


TP 3 – INTERPOLATION POLYNOMIALE

Le but de ce TP est de construire des approximations polynomiales de fonctions. On insistera sur les polynômes interpolateurs de Lagrange.




EXERCICE 3.1 – POLYNÔMES DE LAGRANGE – Cet exercice se concentre sur les polynômes d'interpolations de Lagrange.

1. *Une construction naïve.* On reprend la présentation du cours afin d'implémenter des fonctions permettant la représentation de polynômes interpolateurs de Lagrange.




- a)  Écrire une fonction python `BaseLagrange(X, listX, i)` qui prend en entrée un vecteur X , une liste de nombres `listX` et un entier positif i strictement plus petit que la longueur de `listX` et qui renvoie l'image du vecteur X par la fonction polynomiale

$$\begin{aligned} \mathbb{R} &\longrightarrow \mathbb{R} \\ x &\longmapsto \prod_{\substack{j=1 \\ j \neq i}}^n \frac{x-x_j}{x_i-x_j} . \end{aligned}$$

où les x_i sont les éléments de `listX`.



- b)  Utiliser la fonction précédente pour visualiser ces polynômes avec un choix raisonnable de `listX`, par exemple `listX = [0,1,2]`.
 - c)  Écrire une fonction python `InterLagrange1(X, listX, listY)` qui prend en entrée un vecteur X et deux listes de nombres `listX` et `listY` de même taille n et qui renvoie l'image de X par le polynôme interpolateur de Lagrange de degré au plus n passant les points (x_i, y_i) pour $x_i = \text{listX}[i]$ et $y_i = \text{listY}[i]$.
 - d)  Utiliser la fonction précédente pour visualiser ces polynômes avec des choix raisonnables de `listX` et `listY`, comme par exemple `listX = [0,1,2]` et `listY = [-1,3,2]`.
2. *Interlude : l'algorithme de Horner.* On présente ici une méthode afin d'optimiser le temps de calcul pour l'évaluation d'un polynôme. On considère un polynôme


$$P(X) = a_0 + a_1X + \cdots + a_N X^N .$$

- a)  Implémenter une fonction (naïve) `EvalP(b, listCoef)` qui prend en entrée un nombre b et une liste de coefficients `listCoef = [a0, ..., aN]` et qui renvoie $P(b)$.
- b)  Estimer le nombre d'opérations (sommes et produits) nécessaires au calcul dans la fonction précédente.
- c)  Constater que l'on a la factorisation suivante :

$$P(X) = a_0 + X(a_1 + X(a_2 + \cdots X(a_{N-1} + Xa_N) \cdots)).$$

Estimer le nombre d'opérations (sommes et produits) nécessaires à l'évaluation du polynôme P en b en utilisant la factorisation précédente.

- d)  Implémenter une fonction `Horner(b, listCoef)` qui utilise la factorisation pour évaluer le polynôme P .
 - e)  Implémenter une fonction `compare` qui compare le temps d'exécution de `Horner` et de `EvalP`. On prendra une liste de coefficient relativement grande et on pourra itérer N fois chacune fonction avec N grand afin de mettre en lumière une différence notable.
3. *Une construction moins naïve : l'algorithme des différences divisées.*

- a)  *Forme de Newton des polynômes d'interpolations.* On fixe x_1, \dots, x_n , n réels tous distincts, et y_1, \dots, y_n . On va construire par récurrence le polynôme d'interpolation P passant par les points (x_i, y_i) pour $i \in \llbracket 0, n \rrbracket$.


On pose P_k , le polynôme d'interpolation de Lagrange aux points (x_i, y_i) pour $i \in \llbracket 0, k \rrbracket$. On a donc, pour tout x , $P_0(x) = y_0$. Démontrer qu'il existe un nombre $f[x_0, \dots, x_k]$ tel que

$$P_k(X) - P_{k-1}(X) = f[x_0, \dots, x_k](X - x_0)(X - x_1) \cdots (X - x_{k-1}).$$

En déduire que

$$P(X) = y_0 + \sum_{k=1}^n f[x_0, \dots, x_k](X - x_0)(X - x_1) \cdots (X - x_{k-1}).$$

On appelle cette expression la *forme de Newton du polynôme d'interpolation*.


- b)  On peut montrer que, pour tout $i \in \llbracket 0, n \rrbracket$, $f[x_i] = y_i$ et que pour tout $k \geq 1$, on a

$$f[x_0, \dots, x_k] = \frac{f[x_1, \dots, x_k] - f[x_0, \dots, x_{k-1}]}{x_k - x_0}.$$

En utilisant la relation de récurrence précédente, construire une fonction

`DifferencesDivisees(listX, listY)`

qui prend en entrée deux listes de même taille et qui renvoie la liste des coefficients $[f[x_0], f[x_0, x_1], \dots, f[x_0, x_1, \dots, x_n]]$.



- c)  En utilisant les questions précédentes, implémenter `InterLagrange2(X, listX, listY)` qui prend en entrée un vecteur X et deux listes de nombres `listX` et `listY` de même taille n et qui renvoie l'image de X par le polynôme interpolateur de Lagrange de degré au plus n passant les points (x_i, y_i) pour $x_i = \text{listX}[i]$ et $y_i = \text{listY}[i]$.


4. *Approximation de fonctions.* Dans cette partie, on va procéder à quelques approximations de fonctions par des polynômes et constater l'apparition de phénomènes qui peuvent paraître surprenant au premier abord.

On considère la fonction suivante :

$$\begin{aligned} f: [-1, 1] &\longrightarrow \mathbb{R} \\ x &\longmapsto \frac{1}{1+10x^2}, \end{aligned}$$


que l'on va approcher avec des polynômes interpolateurs de Lagrange. Pour cela, on va choisir un nombre de points de la courbe représentative de chacune de ces fonctions et tracer le polynôme de degré minimal passant par ces points.

- a) On commence par choisir des points répartis uniformément dans l'intervalle $[-5, 5]$.
- (i)  En considérant les points d'abscisses donnés par `absc = np.linspace((-5, 5), N)` et d'ordonnées `ordo = f(absc)`, tracer dans un même graphique les courbes représentatives de la fonction f et du polynôme interpolateur ainsi que les points d'interpolations. Vous ferez varier N entre 3 et 20.
- (ii)  Que constatez-vous lorsque N grandit ?
- b) Le choix de prendre des points équirépartis dans l'intervalle de définition est complètement arbitraire et non optimal. En réalité, on peut choisir les abscisses des points d'interpolations afin de réduire une certaine distance entre la fonction f et le polynôme interpolateur P .

- (i)  Reprenez les questions précédentes en prenant pour abscisses des points d'interpolations la liste suivante :

```
np.polynomial.chebyshev.Chebyshev(np.array([0]*N+[1])).roots()
```

avec N le nombre de points d'interpolation. Cette liste de nombres est la liste des racines d'un polynôme appelé *n-ième polynôme de Tchebychev*.

- (ii)  Que constatez vous lorsque vous faites varier N ?