Informatique tronc commun
Devoir n° 4 – Partie rédigée

26 mai 2018

Durée : 60 minutes, documents interdits.

Vous écrirez les fonctions demandées dans le langage `Python` (version 3) et rendrez sur papier votre composition.

Vous rédigerez soigneusement la ou les fonctions `Python` que vous devrez écrire, sans oublier les indentations, chaînes de documentation et commentaires nécessaires.

On numérotera chaque ligne de chaque bloc de code `Python`.

Lorsque vous écrivez une fonction `Python`, on ne vous demande pas de vérifier que les arguments donnés sont corrects. Cependant, il sera apprécié d'indiquer les préconditions vérifiées par ces arguments dans la chaîne de documentation de la fonction.

Les fonctions `Python` permettant de faire un calcul sur les listes (ex : `max`, `sum`, *etc.*) ne sont pas autorisées, hormis la fonction `len`.

I. Questions de cours.

Q1 Écrire une fonction `newton(f, fp, x0, epsilon)` mettant en œuvre la méthode de Newton pour obtenir une valeur approchée d'un zéro de la fonction `f`, dont la dérivée est `fp`, à partir du point `x0` et avec le critère d'arrêt habituel `epsilon`.

Q2 La méthode de Newton converge-t-elle toujours ? Le cas échéant, que peut-on attendre comme vitesse de convergence ?

II. Modélisation dynamique d'une structure déformable

On étudie dans ce problème le comportement d'un objet déformable (poutre 1D) fixé à une extrémité et soumis à l'action d'une force à son autre extrémité.

On modélise cela par le montage en série de n systèmes ressort-ammortisseur (voir figure 1) présentant, pour tout $0 \leq i < n$, les mêmes masses m, raideurs k et coefficients d'amortissement c. On note $f(t)$ l'intensité de la force appliquée au dernier élément du système.

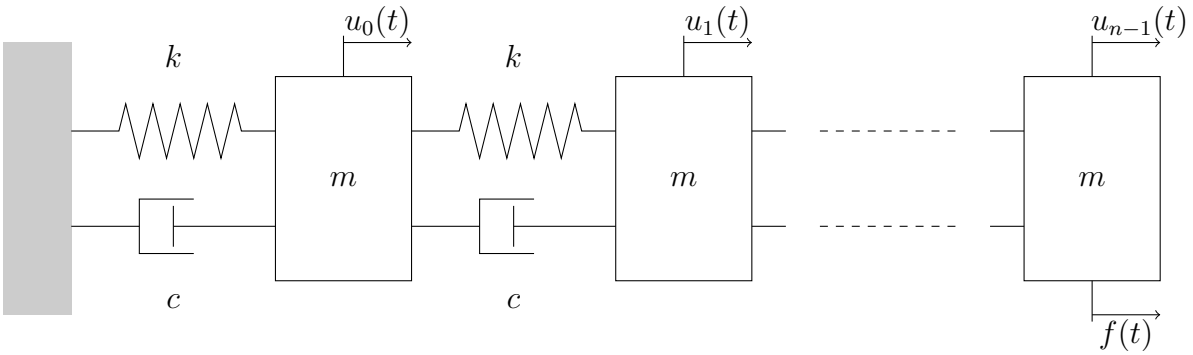

FIGURE 1 – Modélisation de la poutre par un ensemble de masses-ressorts-amortisseurs

Pour tout $0 \leq i < n$ et tout temps t, on note $u_i(t)$ le déplacement de l'élément n$^{\text{o}}$ i par rapport à sa position d'équilibre et l'on introduit le vecteur

$$X(t) = \begin{pmatrix} u_0(t) \\ \vdots \\ u_{n-1}(t) \end{pmatrix}.$$

On peut montrer que X vérifie l'équation différentielle

$$MX'' + CX' + KX = F(t),$$

où

$$M = \begin{pmatrix} m & 0 & \cdots & \cdots & 0 \\ 0 & m & 0 & \cdots & \vdots \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & m & 0 \\ 0 & 0 & \cdots & 0 & m \end{pmatrix}, \quad C = \begin{pmatrix} 2c & -c & 0 & \cdots & 0 \\ -c & 2c & -c & \ddots & \vdots \\ 0 & -c & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 2c & -c \\ 0 & \cdots & 0 & -c & 2c \end{pmatrix},$$

$$K = \begin{pmatrix} 2k & -k & 0 & \cdots & 0 \\ -k & 2k & -k & \ddots & \vdots \\ 0 & -k & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 2k & -k \\ 0 & \cdots & 0 & -k & 2k \end{pmatrix}, \quad F(t) = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ f(t) \end{pmatrix}.$$

Les matrices carrées écrites ici sont de dimension $n \times n$ et le vecteur $F(t)$ est de dimension n.

On utilise la méthode d'Euler pour résoudre de manière approchée cette équation différentielle. On fixe donc un pas de temps `dt` > 0 et l'on calcule de proche en proche les

$$X_q = X(q \times \texttt{dt}).$$

Le système est au repos au départ, on suppose donc que $X_0 = X_{-1} = 0$.

Avec

$$H = \frac{1}{\texttt{dt}^2}M + \frac{1}{\texttt{dt}}C + K$$

et, pour tout $q \geq 1$,

$$G_q = \frac{1}{\texttt{dt}^2}M(2X_{q-1} - X_{q-2}) + \frac{1}{\texttt{dt}}CX_{q-1} + F(q \times \texttt{dt}),$$

on montre que, pour tout $q \geq 1$,

$$H \times X_q = G_q.$$

On remarquera que, comme les matrices C et K, H est tridiagonale.

Pour mettre en œuvre la méthode d'Euler, on résout donc plusieurs systèmes de même membre de gauche H. Nous allons utiliser ici la méthode de Jacobi pour résoudre ce système.

Q3 Écrire une fonction `matrice_H(m,c,k,dt,n)` prenant en argument quatre flottants strictement positifs `m`, `c`, `k` et `dt` ainsi qu'un entier strictement positif `n` et renvoyant la matrice H de dimension `n` $\times$ `n` définie ci-dessus.

On donne la fonction suivante, permettant de calculer le vecteur G_q.

```
import numpy as np

def vecteur_G(m,c,dt,q,X1,X2,f):
    """Renvoie le vecteur Gq de dimension n
    Préconditions : m,c,dt flottants strictements positifs
                    q entier positif
                    X1,X2 vecteurs nx1 (X1 : X_q-1 et X2 = X_q-2)
                    f fonction réelle, appel en O(1)"""
    n,_ = X1.shape
    G = np.zeros((n,1))
    for i in range(n):
        G[i] = m*(2X1[i]-X2[i])/(dt**2)
    G[0] = G[0] +  (2c * X1[0] - c*X1[1])/dt
    G[-1] = G[-1] +  (2c * X1[-1] - c*X1[-2])/dt + f(q*dt)
    for i in range(1,n-1) :
        G[i] = G[i] + (2c * X1[i] - c*X1[i-1] - c*X1[i+1])/dt
    return G
```

Q4 Étudier la complexité temporelle d'un appel de la fonction `vecteur_G(m,c,dt,q,X1,X2,f)`, en fonction d'une grandeur que l'on explicitera.

La méthode du pivot de Gauss étant numériquement peu stable, on lui préfère souvent des méthodes itératives. On expose ici celle de Jacobi.

On résout le système $n \times n$ écrit matriciellement : $H \times Y = G$, à partir d'un vecteur initial Y^0. Pour tout $q \in \mathbb{N}$, on note

$$Y^q = \begin{pmatrix} y_0^q \\ \vdots \\ y_{n-1}^q \end{pmatrix}.$$

Si le vecteur Y^q est construit, la méthode de Jacobi consiste à définir le vecteur Y^{q+1} par, pour tout $0 \leq i < n$,

$$y_i^{q+1} = \frac{1}{h_{i,i}} \left(g_i - \sum_{j \neq i} h_{i,j} y_j^q \right).$$

Q5 Proposer une simplification de la somme écrite ci-dessus utilisant le fait que la matrice H utilisée ici est tridiagonale (on distinguera les cas $i = 0$ et $i = n - 1$).

Q6 Écrire une fonction `iteration_Jacobi(H,G,Yq)` prenant en argument une matrice H tridiagonale ainsi que deux vecteurs `G` et `Yq`, renvoie le vecteur Y^{q+1} décrit ci-dessus.

On s'interdira ici d'utiliser la multiplication matricielle fournie par la bibliothèque ***numpy****.*

On considère la norme euclidienne d'un vecteur :

$$\left\| \begin{pmatrix} x_0 \\ \vdots \\ x_{n-1} \end{pmatrix} \right\| = \sqrt{x_0^2 + \cdots + x_{n-1}^2} = \sqrt{\sum_{k=0}^{n-1} x_k^2}.$$

Q7 Écrire une fonction `carre_norme(X)` prenant en argument un vecteur `X` et renvoyant le carré de sa norme, *i.e.* $\|X\|^2$.

On utilise le critère d'arrêt suivant pour la méthode de Jacobi : on fixe une précision $\varepsilon > 0$, on s'arrête dès que $\|HY_q - G\| \leq \varepsilon \|G\|$ et l'on prend Y_q comme approximation de la solution de ce système.

Q8 Écrire une fonction `Jacobi(H,G,Y0,eps)` prenant en argument une matrice tridiagonale `H`, deux vecteurs `G` et `Y0` ainsi qu'un flottant strictement positif `eps` et renvoyant la valeur approchée du système $\texttt{H} \times X = \texttt{G}$ avec la condition initiale `Y0` et le critère d'arrêt donné par $\varepsilon = \texttt{eps}$.

La méthode de Jacobi converge si la matrice H est à diagonale strictement dominante, *i.e.* si pour tout $0 \leq i < n$,

$$|h_{i,i}| > \sum_{j \neq i} |h_{i,j}|.$$

Q9 Justifier que la méthode de Jacobi est ici convergente.

N'attaquez la dernière question que si vous avez traité toutes les autres questions correctement.

Q10 On effectue ce calcul sur p valeurs de temps distinctes. On rappelle que l'on résout donc p systèmes de la forme $H \times X_q = G_q$. Toujours en utilisant la méthode de Jacobi, peut-il être préférable de mettre en œuvre une autre stratégie de résolution, du point de vue de la complexité temporelle ?

On pourra supposer que tous les appels de la fonction `Jacobi` *ont la même complexité temporelle.*