HAI820I Calcul Formel — TDs et TPs

Ivan Lejeune

3 février 2026

Table des matières

1 Structures mathématiques 2
2 Arithmétique des polynomes 4
2.1 Multiplication et addition naive 4
2.2 Multiplication de Karatsuba 5
2.3 Evaluation et Interpolation 7
2.4 Racines de l'unité et FFT 7
2.5 Division euclidienne 8
2.6 Produit de matrices 8
2.7 Algèbre linéaire rapide : réduction au produit de matrices 9

1 Structures mathématiques

Exercice 1 Objectif. Identifier les propriétés d'un groupe (neutre, inverses, commutativité) à partir d'une table d'opérations symbolique.
Soit $G = \{\alpha, \beta, \gamma, \delta\}$ un ensemble muni d'une loi de composition interne $*$. La table de l'opération est donnée ci-dessous (où l'entrée à la ligne i et la colonne j correspond à $i * j$) :

$*$	α	β	γ	δ
α	α	β	γ	δ
β	β	α	δ	γ
γ	γ	δ	α	β
δ	δ	γ	β	α

1. **Identification de l'élément neutre** : Quel est l'élément neutre e de ce groupe ? Justifiez votre réponse en observant les lignes et colonnes du tableau.
2. **Recherche d'inverses** : Pour chaque élément x de G, trouver son inverse x^{-1} (l'unique élément tel que $x * x^{-1} = e$). Que remarquez-vous sur l'inverse de γ ?
3. **Commutativité** : Le groupe $(G, *)$ est-il abélien (commutatif) ?
4. **Résolution d'équation** : A l'aide de la table, résoudre l'équation suivante d'inconnue x :
$$\gamma * x = \delta$$

Solution.

1. La seule ligne (et colonne) qui ne modifie aucun élément est celle de α donc $e = \alpha$.
2. On a
$$\begin{cases} \alpha^{-1} & = \alpha \\ \beta^{-1} & = \beta \\ \gamma^{-1} & = \gamma \\ \delta^{-1} & = \delta \end{cases}$$
On remarque que γ est son propre inverse.
3. On a bien $\forall x, y \in G, x * y = y * x$ car le tableau est symétrique par rapport à la diagonale. Donc $(G, *)$ est abélien.
4. On cherche dans le tableau l'élément x tel que $\gamma * x = \delta$. On trouve $x = \beta$.

Exercice 2.
On se place dans l'anneau $(M_2(\mathbb{R}), +, \cdot)$. On considère les matrices suivantes :
$$A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \quad \text{et} \quad B = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

1. Calculer le produit $A \cdot B$ puis le produit $B \cdot A$.
2. L'anneau $M_2(\mathbb{R})$ est-il commutatif ?
3. Calculer A^2.
4. Pourquoi cela empeche-t-il $M_2(\mathbb{R})$ d'être un anneau intègre même s'il ne contient que des coefficients réels ?

Solution.

1. On a
$$A \cdot B = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$
et
$$B \cdot A = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$
2. L'anneau $M_2(\mathbb{R})$ n'est pas commutatif parce qu'on a trouvé deux éléments (A et B) de $M_2(\mathbb{R})$ tels que $A \cdot B \neq B \cdot A$.
3. On a
$$A^2 = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$
4. On vient de trouver deux éléments de $M_2(\mathbb{R})$ (A) non nuls tels que $A \cdot A = 0$. Alors l'anneau $M_2(\mathbb{R})$ n'est pas intègre.

Exercice 3.
On travaille dans l'anneau de polynomes $(\mathbb{Z}/6\mathbb{Z})[x]$. Les coefficients sont calculés modulo 6. Soient $P(x) = 2x^2 + 3$ et $Q(x) = 3x + 1$

1. **Anomalie du degré**
   (a) Quels sont les degrés de P et Q ?
   (b) Calculer le produit $P(x) \cdot Q(x)$ en réduisant les coefficients modulo 6.
   (c) Quel est le degré du polynome résultant ? La propriété $\deg(P \cdot Q) = \deg(P) + \deg(Q)$ est-elle vérifiée ? Expliquez pourquoi.
2. **Echec de la division euclidienne standard**. On considère les polynomes $F(x) = x^2 + 1$ et $G(x) = 2x$. Essayez de poser la division euclidienne de F par G.
   Indice : Pour éliminer le terme en x^2, vous devez multiplier $2x$ par un coefficient k tel que $2 \cdot k = 1 \pmod 6$. Un tel entier existe-t-il dans $\mathbb{Z}/6\mathbb{Z}$? Conclure.

Solution.

1. Sur les degrés :
   (a) P est de degré 2 et Q de degré 1.
   (b) On a
$$\begin{aligned} P \cdot Q &= \left(2x^2 + 3\right) \cdot (3x + 1) && \pmod 6 \\ &= 6x^3 + 2x^2 + 9x + 3 && \pmod 6 \\ &= 2x^2 + 3x + 3 && \pmod 6 \end{aligned}$$
   (c) Le degré du polynome résultant est 2. La propriété $\deg(P \cdot Q) = \deg(P) + \deg(Q)$ n'est pas vérifiée car l'anneau n'est pas intègre (par exemple $2 \cdot 3 = 0 \pmod 6$).
2. 2 ne possède pas d'inverse dans $\mathbb{Z}/6\mathbb{Z}$ car $2 \wedge 6 = 2$. On ne peut donc pas poser la division euclidienne de F par G ici.

2 Arithmétique des polynomes

2.1 Multiplication et addition naive

Exercice 4.
Coder dans Sagemath les opérations d'addition et de multiplication de deux polynomes qui sont donnés par la liste de leurs coefficients. Le premier élément de la liste correspondra à l'élément constant du polynome. Vous testerez vos fonctions sur des polynomes aléatoires de degrés dans $\{10, 100, 1000\}$ avec les domaines de coefficients suivants : $\mathbb{GF}(17), \mathbb{Z}, \mathbb{Z}/100\mathbb{Z}$. A chaque fois vous prendrez soin de faire des opérations impliquant aussi bien des polynomes de degrés identiques que des polynomes de degrés différents. Vous devrez comparer les résultats de vos calculs avec celui fait par les opérations natives d'addition et de multiplication de polynomes dans Sage.

Solution. On utilise le code suivant pour additioner nos deux polynomes :

```
def poly_add(A, B, R):
    n = max(len(A), len(B))
    res = [R(0)] * n
    for i in range(n):
        a = A[i] if i < len(A) else 0
        b = B[i] if i < len(B) else 0
        res[i] += a + b
    res = normalise(res, R)
    return res
```

Code 1 – Fonction d'addition de polynomes

et celui-ci pour faire la multiplication :

```
def poly_mult(A, B, R):
    res = [R(0)] * (len(A) + len(B) - 1)
    for i in range(len(A)):
        for j in range(len(B)):
            res[i + j] += A[i] * B[j]
    res = normalise(res, R)
    return res
```

Code 2 – Fonction de multiplication de polynomes

Une fonction importante qu'on a utilisé dans les programmes précédent est celle de normalisation. Elle permet de retirer les zeros en trop après les opérations (par exemple $(X+1)+(-X+1)$ donnera 2 et pas $0X+2$).

```
def normalise(L, R):
    L = list(L)
    while len(L) > 0 and L[-1] == R(0):
        L.pop()
    return L
```

Code 3 – Fonction de normalisation des resultats

Enfin avec les fonctions suivantes on peut tester que nos fonctions font bien ce qu'elles devraient :

```
def list_to_poly(L, R):
    x = R['x'].gen()
    return sum(L[i] * x^i for i in range(len(L)))

def random_poly_list(deg, R):
```

```
    return [R.random_element() for _ in range(deg + 1)]

def test_operations(R, deg1, deg2):
    A = random_poly_list(deg1, R)
    B = random_poly_list(deg2, R)

    PR = PolynomialRing(R, 'x')

    PA = list_to_poly(A, PR)
    PB = list_to_poly(B, PR)

    C_add = poly_add(A, B, R)
    C_mul = poly_mult(A, B, R)

    assert list_to_poly(C_add, PR) == PA + PB
    assert list_to_poly(C_mul, PR) == PA * PB

    print(f"OK pour degres {deg1} et {deg2} sur {R}")
```

Code 4 – Fonctions pour tester

Et alors une fonction comme celle-ci nous confirme bien que nos fonctions effectuent les bonnes opérations :

```
rings = [GF(17), ZZ, Integers(100)]
degrees = [10, 100, 1000]

for R in rings:
    # tests degres identiques
    for d in degrees:
        test_operations(R, d, d)
    # tests degres differents
    for i in range(3):
        for j in range(i + 1, 3):
            test_operations(R, degrees[i], degrees[j])
```

Code 5 – Lancement des tests — Resultats positifs

2.2 Multiplication de Karatsuba

Exercice 5.
Nous revenons ici sur l'analyse de la complexité de l'algorithme de Karatsuba pour des polynomes de $A[X]$ avec A un anneau quelconque.

1. Donner le nombre exact de multiplications dans A qu'effectue au total l'algorithme de Karatsuba pour multiplier deux polynomes de taille n à coefficients dans A.
2. Calculer le nombre d'additions dans A effectuées au premier niveau récursif par l'algorithme de Karatsuba.
3. Utiliser l'analyse de complexité de Karatsuba pour déduire une borne sur la complexité exacte en nombre d'opérations dans A (on ne souhaite plus avoir de constante c dans l'analyse).
4. Déduire, par l'utilisation de sage, une taille de polynome à partir de laquelle il serait intéressant d'utiliser l'algorithme de Karatsuba plutôt que l'algorithme naïf de multiplication de polynomes.

Solution.

Exercice 6.
Calculer les produits des polynomes dans $\mathbb{Z}[X]$ suivants en utilisant l'algorithme de Karatsuba :

1. $P = 3X + 4$ et $Q = 2X + 3$.
2. $P = X + 3$ et $Q = 3X + 2$.
3. $P = X^3 + 2X^2 + 1$ et $Q = 2X^3 + X + 2$.

Solution.

Exercice 7.
Dans un premier temps, vous allez coder l'algorithme de multiplication de polynomes de Karatsuba en faisant l'hypothèse que les polynomes d'entrée sont de taille 2^k pour un entier $k \geq 0$ quelconque. Votre fonction prendra en paramètre la liste des coefficients des deux polynomes à multiplier. Ces listes devront obligatoirement faire la meme taille. Dans un deuxieme temps, vous proposerez une amélioration du code qui permettra de prendre en parametre des polynomes de taille quelconque.

Solution. On peut coder la multiplication de Karatsuba avec des polynomes de taille une puissance de 2 comme suit :

```python
def karatsuba_list(a, b, R):
    a = [R(x) for x in a]
    b = [R(x) for x in b]

    if not a or not b:
        return []

    # cas de base
    if len(a) == 1:
        return [a[0] * bi for bi in b]

    if len(b) == 1:
        return [b[0] * ai for ai in a]

    n = max(len(a), len(b))
    m = n // 2

    a += [R(0)] * (n - len(a))
    b += [R(0)] * (n - len(b))

    a0, a1 = a[:m], a[m:]
    b0, b1 = b[:m], b[m:]

    z0 = karatsuba_list(a0, b0, R)
    z2 = karatsuba_list(a1, b1, R)

    a0a1 = [a0[i] + a1[i] for i in range(len(a1))]
    b0b1 = [b0[i] + b1[i] for i in range(len(b1))]

    z1 = karatsuba_list(a0a1, b0b1, R)

    # on met tout a la meme taille
    L = max(len(z0), len(z1), len(z2))
    z0 += [R(0)] * (L - len(z0))
    z1 += [R(0)] * (L - len(z1))
    z2 += [R(0)] * (L - len(z2))

    for i in range(L):
```

```
        z1[i] -= z0[i] + z2[i]

    # fusion des resultats
    result = [R(0)] * (2 * n)

    for i, v in enumerate(z0):
        result[i] += v

    for i, v in enumerate(z1):
        result[i + m] += v

    for i, v in enumerate(z2):
        result[i + 2*m] += v

    # Trim trailing zeros
    while result and result[-1] == R(0):
        result.pop()

    return result
```

Code 6 – Multiplication de Karatsuba avec $\deg(P) = 2^k - 1$

qo'on peut positivement tester avec le code qui suit :

```
R = GF(5)

a = (1, 2, 3)
b = (2, 1)

# on trouve la premiere puissance de 2 au dessus
n = 1 << (max(len(a), len(b)) - 1).bit_length()

# et on rajoute des zeros jusqu'a l'atteindre
a_two_pow = list(a) + [R(0)] * (n - len(a))
b_two_pow = list(b) + [R(0)] * (n - len(b))

karatsuba_list(a_two_pow, b_two_pow, R)
```

Code 7 – Test de la multiplication de Karatsuba

2.3 Evaluation et Interpolation

Exercice 8 title.

Solution.

2.4 Racines de l'unité et FFT

Exercice 9 title.

Solution.

Exercice 10 title.

Solution.

Exercice 11 title.

Solution.

Exercice 12 title.

Solution.

Exercice 13 title.

Solution.

Exercice 14 title.

Solution.

2.5 Division euclidienne

Exercice 15 title.

Solution.

Exercice 16 title.

Solution.

Exercice 17 title.

Solution.

Exercice 18 title.

Solution.

Exercice 19 title.

Solution.

Exercice 20 title.

Solution.

Exercice 21 title.

Solution.

Exercice 22 title.

Solution.

2.6 Produit de matrices

Exercice 23 title.

Solution.

2.7 Algèbre linéaire rapide : réduction au produit de matrices

Exercice 24 title.

Solution.

Exercice 25 title.

Solution.

Exercice 26 title.

Solution.

Exercice 27 title.

Solution.

Exercice 28 title.

Solution.

Exercice 29 title.

Solution.