TP4 — Relations entre objets

Objectif : Le but de ce TP est de reprendre la classe **`Matrice`** du TP3 représentant des matrices de $\mathcal{M}_n(\mathbb{R})$, d'écrire une classe **`Vecteur`** représentant un vecteur de $\mathbb{R}^n$, et d'écrire des méthodes permettant de modéliser les relations entre les matrices et les vecteurs.

Rappels du TP3

Dans ce TP, on reprendra la classe **`Matrice`** du TP3. On pourra par exemple partir du corrigé disponible sur `moodle`.

De plus, on introduira une classe **`Vecteur`**, permettant de représenter un vecteur colonne de $\mathbb{R}^n$ via une liste.
Un squelette de cette classe est donné ci-dessous.

```python
class Vecteur:

    # constructeur
    def __init__(self, liste):
        self.V = liste
        self.n = len(self.V)

    # surcharge de str(Vecteur) pour le print
    def __str__(self):
        return str(self.V)

    # surcharge des crochets : pour l'accès
    def __getitem__(self, index):
        return self.V[index]

    # surcharge des crochets : pour la modification
    def __setitem__(self, index, valeur):
        self.V[index] = valeur
```

Quelques notations

Soit $A \in \mathcal{M}_n(\mathbb{R})$, que l'on écrit de la façon suivante.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix}$$

Les diagonales numéro 0, 1, 2 et -1 de A seront alors les vecteurs suivants :

$$\begin{pmatrix} a_{1,1} \\ a_{2,2} \\ \vdots \\ a_{n,n} \end{pmatrix} \in \mathbb{R}^n, \quad \begin{pmatrix} a_{1,2} \\ a_{2,3} \\ \vdots \\ a_{n-1,n} \end{pmatrix} \in \mathbb{R}^{n-1}, \quad \begin{pmatrix} a_{1,3} \\ a_{2,4} \\ \vdots \\ a_{n-2,n} \end{pmatrix} \in \mathbb{R}^{n-2}, \quad \begin{pmatrix} a_{2,1} \\ a_{3,2} \\ \vdots \\ a_{n,n-1} \end{pmatrix} \in \mathbb{R}^{n-1}.$$

Interactions possibles entre les classes `Matrice` et `Vecteur`

Dans le reste du TP, le nombre d'étoiles représente la difficulté de l'implémentation.

On pourra par exemple implémenter les méthodes ci-dessous.

Méthodes de la classe `Matrice`

* Une méthode `diag` de la classe **`Matrice`**, qui renvoie une instance de la classe **`Vecteur`** contenant les éléments de la diagonale de la matrice.

** On pourra aussi modifier cette méthode `diag` en lui ajoutant un paramètre optionnel `i`. La méthode renverra alors un **`Vecteur`** de taille $n - |i|$ contenant la i-ème diagonale de la matrice.

** Une méthode `produit` de la classe **`Matrice`**, entre une instance de **`Matrice`** et une instance de **`Vecteur`**. Cette méthode renverra une instance de la classe **`Vecteur`** contenant le résultat du produit matrice-vecteur.

*** Une surcharge de l'opérateur `@` pour renvoyer le résultat de la méthode `produit` lorsque `@` est appliqué entre une **`Matrice`** et un **`Vecteur`**. On rappelle que l'opérateur `@` est surchargé via **`__matmul__`**.
On utilisera la fonction *built-in* `isinstance` pour déterminer le type de l'argument dans la surcharge de `@`. La méthode appropriée (produit matrice-matrice ou matrice-vecteur) sera appelée en fonction du type de l'argument.

**** On propose de résoudre le système $Ax = b$ avec $A \in \mathcal{M}_n(\mathbb{R})$, $b \in \mathbb{R}^n$ et $x \in \mathbb{R}^n$. Pour cela, on souhaite calculer l'inverse A^{-1} de A. On utilisera :

$$A^{-1} = \frac{1}{\det(A)} \left(\operatorname{Com} A\right)^T,$$

où $\operatorname{Com} A \in \mathcal{M}_n(\mathbb{R})$ est la comatrice de A, définie par :

$$\left(\operatorname{Com} A\right)_{ij} = (-1)^{i+j} \det\left(A_{ij}\right),$$

avec A_{ij} un mineur de A, i.e. la matrice A dépourvue de sa i-ème ligne et de sa j-ième colonne. Il faudra également calculer le déterminant de A :

$$\det(A) = \sum_{j=1}^{n} a_{ij} (-1)^{i+j} \det\left(A_{ij}\right),$$

ou toute autre formule qu'on pourra trouver utilisant la comatrice.

Méthodes de la classe `Vecteur`

** Une méthode `diag` de la classe **`Vecteur`**, qui renvoie une instance de la classe **`Matrice`** dont la diagonale est le vecteur.

*** On pourra aussi modifier cette méthode `diag` en lui ajoutant un paramètre optionnel `i`. La méthode renverra alors une **`Matrice`** de taille $n + |i|$ dont la i-ème diagonale est le vecteur.