

```
# -----  
# The following Python code is implemented by Professor Terje Haukaas at  
# the University of British Columbia in Vancouver, Canada. It is made  
# freely available online at terje.civil.ubc.ca together with notes,  
# examples, and additional Python code. One objective in this file  
# is to demonstrate features and capabilities of the finite element  
# software framework OpenSees, here ran via its Python interface,  
# OpenSeesPy. There are many good resources online to learn more about  
# OpenSees. The Portwood Digital blog published at PortwoodDigital.com  
# by Professor Michael Scott at Oregon State is a nice place to start.  
# -----
```

```
# Import external libraries  
import numpy as np  
import matplotlib.pyplot as plt  
from sys import platform  
if platform == "darwin":  
    from openseespymac.opensees import *  
elif platform == "win32":  
    from openseespy.opensees import *  
else:  
    print("Cannot handle this type of operating system")  
    import sys  
    sys.exit()
```

```
# Input for the time series  
ramp = "yes"    # Ramp up from zero (yes) or make all peaks equal  
numCycles = 10 # Number of full cycles, from zero strain and back  
numPoints = 10 # Number of points *inside* ONE line (four lines per cycle)  
peakStrain = 0.01 # Maximum strain applied to the material
```

```
evolve = "yes"    # yes to track plot evolution with a dot, otherwise quicker
```

```
# Number of peaks and points
```

```
numPeaks = 2*numCycles
```

```
totNumLines = 4*numCycles
```

```
totNumPoints = numCycles*4*(numPoints+1)+1
```

```
# Set unit peak heights
```

```
peakHeights = np.ones(numPeaks)
```

```
# Scale peak heights with ramp function, if requested
```

```
if ramp == "yes":
```

```
    peakHeights[0] = 1.0/totNumLines
```

```
    for i in range(numPeaks-1):
```

```
        peakHeights[i+1] = (i+1) * (2.0/totNumLines)
```

```
# Create the first line of the time series
```

```
series = []
```

```
for j in range(numPoints + 1):
```

```
    series.append(0.0 + j * peakHeights[0] / (numPoints + 1))
```

```
# Create lines between peaks
```

```
for i in range(numPeaks-1):
```

```
    for j in range(2*numPoints + 2):
```

```
        if i % 2 == 0: # Downhill walk
```

```
            series.append(peakHeights[i] - j * (peakHeights[i+1]+peakHeights[i]) / (2*numPoints + 2))
```

```

else: # Uphill walk
    series.append(-peakHeights[i] + j * (peakHeights[i+1]+peakHeights[i]) / (2*numPoints + 2))

# Create the last line
for j in range(numPoints + 2):
    series.append(-peakHeights[len(peakHeights)-1] + j * peakHeights[len(peakHeights)-1] /
(numPoints + 1))

# Print the series to file
outFileID = open('OpenSeesUniaxialTesterSeries.out', 'w')
line = ""
for step in range(totNumPoints):

    line += ("%12.6f" % series[step])

    if step == totNumPoints - 1:
        line += '\n'
        outFileID.write(line)

    if (step + 1) % 8 == 0.0:
        line += '\n'
        outFileID.write(line)
        line = ""

outFileID.close()

# Set the model builder in OpenSees
model('basic', '-ndm', 1, '-ndf', 1)

```

```
# Define nodes
```

```
node(1, 0.0)
```

```
node(2, 1.0)
```

```
# Fix node 1
```

```
fix(1, 1)
```

```
uniaxialMaterial("FRPConfinedConcrete", 1, 27.5, 27.5, 0.002, 400., 35., 266000., 0.0, 0.222,  
0.0163, 150., 374., 363., 16., 6., 200000., 0.2, 0.8, 1.)
```

```
# Define truss element with unit area: tag ndI ndJ A matTag
```

```
element('truss', 1, 1, 2, 1.0, 1)
```

```
# Create time series and load pattern
```

```
dt = 1.0          # Increment between data points
```

```
timeSeries("Path", 1, '-dt', dt, '-filePath', 'OpenSeesUniaxialTesterSeries.out', '-factor', peakStrain)
```

```
pattern("Plain", 1, 1)
```

```
sp(2, 1, 1.0)
```

```
# Record nodal displacements (same as strains since truss length is 1.0)
```

```
recorder('Node', '-file', "OpenSeesUniaxialTesterDisplacement.out", '-closeOnWrite', '-node', 2, '-dof',  
1, 'disp')
```

```
# Record truss force (same as stress since truss area is 1.0)
```

```
recorder('Element', '-file', "OpenSeesUniaxialTesterForce.out", '-closeOnWrite', '-ele', 1, 'force')
```

```
# Analysis setup
```

```
system('UmfPack')
```

```
constraints('Penalty', 1.0e12, 1.0e12)
```

```
integrator('LoadControl', dt, 1, dt, dt)
```

```
test('NormDisplIncr', 1.0e-6, 10)
```

```
algorithm('Newton')
```

```
numberer('RCM')
```

```
analysis('Static')
```

```
# Plot the time series
```

```
plt.ion()
```

```
plt.figure(1)
```

```
plt.title("Applied Time-series")
```

```
plt.ylabel('Strain')
```

```
plt.xlabel('Pseudo-time')
```

```
plt.plot(range(totNumPoints), peakStrain*np.array(series), 'k-', linewidth=1.0)
```

```
plt.show()
```

```
# Analyze all steps at once, or track plot evolution
```

```
if evolve == "yes":
```

```
    for i in range(totNumPoints):
```

```
        # Run one analysis step
```

```
        analyze(1)
```

```
# Read output from OpenSees

dispResponse = []

outFileID = open("OpenSeesUniaxialTesterDisplacement.out", "r")
lines = outFileID.readlines()
for oneline in lines:
    splitline = oneline.split()
    for j in range(len(splitline)):
        value = float(splitline[j])
        dispResponse.append(value)
outFileID.close()

forceResponse = []

outFileID = open("OpenSeesUniaxialTesterForce.out", "r")
lines = outFileID.readlines()
for oneline in lines:
    splitline = oneline.split()
    forceResponse.append(float(splitline[1]))
outFileID.close()

# Create the plot

plt.ion()
plt.figure(2)
plt.clf()
plt.title("Material Response")
plt.ylabel('Stress')
plt.xlabel('Strain')
plt.plot(dispResponse, forceResponse, 'k-', linewidth=1.0)
plt.scatter([dispResponse[i]], [forceResponse[i]], s=100, c='k', marker='o', zorder=2)
plt.show()
```

else:

```
# Run the complete analysis
analyze(totNumPoints)

# Read output from OpenSees
dispResponse = []

f = open("OpenSeesUniaxialTesterDisplacement.out", "r")
lines = f.readlines()
for oneline in lines:
    splitline = oneline.split()
    for j in range(len(splitline)):
        value = float(splitline[j])
        dispResponse.append(value)

forceResponse = []

f = open("OpenSeesUniaxialTesterForce.out", "r")
lines = f.readlines()
for oneline in lines:
    splitline = oneline.split()
    forceResponse.append(float(splitline[1]))

# Create the plot
plt.ion()
plt.figure(2)
plt.title("Material Response")
plt.ylabel('Stress')
plt.xlabel('Strain')
plt.plot(dispResponse, forceResponse, 'k-', linewidth=1.0)
plt.show()
```