# Homeworks for SM-I course

Important:

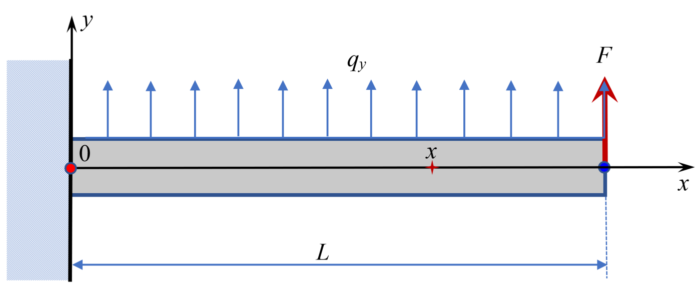
1). Please download the latest grcodes, images, and related chapters before working on the homework.

2). Both pdf files and the source codes must be submitted, or the work will not be marked.

# Homework 6: Solutions for beams

**Question 1:**

Consider a cantilever beam of uniform cross-section, as shown in the following image. The beam is clamped at the left-end. It has length $L = 1m$ a square section area of $A = 0.0001m^2$. It is subjected to a distributed body force $q_y$ and a concentrated force $F$ at the right end.




A cantilever beam of uniform cross-section, subjected to a distributed body-force, and a concentrated force at the right-end.

1. Consider only the body force $b_y = q$ (N/m) where $q$ is a constant, derive by hand the formulas for computing the deflection, $v$ in the $y$-direction, cross-section rotation, moment, shear force, and normal stress $\sigma_{xx}$ in the beam, as functions of $x$. Compare the solutions with the corresponding ones obtained using the code given in the textbook.

2. Given data, $A = 0.0001m^2$, $L = 1m$, Young's modulus of the material $E = 2.1e^{10}Pa$, $q = 500N/m$, $F = 1500N$, compute and plot the distributions of the deflection, cross-section rotation, moment, shear force, and the maximum normal stress on the cross-section, along the coordinate $x$.

In [ ]:
```
# Place curse in this cell, and press Ctrl+Enter to import dependences.
import sys                            # for accessing the computer system
sys.path.append('../grbin/')  # Change to the directory in your system

from commonImports import *        # Import dependences from '../grbin/'
import grcodes as gr                  # Import the module of the author
#importlib.reload(gr)              # When grcodes is modified, reload it

from continuum_mechanics import vector
init_printing(use_unicode=True)       # For latex-like quality printing
np.set_printoptions(precision=4,suppress=True,
       formatter={'float': '{:0.4e}'.format})   # Digits in print-outs
```

1.

$\frac{d^4v}{dx^4} = \frac{b_y}{EI_z}$
$\frac{d^3v}{dx^3} = V = x\frac{b_y}{EI_z} + c_3$
$\frac{d^2v}{dx^2} = M = x^2\frac{b_y}{2EI_z} + xc_3 + c_2$
$\frac{dv}{dx} = \theta = x^3\frac{b_y}{6EI_z} + \frac{1}{2}x^2c_3 + xc_2 + c_1$
$v = x^4\frac{b_y}{24EI_z} + \frac{1}{6}x^3c_3 + \frac{1}{2}x^2c_2 + xc_1 + c_0$

$b_y = q$

Beam is fixed at $x = 0$ , therefore:
$v = 0$
$\theta = 0$

at $x = 0$

$0 = 0^4\frac{q}{24EI_z} + \frac{1}{6}0^3c_3 + \frac{1}{2}0^2c_2 + 0c_1 + c_0$
$c_0 = 0$

$0 = 0^3\frac{q}{6EI_z} + \frac{1}{2}0^2c_3 + 0c_2 + c_1$
$c_1 = 0$

Therefore:
$v = x^4\frac{q}{24EI_z} + \frac{1}{6}x^3c_3 + \frac{1}{2}x^2c_2$
$\theta = x^3\frac{q}{6EI_z} + \frac{1}{2}x^2c_3 + xc_2$

At $x = L$:
$V = qL$
$M = 0$

$qL = L\frac{q}{EI_z} + c_3$
$c_3 = qL(1 - \frac{1}{EI_z})$

$0 = L^2\frac{q}{2EI_z} + LqL(1 - \frac{1}{EI_z}) + c_2$
$c_2 = -L^2q(\frac{1}{2EI_z} + 1 - \frac{1}{EI_z})$
$c_2 = -L^2q(1 - \frac{1}{2EI_z})$

$$\frac{3l^2q}{2} - 2lqx + \frac{qx^2}{2}$$

Therefore:
$\upsilon = x^4\frac{q}{24EI_z} + x^3\frac{qL}{3EI_z} + x^2\frac{3L^2q}{4EI_z}$
$\theta = x^3\frac{q}{6EI_z} + x^2\frac{qL}{EI_z} + x\frac{3L^2q}{2EI_z}$
$M = x^2\frac{q}{2} + 2xqL - \frac{3L^2q}{2}$
$V = 2qL - qx$

Lastly:
$\sigma_{xx} = -y\frac{M}{I_z}$
$\sigma_{xx} = \frac{-y}{I_z}(x^2\frac{q}{2} + 2xqL - \frac{3L^2q}{2})$

In [ ]:

```python
def solver1D4(E, I, by, l, v0, θ0, vl, θl, V0, M0, Vl, Ml, key='c-c'):
    '''Solves the Beam Equation for integrable distributed body force:
    u,x4=-by(x)/EI, with various boundary conditions (BCs):
    c-c, s-c, c-s, f-c, c-f.
    Input: EI: bending stiffness factor; by, body force; l, the length
    of the beam; v0, θ0, V0, M0, deflection, rotation,
    shear force, moment at x=0; vl, θl, Vl, Ml, those at x=l.
    Return: u, v_x, v_x2, v_x3, v_x4 up to 4th derivatives of v
    '''
    c0, c1, c2, c3 = symbols('c0, c1, c2, c3') #integration constant
    EI = E*I # I is the Iz in our chosen coordinates.
    # Integrate 4 times:
    v_x3= sp.integrate(by/EI,(x, 0, x))+ c0 #ci: integration constant
    v_x2= sp.integrate( v_x3,(x, 0, x))+ c1
    v_x = sp.integrate( v_x2,(x, 0, x))+ c2
    v = sp.integrate( v_x,(x, 0, x))+ c3
    # Solve for the 4 integration constants:
    if key == "s-s":
        cs=sp.solve([v.subs(x,0)-v0, v_x2.subs(x,0)-M0/EI,v.subs(x,l)-vl, v_x2.subs
    elif key == "c-c":
        cs=sp.solve([v.subs(x,0)-v0, v_x.subs(x,0)-θ0,v.subs(x,l)-vl, v_x.subs(x,l)
    elif key == "s-c":
        cs=sp.solve([v.subs(x,0)-v0, v_x2.subs(x,0)-M0/EI,v.subs(x,l)-vl, v_x.subs(
        #print('solver1D4:',cs[c0],cs[c1],cs[c2],cs[c3])
    elif key == "c-s":
        cs=sp.solve([v.subs(x,0)-v0, v_x.subs(x,0)-θ0,v.subs(x,l)-vl, v_x2.subs(x,l
    elif key == "c-f":
        cs=sp.solve([v.subs(x,0)-v0, v_x.subs(x,0)-θ0,v_x3.subs(x,l)+Vl/EI, v_x2.su
    elif key == "f-c":
        cs=sp.solve([v_x3.subs(x,0)-V0/EI, v_x2.subs(x,0)-M0/EI,v.subs(x,l)-vl, v_x
    else:
        print("Please specify boundary condition type.")
        sys.exit()
    # Substitute the constants back to the integral solutions
    v = v.subs({c0:cs[c0],c1:cs[c1],c2:cs[c2],c3:cs[c3]})
    v = v.expand().simplify().expand()
    v_x = v_x.subs({c0:cs[c0],c1:cs[c1],c2:cs[c2],c3:cs[c3]})
    v_x = v_x.expand().simplify().expand()
    v_x2=v_x2.subs({c0:cs[c0],c1:cs[c1],c2:cs[c2],c3:cs[c3]})
    v_x2=v_x2.expand().simplify().expand()
    v_x3=v_x3.subs({c0:cs[c0],c1:cs[c1],c2:cs[c2],c3:cs[c3]})
    v_x3=v_x3.expand().simplify().expand()
    v_x4 = sp.diff(v_x3,x).expand()
    print("Outputs form solver1D4(): v, θ, M, V, qy")
    return v,v_x,(EI*v_x2).expand(),(-EI*v_x3).expand(),(v_x4).expand()
```

In [ ]:
```
E, I, EI, A, l, ξ = symbols('E, I, EI, A, l, ξ ', nonnegative=True)
x, q = symbols('x, q ') # geometry & force
c0, c1, c2, c3 = symbols('c0, c1, c2, c3') #integration constant
# for displacement boundary conditions (DBCs):
v0, θ0, vl, θl = symbols('v0, θ0, v_l, θ_l') # DBCs
# for force boundary conditions (FBCs):
V0, M0, Vl, Ml = symbols('V0, M0, V_l, M_l') # FBCs
by = q

v = solver1D4(E,I,by,l,0,0,vl,θl,V0,M0,by*l,0,key='c-f')
```

```
Outputs form solver1D4(): v, θ, M, V, qy
```

$v$:

In [ ]:
```
v[0]
```

Out[ ]:
$$\frac{3l^2qx^2}{4EI} - \frac{lqx^3}{3EI} + \frac{qx^4}{24EI}$$

$$v = x^4\frac{q}{24EI_z} + x^3\frac{qL}{3EI_z} + x^2\frac{3L^2q}{4EI_z}$$

$\theta$:

In [ ]:
```
v[1]
```

Out[ ]:
$$\frac{3l^2qx}{2EI} - \frac{lqx^2}{EI} + \frac{qx^3}{6EI}$$

$$\theta = x^3\frac{q}{6EI_z} + x^2\frac{qL}{EI_z} + x\frac{3L^2q}{2EI_z}$$

$M$:

In [ ]:
```
v[2]
```

Out[ ]:
$$\frac{3l^2q}{2} - 2lqx + \frac{qx^2}{2}$$

$$M = x^2\frac{q}{2} + 2xqL - \frac{3L^2q}{2}$$

$V$:

In [ ]:
```
v[3]
```

Out[ ]:
$$2lq - qx$$

$$V = 2qL - qx$$

2.

```
In [ ]: E, I, EI, A, l, ξ = symbols('E, I, EI, A, l, ξ ', nonnegative=True)
x, q, F= symbols('x, q, F') # geometry & force
c0, c1, c2, c3 = symbols('c0, c1, c2, c3') #integration constant
# for displacement boundary conditions (DBCs):
v0, θ0, vl, θl = symbols('v0, θ0, v_l, θ_l') # DBCs
# for force boundary conditions (FBCs):
V0, M0, Vl, Ml = symbols('V0, M0, V_l, M_l') # FBCs
by = q
Vl = F

v = solver1D4(E,I,by,l,0,0,vl,θl,V0,M0,by*l,F,key='c-f')
v
```

```
Outputs form solver1D4(): v, θ, M, V, qy
```

Out[ ]: $\left(\frac{Fx^2}{2EI} + \frac{3l^2qx^2}{4EI} - \frac{lqx^3}{3EI} + \frac{qx^4}{24EI}, \frac{Fx}{EI} + \frac{3l^2qx}{2EI} - \frac{lqx^2}{EI} + \frac{qx^3}{6EI}, F + \frac{3l^2q}{2} - 2lqx + \right.$

In [ ]:
```
def plot2curveS(u, xL=0., xR=1., title="f_title"):
    '''Print out maximum values and loctions, as well as stationary
    points, and the values at the stationary points, and boundaries'''
    x = sp.symbols('x')
    dx = 0.01; dxr = dx*10       # x-step
    xi = np.arange(xL, xR+dx, dx)
    uf = sp.lambdify((x), u[0], 'numpy')   #convert Sympy f to numpy f
    yi = uf(xi)
    if type(yi) != np.ndarray:       #in case, uf is a constant
        #type(yi) == int or type(yi) == float: # or len(yi)==1:
        xi = np.arange(xL, xR+dxr, dxr)
        yi = float(yi)*np.ones_like(xi)

    fig, ax1 = plt.subplots(figsize=(5.,1.), dpi=300)
    fs = 8          # fontsize
    color = 'black'
    ax1.set_xlabel('location x', fontsize=fs)
    ax1.set_ylabel(title[0], color=color, fontsize=fs)
    ax1.plot(xi, yi, color=color)
    ax1.grid(color='r',ls=':',lw=.3, which='both') # Use both tick
    ax1.tick_params(axis='x', labelcolor=color, labelsize=fs)
    ax1.tick_params(axis='y', labelcolor=color, labelsize=fs)

    vmax = yi[yi.argmax()]
    max_l = np.argwhere(yi == vmax)
    ax1.plot(xi[max_l],  yi[max_l], 'r*', markersize=4)
    print(f'Maximum {title[0]} value={vmax:.3e}, at x={xi[max_l][0][0]}')

    uf = sp.lambdify((x), u[1], 'numpy') #convert Sympy f to numpy f
    xi = np.arange(xL, xR+dx, dx)
    yi2 = uf(xi)
    if type(yi2) != np.ndarray: # or len(yi2) == 1:
        xi = np.arange(xL, xR+dxr, dxr)
        yi2 = float(yi2)*np.ones_like(xi)

    m1, m2, m3 = np.partition(abs(yi2), 2)[0:3]
    msl=[np.where(abs(yi2)==m1)[0][0],np.where(abs(yi2)==m2)[0][0],
         np.where(abs(yi2)==m3)[0][0]]

    vmax = yi2[yi2.argmax()]
    max_l = np.argwhere(yi2 == vmax)
    print(f'Maximum {title[1]} value={vmax:.3e}, at x={xi[max_l][0][0]}')

    if abs(xi[msl[2]]-xi[msl[1]])<2*dx:
        if abs(yi2[msl[2]]-0.)<abs(yi2[msl[1]]-0.): msl.pop(1)
        else: msl.pop(2)
    if len(msl) > 2:
        if abs(xi[msl[2]]-xi[msl[0]])<2*dx:
            if abs(yi2[msl[2]]-0.)<abs(yi2[msl[0]]-0.): msl.pop(0)
            else: msl.pop(2)
    if len(msl) > 1:
        if abs(xi[msl[1]]-xi[msl[0]])<2*dx:
            if abs(yi2[msl[1]]-0.)<abs(yi2[msl[0]]-0.): msl.pop(0)
            else: msl.pop(1)

    ax2 = ax1.twinx()  # instantiate second axes sharing the same x-axis
```

```python
    color = 'blue'
    ax2.set_ylabel(title[1], color=color, fontsize=fs)
    ax2.plot(xi, yi2, color=color)
    ax2.plot(xi[max_l], yi2[max_l], 'r*', markersize=4)
    ax2.plot(xi[msl], yi2[msl], 'r*', markersize=4)
    ax1.plot(xi[msl],  yi[msl], 'r*', markersize=4)
    ax2.plot(xi[0], yi2[0], 'ro', markersize=2)
    ax1.plot(xi[0],  yi[0], 'ro', markersize=2)
    ax2.plot(xi[-1], yi2[-1], 'ro', markersize=2)
    ax1.plot(xi[-1],  yi[-1], 'ro', markersize=2)
    ax2.grid(color='r',ls=':',lw=.5, which='both') # Use both tick
    ax2.tick_params(axis='x', labelcolor=color, labelsize=fs)
    ax2.tick_params(axis='y', labelcolor=color, labelsize=fs)
    np.set_printoptions(formatter={'float': '{: 0.3e}'.format})
    print(f'Extreme {title[0]} values={yi[msl]},\n    at x={xi[msl]}')
    print(f'Critical {title[1]} values={yi2[msl]},\n    at x={xi[msl]}')
    print(f'{title[0]} values at boundary ={yi[0], yi[-1]}')
    print(f'{title[1]} values at boundary ={yi2[0], yi2[-1]}\n')
```

In [ ]:

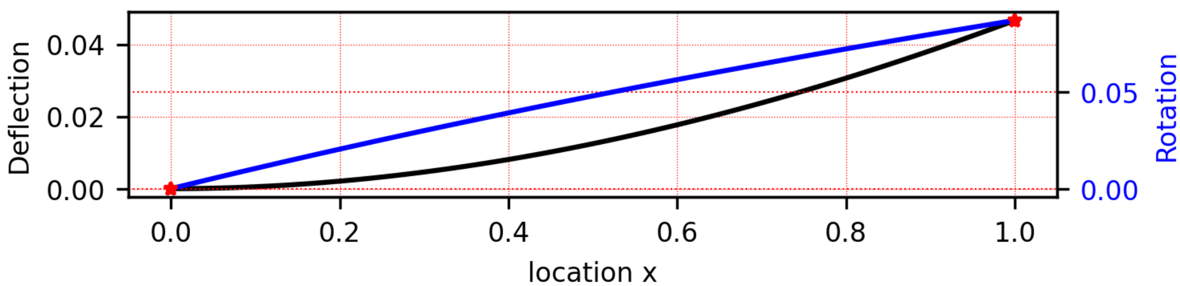
```python
A = 1*(10**-4)
# Since its a square cross section, y = sqrt(A)
y = np.sqrt(A)

Iact = A*y


dic = {E:2.1*(10**10),I:Iact,l:1,q:500,v0:0,θ0:0,vl:0,θl:0,F:1500}
vx=[v[i].subs(dic) for i in range(len(v))]
vx # Outputs are: v, θ, Mx, Vx, qy
```

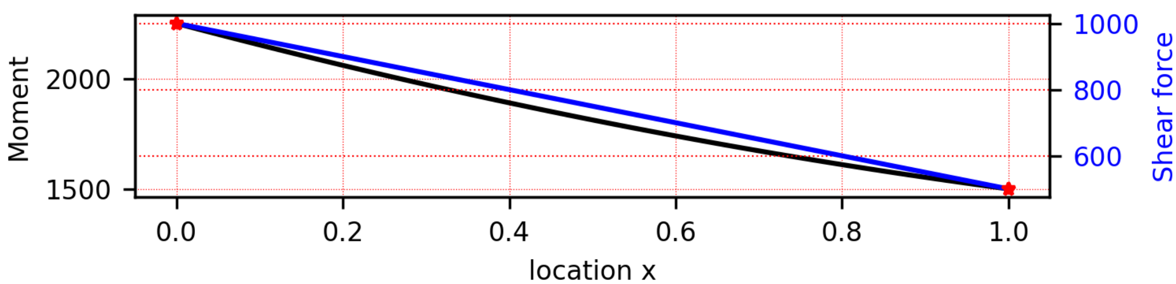
Out[ ]: $[0.000992063492063492x^4 - 0.00793650793650793x^3 + 0.0535714285714286x^2,\ 0.00396$

In [ ]:

```python
title = [["Deflection", "Rotation"], ["Moment", "Shear force"]]
for i in range(len(title)):
    plot2curveS(vx[2*i:2*(i+1)], 0., 1., title=title[i])

#fig.tight_layout()
#plt.savefig('images/beam_cq.png', dpi=500) # save the plot to file
plt.show()
```

```
Maximum Deflection value=4.663e-02, at x=1.0
Maximum Rotation value=8.730e-02, at x=1.0
Extreme Deflection values=[ 0.000e+00],
    at x=[ 0.000e+00]
Critical Rotation values=[ 0.000e+00],
    at x=[ 0.000e+00]
Deflection values at boundary =(0.0, 0.04662698412698416)
Rotation values at boundary =(0.0, 0.08730158730158716)

Maximum Moment value=2.250e+03, at x=0.0
Maximum Shear force value=1.000e+03, at x=0.0
Extreme Moment values=[ 1.500e+03],
    at x=[ 1.000e+00]
Critical Shear force values=[ 5.000e+02],
    at x=[ 1.000e+00]
Moment values at boundary =(2250.0, 1500.0)
Shear force values at boundary =(1000.0, 500.0)
```




In [ ]:
```
def roundS(expr, n_d): # to be used in maxminS()
    '''To limit the number of digits to keep in a variable.
    Usage: roundS(expr, n_d), where n_d: number of digits to keep.
    '''
    return expr.xreplace({n.evalf():round(n,n_d)
    for n in expr.atoms(sp.Number)})
```

In [ ]:
```
def maxminS(f, title="value"):
    '''Find maximum location and values of a symbolic function
    '''
    ndg = 8 # number of digits
    df = sp.diff(f, x)
    ddf = sp.diff(df, x)
    df0_points = sp.solve(df, x) #find stationary points
    df0s = [roundS(point.evalf(),ndg) for point in df0_points]
    print(f"Stationary points for {title}: {df0s}")
    for point in df0s:
        fv = roundS((f.subs(x,point)).evalf(), ndg)
        ddfv = ddf.subs({x:point})
        if ddfv < 0:
            print(f"At x={point}, local maximum {title}={fv}")
        elif ddfv > 0:
            print(f"At x={point}, local minimum {title}={fv}")
        else:
            print(f"At x={point}, max or min {title}={fv}")
    df0s.append(0) # Add in points on the boundaries
    df0s.append(1)
    fs_df0 = [f.subs(x, point).evalf() for point in df0s]
    f_max = roundS(max(fs_df0), ndg)
    x_max=[pnt for pnt in df0s if ma.isclose(f.subs(x,pnt),max(fs_df0))]
    print(f"\nAt x={x_max}, Max {title}={f_max}\n")
    return x_max, f_max
```

In [ ]:
```
maxMX, maxM = maxminS(vx[2], title="moment")
```

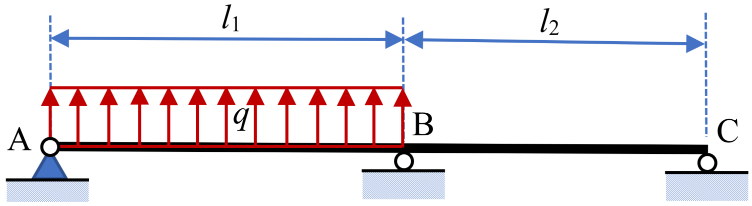
```
Stationary points for moment: [2.00000000000000]
At x=2.00000000000000, local minimum moment=1250.00000000000

At x=[0], Max moment=2250.00000000000
```

In [ ]:
```
maxstress = -y*(maxM/Iact)
print(f"\nAt x={maxMX}, Max Normal Stress={maxstress*(10**-6)}\n")
```

```
At x=[0], Max Normal Stress=-22.5000000000000
```

**Question 2:**

Consider a beam with two spans. Span-1 has a length $l_1$, and span-2 has a length $l_2$. The beam is simple-simple-simple supported (s-s-s), as shown in the following image. It is subjected to a distributed force over span-1.




A beam with two spans. It is simple-simple-simple supported.

1. Derive formulas for solutions, including deflection, cross-section rotation, moment, and shear force in the beam.

2. Set all the variables with unit values, plot the distribution of all these solutions.

3. Discuss about the results obtained.

1.

$l_1$

$$\frac{d^4v}{dx^4} = \frac{b_y}{EI_z}$$

$$\frac{d^3v}{dx^3} = V = x\frac{b_y}{EI_z} + c_3$$

$$\frac{d^2v}{dx^2} = M = x^2\frac{b_y}{2EI_z} + xc_3 + c_2$$

$$\frac{dv}{dx} = \theta = x^3\frac{b_y}{6EI_z} + \frac{1}{2}x^2c_3 + xc_2 + c_1$$

$$v = x^4\frac{b_y}{24EI_z} + \frac{1}{6}x^3c_3 + \frac{1}{2}x^2c_2 + xc_1 + c_0$$

$$b_y = q$$

Beam is pinned at $x = 0$, therefore:
$v = 0$
$M = 0$

at $x = 0$

$v = 0 = 0^4 \frac{q}{24EI_z} + \frac{1}{6}0^3 c_3 + \frac{1}{2}0^2 c_2 + 0c_1 + c_0$
$c_0 = 0$

$M = 0 = 0^2 \frac{q}{2EI_z} + 0c_3 + c_2$
$c_2 = 0$

Beam is pinned at $x = l_1$, therefore:
$v = 0$
$M = 0$

at $x = l_1$, however since there is a second span, $M = M_{l_1}$

$\frac{d^2 v}{dx^2} = M_{l_1} = {l_1}^2 \frac{q}{2EI_z} + l_1 c_3$
$\frac{M_{l_1}}{l_1} = l_1 \frac{q}{2EI_z} + c_3$
$c_3 = \frac{M_{l_1}}{l_1} - l_1 \frac{q}{2EI_z}$

$M = x^2 \frac{q}{2EI_z} + x(\frac{M_{l_1}}{l_1} - l_1 \frac{q}{2EI_z})$
$M = x(\frac{qx}{2EI_z} + \frac{M_{l_1}}{l_1} - \frac{ql_1}{2EI_z})$
$M = \frac{ql_1}{2} - \frac{M_{l_1}}{l_1} - qx$

$v = 0 = {l_1}^4 \frac{q}{24EI_z} + \frac{1}{6}{l_1}^3(\frac{M_{l_1}}{l_1} - l_1 \frac{q}{2EI_z}) + l_1 c_1$
$0 = {l_1}^4 \frac{q}{24EI_z} + \frac{1}{6}M_{l_1}{l_1}^2 - {l_1}^4 \frac{q}{12EI_z} + l_1 c_1$
$0 = \frac{M_{l_1} l_1}{6} - \frac{q{l_1}^3}{24EI_z} + c_1$
$c_1 = \frac{q{l_1}^3}{24EI_z} - \frac{M_{l_1} l_1}{6}$

Therefore:
$v = x^4 \frac{q}{24EI_z} - x^3 \frac{ql_1}{12EI_z} + x^3 \frac{M_{l_1}}{6l_1 EI_z} + x \frac{q{l_1}^3}{24EI_z} - x \frac{M_{l_1} l_1}{6EI_z}$
$\theta = x^3 \frac{q}{6EI_z} - x^2 \frac{ql_1}{4EI_z} + x^2 \frac{M_{l_1}}{2l_1 EI_z} + \frac{q{l_1}^3}{24EI_z} - \frac{M_{l_1} l_1}{6EI_z}$
$M = x^2 \frac{q}{2} - x\frac{ql_1}{2} + x\frac{M_{l_1}}{l_1}$
$V = qx - \frac{ql_1}{2} + \frac{M_{l_1}}{l_1}$

In [ ]:
```python
# Define variables:
title = [["Deflection", "Rotation"], ["Moment", "Shear force"]]
E, I, l1, l2, q = symbols("E, I, l1, l2, q")
x = symbols("x")
# for displacement boundary conditions (DBCs):
v10, θ10, v1l, θ1l = symbols('v10, θ10, v_1l, θ_1l') # DBCs for span-1
v20, θ20, v2l, θ2l = symbols('v20, θ20, v_2l, θ_2l') # DBCs for span-2
# for force boundary conditions (FBCs):
V10, M10, V1l, M1l = symbols('V10, M10, V_1l, M_1l') # FBCs for span-1
V20, M20, V2l, M2l = symbols('V20, M20, V_2l, M_2l') # FBCs for span-2
```

In [ ]:
```python
# Distributed force is zero for span-1
by = q # BC is s-s
v1 = solver1D4(E,I,by,l1, v10,θ10,v1l,θ1l, V10, M10, V1l, M1l,key='s-s')
print(f'Is solution correct? {by/E/I==v1[4]}; The solution v(x) is:')
v1[0] # solution of general displacement function
```

```
Outputs form solver1D4(): v, θ, M, V, qy
Is solution correct? True; The solution v(x) is:
```

Out[ ]:

$$v_{10} - \frac{v_{10}x}{l_1} + \frac{v_{1l}x}{l_1} - \frac{M_{10}l_1x}{3EI} + \frac{M_{10}x^2}{2EI} - \frac{M_{10}x^3}{6EIl_1} - \frac{M_{1l}l_1x}{6EI} + \frac{M_{1l}x^3}{6EIl_1} + \frac{l_1^3qx}{24EI} - \frac{l_1qx^3}{12EI}$$

In [ ]:
```python
# Set known BCs for span-1:
v1x=[v1[i].subs({v10:0,M10:0,v1l:0}) for i in range(len(v1))]
v1x # solution of v, θ, Mx, Vx, qy
```

Out[ ]:

$$\left[-\frac{M_{1l}l_1x}{6EI} + \frac{M_{1l}x^3}{6EIl_1} + \frac{l_1^3qx}{24EI} - \frac{l_1qx^3}{12EI} + \frac{qx^4}{24EI}, \ -\frac{M_{1l}l_1}{6EI} + \frac{M_{1l}x^2}{2EIl_1} + \frac{l_1^3q}{24EI} - \frac{l_1qx^2}{4EI}\right.$$

$l_2$

$$\frac{d^4v}{dx^4} = \frac{b_y}{EI_z}$$
$$\frac{d^3v}{dx^3} = V = x\frac{b_y}{EI_z} + c_3$$
$$\frac{d^2v}{dx^2} = M = x^2\frac{b_y}{2EI_z} + xc_3 + c_2$$
$$\frac{dv}{dx} = \theta = x^3\frac{b_y}{6EI_z} + \frac{1}{2}x^2c_3 + xc_2 + c_1$$
$$v = x^4\frac{b_y}{24EI_z} + \frac{1}{6}x^3c_3 + \frac{1}{2}x^2c_2 + xc_1 + c_0$$

$b_y = 0$

$\frac{d^4v}{dx^4} = 0$

$\frac{d^3v}{dx^3} = V = c_3$

$\frac{d^2v}{dx^2} = M = xc_3 + c_2$

$\frac{dv}{dx} = \theta = \frac{1}{2}x^2c_3 + xc_2 + c_1$

$v = \frac{1}{6}x^3c_3 + \frac{1}{2}x^2c_2 + xc_1 + c_0$

Conditions from previous deirvation at $x = l_1$
$v = 0$
$M = M_{l_1}$

at $x = l_1$

Treat this as its own function, so $x = l_1$ is $x = 0$:
$v = 0 = \frac{1}{6}0^3c_3 + \frac{1}{2}0^2c_2 + 0c_1 + c_0$

$c_0 = 0$

$M = M_{l_1} = 0c_3 + c_2$

$c_2 = M_{l_1}$

Beam is pinned at $x = l_2$, therefore:
$v = 0$
$M = 0$

at $x = l_2$

$M = 0 = l_2c_3 + M_{l_1}$

$c_3 = M_{l_1}$

$v = 0 = \frac{1}{6}{l_2}^3M_{l_1} + \frac{1}{2}{l_2}^2M_{l_1} + l_2c_1$

$c_1 = -\frac{1}{6}{l_2}^3M_{l_1} - \frac{1}{2}{l_2}^2M_{l_1}$

Therefore:
$v = -x^3\frac{M_{l_1}}{6EI_zl_2} + x^2\frac{M_{l_1}}{2EI_z} - x\frac{M_{l_1}l_2}{3EI_z}$

$\theta = -x^2\frac{M_{l_1}}{2EI_zl_2} + x\frac{M_{l_1}}{EI_z} - \frac{M_{l_1}l_2}{3EI_z}$

$M = -x\frac{M_{l_1}}{l_2} + M_{l_1}$

$V = \frac{M_{l_1}}{l_2}$

In [ ]:
```python
# General solution for span-2: (constant q)
# BC is simple-clamp
v2 = solver1D4(E,I,0,l2, v20,θ20,v2l, θ2l, V20, M20, V2l, M2l,key='s-s')
print(f'Is solution correct? {by/E/I==v2[4]}; The solution u(x) is:')
v2 # solution of general displacement function
```

```
Outputs form solver1D4(): v, θ, M, V, qy
Is solution correct? False; The solution u(x) is:
```

Out[ ]: $\left(v_{20} - \frac{v_{20}x}{l_2} + \frac{v_{2l}x}{l_2} - \frac{M_{20}l_2x}{3EI} + \frac{M_{20}x^2}{2EI} - \frac{M_{20}x^3}{6EIl_2} - \frac{M_{2l}l_2x}{6EI} + \frac{M_{2l}x^3}{6EIl_2}, \ -\frac{v_{20}}{l_2} + \frac{v_{2l}}{l_2}\right.$

In [ ]:
```python
# Set known BCs for span-2:
v2x=[v2[i].subs({v20:0,M20:M1l,v2l:0,M2l:0}) for i in range(len(v2))]
v2x # solution of v, θ, Mx, Vx, qy
```

Out[ ]: $\left[-\frac{M_{1l}l_2x}{3EI} + \frac{M_{1l}x^2}{2EI} - \frac{M_{1l}x^3}{6EIl_2}, \ -\frac{M_{1l}l_2}{3EI} + \frac{M_{1l}x}{EI} - \frac{M_{1l}x^2}{2EIl_2}, \ M_{1l} - \frac{M_{1l}x}{l_2}, \ \frac{M_{1l}}{l_2}, \ 0\right]$

2.

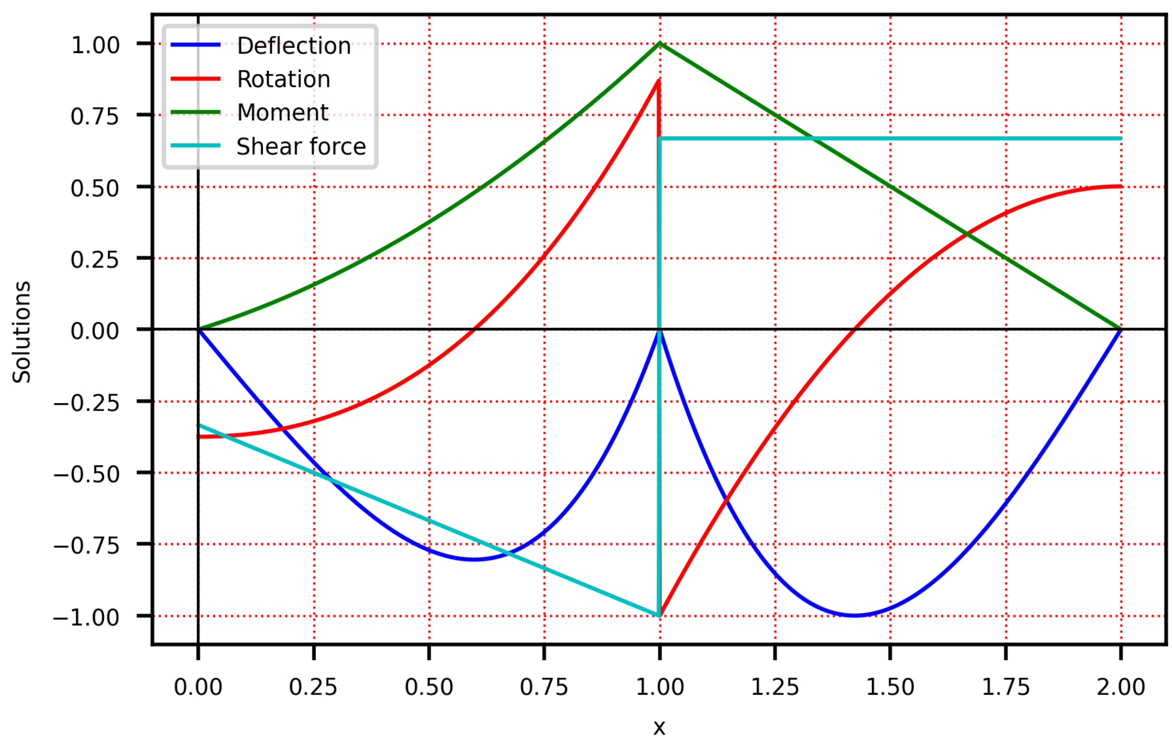
Using the following values: $l_1 = 1m$, $l_2 = 1m$, $q = 50n/m$, $E = 1 * 10^{10}Pa$, $I_z = 1$.
Using this, $M_{l_1} = l_1(ql_1) = 50$.

In [ ]:
```python
sln1x=[v1x[i].subs({q:50,l1:1,l2:1,E:1*(10**10),I:1,M1l:50}) for i in range(len(v1x
sln2x=[v2x[i].subs({l1:1,l2:1,E:1*(10**10),I:1,M1l:50}) for i in range(len(v2x))]
```

In [ ]:
```python
def plot_fs(x, y, labels, xlbl, ylbl, *p_name):
    '''plot multiple x-y curves in one figure.
    x: x data, list of 1D numpy array;
    y: y data, list of 1D numpy array
    labels: labels for these curves
    xlbl: for ax.set_xlabel()
    ylbl: for ax.set_ylabel()
    p_name: string, name of the plot.
    '''
    colors = ['b', 'r', 'g', 'c', 'm','k','y','w']
    plt.rcParams.update({'font.size': 5})          # settings for plots
    fig_s = plt.figure(figsize=(4,2.5))
    ax = fig_s.add_subplot(1,1,1)
    for i in range(len(y)):
        plt.plot(x[i], y[i], label=labels[i], color=colors[i],lw=0.9)

    ax.grid(c='r', linestyle=':', linewidth=0.5)
    ax.axvline(x=0, c="k", lw=0.6);ax.axhline(y=0, c="k", lw=0.6)
    ax.set_xlabel(xlbl); ax.set_ylabel(ylbl)
    ax.legend() #loc='center right', bbox_to_anchor=(1, 0.5))
    #plt.title('x-y curves'+p_name+'')

    plt.savefig('images/'+p_name[0]+'.png',dpi=500,bbox_inches='tight')
    plt.show()
```

In [ ]:

```
np.set_printoptions(formatter={'float': '{: 0.4e}'.format})
l1 = 1.0; l2 = 1.0
dx = 0.002
x1 = np.arange(0.0, l1, dx) # coordinates in l1, left span
x2 = np.arange(0.0, l2, dx) # coordinates in l2, right span
# convert Sympy solution func. to numpy arrays.
n1 = len(sln1x)-1
n2 = len(sln1x)-1
x = symbols("x")
sln1 = [sp.lambdify(x, sln1x[i], 'numpy') for i in range(n1)]
sln2 = [sp.lambdify(x, sln2x[i], 'numpy') for i in range(n2)]
sx1 = [sln1[i](x1) for i in range(n1)]
sx2 = [sln2[i](x2) for i in range(n2)]
# The shear force function could be a constant. Need to generate array
if type(sx1[-1]) != np.ndarray: # in case, func. is a constant
    sx1[-1] = float(sx1[-1])*np.ones_like(x1)
if type(sx2[-1]) != np.ndarray: # in case, func. is a constant
    sx2[-1] = float(sx2[-1])*np.ones_like(x2)
X = np.concatenate((x1, l1+x2)) # put the results in a single array
sx = [np.concatenate((sx1[i], sx2[i])) for i in range(n1)]
sx_max = [np.max(abs(sx[i])) for i in range(n1)]
# Plot the distribution curves:
x_data = [X for i in range(n1)] # put x coordinates to a list
y_data = [sx[i]/sx_max[i] for i in range(n1)] # func. values to a list
labels=["Deflection", "Rotation", "Moment", "Shear force"]
plot_fs(x_data, y_data, labels, 'x', 'Solutions', 'temp')
```




3.

The results obtained seemed to be in line with what is expected. The central pin acts like a fulcrum for the beam, and all 4 diagrams show that the beam reacts around this point.