```python
# -*- coding: utf-8 -*-
"""
Created on Thu Sep  5 13:23:30 2019

@author: VACALDER
"""

# PROGRAM TO CODE COLUMN NLTHA RUN IN OPENSEESPY
#   Victor A Calderon
#   PhD Student/ Research Assistant
#   NC STATE UNIVERSITY
#   2021 (c)
#
#
# --------------------------------------------------------------------------
# |                              IMPORTS
# --------------------------------------------------------------------------

import numpy as np
from LibUnitsMUS import *
import ManderCC
import openseespy.opensees as ops


def Build_RC_Column(Diameter, Height_of_Column, fPrimeC, fy, fy_transverse, dbi, dti, CL, dblc, nb,
                    AxialLoad, GM_file, GM_dt, GM_npt, ALR, alpha):
    # --------------------------------------------------------------------------
    #
    #    ^Y
    #    |
    #    3       __ |
    #    |          |
    #    |          |
    #    |          |
    #  (2)       LCol
    #    |          |
    #    |          |
    #    |          |
    #  =2=         _|_  --------->X
    #  =1=      ZeroLength

    # --------------------------------------------------------------------------
    # |                      IMPORTS
    # --------------------------------------------------------------------------
    ops.wipe()

    # --------------------------------------------------------------------------
    #                            GENERATE GEOMETRY
    # --------------------------------------------------------------------------

    ops.model('basic', '-ndm', 2, '-ndf', 3)
    LCol = Height_of_Column * inch  # column length
    Weight = AxialLoad * kip  # superstructure weight

    # define section geometry
    DCol = Diameter * inch  # Column Diameterepth
```

```
# Weight = Weight  # nodal dead-load weight per column
Mass = Weight / g

ACol = 0.25 * np.pi * DCol ** 2  # cross-sectional area, make stiff
IzCol = 0.25 * np.pi * DCol ** 4  # Column moment of inertia

ops.node(1, 0.0, 0.0)
ops.node(2, 0.0, 0.0)
ops.node(3, 0.0, LCol)
#  Node, Dx, Dy, Rz
ops.fix(1, 1, 1, 1)
ops.fix(2, 1, 0, 0)

ops.mass(3, Mass, 1e-9, 0.0)

# MATERIAL parameters
IDconcC = 1  # material ID tag -- confined cover concrete
IDconcU = 2  # material ID tag -- unconfined cover concrete
IDreinf = 3  # material ID tag -- reinforcement
IDSP = 4  # material ID tag -- Strain Penetration
# Define materials for nonlinear columns
# ------------------------------------------
# Longitudinal steel properties
Fy = fy * ksi * (1 - alpha * CL)  # STEEL yield stress
Fu = 1.375 * Fy  # Steel Ultimate Stress
Es = 29000.0 * ksi  # modulus of steel
Bs = 0.012  # strain-hardening ratio
R0 = 20.0  # control the transition from elastic to plastic branches
cR1 = 0.90  # control the transition from elastic to plastic branches
cR2 = 0.08  # control the transition from elastic to plastic branches
a1 = 0.039
a2 = 1
a3 = 0.029
a4 = 1.0
c = 2 * inch  # Column cover to reinforcing steel NA.
numBarsSec = nb  # number of uniformly-distributed longitudinal-reinforcement bars
barAreaSec = 0.25 * np.pi * dblc ** 2  # area of longitudinal-reinforcement bars
dbl = dblc * inch

# Transverse Steel Properties
fyt = fy_transverse * ksi * (1 - alpha * CLt)  # Yield Stress of Transverse Steel
dbt = dti * inch  # Diameter of transverse steel
st = s_tc * inch  # Spacing of spiral
Ast = 0.25 * np.pi * (dbt ** 2)  # Area of transverse steel
Dprime = DCol - 2 * c - dti * 0.5  # Inner core diameter
Rbl = Dprime * 0.5 - dti * 0.5 - dbi * 0.5  # Location of longitudinal bar

# nominal concrete compressive strength
fpc = fPrimeC * ksi  # CONCRETE Compressive Strength, ksi   (+Tension, -Compression)
Ec = 57.0 * ksi * np.sqrt(fpc / psi)  # Concrete Elastic Modulus

# unconfined concrete
fc1U = -fpc  # UNCONFINED concrete stress
eps1U = -0.003  # strain at maximum strength of unconfined concrete
fc2U = 0.2 * fc1U  # ultimate stress
```

```
eps2U = -0.01  # strain at ultimate stress
lambdac = 0.1  # ratio between unloading slope at $eps2 and initial slope $Ec

mand = ManderCC.ManderCC(fpc, Ast, fyt, Dprime, st)

fc = mand[0]
eps1 = mand[1]
fc2 = mand[2]
eps2 = mand[3]

# CONCRETE                  tag   f'c        ec0   f'cu        ecu
# Core concrete (confined)
ops.uniaxialMaterial('Concrete01', IDconcC, fc, eps1, fc2, eps2)

# Cover concrete (unconfined)
ops.uniaxialMaterial('Concrete01', IDconcU, fc1U, eps1U, fc2U, eps2U)

# STEEL
# Reinforcing steel
params = [R0, cR1, cR2]
#                         tag  fy E0    b
ops.uniaxialMaterial('Steel02', IDreinf, Fy, Es, Bs, R0, cR1, cR2)

# STRAIN PENETRATION MATERIAL
SPalpha = 0.4
SPsy = 0.1 * ((dbl * Fy) * (2 * SPalpha + 1) / (4000 * ((-fc) ** 0.5))) ** (1 / SPalpha) + 0.01
SPsu = 35 * SPsy
SPb = 0.45
SPR = 1.01

# uniaxialMaterial StrPen01   Tag  fy  sy fu su b  R
ops.uniaxialMaterial('Bond_SP01', IDSP, Fy, SPsy, Fu, SPsu, SPb, SPR)

# Writing Material data to file
with open(datadir + "/mat.out", 'w') as matfile:
    matfile.write("%s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s\n" % (
        Fy, fyt, Ast, st, Dprime, Weight, DCol, LCol, barAreaSec, fc, SPsy, SPsu, SPb, SPR, ALF
matfile.close

# ------------------------------------------------------------------------
#               DEFINE PLASTICE HIGE PROPERTIES
# ------------------------------------------------------------------------

k = 0.2 * (Fu / Fy - 1)
if k > 0.08:
    k = 0.08
Leff = LCol
Lpc = k * Leff + 0.4 * DCol
gamma = 0.33  # Assuming unidirectional action
Lpt = Lpc + gamma * DCol
# FIBER SECTION properties -------------------------------------------------------------
# Define cross-section for nonlinear columns
# ------------------------------------------

# set some paramaters Section 1
ColSecTag = 1
```

```
ri = 0.0
ro = DCol / 2.0
nfCoreR = 8
nfCoreT = 8
nfCoverR = 2
nfCoverT = 8
rc = ro - c
theta = 360.0 / numBarsSec

ops.section('Fiber', ColSecTag, '-GJ', 1e+10)

# Create the concrete fibers
ops.patch('circ', 1, nfCoreT, nfCoreR, 0.0, 0.0, ri, rc, 0.0, 360.0)  # Define the core patch
ops.patch('circ', 2, nfCoverT, nfCoverR, 0.0, 0.0, rc, ro, 0.0, 360.0)  # Define Cover Patch

# Create the reinforcing fibers
ops.layer('circ', 3, numBarsSec, barAreaSec, 0.0, 0.0, Rbl, theta, 360.0)

# Set parameters for ZeroLength Element

SecTag2 = 2
ops.section('Fiber', SecTag2, '-GJ', 1e+10)

# Create the concrete fibers
ops.patch('circ', 1, nfCoreT, nfCoreR, 0.0, 0.0, ri, rc, 0.0, 360.0)  # Define the core patch
ops.patch('circ', 2, nfCoverT, nfCoverR, 0.0, 0.0, rc, ro, 0.0, 360.0)  # Define Cover Patch

# Create the reinforcing fibers
ops.layer('circ', IDSP, numBarsSec, barAreaSec, 0.0, 0.0, Rbl, theta, 360.0)

# Creating Elements

ColTransfTag = 1
ops.geomTransf('Linear', ColTransfTag)

ZL_eleTag = 1
ops.element('zeroLengthSection', ZL_eleTag, 1, 2, SecTag2, '-orient', 0., 1., 0., 1., 0., 0.)

ColeleTag = 2

# Defining Fiber Elements as ForceBeamColumn
# element('nonlinearBeamColumn', eleTag, 1, 2, numIntgrPts, ColSecTag, ColTransfTag)
ColIntTag = 1
# beamIntegration('Lobatto',ColIntTag,ColSecTag,numIntgrPts)
ops.beamIntegration('HingeRadau', ColIntTag, ColSecTag, Lpt, ColSecTag, 1e-10, ColSecTag)
ops.element('forceBeamColumn', ColeleTag, 2, 3, ColTransfTag, ColIntTag, '-mass', 0.0)

# Setting Recorders

ops.recorder('Node', '-file', datadir + '/DFree.out', '-time', '-node', 3, '-dof', 1, 2, 3, 'di
ops.recorder('Node', '-file', datadir + '/RBase.out', '-time', '-node', 2, '-dof', 1, 2, 3, 're
ops.recorder('Element', '-file', datadir + '/StressStrain.out', '-time', '-ele', 2, 'section',
             str(Rbl), '0', '3', 'stressStrain')  # Rbl,0, IDreinf
ops.recorder('Element', '-file', datadir + '/StressStrain4.out', '-time', '-ele', 2, 'section',
             str(-Rbl), '0', '3', 'stressStrain')  # Rbl,0, IDreinf
ops.recorder('Element', '-file', datadir + '/StressStrain2.out', '-time', '-ele', 2, 'section',
```

```
            str(-Dprime), '0.0', '1', 'stressStrain')  # Rbl,0, IDreinf
ops.recorder('Element', '-file', datadir + '/StressStrain3.out', '-time', '-ele', 2, 'section',
            str(-DCol), '0.0', '2', 'stressStrain')

# ------------------------------------------------------------------------------
# |                        NLTHA RUN
# ------------------------------------------------------------------------------

dt = GM_dt
npt = GM_npt
with open(datadir + "/PGA.out", 'w') as PGAfile:
    accelerations = open(GM_file)
    linesacc = accelerations.readlines()
    acc = [line.split() for line in linesacc]
    flat_list = []
    for sublist in acc:
        for item in sublist:
            flat_list.append(item)

    ACC = [float(i) for i in flat_list]
    PGA = max(abs(max(ACC)), abs(min(ACC)))
    PGAfile.write("%s \n" % (PGA))
    PGAfile.close

# defining gravity loads
ops.timeSeries('Linear', 1)
ops.pattern('Plain', 1, 1)
ops.load(3, 0.0, -Weight, 0.0)

Tol = 1e-3  # convergence tolerance for test
NstepGravity = 10
DGravity = 1 / NstepGravity
ops.integrator('LoadControl', DGravity)  # determine the next time step for an analysis
ops.numberer('Plain')  # renumber dof's to minimize band-width (optimization), if you want to
ops.system('BandGeneral')  # how to store and solve the system of equations in the analysis
ops.constraints('Plain')  # how it handles boundary conditions
ops.test('NormDispIncr', Tol, 6)  # determine if convergence has been achieved at the end of ar
ops.algorithm('Newton')  # use Newton's solution algorithm: updates tangent stiffness at every
ops.analysis('Static')  # define type of analysis static or transient
ops.analyze(NstepGravity)  # apply gravity

ops.loadConst('-time', 0.0)  # maintain constant gravity loads and reset time to zero

# applying Dynamic Ground motion analysis
GMdirection = 1
GMfile = GM_file
GMfact = 1.0

Lambda = ops.eigen('-fullGenLapack', 2)  # eigenvalue mode 1
Omega = math.pow(Lambda[0], 0.5)
T1 = 2 * np.pi / Omega

with open(datadir + "/Period.out", 'w') as Periodfile:
    Periodfile.write("%s\n" % (T1))
Periodfile.close
```

```
xDamp = 0.04  # 4% damping ratio
betaKcomm = 2 * (xDamp / Omega)
alphaM = 0.0  # M-pr damping; D = alphaM*M
betaKcurr = 0.0  # K-proportional damping;      +beatKcurr*KCurrent
betaKinit = 0.0  # initial-stiffness proportional damping      +beatKinit*Kini

ops.rayleigh(alphaM, betaKcurr, betaKinit, betaKcomm)  # RAYLEIGH damping

# Uniform EXCITATION: acceleration input
IDloadTag = 400  # load tag
Dt = dt  # time step for input ground motion
GMfatt = GMfact * g  # data in input file is in g Units -- ACCELERATION TH
maxNumIter = 50
ops.timeSeries('Path', 2, '-dt', Dt, '-filePath', GMfile, '-factor', GMfatt)
ops.pattern('UniformExcitation', IDloadTag, GMdirection, '-accel', 2)

ops.wipeAnalysis()
ops.constraints('Transformation')
ops.numberer('Plain')
ops.system('BandGeneral')
ops.test('NormUnbalance', Tol, maxNumIter)
ops.algorithm('KrylovNewton')

NewmarkGamma = 0.5
NewmarkBeta = 0.25
ops.integrator('Newmark', NewmarkGamma, NewmarkBeta)
ops.analysis('Transient')
analysis_substeps = 100

DtAnalysis = dt / analysis_substeps
TmaxAnalysis = DtAnalysis * analysis_substeps * npt

Nsteps = int(TmaxAnalysis / DtAnalysis)

ok = ops.analyze(Nsteps, DtAnalysis)

tCurrent = ops.getTime()

# for gravity analysis, load control is fine, 0.1 is the load factor increment (http://opensees

Atest = {1: 'NormDispIncr', 2: 'RelativeEnergyIncr', 4: 'RelativeNormUnbalance', 5: 'RelativeNc
         6: 'NormUnbalance'}
Algorithm = {1: 'KrylovNewton', 2: 'SecantNewton', 4: 'RaphsonNewton', 5: 'PeriodicNewton', 6:
             8: 'NewtonLineSearch'}

for i in Atest:

    for j in Algorithm:

        if ok != 0:
            if j < 4:
                ops.algorithm(Algorithm[j], '-initial')

            else:
                ops.algorithm(Algorithm[j])
```

```
            ops.test(Atest[i], Tol, 1000)
            ok = ops.analyze(Nsteps, DtAnalysis)
            ops.algorithm('ModifiedNewton')
            if ok == 0:
                print('Analysis succesful: ', Atest[i], ' ', Algorithm[j], ' OK = ', ok)

                break
        else:
            continue

print("GroundMotion Done ", ops.getTime())

ops.wipe()
```