Corrigé : Sujet zéro CCP IPT PSI

Introduction Je trouve que le sujet est intéressant et qu'il couvre plutôt bien le programme d'IPT. Cependant, il comporte plusieurs coquilles

1 Modélisation et problème à résoudre

Il y a des erreurs sur les équations (1), (2) et (3) fournies.

$$m_i \cdot \frac{\mathrm{d}^2 u_i(t)}{\mathrm{d}t^2} = -k_i \cdot (u_i(t) - u_{i-1}(t)) - k_{i+1} \cdot (u_i(t) - u_{i+1}(t)) - c_i \cdot \left(\frac{\mathrm{d}u_i(t)}{\mathrm{d}t} - \frac{\mathrm{d}u_{i-1}(t)}{\mathrm{d}t} \right) - c_{i+1} \cdot \left(\frac{\mathrm{d}u_i(t)}{\mathrm{d}t2} - \frac{\mathrm{d}u_{i+1}(t)}{\mathrm{d}t} \right) \quad (1)$$

$$m_1 \cdot \frac{\mathrm{d}^2 u_1(t)}{\mathrm{d}t^2} = -(k_1 + k_2) \cdot u_1(t) + k_2 \cdot u_2(t) - (c_1 + c_2) \cdot \frac{\mathrm{d}u_1(t)}{\mathrm{d}t} + c_2 \cdot \frac{\mathrm{d}u_2(t)}{\mathrm{d}t2} \quad (2)$$

$$m_n \cdot \frac{\mathrm{d}^2 u_n(t)}{\mathrm{d}t^2} = -k_n \cdot (u_n(t) - u_{n-1}(t)) - c_n \cdot \left(\frac{\mathrm{d}u_n(t)}{\mathrm{d}t} - \frac{\mathrm{d}u_{n-1}(t)}{\mathrm{d}t} \right) + f_n(t) \quad (3)$$

2 Structure générale du programme

Réponse 1. Compléter la fonction principale `main()` permettant de renseigner une géométrie en mode manuel.

```
[L, n, m, k, c] = geometrie(0)
```

On aurait pu demander de choisir le mode.

```
print('Modes de définition de la géométrie : 0 "manuel", 1 "fichier", 2 "base de données".')
[L, n, m, k, c] = geometrie( int( input('Mode de définition de la géométrie : ') ) )
```

La question 2 et la table `LIAISON` donnée page 6 ne sont pas en cohérence avec la fonction `geometrie()` de la page 4.

Il faut modifier l'algorithme de la fonction `geometrie()` comme ceci.

```
k_local = []
c_local = []
for i in range(n_local) :
    k_local.append(float(input('Raideur de la liaison ' + str(i) + ' : ')))
for i in range(n_local) :    # <- MODIFICATION
    c_local.append(float(input('Amortissement de la liaison ' + \
                               str(i) + ' : ')))
```

Réponse 2. Donner la valeur de `L`, `n` et des listes `m`, `k` et `c` à la sortie de la fonction `geometrie`.

L	*0.5*
n	*3*
m	*[0.1, 0.2, 0.1]*
k	*[20000., 1000., 20000.]*
c	*[10., 1000., 10.]*

3 Étude de la fonction `geometrie()`

3.1 Initialisation par lecture d'un fichier

Réponse 3. En utilisant la documentation fournie en annexe, proposer une écriture de la fonction `lire_fichier(nom_fic)` utilisée dans le mode 1 de la fonction `geometrie`.

```
def lire_fichier(nom_fic):
    '''renvoie les données de la géométrie renseignée dans le fichier nom_fic.'''
    file = open(nom_fic, 'r')
    L_local = float( file.readline().strip() )
    # la méthode .strip() permet d'ôter les caractères d'espace et de newline (\n) de la chaîne de caractère.
    # Si vous ne vous rappelez pas de cette méthode, on peut s'en sortir avec l'instruction suivante :
    # L_local = float( file.readline()[:-1] ) qui va ôter le \n de la fin de la chaîne.
    n_local = int( file.readline().strip() )
    m_local, k_local, c_local = [], [], []
    for ligne in range(n_local) :
        m_local.append( float( file.readline().strip() ) )
    for ligne in range(n_local) :
        k_local.append( float( file.readline().strip() ) )
    for ligne in range(n_local) :
        c_local.append( float( file.readline().strip() ) )
    file.close()
    return [L_local, n_local, m_local, k_local, c_local]
```

Pour resserrer le code, on peut écrire les boucles sous la forme suivante.

```
d_local = [[], [], []]
for i in range( len(d_local) ) :
    for ligne in range(n_local) :
        d[i].append( float( file.readline().strip() ) )
return [L_local, n_local, d_local[0], d_local[1], d_local[2]]
```

3.2 Initialisation à partir de la base de données

Il y a un manque de cohérence dans le nom des attributs des tables entre le schéma relationnel donné en haut de la page 6 et les tableaux indiqués au centre de la même page.

Je propose de conserver les noms des attributs des tableaux qui portent moins à confusion.

Réponse 4. Définir la chaîne de caractères `requete1` (voir fonction `geometrie`) permettant de renvoyer l'ensemble des noms de calcul avec leur date et leur identifiant.

```
requete1 = 'SELECT "nom", "date", "id" AS "identifiant" FROM "CALCUL" ;'
```

Voici le tableau obtenu avec le logiciel `Base` *de la suite* `LibreOffice` *(avec la numérotation des clés primaires qui démarre automatiquement à 0, et non à 102 comme dans le sujet).*

	nom	date	identifiant
▷	calcul1	18.02.15	0

Réponse 5. Compléter la « Zone à compléter » de la fonction `geometrie` sur votre feuille, permettant de définir les différentes variables de la géométrie `L_local`, `n_local`, `m_local`, `k_local` et `c_local`.

```
requete2 = 'SELECT "L", "n" FROM "CALCUL" WHERE "id" = ' + str(id_calcul) + ' ;'

resultat2 = interroge_bdd(requete2)
L_local, n_local = resultat2[0], resultat2[1]

requete3 = '
SELECT "POUTRE"."num", "masse", "raideur", "amortissement"
FROM "POUTRE", "LIAISON"
WHERE "POUTRE"."id_calcul" = "LIAISON"."id_calcul" AND "POUTRE"."id_calcul" = ' + str(id_calcul) + '
AND "POUTRE"."num" = "LIAISON"."num" ;'

resultat3 = interroge_bdd(requete3)
m_local, k_local, c_local = [], [], []
for ligne in range( len(resultat3) ) :
    m_local.append( resultat3[ligne][1] )
            k_local.append( resultat3[ligne][2] )
            c_local.append( resultat3[ligne][3] )
```

Voici les tableaux obtenus avec le logiciel `Base`.

	L	n
▷	0.5	3

	num	masse	raideur	amortissement
▷	1	0.2	20000	10
	2	0.1	1000	1000
	3	0.2	20000	10

Commentaires sur ces deux questions *Il ne faut pas oublier de bien gérer les chaînes de caractères pour l'écriture des requêtes avec* `Python` *en alternant les simples guillemets et les doubles. Enfin, la présence d'un attribut ayant le même nom dans deux relations différentes ;* `id_calcul` *pour* `POUTRE` *et* `LIAISON` *oblige à utiliser la syntaxe* `"NOM_TABLE"."nom_attribut"`.

4 Simulation numérique

4.1 Création des matrices

$$M \cdot \ddot{X}(t) + C \cdot \dot{X}(t) + K \cdot X(t) = F(t) \tag{4}$$

Réponse 6. En reprenant les équations (1), (2) et (3), déterminer les expressions des matrices M, C et K. Bien préciser les indices des termes non nuls.

Je commence par écrire les équations données sous la forme de (4).

$$m_i \cdot \frac{\mathrm{d}^2 u_i(t)}{\mathrm{d}t^2} + (c_i + c_{i+1}) \cdot \frac{\mathrm{d}u_i(t)}{\mathrm{d}t} - c_i \cdot \frac{\mathrm{d}u_{i-1}(t)}{\mathrm{d}t} - c_{i+1} \cdot \frac{\mathrm{d}u_{i+1}(t)}{\mathrm{d}t} + (k_i + k_{i+1}) \cdot u_i(t) - k_i \cdot u_{i-1}(t) - k_{i+1} \cdot u_{i+1}(t) = 0 \tag{5}$$

$$m_1 \cdot \frac{\mathrm{d}^2 u_1(t)}{\mathrm{d}t^2} + (c_1 + c_2) \cdot \frac{\mathrm{d}u_1(t)}{\mathrm{d}t} - c_2 \cdot \frac{\mathrm{d}u_2(t)}{\mathrm{d}t} + (k_1 + k_2) \cdot u_1(t) - k_2 \cdot u_2(t) = 0 \tag{6}$$

$$m_n \cdot \frac{\mathrm{d}^2 u_n(t)}{\mathrm{d}t^2} + c_n \cdot \frac{\mathrm{d}u_n(t)}{\mathrm{d}t} - c_n \cdot \frac{\mathrm{d}u_{n-1}(t)}{\mathrm{d}t} + k_n \cdot u_n(t) - k_n \cdot u_{n-1}(t) = f_n(t) \tag{7}$$

On pose les vecteurs X sous la forme suivante.

$$X = \begin{bmatrix} u_1(t) \\ \vdots \\ u_n(t) \end{bmatrix}, \ \dot{X} = \begin{bmatrix} \frac{\mathrm{d}u_1(t)}{\mathrm{d}t} \\ \vdots \\ \frac{\mathrm{d}u_n(t)}{\mathrm{d}t} \end{bmatrix}, \ \ddot{X} = \begin{bmatrix} \frac{\mathrm{d}^2 u_1(t)}{\mathrm{d}t^2} \\ \vdots \\ \frac{\mathrm{d}^2 u_n(t)}{\mathrm{d}t^2} \end{bmatrix}$$

Alors les matrices recherchées s'écrivent sous les formes suivantes.

$$M = \begin{bmatrix} m_1 & 0 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & m_i & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & m_n \end{bmatrix}$$

$$C = \begin{bmatrix} c_1 + c_2 & -c_2 & 0 & 0 & 0 & 0 & 0 \\ -c_2 & \ddots & \ddots & 0 & 0 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 & 0 & 0 \\ 0 & 0 & -c_i & c_i + c_{i+1} & -c_{i+1} & 0 & 0 \\ 0 & 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & \ddots & \ddots & -c_n \\ 0 & 0 & 0 & 0 & 0 & -c_n & c_n \end{bmatrix}$$

$$K = \begin{bmatrix} k_1 + k_2 & -k_2 & 0 & 0 & 0 & 0 & 0 \\ -k_2 & \ddots & \ddots & 0 & 0 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 & 0 & 0 \\ 0 & 0 & -k_i & k_i + k_{i+1} & -k_{i+1} & 0 & 0 \\ 0 & 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & \ddots & \ddots & -k_n \\ 0 & 0 & 0 & 0 & 0 & -k_n & k_n \end{bmatrix}$$

Réponse 7. Écrire une fonction `[M, K, C] = creation_operateur(n, m, k, c)` qui réalise la construction de ces différentes matrices.

```
def creation_operateur(n, m, k, c) :
    '''renvoie les matrices carrées M, K et C sous forme d'array de taille n.'''
    M_local, K_local, C_local = zeros((n, n), float) * 3

    # Création de la matrice M
    for i in range(n) : M_local[i, i] = m[i]

    # Création de la matrice C
    C_local[0, 0], C_local[0, 1] = c[0] + c[1], -c[1]
    for j in range(1, n - 1) :
        C_local[i, i - 1] = - c[i]
        C_local[i, i] = c[i] + c[i + 1]
        C_local[i, i + 1] = - c[i]
    C_local[n - 1, n - 2], C_local[n - 1, n - 1] = -c[n - 1], c[n - 1]

    # Création de la matrice K
    K_local[0, 0], K_local[0, 1] = k[0] + k[1], -k[1]
    for p in range(1, n - 1) :
        K_local[i, i - 1] = - k[i]
        K_local[i, i] = c[i] + k[i + 1]
        K_local[i, i + 1] = - k[i]
    K_local[n - 1, n - 2], K_local[n - 1, n - 1] = -k[n - 1], k[n - 1]

    return [M_local, K_local, C_local]
```

4.2 Résolution

Réponse 8. En appliquant le schéma d'Euler explicite deux fois, exprimer V_q, en fonction de dt, X_q et X_{q-1}, puis A_q en fonction de dt. X_q, X_{q-1}, et X_{q-2}. Écrire le système matriciel à l'instant t_q et le mettre sous la forme $H \cdot X_q = G_q$ où vous exprimerez H en fonction de M, K, C et le pas de temps dt et G_q en fonction de M, K, C, X_{q-1}, X_{q-2}, dt et F_q le vecteur force calculé à l'instant t_q.

Le schéma explicite d'Euler permet d'approximer la dérivée d'une fonction $f'(t_q)$ par le quotient $\frac{f(t_q+dt)-f(t_q)}{dt}$.
On a donc :

$$V_q = \frac{X_q - X_{q-1}}{dt}$$

Et $A_q = \frac{V_q - V_{q-1}}{dt} = \frac{1}{dt} \cdot \left(\frac{X_q - X_{q-1}}{dt} - \frac{X_{q-1} - X_{q-2}}{dt} \right)$, donc :

$$A_q = \frac{1}{dt^2} \cdot \left(X_q - 2 \cdot X_{q-1} + X_{q-2} \right)$$

Le système matriciel s'écrit d'après (4) :

$$M \cdot A_q + C \cdot V_q + K \cdot X_q = F_q$$

Avec $A_q = \frac{1}{dt^2} \cdot X_q - \frac{2}{dt^2} \cdot X_{q-1} + \frac{1}{dt^2} \cdot X_{q-2}$ et $V_q = \frac{1}{dt} \cdot X_q - \frac{1}{dt} \cdot X_{q-1}$.

$$M \cdot \left(\frac{1}{dt^2} \cdot X_q - \frac{2}{dt^2} \cdot X_{q-1} + \frac{1}{dt^2} \cdot X_{q-2} \right) + C \cdot \left(\frac{1}{dt} \cdot X_q - \frac{1}{dt} \cdot X_{q-1} \right) + K \cdot X_q = F_q$$

$$M \cdot \frac{1}{dt^2} \cdot X_q + C \cdot \frac{1}{dt} \cdot X_q + K \cdot X_q = F_q + M \cdot \left(\frac{2}{dt^2} \cdot X_{q-1} - \frac{1}{dt^2} \cdot X_{q-2} \right) + C \cdot \left(\frac{1}{dt} \cdot X_{q-1} \right)$$

D'où en posant $H = \frac{1}{dt^2} \cdot M + \frac{1}{dt} \cdot C + K$, et $G_q = F_q + M \cdot \left(\frac{2}{dt^2} \cdot X_{q-1} - \frac{1}{dt^2} \cdot X_{q-2} \right) + C \cdot \left(\frac{1}{dt} \cdot X_{q-1} \right)$, le système matriciel peut bien s'écrire sous la forme $H \cdot X_q = G_q$.

Réponse 9. Écrire une fonction `X = calcul(n, M, K, C, npts, dt, fmax, omega)` qui permette de résoudre le problème en utilisant la fonction `resoud`, qui ne sera pas détaillée ici. On choisira $ntps = 10000$, soit un pas de temps dt égal à $T/ntps$.

Passer en argument $ntps$ et dt ne me semble pas très logique puisque dt est une fonction de $ntps$ et de T. Je pense donc qu'il faut plutôt écrire la fonction `X = calcul(n, M, K, C, npts, T, fmax, omega)`.

```
def calcul(n, M, K, C, npts, T, fmax, omega) :
    '''renvoie la matrice X solution de l'équation (4).'''
    # calcul de la matrice H
    H_local = zeros( (n,n), float )
    for ligne in range(n) :
        for col in range(n) :
            H[ligne, col] = (1 / pow(dt, 2)) * M[ligne, col] + (1 / dt) * C[ligne, col] + K[ligne, col]
```

```
# calcul des temps
dt = T / ntps
t = arange(0, T + dt, dt)

# définition des fonction locales F et G
def F(q) :
    return array([0] * (n - 1) + [fmax * sin(omega * t[q])])

def G(q) :
    if q < 2 :
        return F(q)
    else :
        return F(q) + dot(M, (2 / pow(dt, 2)) * X[q - 1] - (1 / pow(dt, 2)) * X[q - 2]) + \
                      dot(C, (1 / dt) * X[q - 1])

# initialisation de X
X = zeros ( (ntps, n), float )

# définition de X à l'instant initial
X[0] = array([0] * n)

# boucle de calcul des valeurs de X
for q in range(len(t)) :
    X[q] = resoud(H, G(q))

return X
```

Réponse 10. Écrire une fonction `Xq = resoud(H, Gq)` basée sur la méthode du pivot de Gauss adaptée à la forme de la matrice H.

La méthode du pivot de Gauss permet de résoudre $A \cdot X = Y$, elle nécessite de programmer quatre fonctions différentes :

1. `pivot_partiel(A, colonne)`, *qui doit renvoyer l'indice de la ligne de A où $A[ligne, colonne]$ est maximal. Ce sera la valeur du pivot partiel à prendre en compte pour les étapes suivantes.*
2. `echange_lignes(A, ligne1, ligne2)`, *qui doit modifier A en inversant $A[ligne1]$ et $A[ligne2]$. On placera alors la ligne du pivot partiel en amont des autres lignes à traiter.*
3. `transvection(M, i, j, coef)`, *qui effectue la transvection $Li \leftarrow Li + coef * Lj$. Cette fonction doit s'appliquer sur A, et sur Y. Elle modifie directement les matrices, ce qui nécessite d'en faire des* copies profondes *(*`numpy.copy(M)`*) avant de trianguliariser A. De plus, comme elle modifie directement la matrice passée en argument sans rien renvoyer, on dit qu'elle agit par* effet de bord. *Son appel se fera donc par |transvection(M, i, j, coef)|, et non par |resultat = transvection(M, i, j, coef)|.*
4. `remontee(T, Y)`, *qui renvoie la solution de $T \cdot X = Y$ pour une matrice T triangulaire supérieure.*

Dans notre cas, les choses se simplifient pas mal vu la forme particulière de la matrice H (avec les notation du sujet page 7).

$$H \cdot X_q = \begin{bmatrix} H_{1,1} & H_{1,2} & 0 & 0 & 0 & 0 & 0 & 0 \\ H_{1,2} & H_{2,2} & H_{2,3} & 0 & 0 & 0 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 & 0 & 0 & 0 \\ 0 & 0 & H_{i-1,i-2} & H_{i-1,i-1} & H_{i-1,i} & 0 & 0 & 0 \\ 0 & 0 & 0 & H_{i,i-1} & H_{i,i} & H_{i,i+1} & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & H_{n-1,n-2} & H_{n-1,n-1} & H_{n-1,n} \\ 0 & 0 & 0 & 0 & 0 & 0 & H_{n,n-1} & H_{n,n} \end{bmatrix} \cdot \begin{bmatrix} X_{q,1} \\ X_{q,2} \\ \vdots \\ X_{q,i-1} \\ X_{q,i} \\ \vdots \\ X_{q,n-1} \\ X_{q,n} \end{bmatrix} = \begin{bmatrix} G_{q,1} \\ G_{q,2} \\ \vdots \\ G_{q,i-1} \\ G_{q,i} \\ \vdots \\ G_{q,n-1} \\ G_{q,n} \end{bmatrix} = G_q$$

Remarque 1. *Attention, les **indexations** de listes ou de tableaux vont de 0 à $n-1$ en Python, et non de 1 à n comme ici.*

Á la deuxième ligne, le pivot est $H_{1,2}$, la transvection est $L_2 \leftarrow L_2 - \frac{H_{2,1}}{H_{1,1}} * L_1$ en remarquant qu'ici $H_{2,1} = H_{1,2}$ car H est une matrice symétrique.

Ce qui donne pour la matrice H modifiée et notée H' :

$$H'_{2,1} = 0\ ,\ H'_{2,2} = H_{2,2} - \frac{H_{1,2}}{H_{1,1}} * H_{1,2}\ ,\ et\ H'_{2,3} = H_{2,3}$$

et pour la matrice G_q modifiée et notée G'_q :

$$G'_{q,2} = G_{q,2} - \frac{H_{1,2}}{H_{1,1}} * G_{q,2}$$

Á la ligne suivante, le pivot partiel est encore la première valeur non nulle de la ligne suivante de H, et la transvection ne modifie que deux valeurs de H' selon les relations de récurrence suivantes :

$$H'_{i,i-1} = 0\ ,\ H'_{i,i} = H_{i,i} - \frac{H'_{i-1,i}}{H'_{i-1,i-1}} * H'_{i-1,i}\ ,\ et\ H'_{i,i+1} = H'_{i,i+1}$$

.

Il n'y aura donc pas besoin de chercher le pivot partiel ni d'échanger les lignes comme pour la méthode du pivot de Gauss.

De même pour G'_q :

$$G'_{q,i} = G_{q,i} - \frac{H'_{i-1,i}}{H'_{i-1,i-1}} * G'_{q,i-1}$$

Une fois cette descente terminée, on se retrouve avec une matrice H' bi-diagonale et le système s'écrit alors :

$$H' \cdot X_q = \begin{bmatrix} H_{1,1} & H_{1,2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & H'_{2,2} & H'_{2,3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & H'_{i-1,i-1} & H'_{i-1,i} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & H'_{i,i} & H'_{i,i+1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & H'_{n-1,n-1} & H'_{n-1,n} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & H'_{n,n} \end{bmatrix} \cdot \begin{bmatrix} X_{q,1} \\ X_{q,2} \\ \vdots \\ X_{q,i-1} \\ X_{q,i} \\ \vdots \\ X_{q,n-1} \\ X_{q,n} \end{bmatrix} = \begin{bmatrix} G_{q,1} \\ G'_{q,2} \\ \vdots \\ G'_{q,i-1} \\ G'_{q,i} \\ \vdots \\ G'_{q,n-1} \\ G'_{q,n} \end{bmatrix} = G'_q$$

Pour la remontée, on commence par la dernière ligne où $X_{q,n} = G'_{q,n} / H'_{n,n}$.
Puis, $X_{q,n-1} = (G'_{q,n-1} - H'_{n-1,n} * X_{q,n}) / H'_{n,n}$, et par récurrence :

$$X_{q,i} = (G'_{q,i} - H'_{i,i+1} * X_{q,i+1}) / H'_{i,i}$$

Cette remontée est là encore plus simple que pour la méthode du pivot de Gauss puisqu'elle ne nécessite qu'une multiplication à chaque étape plutôt que $(n-1-i)$ pour une matrice triangulaire supérieure complète.

Il ne reste plus qu'à traduire cet algorithme en Python.

```
def resoud(H, Gq) :
    '''renvoie le vecteur Xq solution de H.Xq=Gq. '''
    # copies des matrices passées en argument pour ne pas les modifier
    H_local = copy(H)    # copy est une fonction de numpy
    Gq_local = copy(Gq)

    # transvections
    for ligne in range(1, len(H)) :
        # calcul du coefficient de transvection
        coef = - H_local[ligne, ligne - 1] / H_local[ligne - 1, ligne - 1]
        # modification des deux valeurs de H qui vont changer de valeurs
        H_local[ligne, ligne - 1] = 0
        H_local[ligne, ligne] += coef * H_local[ligne - 1, ligne]
        # modification de la valeur de Gq
        Gq_local[ligne] += coef * Gq_local[ligne - 1]

    # initialisation du résultat
    Xq = zeros( (n, 1), float )
    # calcul du dernier terme
    Xq[n - 1] = Gq_local[n - 1] / H_local[n - 1, n - 1]

    # remontée
    for lig in range(n - 2, -1, -1) :
        # lig décrémente de (n - 2) à 0
        Xq[lig] = ( Gq_local[lig] - H_local[lig, lig + 1] * Xq[lig + 1] ) / \
                  H_local[lig, lig]

    return Xq
```

Remarque 2.
– *Les objets de type* `array` *du module* `numpy` *autorisent une indexation assez souple, puisqu'ici* `Xq[i]` *sera équivalent à* `Xq[i, 0]` *car X_q est un vecteur $n \times 1$.*
– *Pour rappel, cette méthode du pivot de Gauss est disponible dans le module* `numpy`.
Sa syntaxe est `Xq = numpy.linalg.solve(H, Gq)`.

Réponse 11. Calculer la complexité en nombre d'opérations de la fonction `resoud` en fonction de n. Comparer ce résultat à la complexité basée sur une résolution du système matriciel à l'aide d'un pivot de Gauss classique.

On parcourt l'algorithme implanté et on récapitule ici le nombre d'opérations pour chaque étape.

1. Les copies *: naïvement, on peut penser qu'elles demandent* $n^2 \times 2$ *affectations. Mais on peut aussi considérer qu'elles ne demandent qu'au plus trois affectations par lignes, d'où une complexité de* $(3 \times n) \times 2$.
2. Les transvections *:* $\times(n-1)$
   - *coef : 1 affectation + 1 opération,*
   - *H : 2 affectations + 2 opérations,*
   - *Gq : 1 affectation + 2 opérations.*
3. L'initialisation *: n affectations.*
4. Le dernier terme *: 1 affectation + 1 opération.*
5. La remontée *:* $\times(n-1)$
   - *Xq : 1 affectation + 3 opérations.*

Au total, on obtient une complexité $C(n) = 2 \cdot (3 * n) + (n-1) * 9 + n + 2 + (n-1) * 4 = 2 \cdot n^2 + 14 \cdot n - 9$.
C'est à dire une complexité en $O(n)$.
Cette complexité en $O(n)$ *est remarquable comparée à la complexité d'un pivot de Gauss classique qui est en* $O(n^3)$ *: la transvection complète coûtant* $O(n)$ *opérations par terme calculé, à répéter sur les* $O(n)$ *éléments d'une ligne, et cette opération de transvection doit se répéter pour les* $O(n)$ *lignes du système.*

4.3 Évaluation du coût de stockage

Réponse 12. Déterminer la quantitié de mémoire RAM en Go nécessaire pour stocker le résultat d'un calcul sur la durée T. On ne tiendra compte que de la matrice X.
$\dim(X) = n \times ntps$ *avec* $ntps = T/dt = 150000$.
De plus, un nombre réel écrit en virgule flottante en double précision nécessite 64 bits, soit 8 octets de mémoire.

Remarque 3. *Les nombres réels écrits en virgule flottante sont définis par la norme internationale IEEE 754 de 1985 qui spécifie deux formats de nombres en virgule flottante en base 2 selon le tableau suivant.*

Précision	***Encodage***	***Signe***	***Exposant***	***Mantisse***	***Valeur d'un nombre***	***Chiffres significatifs***
Simple	*32 bits*	*1 bit*	*8 bits*	*23 bits*	$(-1)^S \times M \times 2^{(E-127)}$	*environ 7*
Double	*64 bits*	*1 bit*	*11 bits*	*52 bits*	$(-1)^S \times M \times 2^{(E-1023)}$	*environ 16*

La quantité de mémoire RAM nécessaire pour stocker X est donc de 240×10^6 *octets, soit* 0.24 *Go, ce qui représente près de 1 % de la RAM du calculateur, ce qui n'est pas négligeable.*

Remarque 4. *Un* kilo-octets *vaut* 1 Ko $= 10^3$ octets, *et un* kibioctet *vaut* 1 Kio $= 2^{10} = 1024$ octets.

5 Traitement des résultats

5.1 Affichage du résultat

Réponse 13. Écrire une fonction `affiche_deplacement_pdt(q, X, L, n, ampl)` qui affiche le déplacement X_q au pas de temps t_q. L'affichage devra représenter la poutre comme sur la figure 3, la position des masses étant matérialisée par des diamants reliés entre eux (option "D" en Python) et donnée par la position initiale plus le déplacement amplifié d'un facteur `ampl`.

```
import matplotlib.pyplot as plt

def affiche_deplacement_pdt(q, X, L, n, ampl) :
    '''affiche le déplacement X[q] au pas de temps tq. La position des masses
est matérialisée par des diamants reliés entre eux et donnée par la position
initiale plus le déplacement amplifié d'un facteur ampl. '''
    x = [(i + 1) * (L / n) + ampl * X[q, i] for i in range(n)]
    y = [0] * n
    plt.plot(x, y, marker = 'D')
    plt.title('Déplacements amplifiés des noeuds de la poutre.')
    plt.show()
```

5.2 Détermination du pas de temps et du lieu du déplacement maximal

Réponse 14. Écrire une fonction `lieu_temps_depl_max(X)` qui renvoie le numéro du nœud, le pas de temps et la valeur du déplacement maximal en valeur absolue.

```
def lieu_temps_depl_max(X) :
    '''renvoie le numéro du noeud, le pas de temps et la valeur du déplacement
maximal en valeur absolue de X.'''
    n_max, q_max, u_max = 0, 0, 0
    t = shape(X)[0]      # le nombre de vecteurs Xq mémorisés dans X
    n = shape(X)[1]      # le nombre d'éléments dans Xq
    for q in range(t) :
        for i in range(n) :
            if abs(X[q, i]) > u_max :
                n_max = i
                q_max = q
                u_max = abs(X[q, i])
    return [n_max, q_max, u_max]
```

5.3 Détermination de l'énergie dissipée

Réponse 15. Écrire une fonction `Ediss = calcul_energie(X, c, T)` qui calcule l'énergie dissipée totale, puis trace son évolution en fonction du temps et qui renvoie `Ediss`. Vous présenterez quelle méthode d'intégration vous utiliserez.

La vitesse de déplacement d'un nœud est notée V_q, soit, avec le schéma d'Euler choisi à la question 8, $V_q = \frac{X_q - X_{q-1}}{dt}$. $\dot{u}_i(t) - \dot{u}_{i-1}(t)$ *sera donc approché par* `(X[q] - X[q - 1]) / dt - (X[q - 1] - X[q - 2]) / dt`, *soit :*

`(X[q] - 2 * X[q - 1] + X[q - 2]) / dt`

La suite de l'algorithme ne pose pas de problèmes particuliers.

```
def calcul_energie(X, c, T) :
    '''calcule l'énergie dissipée totale Ediss, puis trace son évolution en fonction
du temps et renvoie la valeur de Ediss.'''
    t = shape(X)[0]      # le nombre de vecteurs Xq mémorisés dans X
    n = shape(X)[1]      # le nombre d'éléments dans Xq
    dt = T / t

    # Défintion d'une fonction locale Pdiss
    def Pdiss(q) :
        '''renvoie la puissance dissipée au temps tq.'''
        res = 0
        if q == 0 :
            return res
        if q == 1 :
            for i in range(1, n) :
                res += c[i] * pow( X[q, i] / dt, 2)
            return res
        else :
            for i in range(1, n) :
                res += c[i] * \
                       pow( (X[q, i] - 2 * X[q - 1, i] + X[q - 2, i]) / dt, 2)
            return res

    # Calcul de Ediss sous la forme d'une liste donnant Ediss[q]
    Ediss = []
    res = 0
    # Utilisation de la méthode des trapèzes pour l'intégration
    for q in range(t) :
        res += (Pdiss(q) + Pdiss(q - 1)) * (dt / 2)
        Ediss.append(res)

    # Tracé de l'évolution de Ediss en fonction du temps
    tps = arange(0, T + dt, dt)
    plt.plot(t, Ediss)
    plt.title('Evolution de l\'énergie dissipée au cours du temps.')
    plt.show()

    # Renvoi de l'énergie dissipée totale
    return Ediss[-1]
```