Informatique en CPGE (2017-2018)
Exercices : matrices

Exercice 1 : calculs divers
On travaille avec des matrices carrées.
Pour créer une matrice, on écrira par exemple : `M=[[3,1,1],[0,2,-1],[1,1,4]]`.

1. Transposée d'une matrice.
   Ecrire une fonction `transpose(A)` qui prend en argument une matrice carrée `A` et renvoie la matrice transposée de `A`.
2. Trace d'une matrice.
   Ecrire une fonction `trace(A)` qui prend en argument une matrice carrée `A` et renvoie la trace de la matrice `A`.
3. Somme de deux matrices.
   Ecrire une fonction `somme(A,B)` qui prend en arguments deux matrices `A` et `B` et renvoie la matrice somme `A+B`. Il faudra vérifier les tailles des matrices.
4. Produit d'une matrice par un réel.
   Ecrire une fonction `prod_reel(A,k)` qui prend en arguments une matrice `A` et un scalaire `k` et renvoie la matrice produit de `A` par `k`.
5. Produit de deux matrices.
   Ecrire une fonction `produit(A,B)` qui prend en arguments deux matrices `A` et `B` et renvoie la matrice produit `A.B`. Il faudra vérifier les tailles des matrices.
6. Puissance d'une matrice et valeurs propres.
   (a) Ecrire une fonction `puissance(A,k)` qui prend en arguments une matrice `A` et un entier naturel `k` et renvoie la matrice A^k. (Ecrire une version non récursive et une version récursive).
   (b) Soit `M=[[3,1,1],[0,2,-1],[1,1,4]]`. Ecrire le code permettant de calculer et d'afficher le quotient des traces de M^{k+1} et M^k pour `k` variant de 1 à 50. Vérifier que ce quotient semble tendre vers 4 qui est la plus grande valeur propre de `M`.
7. Matrice orthogonale.
   (a) Ecrire une fonction `orthogonale(A)` qui prend en argument une matrice `A` et renvoie `True` si la matrice `A` est orthogonale et `False` sinon.
   (b) Soit `M=[[1,2,2],[2,-2,1],[2,1,-2]]`. A l'aide des fonctions précédentes, vérifier que $\frac{1}{3}M$ est orthogonale puis que le produit de $\frac{1}{3}M$ par sa transposée est bien égal à la matrice identité.

Exercice 2 : calcul de déterminants

On utilise l'algorithme du pivot de Gauss pour obtenir une matrice triangulaire.

1. Ecrire une fonction **matrice(n)** d'argument n entier strictement positif qui renvoie la matrice carrée nulle de taille n.
2. Ecrire une fonction **copie(m)** d'argument une matrice carrée m qui renvoie une copie de cette matrice.
3. Ecrire une fonction **change(m,i,j)** d'argument une matrice carrée m et deux entiers i et j, qui renvoie la matrice obtenue en échangeant les lignes d'indices i et j de la matrice m. On fera une copie de m pour ne pas la modifier.
4. A chaque étape, on utilise le plus grand pivot possible. Ecire une fonction **pivot(m,s)** qui prend en argument une matrice m et le numéro du pivot que l'on cherche, (0 pour la première étape), et renvoie le numéro de la ligne contenant le pivot qui va être utilisé.
5. Les transvections sont les transformations centrales dans l'algorithme du pivot de Gauss. Si s est le numéro du pivot utilisé, on remplace chaque ligne m[i], pour i variant de s+1 à n-1, par m[i]- k*m[s], où k=m[i][s]/m[s][s], soit $L_i \longleftarrow L_i - \dfrac{a_{i,s}}{a_{s,s}} L_s$.

   Ecrire une fonction **transvection(m,s)** qui prend en argument une matrice m et le numéro s du pivot utilisé et renvoie la nouvelle matrice.
6. Ecrire une fonction **gauss(m)** qui prend en argument une matrice carrée m et renvoie la matrice triangulaire supérieure obtenue avec l'algorithme de Gauss.
7. Le déterminant de m s'obtient alors avec le produit des éléments diagonaux de m, à un détail près : il faut multiplier ce produit par $(-1)^k$, où k est le nombre d'échanges de lignes effectués pour utiliser le meilleur pivot. Compléter la fonction **gauss(m)** afin qu'elle compte le nombre d'appels à la fonction **change** et renvoie ce nombre.

   Ecrire alors une fonction **determinant(m)** d'argument une matrice carrée m, qui renvoie le déterminant de m.
8. Tester le programme avec la matrice m1=[[3,1,1],[0,2,-1],[1,1,4]] de déterminant 24, puis la matrice m2=[[3,1,1],[1,1,4],[0,2,-1]] de déterminant -24 et enfin la matrice m3=[[12,-51,4],[6,167,-68],[-4,24,-41]] de déterminant -85750.

Exercice 3 : projection orthogonale

Un vecteur est représenté par une liste (la liste de ses coordonnées).

1. Ecrire une fonction `prod_scalaire(u,v)` qui prend en argument deux vecteurs `u` et `v`, et renvoie le produit scalaire <`u,v`>.
2. Ecrire une fonction `projection(u,sev)` qui prend en argument un vecteur `u` et un sous-espace vectoriel `sev` et renvoie le projeté orthogonal de `u` sur le sous-espace engendré par $v_0, \dots, v_j$. On supposera que $(v_0, \dots, v_j)$ constitue une base orthonormée du sous-espace vectoriel `sev` qui sera représenté par une liste dont les éléments représentent chaque vecteur v_i.

Exercice 4 : décomposition QR

1. Reprendre les fonctions `transpose`, `produit`, `prod_scalaire` et `projection` des programmes précédents et écrire une fonction `norme(u)` qui prend en argument un vecteur `u` et renvoie la norme de ce vecteur.
2. Ecrire une fonction `gramschmidt(m)` qui prend en argument une matrice inversible `m` et qui renvoie une matrice `q` obtenue de la manière suivante : les colonnes de `m` représentent une famille libre de vecteurs $(v_1, \dots, v_n)$; on commence par transposer la matrice `m` et on obtient une matrice

`tm` telle que les vecteurs soient représentés par les lignes de cette matrice. On utilise ensuite le procédé de Gram-Schmidt pour orthonormaliser cette famille et construire une famille orthonormale $(e_1, \ldots, e_n)$ qui engendre les mêmes espaces vectoriels successifs (pour tout j inférieur à n, $F_j = Vect(e_1, \ldots, e_j) = Vect(v_1, \ldots, v_j)$). On modifie ainsi la matrice `tm` et il ne reste plus qu'à la transposer pour obtenir la matrice `q`.

Rappel : l'étape générale de l'algorithme consiste à soustraire au vecteur v_{j+1} sa projection orthogonale sur l'espace $F_j = Vect(e_1, \ldots, e_j)$.

On note le projecteur sur la direction u par $\text{proj}_u v = \dfrac{< u, v >}{< u, u >} u$.

L'algorithme s'écrit :

$$u_1 = v_1 \ , \quad e_1 = \frac{u_1}{||u_1||}$$

$$u_2 = v_2 - \text{proj}_{e_1} v_2 \ , \quad e_2 = \frac{u_2}{||u_2||}$$

$$u_3 = v_3 - \text{proj}_{e_1} v_3 - \text{proj}_{e_2} v_3 \ , \quad e_3 = \frac{u_3}{||u_3||}$$

$\ldots$

$$u_k = v_k - \sum_{j=1}^{k-1} \text{proj}_{e_j} v_k, \quad e_k = \frac{u_k}{||u_k||}$$

Tester avec la matrice m=[[1,2,2],[2,-2,1],[2,1,-2]] ; on obtient q $= \frac{1}{3}$ m.

3. Ecrire une fonction `decompQR(m)` qui prend en argument une matrice inversible `m` et qui renvoie les matrices `q` et `r` de la décomposition QR de `m`.
   Rappel : la factorisation QR de M s'écrit $M = QR$ avec Q orthogonale ($Q^T Q = I$) et R triangulaire supérieure ; de plus si $M = QR$ alors $R = Q^T M$.
   Tester avec la matrice `m=[[1,2,2],[2,-2,1],[2,1,-2]]`.
   (On obtient q $= \frac{1}{3}$ m et `r=[[3,0,0],[0,3,0],[0,0,3]=3*Id`).
4. Si $R = Q^T M$ alors $RQ = Q^T MQ$ et RQ est orthogonalement semblable à M ; on pose alors $T = RQ$ et on réitère ce processus avec T (décomposition QR de T puis calcul du produit RQ). On montre que si on itère ce processus n fois la matrice T converge quand n tend vers $+\infty$ vers une matrice triangulaire supérieure (orthogonalement semblable à M) avec laquelle on peut alors calculer le déterminant et les valeurs propres de M.
   (a) Ecire une fonction `iterQR(m)` qui itère le processus ci-dessus 100 fois sur la matrice `m` et renvoie la matrice `t` obtenue.
   (b) Ecrire alors deux fonctions `vp(m)` et `det(m)` qui renvoient respectivement les valeurs propres et le déterminant de `m`.
   (c) Tester sur `m=[[5,16,-14],[16,-1,2],[-14,2,14]]` ; on obtient les valeurs propres -18, 9 et 27 et le déterminant -4374.