TP: produit scalaire et algorithme de Gram-Schmidt
Informatique pour tous

Dans ce TP, nous allons appliquer Python au cours de mathématiques sur les espaces euclidiens. Pour simplifier, nous allons nous placer dans l'espace $\mathbb{R}^n$ muni du produit scalaire canonique.

Un vecteur sera représenté par un tableau numpy. Par exemple, $\begin{pmatrix}2\\3\\0\end{pmatrix}$ sera représenté par `np.array([2, 3, 0])`.

On commencera donc par importer numpy (`import numpy as np`). On rappelle que l'on peut additionner (ou soustraire) deux tableaux et multiplier un tableau par une constante (les opérations sont alors effectuées composante par composante).
Toutes les fonctions doivent bien sûr être testées.

I Fonctions utilitaires

1. Écrire une fonction `somme` renvoyant la somme des éléments d'un vecteur.
2. En déduire une fonction `produit_scalaire` effectuant le produit scalaire de deux vecteurs (de même taille...).
3. En déduire une fonction `norme` renvoyant la norme d'un vecteur.

II Base orthonormée et algorithme de Gram-Schmidt

Soit $B = (u_0, ..., u_p)$ des vecteurs linéairement indépendants. On rappelle que l'algorithme de Gram-Schmidt consiste à construire une famille orthogonale $(e_0, ..., e_p)$ telle que que $Vect(e_0, ..., e_p) = Vect(B)$, de la façon suivante:

$$e_0 = u_0$$

$$\forall k \in \{1, ..., p\},\ e_k = u_k - \sum_{i=0}^{k-1} \frac{< u_k, e_i >}{< e_i, e_i >} e_i$$

1. Écrire une fonction `gram_schmidt` ayant en argument une liste contenant les vecteurs $u_0, ..., u_p$ et renvoyant la liste des vecteurs $e_0, ..., e_p$ définie ci-dessus.
   Vérifier que si $u_0 = \begin{pmatrix}1\\2\\1\end{pmatrix}$, $u_1 = \begin{pmatrix}2\\3\\0\end{pmatrix}$, $u_2 = \begin{pmatrix}0\\1\\0\end{pmatrix}$ alors on trouve $e_0 = \begin{pmatrix}1\\2\\1\end{pmatrix}$, $e_1 = \frac{1}{3}\begin{pmatrix}2\\1\\-4\end{pmatrix}$, $e_2 = \frac{1}{7}\begin{pmatrix}-3\\2\\-1\end{pmatrix}$.
2. Écrire une fonction `normaliser` ayant une liste de vecteurs orthogonaux en argument et calculant leur orthonormalisation (c'est à dire diviser chaque vecteur par leur norme). On pourra modifier la liste en argument plutôt qu'en renvoyer une nouvelle.
3. Écrire une fonction `est_orthonormale` déterminant si une liste de vecteurs forme une famille orthonormale.
   Attention: à cause des erreurs d'arrondis sur les flottants, ne pas tester une égalité entre deux flottants mais regarder plutôt si la différence (en valeur absolue) entre les deux est suffisamment petite (par exemple au plus 0.01).
   Vérifier qu'appliquer `gram_schmidt` puis `normaliser` donne bien une famille orthonormale, en utilisant l'exemple de la question II.1.
4. En déduire une fonction `mat_orthogonale` déterminant si une matrice (soit une liste de listes, soit un tableau `numpy`) est orthogonale (c'est à dire si la famille de ses colonnes est une base orthonormale).
   Vérifier que $\frac{1}{\sqrt{6}}\begin{pmatrix}\sqrt{2} & -\sqrt{3} & 1\\ \sqrt{2} & \sqrt{3} & 1\\ \sqrt{2} & 0 & -2\end{pmatrix}$ est orthogonale, mais que ce n'est plus le cas si on change un coefficient.

III Application au calcul de distance

Soit $B = (e_0, ..., e_p)$ une famille orthonormée. On rappelle que la projection orthogonale d'un vecteur u sur $Vect(B)$ est donnée par $p_B(u) = \sum_{k=0}^{p} < u, e_k > e_k$. On rappelle que la distance de u à $Vect(B)$ est alors $\|u - p_B(u)\|$.

1. Écrire une fonction `projection` telle que, si `u` est un vecteur et `B` une liste de vecteurs formant une famille orthonormée, `projection(u, B)` renvoie le projeté orthognal de `u` sur $Vect(B)$.

2. Écrire une fonction `distance` telle que, si `u` est un vecteur et `F` une liste de vecteurs indépendants, `distance(u, F)` renvoie la distance de u à $Vect(F)$.

   Vérifier que la distance de $\begin{pmatrix}0\\1\\1\end{pmatrix}$ à $Vect(\begin{pmatrix}1\\-1\\2\end{pmatrix}, \begin{pmatrix}1\\0\\1\end{pmatrix})$ est environ égale à 1,15 et que la distance de $\begin{pmatrix}2\\-1\\3\end{pmatrix}$ à $Vect(\begin{pmatrix}1\\-1\\2\end{pmatrix}, \begin{pmatrix}1\\0\\1\end{pmatrix})$ est 0 (pourquoi?).