DS-02 - Correction

Euler semi-implicite

1. Avec la méthode d'Euler classique (**explicite**), on peut résoudre un système d'équations différentielles, mais il faut exprimer (x_{k+1}, v_{k+1}) en fonction de $\Delta t, t_k, x_k, v_k$ seulement. Dans une méthode implicite, on peut les exprimer en fonction de x_{k+1} et v_{k+1}, mais il faut résoudre une équation à chaque étape.

   Ici, x_{k+1} dépend de v_{k+1} mais v_{k+1} dépend de x_k : c'est une méthode semi-implicite.

   D'après l'énoncé on a : $\dot{x} = v$ donc $f : (t, v) \to v$ et $g : (t, x) \to -\omega x^2$.

   On obtient le système d'équations (attention, c'est v_{n+1} qui apparaît à la deuxième ligne, alors qu'avec Euler explicite on aurait v_n) :

$$\begin{cases} v_{n+1} = v_n - \omega^2 x_n \Delta t \\ x_{n+1} = x_n + v_{n+1} \Delta t \end{cases}$$

2. Comme pour Euler explicite, mais en faisant attention à v_{n+1}

```
def ressort(x_0, v_0, temps):
    x, v = x_0, v_0
    res = [(x, v)]
    for k in range(len(temps)-1):
        delta_t = temps[k+1]-temps[k]
        v = v - omega**2 * x * delta_t
        x = x + v * delta_t            # `v` a été modifié à la ligne précédente
        res.append((x, v))
    return res
```

ou

```
def ressort(x_0, v_0, temps):
    res = [[0, 0] for _ in range(len(temps)]
    res[0] = [[x, v]]
    for k in range(len(temps)-1):
        delta_t = temps[k+1]-temps[k]
        # idem, il faut mettre à jour v_{k+1} = res[k+1][1] avant x_{k+1}
        res[k+1][1] = res[k][1] - omega**2 * res[k][0] * delta_t
        res[k+1][0] = res[k][0] + res[k+1][1] * delta_t
    return res
```

3. Calculer $E_{n+1} - E_n$ et faire un DL : on n'obtient pas 0, mais un terme petit devant Δt

Pivot de Gauss

4. C'est un calcul de position du maximum classique. On travaille dans la colonne `j`:

```
def pivot_partiel(M: np.ndarray, j: int) -> int:
    p = j                                        # sous la diagonale : p >= j
    for i in range(j, len(M)):
        if abs(M[i][j]) > abs(M[p][j]):
            p = i
    return p
```

5. La ligne `tranvection(A, i, j, -A[i][j]/A[j][j])` modifie la matrice A. À la ligne suivante le terme `-A[i][j]/A[j][j]` n'est plus le même (il vaut 0). On peut écrire (et c'est plus simple à lire, écrire, comprendre) :

```
mu = -A[i][j]/A[j][j]      # C'est bien la même expression pour les deux matrices
transvection(A, i, j, mu)
transvection(B, i, j, mu)
```

6. Comme en maths, on prend $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ et $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ puis exécuter les opérations **données par le code python** et pas la méthode vue en cours (de maths, d'info). On obtiendrait évidement le même résultat.
7. Détailler la complexité de chaque ligne : au final, ce qui coûte cher ce sont les transvections/dilatations *dans les deux boucles* : on obtient $O(p \times n \times n) = O(n^3)$

Tri rapide

8. Voir le cours. Le découpage d'une liste de taille n en 3 morceaux se fait en $O(n)$. Puis la concaténation aussi (ou rien à faire si le tri est en place). Évidement, il faut prendre en compte les appels récursifs.
9. Idem
10. On se place dans le cas où $n = 2^k$. On a $C(2^k) \leq 2 \times C(2^{k-1}) + c.2^k$. Montrons par récurrence sur $k \geq 1$ que $C(2^k) \leq c.k \times 2^k$.
    - Initialisation : On suppose que $C(2^1) \leq c.1 \times 2^1$
    - Hérédité : $C(2^{k+1}) \leq 2 \times (c.k \times 2^{k+1}) + c.2^k = c(k+1)2^{k+1}$

SQL

11. On a 3 tables.
    - La première a 3 colonnes (attributs) de types (domaines) respectifs : entier, texte et date ; `id` est une clé primaire (identifiant unique).
    - Idem pour la deuxième table
    - La table `casting` compte deux clé étrangères (`idf` qui fait référence à `films.id`, et `ida`)
12. L'ordre des mots-clés est *toujours* le même

```
SELECT COUNT(*) as c
FROM films
WHERE 2000 <= annee AND annee <= 2010 -- selon le format il faudrait écrire "2000-01-01" pour la date
GROUP BY annee
HAVING c >= 1000                      -- à rajouter pour la deuxième partie de la question
```

13.

```
SELECT acteurs.nom
FROM acteurs JOIN films JOIN casting
    ON acteurs.id = casting.ida AND films.id = casting.idf
WHERE films.titre LIKE "Batman%"
```

14. Il faut des parenthèses pour la requête imbriquée. À connaître.

```
SELECT films.id, films.titre
FROM films JOIN acteurs JOIN casting
    ON acteurs.id = casting.ida AND films.id = casting.idf
WHERE acteurs.naissance = (SELECT MIN(naissance) FROM acteurs)
```

Valeur propre

15. On peut utiliser les outils de numpy (`np.eye(n)` pour l'identité, ainsi que `np.linalg.matrix_power(A, n)` pour A^n, ou `np.dot(A, B)` pour le produit) ou rédéfinir les fonctions soi-même. Pour la limite, il faut choisir une valeur de n "grande" quelconque. On peut surtout avoir la bonne idée de calculer des puissance de la forme A^{2^k} (comme dans l'exponentiation rapide) pour calculer rapidement un grande puissance de A.