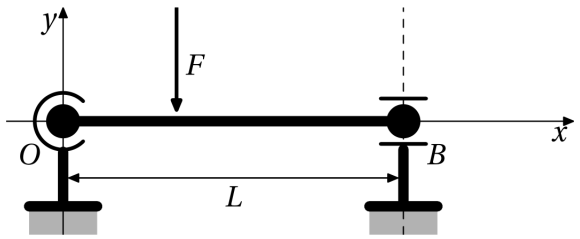
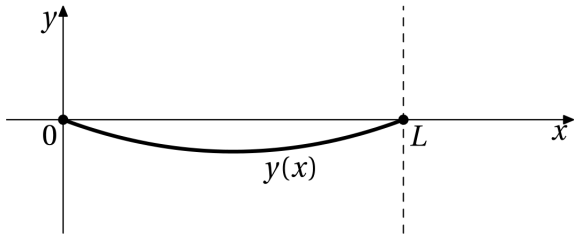
Méthode des différences finies

Un arbre de transmission est en liaison pivot avec le bâti par deux roulements à billes en contact radial situés en O et en B. Cet arbre subit une charge radiale F au tiers de sa longueur et on souhaite la déformation qui en résulte.



La déformation de l'arbre est représentée par une fonction y définie sur l'intervalle $[0, L]$.



Avec les techniques du cours de résistance des matériaux, on obtient les relations :

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} = u(x)$$
$$y(0) = y(L) = 0$$

avec les valeurs numériques $L = 150$ mm (longueur de l'arbre), $F = 6000$ N (intensité de la force appliquée), $D = 25$ mm (diamètre de l'arbre), $E = 2 \cdot 10^5$ MPa, $I = \pi D^4/64$ ainsi que la fonction :

$$u : x \mapsto \frac{2Fx}{3EI} \quad \text{si } x \leqslant \frac{L}{3}$$
$$x \mapsto \frac{F(L-x)}{3EI} \quad \text{si } x > \frac{L}{3}$$

On veut obtenir une approximation de $y(x)$.

Remarques.

- L'équation différentielle ne rentre pas tout à fait dans le cadre du cours de mathématiques (à cause de la forme du second membre).
- Toutefois, sur cet exemple, il est possible de procéder par recherche de primitives successives à partir de la fonction u. C'est faisable mais on a choisi ici de présenter une méthode numérique qui peut s'adapter à d'autres situations.
- Les conditions $y(0) = y(L) = 0$ ne correspondent pas à un problème de Cauchy et on ne peut donc pas appliquer la méthode d'Euler. □

I. Approximation d'une dérivée seconde

On considère une fonction $f : \mathbb{R} \to \mathbb{R}$ de classe C^2 et $a \in \mathbb{R}$

Question 1 : Donner le $DL_2(a)$ de f.

Question 2 : Déterminer $\lim_{h\to 0} \dfrac{f(a+h) - 2f(a) + f(a-h)}{h^2}$.

II. Résolution approchée

On subdivise l'intervalle $[0, L]$ en N intervalles de même longueur $h = L/N$. On note $x_0, \ldots, x_N$ les points régulièrement espacés qui délimitent ces intervalles. On note y_i l'approximation de $y(x_i)$ que l'on veut déterminer. Avec les questions précédentes, on convient d'approcher $y''(x_i)$ par :

$$y''(x_i) \simeq \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$$

On note également $u_i = u(x_i)$.

Question 3 : Écrire les approximations en x_i des relations $y''(x) = u(x)$ ainsi que $y(0) = 0$ et $y(L) = 0$ sous la forme :

$$A \begin{bmatrix} y_0 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} u_0 \\ \vdots \\ u_N \end{bmatrix}$$

Expliciter la matrice A.

III. Résolution avec Python

On définit :

```
from math import *
import matplotlib.pyplot as plt
import numpy as np

L = 150.0
F = 6000.0
D = 25.0
E = 2e5
I = pi*D**4/64
```

On pose également :

```
N = 10
h = float(L)/N
```

mais on prendra garde dans ce qui suit à écrire le code PYTHON en faisant référence à N et h (et pas aux valeurs numériques 10 et 15).

Question 4 : Définir la fonction u.

Question 5 : Définir les vecteurs $x = (x_0, \dots, x_N)$ et $u = (u_0, \dots, u_N)$ (**`np.array`** de taille $N+1$).

Question 6 : Construire la matrice A.

Question 7 : On donne dans l'annexe 1 l'aide en ligne de la fonction **`np.linalg.solve`**. Utiliser cette fonction pour obtenir le vecteur approché $y = (y_0, \dots, y_N)$.

Question 8 : Représenter graphiquement la déformation de l'arbre.

IV. Contrainte de rotulage des roulements

Le bon fonctionnement du roulement à billes ne peut se faire que si la pente de la déformation au point O est inférieure en valeur absolue à $2.9 \cdot 10^{-3}$.

Question 9 : À partir du vecteur y, donner une approximation de la pente de la déformation en O. Le roulement à billes fonctionne-t-il correctement ?

Annexe 1 : la fonction `np.linalg.solve`

```
Help on function solve in module numpy.linalg.linalg:

solve(a, b)
    Solve a linear matrix equation, or system of linear scalar
    equations.

    Computes the "exact" solution, `x`, of the well-determined,
    i.e., full rank, linear matrix equation `ax = b`.

    Parameters
    ----------
    a : (..., M, M) array-like
        Coefficient matrix.
    b : {(..., M,), (..., M, K)}, array-like
        Ordinate or "dependent variable" values.

    Returns
    -------
    x : {(..., M,), (..., M, K)} ndarray
        Solution to the system a x = b.
        Returned shape is identical to `b`.

    Raises
    ------
    LinAlgError
        If `a` is singular or not square.

    Notes
    -----
    Broadcasting rules apply, see the `numpy.linalg` documentation
    for details.

    The solutions are computed using LAPACK routine _gesv

    `a` must be square and of full-rank, i.e., all rows (or,
    equivalently, columns) must be linearly independent; if either is
    not true, use `lstsq` for the least-squares best "solution" of the
    system/equation.

    References
    ----------
    .. [1] G. Strang, *Linear Algebra and Its Applications*, 2nd Ed.,
           Orlando, FL, Academic Press, Inc., 1980, pg. 22.

    Examples
    --------
    Solve the system of equations ``3 * x0 + x1 = 9`` and
```

``x0 + 2 * x1 = 8``:

```
>>> a = np.array([[3,1], [1,2]])
>>> b = np.array([9,8])
>>> x = np.linalg.solve(a, b)
>>> x
array([ 2.,  3.])
```

Check that the solution is correct:

```
>>> np.allclose(np.dot(a, x), b)
True
```

Corrections

Q. 3 : La relation $\dfrac{\mathrm{d}^2 y}{\mathrm{d}x^2} = u(x)$ appliquée en x_i donne :

$$y''(x_i) = u(x_i)$$

ce qui compte-tenu des approximations considérées donne :

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} = u_i$$

Comme y_i est défini pour $0 \leqslant i \leqslant N$, la relation précédente n'est valable que pour $1 \leqslant i \leqslant N-1$. On a cependant $y_0 = 0$ et $y_N = 0$, or $u(0) = u(L) = 0$, on peut donc écrire le système :

$$\begin{cases} y_0 = u_0 \\ \forall i \in [\![1, N-1]\!],\ \dfrac{y_{i+1} - 2y_i + y_{i-1}}{h^2} = u_i \\ y_N = u_N \end{cases}$$

Ce système s'écrit matriciellement :

$$\underbrace{\frac{1}{h^2}\begin{bmatrix} 1 & & & & & & (0) \\ 1 & -2 & 1 & & & & \\ 0 & 1 & -2 & -1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & 1 & -2 & 1 \\ (0) & & & & & & 1 \end{bmatrix}}_{=A} \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \\ y_N \end{bmatrix} = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \\ u_N \end{bmatrix}$$

Q. 4 :

```
def u(x):
    if x<=L/3:
        return 2*F*x/(3*E*I)
    else:
        return F*(L-x)/(3*E*I)
```

Q. 5 :

```
x = np.array([i*h for i in range(N+1)])
u = np.array([u(x[i]) for i in range(N+1)])
```

Q. 6 : La matrice A est carrée de taille $N+1$. On la construit à partir de la matrice identité en apportant les modifications appropriées :

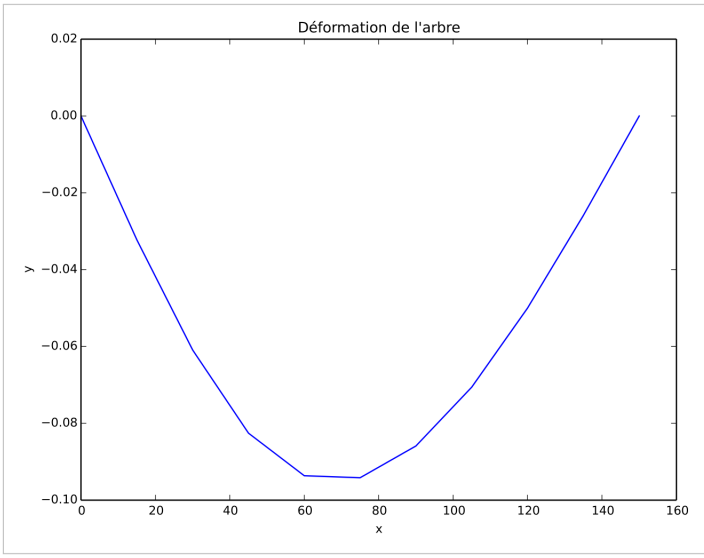
```
A = np.identity(N+1)
for i in range(1,N):
    A[i,i] = -2
    A[i,i-1] = 1
    A[i,i+1] = 1
A = A/h**2
```

Q. 7 :

```
y = np.linalg.solve(A,u)
```

Q. 8 :

```
plt.plot(x,y,'b-')
plt.xlabel('x')
plt.ylabel('y')
plt.title(u"Déformation de l'arbre")
```



Q. 9 : On utilise $\frac{y_1 - y_0}{h}$ comme approximation de $y'(0)$:

```
print (y[1]-y[0])/h
```

```
-0.0021473541429
```

Cette valeur est compatible avec la contrainte donnée pour le bon fonctionnement. Notons que l'on obtient une valeur plus précise de $y'(0)$ en prenant des valeurs de N plus grandes.