In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.polynomial.legendre import leggauss as raices
from IPython.display import clear_output
from PySections import *
import quadpy
```

Punto 2

In [ ]:

```python
#CODIGO FUENTE
def estadoPlasticidadConcentrada(vt,sh,qy,EI,l,EA,tipo,v0,q=[[0],[0],[0]]):
    qy = np.array(qy)
    vt = np.array(vt)
    v0 = np.array(v0)
    q = np.array(q)
    error = 1
    i = 1
    while error > 1*10**-10 and i <= 50:
        q1 = np.min([q[0][0],-1*10**-5])
        psi = np.sqrt(((-q1)*l**2)/(EI))
        fe = _fe(psi,l,EI,EA,tipo)
        fp = _fp(q,qy,EI,l,sh,EA)
        if np.abs(q[0][0]) > np.abs(qy[0][0]):
            fe = np.zeros([3,3])
            fp = np.zeros([3,3])
        kb = np.linalg.pinv(fe + fp)
        ve = fe @ q
        vp = fp @ (q - np.abs(qy)*np.sign(q))
        v = vp + ve
        Re = vt - v0 - v
        dq = kb @ Re
        if np.abs(q[0][0]) > np.abs(qy[0][0]):
            q = [[qy[0][0]],[0],[0]]
            ve = [[l/EA*qy[0][0]],[0],[0]]
            vp = vt - ve
            v = vp + ve
            Re = np.zeros([3,1])
            kb = np.zeros([3,3])
            break
        q = q + dq
        i +=1
        error = np.linalg.norm(Re)
        print('Error q: ' + format(error) + ' iteracion ' + format(i))
    return Re,v,q,kb,ve,vp

def _fp(q,qy,EI,l,sh,EA=1,sh2=None):
    alpha0 = 1*(1-(np.abs(q[0][0]) <= np.abs(qy[0][0])))
    alpha1 = 1*(1-(np.abs(q[1][0]) <= np.abs(qy[1][0])))
    alpha2 = 1*(1-(np.abs(q[2][0]) <= np.abs(qy[2][0])))
    kbc2 = (6*EI/l)*sh
    if sh2 == None:
        sh2 = sh
    kbc3 = (6*EI/l)*sh2
    return np.array([[alpha0*l/EA,0,0],[0,alpha1/kbc2,0],[0,0,alpha2/kbc3]])
def _fe(psi,l,EI,EA,tipo=Tipo.UNO):
    L = l
    if psi < 0.001:
        kb1 = 4*EI/L
        kb2 = 2*EI/L
        kb3 = 3*EI/L
    else:
        kb1=((EI)/(L))*((psi*(np.sin(psi)-psi*np.cos(psi)))/(2-2*np.cos(psi)-psi*np.sin(ps
i)))
        kb2=(EI*(psi*(psi-np.sin(psi))))/(L*(2-2*np.cos(psi)-psi*np.sin(psi)))
```

```
        kb3=(L*(np.sin(psi)-psi*np.cos(psi)))/(EI*(psi**2*np.sin(psi)))
    fe1 = (kb1)/(kb1**2-kb2**2)
    fe2 = -(kb2)/(kb1**2-kb2**2)
    fe3 = kb3
    if tipo == Tipo.UNO:
        fe = np.array([[L/EA,0,0],[0,fe1,fe2],[0,fe2,fe1]])
    elif tipo == Tipo.DOS:
        fe = np.array([[L/EA,0,0],[0,0,0],[0,0,fe3]])
    elif tipo == Tipo.TRES:
        fe = np.array([[L/EA,0,0],[0,fe3,0],[0,0,0]])
    else:
        fe = np.array([[L/EA,0,0],[0,0,0],[0,0,0]])
    return fe
def calcularPsi(q,l,EI):
    q1 = np.min([q[0][0],-1*10**-5])
    return
```

In [2]:

```python
from PySections import *

desp = np.array([0,0,3*10**-3,0,0,2*10**-3])
sh = 0.05
qy = [[1100],[435],[435]]
I = 250000
A = 1
E = 1
l = 8.5


W = 12.5


estructura = Estructura()

acero = Material('Acero', E, 0.2, 9.9*10**-6, 23.54,sh=sh)
seccion = Seccion('Elementos', TipoSeccion.GENERAL, [A,I], acero,qy=qy)

estructura.agregarNodo(x=0,y=0,fix=[False,False,False])
estructura.agregarNodo(x=l,y=0,fix=[False,False,False])
estructura.agregarElemento(nodoInicial=0,nodoFinal=1,seccion=seccion,tipo=Tipo.UNO,defCortante=False)
estructura.agregarCargaDistribuida(WY=-W, elemento=-1)

for i in estructura.elementos:
    i.Ue = desp[np.ix_(i.diccionario)]

KLL,P = estructura.determinacionDeEstado()

print('\nVector {v}:')
print(estructura.elementos[0].v)
print('\nVector {q}:')
print(estructura.elementos[0].q)
print('\nMatriz [Kb]:')
print(estructura.elementos[0].kb)
print('\nVector {ve}:')
print(estructura.elementos[0].ve)
print('\nVector {vp}:')
print(estructura.elementos[0].vp)
print('\nVector {v0}:')
print(estructura.elementos[0].v0)
#estructura.solucionar(True,True)
```

```
Error q: 0.004339668361730001 iteracion 2
Error q: 0.012562847222222236 iteracion 3
Error q: 1.3904866443919908e-17 iteracion 4

Vector {v}:
[[0.   ]
 [0.003]
 [0.002]]

Vector {q}:
[[  0.        ]
 [442.73362688]
 [284.94677668]]

Matriz [Kb]:
[[1.17647059e-01 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 8.20793434e+03 4.10396717e+03]
 [0.00000000e+00 4.10396717e+03 9.02872777e+04]]

Vector {ve}:
[[0.        ]
 [0.00340295]
 [0.00072057]]

Vector {vp}:
[[0.        ]
 [0.00087648]
 [0.        ]]

Vector {v0}:
[[ 0.        ]
 [-0.00127943]
 [ 0.00127943]]
```

Punto 3

In [3]:

```
from PySections import *
from IPython.display import clear_output
desp = np.array([0.2056,-0.0055,-0.0409,0.2029,-0.0058,-0.0305,0.2024,-0.0057,0,0,0,0,0,0,
0,0,0,0]) #Desplazamietos

sh = 0.0345

qy = [[9*10**9],[187],[187]]

IV = 0.00048699076842
AV = 0.01077417

IC = 0.00044537
AC = 0.02277415

IR = 1
AR = 0.00064516

E = 200000000

l = 8
ll = 4

W = 10

estructura = Estructura()

acero = Material('Acero', E, 0.2, 9.9*10**-6,1,sh=sh)


VIGA1 = Seccion('Elementos', TipoSeccion.GENERAL, [AV,IV], acero, qy=qy)
RIOSTRA = Seccion('Elementos', TipoSeccion.GENERAL, [AR,IR], acero, qy=[[187],[0],[0]])
COLUMNA = Seccion('Elementos', TipoSeccion.GENERAL, [AC,IC], acero)

estructura.agregarNodo(x=0,y=0,fix=[False,False,False])
estructura.agregarNodo(x=l,y=0,fix=[False,False,False])
estructura.agregarNodo(x=2*l,y=0,fix=[False,False,False])

estructura.agregarNodo(x=0,y=ll)
estructura.agregarNodo(x=l,y=ll)
estructura.agregarNodo(x=2*l,y=ll)

estructura.agregarElemento(nodoInicial=0,nodoFinal=3,seccion=COLUMNA,tipo=Tipo.DOS,defCort
ante=False)

estructura.agregarElemento(nodoInicial=1,nodoFinal=4,seccion=COLUMNA,tipo=Tipo.DOS,defCort
ante=False)

estructura.agregarElemento(nodoInicial=2,nodoFinal=5,seccion=COLUMNA,tipo=Tipo.CUATRO,defC
ortante=False)


#
estructura.agregarElemento(nodoInicial=3,nodoFinal=4,seccion=VIGA1,tipo=Tipo.UNO,defCortan
te=False)
```

```python
estructura.agregarElemento(nodoInicial=4,nodoFinal=5,seccion=VIGA1,tipo=Tipo.TRES,defCortante=False)

estructura.agregarElemento(nodoInicial=0,nodoFinal=4,seccion=RIOSTRA,tipo=Tipo.CUATRO,defCortante=False)

estructura.agregarElemento(nodoInicial=1,nodoFinal=5,seccion=RIOSTRA,tipo=Tipo.CUATRO,defCortante=False)

estructura.agregarCargaDistribuida(WY=-W, elemento=3)
estructura.agregarCargaDistribuida(WY=-W, elemento=4)

estructura.agregarCargaNodo(nodo=3, px=100, py=-450)
estructura.agregarCargaNodo(nodo=4, py=-450)
estructura.agregarCargaNodo(nodo=5, py=-450)

estructura.solucionar(True,False)

for i in estructura.elementos:
    i.Ue = desp[np.ix_(i.diccionario)]
KLL, PL = estructura.determinacionDeEstado()
clear_output()
import os
cwd = os.getcwd()
np.savetxt(cwd+'/KLL.csv',KLL,delimiter=',')
np.savetxt(cwd+'/PL.csv',PL,delimiter=',')
```

Para elementos no columnas, es decir i = [3,4,5,6]

In [4]:

```
i = 6
EA = estructura.elementos[i].E*estructura.elementos[i].Area
EI =  estructura.elementos[i].E*estructura.elementos[i].Inercia
l = estructura.elementos[i].Longitud
sh = 0.0345
qy = estructura.elementos[i].seccion.qy
tipo = estructura.elementos[i].Tipo
v = estructura.elementos[i].v
v0 = estructura.elementos[i].v0
sh = estructura.elementos[i].seccion.material.sh
Re,v,q,kb,ve,vp = estadoPlasticidadConcentrada(v,sh,qy,EI,l,EA,tipo,v0)
print('Vector {v}')
print(v)
print('Vector v0 {v0}')
print(v0)
print('\nMatriz [kb]')
print(np.round(kb,3))
print('\nVector {Re}')
print(Re)
print('\nVector q {q}')
print(np.round(q,3))
print('\nVector vp {vp}')
print(np.round(vp,3))
print('\nVector ve {ve}')
print(np.round(ve,3))
```

```
Error q: 0.1789839918157199 iteracion 2
Vector {v}
[[0.17898399]
 [0.        ]
 [0.        ]]
Vector v0 {v0}
[[0.]
 [0.]
 [0.]]

Matriz [kb]
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]

Vector {Re}
[[0.]
 [0.]
 [0.]]

Vector q {q}
[[187]
 [  0]
 [  0]]

Vector vp {vp}
[[0.166]
 [0.   ]
 [0.   ]]

Vector ve {ve}
[[0.013]
 [0.   ]
 [0.   ]]
```

Para elementos de columnas i = [0,1,2]

In [5]:

```
i = 2
EA = estructura.elementos[i].E*estructura.elementos[i].Area
EI =  estructura.elementos[i].E*estructura.elementos[i].Inercia
l = estructura.elementos[i].Longitud
sh = 0.0345
qy = estructura.elementos[i].seccion.qy
tipo = estructura.elementos[i].Tipo
v = estructura.elementos[i].v
v0 = estructura.elementos[i].v0
kb = estructura.elementos[i].kb
q = estructura.elementos[i].q
#sh = estructura.elementos[i].seccion.material.sh
#Re,v,q,kb,ve,vp = estadoPlasticidadConcentrada(v,sh,qy,EI,l,EA,tipo,v0)
print('Vector {v}')
print(v)
print('Vector v0 {v0}')
print(v0)
print('\nMatriz [kb]')
print(np.round(kb,3))
print('\nVector q {q}')
print(np.round(q,3))
```

```
Vector {v}
[[-0.00057526]
 [ 0.        ]
 [ 0.        ]]
Vector v0 {v0}
[[0.]
 [0.]
 [0.]]

Matriz [kb]
[[1138707.5       0.        0. ]
 [      0.        0.        0. ]
 [      0.        0.        0. ]]

Vector q {q}
[[-655.053]
 [   0.   ]
 [   0.   ]]
```

Punto 4

In [6]:

```
def estadoPlasticidadDistribuida(vt, l, v0=[[0], [0], [0]],q=[[0], [0], [0]],n=5):
    vt = np.array(vt)
    v0 = np.array(v0)
    q = np.array(q)
    i = 0
    error = 1
    while error > 1*10**-10:
        v, kb, e, S = _vkb(l, q,n)
        Re = vt - v0 - v
        dq = kb @ Re
        q = q + dq
        i +=1
        error = np.max(Re)
        print('Error q: ' + format(error) + ' iteracion ' + format(i))
    return Re, v, q, kb, e, S

def _vkb(L, q, n):
    s = quadpy.line_segment.gauss_lobatto(n)
    X = (np.array(s.points)/2+1/2)*L
    W = np.array(s.weights)/2*L
    v = np.zeros([3, 1])
    kb = np.zeros([3, 3])
    Eg = []
    Sg = []
    for i in range(0, len(X)):
        x = X[i]
        b = np.array([[1, 0, 0], [0, x / L - 1, x / L]])
        St = b @ q
        e = np.zeros([2, 1])
        fibras = crearFibras()
        error = 1
        j = 1
        while error > 1*10**-6 and j < 30:
            S, Ks,FIG = _estadoSeccion(e, fibras)
            Rs = St - S
            de = np.linalg.pinv(Ks) @ Rs
            e = e + de
            error = np.linalg.norm(de)
            clear_output(wait=True)
            print('Error s: ' + format(error) + ' iteracion ' + format(i) + ',' + format(j
))
            j += 1
        v = v + W[i] * (b.T @ e)
        kb = kb + W[i] * (b.T @ np.linalg.pinv(Ks) @ b)
        Eg.append(e)
        Sg.append(FIG)
    kb = np.linalg.pinv(kb)
    return v, kb, Eg,Sg

def _estadoSeccion(e, fibras):
    n = fibras.shape[0]
    ea = e[0][0]
    phi = e[1][0]
    ys = fibras[:, 0].reshape(n, 1)
    ai = fibras[:, 1].reshape(n, 1)
```

```
    epsilon = ea - ys * phi
    sigma, Et = _esfdeft(epsilon)
    Sm = sigma * ai
    C = (Et * ai).T[0]
    km = np.diag(C)
    As = np.array([np.zeros([n]) + 1, -ys.T[0]]).T
    S = As.T @ Sm

    Ks = As.T @ km @ As
    return S, Ks, Et

def _esfdeft(epsilon, sh=0.015,ey = 0.001725,E=200000000):  # CURVA ESFUERZO DEFORMACION
    sh = 1-sh
    et = E - sh * E * (np.abs(epsilon) > ey)
    s = et * (epsilon-ey*np.sign(epsilon)*(np.abs(epsilon) > ey)) + ey*np.sign(epsilon)*E*
(np.abs(epsilon) > ey)
    return s, et


#Cambair la seccion transversar es cuestion del usuario
def crearFibras():
    tf = 0.940*2.54/100
    th = 0.590*2.54/100
    a = 14.670*2.54/100
    b = 14.48*2.54/100 - 2 * tf

    tf = 0.01143*2
    th = 0.01397
    a = 0.3099
    b = 0.09229*3

    a1 = a*tf/2
    a2 = b*th/3

    d1 = tf/2
    d2 = b/3
    fibras = [[-(d2/2+d2+d1+d1/2),a1],
              [-(d2/2+d2+d1/2),a1],
              [-(d2),a2],
              [0,a2],
              [(d2),a2],
              [(d2/2+d2+d1/2),a1],
              [(d2/2+d2+d1+d1/2),a1]]
    return np.array(fibras)

vt = [[-0.001076],[0.05186],[-0.01091]]
l = 4.5

Re, v, q, kb, e, S = estadoPlasticidadDistribuida(vt, l, v0=[[0], [0], [0]],n=5)
print('Vector {v}')
print(v)
print('\nMatriz [kb]')
print(kb)
print('\nVector {Re}')
print(Re)
print('\nVector q {q}')
```

```
print(q)
print('\nVector e {e}')
print(np.array(e))
print('\nVector Et {Et}')
print(np.array(S))
```

```
Error s: 2.6073335674753502e-20 iteracion 4,2
Error q: 4.336808689942018e-19 iteracion 6
Vector {v}
[[-0.001076]
 [ 0.05186 ]
 [-0.01091 ]]

Matriz [kb]
[[372451.2986584    5719.60323452 -11268.31013864]
 [  5719.60323452   3780.67210521   3135.73761191]
 [-11268.31013864   3135.73761191  44533.77799889]]

Vector {Re}
[[ 4.33680869e-19]
 [-6.93889390e-18]
 [ 0.00000000e+00]]

Vector q {q}
[[-3.89429421e+02]
 [ 9.47017648e+02]
 [-1.61440915e-01]]

Vector e {e}
[[[-1.26397130e-03]
  [-1.41465311e-01]]

 [[-3.77421151e-04]
  [-1.37754854e-02]]

 [[-1.07955917e-04]
  [-6.95133933e-03]]

 [[-1.07955917e-04]
  [-2.40217081e-03]]

 [[-1.07955917e-04]
  [-2.36962702e-06]]]

Vector Et {Et}
[[[3.e+06]
  [3.e+06]
  [3.e+06]
  [2.e+08]
  [3.e+06]
  [3.e+06]
  [3.e+06]]

 [[3.e+06]
  [3.e+06]
  [2.e+08]
  [2.e+08]
  [2.e+08]
  [2.e+08]
  [3.e+06]]

 [[2.e+08]
  [2.e+08]
```

```
 [2.e+08]
 [2.e+08]
 [2.e+08]
 [2.e+08]
 [2.e+08]]

[[2.e+08]
 [2.e+08]
 [2.e+08]
 [2.e+08]
 [2.e+08]
 [2.e+08]
 [2.e+08]]

[[2.e+08]
 [2.e+08]
 [2.e+08]
 [2.e+08]
 [2.e+08]
 [2.e+08]
 [2.e+08]]]
```

Graficas Punto 4

In [7]:

```python
import numpy as np
import matplotlib.pyplot as plt
ey = 0.001725
l = []
fig,axes = plt.subplots(1,2,figsize=[16,8])
ax = axes[0]
for i in np.linspace(0,.5,5):
    _X = np.linspace(0,2*ey,100)
    _Y = _esfdeft(_X, sh=i,ey=ey)[0]
    l.append(r'$\alpha_{sh}='+format(np.round(i,2))+'$')
    ax.plot(_X,_Y)
ax.legend(l)
ax.grid()
ax.set_xlabel(r'$\varepsilon$')
ax.set_ylabel(r'$\sigma$')
ax.set_title('Curvas esfuerzo - deformación')
ax = axes[1]
for i in np.linspace(0,.5,5):
    _X = np.linspace(0,2*ey,300)
    _Y = _esfdeft(_X, sh=i,ey=ey)[1]
    l.append(r'$\alpha_{sh}='+format(np.round(i,2))+'$')
    ax.plot(_X,_Y)
ax.legend(l)
ax.grid()
ax.set_xlabel(r'$\varepsilon$')
ax.set_ylabel(r'$E_T$')
ax.set_title('Modulo Tangente')
```

Out[7]:

```
Text(0.5, 1.0, 'Modulo Tangente')
```