Méthode des éléments finis: TP1

Ibrahim ALAME

4/10/2022

Une poutre à treillis est une structure formée par un agencement triangulaire d'éléments linéaires (ou quasi linéaires) dont les extrémités sont reliées au niveau de points d'assemblage appelés nœuds. Les poutres à treillis se composent de triangles, c'est-à-dire des formes géométriquement stables. En effet, un triangle présente des angles fixes qui ne peuvent être ni agrandis, ni rétrécis sans céder au niveau des points d'assemblage, contrairement, notamment, à un rectangle, lequel peut devenir un parallélogramme.

Problème variationnelle et matrice élémentaire

Une barre est une poutre qui ne transmet que des efforts de traction compression à ses extrémités. On rappelle la formulation variationnelle, vue en TD3, d'une barre OL homogène de module E, section constante A et de longueur L soumise à une sollicitation linéïque $p(x)$ supposée nulle ici et à deux forces aux extrémités $\vec{f_1}$ et $\vec{f_2}$ s'écrit comme un problème abstrait de la forme:

$$(\mathcal{P}_v) \begin{cases} \text{Trouver } u \in V = H^1(\Omega) \text{ vérifiant} \\ a(u,v) = \ell(v) \quad \forall v \in V \end{cases}$$

où

$$a(u,v) = EA \int_0^L \frac{\mathrm{d}u}{dx} \cdot \frac{\mathrm{d}v}{dx} \mathrm{d}x \quad \text{et} \quad \ell(v) = f_1 \cdot v(0) + f_2 \cdot v(L)$$

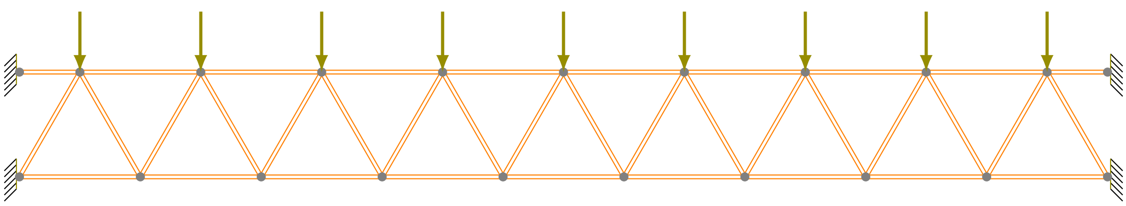
Dans un maillage en $n+1$ points équidistants, si on choisit un élément fini de Lagrange de type (1) de longueur h, le système élémentaire s'écrit alors:

$$\begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = \frac{EA}{h} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$$

Pont en treillis

Le pont en treillis que l'on se propose d'étudier dans cette partie est constitué de $N = 10$ sommets sur le tablier et $N+1$ sommets sur la poutre supérieur. Les triangles formés par les barres articulées comme indique la figure ci-après sont équilatéraux. Les 4 sommets aux extrémités sont encastrés. les poutres sont de même nature et de même section droite. Soient L la longueur du pont, E le module de Young du matériau, A l'aire des sections droites et P est la force appliquée verticalement vers le bas en chaque nœud intérieur de la poutre supérieur (en $N-1$ sommets).

Application numérique : on donne : $A = 100\text{mm}^2$, $L = 10m$, $E = 200000\text{MPa}$, $P = -300\text{N}$.



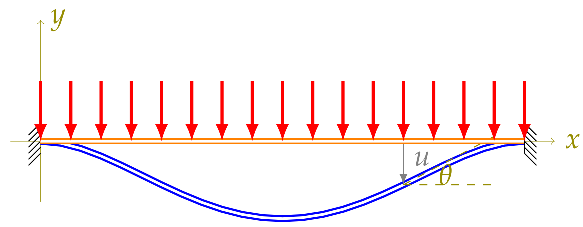
On admet que le système élémentaire d'une barre faisant un angle θ avec l'horizontal est donnée par:

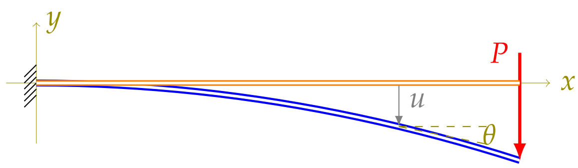
$$\begin{pmatrix} F_1^x \\ F_1^y \\ F_2^x \\ F_2^y \end{pmatrix} = \frac{EA}{L} \begin{pmatrix} C^2 & CS & -C^2 & -CS \\ CS & S^2 & -CS & -S^2 \\ -C^2 & -CS & C^2 & CS \\ -CS & -S^2 & CS & S^2 \end{pmatrix} \begin{pmatrix} U_1^x \\ U_1^y \\ U_2^x \\ U_2^y \end{pmatrix}$$

où $C = \cos\theta$ et $S = \sin\theta$.

1. Copier l'annexe A dans votre document Datalore et faire tourner le programme python.
2. Déterminer les réactions des appuis et la flèche maximale.

Poutre en flexion





La formulation variationnelle d'une poutre en flexion simple est de la forme (TD3):

$$(\mathcal{P}_v) \begin{cases} \text{Trouver } u \in V \text{ vérifiant} \\ a(u,v) = \ell(v) \quad \forall v \in V \end{cases}$$

où

$$a(u,v) = EI_z \int_0^L \frac{\mathrm{d}^2 u}{dx^2} \cdot \frac{\mathrm{d}^2 v}{dx^2} \mathrm{d}x$$

et

$$\ell(v) = \int_0^L p \cdot v(x)\mathrm{d}x + F_0 \cdot v(0) + F_L \cdot v(L) + M_0 \cdot \frac{\partial v}{\partial x}(0) + M_L \cdot \frac{\partial v}{\partial x}(L)$$

On choisit un élément fini de Hermite de type (1) qui utilise les déplacements et leur dérivées comme degrés de liberté. D'où la matrice élémentaire :

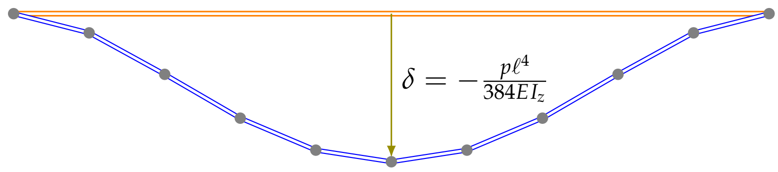
$$\begin{pmatrix} f_1^y \\ M_1 \\ f_2^y \\ M_2 \end{pmatrix} = \frac{EI}{L^3} \begin{pmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{pmatrix} \begin{pmatrix} u_1^y \\ \theta_1 \\ u_2^y \\ \theta_2 \end{pmatrix}$$

Écrire un programme python permettant de résoudre numériquement l'équation de la poutre en flexion simple par la méthode des éléments finis.

Application numérique: on donne : $A = r^2$, $I = \frac{1}{12}r^4$, où $r = 2$cm , $L = 10m$, $E = 200$GPa , $p = -200$N.

Exécuter votre programme python dans les deux cas suivants:

1. La poutre est encastrée aux extrémités (déplacement u et angle θ sont simultanément nuls en 0 et en L).

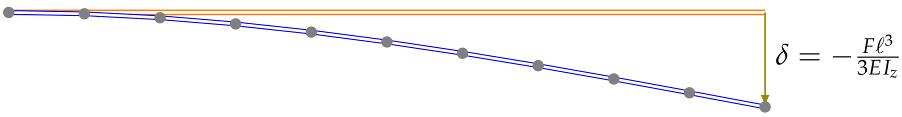


On rappelle (cf TD3) que le second membre élémentaire d'une poutre soumise uniquement à des charges linéiques de densité $p(x)$ est:

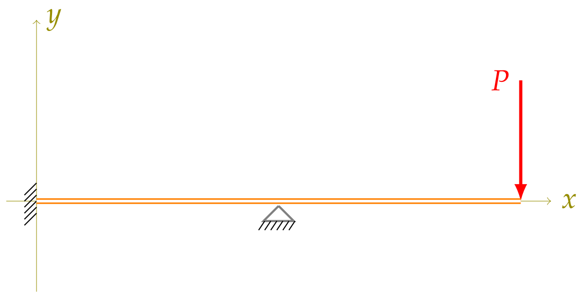
$$\begin{pmatrix} f_1^y \\ M_1 \\ f_2^y \\ M_2 \end{pmatrix} = \begin{pmatrix} \frac{pL}{2} \\ \frac{pL^2}{12} \\ \frac{pL}{2} \\ -\frac{pL^2}{12} \end{pmatrix}$$

2. La poutre est encastrée à son origine 0 (déplacement u et angle θ sont simultanément nuls). Une force verticale $F = -10N$ est appliquée à l'autre extrémité L, avec un moment nul $M_L = 0$. On rappelle (cf TD3) que le second membre élémentaire d'une poutre soumise uniquement à une force ponctuelle F en son extrémité L est:

$$\begin{pmatrix} f_1^y \\ M_1 \\ f_2^y \\ M_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ F \\ 0 \end{pmatrix}$$



3. On reprend la poutre de la question 2 et on ajoute un appui simple au milieu:



Dans les trois cas calculer les flèches maximales, les angles de torsion et les réactions aux points particuliers. Comparer avec les résultats théoriques.

Annexe A: Exercice1

[1]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
```

Données et constantes physiques

- Le pont est de longueur $L = 10$ m.
- Le nombre des sommets sur la partie inférieur du pont est $N + 1 = 11$.
- Les poutres ont une même longueur: $h = 1$m, et une même section rectangulaire 10 mm $\times$ 20 mm, soit $A = 200 \times 10^{-6}$ m^2.
- On applique en chaque noeud du tablier une charge $P = -3000$ N.
- Le module de Young est $E = 200000$ MPa.

[2]:
```
L = 10.
A = 200 * 1E-6
E = 200 * 1E9
P = 30000 #50000
N=10 # Nombre des poutres sur la partie inférieur du pont.
h=L/N    # longueur élémentaire
```

On pose $r_3 = \sqrt{3}$

[3]:
```
r3=math.sqrt(3)
```

Géométrie du problème

Maillage: les éléments finis sont les poutres

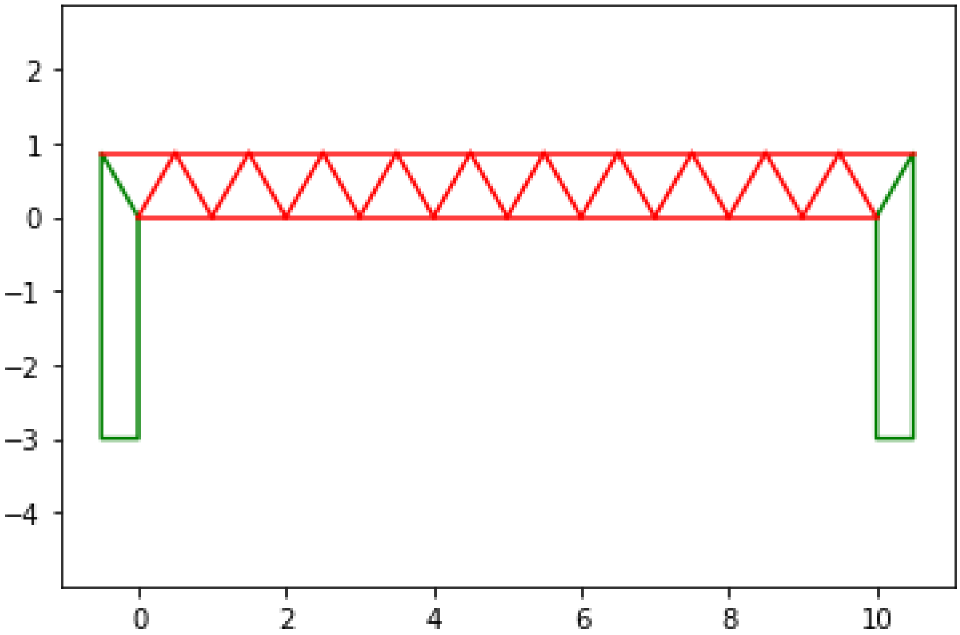
[4]:
```
Points=[]  # Liste des sommets
Points.append([-h/2,r3*h/2])
for i in range(N):
    Points.append([i*h,0])
    Points.append([i*h+h/2, r3*h/2])
Points.append([N*h, 0])
Points.append([N*h+h/2, r3 * h/2])
```

[5]:
```
Barres=[]  # Table de connectivité
for k in range(N):
    Barres.append([2 * k , 2 * k + 2]) # poutres horizontales supérieures
    Barres.append([2*k+1,2*k+3]) # poutres horizontales inférieures
    Barres.append([2 * k + 1, 2 * k + 2]) # poutres obliques /
    Barres.append([ 2 * k+2, 2 * k +3 ]) # poutres obliques \


Barres.append([2*N,2*N+2]) # la dernière poutre horizontale supérieure
```

[6]:
```
def cotes():
    plt.plot([0, -h/2,-h/2,0,0], [0, r3*h/2,-3,-3,0], color='green',␣
 ↪linestyle='solid')
    plt.plot([L, L+h/2,L+h/2,L,L], [0, r3*h/2,-3,-3,0], color='green',␣
 ↪linestyle='solid')
```

[7]:
```
cotes()
for p1, p2 in Barres:
    x1,y1 = Points[p1]
    x2,y2 = Points[p2]
    plt.plot([x1, x2], [y1, y2], color='red', linestyle='solid')
    plt.axis('equal')   # repère orthonormé
```



Charges externes

La charge appliquée en un noeud est un triplet: $[n^{(i)}, F_x^{(i)}, F_y^{(i)}]$

- n_i est le numéro du nœud i.
- $F_x^{(i)}$ est la composante horizontale de la force appliquée au noeud i.
- $F_y^{(i)}$ est la composante verticale de la force appliquée au noeud i.

[8]:
```
n = len(Points) * 2
# Liste des charges externes
Charges =[[2*i,0,-P] for i in range(1,N)]
#print(Charges)
l = [] # Liste des indices où les déplacements nuls sont appliquées
B = np.zeros(n, dtype=float)
for q, a, b in Charges:
    B[2 * q] = a
    B[2 * q + 1] = b
#print(B)
```

Conditions aux appuis :

Une condition d'appui est un triplet: $[n^{(i)}, \varepsilon_x, \varepsilon_y]$

- n_i est le numéro du noeud i.
- $\varepsilon_x = 0$ si le déplacement suivant x est bloqué sinon $\varepsilon_x = 1$.
- $\varepsilon_y = 0$ si le déplacement suivant y est bloqué sinon $\varepsilon_y = 1$.

```
[9]: Conditions=[[0,0,0],[1,0,0],[2*N+1,0,0],[2*N+2,0,0]]
l = [] # Liste des indices où les déplacements nuls sont appliquées
for q, a, b in Conditions:
    if a == 0:
        l.append(2 * q)
    if b == 0:
        l.append(2 * q + 1)
```

Matrice élémentaire et matrice globale

```
[10]: # Matrice globale
M = np.zeros((n, n), dtype=float)
for p1, p2 in Barres:
    x1,y1 = Points[p1]
    x2,y2 = Points[p2]
    #print("(", x1, ",", y1, ") (", x2, ",", y2, ")")
    ell = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
    c = (x2 - x1) / ell
    s = (y2 - y1) / ell
    CS = np.mat([c, s, -c, -s], dtype=float)
    CSt = np.transpose(CS)
    m = np.dot(CSt, CS)*A*E/ell
    # Assemblage de la matrice globale
    for r in range(2):
        for s in range(2):
            M[2 * p1+r, 2 * p1+s] += m[0+r, 0+s]
            M[2 * p1 + r, 2 * p2 + s] += m[0 + r, 2 + s]
            M[2 * p2 + r, 2 * p1 + s] += m[2 + r, 0 + s]
            M[2 * p2 + r, 2 * p2 + s] += m[2 + r, 2 + s]
```

Simplification du système linéaire: élimination des déplacements connus

```
[11]: # L'etat initial de la matrice M est conservé dans M0
M0 = M.copy()
# Suppression des lignes et colonnes correspondant aux indices des␣
 ↪contraintes nulles
l.sort()
l.reverse()
for i in l:
    M = np.delete(M, i, axis=0)
    M = np.delete(M, i, axis=1)
    B = np.delete(B, i)
```

Résolution du système linéaire

```
[12]: # Inversion du système linéaire

deplacement = np.linalg.solve(M, B)
```

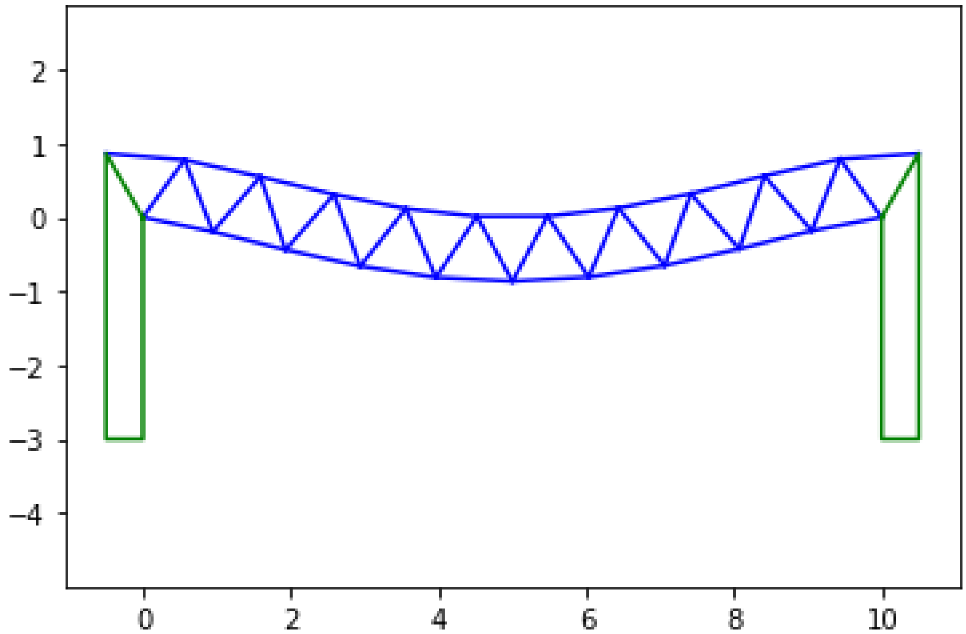
Représentation graphique de la solution

[13]:
```
# Rétablire les déplacements nuls
l.sort()
for i in l:
    deplacement =np.insert(deplacement,i,0)

# Calcul des réactions aux appuis
reactions = M0.dot(deplacement)
#print(100*"F","\n",reactions,"\n",100*"F","\n")
#print(100*"?","\n",deplacement,"\n",100*"?","\n")
# Pour des raisons graphiques on amplifie delta 10 fois
delta=deplacement*10
n=len(Points)
PointsPrim=[] # Position des noeuds après la déformation
for k in range(n):
    PointsPrim.append([delta[2*k]+Points[k][0],delta[2*k+1]+Points[k][1]])

for [p1,p2] in Barres:
    x1,y1=PointsPrim[p1]
    x2, y2 = PointsPrim[p2]
    plt.plot([x1, x2], [y1, y2], color='blue', linestyle='solid')

plt.axis('equal')   # repère orthonormé
cotes()
plt.show()
```



Annexe B: Exercice 2

[14]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
```

Données et constantes physiques

- Le pont est de longueur $L = 10$ m.
- Le nombre des sommets sur la partie inférieur du pont est $N + 1 = 11$ $(0 \leq i \leq N)$ où N est le nombre des sous-intervalles.
- Le pont (poutre) a une section carrée $A = r \times r$ et un moment d'inertie $I = \frac{r^4}{12}$ où $r = 0.02$ m.
- On applique en chaque noeud du tablier une charge $P = 0$ N (Question 1) et une force $P = 10$ N à l'extrémité libre en (Question 2).
- Et une force uniformément répartie de densité linéïque $p = 200$ N/m au (Question 1) et $p = 0$ en (Question 2)
- Le module de Young est $E = 200000$ MPa.

[15]:
```
L = 10.
r=0.02
A = r**2
I=r**4/12
E = 200 * 1E9
#p=-200 # Question 1
p = 0 # Question 2
N=10 # Nombre des poutres sur la partie inférieur du pont.
```

Géométrie du problème

Maillage: les éléments finis sont les poutres

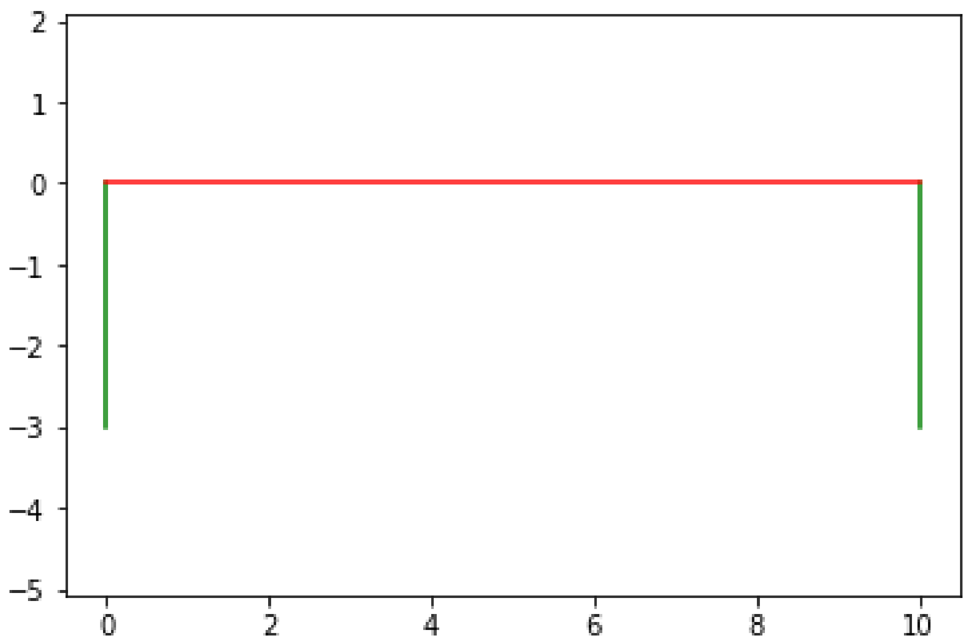
[16]:
```
Points=[]  # Liste des sommets
for i in range(N+1):
    ......
```

[17]:
```
Barres=[]  # Table de connectivité
for k in range(N):
    ....... # poutres horizontales supérieures
```

[18]:
```
def coteGauche():
    plt.plot([0,0], [0,-3], color='green', linestyle='solid')
```

[19]:
```
def coteDroite():
    plt.plot([L,L], [0,-3], color='green', linestyle='solid')
```

[20]:
```
coteGauche()
coteDroite()
plt.plot(...., ....., color='red', linestyle='solid')
plt.axis('equal')   # repère orthonormé
```



Charges externes ponctuelles

La charge appliquée en un noeud est un triplet: $[n^{(i)}, F^{(i)}, M^{(i)}]$ - n_i est le numéro du noeud i. - $F^{(i)}$ est la force verticale appliquée au noeud i. - $M^{(i)}$ est le moment appliqué au noeud i.

[21]:
```
n = len(Points) * 2
# Liste des charges externes
#ForcesPonctuelles =[] # Question 1
ForcesPonctuelles =[[N,-10,0]] # Question 2
l = [] # Liste des indices où les déplacements nuls sont appliquées
Fp = np.zeros(n, dtype=float)
for q, a, b in ForcesPonctuelles:
    .....
    .....
```

Charges réparties:

On rappelle (cf TD3) que le second membre élémentaire d'une poutre soumise uniquement à des charges linéiques de densité $p(x)$ est:

$$\begin{pmatrix} f_1^y \\ M_1 \\ f_2^y \\ M_2 \end{pmatrix} = \begin{pmatrix} \frac{p\ell}{2} \\ \frac{p\ell^2}{12} \\ \frac{p\ell}{2} \\ -\frac{p\ell^2}{12} \end{pmatrix}$$

Conditions aux appuis :

Une condition d'appui est un triplet: $[n^{(i)}, \varepsilon_y, \varepsilon_\theta]$

- n_i est le numéro du noeud i.

- $\varepsilon_y = 0$ si le déplacement suivant x est bloqué sinon $\varepsilon_y = 1$.
- $\varepsilon_\theta = 0$ si l'angle θ est bloqué sinon $\varepsilon_\theta = 1$.

[22]:
```
#Conditions=[[....],[....]]
Conditions=[[.....]]
l = [] # Liste des indices où les déplacements nuls sont appliquées
for q, a, b in Conditions:
    if a == 0:
        l.append(2 * q)
    if b == 0:
        l.append(2 * q + 1)
```

Système élémentaire et système globale

système élémentaire

$$\begin{pmatrix} f_1^y \\ M_1 \\ f_2^y \\ M_2 \end{pmatrix} = \frac{EI}{\ell^3} \begin{pmatrix} 12 & 6\ell & -12 & 6\ell \\ 6\ell & 4\ell^2 & -6\ell & 2\ell^2 \\ -12 & -6\ell & 12 & -6\ell \\ 6\ell & 2\ell^2 & -6\ell & 4\ell^2 \end{pmatrix} \begin{pmatrix} u_1^y \\ \theta_1 \\ u_2^y \\ \theta_2 \end{pmatrix}$$

[24]:
```
# Matrice globale
M = np.zeros((n, n), dtype=float)
Fr = np.zeros(n, dtype=float)
for p1, p2 in Barres:
    x1 = Points[p1]
    x2 = Points[p2]
    ell=math.fabs(x2-x1)
    m = np.array([[12,6*ell,-12,6*ell],[6*ell,4*ell**2,-6*ell,2*ell**2],.....
 ↪)*E*I/ell**3
    f=np.array([p*ell/2,p*ell**2/12,p*ell/2,-p*ell**2/12])
    # Assemblage de la matrice globale
    for r in range(2):
        Fr[.....]+=f[....]
        Fr[.....]+=f[....]
        for s in range(2):
            M[2 * p1+r, 2 * p1+s] += m[0+r, 0+s]
            M[2 * p1 + r, 2 * p2 + s] += m[0 + r, 2 + s]
            M[2 * p2 + r, 2 * p1 + s] += m[2 + r, 0 + s]
            M[2 * p2 + r, 2 * p2 + s] += m[2 + r, 2 + s]
```

[25]:
```
# Forces appliquées: second membre
F=Fp+Fr
```

Simplification du système linéaire: élimination des déplacements connus

[27]:
```
# L'etat initial de la matrice M est conservé dans M0
M0 = M.copy()
# Suppression des lignes et colonnes correspondant aux indices des␣
 ↪contraintes nulles
```

```
l.sort()
l.reverse()

for i in l:
    M = np.delete(.....)
    M = np.delete(.....)
    F = np.delete(.....)
```

Résolution du système linéaire

[28]:
```
# Inversion du système linéaire

deplacement = np.linalg.solve(.....)
```

Représentation graphique de la solution

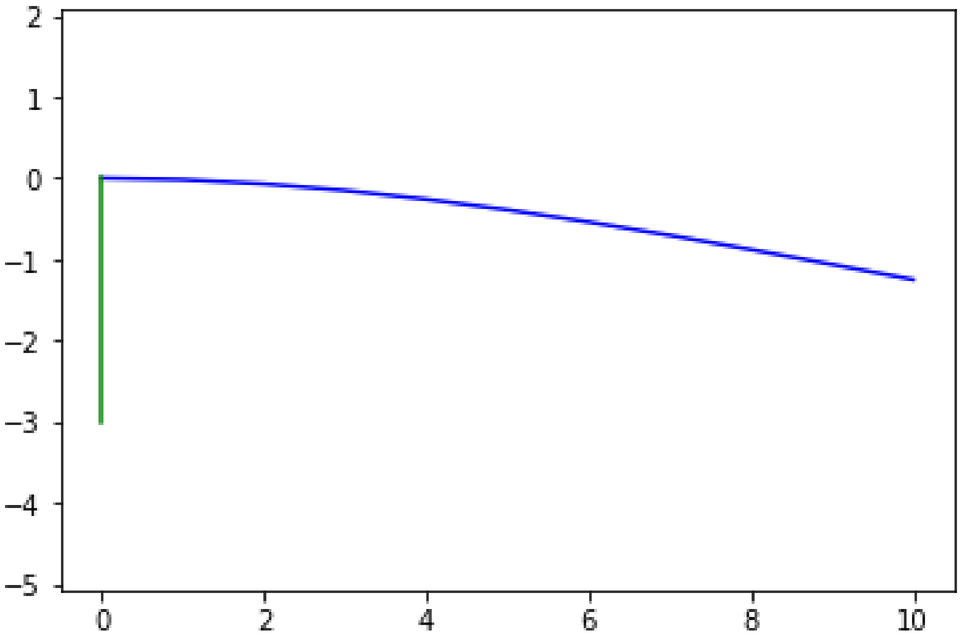
[30]:
```
# Rétablire les déplacements nuls
l.sort()
for i in l:
    deplacement =.....

# Calcul des réactions aux appuis
reactions =......

n=len(Points)
delta = np.zeros(......)
for i in range(.....):
    delta[i]=......


plt.plot(....., ....., color='blue', linestyle='solid')

plt.axis('equal')   # repère orthonormé
coteGauche()
#coteDroite()
plt.show()
```



[31]:
```
#Vérification
fleche=......
print(fleche)
#print(p*L**4/384/E/I) # Question 1
print(10*L**3/3/E/I)    # Question 2
```

```
1.2500000000002807
1.25
```