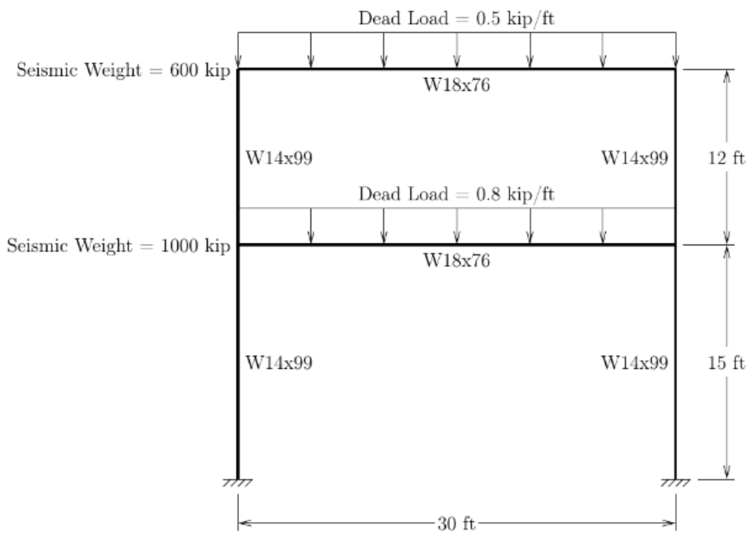
Analisis dinamico no lineal - Portico de acero GMS



In [1]:
```
import openseespy.opensees as ops
import scipy.linalg as slin
import numpy as np
import matplotlib.pyplot as plt
import math
```

1.- Unidades basicas

In [2]:
```
inch = 1.0
kip = 1.0
sec = 1.0
ksi = kip/inch**2
ft = 12*inch
g = 32.2*ft/sec**2
```

2.- Propiedades Geometricas

In [3]:
```
L = 30*ft
H1 = 15*ft
H2 = 12*ft
```

3.- Cargas sismcas

In [4]:
```
DL1 = 0.8*kip/ft
DL2 = 0.5*kip/ft

W1 = 1000*kip
m1 = W1/g + DL1*L/g

W2 = 600*kip
m2 = W2/g + DL2*L/g

print(m1,m2)
```

```
2.6501035196687366 1.591614906832298
```

4.- Propiedades del material

In [5]:
```
E = 29000*ksi
Fyb = 50*ksi
Fyc = 36*ksi
bs = 0.005
```

5.- Propiedades de las secciones

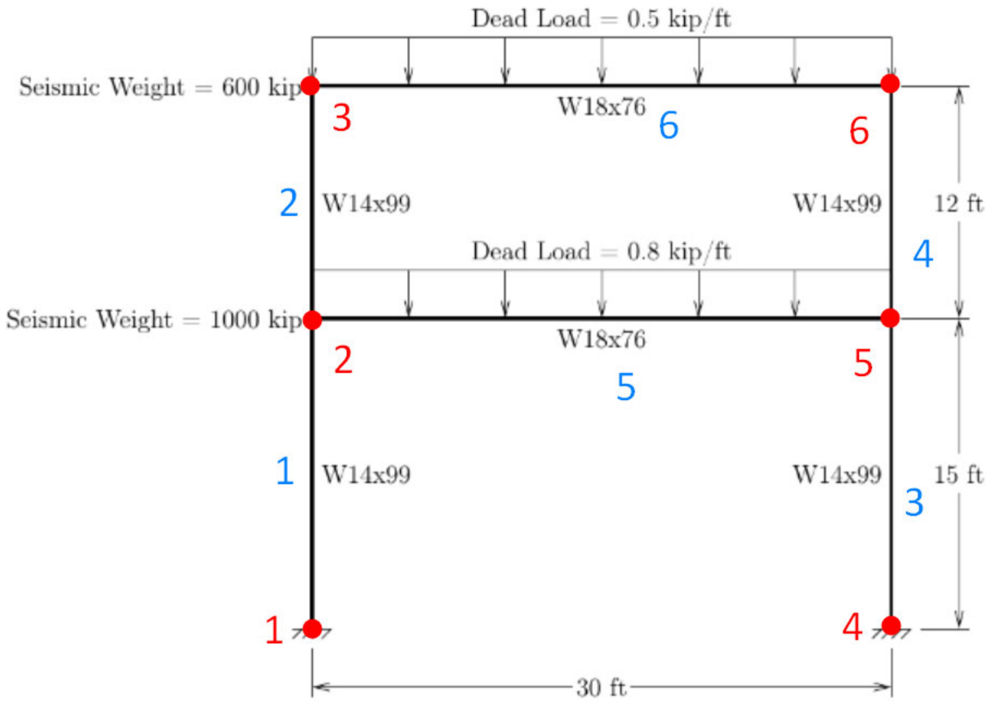
5.1.- Columna W14x99

In [6]:
```
dc = 14.16*inch
twc = 0.485*inch
bfc = 14.565*inch
tfc = 0.780*inch

Ac = 29.1*inch**2
Ic = 1110*inch**4
Sc = 157*inch**3
Zc = 173*inch**3

My = Fyc*Sc
Mp = Fyc*Zc

print(My,Mp)
```

```
5652.0 6228.0
```

5.2- Viga W18x76

In [7]:
```
db = 18.21*inch
twb = 0.425*inch
bfb = 11.035*inch
tfb = 0.680*inch

Ab = 22.3*inch**2
Ib = 1330*inch**4
Sb = 146*inch**3
Zb = 163*inch**3
```



6.- Analisis por gravedad (Cargas Estaticas)

In [8]:
```
ops.wipe()
ops.model('basic','-ndm',2,'-ndf',3)

ops.node(1,0.0,0.0); ops.fix(1,1,1,1)
ops.node(2,0.0,H1)
ops.node(3,0.0,H1+H2)
ops.node(4,L,0.0); ops.fix(4,1,1,1)
ops.node(5,L,H1)
ops.node(6,L,H1+H2)

ops.mass(2,0.5*m1,0.5*m1,0.0)
ops.mass(3,0.5*m2,0.5*m2,0.0)
ops.mass(5,0.5*m1,0.5*m1,0.0)
ops.mass(6,0.5*m2,0.5*m2,0.0)

ops.geomTransf('Linear',10)
ops.geomTransf('Linear',20)

ops.uniaxialMaterial('Steel01',1,Fyc,E,bs)
Nfw = 10
Nff = 2
ops.section('WFSection2d',1,1,dc,twc,bfc,tfc,Nfw,Nff)
Np = 5
ops.beamIntegration('Lobatto',5,1,Np)

ops.element('forceBeamColumn',1,1,2,10,5)
ops.element('forceBeamColumn',2,2,3,10,5)
ops.element('forceBeamColumn',3,4,5,10,5)
ops.element('forceBeamColumn',4,5,6,10,5)

ops.uniaxialMaterial('Steel01',2,Fyb,E,bs)
Nfw = 10
Nff = 2
ops.section('WFSection2d',2,2,db,twb,bfb,tfb,Nfw,Nff)
Np = 5
ops.beamIntegration('Lobatto',6,2,Np)

ops.element('forceBeamColumn',5,2,5,20,6)
ops.element('forceBeamColumn',6,3,6,20,6)

ops.rigidLink('beam',2,5)
ops.rigidLink('beam',3,6)

ops.timeSeries('Constant',1)
ops.pattern('Plain',1,1)
ops.eleLoad('-ele',5,'-type','-beamUniform',-DL1)
ops.eleLoad('-ele',6,'-type','-beamUniform',-DL2)

#ops.constraints('Plain')
ops.constraints('Transformation')
ops.numberer('Plain')
ops.system('FullGeneral')
ops.test('NormDispIncr',1.0e-8,10)
ops.algorithm('Newton')
ops.integrator('LoadControl',0.0)
ops.analysis('Static')
ops.analyze(1)

ops.reactions()

print(ops.nodeReaction(1,2),ops.nodeReaction(4,2))
```

```
#ops.loadConst('-time',0.0)
```

```
19.499999999999993 19.500000000000007
```

7.- Analisis de valores propios

In [9]:
```
omegaSquared = ops.eigen(2)

omega1 = omegaSquared[0]**0.5
omega2 = omegaSquared[1]**0.5

# Peridos de vibracion

T1 = 2*np.pi/omega1
T2 = 2*np.pi/omega2

print(T1,T2)

# Amortiguamiento para modos 1 y 2
zeta1 = 0.03 # Amort modo 1
zeta2 = 0.02 # Amort modo 2

# Calculo de los coeficientes de Rayleigh
B = np.zeros(shape=(2,2))
B[0,0] = 1/omega1; B[0,1] = omega1
B[1,0] = 1/omega2; B[1,1] = omega2

b = np.zeros(2)
b[0] = 2*zeta1
b[1] = 2*zeta2

a = np.linalg.solve(B,b)
```

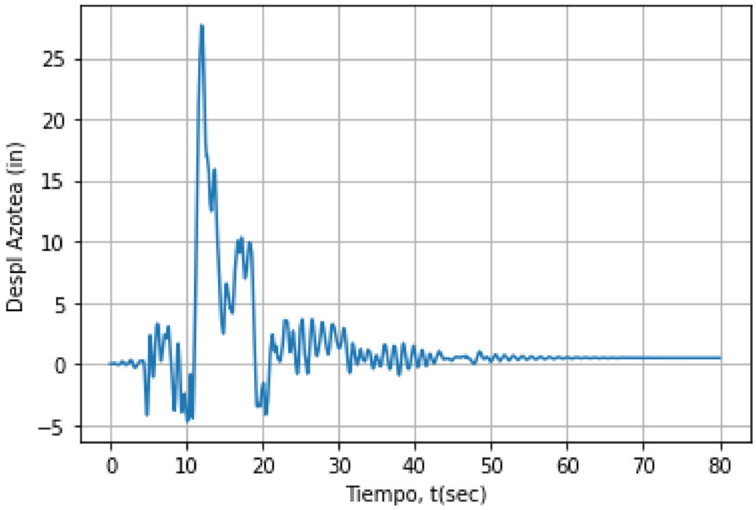
```
1.18106264545646 0.38000552560385165
```

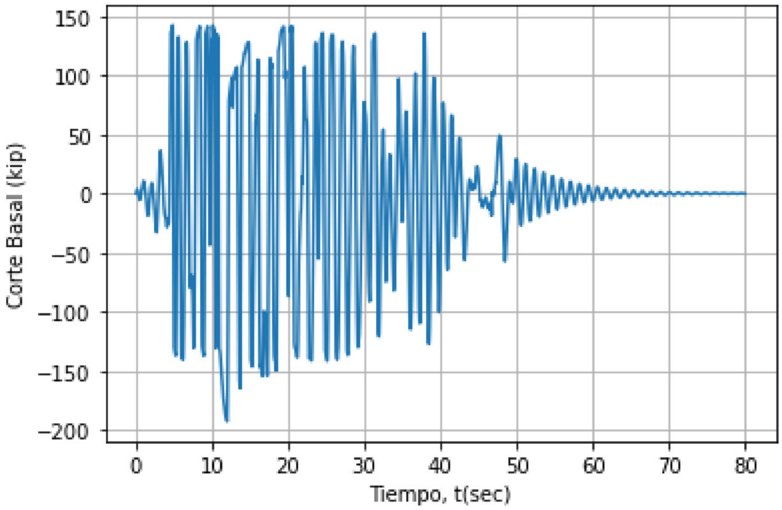
8.- Analisis de respuestas en el tiempo (tiempo-historia) no lineal

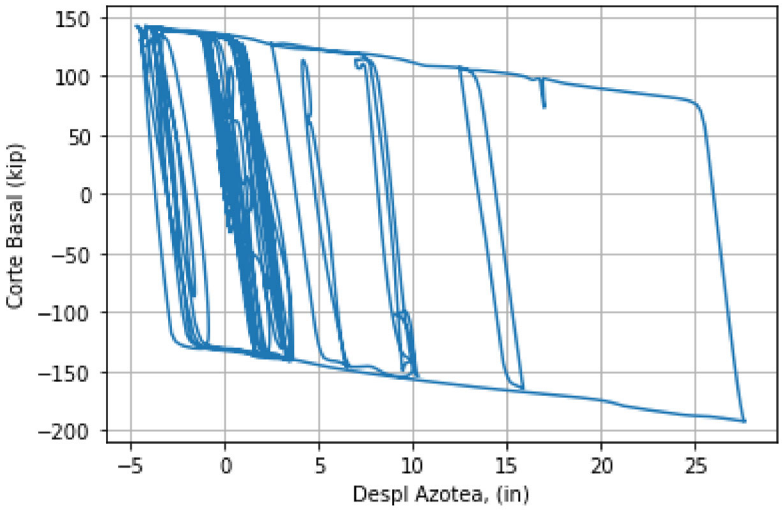
In [10]:
```
#           M    KT   KI   Kn
ops.rayleigh(a[0],0.0,0.0,a[1])

# Definir el registro
# dt = 0.02 es la digitalizacion del registro

ops.timeSeries('Path',5,'-dt',0.02,'-filePath','tabasFN.txt','-factor',1.0*g
ops.pattern('UniformExcitation',10,1,'-accel',5)

ops.integrator('Newmark',0.50,0.25)
ops.analysis('Transient') # Automatica utiliza Newmark Accel Cons gamma=1/2

tPlot = []
uPlot = []
VbPlot = []
Pl31Plot = []
Pl33Plot = []

# Retorna el tiempo
t = ops.getTime()

# Delta de t de analisis y duracion
dt =0.01*sec
Tfinal = 80.0*sec

while t < Tfinal:
    ok = ops.analyze(1,dt)
    if ok < 0:
        break
    ops.reactions()

    t = ops.getTime() # Tiempo actual
    u = ops.nodeDisp(3,1) # Desplazamiento de techo
    Vb = ops.nodeReaction(1,1) + ops.nodeReaction(4,1) # Cortante basal
    MI1 = ops.eleForce(1,3)
    MI3 = ops.eleForce(3,3)

    # Guardar los vectores para ploteo
    tPlot = np.append(tPlot,t)
    uPlot = np.append(uPlot,u)
    VbPlot = np.append(VbPlot,Vb)
    Pl31Plot = np.append(Pl31Plot,MI1)
    Pl33Plot = np.append(Pl33Plot,MI3)

plt.figure()
plt.plot(tPlot,uPlot)
plt.xlabel('Tiempo, t(sec)')
plt.ylabel('Despl Azotea (in)')
plt.grid()

plt.figure()
plt.plot(tPlot,VbPlot)
plt.xlabel('Tiempo, t(sec)')
plt.ylabel('Corte Basal (kip)')
plt.grid()

plt.figure()
plt.plot(uPlot,VbPlot)
plt.xlabel('Despl Azotea, (in)')
plt.ylabel('Corte Basal (kip)')
plt.grid()
```

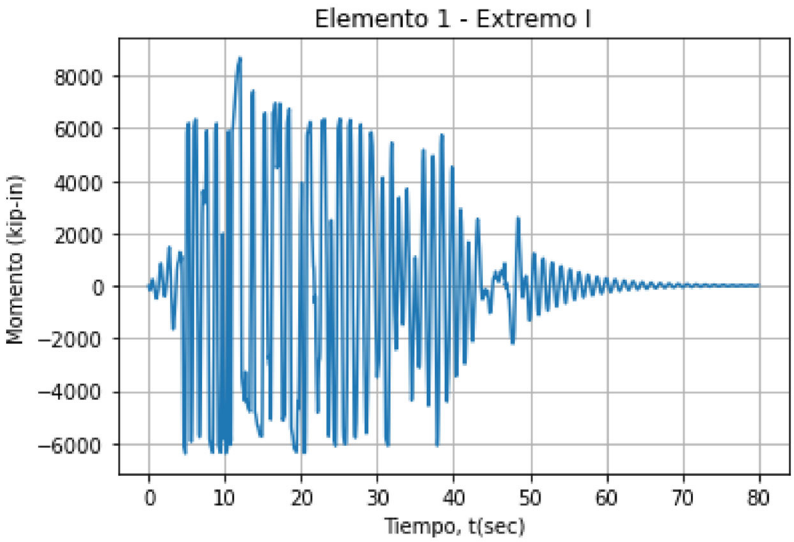
```
plt.figure()
plt.plot(tPlot,Pl31Plot)
plt.title('Elemento 1 - Extremo I')
plt.xlabel('Tiempo, t(sec)')
plt.ylabel('Momento (kip-in)')
plt.grid()

plt.figure()
plt.plot(tPlot,Pl33Plot)
plt.title('Elemento 3 - Extremo I')
plt.xlabel('Tiempo, t(sec)')
plt.ylabel('Momento (kip-in)')
plt.grid()
```

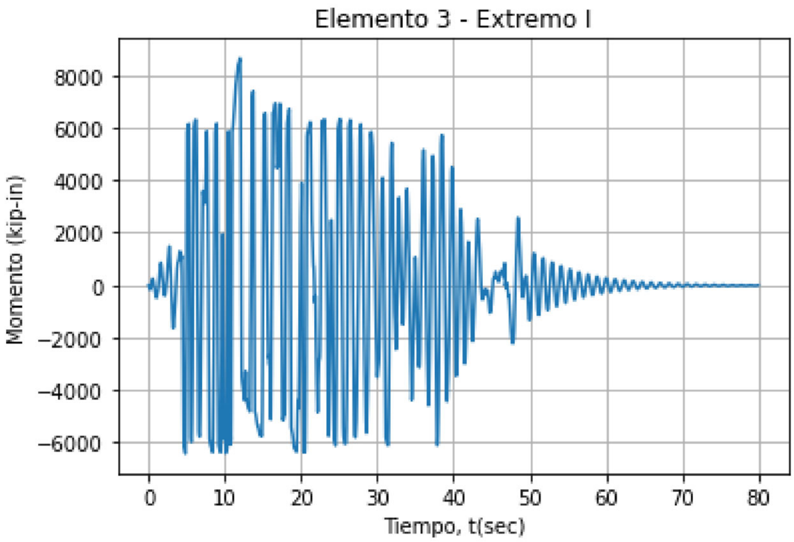
```
WARNING can't set transient integrator in static analysis
WARNING analysis Transient - no Integrator specified,
 TransientIntegrator default will be used
```






Corte Basal (kip)
Despl Azotea, (in)


Elemento 1 - Extremo I
Momento (kip-in)
Tiempo, t(sec)


Elemento 3 - Extremo I
Momento (kip-in)
Tiempo, t(sec)

10.- Analisis de valores propios - Manual

In [11]:
```
k1 = 2*(12*E*Ic/H1**3)
k2 = 2*(12*E*Ic/H2**3)

K = [[k1+k2,-k2],[-k2,k2]]
M = np.diag([m1,m2])

omegaSquared = slin.eig(K,M)[0]
modeShape = slin.eig(K,M)[1]

omega1 = math.sqrt(omegaSquared[0])
omega2 = math.sqrt(omegaSquared[1])


T1 = 2*np.pi/omega1
T2 = 2*np.pi/omega2

print(T1,T2)

x1 = modeShape[1]
x2 = modeShape[0]

print(x1,x2)
```

```
1.16902711008068 0.37463284456240714
[0.77239856 0.80754113] [ 0.63513815 -0.58981126]

C:\Users\skalw\AppData\Local\Temp/ipykernel_26756/1671664552.py:10: Comple
xWarning: Casting complex values to real discards the imaginary part
  omega1 = math.sqrt(omegaSquared[0])
C:\Users\skalw\AppData\Local\Temp/ipykernel_26756/1671664552.py:11: Comple
xWarning: Casting complex values to real discards the imaginary part
  omega2 = math.sqrt(omegaSquared[1])
```