TP : «introduction au calcul matriciel»

On rappelle qu'on peut modéliser une matrice comme une liste de listes. Par exemple, pour écrire la matrice $A = \begin{pmatrix} 4 & 5 & 8 \\ 2 & 1 & 7 \end{pmatrix}$, on écrira la liste de ses lignes : `A = [ [4,5,8], [2,1,7] ]`.

Ainsi `A[0][1]` est l'élément de la ligne d'indice 0 et de la colonne d'indice 1, soit le nombre 5. De même, `A[1][2]` vaut 7.

Les «comprehension list» sont utiles pour construire des matrices. Par exemple, pour la matrice nulle à 3 lignes et 4 colonnes, `M= [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]` on pourra écrire

`M = [ [ 0 for j in range(4)] for i in range(3)]`.

Attention aux copies de listes et de matrices

Les objets de type `list` en Python sont mutables. C'est pratique mais aussi dangereux.

Exercice 1 (Les dangers de la copie) Deviner les valeurs affichées à la fin des scripts et expliquer. On pourra regarder les id des tableaux ou de leurs éléments.

On pourra aussi utiliser un «super et didactique» programme qui trace l'exécution des scripts : il se trouve à l'adresse suivante `http://www.pythontutor.com/visualize.html`.

```
t1 = [5,3,8,9]
t2 = t1
t3 = t1[:]
t1[2] = 7
print(t2[2],t3[2])
```

```
t1 =[[2,3],5]
t3 = t1[:]
t1[0][0] = 7
print(t3)
```

```
v = [3,5,6]
M = 3 * v
N = [v,v,v]
v[0] = 2
print(N)
```

Premières fonctions du calcul matriciel

Exercice 2 (Quelques fonctions utiles pour la suite)

1. Créer une fonction `dimensions(A)` qui prend en argument une matrice A et renvoie le couple (n, p) représentant son nombre de lignes et son nombre de colonnes.
2. Utiliser la fonction `affiche` qui affiche [1] une matrice ligne par ligne, pour visualiser vos futurs résultats :

```
def affiche(A):
    for ligne in A:
        print(ligne)
```

Pour créer une matrice remplie de zéros, on pourra utiliser la fonction suivante `matrice_nulle(n, p)` prend en arguments deux entiers naturels n et p et renvoie la matrice nulle de $\mathcal{M}_{n,p}(\mathbb{R})$.

1. Les tableaux du module numpy de type array ou matrix sont directement affichés ainsi.

```
def matrice_nulle(n,p):
 return [ [0 for j in range(p)] for i in range(n)]
```

3. Créer une fonction `matrice_unite(n)` qui prend en argument un entier naturel n et renvoie la matrice unité de $\mathcal{M}_n(\mathbb{R})$. On vérifiera que sa complexité est linéaire.

4. Créer une fonction `transpose(A)` qui prend en argument une matrice A et renvoie sa transposée.

5. Écrire une fonction booléenne `est_triangulaire_sup(A)` qui teste si la matrice A est triangulaire supérieure. Si A est carréee de taille n, combien de de comparaisons au pire effectue l'algorithme ? Et dans le meilleur des cas ?

Exercice 3 (Les trois opérations matricielles fondamentales)

1. Créer une fonction `somme_matrice` qui code la somme de deux matrices. On pourra compléter le script suivant. La fonction `assert` teste si les matrices ont le même format. Si ce n'est pas le cas, elle renvoie un message d'erreur.

```
def somme_matrice(A,B):
  n,p = dimensions(A)
  q,r = dimensions(B)
  assert (n,p)  == (q,r)
  C = matrice_nulle(...
  ....
  return C
```

2. Créer une fonction `mult_scalaire` qui code la multiplication d'une matrice par un scalaire.

3. Créer une fonction `produit_matrice` qui code le produit de deux matrices.

4. Combien de multiplications et d'additions sont nécessaires pour calculer le produit de deux matrices carrées de taille n ? En déduire la complexité du produit matriciel. Chercher sur Internet s'il existe des algorithmes plus performants.

Exercice 4 (Un exo de blocs) Écrire une fonction `blocs(A,a,b)` qui prend en arguments une matrice $A \in \mathcal{M}_n(\mathbb{R})$ et des entiers $a, b \in \mathbb{N}^*$ et renvoie une matrice «blocs»avec a «lignes» et b «colonnes» et telle que chaque bloc est égal à A. La matrice A vit donc dans $\mathcal{M}_{n*a,n*b}$ et est constituée de $a \times b$ blocs égaux à A. Par exemple, `blocs(A,2,3)` renvoie une matrice blocs de la forme $M = \begin{pmatrix} A & A & A \\ A & A & A \end{pmatrix}$

Exponentiation

Exercice 5 (Exponentiation)

1. Créer une fonction `exponaif(A, n)` qui renvoie la puissance n-ième d'une matrice A.

   Combien de multiplications matricielles sont effectuées par `exponaif(A, n)` ?

Cette méthode est qualifiée de naïve, car il existe une méthode d'exponentiation plus efficace dite exponentiation rapide.

2. Calculer A^0, A^{45} et A^{46} avec $A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$.

3. On note x et y les coefficients de la deuxième ligne de la matrice A^{45}. Rappeler quel est le plus grand entier **relatif** que l'on puisse coder sur 32 bits en utilisant le codage par complément à deux. En déduire que les entiers x et y peuvent être représentés sur 32 bits. Et leur somme $x + y$?

4. `Un peu de numpy` :

   (a) Importer la librairie `numpy` en la renommant `np`.

   (b) Refaire les calculs précédents des puissances de A avec la fonction `matrix_power` de la librairie `numpy.linalg` («linalg» est le diminutif de «linear algebra»).

   (c) Commenter et expliquer les résultats. On pourra regarder le type des coefficients renvoyés par `matrix_power`.

   (d) Exécuter `A = np.array([[0,1], [1,1]],dtype=np.int64)` puis recalculer A^{45} avec `matrix_power`. Commenter.

Exercice 6 (Exponentiation rapide) C'est une méthode très efficace pour calculer les puissances n-ièmes d'un réel x (ou plus généralement d'un élément dans un groupe multiplicatif, par exemple une matrice carrée). Elle repose sur le principe suivant où l'on décompose l'entier n en base 2 :

$$\begin{cases} x^0 &= 1 \\ x^n &= (x^2)^{\frac{n}{2}} & \text{, si } n \text{ est pair et } n > 0 \\ x^n &= x \times (x^2)^{\frac{n-1}{2}} & \text{, si } n \text{ est impair} \end{cases}$$

1. Soit x un réel. Appliquer l'algorithme d'exponentiation rapide pour calculer x^8 puis x^{14}. Indiquer dans les deux exemples le nombre de multiplications effectuées puis comparer avec la méthode «naïve».

2. Pour ceux qui connaissent, écrire une version récursive de l'algorithme d'exponentiation rapide, on le notera `exporap_rec`.

   Sinon, programmer une version itérative, on pourra démarrer ainsi :

```
def exporap(x,n):
    e = n # exposant
    a = x # nombre à exponentier
    p = 1 # produit
    while e...
        ...
    return p
```

3. Démontrer que le nombre de multiplications qu'effectue `exporap(x,n)` est inférieur ou égal à $2(\log_2(n) + 1)$.

4. Adapter votre code pour obtenir une fonction `exporap_matrice` qui renvoie la puissance n-ième d'une matrice.

Exercice 7 (Nombres de Fibonacci) La suite de Fibonacci (F_n) est la suite définie par $F_n = F_{n-1} + F_{n-2}$ pour $n \geqslant 2$ et $F_0 = 0$ et $F_1 = 1$. On pose $X_n = \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}$.

1. Déterminer une matrice $A \in \mathcal{M}_2(\mathbb{R})$ telle que $X_{n+1} = AX_n$.
2. Calculer A^{50}. En déduire F_{50} (réponse 12586269025).
3. En déduire une fonction `fibo_mat` qui renvoie le n-ième nombre de Fibonacci.
4. Expérimenter et comparer les temps de calcul par rapport à l'algorithme classique.