CC2 : Multiplication de polynômes

Durée de l'épreuve : 2h

Le sujet est volontairement long et il n'est pas nécessaire d'en traiter la totalité pour avoir la note de 20/20.

I. Rappels et notations

1. Python

On rappelle qu'en python, si `l` est un objet de type `list`, alors

- `len(l)` désigne le nombre d'éléments de la liste.
- `l[0]` désigne le premier élément de la liste, `l[1]` désigne le deuxième élément, etc. Le dernier élément est donc noté `l[len(l)-1]`.
- `l[-1]` désigne aussi le dernier élément de la liste, `l[-2]` l'avant dernier élément, etc. Le premier élément de la liste peut ainsi être désigné comme `l[-len(l)]`.
- `sum(l)` désigne la somme de ses éléments, c'est à dire `l[0]+l[1]+...+l[len(l)-1]`.
- `l[i:j]` désigne la sous-liste `[l[i],l[i+1],...,l[j-2],l[j-1]]` (c'est à dire qu'on commence à `l[i]` et on s'arrête juste avant `l[j]`).
  - `l[:j]` désigne la sous-liste `[l[0],l[1],...,l[j-2],l[j-1]]` (c'est à dire qu'on commence à `l[0]` en l'absence de précision)
  - en particulier `l[:-1]` contient tous les éléments de `l` sauf le dernier.
  - `l[i:]` désigne la sous-liste `[l[i],l[i+1],...,l[len(l)-2],l[len(l)-1]]` (c'est à dire qu'on va jusqu'au dernier élément de de `l` en l'absence de précision)
- Pour `n`$\in \mathbb{N}$, `l*n` désigne la liste obtenue en mettant bout à bout `n` copies de la liste `l`. Par exemple `[1]*3` est la liste `[1,1,1]`.

De plus, on rappelle que pour `n`$\in \mathbb{N}^*$, `range(n)` désigne la liste `[0,1,...,n-2,n-1]`. En revanche si `n` $\leqslant 0$, alors `range(n)` désigne la liste vide (c'est à dire `[]`).

Enfin, `a//b` désigne le quotient euclidien de `a` par `b` (si `a`$\in \mathbb{N}$ et `b`$\in \mathbb{N}^*$).

2. Polynômes

On rappelle qu'un polynôme P non nul est identifié au multiplet de ses coefficients $(c_0, c_1, \dots, c_d)$ où

$$\begin{cases} P(X) = \sum_{i=0}^{d} c_i X^i \\ c_d \neq 0. \end{cases}$$

Le nombre d s'appelle alors le degré du polynôme, noté $\deg(P)$.

En python, ces coefficients seront fournis sous la forme d'une liste `[`c_0`, `c_1`, ...,`c_d`]`.

De plus, on associe à ce polynôme une "fonction polynomiale" f_P, définie par

$$f_P : \begin{cases} \mathbb{R} \to \mathbb{R} \\ x \mapsto \sum_{i=0}^{d} c_i x^i \end{cases}$$

Quant au polynôme nul, on y associe la fonction nulle.

Par convention le degré du polynôme nul est $-\infty$, et il est représenté en python par la liste vide `[]`.

Dans les raisonnements mathématiques, on pourra utiliser la convention $c_i = 0$ lorsque $i > \deg(P)$. On prendra toutefois garde que si `l` est la liste de coefficients `[`c_0`, `c_1`, ...,`c_d`]`, alors la commande python `l[i]` renvoie un message d'erreur si `i`$> d$.

II. Questions introductives

Soient P et Q deux polynômes, et f_P et f_Q les fonctions polynomiales associées.

1. Montrer que f_P admet un développement de Taylor (à tous les ordres) au voisinage de 0. Le calculer et conclure que si les fonctions f_P et f_Q sont égales, alors les polynômes P et Q sont égaux (c'est à dire qu'ils ont les même coefficients).
2. Montrer qu'il existe un unique polynôme dont la "fonction polynomiale" est $x \mapsto f_P(x) \cdot f_Q(x)$. Exprimer les coefficients, et le degré, de ce polynôme.

III. Algorithme de produit de polynômes

On demande à trois étudiants d'écrire chacun un programme `multiplication_polynôme` pour calculer le produit de deux polynômes (éventuellement en argumentant préalablement par un raisonnement mathématique, et en définissant d'autres fonctions utiles avant de définir la fonction `multiplication_polynôme`).

Leurs réponses sont données ci-après, et pour chacun de ces programmes,

- Indiquez ce que renvoie l'ordinateur quand on calcule `multiplication_polynome([1,3],[2,8])` (c'est à dire quand on demande le produit de $1+3X$ par $2+8X$).
- Indiquer si le programme vous semble correct, ou pas.
- S'il vous semble erroné, proposez-en une version corrigée.

1. Réponse d'un premier étudiant

```
def somme(P,Q): #additionne les polynômes P et Q
    return [P[i]+Q[i] for i in range(len(Q))]

def multiplication_monome(P,Q):
    """multiplie les polynome P et Q en supposant que Q est un monome,
    c'est à dire Q=[0,0,0,...,0,c] où c est un coefficient arbitraire"""
    return [0]*(len(Q)-1)+[P[i]*Q[-1] for i in range(len(P))]

def multiplication_polynome(P,Q):
    R=[] # commencer par le polynome nul
    while len(Q)!=0:
        R=somme(multiplication_monome(P,[0]*(len(Q)-1)+[Q[-1]]),R)
        Q=Q[:-1]
    return R
```

2. Réponse d'un second étudiant

Le programme ci-dessous s'appuie sur la formule qui donne directement les coefficients c_i du polynôme $P \cdot Q$: si p_i et q_i désignent respectivement les coefficients de P et Q alors $c_k = \sum_{i=0}^{k} p_i q_{k+i}$.

```
def degre(Q):#degre d'un polynome
    return len(Q)-1
```

```
def coef(P,d): #coefficient de P de degré d
    if d<len(P): return P[d]
    else: return 0

def multiplication_polynome(P,Q):
    L=[]
    for k in range(degre(P)+degre(Q)+1):
        L=L+[sum([coef(P,i)*coef(Q,k+i) for i in range(k+1)])]
    return L
```

3. Réponse d'un troisième étudiant

Soient $k \in \mathbb{N}$, $P = A + BX^k$ et $Q = C + DX^k$, où A, B, C, et D sont des polynômes.
On remarque que $P \cdot Q = (B \cdot D)(X^{2k} - X^k) + (A + B)(C + D)X^k + A \cdot C(1 - X^k)$
On en déduit l'algorithme ci-dessous :

```
def coef(P,d):
    if d<len(P): return P[d]
    else: return 0

def comb_lin(P,Q,a=1):
    """ calcule la combinaison linéaire P+a*Q
    où P et Q sont des polynomes, et a est un nombre.
    Par défaut, a=1, et on calcule P+Q)"""
    m=max(len(P),len(Q))
    return [coef(P,i)+a*coef(Q,i) for i in range(m)]

def multiplication_polynome(P,Q):
    if P==[] or Q==[]: return []
    if len(P)==1: return [P[0]*k for k in Q]
    if len(Q)==1: return [Q[0]*k for k in P]
    k=min(len(P),len(Q))//2
    A=P[:k]
    B=P[k:]
    C=Q[:k]
    D=Q[k:]
    BD=multiplication_polynome(B,D)
    ABCD=multiplication_polynome(comb_lin(A,B),comb_lin(C,D)) #calcul de (A+B)(C+D)
    AC=multiplication_polynome(A,C)
    R1=([0]*(2*k)+BD) # c'est à dire B*D*X**(2*k)
    R2=([0]*k+BD) # calcul de B*D*X**k
    R=comb_lin(R1,R2,-1)
    ABCD=[0]*k+ABCD # c'est à dire (A+B)*(C+D)*X**k
    R=comb_lin(R,ABCD)
    AC=comb_lin(AC,[0]*k+AC,-1)# c'est à dire A*C*(1-X**k)
    return comb_lin(R,AC)
```

IV. Étude de la complexité

On cherche désormais à déterminer le temps de calcul avec chacune des méthodes. On le mesure en comptant le nombre de fois qu'un coefficients du polynôme P a été multiplié par un coefficient du polynôme Q.

1. Lorsque P et Q sont de degré $2^m - 1$ (c'est à dire qu'ils ont 2^m coefficients), le nombre de multiplications de coefficients est noté n_m.
   (a) Pour l'algorithme proposé par le premier étudiant (après éventuelle correction de votre part s'il était erroné), donner un équivalent de n_m lorsque $m \to \infty$.
   (b) Pour l'algorithme proposé par le deuxième étudiant (après éventuelle correction de votre part s'il était erroné), donner un équivalent de n_m lorsque $m \to \infty$.
   (c) Pour l'algorithme proposé par le troisième étudiant (après éventuelle correction de votre part s'il était erroné), établir une relation de récurrence satisfaite par la suite n_m. En déduire un équivalent de n_m lorsque $m \to \infty$.
2. Conclure : lequel (ou lesquels) de ces algorithmes est (ou sont) le(s) plus rapide(s) lors que m est très grand.

V. Calcul de puissance

En s'inspirant de ce qui a été fait en TD, on écrit les relations suivantes pour calculer les puissances d'un polynôme :

$$P^{2k} = P^k \cdot P^k$$
$$P^{2k+1} = P \cdot P^k \cdot P^k$$

1. En utilisant ces relations, écrire un programme qui calcule la puissance `P^n` d'un polynôme.
2. Dans le cas où $\mathtt{n} = 2^k$ et où `P` est de degré $2^m - 1$, déterminer le nombre de multiplications de coefficients nécessaires au calcul de la puissance.