Supplemental Contents: Python Scripts

Table of Contents

Please note: The highlighted text on page 1 requires custom modification according to the patient number and where the data is located in the directory of the computer.

1. Model_setup_Part1.py

```
#

#

# Part 1 imports the model from a .inp Abaqus file and sets up functions


from part import *

from material import *

from section import *

from assembly import *

from step import *

from interaction import *

from load import *

from mesh import *

from optimization import * from

job import * from sketch import

* from visualization import *

from connectorBehavior import *


from abaqus import getInput


# Import message box modules

import ctypes

MessageBox = ctypes.windll.user32.MessageBoxA


# Set up random list

import random

random_list = ['right', 'left']


# Define directories

fields = (('Model directory:','C:/Users/gandhi/Desktop/Recent Model/Recent model/Pt6/'),

   ('Plugin directory', 'C:/SIMULIA/CAE/2017/win_b64/code/python2.7/lib/abaqus_plugins/findNearestNode'),

   ('Script directory', 'C:/Users/gandhi/Desktop/Recent Model/Recent Scripts/'))

directory, plugin_dir, script_dir = getInputs(fields=fields, label='Specify file paths', dialogTitle='Define directories', )
```

```
# Import parts
model_name = getInput('Model file:', 'Pt2.inp')
model_file='{}{}'.format(directory, model_name)

mdb.models.changeKey(fromName='Model-1', toName='Simulation')
mdb.models['Simulation'].PartFromInputFile(
  inputFileName=model_file)

# Set model variables
sim_model = mdb.models['Simulation']
max_part  = sim_model.parts['MAXILLA']

# Create element sets
max_elems = max_part.elements[:]
max_elems_set = max_part.Set(elements=max_elems, name='MAXILLA_elem_set')

for i in range(1,8):
  p = sim_model.parts['UL{}'.format(i)]
  elems = p.elements[:]
  p.Set(elements=elems, name='UL{}_elem_set'.format(i))
  p = sim_model.parts['UL{}_PDL'.format(i)]
  elems = p.elements[:]
  p.Set(elements=elems, name='UL{}_PDL_elem_set'.format(i))
  p = sim_model.parts['UR{}'.format(i)]
  elems = p.elements[:]
  p.Set(elements=elems, name='UR{}_elem_set'.format(i))
  p = sim_model.parts['UR{}_PDL'.format(i)]
  elems = p.elements[:]
  p.Set(elements=elems, name='UR{}_PDL_elem_set'.format(i))

# User manually creates surfaces of interest (sockets, inner/outer PDL, teeth)
MessageBox(None, 'Please create socket, PDL, and tooth surfaces.\nRun Model_setup_Part2 when finished.', 'Model_setup_Part1 Completed',
0)
```

2. Model_setup_Part2.py

```
#
#
# Part 2 assigns material definitions and creates all instances needed for simulation

# Run function List

def fx_list():
    file_name = 'Functions.py'
    file_path = script_dir + file_name
    execfile(file_path, main .__dict )

fx_list()

# Create materials
sim_model.Material(name='Bone')
sim_model.materials['Bone'].Elastic(table=((17000.0, 0.3), ))
sim_model.materials['Bone'].Density(table=((1.85e-09, ), ))
sim_model.Material(name='Tooth')
sim_model.materials['Tooth'].Elastic(table=((17000.0, 0.3), ))
sim_model.materials['Tooth'].Density(table=((2.02e-09, ), ))
sim_model.Material(name='PDL')
sim_model.materials['PDL'].Hyperelastic(
        materialType=ISOTROPIC, testData=OFF, type=OGDEN,
        volumetricResponse=VOLUMETRIC_DATA, table=((0.07277, 16.95703, 3e-07), ))
sim_model.materials['PDL'].Density(table=((1.0e-09, ), ))

# Define sections
sim_model.HomogeneousSolidSection(name='Max_section',
    material='Bone', thickness=None)
sim_model.HomogeneousSolidSection(name='Tooth_section',
    material='Tooth', thickness=None)
sim_model.HomogeneousSolidSection(name='PDL_section',
    material='PDL', thickness=None)

# Assign sections
region = max_part.sets['MAXILLA_elem_set']
max_part.SectionAssignment(region=region, sectionName='Max_section', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField=",
    thicknessAssignment=FROM_SECTION)

for i in range(1,8):
    p = sim_model.parts['UL{}'.format(i)]
    region = p.sets['UL{}_elem_set'.format(i)]
    p.SectionAssignment(region=region, sectionName='Tooth_section'.format(i), offset=0.0,
    offsetField=",)
    p = sim_model.parts['UL{}_PDL'.format(i)]
    region = p.sets['UL{}_PDL_elem_set'.format(i)]
    p.SectionAssignment(region=region, sectionName='PDL_section'.format(i), offset=0.0,
    offsetField=",)
    p = sim_model.parts['UR{}'.format(i)]
    region = p.sets['UR{}_elem_set'.format(i)]
    p.SectionAssignment(region=region, sectionName='Tooth_section'.format(i), offset=0.0,
    offsetField=",)
    p = sim_model.parts['UR{}_PDL'.format(i)]
    region = p.sets['UR{}_PDL_elem_set'.format(i)]
    p.SectionAssignment(region=region, sectionName='PDL_section'.format(i), offset=0.0,
    offsetField=",)

# Assign element types
import mesh
```

```
hybrid_tet = mesh.ElemType(elemCode=C3D4H, elemLibrary=STANDARD,
  secondOrderAccuracy=OFF, distortionControl=DEFAULT)

for i in range(1,8):
  p = sim_model.parts['UL{}_PDL'.format(i)]
  region = p.sets['UL{}_PDL_elem_set'.format(i)]
  p.setElementType(regions=region, elemTypes=(hybrid_tet, ))
  p = sim_model.parts['UR{}_PDL'.format(i)]
  region = p.sets['UR{}_PDL_elem_set'.format(i)]
  p.setElementType(regions=region, elemTypes=(hybrid_tet, ))

# Set root assemblies variables
sim_root = sim_model.rootAssembly

# Create instances
sim_root.Instance(name='MAXILLA', part=max_part, dependent=ON)

for i in range(1,8):
  p = sim_model.parts['UL{}'.format(i)]
  sim_root.Instance(name='UL{}'.format(i), part=p, dependent=ON)
  p = sim_model.parts['UL{}_PDL'.format(i)]
  sim_root.Instance(name='UL{}_PDL'.format(i), part=p, dependent=ON)
  p = sim_model.parts['UR{}'.format(i)]
  sim_root.Instance(name='UR{}'.format(i), part=p, dependent=ON)
  p = sim_model.parts['UR{}_PDL'.format(i)]
  sim_root.Instance(name='UR{}_PDL'.format(i), part=p, dependent=ON)

# Create merged instances
instance_list = []

for i in range(1, 8):
  instance_list.append(sim_root.instances['UL{}'.format(i)])
  instance_list.append(sim_root.instances['UR{}'.format(i)])
  if i > 1:
    # Merge instances
    sim_root.InstanceFromBooleanMerge(name='U{}_{}'.format(i, i), instances=instance_list,
       originalInstances=SUPPRESS, mergeNodes=BOUNDARY_ONLY,
       nodeMergingTolerance=1e-06, domain=BOTH)
    # Rename merged instance
    sim_root.features.changeKey(fromName='U{}_{}-1'.format(i, i),
      toName='U{}_{}'.format(i, i))
    # Resume suppressed teeth
    for n in range(1, i + 1):
      sim_root.features['UL{}'.format(n)].resume()
      sim_root.features['UR{}'.format(n)].resume()

# Define constraints
for i in range(1,8):
  region1 = sim_root.instances['MAXILLA'].surfaces['UL{}_socket'.format(i)]
  region2 = sim_root.instances['UL{}_PDL'.format(i)].surfaces['UL{}_PDL_outer'.format(i)]
  sim_model.Tie(name='UL{}_socket_PDL'.format(i), master=region1,
    slave=region2, positionToleranceMethod=COMPUTED, adjust=OFF,
    tieRotations=ON, thickness=ON)
  region1 = sim_root.instances['MAXILLA'].surfaces['UR{}_socket'.format(i)]
  region2 = sim_root.instances['UR{}_PDL'.format(i)].surfaces['UR{}_PDL_outer'.format(i)]
  sim_model.Tie(name='UR{}_socket_PDL'.format(i), master=region1,
    slave=region2, positionToleranceMethod=COMPUTED, adjust=OFF,
    tieRotations=ON, thickness=ON)
  # Create single tooth constraints
  if i == 2 or 4 or 5 or 7:
    # Randomly pick between right/left
    side = random.choice(random_list)
    if side == 'left':
      region2 = sim_root.instances['UL{}_PDL'.format(i)].surfaces['UL{}_PDL_inner'.format(i)]
      region1 = sim_root.instances['UL{}'.format(i)].surfaces['UL{}'.format(i)]
      sim_model.Tie(name='UL{}_PDL'.format(i), master=region1,
          slave=region2, positionToleranceMethod=COMPUTED, adjust=OFF,
          tieRotations=ON, thickness=ON)
      if i == 2:
          U2 = 'left'
      elif i == 4:
          U4 = 'left'
      elif i == 5:
          U5 = 'left'
      else:
          U7 = 'left'
    else:
      region2 = sim_root.instances['UR{}_PDL'.format(i)].surfaces['UR{}_PDL_inner'.format(i)]
      region1 = sim_root.instances['UR{}'.format(i)].surfaces['UR{}'.format(i)]
      sim_model.Tie(name='UR{}_PDL'.format(i), master=region1,
          slave=region2, positionToleranceMethod=COMPUTED, adjust=OFF,
          tieRotations=ON, thickness=ON)
      if i == 2:
          U2 = 'right'
      elif i == 4:
          U4 = 'right'
      elif i == 5:
          U5 = 'right'
      else:
          U7 = 'right'
  # Create multi-tooth constraints
  if i > 1:
    for n in range(1, i + 1):
      region2 = sim_root.instances['UL{}_PDL'.format(n)].surfaces['UL{}_PDL_inner'.format(n)]
      region1 = sim_root.instances['U{}_{}'.format(i, i)].surfaces['UL{}'.format(n)]
      sim_model.Tie(name='UL{}_PDL_{}'.format(n, i), master=region1,
          slave=region2, positionToleranceMethod=COMPUTED, adjust=OFF,
          tieRotations=ON, thickness=ON)
      region2 = sim_root.instances['UR{}_PDL'.format(n)].surfaces['UR{}_PDL_inner'.format(n)]
      region1 = sim_root.instances['U{}_{}'.format(i, i)].surfaces['UR{}'.format(n)]
      sim_model.Tie(name='UR{}_PDL_{}'.format(n, i), master=region1,
          slave=region2, positionToleranceMethod=COMPUTED, adjust=OFF,
          tieRotations=ON, thickness=ON)

# Create posterior instances and constraints
# Randomly pick between right/left
# U4_7
instance_list = []
side = random.choice(random_list)
if side == 'left':
  # Create merged part
  # Generate instance list
  for i in range(4, 8):
    sim_root.features['UL{}'.format(i)].resume()
    instance_list.append(sim_root.instances['UL{}'.format(i)])
  # Merge instances
  sim_root.InstanceFromBooleanMerge(name='U4_7', instances=instance_list,
    originalInstances=SUPPRESS, mergeNodes=BOUNDARY_ONLY,
    nodeMergingTolerance=1e-06, domain=BOTH)
  # Rename merged instance
  sim_root.features.changeKey(fromName='U4_7-1',
        toName='U4_7')
    # Resume suppressed teeth
    for i in range(4, 8):
        sim_root.features['UL{}'.format(i)].resume()
    # Create constraints
    for i in range(4, 8):
        region2 = sim_root.instances['UL{}_PDL'.format(i)].surfaces['UL{}_PDL_inner'.format(i)]
        region1 = sim_root.instances['U4_7'].surfaces['UL{}'.format(i)]
        sim_model.Tie(name='UL{}_PDL_4_7'.format(i),    master=region1,
            slave=region2, positionToleranceMethod=COMPUTED, adjust=OFF,
            tieRotations=ON, thickness=ON)
    U4_7 = 'left'
else:
    # Create merged part
    # Generate instance list
    for i in range(4, 8):
        sim_root.features['UR{}'.format(i)].resume()
        instance_list.append(sim_root.instances['UR{}'.format(i)])
    #       Merge       instances
    sim_root.InstanceFromBooleanMerge(name='U4_7',   instances=instance_list,
      originalInstances=SUPPRESS, mergeNodes=BOUNDARY_ONLY,
      nodeMergingTolerance=1e-06,   domain=BOTH)
    # Rename merged instance
    sim_root.features.changeKey(fromName='U4_7-1',
        toName='U4_7')
    # Resume suppressed teeth
    for i in range(4, 8):
        sim_root.features['UR{}'.format(i)].resume()
    # Create constraints
    for i in range(4, 8):
        region2 = sim_root.instances['UR{}_PDL'.format(i)].surfaces['UR{}_PDL_inner'.format(i)]
        region1 = sim_root.instances['U4_7'].surfaces['UR{}'.format(i)]
        sim_model.Tie(name='UR{}_PDL_4_7'.format(i),    master=region1,
            slave=region2, positionToleranceMethod=COMPUTED, adjust=OFF,
            tieRotations=ON, thickness=ON)
    U4_7 = 'right'

#U5_7
instance_list            =            []
side = random.choice(random_list)
if side == 'left':
    # Create merged part
    # Generate instance list
    for i in range(5, 8):
        sim_root.features['UL{}'.format(i)].resume()
        instance_list.append(sim_root.instances['UL{}'.format(i)])
    #       Merge       instances
    sim_root.InstanceFromBooleanMerge(name='U5_7',   instances=instance_list,
      originalInstances=SUPPRESS, mergeNodes=BOUNDARY_ONLY,
      nodeMergingTolerance=1e-06,   domain=BOTH)
    # Rename merged instance
    sim_root.features.changeKey(fromName='U5_7-1',
        toName='U5_7')
    # Resume suppressed teeth
    for i in range(5, 8):
        sim_root.features['UL{}'.format(i)].resume()
    # Create constraints
    for i in range(5, 8):
        region2 = sim_root.instances['UL{}_PDL'.format(i)].surfaces['UL{}_PDL_inner'.format(i)]
        region1 = sim_root.instances['U5_7'].surfaces['UL{}'.format(i)]
        sim_model.Tie(name='UL{}_PDL_5_7'.format(i),    master=region1,
            slave=region2, positionToleranceMethod=COMPUTED, adjust=OFF,
            tieRotations=ON, thickness=ON)
    U5_7 = 'left'
else:
    # Create merged part
    # Generate instance list
    for i in range(5, 8):
        sim_root.features['UR{}'.format(i)].resume()
        instance_list.append(sim_root.instances['UR{}'.format(i)])
    # Merge instances
    sim_root.InstanceFromBooleanMerge(name='U5_7', instances=instance_list,
          originalInstances=SUPPRESS, mergeNodes=BOUNDARY_ONLY,
          nodeMergingTolerance=1e-06, domain=BOTH)
    # Rename merged instance
    sim_root.features.changeKey(fromName='U5_7-1',
        toName='U5_7')
    # Resume suppressed teeth
    for i in range(5, 8):
        sim_root.features['UR{}'.format(i)].resume()
    # Create constraints
    for i in range(5, 8):
        region2 = sim_root.instances['UR{}_PDL'.format(i)].surfaces['UR{}_PDL_inner'.format(i)]
        region1 = sim_root.instances['U5_7'].surfaces['UR{}'.format(i)]
        sim_model.Tie(name='UR{}_PDL_5_7'.format(i), master=region1,
            slave=region2, positionToleranceMethod=COMPUTED, adjust=OFF,
            tieRotations=ON, thickness=ON)
    U5_7 = 'right'

# User manually sets BC, creates sets for force points
MessageBox(None, 'Please set BC, create sets for force points, and orient model.\nRun Model_setup_Part3 when finished.',
'Model_setup_Part2 Completed', 0)
```

3. Model_setup_Part3.py

```
#
#
# Part 3 creates steps and jobs for each simulation

#       U1_y
if U1 == 'left':
   tooth = 'UL1'
else:
   tooth = 'UR1'

name = 'U1_y_force'
direction = 'y'

resume_tooth(tooth)
first_step(name, tooth, direction)
new_job(name)

#          UL1_z
name = 'U1_z_force'
direction = 'z'

next_step(name, tooth, direction)
new_job(name)

#       U3_y
if U3 == 'left':
   tooth = 'UL3'
else:
   tooth = 'UR3'

name = 'U3_y_force'
direction = 'y'

resume_tooth(tooth)
next_step(name, tooth, direction)
new_job(name)

#           U3_z
name = 'U3_z_force'
direction = 'z'

next_step(name, tooth, direction)
new_job(name)

#       U6_y
if U6 == 'left':
   tooth = 'UL6'
else:
   tooth = 'UR6'

name    =    'U6_y_force'
direction = 'y'

next_step(name, tooth, direction)
new_job(name)

#           U6_z
name = 'U6_z_force'
direction = 'z'
```

```
next_step(name, tooth, direction)
new_job(name)

# Multi tooth groups
for i in range(2,8):
    teeth = 'U{}_{}'.format(i, i)
    name = 'U{}_{}_y_force'.format(i, i)
    direction = 'y'
    next_step(name, teeth, direction)
    new_job(name)
    name = 'U{}_{}_z_force'.format(i, i)
    direction = 'z'
    next_step(name, teeth, direction)
    new_job(name)

# Posterior tooth groups
# U4_7
teeth = 'U4_7'
name = 'U4_7_y_force'
direction = 'y'

next_step(name, teeth, direction)
new_job(name)

name = 'U4_7_z_force'
direction = 'z'

next_step(name, teeth, direction)
new_job(name)

# U5_7
teeth = 'U5_7'
name = 'U5_7_y_force'
direction = 'y'

next_step(name, teeth, direction)
new_job(name)

name = 'U5_7_z_force'
direction = 'z'

next_step(name, teeth, direction)
new_job(name)

# User selects which jobs to run
MessageBox(None, 'Please proceed to Job_submission to select jobs to run.', 'Model setup completed', 0)
```

4. Functions.py

```
#

#

# List of all functions used in setting up and running analyses


from part import *

from material import *

from section import *

from assembly import *

from step import *

from interaction import *

from load import *

from mesh import *

from optimization import * from

job import * from sketch import

* from visualization import *

from connectorBehavior import *


from abaqus import getInput

import numpy as np

import math


# Import message box modules

import ctypes

MessageBox = ctypes.windll.user32.MessageBoxA


# Set up random list

import random

random_list = ['right', 'left']


# Define model variables

sim_model = mdb.models['Simulation']

sim_root = sim_model.rootAssembly
```

```
# Define directories
fields=(('Model directory:','C:/Users/gandhi/Desktop/Recent Model/Recent model/Pt2/'),
    ('Plugin directory', 'C:/SIMULIA/CAE/2017/win_b64/code/python2.7/lib/abaqus_plugins/findNearestNode'),
    ('Script directory', 'C:/Users/gandhi/Desktop/Recent Model/Recent Scripts/'))
directory, plugin_dir, script_dir = getInputs(fields=fields, label='Specify file paths', dialogTitle='Define directories', )

# Define functions
def PP3D(name):
    # Define post-processing script directory
    file_name = '3D_processing_fx.py'
    file_path = script_dir + file_name
    execfile(file_path,    main    .__dict   )

def suppress_tooth(tooth_number):
    # Suppress tooth_number materials
    sim_root.features[tooth_number].suppress()
    sim_root.features[tooth_number + '_PDL'].suppress()
    sim_model.constraints[tooth_number + '_socket_PDL'].suppress()
    sim_model.constraints[tooth_number + '_PDL'].suppress()

def  suppress_merge(number):
    # Suppress merged part materials
    sim_root.features['U{}_{}'.format(number, number)].suppress()
    for i in range(1, int(number) + 1):
        sim_root.features['UL{}_PDL'.format(i)].suppress()
        sim_root.features['UR{}_PDL'.format(i)].suppress()
        sim_model.constraints['UL{}_PDL_{}'.format(i, number)].suppress()
        sim_model.constraints['UR{}_PDL_{}'.format(i, number)].suppress()
        sim_model.constraints['UL{}_socket_PDL'.format(i)].suppress()
        sim_model.constraints['UR{}_socket_PDL'.format(i)].suppress()

def resume_tooth(tooth_number):
    # Resume tooth_number materials
    sim_root.features[tooth_number].resume()
    sim_root.features[tooth_number  + '_PDL'].resume()
    sim_model.constraints[tooth_number + '_socket_PDL'].resume()
    sim_model.constraints[tooth_number + '_PDL'].resume()

def  resume_merge(number):
    # Suppress merged part materials
    sim_root.features['U{}_{}'.format(number, number)].resume()
    for i in range(1, int(number) + 1):
        sim_root.features['UL{}_PDL'.format(i)].resume()
        sim_root.features['UR{}_PDL'.format(i)].resume()
        sim_model.constraints['UL{}_PDL_{}'.format(i, number)].resume()
        sim_model.constraints['UR{}_PDL_{}'.format(i, number)].resume()
        sim_model.constraints['UL{}_socket_PDL'.format(i)].resume()
        sim_model.constraints['UR{}_socket_PDL'.format(i)].resume()

def resume_step(name):
    sim_model.steps[name].resume()

def suppress_step(name):
    sim_model.steps[name].suppress()

def first_step(name, tooth, direction):
    #     Create     step
    sim_model.StaticStep(name=name,  previous='Initial',
        timePeriod=0.1, maxNumInc=10000, initialInc=0.001, minInc=1e-06,
        amplitude=RAMP, nlgeom=ON)
    if direction == 'y':
        # Field Output Request
        sim_model.fieldOutputRequests.changeKey(fromName='F-Output-1',
                toName='F-Output-{}_y'.format(tooth))
        sim_model.fieldOutputRequests['F-Output-{}_y'.format(tooth)].setValues(
            variables=('S','LE','U','RF','CF','COORD'))
        # Create load (y direction)
        region_name        =        name
        region =  sim_root.sets[region_name]
        sim_model.ConcentratedForce(name=region_name,
            createStepName=region_name, region=region, cf2=1.0,
            distributionType=UNIFORM, field='', localCsys=None)
    else:
        # Field Output Request
        sim_model.fieldOutputRequests.changeKey(fromName='F-Output-1',
                toName='F-Output-{}_z'.format(tooth))
        sim_model.fieldOutputRequests['F-Output-{}_z'.format(tooth)].setValues(
            variables=('S','LE','U','RF','CF','COORD'))
        # Create load (z direction)
        region_name = name
        region = sim_root.sets[region_name]
        sim_model.ConcentratedForce(name=region_name,
            createStepName=region_name, region=region, cf3=1.0,
            distributionType=UNIFORM, field='', localCsys=None)


def next_step(name, tooth, direction):
    # Create step
    sim_model.StaticStep(name=name, previous='Initial',
        timePeriod=0.1, maxNumInc=10000, initialInc=0.001, minInc=1e-06,
        amplitude=RAMP, nlgeom=ON)
    if direction == 'y':
        # Field Output Request
        sim_model.FieldOutputRequest(name='F-Output-{}_y'.format(tooth),
                createStepName=name, variables=('S','LE','U','RF','CF','COORD'))
        # Create load (y direction)
        region_name = name
        region = sim_root.sets[region_name]
        sim_model.ConcentratedForce(name=region_name,
            createStepName=region_name, region=region, cf2=1.0,
            distributionType=UNIFORM, field='', localCsys=None)
    else:
        # Field Output Request
        sim_model.FieldOutputRequest(name='F-Output-{}_z'.format(tooth),
                createStepName=name, variables=('S','LE','U','RF','CF','COORD'))
        # Create load (z direction)
        region_name = name
        region = sim_root.sets[region_name]
        sim_model.ConcentratedForce(name=region_name,
            createStepName=region_name, region=region, cf3=1.0,
            distributionType=UNIFORM, field='', localCsys=None)

def new_job(name):
    # Create job
    mdb.Job(name=name, model='Simulation', description='', type=ANALYSIS,
        atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90,
        memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
        explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, echoPrint=OFF,
        modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSubroutine='',
        scratch='', resultsFormat=ODB, multiprocessingMode=DEFAULT, numCpus=2,
        numDomains=2, numGPUs=1)

def submit_job(name):
    mdb.jobs[name].submit(consistencyChecking=OFF)

def write_input(name):
    mdb.jobs[name].writeInput(consistencyChecking=OFF)

def run_job(name):
    mdb.jobs[name].submit(consistencyChecking=OFF)
    mdb.jobs[name].waitForCompletion()

def iterate(name, dir):
    # Delete .lck file/.odb file
    import os
    file_name = '{}.lck'.format(name)
    file_path = dir + file_name
    os.remove(file_path)
    file_name = '{}.odb'.format(name)
    file_path = dir + file_name
    os.remove(file_path)
```

```
    # Re-run job
    run_job(name)
    # Re-run post-processing
    if direction == 'y':
        PPy(name)
    else:
        PPz(name)

defnearest_node(dir, instance, xcoord, ycoord, zcoord, set_name):
    import sys
    sys.path.insert(0, dir)
    import  nearestNodeModule
    session.viewports['Viewport: 1'].assemblyDisplay.setValues(mesh=ON)
    session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.setValues(
    meshTechnique=ON)
    nearestNodeModule.hideTextAndArrow()
    n1 = sim_root.instances[instance].nodes
    pickedSelectedNodes = n1[:]
    n = nearestNodeModule.findNearestNode(xcoord = xcoord, ycoord = ycoord, zcoord = zcoord, name='',
    selectedNodes=pickedSelectedNodes,   instanceName=instance)
    label    =    n[0]
    coordinates    =    n[3]
    nodes1 = n1[label - 1:label]
    force_location =  sim_root.sets[name].nodes[0].coordinates
    if coordinates != force_location:
        sim_root.Set(nodes=nodes1, name=set_name)
    else:
        iterate_check = False

def create_set_from_node(node_number, instance, set_name):
    n1          =          sim_root.instances[instance].nodes
    CR_node = n1[node_number - 1:node_number]
    sim_root.Set(nodes=CR_node,  name=set_name)

def bool_set(set1_name, set2_name, set_name):
    set1 = sim_root.sets[set1_name]
    set2 = sim_root.sets[set2_name]
    sim_root.SetByBoolean(set_name, [set1, set2], DIFFERENCE)

def bool_add_set(set1_name, set2_name, set_name):
    set1 = sim_root.sets[set1_name]
    set2 = sim_root.sets[set2_name]
    sim_root.SetByBoolean(set_name, [set1, set2], UNION)

def hide_instances(number):
    instance_list = ['MAXILLA']
    for i in range(1, int(number) + 1):
        instance_list.append('UL{}_PDL'.format(i))
        instance_list.append('UR{}_PDL'.format(i))
    session.viewports['Viewport: 1'].assemblyDisplay.hideInstances(instances=instance_list)

def suppress_all():
    file_name = 'Suppress_all.py'
    file_path = script_dir + file_name
    execfile(file_path,   main   .__dict   )
```

5. Suppress_all.py

```
#
#
# This script suppresses all instances, constraints, and steps

# Get user input
fields=(('U1:','left'),('U2:','left'),('U3:','left'),('U4:','left'),('U5:','left'),('U6:','left'),('U7:','left'),('U4_7:','left'),('U5_7:','left'))
U1, U2, U3, U4, U5, U6, U7, U4_7, U5_7 = getInputs(fields=fields, label='Specify sides for unilateral groups', dialogTitle='Suppress all', )

#  Suppress  all
for i in range(1,8):
    sim_root.features['UL{}'.format(i)].suppress()
    sim_root.features['UL{}_PDL'.format(i)].suppress()
    sim_root.features['UR{}'.format(i)].suppress()
    sim_root.features['UR{}_PDL'.format(i)].suppress()
    sim_model.constraints['UL{}_socket_PDL'.format(i)].suppress()
    sim_model.constraints['UR{}_socket_PDL'.format(i)].suppress()

# Suppress merged parts
for i in range(2, 8):
    sim_root.features['U{}_{}'.format(i, i)].suppress()
    for n in range(1, i + 1):
        sim_model.constraints['UL{}_PDL_{}'.format(n, i)].suppress()
        sim_model.constraints['UR{}_PDL_{}'.format(n,   i)].suppress()

# Suppress unilateral parts
if U1 == 'left':
    sim_model.constraints['UL1_PDL'.format(i)].suppress()
else:
    sim_model.constraints['UR1_PDL'.format(i)].suppress()
if U2 == 'left':
    sim_model.constraints['UL2_PDL'.format(i)].suppress()
else:
    sim_model.constraints['UR2_PDL'.format(i)].suppress()
if U3 == 'left':
    sim_model.constraints['UL3_PDL'.format(i)].suppress()
else:
    sim_model.constraints['UR3_PDL'.format(i)].suppress()
if U4 == 'left':
    sim_model.constraints['UL4_PDL'.format(i)].suppress()
else:
    sim_model.constraints['UR4_PDL'.format(i)].suppress()
if U5 == 'left':
    sim_model.constraints['UL5_PDL'.format(i)].suppress()
else:
    sim_model.constraints['UR5_PDL'.format(i)].suppress()
if U6 == 'left':
    sim_model.constraints['UL6_PDL'.format(i)].suppress()
else:
    sim_model.constraints['UR6_PDL'.format(i)].suppress()
if U7 == 'left':
    sim_model.constraints['UL7_PDL'.format(i)].suppress()
else:
    sim_model.constraints['UR7_PDL'.format(i)].suppress()

# Posterior groups
sim_root.features['U4_7'].suppress()
for i in range(4, 8):
    if U4_7 == 'left':
        sim_model.constraints['UL{}_PDL_4_7'.format(i)].suppress()
    else:
        sim_model.constraints['UR{}_PDL_4_7'.format(i)].suppress()

sim_root.features['U5_7'].suppress()
for i in range(5, 8):
    if U5_7 == 'left':
        sim_model.constraints['UL{}_PDL_5_7'.format(i)].suppress()
    else:
        sim_model.constraints['UR{}_PDL_5_7'.format(i)].suppress()
```

```
# Suppress steps

name = 'U1_y_force'

suppress_step(name)

name = 'U1_z_force'

suppress_step(name)

name = 'U2_y_force'

suppress_step(name)

name = 'U2_z_force'

suppress_step(name)

name = 'U3_y_force'

suppress_step(name)

name = 'U3_z_force'

suppress_step(name)

name = 'U4_y_force'

suppress_step(name)

name = 'U4_z_force'

suppress_step(name)

name = 'U5_y_force'

suppress_step(name)

name = 'U5_z_force'

suppress_step(name)

name = 'U6_y_force'

suppress_step(name)

name = 'U6_z_force'

suppress_step(name)

name = 'U7_y_force'

suppress_step(name)

name = 'U7_z_force'

suppress_step(name)


# Suppress multi tooth groups

for i in range(2, 8):

    name = 'U{}_{}_y_force'.format(i, i)

    suppress_step(name)

    name = 'U{}_{}_z_force'.format(i, i)
    suppress_step(name)

# Suppress posterior tooth groups
name = 'U4_7_y_force'
suppress_step(name)
name = 'U4_7_z_force'
suppress_step(name)
name = 'U5_7_y_force'
suppress_step(name)
name = 'U5_7_z_force'
suppress_step(name)
```

6. Job_submission.py

```
#
#
# Submits multiple jobs to run in parallel

# Suppress all to start
suppress_all()

# User selects which jobs to run
fields=(('U1:','Y'),('U2:','Y'),('U3:','Y'),('U4:','Y'),('U5:','Y'),('U6:','Y'),('U7:','Y'),('U2_2:','Y'),('U3_3:','Y'),('U4_4:','Y'),
    ('U5_5:', 'Y'), ('U6_6:', 'Y'), ('U7_7:', 'Y'), ('U4_7:', 'Y'), ('U5_7:', 'Y'))
U1_job, U2_job, U3_job, U4_job, U5_job, U6_job, U7_job, U2_2_job, U3_3_job, U4_4_job, U5_5_job, U6_6_job, U7_7_job, U4_7_job, U5_7_job=getInputs(fields=fields,
    label='Specify jobs to submit', dialogTitle='Job submission', )

job_list=[U1_job, U2_job, U3_job, U4_job, U5_job, U6_job, U7_job, U2_2_job, U3_3_job, U4_4_job, U5_5_job, U6_6_job, U7_7_job, U4_7_job,  U5_7_job]
step_list = ['U1', 'U2', 'U3', 'U4', 'U5', 'U6', 'U7', 'U2_2', 'U3_3', 'U4_4', 'U5_5', 'U6_6', 'U7_7', 'U4_7', 'U5_7']

# U1
if U1_job == 'Y':
    if U1 == 'left':
        resume_tooth('UL1')
    else:
        resume_tooth('UR1')
    fields=(('Y force:','Y'),('Z force:','Y'))
    y_force, z_force = getInputs(fields=fields, label='Specify desired directions for U1', dialogTitle='Directions for analysis', )
    if y_force=='Y':
        resume_step('U1_y_force')
        submit_job('U1_y_force')
        suppress_step('U1_y_force')
    if z_force == 'Y':
        resume_step('U1_z_force')
        submit_job('U1_z_force')
        suppress_step('U1_z_force')
    if U1 == 'left':
        suppress_tooth('UL1')
    else:
        suppress_tooth('UR1')
# U2
if U2_job == 'Y':
    if U2 == 'left':
        resume_tooth('UL2')
    else:
        resume_tooth('UR2')
    fields=(('Y force:','Y'),('Z force:','Y'))
    y_force, z_force = getInputs(fields=fields, label='Specify desired directions for U2', dialogTitle='Directions for analysis', )
    if y_force=='Y':
        resume_step('U2_y_force')
        submit_job('U2_y_force')
        suppress_step('U2_y_force')
    if z_force == 'Y':
        resume_step('U2_z_force')
        submit_job('U2_z_force')
        suppress_step('U2_z_force')
    if U2 == 'left':
        suppress_tooth('UL2')
    else:
        suppress_tooth('UR2')

# U3
if U3_job == 'Y':
    if U3 == 'left':
        resume_tooth('UL3')
    else:
        resume_tooth('UR3')
    fields=(('Y force:','Y'),('Z force:','Y'))
    y_force, z_force = getInputs(fields=fields, label='Specify desired directions for U3', dialogTitle='Directions for analysis', )
    if y_force=='Y':
        resume_step('U3_y_force')
        submit_job('U3_y_force')
        suppress_step('U3_y_force')
    if z_force == 'Y':
        resume_step('U3_z_force')
        submit_job('U3_z_force')
        suppress_step('U3_z_force')
    if U3 == 'left':
        suppress_tooth('UL3')
    else:
        suppress_tooth('UR3')

# U4
if U4_job == 'Y':
    if U4 == 'left':
        resume_tooth('UL4')
    else:
        resume_tooth('UR4')
    fields=(('Y force:', 'Y'), ('Z force:', 'Y'))
    y_force, z_force = getInputs(fields=fields, label='Specify desired directions for U4', dialogTitle='Directions for analysis', )
    ify_force=='Y':
        resume_step('U4_y_force')
        submit_job('U4_y_force')
        suppress_step('U4_y_force')
    if z_force == 'Y':
        resume_step('U4_z_force')
        submit_job('U4_z_force')
        suppress_step('U4_z_force')
    if U4 == 'left':
        suppress_tooth('UL4')
    else:
        suppress_tooth('UR4')

# U5
if U5_job == 'Y':
    if U5 == 'left':
        resume_tooth('UL5')
    else:
        resume_tooth('UR5')
    fields=(('Y force:', 'Y'), ('Z force:', 'Y'))
    y_force, z_force = getInputs(fields=fields, label='Specify desired directions for U5', dialogTitle='Directions for analysis', )
    ify_force=='Y':
        resume_step('U5_y_force')
        submit_job('U5_y_force')
        suppress_step('U5_y_force')
    if z_force == 'Y':
        resume_step('U5_z_force')
        submit_job('U5_z_force')
        suppress_step('U5_z_force')
    if U5 == 'left':
        suppress_tooth('UL5')
    else:
        suppress_tooth('UR5')

# U6
if U6_job == 'Y':
    if U6 == 'left':
        resume_tooth('UL6')
    else:
        resume_tooth('UR6')
    fields=(('Y force:', 'Y'), ('Z force:', 'Y'))
    y_force, z_force = getInputs(fields=fields, label='Specify desired directions for U6', dialogTitle='Directions for analysis', )
    ify_force=='Y':
        resume_step('U6_y_force')
        submit_job('U6_y_force')
        suppress_step('U6_y_force')
    if z_force == 'Y':
        resume_step('U6_z_force')
        submit_job('U6_z_force')
        suppress_step('U6_z_force')
    if U6 == 'left':
        suppress_tooth('UL6')
    else:
        suppress_tooth('UR6')
# U7
if U7_job == 'Y':
    if U7 == 'left':
        resume_tooth('UL7')
    else:
        resume_tooth('UR7')
    fields=(('Y force:', 'Y'), ('Z force:', 'Y'))
    y_force, z_force = getInputs(fields=fields, label='Specify desired directions for U7', dialogTitle='Directions for analysis', )
    ify_force=='Y':
        resume_step('U7_y_force')
        submit_job('U7_y_force')
        suppress_step('U7_y_force')
    if z_force == 'Y':
        resume_step('U7_z_force')
        submit_job('U7_z_force')
        suppress_step('U7_z_force')
    if U7 == 'left':
        suppress_tooth('UL7')
    else:
        suppress_tooth('UR7')

# Multi tooth groups
for i in range(3, 9):
    if job_list[i] == 'Y':
        resume_merge('{}'.format(i - 1))
        fields=(('Y force:', 'Y'), ('Z force:', 'Y'))
        y_force, z_force = getInputs(fields=fields, label='Specify desired directions for {}'.format(step_list[i]), dialogTitle='Directions for analysis', )
        ify_force=='Y':
            resume_step('U{}_{}_y_force'.format(i - 1, i - 1))
            submit_job('U{}_{}_y_force'.format(i - 1, i - 1))
            suppress_step('U{}_{}_y_force'.format(i-1, i-1))
        if z_force == 'Y':
            resume_step('U{}_{}_z_force'.format(i-1, i-1))
            submit_job('U{}_{}_z_force'.format(i - 1, i - 1))
            suppress_step('U{}_{}_z_force'.format(i-1,i-1))
        suppress_merge('{}'.format(i - 1))

# U4_7
if U4_7_job == 'Y':
    sim_root.features['U4_7'].resume()
    ifU4_7=='left':
        for i in range(4, 8):
            sim_root.features['UL{}_PDL'.format(i)].resume()
            sim_model.constraints['UL{}_PDL_4_7'.format(i)].resume()
            sim_model.constraints['UL{}_socket_PDL'.format(i)].resume()
    else:
        for i in range(4, 8):
            sim_root.features['UR{}_PDL'.format(i)].resume()
            sim_model.constraints['UR{}_PDL_4_7'.format(i)].resume()
            sim_model.constraints['UR{}_socket_PDL'.format(i)].resume()
    fields = (('Y force:', 'Y'), ('Z force:', 'Y'))
    y_force, z_force = getInputs(fields=fields, label='Specify desired directions for U4_7', dialogTitle='Directions for analysis', )
    ify_force=='Y':
        resume_step('U4_7_y_force')
        submit_job('U4_7_y_force')
        suppress_step('U4_7_y_force')
    if z_force == 'Y':
        resume_step('U4_7_z_force')
        submit_job('U4_7_z_force')
        suppress_step('U4_7_z_force')
    sim_root.features['U4_7'].suppress()
    if U4_7 == 'left':
        for i in range(4, 8):
            sim_root.features['UL{}_PDL'.format(i)].suppress()
            sim_model.constraints['UL{}_PDL_4_7'.format(i)].suppress()
            sim_model.constraints['UL{}_socket_PDL'.format(i)].suppress()
    else:
        for i in range(4, 8):
            sim_root.features['UR{}_PDL'.format(i)].suppress()
            sim_model.constraints['UR{}_PDL_4_7'.format(i)].suppress()
            sim_model.constraints['UR{}_socket_PDL'.format(i)].suppress()


# U5_7
if U5_7_job == 'Y':
    sim_root.features['U5_7'].resume()
    if U5_7 == 'left':
        for i in range(5, 8):
            sim_root.features['UL{}_PDL'.format(i)].resume()
            sim_model.constraints['UL{}_PDL_5_7'.format(i)].resume()
            sim_model.constraints['UL{}_socket_PDL'.format(i)].resume()
    else:
        for i in range(5, 8):
            sim_root.features['UR{}_PDL'.format(i)].resume()
            sim_model.constraints['UR{}_PDL_5_7'.format(i)].resume()
            sim_model.constraints['UR{}_socket_PDL'.format(i)].resume()
    fields = (('Y force:', 'Y'), ('Z force:', 'Y'))
    y_force, z_force = getInputs(fields=fields, label='Specify desired directions for U5_7', dialogTitle='Directions for analysis', )
    if y_force == 'Y':
        resume_step('U5_7_y_force')
        submit_job('U5_7_y_force')
        suppress_step('U5_7_y_force')
    if z_force == 'Y':
        resume_step('U5_7_z_force')
        submit_job('U5_7_z_force')
        suppress_step('U5_7_z_force')
    sim_root.features['U5_7'].suppress()
    if U5_7 == 'left':
        for i in range(5, 8):
            sim_root.features['UL{}_PDL'.format(i)].suppress()
            sim_model.constraints['UL{}_PDL_5_7'.format(i)].suppress()
            sim_model.constraints['UL{}_socket_PDL'.format(i)].suppress()
    else:
        for i in range(5, 8):
```

```
sim_root.features['UR{}_PDL'.format(i)].suppress()

sim_model.constraints['UR{}_PDL_5_7'.format(i)].suppress()

sim_model.constraints['UR{}_socket_PDL'.format(i)].suppress()
```

7. 3D_processing_fx.py

```
#
#
# After job is run, analyzes rxn forces from .odb and estimates CR in vertical axis
# (force with y-vector)

from  odbAccess  import  *
from abaqusConstants import *
from odbMaterial import *
from odbSection import *

import numpy as np
import math

# Access ODB
# Define job specific variables
job_name = name + '.odb'
step_name  = name
print("""
Job {}
""".format(job_name))
odb = openOdb(job_name)
print("""
Step {}
""".format(step_name))

#          Get          coord          from          set
sim_model          =          mdb.models['Simulation']
sim_root       =       sim_model.rootAssembly
force_location = sim_root.sets[name].nodes[0].coordinates
moment_ref =  np.array(force_location)

# Read field output from last frame (along Y-axis)

last_frame  = odb.steps[step_name].frames[-1]
```

```
rxn_forces = last_frame.fieldOutputs['RF']
orig_coord = last_frame.fieldOutputs['COORD']
displacement = last_frame.fieldOutputs['U']

rxn_field_values = rxn_forces.values
orig_node_coord = orig_coord.values
U = displacement.values

length = len(rxn_field_values)

# Define initial sum of forces and moments

pos_forces = np.zeros(3)
neg_forces = np.zeros(3)
pos_moments = np.zeros(3)
neg_moments = np.zeros(3)

for i in range(length):

    # Find deformed coordinates

    orig_coord_array = np.array(orig_node_coord[i].data)
    node_displacement   =   np.array(U[i].data)
    new_node_coord = orig_coord_array +  node_displacement

    # Create array of ([r_y, r_z], [F_y, F_z]) for moment about x-axis

    current_force_array = np.array(rxn_field_values[i].data)

    r = new_node_coord - moment_ref
    F = current_force_array

    # Determine moment at node

    current_moment = np.cross(r, F)
    # Sort forces/moments according to +/- direction
    # For force vectors in the same direction as original
    if np.dot(orig_force_vector, F) > 0:
        pos_forces += current_force_array
        pos_moments += current_moment
    elif np.dot(orig_force_vector, F) < 0:
        neg_forces += current_force_array
        neg_moments += current_moment

print ("""
The reaction forces are {} and {}, and the two moments created by those
forces about the point {} are {} and {}.
""".format(pos_forces, neg_forces, moment_ref,
pos_moments, neg_moments))

# Test for CR
# Determine magnitude of moment vectors in order to then divide by force and find magnitude of r vectors
a2 = pos_moments[0]**2 + pos_moments[1]**2 + pos_moments[2]**2
pos_moment_mag=math.sqrt(a2)
print('pos_moment_mag = ' + str(pos_moment_mag))
a2 = neg_moments[0]**2 + neg_moments[1]**2 + neg_moments[2]**2
neg_moment_mag = math.sqrt(a2)
print('neg_moment_mag = ' + str(neg_moment_mag))

if abs(pos_moment_mag) <= 0.01 and abs(neg_moment_mag) <= 0.01:
    print("The approximate location of CR is {}.".format(moment_ref))
    iterate_check = False
else:
    iterate_check = True
    # Determine magnitude of force vectors
    a2 = pos_forces[0]**2 + pos_forces[1]**2 + pos_forces[2]**2
    pos_force_mag = math.sqrt(a2)
    print('pos_force_mag = ' + str(pos_force_mag))
    a2 = neg_forces[0]**2 + neg_forces[1]**2 + neg_forces[2]**2
```

```
neg_force_mag = math.sqrt(a2)
print('neg_force_mag = ' + str(neg_force_mag))

# Determine magnitude of r vectors
r_mag_pos = pos_moment_mag / pos_force_mag
print('r_mag_pos = ' + str(r_mag_pos))
r_mag_neg = neg_moment_mag / neg_force_mag
print('r_mag_neg = ' + str(r_mag_neg))

# Determine unit vector for r (F X moment per right hand rule)
r_pos = np.cross((pos_forces + orig_force_vector), pos_moments)
r_neg = np.cross(neg_forces, neg_moments)
a2 = r_pos[0]**2 + r_pos[1]**2 + r_pos[2]**2
r_cross_pos_mag = math.sqrt(a2)
a2 = r_neg[0]**2 + r_neg[1]**2 + r_neg[2]**2
r_cross_neg_mag = math.sqrt(a2)

r_pos_unit = r_pos / r_cross_pos_mag
print('r_pos_unit = ' + str(r_pos_unit))
r_neg_unit= r_neg / r_cross_neg_mag
print('r_neg_unit='+str(r_neg_unit))

# Determine resultant r vectors
r_pos_forces = r_pos_unit * r_mag_pos
print('r_pos_forces='+str(r_pos_forces))
r_neg_forces = r_neg_unit * r_mag_neg
print('r_neg_forces = ' + str(r_neg_forces))

# Estimate new locations
pos_est = moment_ref + r_pos_forces
neg_est = moment_ref + r_neg_forces
print ("The estimated locations of the balancing forces from {} are {} and {}.".format(moment_ref, pos_est, neg_est))
avg = (pos_est + neg_est) / 2
new_force_location = avg
a2 = pos_est[0]**2 + pos_est[1]**2 + pos_est[2]**2
```

```
    pos_est_mag = math.sqrt(a2)
    a2 = neg_est[0]**2 + neg_est[1]**2 + neg_est[2]**2
    neg_est_mag =  math.sqrt(a2)
    print("The estimated location is {}. Please verify force system at this location.".format(avg))
    xcoord = new_force_location[0]
    ycoord = new_force_location[1]
    zcoord = new_force_location[2]

closeOdb(odb)

file_name  =  '{}.lck'.format(name)
file_path='{}{}'.format(directory,file_name)
os.remove(file_path)
```

8. Bulk_process.py

```
#

#

# Run post-process function for several jobs at a time


# Get user input regarding which jobs to process

fields=(('U1:','Y'),('U2:','Y'),('U3:','Y'),('U4:','Y'),('U5:','Y'),('U6:','Y'),('U7:','Y'),('U2_2:','Y'),('U3_3:','Y'),('U4_4:','Y'),
  ('U5_5:', 'Y'), ('U6_6:', 'Y'), ('U7_7:', 'Y'), ('U4_7:', 'Y'), ('U5_7:', 'Y'))

U1_job, U2_job, U3_job, U4_job, U5_job, U6_job, U7_job, U2_2_job, U3_3_job, U4_4_job, U5_5_job, U6_6_job, U7_7_job, U4_7_job, U5_7_job=getInputs(fields=fields,
  label='Specify jobs to analyze', dialogTitle='Analyze multiple jobs', )


job_list = [U1_job, U2_job, U3_job, U4_job, U5_job, U6_job, U7_job, U2_2_job, U3_3_job, U4_4_job, U5_5_job, U6_6_job, U7_7_job, U4_7_job,  U5_7_job]

step_list = ['U1', 'U2', 'U3', 'U4', 'U5', 'U6', 'U7', 'U2_2', 'U3_3', 'U4_4', 'U5_5', 'U6_6', 'U7_7', 'U4_7', 'U5_7']


for n in range(0, 11):
  if job_list[n] == 'Y':
    fields = (('Y force:', 'Y'), ('Z force:', 'Y'))
    y_force, z_force = getInputs(fields=fields, label='Specify desired directions for {}'.format(step_list[n]), dialogTitle='Directions for analysis', )
    if n < 8:
      instance = getInput('Instance for nearest_node module:')
    else:
      instance = step_list[n]
    # Resume suppressed components for analysis
    sim_root.features[instance].resume()
    if y_force=='Y':
      name = '{}_y_force'.format(step_list[n])
      orig_force_vector = np.array([0, 1, 0])
      PP3D(name)
      if iterate_check == True:
        nearest_node(plugin_dir, instance, xcoord, ycoord, zcoord,  name)
    if z_force == 'Y':
      name = '{}_z_force'.format(step_list[n])
      orig_force_vector = np.array([0, 0, 1])
      PP3D(name)
    if          iterate_check          ==          True:
        nearest_node(plugin_dir, instance, xcoord, ycoord, zcoord,  name)
sim_root.features[instance].suppress()
```