitreous leading up to the bonded
    interface with biased mesh to weight the attachment area '''
    p = abqModel.parts['V']
    e = p.edges
    pickedEdges1 = e.getSequenceFromMask(mask=('[#0 #100040 ]', ), )
    pickedEdges2 = e.getSequenceFromMask(mask=('[#0 #400000 ]', ), )
    p.seedEdgeByBias(biasMethod=SINGLE,
                     end1Edges=pickedEdges1,
                     end2Edges=pickedEdges2,
                     minSize=v1Seed,
                     maxSize=v2Seed,
                     constraint=FINER)


def vitreous_Seed_Bottom_Bias(v1Seed, v2Seed):
    ''' Seed the bottom of the vitreous '''
    p = abqModel.parts['V']
    e = p.edges
    pickedEdges1 = e.getSequenceFromMask(mask=('[#0 #10000 ]', ), )
    pickedEdges2 = e.getSequenceFromMask(mask=('[#0 #1000 ]', ), )
    p.seedEdgeByBias(biasMethod=SINGLE,
                     end1Edges=pickedEdges1,
                     end2Edges=pickedEdges2,
                     minSize=v1Seed,
                     maxSize=v2Seed,
                     constraint=FINER)


def vHex():
    ''' Hexahedral mesh definition for the vitreous '''
    p = abqModel.parts['V']
    elemType1 = mesh.ElemType(elemCode=C3D8R,
                              elemLibrary=EXPLICIT,
                              kinematicSplit=AVERAGE_STRAIN,
                              secondOrderAccuracy=ON,
                              hourglassControl=ENHANCED,
                              distortionControl=ON,
                              lengthRatio=0.100000001490116)
    elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
    elemType3 = mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT)

    c = p.cells
    cells = c.getSequenceFromMask(mask=('[#4bf ]', ), )
    pickedRegions =(cells, )
```

```
    p.setElementType(regions=pickedRegions,
                     elemTypes=(elemType1, elemType2, elemType3))


def vTet():
    ''' Tetrahedral mesh definition for the vitreous '''
    p = abqModel.parts['V']
    c = p.cells
    pickedRegions = c.getSequenceFromMask(mask=('[#b40 ]', ), )
    p.setMeshControls(regions=pickedRegions, elemShape=TET, technique=FREE)
    elemType1 = mesh.ElemType(elemCode=C3D20R)
    elemType2 = mesh.ElemType(elemCode=C3D15)
    elemType3 = mesh.ElemType(elemCode=C3D10)

    c = p.cells
    cells = c.getSequenceFromMask(mask=('[#b40 ]', ), )
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2,
        elemType3))
    elemType1 = mesh.ElemType(elemCode=C3D8R, elemLibrary=EXPLICIT)
    elemType2 = mesh.ElemType(elemCode=C3D6, elemLibrary=EXPLICIT)
    elemType3 = mesh.ElemType(elemCode=C3D4,
                              elemLibrary=EXPLICIT,
                              secondOrderAccuracy=ON,
                              distortionControl=ON,
                              lengthRatio=0.100000001490116)

    cells = c.getSequenceFromMask(mask=('[#b40 ]', ), )
    pickedRegions =(cells, )
    p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2,
        elemType3))


def V_generate_mesh():
    ''' Mesh the vitreous '''
    p = abqModel.parts['V']
    # (unique node numbering)
    p.setValues(startNodeLabel=5000000, startElemLabel=5000000)
    p.generateMesh()


def V_sym_constrain_msh(v1Seed, v2Seed):
    ''' Mesh the vitreous with the two different seed sizes
    Seed the part
    Seed the top
    bias the edge
    seed the bottom
    hexahedral elements
    tetrahedral elements
    generate mesh '''
    VseedPart(v2Seed)
    V_SeedTop(v1Seed)
    vitreous_seed_bias(v1Seed, v2Seed)
    vitreous_Seed_Bottom_Bias(v1Seed, v2Seed)
    vHex()
    vTet()
    V_generate_mesh()
```

```


def V_SYM_Constrain_BC(stepName):
    a = abqModel.rootAssembly
    f = a.instances['V-1'].faces
    faces = f.getSequenceFromMask(mask=('[#17000042 #6a ]', ), )
    region = a.Set(faces=faces, name='V_SYM_BC_SET')
    abqModel.ZsymmBC(name='V_sym',
                     createStepName=stepName,
                     region=region,
                     localCsys=None)


def R_SYM_Constrain_BC(stepName):
    a = abqModel.rootAssembly
    f = a.instances['R-1'].faces
    faces = f.getSequenceFromMask(mask=('[#1 ]', ), )
    region = a.Set(faces=faces, name='R_SYM_BC_SET')
    abqModel.ZsymmBC(name='R_sym',
                     createStepName=stepName,
                     region=region,
                     localCsys=None)


""" Write the FEA Code """
def FEA_Symmetry():
    """
    Function that generates FEA code to model vitreoretinal adhesion

    # Steps are as follows:
        1 - Create new model database
        2 - Import SolidWorks STEP file (Includes all parts)
        3 - Material property definitions
        4 - Part features (Element & Node Sets & Reference Points ...)
        5 - Mesh parts (Specify seed size)
        6 - Assembly
        7 - Step (Dynamic Explicit with Mass Scaling)
        8 - Outputs (Field & History)
        9 - Contact (General Contact)
        10 - Contact pair (Retina/Vitreous - Bonded Surface)
        11 - Tie Constraint (Retina - Glue)
        12 - Amplitude definition
        13 - BC's'
        14 - Submit Job  :)
    """

    # Import SolidWorks STEP file
    ImportStepEyeConstrained()

    # Mat Props
    Retina_Mat_Prop(RetinaProp)
    Vitreous_Mat_Prop(VitreousProp)

    """ Constrained vitreous """
    # Pat Geometry/RPs/Sets/Surfaces
    E_sym_Constrained()
    G_sym_Constrained()
    T_sym_constrained()
```

```python
    R_sym_constrained()

    # Define and then merge in the assembly to reduce computational time
    V_Internal_Sphere()

    # # Assembly
    Assembly_sym_constrain()

    # partition Vitreous x,y,z plane
    V_partition_Sphere()

    # Merge V and V Int
    mergeV_sym()

    # Update V sets
    V_sym_constrained()

    # Mesh parts
    E_sym_constrain_msh(e1Seed, e2Seed)
    G_sym_constrain_msh(gSeed)
    T_sym_constrain_msh(ptSeed)
    V_sym_constrain_msh(v1Seed, v2Seed)
    R_sym_constrain_msh(rSeed)

    # # Convert Hexahedral elements to quadratic tetrahedral elements
    # QuadraticTetVitreous()
    # QuadraticTetRetina()

    # Eliminate the glue and tab from the model
    a = abqModel.rootAssembly
    a.features['G-1'].suppress()
    a.features['T-1'].suppress()

    # Gravity Step
    previousStep = 'Initial'
    if gravity == True:
        stepName = 'Gravity_Step'
        descrip = ('Applying gravity to the model and letting the ' +
                   'vitreous and retina settle')

        GravityStep(200, previousStep, scaleFactor, 0.03125, stepName, descrip)
        Gravity(stepName)
        smoothGravity()

        # Interactions
        cohTieName = 'Cohesive_Gravity_Tie'
        General_Contact(stepName, cohTieName)

        # Interaction properties
        turnTieCohesive(stepName, cohTieName)

        V_SYM_Constrain_BC(stepName)
        R_SYM_Constrain_BC(stepName)

        # Zero movement boundary conditions
        Amp()
        EHR_BC_Fixed(stepName)

```

```
        # # Model outputs for gravity step
        F_output(stepName)
        H_output(stepName)

        previousStep = stepName # Update the previous step to be gravity
    else:
        ''' General contact ''' # fix here if no gravity is specified
        peelCoh = 'Cohesive_Peel_Int'
        General_Contact(previousStep, peelCoh)

    # # Peel Step
    stepName = 'Peel_Test_Dynamic_Explicit'
    descrip = 'Peel the retina away from the vitreous (rotational peel test)'
    peelStepPostGravity(time, stepName, previousStep, descrip, scaleFactor,
                        MSTI)

    updateGeneralContact(stepName, Knn, Kss, Ktt, damageInitiation,
                         tn, ts, tt, damageEvolution, FE)

    if tieInterface == True:
        # Tie the interface together
        VR_sym_tie()

    # Boundary Conditions
    Amp()
    if gravity == True:
        peelTestBCUpdate_With_Gravity(stepName)

    else:
        EHR_BC(stepName)
        # GD_BC(stepName) # Not used anymore
        # TD_BC(stepName) # Not used anymore
        V_SYM_Constrain_BC(stepName)
        R_SYM_Constrain_BC(stepName)

        # Model Outputs
        F_output(stepName)
        H_output(stepName)

    Retina_Disp_BC(stepName)

    # Undo the spacing to pass in the job description
    global jobDescription
    # replace new lines, spaces, equal signs
    jobDescription = jobDescription.replace('NEWLINE', '\n')
    jobDescription = jobDescription.replace('TAB', '\t')
    jobDescription = jobDescription.replace('SPACE', ' ')
    jobDescription = jobDescription.replace('EQUALSSIGN', '=')

    Write_Job(jobName, modelName, jobDescription)
    print('Job has been written')
    Save_INP(jobName)
    Submit_job(jobName)
    print('Job has been submitted')
    del mdb.models['Model-1']


# In[Main import info]
```

```
if __name__ == '__main__':
    """ Run the following function """

    # Print File of tests & attributes ran in order to make sure they are
    # being properly pass through
    print("\nWriting out the Argument Data...")
    filename = os.path.join(abqWD, 'FEAArgumentData' + '.txt')
    outfile = open(filename,'w')
    outfile.write('sys.argv\n')
    outfile.write('\n'.join(sys.argv)) # write all arguments passed into abaqus
    outfile.close()
    print("\nDone!")
    print("\nThe output file will be named '{}".format(filename) + "'")
    print("\nIt will be in the same working directory as your Abaqus model\n")

    # # Testing when importing into abaqus script
    # gravity =              eval('True') # gravity
    # symmetry =             eval('False') # symmetry
    # simplified=            eval('True') # simplified model
    # modelName =                 'T1Si' # model name
    # jobName =                   'test' # file name/job name
    # time =                float('100')
    # e1Seed =                    '[10,1,0.0009765625]'
    # e2Seed =                    '[8,1,0.00390625]'
    # ptSeed =                    '[6,1,0.015625]'
    # gSeed =                     '[7,1,0.0078125]'
    # v1Seed =                    '[10,1,0.0009765625]' #
    ↪  '[11.38,1,0.00037521366730664343]' #
    # v2Seed =                    '[8,1,0.00390625]'
    # rSeed =                     '[10,1,0.0009765625]' #
    ↪  '[11.3275,1,0.0003891192571059363]' #
    # scaleFactor =               '[0,1,1]'
    # MSTI =                      '[4,1,0.0625]' # MassScaleTimeIncrement
    # RetinaProp =          float('11120.0') # Young's modulus for retina
    # VitreousProp =        float('100') # '69.56549028991259') # Young's modulus for
    ↪  vitreous 386.717932801091
    # KnnString =              str([26.21216496521396,1,77740603.15760481]) #
    # KssString =              str([27.992885300905385,1,267114916.34363237]) #
    # KttString =              str([27.65583405906571,1,211463592.90645516])
    # damageInitiation =          True # True/False
    # tnString =               str([18.830816653206917,1,466273.4160693089])
    # tsString =               str([17.49221225177773,1,184365.8917311695]) # Damage
    ↪  initiation
    # ttString =               str([6.532865971598381 4,1,92.59523006880973]) # Damage
    ↪  initiation
    # damageEvolution =           True # True/false convert to bool
    # FEString =               str([-0.9185766704351879,1,0.5290306923507394])
    # OptimizationStatus =        True
    # tieInterface =              True
    # jobDescription =            """Test""" #'Test MODEL Cube Script'


    # Pass in arguments from previous file Strip the brackets from the strings
    gravity =              eval(sys.argv[-29]) # gravity
    symmetry =             eval(sys.argv[-28]) # symmetry
    simplified =           eval(sys.argv[-27]) # simplified model
    modelName =                 sys.argv[-26] # model name
```

```
    jobName =                   sys.argv[-25] # file name/job name
    time =                float(sys.argv[-24])
    e1Seed =                    sys.argv[-23]
    e2Seed =                    sys.argv[-22]
    ptSeed =                    sys.argv[-21]
    gSeed =                     sys.argv[-20]
    v1Seed =                    sys.argv[-19]
    v2Seed =                    sys.argv[-18]
    rSeed =                     sys.argv[-17]
    scaleFactor =               sys.argv[-16]
    MSTI =                      sys.argv[-15] # MassScaleTimeIncrement
    RetinaProp =          float(sys.argv[-14]) # Young's modulus for retina
    VitreousProp =        float(sys.argv[-13]) # Young's modulus for vitreous
    KnnString =                 sys.argv[-12] # Cohesive behavior
    KssString =                 sys.argv[-11] # Cohesive behavior
    KttString =                 sys.argv[-10] # Cohesive behavior
    damageInitiation =     eval(sys.argv[-9]) # True/false convert to bool
    tnString =                  sys.argv[-8] # Damage initiation
    tsString =                  sys.argv[-7] # Damage initiation
    ttString =                  sys.argv[-6] # Damage initiation
    damageEvolution =      eval(sys.argv[-5]) # True/false convert to bool
    FEString =                  sys.argv[-4] # Fracture energy
    optimizationStatus =        sys.argv[-3] # None/variables to be optimized
    tieInterface =         eval(sys.argv[-2]) # True/false convert to bool
    jobDescription =            sys.argv[-1] # String


    """ Convert the strings back to lists of floats """
    e1SeedStrip = str(e1Seed)[1:-1] # Strip the brackets from the string
    e1SeedList = [float(i) for i in e1SeedStrip.split(',')]
    e1Seed = e1SeedList[2] # value

    e2SeedStrip = str(e2Seed)[1:-1] # Strip the brackets from the string
    e2SeedList = [float(i) for i in e2SeedStrip.split(',')]
    e2Seed = e2SeedList[2] # value

    ptSeedStrip = str(ptSeed)[1:-1] # Strip the brackets from the string
    ptSeedList = [float(i) for i in ptSeedStrip.split(',')]
    ptSeed = ptSeedList[2] # value

    gSeedStrip = str(gSeed)[1:-1] # Strip the brackets from the string
    gSeedList = [float(i) for i in gSeedStrip.split(',')]
    gSeed = gSeedList[2] # value

    v1SeedStrip = str(v1Seed)[1:-1] # Strip the brackets from the string
    v1SeedList = [float(i) for i in v1SeedStrip.split(',')]
    v1Seed = v1SeedList[2] # value

    v2SeedStrip = str(v2Seed)[1:-1] # Strip the brackets from the string
    v2SeedList = [float(i) for i in v2SeedStrip.split(',')]
    v2Seed = v2SeedList[2] # value

    rSeedStrip = str(rSeed)[1:-1] # Strip the brackets from the string
    rSeedList = [float(i) for i in rSeedStrip.split(',')]
    rSeed = rSeedList[2] # value

    # Strip the brackets from the string
    scaleFactorStrip = str(scaleFactor)[1:-1]
```

```
    scaleFactorList = [float(i) for i in scaleFactorStrip.split(',')]
    scaleFactor = scaleFactorList[2] # value

    # Strip the brackets from the string
    # MassScaleTimeIncrement
    MSTIStrip = str(MSTI)[1:-1]
    MSTIList = [float(i) for i in MSTIStrip.split(',')]
    MSTI = MSTIList[2] # value

    KnnStrip = str(KnnString)[1:-1] # Strip the brackets from the string
    KnnList = [float(i) for i in KnnStrip.split(',')]
    Knn = KnnList[2] # value

    KssStrip = str(KssString)[1:-1] # Strip the brackets from the string
    KssList = [float(i) for i in KssStrip.split(',')]
    Kss = KssList[2] # value

    KttStrip = str(KttString)[1:-1] # Strip the brackets from the string
    KttList = [float(i) for i in KttStrip.split(',')]
    Ktt = KttList[2] # value

    tnStrip = str(tnString)[1:-1] # Strip the brackets from the string
    tnList = [float(i) for i in tnStrip.split(',')]
    tn = tnList[2] # value

    tsStrip = str(tsString)[1:-1] # Strip the brackets from the string
    tsList = [float(i) for i in tsStrip.split(',')]
    ts = tsList[2] # value

    ttStrip = str(ttString)[1:-1] # Strip the brackets from the string
    ttList = [float(i) for i in ttStrip.split(',')]
    tt = ttList[2] # value

    FEStrip = str(FEString)[1:-1] # Strip the brackets from the string
    FEList = [float(i) for i in FEStrip.split(',')]
    FE = FEList[2] # value

    """ Write the FEA Code """
    Mdb()
    modelDescription = ('Measure adhesion between the retina & vitreous of ' +
                        'the human eye')
    abqModel = mdb.Model(name=modelName,
                         description=modelDescription,
                         modelType=STANDARD_EXPLICIT,
                         copyInteractions=ON,
                         copyConstraints=ON)

    if symmetry == True:
        print('FEA SYM model')
        FEA_Symmetry()
    else:
        print('FEA Non-SYM model')
        FEA()
```

1.6.4 Abaqus Extract Data Script

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

```
# -*- coding: utf-8 -*-
"""
Created on Fri Jan 29 15:36:37 2021

@author: Kiffer Creveling
Instructions:
    1) Save this script in a folder containing your ODB file
    2) Open a command window and navigate to your directory containing this
    script and your ODB file
    3) Create a .bat file
    3) Issue the command to call the script and extract data:
        abaqus python -c "import BpT; BpT.data_extract('xxxxxxx.odb')"
"""
# ***************************
from odbAccess import *
import odbAccess as oa
from sys import argv, exit
from abaqusConstants import *
from textRepr import *
import pdb
import numpy as np
import os

""" Pass arguments into this script """
# Arguments from the previous script
script =             sys.argv[0]
jobName =            sys.argv[1]
gravity =       eval(sys.argv[2]) # True/False
symmetry =      eval(sys.argv[3]) # True/False
simplified =    eval(sys.argv[4]) # True/False
DMGInitiation = eval(sys.argv[5]) # True/False
DMGEvolution =  eval(sys.argv[6]) # True/False # not used in the extraction

def openOdb(jobName):
    """
    Function used to locate the .odb given a file name

    Parameters
    ----------
    jobName : Name of the ABAQUS .odb file

    Returns
    -------
    odb : Abaqus output file
    """
    if jobName.endswith('.odb'):
        odbFile = jobName
        try:
            odb=oa.openOdb(path=odbFile, readOnly=TRUE)
            print("\nOpening the odb file... (.odb was specified)")
            return odb
        except:
            print("ERROR: Unable to open the specified odb %s.  Exiting."
```

```
                    % odbFile)
              exit(0)

    else:
        odbFile = jobName + '.odb'
        # Try opening the odb file
        try:
            odb=oa.openOdb(path=odbFile, readOnly=TRUE)
            print("\nOpening the odb file... (Searching for .odb)")
            return odb
        except:
            print("ERROR: Unable to open the specified odb %s.  Exiting."
                  % odbFile)
            exit(0)

def data_extract(jobName):
    """
    Function used to extract data from the .odb file

    Parameters
    ----------
    jobName : The name of ABAQUS .odb file

    Returns
    -------
    Two files of data used for plotting
    """

    # due to symmetry multiply the values by 2
    if symmetry == True:
        mult = 2
    else:
        mult = 1

    frames = []
    try:
        odb = openOdb(jobName)
    except:
        print(os.getcwd())
        print("Looks like there is a problem with the job name or odb file")

    theta = 30
    LoadCellDirection = [np.cos(theta*np.pi/180), np.sin(theta*np.pi/180), 0]

    """ Field Output data arrays """
    RF = []

    # vector components of the reaction force
    RFx = []
    RFy = []
    RFz = []

    U_top = [] # values to append
    U_bot = [] # values to append
    Nforc = []

    # Used to calculate bond distance
    R_bot = [] # bottom of retina
```

```python
    V_top = [] # top of vitreous
    Bond_disp = [] # Bond separation distance

    Stress = [] # Stress

    CSDMG = [] # Damage variable for cohesive surfaces in general contact.
    # Maximum stress-based damage initiation criterion for cohesive surfaces
    # in general contact.
    CSMAXSCRT = []

    CSDMG_List = [] # values
    CSMAXSCRT_List = [] # values

    CSDMG_Nodes = [] # nodes
    CSMAXSCRT_Nodes = [] # nodes

    frames = [] # List of frames
    time = [] # Time array

    # Used for reaction force simplicity further in the code
    # Temporary array used for iterating (Clears after each iteration)
    temp = []
    tempx = []
    tempy = []
    tempz = []

    """ History Output data arrays """
    Hist_Time = []
    IE = []
    KE = []

    """ Loop over the field outputs"""
    # determines the step in the abaqus odb file (typically displacement)
    step = odb.steps.keys()

    if gravity != True:
        disp_step = step[0] # Defines the step as a variable name
    else:
        # Step that includes the gravity kinetic energy settling
        disp_step = step[1]

    for frame, odbFrame in enumerate(odb.steps[disp_step].frames):
        frames.append(frame) # Construct a list of all of the frames

        """ Extract ODB fieldOutputs """
        fieldOutput = odbFrame.fieldOutputs

        # Print the time during the simulation
        print(odbFrame.description)
        time.append(odbFrame.frameValue)

        """ Abaqus Instances (Parts) """
        odbInstance = odb.rootAssembly.instances

        if simplified == False:
            # If Simp is not in the title
            # Parts
            E = odbInstance.keys(0)[0]
```

```
            G = odbInstance.keys(0)[1]
            R = odbInstance.keys(0)[2]
            T = odbInstance.keys(0)[3]
            V = odbInstance.keys(0)[4]

        elif simplified == True:
            # If simplification exists, omit the glue & tab
            E = odbInstance.keys(0)[0]
            R = odbInstance.keys(0)[1]
            V = odbInstance.keys(0)[2]
        else:
            print('Error in part definitions')

        """ Nodal displacements """
        fO_U = fieldOutput['U'] # displacements

        if simplified == False:
            # If Simp is not in the title

            # Glue
            Displacements = fO_U.getSubset(region=odbInstance[G]
                                                    .nodeSets['G_RP_SET'])
            # Loops over each node in the "SET" defined by the displacement
            for Uyi in Displacements.values:
                Uyi_vec = [Uyi.data[0], Uyi.data[1], Uyi.data[2]]
                # Find the magnitude
                # Creates a list of displacements in the "SET"
                temp.append(np.dot(Uyi_vec, LoadCellDirection))

            # Sums up the list of displacements from the "SET"
            SU = np.sum(temp)
            # Divide by the number of nodes in the set to get average
            AvgU_top = SU/len(temp)
            # Adds the total displacement to the U-array by summing across
            # each step
            U_top.append(AvgU_top)
            temp = [] # Clear the array for the next iteration in the loop

        elif simplified == True:
            # If simplification exists, omit the values

            Displacements = fO_U.getSubset(region=odbInstance[R]
                                                    .nodeSets['R_G_SET'])
            # Loops over each node in the "SET" defined by the displacement
            for Uyi in Displacements.values:
                Uyi_vec = [Uyi.data[0], Uyi.data[1], Uyi.data[2]]
                # Find the magnitude
                # Creates a list of displacements in the "SET"
                temp.append(np.dot(Uyi_vec, LoadCellDirection))

            # Sums up the list of displacements from the "SET"
            SU = np.sum(temp)
            # Divide by the number of nodes in the set to get average
            AvgU_top = SU/len(temp)
            # Adds the total displacement to the U-array by summing across
            # each step
            U_top.append(AvgU_top)
            temp = [] # Clear the array for the next iteration in the loop
```

```python

        else:
            print('Error in nodal displacements')

        """ Bond Distance """
        Displacements = fO_U.getSubset(region=odbInstance[R]
                                        .nodeSets['R_V_SET'])
        # Loops over each node in the "SET" defined by the displacement
        for Uyi in Displacements.values:
            Uyi_vec = [Uyi.data[0], Uyi.data[1], Uyi.data[2]]
            # Find the magnitude
            # Creates a list of displacements in the "SET"
            temp.append(np.dot(Uyi_vec, LoadCellDirection))

        # Sums up the list of displacements from the "SET"
        SU = np.sum(temp)
        # Divide by the number of nodes in the set to get average
        AvgR_bot = SU/len(temp)
        # Adds the total displacement to the U-array by summing across
        # each step
        R_bot.append(AvgR_bot)
        temp = [] # Clear the array for the next iteration in the loop

        Displacements = fO_U.getSubset(region=odbInstance[V]
                                        .nodeSets['V_R_SET'])
        # Loops over each node in the "SET" defined by the displacement
        for Uyi in Displacements.values:
            Uyi_vec = [Uyi.data[0], Uyi.data[1], Uyi.data[2]]
            # Find the magnitude
            # Creates a list of displacements in the "SET"
            temp.append(np.dot(Uyi_vec, LoadCellDirection))

        # Sums up the list of displacements from the "SET"
        SU = np.sum(temp)
        # Divide by the number of nodes in the set to get average
        AvgV_top = SU/len(temp)
        # Adds the total displacement to the U-array by summing across
        # each step
        V_top.append(AvgV_top)
        temp = [] # Clear the array for the next iteration in the loop

        # average difference in nodal positions between the *bonded surfaces
        Bond_disp.append(AvgR_bot - AvgV_top)

        """ Cohesive Info """
        if DMGInitiation == True:
            # fieldObject_CSMAXSCRT
            fO_CMS = fieldOutput['CSMAXSCRT General_Contact_Domain']

            # Specify only the bonded interface
            BONDED_Surface_R_CMS = fO_CMS.getSubset(region=odbInstance[R]
                                            .nodeSets['R_V_SET'])
            BONDED_Surface_V_CMS = fO_CMS.getSubset(region=odbInstance[V]
                                            .nodeSets['V_R_SET'])

            """ Contact initiation for cohesive surfaces """
            # Retina-Vitreous cohesive initiation value

```

```
            # Loop over all retina nodes
            for CSMAXSCRT_i in BONDED_Surface_R_CMS.values:
                temp.append(CSMAXSCRT_i.data) # nodal value
                if frame == 0:
                    CSMAXSCRT_Nodes.append(CSMAXSCRT_i.nodeLabel)

            # Loop over all vitreous nodes
            for CSMAXSCRT_i in BONDED_Surface_V_CMS.values:
                temp.append(CSMAXSCRT_i.data) # nodal value
                if frame == 0:
                    CSMAXSCRT_Nodes.append(CSMAXSCRT_i.nodeLabel)

            # Mean of the list of initiation values from the "SET"
            Mean_CMS = np.mean(temp)
            # append the list of nodal values
            CSMAXSCRT_List.append(temp)
            # Adds the average value to the array by summing across each step
            CSMAXSCRT.append(Mean_CMS)
            temp = [] # Clear the array for the next iteration in the loop

        else:
            print('No cohesive initiation info to update... ** Updating ' +
                  'with nans')
            CSMAXSCRT_List.append(np.nan)
            CSMAXSCRT.append(np.nan)
            CSMAXSCRT_Nodes.append(np.nan)

        if DMGEvolution == True:
            # fieldObject_CSDMG
            fO_CDG = fieldOutput['CSDMG General_Contact_Domain']

            # Specify only the bonded interface
            BONDED_Surface_R_CSDMG = fO_CDG.getSubset(region=odbInstance[R]
                                                      .nodeSets['R_V_SET'])
            BONDED_Surface_V_CSDMG = fO_CDG.getSubset(region=odbInstance[V]
                                                      .nodeSets['V_R_SET'])

            """ Contact damage for cohesive surfaces """
            # Retina-Vitreous cohesive damage value

            # Loop over all retina nodes
            for CSDMG_i in BONDED_Surface_R_CSDMG.values:
                temp.append(CSDMG_i.data)
                if frame == 0:
                    CSDMG_Nodes.append(CSDMG_i.nodeLabel)

            # Loop over all vitreous nodes
            for CSDMG_i in BONDED_Surface_V_CSDMG.values:
                temp.append(CSDMG_i.data)
                if frame == 0:
                    CSDMG_Nodes.append(CSDMG_i.nodeLabel)

            # Mean of the list of initiation values from the "SET"
            Mean_CSDMG = np.mean(temp)
            # append the list of nodal values
            CSDMG_List.append(temp)
            # Adds the average value to the array by summing across each step
            CSDMG.append(Mean_CSDMG)
```

```
            temp = [] # Clear the array for the next iteration in the loop
        else:
            print('No cohesive damage info to update... ** Updating with nans')
            CSDMG_List.append(np.nan)
            CSDMG.append(np.nan)
            CSDMG_Nodes.append(np.nan)

        """ Contact Node Lists """
        R_V_SetNodeNames = []
        V_R_SetNodeNames = []
        for i, NodeLabeli in enumerate(odbInstance[R]
                                       .nodeSets['R_V_SET'].nodes):
            R_V_SetNodeNames.append(NodeLabeli.label)

        for i, NodeLabeli in enumerate(odbInstance[V]
                                       .nodeSets['V_R_SET'].nodes):
            V_R_SetNodeNames.append(NodeLabeli.label)

        """ Reaction forces """
        fO_RF = fieldOutput['RF'] # reaction forces
        if simplified == False:
            # If Simp is not in the title

            # Glue-Retina G_RP_Set Reaction forces
            Reaction_Forces = fO_RF.getSubset(region=odbInstance[G]
                                              .nodeSets['G_RP_SET'])

        elif simplified == True:

            # Retina R_G_Set Reaction forces
            Reaction_Forces = fO_RF.getSubset(region=odbInstance[R]
                                              .nodeSets['R_G_SET'])

        else:
            print('Error in RF output')

        # Loops over each node in the "SET" defined by the reaction force
        for RFi in Reaction_Forces.values:
            RFxi = RFi.data[0]
            RFyi = RFi.data[1]
            RFzi = RFi.data[2]
            RFi_vec = [RFxi, RFyi, RFzi]

            # Find the component in the direction of the load cell
            # Creates a list of reaction forces in the "SET"
            temp.append(np.dot(RFi_vec, LoadCellDirection)*mult)
            tempx.append(RFxi*mult) # X reaction forces along the R_G_SET
            tempy.append(RFyi*mult) # Y reaction forces along the R_G_SET
            tempz.append(RFzi*mult) # Z reaction forces along the R_G_SET

        SRF = np.sum(temp) # Sums up the list of reaction forces from the "SET"
        # Adds the total reaction force to the RF-array by summing across
        # each step
        RF.append(SRF)
        temp = [] # Clear the array for the next iteration in the loop

        SRFX = np.sum(tempx)
        RFx.append(SRFX)
```

```python

        SRFY = np.sum(tempy)
        RFy.append(SRFY)

        SRFZ = np.sum(tempz)
        RFz.append(SRFZ)

        """ Nodal Forces """
        ''' Forces at the nodes of an element from both the hourglass and the
        regular deformation modes of that element (negative of the internal
        forces in the global coordinate system). The specified position in
        data and results file requests is ignored.'''

        # Searches if the repository has the value
        if fieldOutput.has_key('NFORC1') == 1:
            fO_NFORC1 = fieldOutput['NFORC1'] # Normal force 1
            fO_NFORC2 = fieldOutput['NFORC2'] # Normal force 2
            fO_NFORC3 = fieldOutput['NFORC3'] # Normal force 3

            # Retina nodal forces on the glue interface
            nodeSet_R_G_SET = odbInstance[R].nodeSets['R_G_SET']
            NF1 = fO_NFORC1.getSubset(region=nodeSet_R_G_SET)
            NF2 = fO_NFORC2.getSubset(region=nodeSet_R_G_SET)
            NF3 = fO_NFORC3.getSubset(region=nodeSet_R_G_SET)

            # Loops over each node in the "SET" defined by the reaction force
            for NFi in range(len(NF1.values)):
                NFi_vec = [NF1.values[NFi].data,
                           NF2.values[NFi].data,
                           NF3.values[NFi].data]
                NFi_veclabel = [NF1.values[NFi].nodeLabel,
                                NF1.values[NFi].data,
                                NF2.values[NFi].nodeLabel,
                                NF2.values[NFi].data,
                                NF3.values[NFi].nodeLabel,
                                NF3.values[NFi].data]
                # Find the component in the direction of the load cell
                # Creates a list of reaction forces in the "SET"
                temp.append(np.dot(NFi_vec, LoadCellDirection)*mult)

            # Sums up the list of reaction forces from the "SET"
            SNf = np.sum(temp)
            # Adds the total reaction force to the RF-array by summing across
            # each step (negative indicates the direction, which is opposite
            # of tension when -1)
            Nforc.append(SNf*-1)
            temp = [] # Clear the array for the next iteration in the loop
        else:
            Nforc.append(0)
            print('No NFORC... ** Updating with 0')

        """ Stress """
        fO_S = fieldOutput['S'] # stress
        # Glue-Retina set-forces
        # Loops over each node in the "SET" defined by the reaction force
        for Si in fO_S.values:
            stress_vec = [Si.data[0], Si.data[1], Si.data[2]]
            # Append the component of stress in the load cell direction
```

```
            Stress.append(np.dot(stress_vec, LoadCellDirection))

    # In[History Output]
    """ Loop over the history outputs"""
    # List all of the items in the dictionary
    # odb.steps[disp_step].historyRegions.keys()
    odbHistoryRegion = odb.steps[disp_step].historyRegions
    odbHistAssem = 'Assembly ASSEMBLY'
    Assembly = odbHistoryRegion[odbHistAssem]

    # Energy output
    ALLIE_KE = Assembly.historyOutputs.keys()[0]
    Hist_ELEM = Assembly.historyOutputs.keys()[1]
    Whole_Model_Energy = Assembly.historyOutputs
    Internal_Energy = Whole_Model_Energy.keys()[0] # Internal energy
    Kinetic_Energy = Whole_Model_Energy.keys()[1] # Kintic energy
    for i,j in enumerate(Whole_Model_Energy[Internal_Energy].data):
        Hist_Time.append(j[0]) # History Output Time Array
        IE.append(j[1]) # Internal Energy
        # Kinetic Energy
        KE.append(Whole_Model_Energy[Kinetic_Energy].data[i][1])

    # Glue Reference point
    if simplified == False:
        # If Simp is not in the title

        odbHist_gRP = odbHistoryRegion.keys()[1]
        gRP_Hist = odbHistoryRegion[odbHist_gRP]
        gRP_Hist = gRP_Hist.historyOutputs
        gRP_HistRF1 = gRP_Hist.keys()[0]
        gRP_HistRF2 = gRP_Hist.keys()[1]
        gRP_HistRF3 = gRP_Hist.keys()[2]
        gRP_HistU1 = gRP_Hist.keys()[6]
        gRP_HistU2 = gRP_Hist.keys()[7]
        gRP_HistU3 = gRP_Hist.keys()[8]

    elif simplified == True:
        # If simplification, omit the tab and glue
        print('Simplification')
    else:
        print('Error in simplification')

    # In[Print Field Outputs]
    """ Specify folder name where the files go..."""
    folderName = jobName
    folder_sub_directory = 'Output'

    """ Print the odbFieldOutput Data """
    print("\nWriting out the load data...")
    filename = os.path.join(folderName, folder_sub_directory,
                            'output_Field_' + jobName + '.txt')
    outfile = open(filename,'w')

    Header = [] # Header information for the dataframe
    Header.append('frame')
    Header.append('Time [s]')
    Header.append('Reaction force dotted in y direction [N]')
    Header.append('Reaction force X [N]')
```

```
    Header.append('Reaction force Y [N]')
    Header.append('Reaction force Z [N]')
    Header.append('Sum Nodal Force [N]')
    Header.append('Glue Displacements [m]')
    Header.append('Bond Displacements [m]')
    Header.append('Stress [Pa]')
    Header.append('AVG CSMAXSCRT')
    Header.append('AVG CSDMG')

    lineWrite = '\t'.join(str(item) for item in Header)
    outfile.write(lineWrite)

    for i in frames:

        lineNums = []
        lineNums.append(time[i])
        lineNums.append(RF[i])
        lineNums.append(RFx[i])
        lineNums.append(RFy[i])
        lineNums.append(RFz[i])
        lineNums.append(Nforc[i])
        lineNums.append(U_top[i])
        lineNums.append(Bond_disp[i])
        lineNums.append(Stress[i])
        lineNums.append(CSMAXSCRT[i])
        lineNums.append(CSDMG[i])

        # format the list to have a float with twenty decimal places
        # Add floats
        formatted_list = ['{:.20f}'.format(item) for item in lineNums]
        line = '\n' + '{}\t'.format(i) + '\t'.join(str(item)
                                                   for item in formatted_list)
        outfile.write(line)

    outfile.close()

    print("\nDone!")
    print("\nThe output file will be named '{}".format(filename) + "'")
    print("\nIt will be in the same working directory as your Abaqus" +
          " model\n")

    # In[Print History Output]
    """ Print the odbHistoryOutput Data """
    print("\nWriting out the History Output data...")
    filename = os.path.join(folderName, 'Output', 'output_History_' +
                            jobName + '.txt')
    outfile = open(filename, 'w')

    Header = []
    Header.append('frame')
    Header.append('Time [s]')
    Header.append('Internal Energy [J]')
    Header.append('Kinetic Energy [J]')
    lineWrite = '\t'.join(str(item) for item in Header)
    outfile.write(lineWrite)

    for i, j in enumerate(Hist_Time):
        line = []
```

```python
        line.append('{}'.format(i)) # Integer for frame number
        line.append('{:.10f}'.format(j))
        line.append('{:.30f}'.format(IE[i]))
        line.append('{:.30f}'.format(KE[i]))
        lineWrite = '\n' + '\t'.join(str(item) for item in line)
        outfile.write(lineWrite)

    outfile.close()

    print("\nDone!")
    print("\nThe output file will be named '{}".format(filename) + "'")
    print("\nIt will be in the same working directory as your Abaqus" +
          " model\n")
    # In[DMG Criteria]
    if DMGInitiation == True:
        """ Print the CSMAXSCRT Data """
        print("\nWriting out the Field Output CSMAXSCRT data...")
        filename = os.path.join(folderName, 'Output', 'CSMAXSCRT_' +
                                jobName + '.txt')
        outfile = open(filename, 'w')
        outfile.write('Time (s)\t' + '\t'.join(str(item)
                                               for item in CSMAXSCRT_Nodes))
        for i, j in enumerate(CSMAXSCRT_List):
            outfile.write('\n')
            tempList = [time[i]]
            for k in list(j):
                tempList.append(k)
            outfile.write('\t'.join(str(item) for item in tempList))
        outfile.close()
        print("\nDone!")
        print("\nThe output file will be named '{}".format(filename) + "'")
        print("\nIt will be in the same working directory as your Abaqus" +
              " model\n")
    if DMGEvolution == True:
        """ Print the CSDMG Data """
        print("\nWriting out the Field Output CSDMG data...")
        filename = os.path.join(folderName, 'Output', 'CSDMG_' +
                                jobName + '.txt')
        outfile = open(filename, 'w')
        outfile.write('Time (s)\t' + '\t'.join(str(item)
                                               for item in CSDMG_Nodes))
        for i, j in enumerate(CSDMG_List):
            outfile.write('\n')
            tempList = [time[i]]
            for k in list(j):
                tempList.append(k)
            outfile.write('\t'.join(str(item) for item in tempList))
        outfile.close()
        print("\nDone!")
        print("\nThe output file will be named '{}".format(filename) + "'")
        print("\nIt will be in the same working directory as your Abaqus" +
              " model\n")
    return

# Run the function
data_extract(jobName)
```

1.6.5 Plotting Script

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

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Jan 29 23:49:03 2021

@author: Kiffer Creveling
python3

"""
# Packages & path folder
#from sys import argv, exit
#sys.path.append(r'F:\Abaqus Working Directory')
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import cm
import matplotlib.patheffects as pe
import numpy as np
import os
import os.path
import sys
import pdb
plt.rcParams['figure.figsize'] = [16, 9]

def plot_Field_Output(fileName, dataDirectory, dataCompare,
                      DMGInitiation, DMGEvolution):

    """ Field Output Data """
    df = pd.read_csv(os.path.join(dataDirectory, fileName),
                     sep="\t", header=0)

    Header = [] # Header information for the dataframe
    Header.append('Frame')
    Header.append('Time')
    Header.append('RF_y_dot')
    Header.append('RFx')
    Header.append('RFy')
    Header.append('RFz')
    Header.append('Nodal_Force')
    Header.append('Tab_Displacement')
    Header.append('Bond_Displacement')
    Header.append('Stress')
    Header.append('AVG_CSMAXSCRT')
    Header.append('AVG_CSDMG')

    df.columns = Header

    t = df.Time
    RF = df.RF_y_dot*1e3 # Convert from N to mN
    NF = df.Nodal_Force*1e3 # Convert from N to mN
    TD = df.Tab_Displacement*1e3 # convert from N to mN
    B = df.Bond_Displacement*1e3 # convert from N to mN
    S = df.Stress
    AVG_CSMAXSCRT = df.AVG_CSMAXSCRT
    AVG_CSDMG = df.AVG_CSDMG
```

```
    (figureName, ext) = os.path.splitext(fileName) # Split the file extension

    """ Read in the csv file """
    dfValsn = pd.read_csv(dataCompare, sep="\t", nrows=22, header=None,
                          names=['Var', 'Attribute'])

    """ File Attributes """
    HID =             dfValsn['Attribute'][0]
    HAGE =            dfValsn['Attribute'][1]
    HG =              dfValsn['Attribute'][2]
    HLR =             dfValsn['Attribute'][3]
    HR =              dfValsn['Attribute'][4]
    HSSi =      float(dfValsn['Attribute'][12])
    HSSf =      float(dfValsn['Attribute'][13])
    HTMax =     float(dfValsn['Attribute'][14])
    HDispMax = float(dfValsn['Attribute'][15])
    HFMax =     float(dfValsn['Attribute'][16]) # (mN)
    HFSS =      float(dfValsn['Attribute'][17])
    # (mN/m) slope from 20 seconds prior to max force value
    HSlope20 = float(dfValsn['Attribute'][20])

    dfn = pd.read_csv(os.path.join(dataCompare), sep="\t", header=30)
    dfn.columns = ['Time', 'Extension', 'Force']
    dfn_time = dfn.Time
    dfn_extension = dfn.Extension
    dfn_force = dfn.Force*1e3 # convert from N to mN

    # SS Array
    ssTimeArray = np.array([HSSi, HSSf])
    ssValArray = np.array([HFSS, HFSS])

    # Max peel force displacement at max and steady state
    dfn_max_Disp = dfn_extension[dfn_time == HTMax]
    # .flatten()
    dfn_ss_Disp = np.array([dfn_extension[dfn_time == HSSi].values[0],
                            dfn_extension[dfn_time == HSSf].values[0]])

    # Plot the data trace to compare the simulated results with the force
    # displacement curves
    plt.plot(dfn_extension, dfn_force, '-', color='r', linewidth=1,
             markersize=2, label = '{}, Age: {}'.format(HID, HAGE))
    if str(HFMax) == 'nan' and str(HSSi) == 'nan':
        print('No max or steady state')
        pass

    if str(HFMax) != 'nan':
        plt.plot(dfn_max_Disp, HFMax, '.', color='k', linewidth=1,
                 markersize=20, label='Max Peel - {:.4f} (mN)'.format(HFMax),
                 path_effects=[pe.Stroke(linewidth=4, foreground='k'),
                               pe.Normal()])

    if str(HSSi) != 'nan':
        plt.plot(dfn_ss_Disp, ssValArray, '-', color='c', linewidth=3,
                 markersize=2, label='Steady State - {:.4f} (mN)'.format(HFSS),
                 path_effects=[pe.Stroke(linewidth=5, foreground='k'),
                               pe.Normal()])

```

```python
    """ Plots """
    ################## Plot Data #########################
    plt.plot(TD, RF, '-', color='blue', linewidth=2, markersize=2,
             label = r'Simulated Reaction force $\Sigma F_{Retina}$')
    plt.xlabel('Displacement (mm)', fontsize=18)
    plt.ylabel('Force (mN)', fontsize=18)
    plt.title('Vitreous', fontsize=20)
    plt.grid()
    plt.legend(loc = 'best', fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory, 'Figures/' +
                             figureName + '_RF_vs_Disp.pdf'),
                dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    # Plot the data trace to compare the simulated results
    plt.plot(dfn_time, dfn_force, '-', color='r', linewidth=1, markersize=2,
             label = '{}, Age: {}'.format(HID, HAGE))
    if str(HFMax) == 'nan' and str(HSSi) == 'nan':
        print('No max or steady state')
        pass

    if str(HFMax) != 'nan':
        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()])

    if str(HSSi) != 'nan':
        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()])

    """ Plots """
    ################## Plot Data #########################
    plt.plot(t, RF, '-', color='blue', linewidth=2, markersize=2,
             label = r'Simulated Reaction force $\Sigma F_{Retina}$')
    plt.xlabel('Time (sec)', fontsize=18)
    plt.ylabel('Force (mN)', fontsize=18)
    plt.title('Vitreous', fontsize=20)
    plt.grid()
    plt.legend(loc = 'best', fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory, 'Figures/' +
                             figureName + '_RF_vs_t.pdf'),
                dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    ################## Plot Data #########################
    plt.plot(t, NF, '-', color='blue', linewidth=2, markersize=2,
             label = 'Reaction force NForce')
    plt.xlabel('Time (sec)', fontsize=18)
    plt.ylabel('Force (N)', fontsize=18)
    plt.title('Vitreous', fontsize=20)
    plt.grid()
    plt.legend(loc = 'best', fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory, 'Figures/' +
                             figureName + '_NF_vs_t.pdf'),
                dpi=300, bbox_inches='tight') # Save figure
```

```
    plt.close()

    ################## Plot Data #########################
    plt.plot(t, B, '-', color='blue', linewidth=2, markersize=2,
             label = 'Bond - Disp')
    plt.xlabel('Time (sec)', fontsize=18)
    plt.ylabel('Bond Disp (mm)', fontsize=18)
    plt.title('VR Interface', fontsize=20)
    plt.grid()
    plt.legend(loc = 'best', fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory, 'Figures/' +
                             figureName + '_B_vs_t.pdf'),
                dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    ################## Plot Data #########################
    plt.plot(t, B,'-', color='blue', linewidth=2, markersize=2,
             label = 'Bond - Disp')
    plt.plot(t, TD, '-.', color='red', linewidth=2, markersize=2,
             label = 'Top - Disp')
    plt.xlabel('Time (sec)', fontsize=18)
    plt.ylabel('Bond Disp (mm)', fontsize=18)
    plt.title('Vitreous', fontsize=20)
    plt.grid()
    plt.legend(loc = 'best', fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory, 'Figures/' +
                             figureName + '_disp_vs_t.pdf'),
                dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    ################## Plot Data #########################
    plt.plot(t, S, '-', color='blue', linewidth=2, markersize=2,
             label = 'Stress')
    plt.xlabel('Time (sec)', fontsize=18)
    plt.ylabel('Stress (Pa)', fontsize=18)
    plt.title('Vitreous', fontsize=20)
    plt.grid()
    plt.legend(loc = 'best', fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory, 'Figures/' +
                             figureName + '_Stress_vs_t.pdf'),
                dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    if DMGInitiation == True:
        ################## Plot Data #########################
        plt.plot(t, AVG_CSMAXSCRT, '-', color='blue', linewidth=2,
                 markersize=2, label = r'CSMAXSCRT$_{AVG}$')
        plt.xlabel('Time (sec)', fontsize=18)
        plt.ylabel('Maximum Displacement Criterion Value', fontsize=18)
        plt.title('Vitreous', fontsize=20)
        plt.grid()
        plt.legend(loc = 'best', fontsize = 'medium')
        plt.savefig(os.path.join(dataDirectory, 'Figures/' +
                                 figureName + '_AVG_CSMAXSCRT_vs_t.pdf'),
                    dpi=300, bbox_inches='tight') # Save figure
        plt.close()

    if DMGEvolution == True:
```

```python
        ################## Plot Data #########################
        plt.plot(t, AVG_CSDMG, '-', color='blue', linewidth=2,
                 markersize=2, label = r'CSDMG$_{AVG}$')
        plt.xlabel('Time (sec)', fontsize=18)
        plt.ylabel('Maximum Damage Value', fontsize=18)
        plt.title('Vitreous', fontsize=20)
        plt.grid()
        plt.legend(loc = 'best', fontsize = 'medium')
        plt.savefig(os.path.join(dataDirectory, 'Figures/' +
                                 figureName + '_AVG_CSDMG_vs_t.pdf'),
                    dpi=300, bbox_inches='tight') # Save figure
        plt.close()


def plot_History_Output(fileName, dataDirectory):
    """ History Output Data """
    df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)
    df.columns = ["Frame", "Time", "Internal_Energy", "Kinetic_Energy"]

    t_h = df.Time
    IE = df.Internal_Energy
    KE = df.Kinetic_Energy

    (figureName, ext) = os.path.splitext(fileName) # Split the file extension

    """ Plots History Outputs """
    ################## Plot Data #########################
    plt.plot(t_h, IE, '-', color='blue', linewidth=2, markersize=2,
             label = 'Internal Energy')
    plt.plot(t_h, KE, '-', color='red', linewidth=2, markersize=2,
             label = 'Kinetic Energy')
    plt.xlabel('Time (sec)', fontsize=18)
    plt.ylabel('Energy (J)', fontsize=18)
    plt.title('Energy', fontsize=20)
    plt.grid()
    plt.legend(loc = 'best', fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory, 'Figures/' +
                             figureName + '_Energy.pdf'),
                dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    ################## Plot Data #########################
    plt.semilogy(t_h, KE/IE, '-', color='blue', linewidth=2,
                 markersize=2, label = r'Ratio $\frac{KE}{IE}$')
    plt.semilogy(t_h, 0.1*np.ones(len(t_h)), '-', color='red',
                 linewidth=2, markersize=2, label = '10%')
    plt.xlabel('Time (sec)', fontsize=18)
    plt.ylabel('Ratio of KE to IE', fontsize=18)
    plt.title('Energy ratio', fontsize=20)
    plt.grid()
    plt.legend(loc = 'best', fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory, 'Figures/' +
                             figureName + '_Ratio_KE_IE.pdf'),
                dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    print("Plots will be in the figures folder")

```

```python
def plot_CohesiveCSMAXSCRT_Output(fileName, dataDirectory):
    """ CohesiveCSMAXSCRT Output Data """
    df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)

    t = df['Time (s)']

    """
    The incoming data has both the Retina and Vitreous nodes associated
    with it.  We need to split them apart and create plots for each data
    set separately
    """

    # Filter data by the "name" of the node that begins with 1 i.e. '1000002'
    # and create a new dataframe
    dfR = df.loc[:, df.columns.str.startswith('4')] # Retina
    dfV = df.loc[:, df.columns.str.startswith('5')] # Vitreous

    """ Retina """

    # determine the length of the number of bonded nodes
    # linspace from 0 to 1 by the number of nodes for the y-position
    # Loop over the number of bonded nodes and plot the y-th
    # vs time with the color of the bond load on a single plot

    fig1, ax1 = plt.subplots()
    nRows  = np.shape(dfR)[0]
    nCols = np.shape(dfR)[1]
    y = np.linspace(0, 1, nCols)
    count = 0
    for (colName, colData) in dfR.iteritems():
        if colName.find('Time') == -1:
            """ Plots CSMAXSCRT Outputs """
            ################# Plot Data ########################
            sc = ax1.scatter(t, np.ones(nRows)*y[count], c=colData,
                             cmap=cm.cool, s=5, edgecolors='none',
                             vmin=0, vmax=1)
            count += 1 # update the counter
        else:
            continue

    # plt.gray() # turns image to grayscale
    plt.colorbar(sc)
    ax1.set_xlabel('Time (sec)', fontsize=18)
    ax1.set_ylabel('Cohesive CSMAXSCRT', fontsize=18)
    ax1.set_title('Retina CSMAXSCRT (Color indicates status)', fontsize=20)
    (figureName, ext) = os.path.splitext(fileName) # Split the file extension
    fig1.savefig(os.path.join(dataDirectory, 'Figures/' +
                              figureName + '_CSMAXSCRT_vs_t_Retina.pdf'),
                 dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    """ Vitreous """

    # determine the length of the number of bonded nodes
    # linspace from 0 to 1 by the number of nodes for the y-position
    # Loop over the number of bonded nodes and plot the y-th
    # position vs time with the color of the bond load on a single plot

```

```
    fig1, ax1 = plt.subplots()
    nRows  = np.shape(dfV)[0]
    nCols = np.shape(dfV)[1] #  - 1 # subtract the time column
    y = np.linspace(0, 1, nCols)
    count = 0
    for (colName, colData) in dfV.iteritems():
        if colName.find('Time') == -1:
            """ Plots CSMAXSCRT Outputs """
            ################# Plot Data ########################
            sc = ax1.scatter(t, np.ones(nRows)*y[count], c=colData,
                             cmap=cm.cool, s=5, edgecolors='none',
                             vmin=0, vmax=1)
            count += 1 # update the counter
        else:
            continue

    # plt.gray() # turns image to grayscale
    plt.colorbar(sc)
    ax1.set_xlabel('Time (sec)', fontsize=18)
    ax1.set_ylabel('Cohesive CSMAXSCRT', fontsize=18)
    ax1.set_title('Vitreous CSMAXSCRT (Color indicates status)', fontsize=20)
    (figureName, ext) = os.path.splitext(fileName) # Split the file extension
    fig1.savefig(os.path.join(dataDirectory, 'Figures/' +
                              figureName + '_CSMAXSCRT_vs_t_Vitreous.pdf'),
                 dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    print("Plots will be in the figures folder")

def plot_CohesiveCSDMG_Output(fileName, dataDirectory):
    """ CohesiveCSDMG Output Data """
    df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)

    t = df['Time (s)']

    """
    The incoming data has both the Retina and Vitreous nodes associated
    with it.  We need to split them apart and create plots for each data
    set separately
    """

    # Filter data by the "name" of the node that begins with 1 i.e. '1000002'
    dfR = df.loc[:, df.columns.str.startswith('4')] # Retina
    dfV = df.loc[:, df.columns.str.startswith('5')] # Vitreous

    """ Retina """

    # determine the length of the number of bonded nodes
    # linspace from 0 to 1 by the number of nodes for the y-position
    # Loop over the number of bonded nodes and plot the y-th
    # position vs time with the color of the bond load on a single plot

    fig1, ax1 = plt.subplots()
    nRows  = np.shape(dfR)[0]
    nCols = np.shape(dfR)[1] #  - 1 # subtract the time column
    y = np.linspace(0, 1, nCols)
    count = 0
    for (colName, colData) in dfR.iteritems():
```

```
        if colName.find('Time') == -1:
            """ Plots CohesiveCSDMG Outputs """
            ################# Plot Data ########################
            sc = ax1.scatter(t, np.ones(nRows)*y[count], c=colData,
                             cmap=cm.cool, s=5, edgecolors='none',
                             vmin=0, vmax=1)
            count += 1 # update the counter
        else:
            continue

    # plt.gray() # turns image to grayscale
    plt.colorbar(sc)
    ax1.set_xlabel('Time (sec)', fontsize=18)
    ax1.set_ylabel('Cohesive CSDMG', fontsize=18)
    ax1.set_title('Retina CSDMG (Color indicates status)', fontsize=20)
    (figureName, ext) = os.path.splitext(fileName) # Split the file extension
    fig1.savefig(os.path.join(dataDirectory, 'Figures/' +
                              figureName + '_CSDMG_vs_t_Retina.pdf'),
                 dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    """ Vitreous """

    # determine the length of the number of bonded nodes
    # linspace from 0 to 1 by the number of nodes for the y-position
    # Loop over the number of bonded nodes and plot the y-th
    # position vs time with the color of the bond load on a single plot

    fig1, ax1 = plt.subplots()
    nRows  = np.shape(dfV)[0]
    nCols = np.shape(dfV)[1] #  - 1 # subtract the time column
    y = np.linspace(0, 1, nCols)
    count = 0
    for (colName, colData) in dfV.iteritems():
        if colName.find('Time') == -1:
            """ Plots CohesiveCSDMG Outputs """
            ################# Plot Data ########################
            sc = ax1.scatter(t, np.ones(nRows)*y[count], c=colData,
                             cmap=cm.cool, s=5, edgecolors='none',
                             vmin=0, vmax=1)
            count += 1 # update the counter
        else:
            continue

    # plt.gray() # turns image to grayscale
    plt.colorbar(sc)
    ax1.set_xlabel('Time (sec)', fontsize=18)
    ax1.set_ylabel('Cohesive CSDMG', fontsize=18)
    ax1.set_title('Vitreous CSDMG (Color indicates status)', fontsize=20)
    (figureName, ext) = os.path.splitext(fileName) # Split the file extension
    fig1.savefig(os.path.join(dataDirectory, 'Figures/' +
                              figureName + '_CSDMG_vs_t_Vitreous.pdf'),
                 dpi=300, bbox_inches='tight') # Save figure
    plt.close()

    print("Plots will be in the figures folder")

def PlotAbqData(fileName, dataDirectory, dataCompare,
```

```
                DMGInitiation, DMGEvolution):

    # """ Change directory to correct path """
    # filePath = os.getcwd()
    # data_directory = os.path.join(filePath, jobName)
    # figures_directory = os.path.join(filePath, jobName, 'Figures')
    # if not os.path.exists(figures_directory):
    #     os.makedirs(figures_directory)

    """ Call both functions to plot Field/History data """
    field_files = [f for f in os.listdir(dataDirectory)
                   if os.path.isfile(os.path.join(dataDirectory, f))
                   and f.startswith('output_Field')]
    for fname in field_files:
        plot_Field_Output(fname, dataDirectory, dataCompare,
                          DMGInitiation, DMGEvolution)

    history_files = [f for f in os.listdir(dataDirectory)
                     if os.path.isfile(os.path.join(dataDirectory, f))
                     and f.startswith('output_History')]
    for hname in history_files:
        plot_History_Output(hname, dataDirectory)

    if DMGInitiation == True:
        CSMAXSCRT_files = [f for f in os.listdir(dataDirectory)
                           if os.path.isfile(os.path.join(dataDirectory, f))
                           and f.startswith('CSMAXSCRT')]
        for CSMAXSCRTname in CSMAXSCRT_files:
            plot_CohesiveCSMAXSCRT_Output(CSMAXSCRTname, dataDirectory)

    if DMGEvolution == True:
        CSDMG_files = [f for f in os.listdir(dataDirectory)
                       if os.path.isfile(os.path.join(dataDirectory, f))
                       and f.startswith('CSDMG')]
        for CSDMGname in CSDMG_files:
            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.* </>

```
# -*- coding: utf-8 -*-
"""
Created on Sat Jan 30 01:06:55 2021

@author: Kiffer Creveling

"""

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import os
import pdb
```

```
plt.rcParams['figure.figsize'] = [16, 9]

def MaxCohesiveCSMAXSCRT_Output(fileName, dataDirectory, maxForceTime,
                                dataCompare):

    """ Read in the csv file """
    dfValsn = pd.read_csv(os.path.join(dataCompare), sep="\t", nrows=29,
                          header=None, names=['Var', 'Attribute'])

    """ File Attributes """
    HID =            dfValsn['Attribute'][0]
    HAGE =           dfValsn['Attribute'][1]
    HG =             dfValsn['Attribute'][2]
    HLR =            dfValsn['Attribute'][3]
    HR =             dfValsn['Attribute'][4]
    HSSi =     float(dfValsn['Attribute'][12])
    HSSf =     float(dfValsn['Attribute'][13])
    HTMax =    float(dfValsn['Attribute'][14])
    HDispMax = float(dfValsn['Attribute'][15])
    HFMax =    float(dfValsn['Attribute'][16]) # (mN)
    HFSS =     float(dfValsn['Attribute'][17])
    # (mN/m) slope from 20 seconds prior to max force value
    HSlope20 = float(dfValsn['Attribute'][20])

    dfn = pd.read_csv(os.path.join(dataCompare), sep="\t", header=30)
    dfn.columns = ['Time', 'Extension', 'Force']
    dfn_time = dfn.Time
    dfn_extension = dfn.Extension # mm
    dfn_force = dfn.Force*1e3 # N ---> mN

    """ CohesiveCSMAXSCRT Output Data """
    df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)

    t = df['Time (s)']

    """ The incoming data has both the Retina and Vitreous nodes associated
    with it.  We need to split them apart and create plots for each data set
    separately """

    # Filter data by the "name" of the node that begins with 1 i.e. '1000002'
    # and create a new dataframe
    dfR = df.loc[:, df.columns.str.startswith('4')] # Retina
    dfV = df.loc[:, df.columns.str.startswith('5')] # Vitreous

    # Turns out this is unnecessary as the time value interferes with the max
    # # Add time to dfR & dfV
    # dfR.insert(loc=0, column='Time', value=t)
    # dfV.insert(loc=0, column='Time', value=t)

    # Max value at specific time
    specificTime = maxForceTime

    # Value in the data frame that is closest to the specified time
    actualTime = min(t, key=lambda x:abs(x - specificTime))

    index = t[t == actualTime].index.values[0] # index

    dfRSelect = dfR[t < actualTime] # Selection of the data frame
```

```
    dfVSelect = dfV[t < actualTime] # Selection of the data frameSelect

    # determine max value in the dataframe
    retinaMaxUCRT = dfRSelect.max().max()
    vitreousMaxUCRT = dfVSelect.max().max()

    # return values
    return retinaMaxUCRT, vitreousMaxUCRT


def CSMAXSCRTAbqData(fileName, dataDirectory, maxForceTime, dataCompare):

    # """ Change directory to correct path """
    # filePath = os.getcwd()
    # data_directory = os.path.join(filePath,jobName)
    # figures_directory = os.path.join(filePath,jobName,'Figures')
    # if not os.path.exists(figures_directory):
    #     os.makedirs(figures_directory)

    """ Call both functions to plot Field/History data """
    global maxCohesiveCSMUCRT

    CSMAXSCRT_files = [f for f in os.listdir(dataDirectory)
                       if os.path.isfile(os.path.join(dataDirectory, f))
                       and f.startswith('CSMAXSCRT')]
    for CSMAXSCRTname in CSMAXSCRT_files:
        maxCohesiveCSMUCRT = MaxCohesiveCSMAXSCRT_Output(CSMAXSCRTname,
                                                         dataDirectory,
                                                         maxForceTime,
                                                         dataCompare)
    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.* </>

```
# -*- coding: utf-8 -*-
"""
Created on Sat Nov  7 17:27:47 2020

@author: Kiffer2

"""
import numpy as np
import pandas as pd
from scipy import interpolate
import matplotlib.pyplot as plt
from matplotlib.pyplot import cm
import matplotlib.patheffects as pe
import os
import os.path
import sys
import pdb

```

```python
def Least_Squares(x, y):
    """
    Calculate the slope and y-intercept using matrix math
    x & y are the coordinates of points

    parameters (X,Y) Data

    Returns:
        Curve fit data and parameters m*x + b, R squared value
    """
    Z = np.ones((len(x),2))
    Z[:,1] = x
    # Calculate the matrix inverse for the constants of the regression
    A = np.dot(np.linalg.inv(np.dot(Z.T,Z)),(np.dot(Z.T,y)))
    linFit = x*A[1] + A[0]

    # Stats
    SS_tot = np.sum((y - np.mean(y))**2)
    SS_res = np.sum((y - linFit)**2)
    Rsqd = 1 - SS_res/SS_tot

    return linFit, A, Rsqd


def residualFcn(fileName, dataDirectory, maxForceTime, dataCompare, objErr,
                slopeFlag, maxForceFlag, ssForceFlag, timeBeforePeak):
    """
    Parameters
    ----------
    fileName : Output txt file with the odb data
    dataDirectory : Location of the output file

    Returns
    -------
    Maximum force from the txt file
    """

    # In[Simulated data]
    df = pd.read_csv(os.path.join(dataDirectory, fileName), sep="\t", header=0)

    Header = [] # Header information for the dataframe
    Header.append('Frame') #                    h1
    Header.append('Time') #                     h2
    Header.append('RF_y_dot') #                 h3
    Header.append('RFx') #                      h4
    Header.append('RFy') #                      h5
    Header.append('RFz') #                      h6
    Header.append('Nodal_Force') #              h7
    Header.append('Tab_Displacement') #         h8
    Header.append('Bond_Displacement') #        h9
    Header.append('Stress') #                   h10
    Header.append('AVG_CSMAXSCRT') #            h11
    Header.append('AVG_CSDMG') #                h12
    df.columns = Header

    tt = df.Time
    RF = df.RF_y_dot*1000 # N to mN
    dn = df.Tab_Displacement*1000 # m
```

```
    # maybe try to output the maximum force at a specific time
    specificTime = maxForceTime
    actualTime = min(df['Time'], key=lambda x:abs(x - specificTime))
    force_at_time = RF[df['Time'] == actualTime].values[0]

    # In[Experimental data]
    """ Read in the csv file """
    dfValsn = pd.read_csv(os.path.join(dataCompare), sep="\t", nrows=29,
                          header=None, names=['Var', 'Attribute'])

    """ File Attributes """
    HID =            dfValsn['Attribute'][0]
    HAGE =           dfValsn['Attribute'][1]
    HG =             dfValsn['Attribute'][2]
    HLR =            dfValsn['Attribute'][3]
    HR =             dfValsn['Attribute'][4]
    HSSi =     float(dfValsn['Attribute'][12])
    HSSf =     float(dfValsn['Attribute'][13])
    HTMax =    float(dfValsn['Attribute'][14])
    HDispMax = float(dfValsn['Attribute'][15])
    HFMax =    float(dfValsn['Attribute'][16]) # (mN)
    HFSS =     float(dfValsn['Attribute'][17]) # (mN)
    # slope from 20 seconds prior to max force value
    HSlope20 = float(dfValsn['Attribute'][20]) # (mN/m)

    dfn = pd.read_csv(os.path.join(dataCompare), sep="\t", header=30)
    dfn.columns = ['Time', 'Extension', 'Force']
    dfn_time = dfn.Time
    dfn_extension = dfn.Extension # mm
    dfn_force = dfn.Force*1e3 # N ---> mN

    # if fileName.find('sym') >= 0:
    #     # divide all data trace values by 2
    #     dfn_force = dfn_force/2
    #     HFMax = HFMax/2
    #     HFSS = HFSS/2

    # SS Array
    ssTimeArray = np.array([HSSi, HSSf])
    ssValArray = np.array([HFSS, HFSS])

    # In[Experimental data isolate linear region up to peak]

    # slope calculation for 20 seconds prior to the max peel force
    # (Experimental Data)
    maxIndex = dfn_time[dfn_time == HTMax].index.values[0]

    # Convert to data array length
    timeBeforePeak = timeBeforePeak*10

    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec) to location of max
    ↪  force
    x_n = dfn_extension[maxIndex - timeBeforePeak:maxIndex]
    y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
    # Perform least squares
    curveFit_n, Params_n, R_Squared_n = Least_Squares(x_n, y)

```

```
    # Shift extension data so that the linear region is extrapolated
    # through the origin
    shift_disp = abs(Params_n[0]/Params_n[1])
    if Params_n[0] > 0:
        dfn_extension_shift = dfn_extension + shift_disp

        if min(dfn_extension_shift) > 0:
            # Add zero to prevent mishaps with interpolation
            dfn_extension_shift = [0] + dfn_extension_shift
    else:
        dfn_extension_shift = dfn_extension - shift_disp

    # Now that the data has been shifted, recalculate the linear regression
    # using the reduced data set
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec) to location of max
    ↪  force
    x_n = dfn_extension_shift[maxIndex - timeBeforePeak:maxIndex]
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec) to location of max
    ↪  force
    y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
    # Perform least squares
    curveFit_n, Params_n, R_Squared_n = Least_Squares(x_n,y)

    # Slope of the curve up to the max force !!!(from the simulated data)!!!
    # find the closest simulated displacement to the experimental
    # max displacement
    # adjustDisp = min(dn, key=lambda x:abs(x - dfn_extension[maxIndex]))
    # index = RF[dn == adjustDisp].index.values[0] # index determination
    # Index where the max reaction force is in the array
    simMaxIndex = RF.idxmax()
    simMaxForce = RF.max() # maximum simulated force value
    simMaxDisp = dn[RF == simMaxForce] # displacement at the max force value

    # If the max index is the second data point add one to it (Difficulty in
    # selecting the pandas series value) to select the fist two values in the
    # pandas array it needs to be RF[0:2] instead of RF[0:1] but the index
    # value of the max force is 1.  Try to fix this issue
    if simMaxIndex == 1:
        simMaxIndex += 1

    x = dn[0:simMaxIndex] # Array from 0 to location of max force/n
    y = RF[0:simMaxIndex] # Array from 0 to location of max force/n
    # Perform least squares
    curveFit, Params, R_Squared = Least_Squares(x,y)

    # Updated force at specific max disp with adjusted value (Simulated data)
    specificTime = maxForceTime
    actualDisp = min(dn, key=lambda x:abs(x - dfn_extension_shift[maxIndex]))
    force_at_Disp = RF[dn == actualDisp].values[0]

    # Max peel force displacement at max and steady state
    dfn_max_Disp = dfn_extension_shift[dfn_time == HTMax]
    dfn_ss_Disp = [dfn_extension_shift[dfn_time == HSSi].values[0],
                   dfn_extension_shift[dfn_time == HSSf].values[0]] # flatten()

    """ Simulated Steady State calculation """
    if simMaxIndex == len(RF):
        simMaxGreaterIndex = len(RF) - 1
```

```python
    else:
        # return the mean and median of the points after the peak force value
        # This will always round down
        simMaxGreaterIndex = int(simMaxIndex + (len(RF) - simMaxIndex)*(31/64))

    # Steady state values from the max force index half way to the end
    # Force values after the peak force
    RF_SteadyState = RF[simMaxGreaterIndex:]
    # Displacement values after the peak force
    dn_SteadyState = dn[simMaxGreaterIndex:]

    SSMean = np.mean(RF_SteadyState) # Mean
    SSMedian = np.median(RF_SteadyState) # Median

    # In[Plots]
    """ Plots """
    # Plot the experimental, simulated, and curve fit data

    # Split the file extension
    (figureName, ext) = os.path.splitext(fileName)

    # Plot the data trace to compare the simulated results with the force
    # displacement curves
    plt.plot(dfn_extension_shift, dfn_force,'-', color='r', linewidth=1,
             markersize=2, label = '{}, Age: {}'.format(HID, HAGE),
             alpha = 0.5)

    if str(HFMax) == 'nan' and str(HSSi) == 'nan':
        print('No max or steady state')
        pass

    if str(HFMax) != 'nan':
        plt.plot(dfn_max_Disp, HFMax,'.', color='k', linewidth=1,
                 markersize=20,
                 label = 'Max Peel - {:.4f} (mN)'.format(HFMax),
                 path_effects=[pe.Stroke(linewidth=4, foreground='k'),
                               pe.Normal()])
        plt.plot(x_n, curveFit_n, '-', color='tab:blue', linewidth=2,
                 label=r'Curve fit Max - {} (s) '.format(timeBeforePeak/10) +
                 'y = {:.4f}x + '.format(Params_n[1]) +
                 '{:.4f} (mN), '.format(Params_n[0]) +
                 '$r^2$ = {:.4f}'.format(R_Squared_n), alpha = 1)

    if str(HSSi) != 'nan':
        plt.plot(dfn_ss_Disp, ssValArray,'-', color='c', linewidth=3,
                 markersize=2,
                 label = 'Steady State - {:.4f} (mN)'.format(HFSS),
                 path_effects=[pe.Stroke(linewidth=5, foreground='k'),
                               pe.Normal()])

    # Plot the simulated data
    plt.plot(dn, RF,'-', color='blue', linewidth=2, markersize=2,
             label = r'Simulated Reaction force $\Sigma F_{Retina}$')
    plt.plot(x, curveFit,'-', color='tab:green', linewidth=2, markersize=2,
             label= 'y = {:.4f}x + '.format(Params[1]) +
             '{:.4f} (mN), '.format(Params[0]) +
             '$r^2$ = {:.4f}'.format(R_Squared))
    plt.plot(simMaxDisp, simMaxForce, '.', color='tab:red', linewidth=1,
```

```
            markersize = 20,
            label = 'Simulated maximum Force {:.4f} (mN)'.format(simMaxForce))
    plt.plot(dn_SteadyState, np.ones(len(RF_SteadyState))*SSMean, '-',
            color='tab:gray', label = 'Simulated steady state force ' +
            '{:.4f} (mN)'.format(np.mean(RF_SteadyState)))

    # In[Error Calculation]
    # error between slope, force, and steady-state value

    maxSlopeMeasured = Params_n[1] # Experimental slope
    maxSlopeSimulated = Params[1] # Simulated slope
    maxForceMeasured = HFMax # Experimental max force
    maxForceSimulated = simMaxForce # Simulated max force
    SS_Measured = HFSS # Experimental SS force
    SSmeanSimulated = SSMean # Simulated SS force (mean)
    SSmedianSimulated = SSMedian # Simulated SS force (median)

    # Error calculation
    errorDict = {} # Dictionary
    if objErr == 'Difference':
        errorDict['slope']    = (maxSlopeMeasured - maxSlopeSimulated) if slopeFlag == True else []
        errorDict['maxForce'] = (maxForceMeasured - maxForceSimulated) if maxForceFlag == True else []
        errorDict['ssForce']  = (SS_Measured - SSmeanSimulated)        if ssForceFlag == True else []
    elif objErr == 'Ratio':
        errorDict['slope']    = (1 - maxSlopeMeasured / maxSlopeSimulated) if slopeFlag == True else []
        errorDict['maxForce'] = (1 - maxForceMeasured / maxForceSimulated) if maxForceFlag == True else []
        errorDict['ssForce']  = (1 - SS_Measured / SSmeanSimulated)        if ssForceFlag == True else []
    elif objErr == 'Relative uncertainty':
        errorDict['slope']    = ((maxSlopeMeasured - maxSlopeSimulated)/maxSlopeMeasured) if slopeFlag == True else []
        errorDict['maxForce'] = ((maxForceMeasured - maxForceSimulated)/maxForceMeasured) if maxForceFlag == True else []
        errorDict['ssForce']  = ((SS_Measured - SSmedianSimulated)/SS_Measured) if ssForceFlag == True else []
    else:
        print('Error in MaxForceError')
        sys.exit()

    # Error array values
    errorList = list(errorDict.values()) # convert to list
    errorList = [x for x in errorList if x] # get rid of empty values

    # String for the error array
    errorString = ', '.join('{:.4}'.format(i) for i in errorList)

    plt.plot([dfn_max_Disp, simMaxDisp], [HFMax, simMaxForce], '--',
            linewidth = 1, color = 'magenta', label = r'Difference ' +
            'between simulated & experiment max force: ' +
            '{:.4f}'.format(HFMax - np.max(RF)))

    # Plot the different conditions if they are to be compared
    if slopeFlag == True:
```

```
        plt.plot([], [], 'white', label = r'{} '.format(objErr) +
                 'between slopes is:  ' +
                 '{:.4f}'.format(errorDict['slope']))

    if maxForceFlag == True:
        plt.plot([], [], 'white', label = r'{} '.format(objErr) +
                 'between max force is:  ' +
                 '{:.4f}'.format(errorDict['maxForce']))

    if ssForceFlag == True:
        plt.plot([], [], 'white', label = r'{} '.format(objErr) +
                 'between steady state is:  ' +
                 '{:.4f}'.format(errorDict['ssForce']))

    plt.plot([], [], 'white',
             label = r'Objective error array:  [' + errorString + ']')
    plt.plot([], [], 'white', label = r'Error $L^2$ Norm: ' +
             '{:.4f}'.format(np.sqrt(np.dot(errorList, errorList))))

    ################## Plot Data #########################
    plt.axhline(0, color='black')
    plt.axvline(0, color='black')
    plt.ylabel('Force (mN)',fontsize=18)
    plt.xlabel('Distance (mm)',fontsize=18)
    plt.title('Simulation vs. Experimental Data Trace',fontsize=20)
    plt.grid()
    plt.legend(loc = 'best',fontsize = 'medium')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
                             '_SlopeCompare.pdf'), dpi=300,
                bbox_inches='tight')
    plt.close()

    # In[Calculate interpolated Experimental and Simulated data]

    # slope calculation for 20 seconds prior to the max peel force
    # (Experimental Data)
    maxIndex = dfn_time[dfn_time == HTMax].index.values[0]
    # Array from maxIndex - timeBeforePeak*10 (timeBeforePeak sec) to location of max
    ↪  force
    t_n = dfn_time[maxIndex - timeBeforePeak:maxIndex]
    y = dfn_force[maxIndex - timeBeforePeak:maxIndex]
    # Perform least squares and return
    curveFit_n, Params_n_time, R_Squared_n = Least_Squares(t_n, y)

    # Shift extension data so that the linear region is extrapolated
    # through the origin
    shift_time = abs(Params_n_time[0]/Params_n_time[1])

    # shift time data for visual purposes
    if Params_n_time[0] > 0:
        dfn_time_shift = dfn_time + shift_time

        if min(dfn_time_shift) > 0:
            # Add zero to prevent mishaps with interpolation
            dfn_time_shift = [0] + dfn_time_shift
    else:
        dfn_time_shift = dfn_time - shift_time
```

```python
    # x array for the linear region leading up to the peak force
    Fmax_t_shift = dfn_time_shift[maxIndex]
    fit_t = np.linspace(0, Fmax_t_shift, 200) # Selected value
    # fit_t = np.linspace(0, dfn_time_shift[np.argmax(dfn_force)], 200) # true max
    Fmax_x_shift = dfn_extension_shift[maxIndex]
    # fit_x = np.linspace(0, dfn_extension_shift[np.argmax(dfn_force)], 200) # true
    ↪  max
    fit_x = np.linspace(0, Fmax_x_shift, 200) # Selected value

    # create the linear region leading up to the peak force
    def fit(params, x):
        b, m = params
        return m*x + b
    fit_vals_y_time = fit(Params_n_time, fit_t)
    fit_vals_y_force = fit(Params_n, fit_x)

    # Trim the shifted experimental data to be greater than zero
    t_exp = dfn_time_shift[dfn_time_shift >= 0]
    x_exp = dfn_extension_shift[dfn_time_shift >= 0]
    y_exp = dfn_force[dfn_time_shift >= 0]

    # data frame with original data only shifted
    dfdata = pd.DataFrame(np.array([t_exp, x_exp, y_exp]).T,
                          columns=['t', 'x', 'y'])

    # Select time beyond the max time to the end of the data
    t_geq_max = dfn_time_shift[maxIndex:]
    x_geq_max = dfn_extension_shift[maxIndex:]
    y_geq_max = dfn_force[maxIndex:]

    # dataframe of data points from the max value to the end
    dfgmax = pd.DataFrame(np.array([t_geq_max, x_geq_max, y_geq_max]).T,
                          columns=['t', 'x', 'y'])

    # data frame of points from zero to the max value
    linArray = np.array([fit_t, fit_x, fit_vals_y_force])
    dfLin = pd.DataFrame(linArray.T, columns=['t', 'x', 'y'])

    # create the new data frame of linear points up to the peak and all points
    # beyond
    dfNew = dfLin.append(dfgmax, ignore_index=True)

    # Interpolate the experimental data
    n_data_pts = 100
    start_point_time = tt[RF.argmax()] # Time at the peak (simulated)
    start_point_disp = dn[RF.argmax()] # Disp at the peak (simulated)
    f_exp_time = interpolate.interp1d(dfNew['t'], dfNew['y'])
    f_exp_disp = interpolate.interp1d(dfNew['x'], dfNew['y'])
    t_new_exp = np.linspace(start_point_time, tt[tt.argmax()],
                            n_data_pts) # (s)
    x_new_exp = np.linspace(start_point_disp, dn[tt.argmax()],
                            n_data_pts) # (mm)
    y_new_exp_time = f_exp_time(t_new_exp) # Interpolate `interp1d`
    y_new_exp_disp = f_exp_disp(x_new_exp) # Interpolate `interp1d`

    # In[Interpolated Simulated Trace]

    # Interpolate the simulated data
```

```python
    f_sim_time = interpolate.interp1d(tt, RF)
    f_sim_disp = interpolate.interp1d(dn, RF)
    t_new_sim = np.linspace(start_point_time, tt[tt.argmax()],
                            n_data_pts) # (s)
    x_new_sim = np.linspace(start_point_disp, dn[tt.argmax()],
                            n_data_pts) # (mm)
    y_new_sim_time = f_sim_time(t_new_sim) # Interpolate `interp1d`
    y_new_sim_disp = f_sim_disp(x_new_sim) # Interpolate `interp1d`

    # In[Plots]
    ''' Time curve '''
    fit, ax = plt.subplots()
    ax.plot()
    ax.plot(dfdata['t'], dfdata['y'], label='Original Shifted Data',
            alpha = 0.5)
    ax.plot(dfNew['t'], dfNew['y'], label='Merged Data',
            alpha = 0.5)
    ax.plot(t_new_exp, y_new_exp_time, '--', label='Interp Experimental Data')
    ax.plot(tt, RF, label='Simulated Data')
    ax.plot(t_new_sim, y_new_sim_time, ':', label='Interp Simulated Data')
    ax.axhline(color='k')
    ax.set_xlim([0, 300])
    ax.set_xlabel('Time (s)', fontsize=14)
    ax.set_ylabel('Force (N)', fontsize=14)
    ax.legend(loc='best', fontsize=14)
    ax.grid('on')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
                             '_Interp_Time.pdf'), dpi=300,
                bbox_inches='tight')
    plt.close()

    ''' Displacement curve '''
    fit, ax = plt.subplots()
    ax.plot()
    ax.plot(dfdata['x'], dfdata['y'], label='Original Shifted Data',
            alpha = 0.5)
    ax.plot(dfNew['x'], dfNew['y'], label='Merged Data',
            alpha = 0.5)
    ax.plot(x_new_exp, y_new_exp_disp, '--', label='Interp Experimental Data')
    ax.plot(dn, RF, label='Simulated Data')
    ax.plot(x_new_sim, y_new_sim_disp, ':', label='Interp Simulated Data')
    ax.axhline(color='k')
    ax.set_xlim([0, max(dn)])
    ax.set_xlabel('Displacement (mm)', fontsize=14)
    ax.set_ylabel('Force (N)', fontsize=14)
    ax.legend(loc='best', fontsize=14)
    ax.grid('on')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
                             '_Interp_Disp.pdf'), dpi=300,
                bbox_inches='tight')
    plt.close()

    ''' Displacement curve only showing interpolated data '''
    residual = y_new_exp_disp - y_new_sim_disp # residual calculation
    L2Norm = np.sqrt(np.dot(residual, residual))

    fit, ax = plt.subplots()
    ax.plot()
```

```python
    ax.plot(x_new_exp, y_new_exp_disp, '-', label='Interp Experimental Data')
    ax.plot(x_new_sim, y_new_sim_disp, '-', label='Interp Simulated Data')
    ax.plot(x_new_sim, residual, ':', label=r'Residual = $(exp - sim)$',
            alpha = 0.8)
    ax.plot([], [], color='white', label=r'$L^2$ norm = {:.4f}'.format(L2Norm))
    ax.axhline(color='k', linewidth=0.25)
    ax.set_xlim([0, max(x_new_exp)])
    ax.set_xlabel('Displacement (mm)', fontsize=14)
    ax.set_ylabel('Force (N)', fontsize=14)
    ax.legend(loc='best', fontsize=14)
    ax.grid('on')
    plt.savefig(os.path.join(dataDirectory,'Figures/' + figureName +
                             '_Interp_Disp_Clean.pdf'), dpi=300,
                bbox_inches='tight')
    plt.close()

    returnList = [Params[1], simMaxForce, SSMean, SSMedian, y_new_exp_disp,
                  y_new_sim_disp]
    return returnList

# In[Function that calls the nested function to compute the residual]
def findResidual(fileName, dataDirectory, maxForceTime, dataCompare, objErr,
                 slopeFlag, maxForceFlag, ssForceFlag, timeBeforePeak):
    """
    Parameters
    ----------
    fileName : Output txt file with the odb data
    dataDirectory : Location of the output file

    Returns
    -------
    maximumForce : Maximum force from the txt file
    """

    global residual
    """ Call function to return max displacement """
    ModelParamsFile = [f for f in os.listdir(dataDirectory)
                       if os.path.isfile(os.path.join(dataDirectory, f))
                       and f.startswith('output_Field')]
    for mpFile in ModelParamsFile:
        residual = residualFcn(mpFile, dataDirectory, maxForceTime,
                               dataCompare, objErr, slopeFlag, maxForceFlag,
                               ssForceFlag, timeBeforePeak)

    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.* </>

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Jun 19 16:02:44 2020

```

```
@author: Kiffer Creveling
"""

# importing os module
import os
import glob
import shutil

def MoveAbqFiles(fileName, folderDirectory, abqWD):

    # """ Change directory to correct path """

    # dataDirectory = os.path.join(abqWD, fileName)
    # if not os.path.exists(dataDirectory):
    #     os.makedirs(dataDirectory)

    # List of files in the ABQ working directory with the same name as the
    # 'fileName''
    fileList = glob.glob('{}.*'.format(os.path.join(abqWD, fileName)))
    for i in fileList:
        if i == folderDirectory:
            # Skip the file with the exact same name (i.e. Folder name...)
            continue
        source = os.path.join(abqWD,i)
        destination = os.path.join(folderDirectory)
        # copy (since shutil.move wouldn't overwrite)
        dest = shutil.copy(source, destination)
        os.remove(source) # remove the source file

    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.* </>

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 18 12:12:57 2021

@author: Kiffer2
"""

import os
from pathlib import Path
import shutil
import pandas as pd
import numpy as np
import pdb

cwd = os.getcwd() # Get current working directory

```

```python
OP = 'Results'

# In[Elastic Modulus]
ElastModResults = os.path.join(cwd, OP, 'ElasticModulusPlots')

elasticModFileName = '*optTIE_SlopeCompare.pdf'

# Search the folder directory for the file name that matches
for path in Path(cwd).rglob(elasticModFileName):


    filePathList = os.path.normpath(path).split(os.path.sep)

    # Look in the "Finished" folder
    if 'Finished' in filePathList:

        # Data trace
        dataTrace = filePathList[4] # Specific data trace

        print(dataTrace, path.name)

        # New file name (NFN)
        NFN = dataTrace + '.pdf'

        # Copy search results to the destination folder
        try:
            NP = os.path.join(cwd, ElastModResults) # New path

            # Create folder if it doesn't exist
            os.makedirs(NP, exist_ok=True)

            shutil.copy(path, os.path.join(NP, NFN)) # move files
        except shutil.SameFileError:
            pass



# In[Elastic Modulus Attributes]
ElastModAttr = os.path.join(cwd, OP, 'ElasticModulusAttr')

initialName = 'output'
elasticModFileAttr = '*optTIE.txt'

# Search the folder directory for the file name that matches
for path in Path(cwd).rglob(elasticModFileAttr):


    # Search for the input parameters
    if path.name.find(initialName) < 0:


        filePathList = os.path.normpath(path).split(os.path.sep)

        # Look in the "Finished" folder
        if 'Finished' in filePathList:

            # Data trace
            dataTrace = filePathList[4] # Specific data trace
```

```python

            print(dataTrace, path.name)

            # New file name (NFN)
            NFN = dataTrace + '.txt'

            # Copy search results to the destination folder
            try:
                NP = os.path.join(cwd, ElastModAttr) # New path

                # Create folder if it doesn't exist
                os.makedirs(NP, exist_ok=True)

                shutil.copy(path, os.path.join(NP, NFN)) # move files
            except shutil.SameFileError:
                pass


# In[Elastic Modulus Convergence]
ElastModConv = os.path.join(cwd, OP, 'ElasticModulusConvergence')

ElastModConvAttr = 'FEAAttributes.txt'

# Search the folder directory for the file name that matches
for path in Path(cwd).rglob(ElastModConvAttr):

    # Split file path to a list
    filePathList = os.path.normpath(path).split(os.path.sep)

    # Look in the "Finished" folder
    if 'Finished' in filePathList:

        if filePathList[6].find('optTIE') > 0:

            # Data trace
            dataTrace = filePathList[4] # Specific data trace

            print(dataTrace, path.name)

            # New file name (NFN)
            NFN = dataTrace + '.txt'

            # Load Data

            # Add names to file because 'SimSlope','SimMax', 'SimSS'
            # were missing
            names = ['FileName', 'Time', 'E1', 'E2', 'PT', 'G', 'V1', 'V2',
                     'R', 'F', 'MS', 'RE', 'VE', 'Knn', 'Kss', 'Ktt',
                     'DamageInitiation', 'tn', 'ts', 'tt', 'DamageEvolution',
                     'FE', 'Optimization', 'TIE', 'errorListL2Norm',
                     'ObjectiveFunction', 'SimSlope','SimMax', 'SimSS',
                     'simTime']
            df = pd.read_csv(path, names=names, sep='\t', header=0)

            # Save Dat
            df.to_csv(path, sep='\t', index=False, na_rep='nan')

            # Copy search results to the destination folder
```

```
            try:
                NP = os.path.join(cwd, ElastModConv) # New path

                # Create folder if it doesn't exist
                os.makedirs(NP, exist_ok=True)

                shutil.copy(path, os.path.join(NP, NFN)) # move files

                df.to_csv(os.path.join(NP, NFN), sep='\t', index=False,
                      na_rep='nan')
            except shutil.SameFileError:
                pass

# In[Cohesive Behavior Plots]
CohesiveResults = os.path.join(cwd, OP, 'CohesiveBehaviorPlots')

CohesiveFileName = '*opt_SlopeCompare.pdf'

# Search the folder directory for the file name that matches
for path in Path(cwd).rglob(CohesiveFileName):

    filePathList = os.path.normpath(path).split(os.path.sep)

    # Look in the "Finished" folder
    if 'Finished' in filePathList:

        # Data trace
        dataTrace = filePathList[4] # Specific data trace

        print(dataTrace, path.name)

        # New file name (NFN)
        NFN = dataTrace + '.pdf'

        # Copy search results to the destination folder
        try:
            NP = os.path.join(cwd, CohesiveResults) # New path

            # Create folder if it doesn't exist
            os.makedirs(NP, exist_ok=True)

            shutil.copy(path, os.path.join(NP, NFN)) # move files
        except shutil.SameFileError:
            pass


# In[Cohesive Behavior Attributes]
CohesiveAttr = os.path.join(cwd, OP, 'CohesiveBehaviorAttr')

initialName = '_T3_C'
elasticModFileAttr = '*opt.txt'

# Search the folder directory for the file name that matches
for path in Path(cwd).rglob(elasticModFileAttr):

    # Search for the input parameters
    if path.name.find(initialName) < 0:

```

```
        filePathList = os.path.normpath(path).split(os.path.sep)

        # Look in the "Finished" folder
        if 'Finished' in filePathList:

            # Data trace
            dataTrace = filePathList[4] # Specific data trace

            print(dataTrace, path.name)

            # New file name (NFN)
            NFN = dataTrace + '.txt'

            # Copy search results to the destination folder
            try:
                NP = os.path.join(cwd, CohesiveAttr) # New path

                # Create folder if it doesn't exist
                os.makedirs(NP, exist_ok=True)

                shutil.copy(path, os.path.join(NP, NFN)) # move files
            except shutil.SameFileError:
                pass


# In[Cohesive Behavior Convergence]
CohConv = os.path.join(cwd, OP, 'CohesiveBehaviorConvergence')

CohConvAttr = 'FEAAttributes.txt'

# Search the folder directory for the file name that matches
for path in Path(cwd).rglob(CohConvAttr):

    # Split file path to a list
    filePathList = os.path.normpath(path).split(os.path.sep)

    # Look in the "Finished" folder
    if 'Finished' in filePathList:

        if filePathList[5].find('optTIE') == -1:

            # Data trace
            dataTrace = filePathList[4] # Specific data trace

            print(dataTrace, path.name)

            # New file name (NFN)
            NFN = dataTrace + '.txt'

            # Load Data

            # Add names to file because 'SimSlope','SimMax', 'SimSS'
            # were missing
            names = ['FileName', 'Time', 'E1', 'E2', 'PT', 'G', 'V1', 'V2',
                     'R', 'F', 'MS', 'RE', 'VE', 'Knn', 'Kss', 'Ktt',
                     'DamageInitiation', 'tn', 'ts', 'tt', 'DamageEvolution',
                     'FE', 'Optimization', 'TIE', 'errorListL2Norm',
                     'ObjectiveFunction', 'SimSlope','SimMax', 'SimSS',
```

```
                      'simTime']
            df = pd.read_csv(path, names=names, sep='\t', header=0)

            # Save Dat
            df.to_csv(path, sep='\t', index=False, na_rep='nan')



            # Copy search results to the destination folder
            try:
                NP = os.path.join(cwd, CohConv) # New path

                # Create folder if it doesn't exist
                os.makedirs(NP, exist_ok=True)

                shutil.copy(path, os.path.join(NP, NFN)) # move files

                df.to_csv(os.path.join(CohConv, NFN), sep='\t', index=False,
                      na_rep='nan')
            except shutil.SameFileError:
                pass

# In[Fracture Energy Integral]

FEInt = os.path.join(cwd, OP, 'FractureEnergyIntegrals')

elasticModFileAttr = 'GcSelection.pdf'

# Search the folder directory for the file name that matches
for path in Path(cwd).rglob(elasticModFileAttr):

    filePathList = os.path.normpath(path).split(os.path.sep)

    # Look in the "Finished" folder
    if 'Finished' in filePathList:

        # Data trace
        dataTrace = filePathList[4] # Specific data trace

        print(dataTrace, path.name)

        # New file name (NFN)
        NFN = dataTrace + '.pdf'

        # Copy search results to the destination folder
        try:
            NP = os.path.join(cwd, FEInt) # New path

            # Create folder if it doesn't exist
            os.makedirs(NP, exist_ok=True)

            shutil.copy(path, os.path.join(NP, NFN)) # move files
        except shutil.SameFileError:
            pass


# In[YouTube video links]

```

```python
YouTube = os.path.join(cwd, OP, 'YouTube')

YouTubeFile = 'YouTubeLink.txt'

# Search the folder directory for the file name that matches
for path in Path(cwd).rglob(YouTubeFile):

    filePathList = os.path.normpath(path).split(os.path.sep)

    # Look in the "Finished" folder
    if 'Finished' in filePathList:

        # Data trace
        dataTrace = filePathList[4] # Specific data trace

        print(dataTrace, path.name)

        # New file name (NFN)
        NFN = dataTrace + '.txt'

        # Copy search results to the destination folder
        try:
            NP = os.path.join(cwd, YouTube) # New path

            # Create folder if it doesn't exist
            os.makedirs(NP, exist_ok=True)

            shutil.copy(path, os.path.join(NP, NFN)) # move files
        except shutil.SameFileError:
            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.* </>

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 18 17:25:44 2021

@author: Kiffer
"""

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

```python
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [16, 10]
from scipy import stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
import pdb
import os
import glob
import re

cwd = os.getcwd()

# In[Peel test data]]

fp = os.path.join(cwd, 'Results', 'ExpPeelDataCompare')

# Grab all files that have "trace_"
fileList = glob.glob(os.path.join(fp, "trace_*"))

finalAttr = []

for i,j in enumerate(fileList):

    # Load each data set
    df = pd.read_csv(j, sep = '\t', nrows=29, names=['Variable', 'Value'])
    df = df.set_index('Variable').T # Transpose

    filePathList = os.path.normpath(j).split(os.path.sep)
    DataTrace = filePathList[-1]

    # Add the trace name to the dataframe
    df['DataTrace'] = DataTrace

    # Specific trace number from the string
    result = re.search('Trace_(.*)_Instron_Data.txt', DataTrace)

    # convert the string to an integer for sorting
    df['DataTrace#'] = int(result.group(1))

    finalAttr.append(df.tail(1).values.tolist())

# Compress list
finalAttr = [item for sublist in finalAttr for item in sublist]

# Create the new dataframe with the names from the previous data
df = pd.DataFrame(finalAttr, columns=df.columns.values)

# Experimental Data (ed)
ed = df.sort_values('DataTrace#').reset_index(drop=True)

#--- Save Data ---#
outputFileDirectory = os.path.join('Results', 'OutputFiles') # Folder

# Make folder if it doesn't exist
os.makedirs(outputFileDirectory, exist_ok=True)

# new File
outputFile = os.path.join(outputFileDirectory, 'PeelDataSummary.txt')
```

```
print('New file:', outputFile)

# Save results
ed.to_csv(outputFile, sep='\t', index=False, na_rep='nan')


# In[simulation results]
# Headers
names = ['FileName', 'Time', 'E1', 'E2', 'PT', 'G', 'V1', 'V2', 'R',
         'F', 'MS', 'RE', 'VE', 'Knn', 'Kss', 'Ktt',
         'DamageInitiation', 'tn', 'ts', 'tt', 'DamageEvolution',
         'FE', 'Optimization', 'TIE', 'errorListL2Norm',
         'ObjectiveFunction', 'SimSlope','SimMax', 'SimSS', 'simTime']

SF = os.path.join('Results', 'StatisticsFigures')

# Create folder if it doesn't exist
os.makedirs(SF, exist_ok=True)

# In[Elastic Modulus Convergende]

fp = os.path.join(cwd, 'Results', 'ElasticModulusConvergence')

fileList = glob.glob(os.path.join(fp, "sample*.txt"))

finalAttr = []
ElasticSummary = {} # Dictionary to look at each optimization routine

for i in fileList:

    # Load each data set
    df = pd.read_csv(i, sep = '\t')
    finalAttr.append(df.tail(1).values.tolist())

    # Append each data set to a single dictionary
    ElasticSummary[i] = df

# Compress list
finalAttr = [item for sublist in finalAttr for item in sublist]

# Create the new dataframe
df = pd.DataFrame(finalAttr, columns=names)

#--- Save Data ---#
outputFileDirectory = os.path.join('Results', 'OutputFiles') # Folder

# Make folder if it doesn't exist
os.makedirs(outputFileDirectory, exist_ok=True)

# new File
outputFile = os.path.join(outputFileDirectory, 'ElasticConvergenceSummary.txt')

print('New file:', outputFile)

# Save results
df.to_csv(outputFile, sep='\t', index=False, na_rep='nan')
```

```python

Ev = df['VE'] # Elastic modulus

# In[Plots]

standardError = 68 # Used for confidence intervals

sns.set_theme(context='paper', style='darkgrid', palette="Paired",
              font_scale=2)
sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                             "axes.labelsize":12})
custom_style = {'axes.facecolor': 'white',
                'axes.edgecolor': 'black',
                'axes.grid': False,
                'axes.axisbelow': True,
                'axes.labelcolor': 'black',
                'figure.facecolor': 'white',
                'grid.color': '.8',
                'grid.linestyle': '-',
                'text.color': 'black',
                'xtick.color': 'black',
                'ytick.color': 'black',
                'xtick.direction': 'out',
                'ytick.direction': 'out',
                'lines.solid_capstyle': 'round',
                'patch.edgecolor': 'w',
                'patch.force_edgecolor': True,
                'image.cmap': 'rocket',
                'font.family': ['sans-serif'],
                'font.sans-serif': ['Arial', 'DejaVu Sans', 'Liberation Sans',
                                    'Bitstream Vera Sans', 'sans-serif'],
                'xtick.bottom': True,
                'xtick.top': False,
                'ytick.left': True,
                'ytick.right': False,
                'axes.spines.left': True,
                'axes.spines.bottom': True,
                'axes.spines.right': False,
                'axes.spines.top': False,
                'xtick.labelsize' : 16,
                'ytick.labelsize' : 16,
                'legend.title_fontsize' : 20}

# White background with ticks and black border lines, Turns grid off
ax = sns.set_style(rc=custom_style)

# In[Functions]

# fcn for plotting
def yfit(x):
    return slope*x + intercept

# In[Pivot table info]

pvtOut = {'count', np.median, np.mean, np.std} # pivot table outputs

# In[Plot simplifications]

```

```
R = 'Region'
Eq = 'Equator'
Po = 'Posterior'
AG = 'AgeGroup'
A60 = 'Age60'
Aleq60 = r'Age $\leq$ 60'
Ag60 = 'Age $>$ 60'
A = 'Age'

# Units
MPF = 'Maximum Peel Force (mN)'
SSPF = 'Steady-State Peel Force (mN)'
KDEUnit = r'Kernel Density Estimation'
ElasticUnit = r'Elastic Modulus (Pa)'
CohBehUnit = r'Cohesive Behavior (Pa)'
CohDMGUnit = r'Cohesive Damage Initiation (Pa)'
FEUnit = r'Fracture Energy (J)'

A_yrs = 'Age (yr.)'
A_G = 'Age Group (yr.)'


# In[Cohesive Behavior Convergence]

fp = os.path.join(cwd, 'Results', 'CohesiveBehaviorConvergence')

fileList = glob.glob(os.path.join(fp, "sample*.txt"))

finalAttr = []

for i,j in enumerate(fileList):

    # Load each data set
    df = pd.read_csv(j, sep = '\t')


    filePathList = os.path.normpath(j).split(os.path.sep)
    fileName = filePathList[-1]

    # Add the filename
    df['SimulationFileName'] = fileName

    # Specific trace number from the string
    result = re.search('Sample#(.*).txt', fileName)

    # convert the string to an integer for sorting
    df['DataTrace#'] = int(result.group(1))

    # Grab the final row (converged results)
    finalAttr.append(df.tail(1).values.tolist())

# Compress list
finalAttr = [item for sublist in finalAttr for item in sublist]

# Create the new dataframe
df = pd.DataFrame(finalAttr, columns=df.columns.values)

# Simulation Data (sd)
```

```
sd = df.sort_values('DataTrace#').reset_index(drop=True)

#--- Save Data ---#
outputFileDirectory = os.path.join('Results', 'OutputFiles') # Folder

# Make folder if it doesn't exist
os.makedirs(outputFileDirectory, exist_ok=True)

# new File
outputFile = os.path.join(outputFileDirectory, 'CohesiveBehaviorSummary.txt')

print('New file:', outputFile)

# Save results
sd.to_csv(outputFile, sep='\t', index=False, na_rep='nan')

Ev = sd['VE'] # Elastic modulus

# Cohesive Behaviour
sd['Knn'] = 2**sd['Knn'] # Elastic modulus
sd['Kss'] = 2**sd['Kss'] # Elastic modulus
sd['Ktt'] = 2**sd['Ktt'] # Elastic modulus

# Damage Initiation
sd['tn'] = 2**sd['tn'] # Normal 1
sd['ts'] = 2**sd['ts'] # Shear 1
sd['tt'] = 2**sd['tt'] # Shear 2

# Fracture energy
sd['FE'] = 2**sd['FE'] # Elastic modulus

# Outputs
errList = sd['errorListL2Norm']

SS = sd['SimSlope']
SM = sd['SimMax']
Ss = sd['SimSS']

# In[YouTube Links]

fp = os.path.join(cwd, 'Results', 'YouTube')

fileList = glob.glob(os.path.join(fp, "sample*.txt"))

finalAttr = []

for i,j in enumerate(fileList):

    # Load each data set
    df = pd.read_csv(j, names=['Link'])

    filePathList = os.path.normpath(j).split(os.path.sep)
    fileName = filePathList[-1]

    # Add the filename
    df['SimulationFileName'] = fileName

    # Specific trace number from the string
```

```
    result = re.search('Sample#(.*).txt', fileName)

    tNum = int(result.group(1)) # Trace number

    # convert the string to an integer for sorting
    df['DataTrace#'] = tNum

    # Hyperlink for LaTeX
    df['HyperLink'] = ('\href{' + '{}'.format(df['Link'][0]) +
                       '}{' + '{}'.format(tNum) + '}')

    # Grab the final row (converged results)
    finalAttr.append(df.tail(1).values.tolist())

# Compress list
finalAttr = [item for sublist in finalAttr for item in sublist]

# Create the new dataframe
df = pd.DataFrame(finalAttr, columns=df.columns.values)

# YouTube (yt)
yt = df.sort_values('DataTrace#').reset_index(drop=True)

#--- Save Data ---#
outputFileDirectory = os.path.join('Results', 'OutputFiles') # Folder

# Make folder if it doesn't exist
os.makedirs(outputFileDirectory, exist_ok=True)

# new File
outputFile = os.path.join(outputFileDirectory, 'YouTube.txt')

print('New file:', outputFile)

# In[Cohesive Behavior Convergence]

fp = os.path.join(cwd, 'Results', 'CohesiveBehaviorConvergence')

fileList = glob.glob(os.path.join(fp, "sample*.txt"))

finalAttr = []

CohesiveSummary = {}

for i,j in enumerate(fileList):

    # Load each data set
    df = pd.read_csv(j, sep = '\t')


    filePathList = os.path.normpath(j).split(os.path.sep)
    fileName = filePathList[-1]

    # Add the filename
    df['SimulationFileName'] = fileName

    # Specific trace number from the string
    result = re.search('Sample#(.*).txt', fileName)
```

```
    # convert the string to an integer for sorting
    df['DataTrace#'] = int(result.group(1))

    # Grab the final row (converged results)
    finalAttr.append(df.tail(1).values.tolist())

    # Append each data set to a single dictionary
    CohesiveSummary[i] = df

# Compress list
finalAttr = [item for sublist in finalAttr for item in sublist]

# Create the new dataframe
df = pd.DataFrame(finalAttr, columns=df.columns.values)

# Simulation Data (sd)
sd = df.sort_values('DataTrace#').reset_index(drop=True)

#--- Save Data ---#
outputFileDirectory = os.path.join('Results', 'OutputFiles') # Folder

# Make folder if it doesn't exist
os.makedirs(outputFileDirectory, exist_ok=True)

# new File
outputFile = os.path.join(outputFileDirectory, 'CohesiveBehaviorSummary.txt')

print('New file:', outputFile)

# In[Merge data sets (Experimental & Simulation)]

# Merge experimental and simulation data
md1 = pd.merge(sd, ed, on='DataTrace#')
md = pd.merge(md1, yt, on='DataTrace#')

# Simplifications
R = 'Region'
Eq = 'Equator'
Po = 'Posterior'
A60 = 'Age60'
Aleq60 = r'Age $\leq$ 60'
Ag60 = r'Age $>$ 60'
A = 'Age'

# Redo some columns for plotting
md[A] = md['Human Age:']
md[R] = md['Human Region:']

# Break age groups into bins
bins = [0, 60, 90]
labelsAge60 = [Aleq60, Ag60]

# Properly update parameters
# Cohesive Behaviour
md['Knn'] = 2**md['Knn'] # Elastic modulus
md['Kss'] = 2**md['Kss'] # Elastic modulus
md['Ktt'] = 2**md['Ktt'] # Elastic modulus
```

```python

# Damage Initiation
md['tn'] = 2**md['tn'] # Normal 1
md['ts'] = 2**md['ts'] # Shear 1
md['tt'] = 2**md['tt'] # Shear 2

# Fracture energy
md['FE'] = 2**md['FE'] # Elastic modulus

# Create binned AgeGroups
md[A60] = pd.cut(md[A].astype(int), bins, labels=labelsAge60, right=True)

# Convert Strings to floats/integers
md['EV'] = pd.to_numeric(md['VE'], downcast="float")

md['Trace'] = pd.to_numeric(md['DataTrace#'], downcast="integer")
md['$Exp_\max$'] = pd.to_numeric(md['FMax (mN):'], downcast="float")
md['$Exp_{SS}$'] = pd.to_numeric(md['FSS (mN):'], downcast="float")
md['$Sim_\max$'] = pd.to_numeric(md['SimMax'], downcast="float")
md['$Sim_{SS}$'] = pd.to_numeric(md['SimSS'], downcast="float")
md['$L^2$ Norm'] = pd.to_numeric(md['errorListL2Norm'], downcast="float")
md[A] = pd.to_numeric(md[A], downcast="integer")

# Simplify for later
md['L/R'] = np.where(md['Human Left/Right:'] == 'Left', 'L', 'R')
md['Region'] = np.where(md['Region'] == 'Equator', 'Eq.', 'Po.')

#--- Save Data ---#
outputFileDirectory = os.path.join('Results', 'OutputFiles') # Folder

# Make folder if it doesn't exist
os.makedirs(outputFileDirectory, exist_ok=True)

# new File
outputFile = os.path.join(outputFileDirectory, 'ExpSimSummary.txt')

print('New file:', outputFile)

# Save results
md.to_csv(outputFile, sep='\t', index=False, na_rep='nan')

# Create specific LaTeX table

# Add the index groups and convert NaN's to "-"'s
tabColumns = ['Trace',
              A,
              'L/R',
              'Region',
              '$Exp_\max$',
              '$Exp_{SS}$',
              '$Sim_\max$',
              '$Sim_{SS}$',
              '$L^2$ Norm',
              'HyperLink']

print(md.to_latex(index=False, columns=tabColumns, na_rep='-', escape=False,
                  float_format="{:0.3f}".format))

```

```
# In[Add full name to region after the table was created]

# md[R] = np.where(md[R] == 'Eq.', 'Equator', 'Posterior') # Difficult to use
md.loc[md[R] == 'Eq.', R] = Eq
md.loc[md[R] == 'Po.', R] = Po

# In[Smart Plot]

def boxPlotBlackBorder(ax):
    # iterate over boxes in the plot to make each line black
    for i,box in enumerate(ax.artists):
        box.set_edgecolor('black')
        # box.set_facecolor('white')

        # iterate over whiskers and median lines
        for j in range(6*i, 6*(i+1)):
            ax.lines[j].set_color('black')

def smartPlot(data=None, x=None, y=None, hue=None, hue_order=None,
              addBoxPair=None, ci=None, errcolor=None, capsize=None,
              plot=None, test=None, sigLoc=None, text_format=None,
              line_offset=None, line_offset_to_box=None, line_height=None,
              fontsize=None, legLoc=None, verbose=None, yAxis=None,
              xlabel=None, ylabel=None, legendTitle=None, figName=None,
              folderName=None, dataPoints=None, stats=None):

    # barplot
    scale = 1.6
    base = 10
    f, ax = plt.subplots(figsize=(base*scale, base))

    if plot == 'barplot':
        ax = sns.barplot(data=data, x=x, y=y, hue=hue, hue_order=hue_order,
                         ci=ci, errcolor=errcolor, capsize=capsize)

    elif plot == 'boxplot':
        ax = sns.boxplot(data=data, x=x, y=y, hue=hue, hue_order=hue_order)

    # Statistical test for differences
    x_grps = list(data[x].unique()) # List of groups
    if hue != None:
        # Create combinations to compare
        box_pairs_1 = [((x_grps_i, hue_order[0]),
                        (x_grps_i, hue_order[1]))
                       for x_grps_i in x_grps]
        box_pairs = box_pairs_1

        if addBoxPair != None:
            # Additional box pairs
            box_pairs =  box_pairs_1 + addBoxPair

    elif hue_order != None:
        box_pairs = [(hue_order[0], hue_order[1])]

    if yAxis != None:
            ax.set_yscale("log")

    if stats != None:
```

```python
        #Stats results and significant differences (SR)
        SR = add_stat_annotation(ax, plot=plot, data=data, x=x, y=y, hue=hue,
                                 hue_order=hue_order, box_pairs=box_pairs,
                                 test=test, loc=sigLoc,
                                 text_format=text_format, verbose=verbose,
                                 comparisons_correction=None,
                                 line_offset=line_offset,
                                 line_offset_to_box=line_offset_to_box,
                                 line_height= line_height,
                                 fontsize=fontsize) # 'bonferroni'

    if plot == 'boxplot':
        boxPlotBlackBorder(ax) # Make borders black

    if dataPoints == True:
        # Add data points to the box plot
        sns.stripplot(data=data, x=x, y=y, hue=hue, hue_order=hue_order,
                      color='.5', size=5, linewidth=1, dodge=True)

        # gather plot attributes for legends
        handles, labels = ax.get_legend_handles_labels()

        if hue != None:
            l = plt.legend(handles[0:2], labels[0:2], title=legendTitle,
                           fontsize=18)

    else:
        if hue != None:
            ax.legend(loc=legLoc, fontsize=18).set_title(legendTitle)

    if hue != None and hue_order != None:
        # for legend title
        plt.setp(ax.get_legend().get_title(), fontsize=18)

    ax.set_xlabel(xlabel, fontsize=18)
    ax.set_ylabel(ylabel, fontsize=18)

    # Adjust fonts, because it doesn't seem to work
    for tick in ax.xaxis.get_major_ticks():
        tick.label.set_fontsize(16)

    for tick in ax.yaxis.get_major_ticks():
        tick.label.set_fontsize(16)

    ax = sns.despine() # takes the lines off on the right and top of the graph

    if folderName != None:
        # If a new folder name is given, put the files there

        # New file path
        NP = os.path.join(SF, folderName)

        # Create folder if it doesn't exist
        os.makedirs(NP, exist_ok=True)

    else:
        # Put the file in the same folder
        NP = SF
```

```

    f.savefig(os.path.join(NP, '{}.pdf'.format(figName)),
              bbox_inches='tight')
    plt.close()

def twoWayAnova(data=None, var=None, A=None, B=None, fileName=None,
                filePath=None):
    """
    Two-Way Anova
    Parameters
    ----------
    data : dataframe
    var : continuous variable
    A : Effect #1
    B : Effect #2
    fileName : Filename
    filePath : filepath
    """
    model = ols(f'{var} ~ {A} + {B} + {A}:{B}', data=data).fit()

    res = sm.stats.anova_lm(model, typ=2)

    print(80*'-', 2*'\n', 'Two-way ANOVA\n', res, 2*'\n')

    NP = os.path.join(SF, filePath)

    # Create folder if it doesn't exist
    os.makedirs(NP, exist_ok=True)

    f = open(os.path.join(NP, f'{fileName}.txt'), "w")
    f.write(var + '\twith effects:  ' + A + ' and ' + B)
    f.write('\n')
    f.write(res.to_string())
    f.close()

# In[Elastic modulus group plots by region and age +/- 60]
EV = 'EV'
Ev = [EV]
Folder = 'ElasticModulus'

# Additional group
addBoxPair1 = [((Aleq60, Eq), (Age_Group_i, Eq))
               for Age_Group_i in list(md[A60].unique())]

addBoxPair2 = [((Aleq60, Eq), (Ag60, Po))]

addBoxPair = addBoxPair1 #+ addBoxPair2

for i in Ev:
    pivotEv = pd.pivot_table(md, values=i, index=[A60, R],
                                                  aggfunc=pvtOut)

    print('pivotEv')
    print(pivotEv)
    # Add the index groups and convert NaN's to "-"'s
    print(pivotEv.to_latex(index=True, na_rep='-', escape=False,
                           float_format="{:0.3f}".format))

```

```
    # Barplot
    smartPlot(data=md, x=A60, y=i, hue=R, hue_order=[Eq, Po], ci=68,
              errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
              sigLoc='outside', text_format='star', line_offset=0.015,
              line_offset_to_box=0.0, line_height=0.015, fontsize=16,
              legLoc='best', verbose=2, yAxis=None,
              xlabel=A_G, ylabel=ElasticUnit, legendTitle=R,
              figName='RegionAge_BarPlot', folderName=Folder,
              addBoxPair=addBoxPair, stats=None)

    # Boxplot
    smartPlot(data=md, x=A60, y=i, hue=R, hue_order=[Eq, Po], plot='boxplot',
              test='t-test_ind', sigLoc='outside', text_format='star',
              line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
              fontsize=16, legLoc='best', verbose=2, yAxis=None,
              xlabel=A_G, ylabel=ElasticUnit, addBoxPair=addBoxPair,
              legendTitle=R, figName='RegionAge_BoxPlot', folderName=Folder,
              stats=None)

    # Boxplot with data
    smartPlot(data=md, x=A60, y=i, hue=R, hue_order=[Eq, Po], plot='boxplot',
              test='t-test_ind', sigLoc='outside', text_format='star',
              line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
              fontsize=16, legLoc='best', verbose=2, yAxis=None,
              xlabel=A_G, ylabel=ElasticUnit, addBoxPair=addBoxPair,
              legendTitle=R,
              figName='RegionAge_BoxPlotWithData', folderName=Folder,
              dataPoints=True, stats=None)

twoWayAnova(data=md, var='VE', A='Age', B='Region',
            fileName='Age_Region_2wayAnova',
            filePath=Folder)

twoWayAnova(data=md, var='VE', A=A60, B=R,
            fileName='AgeGroup_Region_2wayAnova',
            filePath=Folder)

# In[Elastic modulus group plots by Age Group +/- 60]
EV = 'EV'
Ev = [EV]
Folder = 'ElasticModulus'

# Additional group
addBoxPair1 = [(Aleq60, Ag60)]

addBoxPair = addBoxPair1

# Create a matching column for repeated measures
md['MatchingID'] = md['Human ID:'].map(str) + md['Human Region:'].map(str)

# matched_pairs student's t-test

# dfMP = md[md.duplicated(['MatchingID'], keep=False)]
# f, p = stats.ttest_rel(dfMP[Ev][dfMP[R] == Eq],
#                        dfMP[Ev][dfMP[R] == Po])

data = md # dfMP

```

```
for i in Ev:
    pivotEv = pd.pivot_table(md, values=i, index=[A60, R],
                                                 aggfunc=pvtOut)

    print('pivotEv')
    print(pivotEv)
    # Add the index groups and convert NaN's to "-"'s
    print(pivotEv.to_latex(index=True, na_rep='-', escape=False,
                           float_format="{:0.3f}".format))

    # Barplot
    smartPlot(data=data, x=A60, y=i, hue=None, hue_order=[Aleq60, Ag60], ci=68,
              errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
              sigLoc='outside', text_format='star', line_offset=0.015,
              line_offset_to_box=0.0, line_height=0.015, fontsize=16,
              legLoc='best', verbose=2, yAxis=None,
              xlabel=A_G, ylabel=ElasticUnit, legendTitle=None,
              figName='Age_BarPlot', folderName=Folder,
              addBoxPair=addBoxPair, stats=True)

    # Boxplot
    smartPlot(data=data, x=A60, y=i, hue=None, hue_order=[Aleq60, Ag60],
              plot='boxplot',
              test='t-test_ind', sigLoc='outside', text_format='star',
              line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
              fontsize=16, legLoc='best', verbose=2, yAxis=None,
              xlabel=A_G, ylabel=ElasticUnit, addBoxPair=addBoxPair,
              legendTitle=R, figName='Age_BoxPlot', folderName=Folder,
              stats=True)

    # Boxplot with data
    smartPlot(data=data, x=A60, y=i, hue=None, hue_order=[Aleq60, Ag60],
              plot='boxplot',
              test='t-test_ind', sigLoc='outside', text_format='star',
              line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
              fontsize=16, legLoc='best', verbose=2, yAxis=None,
              xlabel=A_G, ylabel=ElasticUnit, addBoxPair=addBoxPair,
              legendTitle=R,
              figName='Age_BoxPlotWithData', folderName=Folder,
              dataPoints=True, stats=True)

# In[Elastic modulus group plots by region]
EV = 'EV'
Ev = [EV]
Folder = 'ElasticModulus'

# Additional group
addBoxPair1 = [(Eq, Po)]

addBoxPair = addBoxPair1

for i in Ev:
    pivotEv = pd.pivot_table(md, values=i, index=[A60, R],
                                                 aggfunc=pvtOut)

    print('pivotEv')
    print(pivotEv)
    # Add the index groups and convert NaN's to "-"'s
```

```
    print(pivotEv.to_latex(index=True, na_rep='-', escape=False,
                           float_format="{:0.3f}".format))

    # Barplot
    smartPlot(data=md, x=R, y=i, hue=None, hue_order=[Eq, Po], ci=68,
              errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
              sigLoc='outside', text_format='star', line_offset=0.015,
              line_offset_to_box=0.0, line_height=0.015, fontsize=16,
              legLoc='best', verbose=2, yAxis=None,
              xlabel=A_G, ylabel=ElasticUnit, legendTitle=None,
              figName='Age_BarPlot', folderName=Folder,
              addBoxPair=addBoxPair, stats=True)

    # Boxplot
    smartPlot(data=md, x=R, y=i, hue=None, hue_order=[Eq, Po], plot='boxplot',
              test='t-test_ind', sigLoc='outside', text_format='star',
              line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
              fontsize=16, legLoc='best', verbose=2, yAxis=None,
              xlabel=A_G, ylabel=ElasticUnit, addBoxPair=addBoxPair,
              legendTitle=R, figName='Age_BoxPlot', folderName=Folder,
              stats=True)

    # Boxplot with data
    smartPlot(data=md, x=R, y=i, hue=None, hue_order=[Eq, Po], plot='boxplot',
              test='t-test_ind', sigLoc='outside', text_format='star',
              line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
              fontsize=16, legLoc='best', verbose=2, yAxis=None,
              xlabel=A_G, ylabel=ElasticUnit, addBoxPair=addBoxPair,
              legendTitle=R,
              figName='Age_BoxPlotWithData', folderName=Folder,
              dataPoints=True, stats=True)

# In[Elastic Modulus Regression by Age in both Regions]

Folder = 'ElasticModulus'

# Linear regression
f, ax = plt.subplots()
sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                             "axes.labelsize":12})
ax = sns.lmplot(data=md, x=A, y=EV, hue=R, markers=["o", "x"],
                legend_out=False, fit_reg=True, height=5, aspect=1.6,
                palette="Set1", truncate=True, ci=95, line_kws={'lw':0})

ax.set(xlabel=A_yrs, ylabel=ElasticUnit)

ax = ax.axes[0][0] # Convert faceted grid to matplotlib fig

# Remove all NaN's from the data for regressions
# remove nans from ILM thickness & Max
df_no_Nan = md.dropna(subset=[A, EV])

# linear regressions for fitting
x = df_no_Nan[A][df_no_Nan[R] == Eq]
# Convert to N
y = df_no_Nan[EV][df_no_Nan[R] == Eq]

x_plot = np.linspace(min(x), max(x), 100)
```

```
slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
ax.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
ax.text(62, yfit(62) + 5, r'$r={:.4f}$'.format(r_value1), color='r',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

print('Values for correlation between ' +
      'Elastic Modulus and Age in the Equator\n',
      'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value1))

# linear regressions for fitting
x = df_no_Nan[A][df_no_Nan[R] == Po]
y = df_no_Nan[EV][df_no_Nan[R] == Po]

x_plot = np.linspace(min(x), max(x), 100)
slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
ax.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
ax.text(62, yfit(62) + 5, r'$r={:.4f}$'.format(r_value2), color='b',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

print('Values for correlation between ' +
      'Elastic Modulus and Age in the Equator\n',
      'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value2))

# Axis limits
ax.set(ylim=(0, 500))
ax.set(xlim=(29, 80))

# New path
NP = os.path.join(SF, Folder)

# Create folder if it doesn't exist
os.makedirs(NP, exist_ok=True)

plt.savefig(os.path.join(NP, 'Regression_Age_by_Region.pdf'),
            bbox_inches='tight')
plt.close()


# In[Elastic Modulus Regression by Max Peel Force in both regions]

Fmax = 'SimMax'
Folder = 'ElasticModulus'

# Linear regression
f, ax = plt.subplots()
sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                             "axes.labelsize":12})
ax = sns.lmplot(data=md, x=EV, y=Fmax, hue=R, hue_order= [Eq, Po],
                markers=["o", "x"], legend_out=False, fit_reg=True,
                height=5, aspect=1.6, palette="Set1", truncate=True, ci=95,
                line_kws={'lw':0})

ax.set(xlabel=ElasticUnit, ylabel=MPF)

ax = ax.axes[0][0] # Convert faceted grid to matplotlib fig

# Remove all NaN's from the data for regressions
```

```python
# remove nans from ILM thickness & Max
df_no_Nan = md.dropna(subset=[Fmax, EV])

# linear regressions for fitting
x = df_no_Nan[EV][df_no_Nan[R] == Eq]
# Convert to N
y = df_no_Nan[Fmax][df_no_Nan[R] == Eq]

x_plot = np.linspace(min(x), max(x), 100)

slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
ax.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
ax.text(400, yfit(400) - 1, r'$r={:.4f}$'.format(r_value1), color='r',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

print('Values for correlation between ' +
      'Elastic Modulus and Max Force in the Equator\n',
      'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value1))

# linear regressions for fitting
x = df_no_Nan[EV][df_no_Nan[R] == Po]
y = df_no_Nan[Fmax][df_no_Nan[R] == Po]

x_plot = np.linspace(min(x), max(x), 100)
slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
ax.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
ax.text(220, yfit(220) - 1, r'$r={:.4f}$'.format(r_value2), color='b',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

print('Values for correlation between ' +
      'Elastic Modulus and Max Force in the Equator\n',
      'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value2))

# Axis limits
ax.set(ylim=(0, 18))
ax.set(xlim=(0, 500))

# New path
NP = os.path.join(SF, Folder)

# Create folder if it doesn't exist
os.makedirs(NP, exist_ok=True)

plt.savefig(os.path.join(NP, 'Regression_MaxForce_by_Region.pdf'),
           bbox_inches='tight')
plt.close()


# In[Elastic Modulus Regression by Steady-State Peel Force in both regions]

FSS = 'SimSS'

# Linear regression
f, ax = plt.subplots()
sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                             "axes.labelsize":12})
ax = sns.lmplot(data=md, x=EV, y=FSS, hue=R, hue_order=[Eq, Po],
                markers=["o", "x"], legend_out=False, fit_reg=True,
```

```
                height=5, aspect=1.6, palette="Set1", truncate=True, ci=95,
                line_kws={'lw':0})

ax.set(xlabel=ElasticUnit, ylabel=SSPF)

ax = ax.axes[0][0] # Convert faceted grid to matplotlib fig

# Remove all NaN's from the data for regressions
# remove nans from ILM thickness & Max
df_no_Nan = md.dropna(subset=[FSS, EV])

# linear regressions for fitting
x = df_no_Nan[EV][df_no_Nan[R] == Eq]
# Convert to N
y = df_no_Nan[FSS][df_no_Nan[R] == Eq]

x_plot = np.linspace(min(x), max(x), 100)

slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
ax.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
ax.text(400, yfit(400) - 1, r'$r={:.4f}$'.format(r_value1), color='r',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

print('Values for correlation between ' +
      'Elastic Modulus and SS Force in the Equator\n',
      'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value1))

# linear regressions for fitting
x = df_no_Nan[EV][df_no_Nan[R] == Po]
y = df_no_Nan[FSS][df_no_Nan[R] == Po]

x_plot = np.linspace(min(x), max(x), 100)
slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
ax.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
ax.text(220, yfit(220) - 1, r'$r={:.4f}$'.format(r_value2), color='b',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

print('Values for correlation between ' +
      'Elastic Modulus and SS Force in the Equator\n',
      'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value2))

# Axis limits
ax.set(ylim=(0, 12))
ax.set(xlim=(0, 500))

# New path
NP = os.path.join(SF, 'ElasticModulus')

# Create folder if it doesn't exist
os.makedirs(NP, exist_ok=True)

plt.savefig(os.path.join(NP, 'Regression_SSForce_by_Region.pdf'),
           bbox_inches='tight')
plt.close()

# In[Cohesive parameter group plots]

Knn = 'Knn'
```

```
Kss = 'Kss'
Ktt = 'Ktt'

Folder = 'CohesiveBehavior'

# Filter data (Brittany)
dfKnn = md[md[Knn] > 25e6]
dfKss = md[md[Kss] > 100e6]
dfKtt = md[md[Ktt] > 150e6]

dfFilt = {Knn: dfKnn,
          Kss: dfKss,
          Ktt: dfKtt}

for key, val in dfFilt.items():
    pivotCohBeh = pd.pivot_table(val, values=key, index=[A60, R],
                                                  aggfunc=pvtOut)

    print('pivotCohBeh')
    print(pivotCohBeh)
    # Add the index groups and convert NaN's to "-"'s
    print(pivotCohBeh.to_latex(index=True, na_rep='-', escape=False,
                               float_format="{:0.3f}".format))

    # Barplot
    smartPlot(data=val, x=A60, y=key, hue=R, hue_order=[Eq, Po], ci=68,
              errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
              sigLoc='outside', text_format='star', line_offset=0.015,
              line_offset_to_box=0.0, line_height=0.015, fontsize='small',
              legLoc='best', verbose=2, yAxis='log',
              xlabel=A_G, ylabel=CohBehUnit, legendTitle=R,
              figName=f'Region_BarPlot_{key}', folderName=Folder)

    # Boxplot
    smartPlot(data=val, x=A60, y=key, hue=R, hue_order=[Eq, Po],
              plot='boxplot',
              test='t-test_ind', sigLoc='outside', text_format='star',
              line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
              fontsize='small', legLoc='best', verbose=2, yAxis='log',
              xlabel=A_G, ylabel=CohBehUnit,
              legendTitle=R, figName=f'Region_BoxPlot_{key}',
              folderName=Folder)

    # Boxplot with data
    smartPlot(data=val, x=A60, y=key, hue=R, hue_order=[Eq, Po],
              plot='boxplot',
              test='t-test_ind', sigLoc='outside', text_format='star',
              line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
              fontsize='small', legLoc='best', verbose=2, yAxis='log',
              xlabel=A_G, ylabel=CohBehUnit,
              legendTitle=R,
              figName=f'Region_BoxPlotWithData_{key}', folderName=Folder,
              dataPoints=True)

# In[Kss Regression by Max Peel Force in both regions]

Fmax = 'SimMax'
Folder = 'CohesiveBehavior'
```

```python

# Linear regression
f, ax = plt.subplots()
sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                             "axes.labelsize":12})
ax = sns.lmplot(data=dfKnn, x='Kss', y=Fmax, hue=R, hue_order= [Eq, Po],
                markers=["o", "x"], legend_out=False, fit_reg=True,
                height=5, aspect=1.6, palette="Set1", truncate=True, ci=95,
                line_kws={'lw':0})

ax.set(xscale="log")

ax.set(xlabel=CohBehUnit, ylabel=MPF)

ax = ax.axes[0][0] # Convert faceted grid to matplotlib fig

# Remove all NaN's from the data for regressions
# remove nans from ILM thickness & Max
df_no_Nan = dfKnn.dropna(subset=[Fmax, 'Kss'])

# linear regressions for fitting
x = df_no_Nan['Kss'][df_no_Nan[R] == Eq]
# Convert to N
y = df_no_Nan[Fmax][df_no_Nan[R] == Eq]

x_plot = np.linspace(min(x), max(x), 100)

slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
ax.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
ax.text(5e7, yfit(5e7) + 1, r'$r={:.4f}$'.format(r_value1), color='r',
        horizontalalignment='left', fontsize=8, weight='semibold') # r value

print('Values for correlation between ' +
      'Kss and Max Force in the Equator\n',
      'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value1))

# linear regressions for fitting
x = df_no_Nan['Kss'][df_no_Nan[R] == Po]
y = df_no_Nan[Fmax][df_no_Nan[R] == Po]

x_plot = np.linspace(min(x), max(x), 100)
slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
ax.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
ax.text(6e7, yfit(6e7) + 1, r'$r={:.4f}$'.format(r_value2), color='b',
        horizontalalignment='left', fontsize=8, weight='semibold') # r value

print('Values for correlation between ' +
      'Kss and Max Force in the Equator\n',
      'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value2))

# Axis limits
# ax.set(ylim=(0, 18))
ax.set(xlim=(3e7, 9e7))

# New path
NP = os.path.join(SF, Folder)

# Create folder if it doesn't exist
```

```
os.makedirs(NP, exist_ok=True)

plt.savefig(os.path.join(NP, 'Kss_vs_MaxForce_by_Region.pdf'),
            bbox_inches='tight')
plt.close()

# In[Kss Regression by Steady State Peel Force in both regions]

FSS = 'SimSS'
Folder = 'CohesiveBehavior'

# Linear regression
f, ax = plt.subplots()
sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                             "axes.labelsize":12})
ax = sns.lmplot(data=dfKnn, x='Kss', y=FSS, hue=R, hue_order= [Eq, Po],
                markers=["o", "x"], legend_out=False, fit_reg=True,
                height=5, aspect=1.6, palette="Set1", truncate=True, ci=95,
                line_kws={'lw':0})

ax.set(xscale="log")

ax.set(xlabel=CohBehUnit, ylabel=SSPF)

ax = ax.axes[0][0] # Convert faceted grid to matplotlib fig

# Remove all NaN's from the data for regressions
# remove nans from ILM thickness & Max
df_no_Nan = dfKnn.dropna(subset=[FSS, 'Kss'])

# linear regressions for fitting
x = df_no_Nan['Kss'][df_no_Nan[R] == Eq]
# Convert to N
y = df_no_Nan[FSS][df_no_Nan[R] == Eq]

x_plot = np.linspace(min(x), max(x), 100)

slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
ax.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
ax.text(5e7, yfit(5e7) + 1, r'$r={:.4f}$'.format(r_value1), color='r',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

print('Values for correlation between ' +
      'Kss and Steady State Peel Force in the Equator\n',
      'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value1))

# linear regressions for fitting
x = df_no_Nan['Kss'][df_no_Nan[R] == Po]
y = df_no_Nan[FSS][df_no_Nan[R] == Po]

x_plot = np.linspace(min(x), max(x), 100)
slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
ax.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
ax.text(6e7, yfit(6e7) + 1, r'$r={:.4f}$'.format(r_value2), color='b',
         horizontalalignment='left', fontsize=8, weight='semibold') # r value

print('Values for correlation between ' +
      'Kss and Steady State Peel Force in the Equator\n',
```

```
      'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value2))

# Axis limits
# ax.set(ylim=(0, 18))
ax.set(xlim=(3e7, 9e7))

# New path
NP = os.path.join(SF, Folder)

# Create folder if it doesn't exist
os.makedirs(NP, exist_ok=True)

plt.savefig(os.path.join(NP, 'Kss_vs_SSForce_by_Region.pdf'),
           bbox_inches='tight')
plt.close()

# In[Cohesive Damage Initiation parameter group plots]

tn = 'tn'
ts = 'ts'
tt = 'tt'

Folder = 'CohesiveDamage'

# Filter data (Brittany)
dftn = md[md[tn] < 3000]
dfts = md[md[ts] < 3000]
dftt = md[md[tt] < 3000]

dfFilt = {tn: dftn,
          ts: dfts,
          tt: dftt}

for key, val in dfFilt.items():
    pivotCohDMG = pd.pivot_table(val, values=key, index=[A60, R],
                                             aggfunc=pvtOut)

    print('pivotCohDMG')
    print(pivotCohDMG)
    # Add the index groups and convert NaN's to "-"'s
    print(pivotCohDMG.to_latex(index=True, na_rep='-', escape=False,
                               float_format="{:0.3f}".format))

    # Barplot
    smartPlot(data=val, x=A60, y=key, hue=R, hue_order=[Eq, Po], ci=68,
              errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
              sigLoc='outside', text_format='star', line_offset=0.015,
              line_offset_to_box=0.0, line_height=0.015, fontsize='small',
              legLoc='best', verbose=2, yAxis=None,
              xlabel=A_G, ylabel=CohDMGUnit, legendTitle=R,
              figName=f'Region_BarPlot_{key}', folderName=Folder)

    # Boxplot
    smartPlot(data=val, x=A60, y=key, hue=R, hue_order=[Eq, Po],
              plot='boxplot', test='t-test_ind', sigLoc='outside',
              text_format='star', line_offset=0.015, line_offset_to_box=0.0,
              line_height=0.015,
              fontsize='small', legLoc='best', verbose=2, yAxis=None,
```

```
                xlabel=A_G, ylabel=CohDMGUnit,
                legendTitle=R, figName=f'Region_BoxPlot_{key}', folderName=Folder)

    # Boxplot with data
    smartPlot(data=val, x=A60, y=key, hue=R, hue_order=[Eq, Po],
                plot='boxplot', test='t-test_ind', sigLoc='outside',
                text_format='star', line_offset=0.015, line_offset_to_box=0.0,
                line_height=0.015,
                fontsize='small', legLoc='best', verbose=2, yAxis=None,
                xlabel=A_G, ylabel=CohDMGUnit,
                legendTitle=R,
                figName=f'Region_BoxPlotWithData_{key}', folderName=Folder,
                dataPoints=True)

# In[ts Regression by Max Peel Force in both regions]


mod = sm.OLS(df['VE'], df[Fmax])
res = mod.fit()
print(80*'-', 2*'\n', 'Correlation between Age & E\n', res.summary())

text_file = open('Correlation.txt', "w")
text_file.write(res.summary().as_text())
text_file.close()


stats.ttest_ind(df['VE'], df[Fmax])


Fmax = 'SimMax'
Folder = 'CohesiveDamage'

# Linear regression
f, ax = plt.subplots()
sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                             "axes.labelsize":12})
ax = sns.lmplot(data=dfts, x=ts, y=Fmax, hue=R, hue_order= [Eq, Po],
                  markers=["o", "x"], legend_out=False, fit_reg=True,
                  height=5, aspect=1.6, palette="Set1", truncate=True, ci=95,
                  line_kws={'lw':0})

# ax.set(xscale="log")

ax.set(xlabel=CohDMGUnit, ylabel=MPF)

ax = ax.axes[0][0] # Convert faceted grid to matplotlib fig

# Remove all NaN's from the data for regressions
# remove nans from ILM thickness & Max
df_no_Nan = dfts.dropna(subset=[Fmax, ts])

# linear regressions for fitting
x = df_no_Nan[ts][df_no_Nan[R] == Eq]
# Convert to N
y = df_no_Nan[Fmax][df_no_Nan[R] == Eq]

x_plot = np.linspace(min(x), max(x), 100)

```

```
xt = 3*10**2 # Location of text

slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
ax.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
plt.text(xt, yfit(xt) + 1, r'$r={:.4f}$'.format(r_value1), color='r',
          horizontalalignment='left', fontsize=8, weight='semibold') # r value

print('Values for correlation between ' +
      'ts and Max Force in the Equator\n',
      'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value1))

# linear regressions for fitting
x = df_no_Nan[ts][df_no_Nan[R] == Po]
y = df_no_Nan[Fmax][df_no_Nan[R] == Po]

xt = 8*10**2 # Location of text

x_plot = np.linspace(min(x), max(x), 100)
slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
ax.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
plt.text(xt, yfit(xt) + 1, r'$r={:.4f}$'.format(r_value2), color='b',
          horizontalalignment='left', fontsize=8, weight='semibold') # r value

print('Values for correlation between ' +
      'ts and Max Force in the Equator\n',
      'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value2))

# Axis limits
# ax.set(ylim=(0, 18))
# ax.set(xlim=(3e7, 9e7))
ax.set(xlim=(1.8*10**2, 1.3*10**3))

# New path
NP = os.path.join(SF, Folder)

# Create folder if it doesn't exist
os.makedirs(NP, exist_ok=True)

plt.savefig(os.path.join(NP, 'ts_vs_MaxForce_by_Region.pdf'),
           bbox_inches='tight')
plt.close()

# In[tt Regression by Max Peel Force in both regions]

Fmax = 'SimMax'
Folder = 'CohesiveDamage'

# Linear regression
f, ax = plt.subplots()
sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                             "axes.labelsize":12})
ax = sns.lmplot(data=dftt, x=tt, y=Fmax, hue=R, hue_order= [Eq, Po],
                markers=["o", "x"], legend_out=False, fit_reg=True,
                height=5, aspect=1.6, palette="Set1", truncate=True, ci=95,
                line_kws={'lw':0})

# ax.set(xscale="log")
```

```
ax.set(xlabel=CohDMGUnit, ylabel=MPF)

ax = ax.axes[0][0] # Convert faceted grid to matplotlib fig

# Remove all NaN's from the data for regressions
# remove nans from ILM thickness & Max
df_no_Nan = dftt.dropna(subset=[Fmax, tt])

# linear regressions for fitting
x = df_no_Nan[tt][df_no_Nan[R] == Eq]
# Convert to N
y = df_no_Nan[Fmax][df_no_Nan[R] == Eq]

x_plot = np.linspace(min(x), max(x), 100)

xt = 2*10**2 # Location of text

slope, intercept, r_value1, p_value, std_err = stats.linregress(x, y)
ax.plot(x_plot, yfit(x_plot), '-', color='r', linewidth=1, label='line')
ax.text(xt, yfit(xt) + 2, r'$r={:.4f}$'.format(r_value1), color='r',
          horizontalalignment='left', fontsize=8, weight='semibold') # r value

print('Values for correlation between ' +
      'tt and Max Force in the Equator\n',
      'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value1))

# linear regressions for fitting
x = df_no_Nan[tt][df_no_Nan[R] == Po]
y = df_no_Nan[Fmax][df_no_Nan[R] == Po]

xt = 1.2*10**3 # Location of text

x_plot = np.linspace(min(x), max(x), 100)
slope, intercept, r_value2, p_value, std_err = stats.linregress(x, y)
ax.plot(x_plot, yfit(x_plot), '-', color='b', linewidth=1, label='line')
ax.text(xt, yfit(xt) + 1, r'$r={:.4f}$'.format(r_value2), color='b',
          horizontalalignment='left', fontsize=8, weight='semibold') # r value

print('Values for correlation between ' +
      'tt and Max Force in the Equator\n',
      'P={:.4f}'.format(p_value), 'r={:.4f}'.format(r_value2))

# Axis limits
ax.set(ylim=(0, None))
ax.set(xlim=(1.*10**1, 2*10**3))

# New path
NP = os.path.join(SF, Folder)

# Create folder if it doesn't exist
os.makedirs(NP, exist_ok=True)

plt.savefig(os.path.join(NP, 'tt_vs_MaxForce_by_Region.pdf'),
           bbox_inches='tight')
plt.close()

# In[tt Regression by Age regions]

```

```
Folder = 'CohesiveDamage'

Aleq63 = r'Age $\leq$ 63'
Ag63 = 'Age $>$ 63'

bins = [0, 63, 90]
labelsAge63 = [Aleq63, Ag63]

# Create binned AgeGroups
A63 = 'A63'
md[A63] = pd.cut(md[A], bins, labels=labelsAge63, right=True)

dftt = md[md[tt] < 3000]

# Linear regression
f, ax = plt.subplots()
# sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
#                               "axes.labelsize":12})

# scatter plot
# Averages for the +/- 63 age group
y_less_63 = dftt[tt][(dftt[A] <= 63)].dropna()
y_greater_63 = dftt[tt][(dftt[A] > 63)].dropna()

f, ax = plt.subplots()
ax = sns.scatterplot(data=dftt, x=A, y=tt, hue=A63, style=A63,
                     hue_order=None,
                     palette='Set1', s=200, legend=True)

ax.get_legend_handles_labels()[0][0]._sizes = [200.]
ax.get_legend_handles_labels()[0][1]._sizes = [200.]

legend = ax.legend(loc='best', fontsize=18, title=A_G)

plt.setp(legend.get_title(), fontsize=18)

# Axis labels
ax.set_xlabel(A_yrs, fontsize=18)
ax.set_ylabel(CohDMGUnit, fontsize=18)


x_plot_less_63 = np.linspace(30, 63, 100)
x_plot_greater_63 = np.linspace(63, 80, 100)

# Plot averages
plt.plot(x_plot_less_63, np.mean(y_less_63)*np.ones(len(x_plot_less_63)),
         '-.', color='r', linewidth=3) # , label=r'Age $\leq$ 60 AVG')

ax.text(np.mean(x_plot_less_63)*1.1, np.mean(y_less_63) + 50,
        r'Average', color='r', horizontalalignment='left',
        fontsize=18, weight='semibold')

plt.plot(x_plot_greater_63,
         np.mean(y_greater_63)*np.ones(len(x_plot_greater_63)), '-.',
         color='b', linewidth=3) #, label=r'Age $>$ 60 AVG')

ax.text(np.mean(x_plot_greater_63)*0.9, np.mean(y_greater_63) - 75,
        r'Average', color='b', horizontalalignment='left',
```

```
        fontsize=18, weight='semibold')

ax.tick_params(axis='x', labelsize=14)
ax.tick_params(axis='y', labelsize=14)

# New path
NP = os.path.join(SF, Folder)

# Create folder if it doesn't exist
os.makedirs(NP, exist_ok=True)

plt.savefig(os.path.join(NP, 'tt_vs_Age.pdf'), bbox_inches='tight')

plt.close()
# In[Fracture Energy group plots]

FE = 'FE'

Folder = 'FractureEnergy'

# Filter data (Brittany)
dfFE = md[md[FE] < 0.0009]

dfFilt = {FE: dfFE}

for key, val in dfFilt.items():

    pivotFE = pd.pivot_table(val, values=key, index=[A60, R],
                                                 aggfunc=pvtOut)

    print('pivotFE')
    print(pivotFE)
    # Add the index groups and convert NaN's to "-"'s
    print(pivotFE.to_latex(index=True, na_rep='-', escape=False,
                           float_format="{:0.3f}".format))

    # Barplot
    smartPlot(data=val, x=A60, y=key, hue=R, hue_order=[Eq, Po], ci=68,
              errcolor='black', capsize=.2, plot='barplot', test='t-test_ind',
              sigLoc='outside', text_format='star', line_offset=0.015,
              line_offset_to_box=0.0, line_height=0.015, fontsize='small',
              legLoc='best', verbose=2, yAxis='log',
              xlabel=A_G, ylabel=FEUnit, legendTitle=R,
              figName='Region_BarPlot', folderName=Folder)

    # Boxplot
    smartPlot(data=val, x=A60, y=key, hue=R, hue_order=[Eq, Po], plot='boxplot',
              test='t-test_ind', sigLoc='outside', text_format='star',
              line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
              fontsize='small', legLoc='best', verbose=2, yAxis='log',
              xlabel=A_G, ylabel=FEUnit,
              legendTitle=R, figName='Region_BoxPlot', folderName=Folder)

    # Boxplot with data
    smartPlot(data=val, x=A60, y=key, hue=R, hue_order=[Eq, Po], plot='boxplot',
              test='t-test_ind', sigLoc='outside', text_format='star',
              line_offset=0.015, line_offset_to_box=0.0, line_height=0.015,
              fontsize='small', legLoc='best', verbose=2, yAxis='log',
```

```
                xlabel=A_G, ylabel=FEUnit,
                legendTitle=R,
                figName='Region_BoxPlotWithData', folderName=Folder,
                dataPoints=True)


# In[Summary Convergence Table Elastic]

# value summary
simList = []

for key, val in ElasticSummary.items():
    d = val[1:] # Subset - skip first row
    d['simTime'] = pd.to_numeric(d['simTime'], downcast="float")

    L = len(d.index)
    s = np.sum(d['simTime'])
    avg = s/L

    simList.append([L, s, avg])

simDF = pd.DataFrame(simList, columns=['N', 'TotalTime', 'AVGTime'])

print(np.mean(simDF['N']),
      np.mean(simDF['TotalTime']),
      np.mean(simDF['AVGTime']))

# In[Summary Convergence Table Cohesive]

# value summary
simList = []

for key, val in CohesiveSummary.items():
    d = val[1:] # Subset - skip first row
    d['simTime'] = pd.to_numeric(d['simTime'], downcast="float")

    L = len(d.index)
    s = np.sum(d['simTime'])
    avg = s/L

    simList.append([L, s, avg])

simDF = pd.DataFrame(simList, columns=['N', 'TotalTime', 'AVGTime'])

print(np.mean(simDF['N']),
      np.mean(simDF['TotalTime']),
      np.mean(simDF['AVGTime']))
```

1.8.2 Visualization Distributions

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

```
# -*- coding: utf-8 -*-
"""
Created on Wed Apr 28 23:59:51 2021

```

```python
@author: Kiffer
"""

import pandas as pd
import numpy as np
import seaborn as sns
from statannot import add_stat_annotation
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [16, 10]
from scipy import stats
import pdb
import os
import glob
import re
from scipy import stats

cwd = os.getcwd()

SF = os.path.join('Results', 'StatisticsFigures')

# Create folder if it doesn't exist
os.makedirs(SF, exist_ok=True)

# In[KDE plot function]

def KDEplot(data=None, x=None, hue=None, hue_order=None,
            Regions=None, figName=None, legendTitle=None, legendLoc=None,
            xlabel=None, ylabel=None, bw_adjust=None, alpha=None,
            initGuess=None, constraints=None, folderName=None,
            optLegendLoc=None, bounds=None):
    colors = [plt.cm.tab10.colors[i:i + 2] for i in
              range(0, len(data[R].unique()) * 2, 2)]
    hatches = ['', '//////']

    f, ax = plt.subplots(figsize=(9.6, 6))
    sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                                 "axes.labelsize":12})
    handles = []
    for region, palette in zip(Regions, colors):

        # Data subset
        dataSubset = data[(data[R] == region)]

        # KDE plot
        ax = sns.kdeplot(data=dataSubset, x=x, hue=hue,
                         hue_order=hue_order, multiple='stack', fill=True,
                         palette=palette, ax=ax, log_scale=True,
                         alpha=alpha, bw_adjust=bw_adjust)
        # pdb.set_trace()
        for h, age, hatch in zip(ax.legend_.legendHandles, hue_order,
                                 hatches):
            h.set_label(f'{region}, {age}')
            h.set_hatch(hatch)
            handles.append(h)


    extra = []
    if initGuess != None:
```

```
        ax.axvline(x = initGuess, color='black', linestyle='-',
                   linewidth=1, label=r'Initial Guess')
        extra.append(0)

    if bounds != None and bounds == True:
        # Only add bounds if "True"
        if constraints != None:
            ax.axvline(x = constraints[0], color='black', linestyle=':',
                       linewidth=1, label=r'Lower Bound', ymax=0.4)
            extra.append(1)
            ax.axvline(x = constraints[1], color='black', linestyle='--',
                       linewidth=1, label=r'Upper Bound', ymax=0.4)
            extra.append(2)

    ax.legend_.remove() # remove the automatic legends
    ax.set(xlabel=xlabel, ylabel=ylabel)
    for collection, hatch in zip(ax.collections[::-1],
                                 hatches * len(Regions)):
        collection.set_hatch(hatch)

    # Add bounds
    if initGuess != None and optLegendLoc !=None:
        lines = f.gca().get_lines()
        # pdb.set_trace()
        legend2 = ax.legend([lines[i] for i in extra],
                            [lines[i].get_label() for i in extra],
                            prop={"size":10}, loc=optLegendLoc,
                            title='Optimization')
        ax.add_artist(legend2)

    legend1 = ax.legend(handles=handles, loc=legendLoc).set_title(legendTitle)

    # This doesn't work using the method so the bounds need to be plotted
    # before the custom legend with handles
    # plt.gca().add_artist(legend1)

    if folderName != None:
        # If a new folder name is given, put the files there

        # New file path
        NP = os.path.join(SF, folderName)

        # Create folder if it doesn't exist
        os.makedirs(NP, exist_ok=True)

    else:
        # Put the file in the same folder
        NP = SF

    f.savefig(os.path.join(NP, f'{figName}_{x}.pdf'),
                    bbox_inches='tight')

    plt.close(f)

    return f, ax


# In[Plot simplifications]
```

```
R = 'Region'
Eq = 'Equator'
Po = 'Posterior'
AG = 'AgeGroup'
A60 = 'Age60'
Aleq60 = r'Age $\leq$ 60'
Ag60 = 'Age $>$ 60'
A = 'Age'

# Units
MPF = 'Maximum Peel Force (mN)'
SSPF = 'Steady-State Peel Force (mN)'
KDEUnit = r'Kernel Density Estimation'
ElasticUnit = r'Elastic Modulus (Pa)'
CohBehUnit = r'Cohesive Behavior (Pa)'
CohDMGUnit = r'Cohesive Damage Initiation (Pa)'
FEUnit = r'Fracture Energy (J)'

A_yrs = 'Age (yr.)'
A_G = 'Age Group (yr.)'

# In[Load data]

data = os.path.join(cwd, 'Results', 'OutputFiles', 'ExpSimSummary.txt')
df = pd.read_csv(data, sep = '\t')

df.loc[df[R] == 'Eq.', R] = Eq
df.loc[df[R] == 'Po.', R] = Po

# In[Elastic Modulus distribution]

Folder = 'ElasticModulus'

# Normal distribution plots
f, ax = plt.subplots(figsize=(9.6, 6))
sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
                             "axes.labelsize":12})
ax = sns.kdeplot(data=df, x='EV', fill=True, legend=False, palette='Paired',
                 cut=0, bw_adjust=0.5)

ax.set(xlabel=ElasticUnit, ylabel=KDEUnit)

# New path
NP = os.path.join(SF, Folder)

# Make folder if it doesn't exist
os.makedirs(NP, exist_ok=True)

plt.savefig(os.path.join(NP, 'Distribution_AllData.pdf'),
            bbox_inches='tight')

# In[Distributions of cohesive parameters]

alpha = 0.4
bw_adjust = 0.8 # Normal distribution smoothing (smaller is less smooth)
figName = 'Optimization'
F1 = 'ElasticModulus'
```

```
F2 = 'CohesiveBehavior'
F3 = 'CohesiveDamage'
F4 = 'FractureEnergy'

# Bandwidth selector
bwE = 1.06*np.std(df['EV'])*np.count_nonzero(df['EV'])**(-1/5)
bwE = 0.9*min(np.std(df['EV']),
              stats.iqr(df['EV']))*np.count_nonzero(df['EV'])**(-1/5)

bwKnn = 0.9*min(np.std(df['Knn']),
                stats.iqr(df['Knn']))*np.count_nonzero(df['Knn'])**(-1/5)

# Elastic Modulus
f, ax = KDEplot(data=df, x='EV', hue=A60, hue_order=[Aleq60, Ag60],
                Regions=[Eq, Po], figName=figName, folderName=F1,
                legendLoc='best', legendTitle=R,
                xlabel=ElasticUnit, ylabel=KDEUnit,
                bw_adjust=bw_adjust, alpha=alpha,
                initGuess=172, constraints=[50, 2100],
                optLegendLoc='center right', bounds=False)

# Cohesive Behavior
f, ax = KDEplot(data=df, x='Knn', hue=A60, hue_order=[Aleq60, Ag60],
                Regions=[Eq, Po], figName=figName, folderName=F2,
                legendLoc='best', legendTitle=R,
                xlabel=CohBehUnit, ylabel=KDEUnit,
                bw_adjust=bw_adjust, alpha=alpha,
                initGuess=2**20.872765304828103, constraints=[2**10, 2**28],
                optLegendLoc='center right', bounds=False)

f, ax = KDEplot(data=df, x='Kss', hue=A60, hue_order=[Aleq60, Ag60],
                Regions=[Eq, Po], figName=figName, folderName=F2,
                legendLoc='best', legendTitle=R,
                xlabel=CohBehUnit, ylabel=KDEUnit,
                bw_adjust=bw_adjust, alpha=alpha,
                initGuess=2**26.094732037712763, constraints=[2**10, 2**28],
                optLegendLoc='center right', bounds=False)

f, ax = KDEplot(data=df, x='Ktt', hue=A60, hue_order=[Aleq60, Ag60],
                Regions=[Eq, Po], figName=figName, folderName=F2,
                legendLoc='best', legendTitle=R,
                xlabel=CohBehUnit, ylabel=KDEUnit,
                bw_adjust=bw_adjust, alpha=alpha,
                initGuess=2**26.20110650892766, constraints=[2**10, 2**28],
                optLegendLoc='center right', bounds=False)

# Damage Initiation
f, ax = KDEplot(data=df, x='tn', hue=A60, hue_order=[Aleq60, Ag60],
                Regions=[Eq, Po], figName=figName, folderName=F3,
                legendLoc='best', legendTitle=R,
                xlabel=CohDMGUnit, ylabel=KDEUnit,
                bw_adjust=bw_adjust, alpha=alpha,
                initGuess=2**9.712181223168551, constraints=[2**3, 2**20],
                optLegendLoc='center right', bounds=False)

f, ax = KDEplot(data=df, x='ts', hue=A60, hue_order=[Aleq60, Ag60],
                Regions=[Eq, Po], figName=figName, folderName=F3,
                legendLoc='best', legendTitle=R,
```

```
                xlabel=CohDMGUnit, ylabel=KDEUnit,
                bw_adjust=bw_adjust, alpha=alpha,
                initGuess=2**9.931687876075074, constraints=[2**3, 2**20],
                optLegendLoc='center right', bounds=False)

f, ax = KDEplot(data=df, x='tt', hue=A60, hue_order=[Aleq60, Ag60],
                Regions=[Eq, Po], figName=figName, folderName=F3,
                legendLoc='best', legendTitle=R,
                xlabel=CohDMGUnit, ylabel=KDEUnit,
                bw_adjust=bw_adjust, alpha=alpha,
                initGuess=2**9.022372079206395, constraints=[2**3, 2**20],
                optLegendLoc='center right', bounds=False)

# Fracture Energy
f, ax = KDEplot(data=df, x='FE', hue=A60, hue_order=[Aleq60, Ag60],
                Regions=[Eq, Po], figName=figName, folderName=F4,
                legendLoc='best', legendTitle=R,
                xlabel=FEUnit, ylabel=KDEUnit,
                bw_adjust=bw_adjust, alpha=alpha,
                initGuess=3.738925970000001e-6, constraints=[2**-30, 2**0],
                optLegendLoc='center right', bounds=False)

# In[Stack overflow]

# fig, axs = plt.subplots(ncols=3, figsize=(15, 3), sharex=True, sharey=True)
# f, axs = plt.subplots()
# colors = [plt.cm.tab20.colors[i:i + 2] for i in range(0, len(df_CohDmg['Region'].unique()) * 2, 2)]
# hatches = ['', '//']
# for ax, coh_dmg in zip(axs, ['tn', 'ts', 'tt']):
#     handles = []
    # for region, palette in zip([Eq, Po], colors):
    #     sns.kdeplot(data=df_CohDmg[(df_CohDmg['CohDmg'] == coh_dmg) &
    #                                (df_CohDmg['Region'] == region)],
    #                 x='value', hue='Age60', hue_order=[Aleq60, Ag60],
    #                 multiple='stack', palette=palette, ax=ax, log_scale=True,)
    #     for h, age, hatch in zip(ax.legend_.legendHandles, [Aleq60, Ag60], hatches):
    #         h.set_label(f'{region}, {age}')
    #         h.set_hatch(hatch)
    #         handles.append(h)
    # ax.legend_.remove() # remove the automatic legends
    # ax.set_title(f'CohDmg={coh_dmg}')
    # for collection, hatch in zip(ax.collections[::-1], hatches * len([Eq, Po])):
    #     collection.set_hatch(hatch)

    # ax.legend(handles=handles, loc='best')
    # plt.tight_layout()

    # fig.savefig(os.path.join(SF, f'StackOverflow_{coh_dmg}.pdf'),
    #                 bbox_inches='tight')

# In[Successful KDF plot example]
# colors = [plt.cm.tab20.colors[i:i + 2] for i in
#                 range(0, len(df_CohDmg['Region'].unique()) * 2, 2)]
# hatches = ['', '//']

# for i in ['tn', 'ts', 'tt']:
```

```
#     f, ax = plt.subplots(figsize=(8, 5))
#     sns.set_context("paper", rc={"font.size":12, "axes.titlesize":8,
#                                  "axes.labelsize":12})
#     handles = []
#     for region, palette in zip([Eq, Po], colors):
#         ax = sns.kdeplot(data=df_CohDmg[(df_CohDmg['CohDmg'] == i) &
#                                         (df_CohDmg['Region'] == region)],
#                          x='value', hue='Age60', hue_order=[Aleq60, Ag60],
#                          multiple='stack', palette=palette, ax=ax,
#                          log_scale=True, alpha=0.9, bw_adjust=0.5)
#         for h, age, hatch in zip(ax.legend_.legendHandles, [Aleq60, Ag60],
#                                  hatches):
#             h.set_label(f'{region}, {age}')
#             h.set_hatch(hatch)
#             handles.append(h)
#     ax.legend_.remove() # remove the automatic legends
#     ax.set(xlabel='Elastic Modulus [Pa]',
#            ylabel='Kernel Density Estimation')
#     for collection, hatch in zip(ax.collections[::-1],
#                                  hatches * len([Eq, Po])):
#         collection.set_hatch(hatch)

#     ax.legend(handles=handles, loc='best')

#     f.savefig(os.path.join(SF, f'StackOverflow_{i}.pdf'),
#                       bbox_inches='tight')
```