Polynômes

Dans ce TP, on modélise $P = \sum_{k=0}^{n} a_k X^k \in \mathbb{R}[X]$ comme une liste `P` telle que pour tout $k \in [\![0 ; n]\!]$, `P[k]` vaille a_k. On remarque que $n+1 = \text{len(P)}$ et donc que $n = \text{len(P)} - 1$. Par exemple, si $P = 3X^2 + 8$, `[8,0,3]` est une liste représentant P, tout comme la liste `[8,0,3,0,0]`, en effet : $P = 0X^4 + 0X^3 + 3X^2 + 0X + 8$. Il n'a y a donc pas unicité de la liste représentant un polynôme.

1. Écrire une fonction `ProduitNbPol(v,P)` qui, à $v \in \mathbb{R}$ et $P \in \mathbb{R}[X]$, renvoie vP.
2. Écrire une fonction `Somme(P,Q)` qui, $(P, Q) \in \mathbb{R}[X]^2$, renvoie $P + Q$. Attention, P et Q n'ont pas forcément la même longueur.
3. Écrire une fonction `Lotus(P,Q)` qui, à $(P, Q) \in \mathbb{R}[X]^2$, renvoie PQ.
4. En déduire une fonction **récursive** `Puissance(P,n)` qui, à $P \in \mathbb{R}[X]$ et $n \in \mathbb{N}$, renvoie P^n.
5. Écrire une fonction `Composition(P,Q)` qui, à $(P, Q) \in \mathbb{R}[X]^2$, renvoie $P \circ Q$.
6. Écrire une fonction `Derivée(P)` qui, à $P \in \mathbb{R}[X]$, renvoie P'.
7. Écrire une fonction `Évaluation(P,a)` qui, à $P \in \mathbb{R}[X]$ et $a \in \mathbb{R}$, renvoie $P(a)$.
8. Le professeur Tournesol, a développé un polynôme sous forme factorisée et a obtenu :
$$\begin{aligned} P =& 3X^6 - 201X^5 - 44694X^4 + 2424642X^3 \\ &+ 216702003X^2 - 7321221369X - 339831876384 \end{aligned}$$
Mais comme il est très distrait, il a perdu la forme factorisée. Grâce aux fonctions précédentes, aidez-le à retrouver la factorisation, sachant qu'il se souvient seulement que les racines sont des entiers entre -100 et 100.
9. Une fois que vous avez la factorisation, redéveloppez (avec Python pas à la main) pour vérifier vos calculs.
10. Écrire une fonction `Degree(P)` qui, à $P \in \mathbb{R}[X]$ renvoie $d^\circ P$. Pour le polynôme nul, renvoyer $-\infty$ grâce à la commande `-float("inf")`
11. Soit $n \in \mathbb{N}^*$ et $k \in [\![0 ; n]\!]$, on rappelle l'expression du polynôme de Bernstein :
$$B_{n,k} = \binom{n}{k} X^k (1 - X)^{n-k}$$
On considère $P_0, \dots, P_n$ des points du plan sous la forme $P_i = (x_i, y_i)$ et on appelle courbe de Bézier[1] des points $P_0 \dots, P_n$, la courbe $t \mapsto (x(t), y(t))$ pour $t \in [\, 0 ; 1 \,]$ où
$$x(t) = \sum_{k=0}^{n} x_k B_{n,k}(t) \quad \text{et} \quad y(t) = \sum_{k=0}^{n} y_k B_{n,k}(t)$$
Écrire une fonction `Bézier(P)` qui ne renvoie rien mais qui affiche la courbe de Bézier des points P_i, où $P = [P_0, \dots, P_n]$, ainsi que les points P_i.

Inversion de matrices

Dans cette partie, les matrices sont, au choix, modélisées comme des listes de listes ou comme des array de numpy. Le but étant de reprogrammer nous-même des fonctions sur les matrices : comme la somme, le produit par un scalaire, le produit entre deux matrices et l'inversion d'une matrice inversible (ces fonctions sont bien sûr déjà implémentés dans numpy). Contrairement aux notations mathématiques usuelles, les indices commenceront à 0, ainsi, si $M \in \mathscr{M}_{n,p}(\mathbb{R})$, $M = (m_{i,j})_{\substack{0 \leqslant i \leqslant n-1 \\ 0 \leqslant j \leqslant p-1}}$

1. Écrire une fonction `Somme(A,B)`, qui a deux matrices de même taille, renvoie $A + B$.
2. Écrire une fonction `Produit(A,B)` qui, à deux matrices A et B, renvoie $A \times B$ (on vérifiera avant que le produit est bien possible et on renverra une erreur dans le cas contraire).
3. Écrire une fonction `MultiLigne(M,i,v)` qui multiplie la i-ième ligne de M par v, cette fonction modifie M par effet de bord.
4. Écrire une fonction `AjoutLigne(M,i,j,v)` qui effectue l'opération $L_i \leftarrow L_i + vL_j$ (aussi par effet de bord)
5. Écrire une fonction `PermutLigne(M,i,j)` qui permute les lignes i et j de la matrice (aussi par effet de bord).

1. Inventée par Pierre Bézier ingénieur chez Renault et de façon indépendante par Paul de Casteljau ingénieur chez Citroën.

6. Écrire une fonction `RangPivot(M,j)` qui renvoie un indice de ligne $i \in [\![j\,; n-1]\!]$ tel que $M[i][j] \neq 0$ (on suppose qu'il y a existence d'un tel coefficient)
7. Écrire une fonction `Échelonnement(M)` qui échelonne en ligne la matrice M ayant n ligne de la façon suivante : pour chaque colonne $j \in [\![0\,; n-1]\!]$:
   - Trouver le pivot
   - L'échanger avec la j-ième ligne.
   - Multiplier la j-ième ligne par $M[j][j]^{-1}$, ainsi $M[j][j]$ vaut maintenant 1.
   - Pour $i \neq j$, effectuer $L_i \leftarrow L_i - M[i][j]L_j$ de façon à ce qu'il y n'ait que des zéros sur la j-ième colonne (sauf $M[j][j] = 1$)
8. Soit $P \in \mathscr{M}_n(\mathbb{R})$ une matrice supposée inversible. Écrire une fonction `Augmenter(P)` qui, à P, renvoie (sans modifier P) $B \in \mathscr{M}_{n,2n}(\mathbb{R})$ dont les n premières colonnes de B soient celles de P et les n dernières de B soit les colonnes de I_n.
9. Écrire une fonction `Inverser(P)` qui échelonne la matrice B, obtenue par la fonction précédente. Après échelonnement, les n dernières colonnes de B forment donc P^{-1}. Renvoyer ces n dernières colonnes renvoie donc P^{-1}.
10. Modifier les fonctions précédente pour que `Inverse(P)` renvoie "P non inversible" si, au moment de traiter la j-ième colonne, on s'aperçoit que tous les coefficients de la colonne j à partir de la j-ième ligne sont tous nuls.

Retour sur les polynômes

1. Soit $P = \sum_{k=0}^{n} a_k X^k \in \mathbb{R}[X]$, calculer le nombre de multiplications nécessaires pour calculer $P(x) = \sum_{k=0}^{n} a_k x^k$.
2. Si $P = aX^3 + bX^2 + cX + d$ et $x \in \mathbb{R}$, alors $P(x) = d + x(c + x(b + ax))$. De même pour des polynômes de n'importe quel degré. Combien cela fait-il de multiplications pour un polynôme de degré n ? Cette méthode s'appelle **la méthode de Horner**
3. Implémenter une fonction `ÉvaluationHorner(P,a)` renvoyant la valeur de $P(a)$ utilisant cette factorisation