Python - DS 2

Euler semi-implicite

On étudie ici une variante de la méthode d'Euler explicite : la méthode d'Euler semi-implicite (qui est aussi d'ordre 1). Cette méthode est applicable à un système d'équations différentielles de la forme (x et v sont les fonctions solutions recherchées) :

$$\frac{dx}{dt} = f(t, v); \frac{dv}{dt} = g(t, x)$$

La méthode d'Euler semi-implicite est alors donnée par : $\begin{cases} v_{n+1} = v_n + g(t_n, x_n) \times \Delta t \\ x_{n+1} = x_n + f(t_n, v_{n+1}) \times \Delta t \end{cases}$

On s'intéresse ici à un ressort de masse m et de raideur k fixées. On a de plus $v = \dot{x}$ et $\frac{dv}{dt} = -\omega^2 x$ avec $\omega^2 = \frac{k}{m}$

1. Quelle est la différence avec la méthode d'Euler explicite ? Expliquer le nom "semi-implicite". Donner l'expression de v_{n+1} et x_{n+1} en fonction de $x_n, x_{n+1}, v_n, v_{n+1}, \omega, \Delta t$ (facile [1]).
2. On suppose ω fixé (variable globale appelée `omega`). Écrire une fonction `ressort(x_0, v_0, temps)` qui renvoie un tableau (ou liste de liste, ou ce que vous voulez), à n lignes et 2 colonnes, qui correspond aux valeurs de (x_k, v_k) données par cette méthode (où n est la taille de `temps`, à définir)
3. En considérant l'énergie cinétique ($\frac{1}{2}.m.v^2$) et l'énergie potentielle ($\frac{1}{2}.k.x^2$), que peut-on dire de la méthode d'Euler semi-implicite concernant la conservation de l'énergie mécanique ?

Pivot de Gauss

On suppose connues les fonctions suivantes :

- `dim(A)` : renvoie les dimensions de la matrice `A` : `(nb_lignes, nb_colonnes)`
- `pivot_partiel(A, j)` : renvoie le numéro de ligne correspondant au "pivot partiel" dans la colonne `j`
- `echanger(A, i1, i2)` : modifier la matrice `A` en échangeant les lignes `i1` et `i2`
- `transvection(A, i1, i2, mu)` : $L_{i_1} \leftarrow L_{i_1} + \mu L_{i_2}$
- `dilatation(A, i, c)` : $L_i \leftarrow c \times L_i$

```
def gauss(A: np.ndarray, B: np.ndarray) -> np.ndarray:
    n, p = dim(A)
    for j in range(p):
        ligne_pivot_partiel = pivot_partiel(A, j)
        echanger(A, j, ligne_pivot_partiel)
        echanger(B, j, ligne_pivot_partiel)
        for i in range(n):
            if i != j:   # pas sur la diagonale
                transvection(A, i, j, -A[i][j]/A[j][j])
                transvection(B, i, j, -A[i][j]/A[j][j])  # <--- Erreur à corriger : cf. questions ci-dessous
            else:
                coeff = 1/A[i][i]
                dilatation(A, i, coeff)
                dilatation(B, i, coeff)
    return A, B
```

4. Pivot partiel : écrire la fonction `pivot_partiel(M: np.ndarray, j: int) -> int` qui prend en entrée une matrice numpy et un numéro de colonne, et renvoie le numéro de ligne (sous la diagonale, au sens large) qui contient la plus grande valeur en valeur absolue.

[1] les réponses du type $x_{n+1} = x_{n+1}$ ne seront pas acceptées...

5. Le code précédent ne fonctionne pas comme il faudrait (cf. ligne commentée) : expliquer ce qui ne va pas et comment corriger ce problème. Dans la suite on supposera qu'on travaille avec la version corrigée.
6. Expliquez comment utiliser cette fonction pour calculer l'inverse de la matrice donnée par $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$. Et faites les calculs à la main (pour les matrices `A` et `B`).
7. Complexité : donner la complexité de cette fonction en précisant si besoin la complexité des différentes fonctions appelées.

Tri rapide : Complexité

8. Expliquer le principe du tri rapide, et préciser la complexité de chaque petite opération utilisée.
9. Écrire une fonction python récursive `rapide(t: list) -> list` qui renvoie une nouvelle liste triée.
10. On admet que la complexité du tri rapide vérifie $C(n) \leq 2 \times C(\frac{n}{2}) + O(n)$. Montrer que $C(n) = O(n.\log_2(n))$.

SQL - Cinéma

On dispose d'une base de donnée contenant les tables suivantes :

$films(\underline{id : integer}; titre : text; annee : date)$

$acteurs(\underline{id : integer}; nom : text; naissance : date)$

$casting(idf, ida)$

Remarques :

- *casting* permet de savoir qui a joué dans quel film : *idf* représente un identifiant de film et *ida* un identifiant d'acteur,
- on peut sans problème comparer des dates en SQL (ou utiliser MIN/MAX...).

11. Décrire cette base donnée en précisant quels sont les domaines, ainsi que les clés primaires et les clés étrangères.
12. Agrégats : écrire une requête SQL qui renvoie le nombre de films sortis chaque année entre 2000 et 2010. Que faut-il rajouter pour ne garder que les années où plus de 1000 films sont sortis ?
13. Jointure : écrire une requête SQL qui donne le nom de tous les acteurs de films dont le noms commence par "Batman" (on pourra utiliser l'opérateur LIKE). Attention aux noms d'attributs ambigus...
14. Requête imbriquée : écrire une requête SQL donnant les films dans lesquels ont joués les plus vieux acteurs (plus petite date de naissance).

Une valeurs propre

Une technique peu utilisée (et pas très efficace) pour calculer une valeur propre d'une matrice :

On considère une matrice $A \in \mathcal{M}_n(K)$ diagonalisable avec des valeurs propres $\lambda_1, ...\lambda_n$ par nécessairement distinctes, mais on suppose qu'une de ces VP, qu'on notera λ, a un module strictement plus grand que toutes les autres valeurs propres de A. Alors :

$$\lim_{k \to +\infty} \frac{Tr(A^{k+1})}{Tr(A^k)} = \lambda$$

Remarque : cette propriété n'a aucune raison d'être vraie dans le cas général. On suppose que notre matrice a cette propriété.

15. En supposant qu'on travaille avec une telle matrice, proposez une fonction python permettant de calculer cette valeur propre. Soyez créatifs.

- (Bonus, non corrigé) Justifiez mathématiquement ce résultat.