TP 6 – Interpolation de Lagrange

Pour ce TP, il faudra charger les packages suivants :

```
from sympy import *
x=Symbol('x')
from matplotlib import pyplot
import numpy as np
```

On rappelle le théorème d'interpolation de Lagrange :

Théorème

Soit $n \in \mathbb{N}$. Soit $(x_0, x_1, \dots, x_n) \in \mathbb{R}^{n+1}$ tel que pour tout $k \neq l$, $x_k \neq x_l$. Soit $(y_0, y_1, \dots, y_n) \in \mathbb{R}^{n+1}$.
Il existe un unique polynôme $P \in \mathbb{R}_n[X]$ tel que pour tout $k \in \{0, \dots, n\}$, $P(x_k) = y_k$

On rappelle par ailleurs que la construction d'un tel polynôme se fait en deux étapes :

- On définit, pour tout $i \in \{0, \dots, n\}$, $L_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}$.
- On définit $L(x) = \sum_{i=0}^{n} y_i L_i(x)$.

Partie A - Calcul des L_i

1. Définir une fonction Python nommée **L**, prenant pour entrée un nombre i, une liste A (contenant les valeurs des x_k) et un réel x et renvoyant la valeur $L_i(x)$.
   Remarque : L'algorithme commence nécessairement par la ligne suivante :

```
def L(i,A,x)
```

2. Essayer l'algorithme en prenant quelques exemples pour i, A et x.
3. Vérifier sur des exemples que $L_i(x_i) = 1$ et que, pour $k \neq i$, $L_i(x_k) = 0$.

Partie B - Calcul du polynôme interpolateur de Lagrange

4. Définir une fonction Python nommée **Lagrange**, prenant pour entrée une liste A (contenant les valeurs des x_k), une liste B (contenant les valeurs des y_k) et un réel x et renvoyant son image $L(x)$ (où L est le polynôme interpolateur de Lagrange).
5. Tester l'algorithme et vérifier que le polynôme de $\mathbb{R}_3[X]$ vérifiant $P(0) = 1$, $P(1) = 1$, $P(3) = 2$ et $P(5) = 7$ est le polynôme suivant :
   `x**3/15 - x**2/10 + x/30 + 1`
6. Tracer le polynôme obtenu à la question précédente à l'aide du package matplotlib.

Partie C - Factorisation dans $\mathbb{Z}[X]$

Dans cette partie, l'objectif est de déterminer s'il est possible de factoriser un polynôme en utilisant uniquement des coefficients entiers.
On considère ainsi un polynôme $P \in \mathbb{Z}[X]$ de degré n. L'objectif est de déterminer $A \in \mathbb{Z}[X]$ et $B \in \mathbb{Z}[X]$ tels que $P = A \times B$.

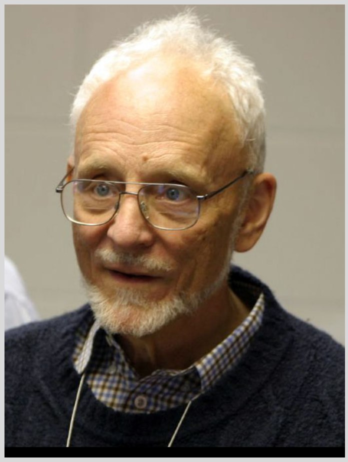
On commence par calculer les nombres $P(k)$ (pour tout $0 \leqslant k \leqslant n$). On remarque que si les polynômes A et B existent, alors pour tout $0 \leqslant k \leqslant n$, $A(k)$ divise $P(k)$. Par conséquent, il n'y a qu'un nombre fini de valeurs possibles pour $A(k)$ (les diviseurs de $P(k)$). Il suffit alors de tester toutes les valeurs de polynômes possibles pour A et déterminer si un tel polynôme A existe ou non.

Écrire un algorithme en langage Python permettant de déterminer si un polynôme à coefficients entiers peut se factoriser comme produit de deux polynômes à coefficients entiers de degré strictement inférieur. Dans le cas échéant, l'algorithme devra renvoyer la factorisation. Le tester avec les exemples suivants :

- $P_1(x) = x^5 - 4x^3 + 5x^2 + 3x + 3$
- $P_2(x) = x^5 + 2x^4 + 3x^3 + 3x^2 + 4x + 2$

Histoire

La méthode de factorisation d'un polynôme à coefficients entiers exposée ci-dessus porte le nom de factorisation de **Bernoulli-Schubert** étant donné qu'elle a été présentée dans des ouvrages peu connus de ces deux mathématiciens au XVIII$^{\text{e}}$ siècle. Elle a ensuite été retrouvée par **Léopold Kronecker** au XIX$^{\text{e}}$ siècle. Ces méthodes s'avère toutefois très vite fastidieuse et le temps de calcul augmente rapidement. Une autre technique apparue au cours du XIX$^{\text{e}}$ siècle, avec en particulier l'algorithme de **Berlekamp**, consistant à déterminer la factorisation du polynôme modulo un entier bien choisi avant de remonter à la factorisation dans $\mathbb{Z}$.



Elwyn Berlekamp