A. Mubarak (2021)_Python Scripts for BNWF models in ABAQUS_REV00

Mubarak, Ahmed

DOI:
10.20933/100001310

Publication date:
2021

Appendix []: Summary of the python scripts used to build Beam-on-Winkler-Foundation (BNWF) models using the Abaqus/Standard software:

1. Python code for Asynchronous Motion Amplitude Vector for straight line tunnels

```
maxamps= int (input(f'Enter max number of Amplitudes'))
maxampssteps= int (input(f'Enter max number of time steps in the lagged Amplitude'))

time = [] #Enter the time vector
H1amp= [] #Enter the amplitude values of the earthquake component in X.
H2amp = [] # Enter the amplitude values of the earthquake component in Y.
Vamp= [] #Enter the amplitude values of the earthquake component in Z.


coordinate_1x = []
coordinate_1y = []
count = 0
timeno = 0 #Variable used in While loop to increase in ascending order at each input value from the time list.
serialno= 0 #Variable used after breaking the While loop and increases monotonically to reset the value of the
count variable.
Amp = 1 #Amplitude sequence is define through this variable, the user is advised to enter the numeric value of
first amplitude in the sequence. Amp =1 is chosen by default, but the user could change as necessary.
ampcount = 1 #This variable counts the number of amplitudes in the whole model, always = 1.
while ampcount < maxamps:
   appendFile = open('H1_COMPO_StraightTunnelAmps.txt', 'a')
   appendFile.write ('\n')
   appendFile.write (f"mdb.models['Model-1'].TabularAmplitude(name='Amp-{Amp}', timeSpan=TOTAL,
smooth=SOLVER_DEFAULT, data=(")

   while count < maxampssteps:
    coordinate_1x = time[timeno]
    coordinate_1y = H1amp[count] #The variable H1amp is subject to change to H2amp or Vamp as necessary
    #coordinate_1y = settlement[count]
    appendFile.write (f"({coordinate_1x}, {coordinate_1y}), ")
    count += 1
    timeno += 1
    if count > maxampssteps:
      break
   Amp += 1
   ampcount +=1
   timeno = 0
   serialno += 1
   count = serialno
   maxampssteps+= 1
   if ampcount > maxamps:
      break
   appendFile.close()
```

2. Python code for 3D Synchronous/Asynchronous Motion Amplitude Vector for the circular LHC tunnel model

```
maxamps= int (input(f'Enter max number of Amplitudes'))

time = [] #Enter the time vector
H1= [] # Enter the amplitude values of the earthquake component in X.
H2 = [] # Enter the amplitude values of the earthquake component in Y.
V= [] # Enter the amplitude values of the earthquake component in Z.

lagtime= [] #Enter the t-lag vector for the lagged earthquake motion.
steps= [] #Enter the number of steps vector at each time step, to keep constant number of amplitude values.
coordinate_1x = []
coordinate_1y = []
cos_H1= [] #Enter the cos vector for the earthquake component in X.
sin_H1= [] #Enter the sin vector for the earthquake component in X.
cos_H2= [] #Enter the cos vector for the earthquake component in Y.
sin_H2= [] #Enter the sin vector for the earthquake component in Y.
count1 = 0
count2 = lagtime[count1]
maxampssteps = steps[count1]
timeno = 0

Amp = 1
while Amp < maxamps:
   appendFile = open('FullCircleAmps.txt', 'a')
   appendFile.write ('\n')
   appendFile.write (f"mdb.models['Model-1'].TabularAmplitude(name='Amp-{Amp}', timeSpan=TOTAL,
smooth=SOLVER_DEFAULT, data=(")
   while count2 < maxampssteps:
    coordinate_1x = time[timeno]
    coordinate_1y = round ((H1[count2]*cos_H1[count1])+(H2[count2]*sin_H2[count1]), 6) #for Inner
springs with H1 and H2 COMPOS
    #OR coordinate_1y = round((H1[count2] * sin_H1[count1]) + (H2[count2] * cos_H2[count1]), 6) #for TZ
springs with H1 and H2 COMPOS
    coordinate_1y = v[count2]   #for V springs , No. of Amps = 16212+1
    appendFile.write (f"({coordinate_1x}, {coordinate_1y}), ")
    count2 += 1
    timeno += 1
    if count2 > maxampssteps:
      break
   appendFile.write(f"(49999, 0000), ")
   Amp += 1
   timeno = 0
   count1 += 1
   #count2 = 0                              #Active this script for synchronous motion only
   serialno = lagtime[count1]              # Active this script for asynchronous motion only
   count2 = serialno                       # Active this script for asynchronous motion only
   maxampssteps =steps[count1]             # Active this script for asynchronous motion only
   if Amp > maxamps:
      break
   appendFile.close()
```

3. Python code to create global/local boundary conditions considering the amplitude sequence

```
maxnode= int (input(f' Enter max number of boundary conditions to be created'))

H1AMPS = []              # Enter the amplitudes sequence vector to be assigned in the X direction
H2AMPS = []              # Enter the amplitudes sequence vector to be assigned in the Z/tz direction
VAMPS = []               # Enter the amplitudes sequence vector to be assigned in the Y direction

count = 0
rvalue = 27022           # Represents the ID number of each reference point
bc = 1                   # Boundary conditions count
datum = 68432            # Local datum reference number
Ampcount = 1             # Amplitude count


while count < maxnode:
    appendFile = open('H1 BCs -Asynchronous REV00.txt', 'a')
    appendFile.write('\n')
    appendFile.write("a = mdb.models['Model-1'].rootAssembly")
    appendFile.write('\n')
    appendFile.write("r1 = a.referencePoints")
    appendFile.write('\n')
    appendFile.write(f"refPoints1=(r1[{rvalue}], )")
    appendFile.write('\n')
    appendFile.write("region = regionToolset.Region(referencePoints=refPoints1)")
    appendFile.write('\n')

    #This command is only used for local datums:
    appendFile.write(f"datum = mdb.models['Model-1'].rootAssembly.datums[{datum}]")
    appendFile.write('\n')

    #Modify this command for either global axes "None" or local datums "datum":
    appendFile.write(f"mdb.models['Model-1'].DisplacementBC(name='BC-{bc}', createStepName='Initial',
region=region, u1=SET, u2=SET, u3=SET, ur1=SET, ur2=SET, ur3=SET, amplitude=UNSET,
distributionType=UNIFORM, fieldName='', localCsys=None/datum)")
    appendFile.write('\n')
    appendFile.write(f"mdb.models['Model-1'].boundaryConditions['BC-
{bc}'].setValuesInStep(stepName='Loading', u1=1.0, ur1=FREED, amplitude='Amp-{H1AMPS[count]}')")
    appendFile.write('\n')

    count+=1
    bc+=1
    Ampcount +=1
    datum+=1
    rvalue-=1
    rvalue+=1
    if count > maxnode:
      break
    appendFile.close();
```

4. Python code to assign reference points at a predefined coordinates

```
maxnode= int (input(f'Enter max number of reference points '))

coordx = []          #Enter the x-coordinate vector
coordy = []          #Enter the y-coordinate vector
coordz = []          #Enter the z-coordinate vector

coordinate_1x = []
coordinate_1y = []
coordinate_1z = []

count = 0
while count < maxnode:
   coordinate_1x = coordx[count]
   coordinate_1y = coordy[count]
   coordinate_1z = coordz[count]
   appendFile = open('New_RPS.txt', 'a')
   appendFile.write('\n')
   appendFile.write("a = mdb.models['Model-1'].rootAssembly")
   appendFile.write('\n')
   appendFile.write(f'a.ReferencePoint(point=({coordinate_1x}, {coordinate_1y}, {coordinate_1z}))')
   count += 1
   if count > maxnode:
      break
```

5. Python code to assign local datums at a predefined reference points

```
maxnode= int (input(f'Enter max number of datums'))

rvalue = []        # Enter the vector of the r-values
count = 1
datum = 5406
origin = 47374
point1 = 41969                  # Enter the value of the first reference point on the x-axes
point2 = 41970                  # Enter the value of the second reference point on the x-y plane

while count < maxnode:

   appendFile = open('LocalDatums_TZCurve.txt', 'a')
   appendFile.write('\n')
   appendFile.write("a = mdb.models['Model-1'].rootAssembly")
   appendFile.write('\n')
   appendFile.write(f"r{rvalue[count]} = a.referencePoints")
   appendFile.write('\n')
   appendFile.write(f"a.DatumCsysByThreePoints(origin=r{rvalue[count]}[{origin}],
point1=r{rvalue[count]}[{point1}], point2=r{rvalue[count]}[{point2}], name='Datum csys-{datum}',
coordSysType=CARTESIAN)")
   count+=1
   datum+=1
   origin+=1
   point1+=1
   point2+=1
   if count > maxnode:
      break
   appendFile.close()
```

6. Python code to assign a large number of connectors between reference points

```
maxnode= int (input(f'Enter max number of connectors '))

count = 1
r_1 = 41969          #Enter the r-value of the first reference point
r_2 = 47374          #Enter the r-value of the corresponding reference point

while count < maxnode:
   appendFile = open('ConnectorSprings.txt', 'a')
   appendFile.write('\n')
   appendFile.write(f'(r11[{r_1}], r11[{r_2}]), ')
   count+=1
   r_1 += 1
   r_2 += 1
   if count > maxnode:
      break
   appendFile.close()
```

7. Python code to create a continuous wire (i.e. the tunnel line)

```
maxnode= int (input(f' Enter max number of points '))

coordx= []     #Enter the x-coordinate vector
coordy= []     #Enter the y-coordinate vector
coordinate_1x = []
coordinate_1y = []

count = 0
while count < maxnode:
   coordinate_1x = coordx[count]
   coordinate_1y = coordy[count]
   appendFile = open('SP-Tunnel_Line.txt', 'a')
   appendFile.write('\n')
   appendFile.write(f'({coordinate_1x}, {coordinate_1y}), ')
   count += 1
   if count > maxnode:
      break
```

8. Python code to delete a large number of amplitudes and boundary conditions

```
maxnode= int (input(f'Enter max # of Amps/BCs to Delete'))

Ampcount = 1
while Ampcount < maxnode:
    appendFile = open('Delete.txt', 'a')
    appendFile.write(f"del mdb.models['Model-1'].amplitudes['Amp-{Ampcount}']")
    appendFile.write(f"del mdb.models['Model-1'].boundaryConditions['BC-{Ampcount}']")
    appendFile.write('\n')
    Ampcount +=1
    if Ampcount > maxnode:
      break
    appendFile.close();
```

9. Python code to convert point vertices from ‘edges’ to ‘user defined coordinates’

Commonly, the set of edge points in the GUI is defined using getSequenceFromMask()-method. This method defines the edges of a particular instance using series of numbers which are not comprehensible for the user. It basically defines the location ‘coordinate’ for the selected edge in the GUI frame. According to Abaqus Scripting Reference Guide 6.14 - 7.2.2 the method getSequenceFromMask() is highly efficient when a large number of objects are involved. However, this is not very helpful if you are trying to customize your project to define a complex geometry element to work with. Hence, it is much easier for the user to insert the edges of an instance using a coordinate system instead of masked instances. To do this, copy and past below script into the command bar at the bottom of Abaqus CAE:

```
session.journalOptions.setValues(replayGeometry=COORDINATE,recoverGeometry=COORDINATE)
```

You can then modify the coordinates in order to customize your model geometry and to select the edge locations you like to use in subsequent modelling steps.