

Examen de M.A.O. Calcul Formel Durée : 3 heures

Juin 2020
Master 1 M.F., Orsay

Problème 1

1. On a

$$X^{\frac{n-1}{2}} V_n\Big(X+\frac{1}{X}\Big) = X^{\frac{n-1}{2}} \sum_{j=0}^{(n-1)/2} a_j \Big(X+\frac{1}{X}\Big)^j = \sum_{j=0}^{(n-1)/2} \sum_{k=0}^{j} \binom{j}{k} a_j X^{\frac{n-1}{2}+2k-j}.$$

   On pose alors $i = \frac{n-1}{2} + 2k - j$. Pour chaque entier j, il existe au plus un entier k tel que $i = \frac{n-1}{2} + 2k - j$: c'est $\frac{2(i+j)-n+1}{4}$. Il en existe un si, et seulement si, $\frac{2(i+j)-n+1}{4}$ est un entier compris entre 0 et j. Le fait que ce soit un entier équivaut à ce que $i+j-\frac{n-1}{2}$ soit pair, c'est-à-dire $j \equiv \kappa_i \bmod 2$. L'encadrement $0 \le k \le j$ équivaut au système formé par les inégalités $j \ge \kappa_i$ et $j \ge -\kappa_i$, ce qui signifie $j \ge |\kappa_i|$. On a donc finalement la relation suivante, qui donne le résultat en identifiant les coefficients :

$$X^{\frac{n-1}{2}} V_n\Big(X+\frac{1}{X}\Big) = \sum_{i=0}^{n-1} \Big(\sum_{\substack{|\kappa_i| \le j \le \frac{n-1}{2} \\ j \equiv \kappa_i \bmod 2}} \binom{j}{(2(i+j)-n+1)/4} a_j \Big) X^i.$$

2. Le système précédent est un système linéaire à $\frac{n+1}{2}$ inconnues $a_0, \ldots, a_{(n-1)/2}$, avec second membre, à coefficients entiers. Il est formé par $n+1$ équations, mais celles correspondant à i et à $n-1-i$ sont les mêmes puisque κ_{n-1-i} et κ_i sont opposés (donc ont la même valeur absolue et la même parité). On peut donc se restreindre aux entiers i compris entre 0 et $(n-1)/2$, ce qui donne $(n+1)/2$ équations. En ordonnant ces équations par valeurs décroissantes de i on obtient un système triangulaire à diagonale de 1, puisque $(2(i+j)-n+1)/4 = 0$ lorsque $j = \kappa_i$. Ce système possède donc une solution et une seule avec $a_0, \ldots, a_{(n-1)/2} \in \mathbb{Z}$. On pourrait invoquer le pivot de Gauß pour résoudre ce système, mais en fait la moitié du travail est déjà faite. L'équation qui correspond à $i \in \{0, \ldots, (n-1)/2\}$ permet de calculer a_{κ_i} en fonction de a_{κ_i+1}, …, $a_{(n-1)/2}$ qu'on suppose déjà calculés (en procédant par valeurs croissantes de i : $i = 0$ permet de calculer $a_{(n-1)/2}$). Le coût du calcul de a_{κ_i} est $O(n)$ opérations arithmétiques dans $\mathbb{Z}$ (multiplications, additions, soustractions) à condition de disposer d'une table des coefficients binomiaux, qui permette de les calculer en temps constants. Sous cette hypothèse on obtient donc le polynôme V_n en $O(n^2)$ opérations arithmétiques dans $\mathbb{Z}$. Si on ne dispose pas de cette table, le

plus efficace est de la construire : on calcule tous les coefficients binomiaux $\binom{j}{k}$ pour $0 \leq j \leq (n-1)/2$ et $0 \leq k \leq j$ par la formule de Pascal : $\binom{j}{k} = \binom{j-1}{k} + \binom{j-1}{k-1}$ si $1 \leq k \leq j-1$. Cela permet de calculer chaque coefficient en fonction des précédents en utilisant seulement une addition, donc on calcule la table en $O(n^2)$ opérations arithmétiques. Finalement ce précalcul n'a donc pas d'influence sur le coût global du calcul.

3. ($\star$) On donne une première version qui fait appel au pivot de Gauß pour calculer les $a_0, \ldots, a_{(n-1)/2}$ (et qui n'est donc pas efficace comme expliqué en question précédente) [1] :

```
def vn(n):
    if n%2==0:
        return "n doit être impair!"
    A.<X>=ZZ[]
    N=(n-1)/2
    Vect=VectorSpace(QQ,N+1)
    Mat=MatrixSpace(QQ,N+1)
    V=Vect([1 for i in range(N+1)])
    M=Mat([0 for i in range((N+1)^2)])
    for i in range(N+1):
        kappa=N-i
        L=list(range(abs(kappa),N+1))
        for r in range(len(L)):
            l=L[r]
            if l%2==kappa%2:
                L[r]=binomial(l,(2*(i+l)-n+1)/4)
            else:
                L[r]=0
        M[i]=[0 for k in range(abs(kappa))]+L
    S=M.solve_right(V)
        return sum(S[i]*X**i for i in range(len(S)))
```

On donne alors la version efficace qui tire parti du fait que la matrice du système est directement triangulaire avec des 1 sur la diagonale :

```
def vnbis(n):
    if n%2==0:
        return "n doit être impair!"
    A.<X>=ZZ[]
    N=(n-1)/2
    Vect=VectorSpace(QQ,N+1)
```

1. Toutes les fonction ici et même un peu plus se trouvent dans le Notebook joint à ce pdf où elles sont en plus commentées.

```
    V=Vect([1 for i in range(N+1)])
    for i in range(1,N+1):
        kappa=N-i
        L=list(range(abs(kappa)+1,N+1))
        for r in range(len(L)):
            l=L[r]
            if l%2==kappa%2:
                L[r]=binomial(l,(2*(i+l)-n+1)/4)
            else:
                L[r]=0
        V[kappa]=1-sum(L[r]*V[r+abs(kappa)+1] for r in range(len(L)))
    return sum(V[i]*X**i for i in range(len(V)))
```

Pour finir, on donne une version qui ne fait pas non plus appel au calcul des coefficients binomiaux de `Sage` mais qui calcule en $O(n^2)$ ceux dont on a besoin grâce à la formule du triangle de Pascal :

```
def vnter(n):
    if n%2==0:
        return "n doit être impair!"
    A.<X>=ZZ[]
    N=(n-1)/2
    Vect=VectorSpace(QQ,N+1)
    V=Vect([1 for i in range(N+1)])
    Mat=MatrixSpace(QQ,N+1)
    B=Mat([1 for i in range((N+1)^2)])
    for i in range(1,N+1):
        for j in range(1,i):
            B[i,j]=B[i-1,j]+B[i-1,j-1]
    for i in range(1,N+1):
        kappa=N-i
        L=list(range(abs(kappa)+1,N+1))
        for r in range(len(L)):
            l=L[r]
            if l%2==kappa%2:
                L[r]=B[l,(2*(i+l)-n+1)/4]
            else:
                L[r]=0
        V[kappa]=1-sum(L[r]*V[r+abs(kappa)+1] for r in range(len(L)))
    return sum(V[i]*X**i for i in range(len(V)))
```

4. ($\star$) On obtient avec le code suivant

```
V11=vnter(11)
V11
```

que $V_{11}(X) = X^5 + X^4 - 4X^3 - 3X^2 + 3X + 1$. On peut vérifier que V_{11} convient bien grâce à

```
A.<X>=QQ[]
X^((11-1)/2)*V11.substitute(X=X+1/X)
```

5. Pour $i = 0$ la relation de la question 1 donne $a_{(n-1)/2} = 1$ donc V_n est unitaire de degré égal à $\frac{n-1}{2}$. Son coefficient dominant n'est donc pas multiple de p : on a $\deg V_{n,p} = \frac{n-1}{2}$.
6. En prenant $X = 1$ dans la relation (1) on obtient $V_n(2) = n$. En réduisant modulo p on en déduit immédiatement $V_{n,p}(2) = \overline{n}$.
7. Soit $x \in \mathbb{K}$ tel que $\sum_{i=0}^{p-1} x^i = 0$ et $x \neq 1$. En sommant la série géométrique on a $\frac{1-x^p}{1-x} = 0$ donc $x^p = 1$. En caractéristique p, cela implique $x = 1$ puisque $X^p - 1 = (X-1)^p$, ce qui est contradictoire. On pouvait aussi conclure en notant $q = p^d$ le cardinal de $\mathbb{K}$, avec $d = [\mathbb{K} : \mathbb{F}_p]$: comme l'ordre de x dans le groupe $\mathbb{K}^*$ divise à la fois p et le cardinal de $\mathbb{K}^*$, qui vaut $q-1$, il divise $\mathrm{pgcd}(p, q-1) = 1$ donc $x = 1$. Alternativement, une autre approche est de dire que $x = x^q = (x^p)^{p^{d-1}} = 1^{p^{d-1}} = 1$. Enfin on peut invoquer l'injectivité du morphisme de Frobenius $x \mapsto x^p$ pour dire que $x^p = 1^p$ implique $x = 1$. *Attention, on ne peut pas affirmer a priori que $x = x^p$: ceci n'est vrai que si $x \in \mathbb{F}_p$, et ici x appartient au corps $\mathbb{K}$ qui est une extension de $\mathbb{F}_p$.*
8. Il existe une extension $\mathbb{K}$ de $\mathbb{F}_p$ sur laquelle $V_{p,p}$ est scindé ; comme $V_{p,p}$ est unitaire de degré $(p-1)/2$, il suffit de démontrer que 2 est la seule racine possible de $V_{p,p}$ dans $\mathbb{K}$ pour en déduire que $V_{p,p}(X) = (X-2)^{(p-1)/2}$. En effet, soit $z \in \mathbb{K}$ une racine de P. Quitte à remplacer $\mathbb{K}$ par une extension qui la contient, on peut supposer qu'il existe $x \in K^*$ tel que $x + \frac{1}{x} = z$ (puisque ceci équivaut à $x^2 - zx + 1 = 0$). On prend alors $X = x$ dans la relation (1), préalablement réduite modulo p (c'est-à-dire vue dans $\mathbb{F}_p[X]$) : on obtient $0 = x^{(n-1)/2} V_{p,p}(z) = \sum_{i=0}^{p-1} x^i$. D'après la question précédente on a donc $x = 1$, d'où $z = 2$.
9. Comme $V_{p,p}(X)$ est unitaire, scindé, et admet 2 pour unique racine (de multiplicité $(p-1)/2$), on a $\mathrm{Res}(V_{p,p}, V_{q,p}) = (V_{q,p}(2))^{(p-1)/2} = \overline{q}^{(p-1)/2}$ d'après la question 6.
10. Supposons par l'absurde que $\mathrm{Res}(V_{a,\ell}, V_{b,\ell}) = 0$ pour un certain nombre premier ℓ. Alors il existe une extension $\mathbb{K}$ de $\mathbb{F}_\ell$ sur laquelle $V_{a,\ell}$ et $V_{b,\ell}$ possèdent une racine commune z. Comme à la question 8, quitte à étendre $\mathbb{K}$ on peut supposer qu'il existe $x \in \mathbb{K}^*$ tel que $x + \frac{1}{x} = z$, et on a $\sum_{i=0}^{a-1} x^i = \sum_{i=0}^{b-1} x^i = 0$. Si $x = 1$ alors $z = 2$ donc $V_{a,\ell}(2) = V_{b,\ell}(2) = 0$ dans $\mathbb{F}_\ell$; en utilisant la question 6 (dont la preuve montre qu'elle est valable même pour $\ell = 2$) on déduit que ℓ divise a et b, ce qui est impossible puisque a et b sont premiers entre eux. Donc on a $x \neq 1$ et en sommant la série géométrique on a $x^a = x^b = 1$: l'ordre de x dans $\mathbb{K}^*$ divise a et b, donc divise $\mathrm{pgcd}(a, b) = 1$: c'est une contradiction puisque $x \neq 1$. On a donc montré que $\mathrm{Res}(V_{a,\ell}, V_{b,\ell}) \neq 0$ pour tout nombre premier ℓ. Or $\mathrm{Res}(V_{a,\ell}, V_{b,\ell})$ est la classe modulo ℓ de $\mathrm{Res}(V_a, V_b)$, puisque V_a et V_b sont unitaires. Donc l'entier $\mathrm{Res}(V_a, V_b)$ n'est divisible par aucun nombre premier : c'est 1 ou -1.

11. On a vu à la question précédente que la classe modulo p de $\mathrm{Res}(V_p, V_q)$ est donnée par $\mathrm{Res}(V_{p,p}, V_{q,p}) = \overline{q}^{(p-1)/2}$ d'après la question 9. On a donc $\mathrm{Res}(V_p, V_q) \equiv q^{(p-1)/2} \equiv \left(\frac{q}{p}\right) \bmod p$. Comme $p \geq 3$ et que $\mathrm{Res}(V_p, V_q)$ et $\left(\frac{q}{p}\right)$ appartiennent à $\{-1, 1\}$ on en déduit que $\mathrm{Res}(V_p, V_q) = \left(\frac{q}{p}\right)$.

12. Comme $\deg V_p = \frac{p-1}{2}$ et $\deg V_q = \frac{q-1}{2}$ on a $\mathrm{Res}(V_p, V_q) = (-1)^{\frac{p-1}{2}\cdot\frac{q-1}{2}}\mathrm{Res}(V_q, V_p)$, ce qui démontre la loi de réciprocité quadratique en utilisant la question précédente.

13. Déterminer V_p et V_q coûte respectivement $O(p^2)$ et $O(q^2)$. Le calcul du résultant de V_p et V_q, mené à l'aide de la méthode efficace vue en cours, coûte $O(pq)$ opérations arithmétiques dans $\mathbb{Z}$, ce qui donne une complexité totale de $O(\max\{p^2, q^2\})$. En fait on peut accélérer ce calcul en choisissant un petit nombre premier $\ell \geq 3$ et en calculant $V_{p,\ell}$ et $V_{q,\ell}$ ainsi que leur résultant, ce qui coûte $O(\max\{p^2, q^2\})$ opérations arithmétiques dans $\mathbb{F}_\ell$ (et évidemment une opération dans $\mathbb{F}_\ell$ est plus rapide qu'une opération dans $\mathbb{Z}$ qui peut impliquer de grands entiers, surtout si $\ell = 3$ par exemple). En effet $\left(\frac{q}{p}\right) \in \{-1, 1\}$ est déterminé par sa classe modulo ℓ (comme à la question 11). Ceci dit, pour des applications éventuelles en cryptographie avec des nombres premiers à plusieurs milliers de chiffres, ce coût est beaucoup trop élevé... L'autre approche est beaucoup plus efficace : on calcule par exponentiation rapide la puissance $\frac{p-1}{2}$-ième de l'image de q dans $\mathbb{F}_p$, ce qui coûte $O(\log p)$ multiplications dans $\mathbb{F}_p$. *Précisément, on réduit q modulo p (ce qui consiste à faire une division euclidienne), puis chaque multiplication dans $\mathbb{F}_p$ consiste à multiplier deux entiers compris entre 0 et $p-1$, puis à prendre le reste dans la division euclidienne du produit par p. Chaque multiplication dans $\mathbb{F}_p$ correspond donc à deux opérations arithmétiques dans $\mathbb{Z}$, mais l'intérêt est qu'on ne manipule que dans entiers compris entre 0 et $p-1$ (voire $(p-1)^2$ quand on fait le produit). Ceci est beaucoup plus efficace que de calculer $q^{(p-1)/2}$ puis de réduire modulo p : en effet cette dernière méthode oblige à travailler avec des entiers extrêmement grands, ce qui est nettement plus long.*

14. ($\star$) On commence par appliquer l'algorithme issu de la question [2] 11 :

```
def legendre1(p,q):
    if not(p.is_prime()) or not(q.is_prime()) or p%2==0 or q%2==0:
        return "p et q doivent être des nombres premiers pairs!"
    Vp=vnter(p)
    Vq=vnter(q)
    return Vp.resultant(Vq)
```

On donne également la même version mais en faisant les calculs modulo 3 :

```
def legendre1bis(p,q):
    if not(p.is_prime()) or not(q.is_prime()) or p%2==0 or q%2==0:
```

2. Dans le Notebook, on compare même les différents temps d'exécution des diverses versions de calcul de V_n et on recode l'algorithme efficace de calcul du résultant. On ne donne dans ce corrigé que la plus efficace d'entre elle, `vnter`.

```
        return "p et q doivent être des nombres premiers pairs!"
    l=3
    Vp=vnter(p)
    Vq=vnter(q)
    A.<X>=GF(p)[]
    R=résultant(A(Vp),A(Vq))
    if R==1:
        return R
    else:
        return -1
```

Enfin, on a la version à base d'exponentiation rapide[3] et utilisant que $\left(\frac{q}{p}\right) \equiv q^{\frac{p-1}{2}} \bmod p$:

```
def exponentiation_rapide(a,n):
    r=1
    N=n
    A=a
    while(N>0):
        if N%2==1:
            r=r*A
        A=A*A
        N=N//2
    return r
def legendre2(p,q):
    if not(p.is_prime()) or not(q.is_prime()) or p%2==0 or q%2==0:
        return "p et q doivent être des nombres premiers pairs!"
    k=GF(p)
    R=exponentiation_rapide(k(q),(p-1)//2)
    if R==1:
        return R
    else:
        return -1
```

On peut alors tester nos fonctions et les comparer à la fonction déjà implémentée dans Sage :

```
legendre1(3,5),legendre1bis(3,5),legendre2(3,5),legendre_symbol(3,5)
```

On compare alors les temps d'exécution sur $p = 577$ et $q = 1237$ avec le code suivant :

```
import time
t=time.time()
legendre1(577,1237)
```

3. Il est conseillé ici d'utiliser un version itérative sinon la complexité en mémoire explose pour des nombres premiers trop grands !

```
time.time()-t
t=time.time()
legendre1bis(577,1237)
time.time()-t
t=time.time()
legendre2(577,1237)
time.time()-t
```

On constate bien que la méthode par exponentiation rapide est (beaucoup) plus rapide (`0.0002` s contre respectivement `2.82` s et `2.80` s). Noter que la version avec pivot de Gauß s'exécute elle, en `9.54` s.

Problème 2

1. Le point crucial, vu en cours, est que pour tout entier k le polynôme $X^{q^k} - X$ est le produit des polynômes irréductibles unitaires de $\mathbb{F}_q[X]$ de degré divisant k. Dans un premier temps, supposons P est irréductible de degré n ; alors pour tout $k < n$, P n'est associé à aucun des facteurs irréductibles de $X^{q^k} - X$, puisque $n = \deg P$ ne divise pas k. Donc on a $\text{pgcd}(P(X), X^{q^k} - X) = 1$. Réciproquement, si P est réductible de degré n alors il possède au moins un facteur irréductible de degré inférieur ou égal à $n/2$; en notant k le degré d'un tel facteur, on a $\text{pgcd}(P(X), X^{q^k} - X) \neq 1$. Cet algorithme est implémenté dans sa version naïve et dans sa version efficace à la fin du Notebook qui accompagne ce corrigé.
2. A priori le coût du calcul de $\text{pgcd}(P(X), X^{q^k} - X)$ par l'algorithme d'Euclide est $O(nq^k)$ opérations arithmétiques dans $\mathbb{F}_q$, pour k allant de 1 à la partie entière de $n/2$ cela donne un coût en $O(nq^{\lfloor n/2 \rfloor})$ puisque $\sum_{k=1}^{\lfloor n/2 \rfloor} q^k = O(q^{\lfloor n/2 \rfloor})$. Pour $q = 19$ et $n = 40$ cela donne environ $19^{20} > 10^{25}$ opérations ce qui n'est pas raisonnable.
3. La seule étape coûteuse de l'algorithme d'Euclide, dans l'estimation précédente, est la première puisque $X^{q^k} - X$ a un degré qui peut être beaucoup plus grand que celui de P. On peut en fait modifier cette étape, qui consiste à calculer le reste dans la division euclidienne de $X^{q^k} - X$ par P. En effet il suffit de calculer l'image de X^{q^k} dans le quotient $\mathbb{F}_q[X]/(P)$, notée W_k, pour k allant de 1 à la partie entière de $n/2$. En posant $W_0 = 1$, pour calculer W_k il suffit d'élever W_{k-1} à la puissance q, ce qui peut se faire par exponentiation rapide dans $\mathbb{F}_q[X]/(P)$. Cela coûte $O(\log q)$ multiplications dans $\mathbb{F}_q[X]/(P)$; comme on l'a vu en cours, chacune d'entre elles consiste à faire un produit puis une division euclidienne entre polynômes de degrés $O(n)$, donc coûte $O(n^2)$ opérations arithmétiques dans $\mathbb{F}_q$. On peut donc calculer W_k à partir de W_{k-1} en $O(n^2 \log q)$ opérations arithmétiques dans $\mathbb{F}_q$. Quand k varie de 1 à la partie entière de $n/2$, le coût est donc de $O(n^3 \log q)$ opérations arithmétiques dans $\mathbb{F}_q$. La suite de l'algorithme d'Euclide consiste à calculer le pgcd de deux polynômes de degré au plus n à coefficients dans $\mathbb{F}_q$, ce qui ne coûte que $O(n^2)$ opérations pour chaque valeur de k, donc $O(n^3)$ au total. Finalement le coût global est donc en $O(n^3 \log q)$ opérations

arithmétiques dans $\mathbb{F}_q$: pour $q = 19$ et $n = 40$ cela représente moins d'un million d'opérations.

4. Soient $\mathbb{K}$ une extension de $\mathbb{F}_q$ de degré n, et x une racine de P dans $\mathbb{K}$ telle que x soit un générateur du groupe cyclique $\mathbb{K}^*$. Notons P_0 le polynôme minimal de x sur $\mathbb{F}_q$, d son degré, et $x_1 = x$, x_2, …, x_d les racines de P_0 dans $\mathbb{K}$. Comme P_0 est irréductible sur $\mathbb{F}_q$, on sait d'après le cours que x_1, …, x_d sont deux à deux à distincts et que (quitte à permuter les x_i) on a $x_i = x_1^{q^{i-1}}$ pour tout $i \in \{1, \dots, d\}$, et $x_1^{q^d} = x_1$. Cette dernière relation donne $x_1^{q^d-1} = 1$; comme $x_1 = x$ est un générateur de $\mathbb{K}^*$, il est d'ordre $q^n - 1$. On en déduit que $q^n - 1$ divise $q^d - 1$. Comme $d \leq n$, on a donc $d = n$: $P = P_0$ est irréductible. De plus pour tout entier $i \in \{1, \dots, d\}$ on a $\text{pgcd}(q^{i-1}, q^n - 1) = 1$ donc $x_i = x_1^{q^{i-1}}$ est aussi d'ordre $q^n - 1$: toutes les racines de P sont des générateurs de $\mathbb{K}^*$. Enfin si on prend une autre extension $\mathbb{K}'$ de $\mathbb{F}_q$, de même degré n que $\mathbb{K}$, il existe un $\mathbb{F}_q$-isomorphisme de $\mathbb{K}$ dans $\mathbb{K}'$ qui envoie les racines de P dans $\mathbb{K}$ sur ses racines dans $\mathbb{K}'$: on en déduit que la conclusion est vraie également dans $\mathbb{K}'$.

5. ($\star$) D'après la question 4, il suffit de vérifier que P est irréductible et s'il l'est que la racine privilégiée x de P dans $\mathbb{F}_2[X]/(P) = \mathbb{F}_{16}$ engendre $\mathbb{F}_{16}^\times$. On commence donc par définir $\mathbb{F}_2$, $\mathbb{F}_2[X]$ ainsi que P à l'aide du code suivant :

```
k=GF(2)
A.<x>=k[]
P=x^4+x+1
```

La commande `P.is_irreducible()` (ou l'algorithme des questions précédentes) permet de vérifier que P est irréductible sur $\mathbb{F}_2[X]$. On définit alors $\mathbb{F}_2[X]/(P) = \mathbb{F}_{16}$ en notant u la racine privilégiée de P grâce à

```
K.<u>=GF(2**4,modulus=P)
```

On a alors que $\text{card}\left(\mathbb{F}_{16}^\times\right) = 15 = 3 \times 5$ si bien que u engendre $\mathbb{F}_{16}^\times$ si, et seulement si, $u^3 \neq 1$ et $u^5 \neq 1$, ce que l'on vérifie avec `Sage` grâce à la commande

```
u^((2**4-1)/3),u^((2**4-1)/5)
```

En conclusion, on a trouvé une extension $\mathbb{K}$ de degré 4 sur $\mathbb{F}_2$ et une racine de P qui engendre $\mathbb{K}^\times$, ce qui établit que $P \in \mathcal{E}_{2,4}$.

6. ($\star$) Les commandes (ou l'algorithme des questions 1 à 3)

```
Q=x^4+x^2+x+1
Q.is_irreducible()
```

fournit que $X^4 + X^2 + X + 1$ n'est pas irréductible sur $\mathbb{F}_2[X]$. D'après la question 4, on peut donc en conclure qu'il n'appartient pas à $\mathcal{E}_{2,4}$. On pouvait plus simplement remarquer que 1 est racine évidente.

7. ($\star$) On exploite ici le fait qu'un polynôme de degré 2 est irréductible sur $\mathbb{F}_4$ si, et seulement si, il n'a pas de racines dans $\mathbb{F}_4$.

```
A.<x>=GF(2)[]
P=x^2+x+1
k.<u>=GF(2**2,modulus=P)
def irred_degre_2(k):
    A.<x>=k[]
    L=[]
    for i in k:
        for j in k:
                P=x^2+i*x+j
                if not(0 in [P(r) for r in k]):
                    L.append(P)
    return L
L=irred_degre_2(k)
L,len(L)
```

On obtient 6 polynômes unitaires irréductibles qui sont donnés par

$$\begin{cases} X^2+uX+u, \\ X^2+uX+1, \\ X^2+(u+1)X+u+1, \\ X^2+(u+1)X+1, \\ X^2+X+u, \\ X^2+X+u+1 \end{cases}$$

si $\mathbb{F}_4 = \mathbb{F}_2[X]/(X^2+X+1)$ et que l'on note u la racine privilégiée de X^2+X+1. On pouvait aussi factoriser $X^{16}-X$ sur $\mathbf{F}_4[X]$ et ne garder que les polynômes de degré 2, mais la complexité est alors plus mauvaise.

8. ($\star$) Il suffit comme en question 5 de parcourir la liste obtenue en question 7 et de vérifier pour chaque polynôme P irréductible unitaire de cette liste si la racine privilégiée de P dans $\mathbb{F}_4[X]/(P) = \mathbb{F}_{16}$ engendre $\mathbb{F}_{16}^\times$. Puisque $\text{card}\left(\mathbb{F}_{16}^\times\right) = 15 = 3\times 5$, il suffit de vérifier que le cube et la puissance cinquième de cette racine privilégiée ne sont pas égaux à 1. Le code suivant implémente cet algorithme :

```
A.<x>=GF(2)[]
P=x^2+x+1
k.<u>=GF(2**2,modulus=P)
def e42():
    R.<x>=k[]
    L=irred_degre_2(k)
    S=[]
    for P in L:
        K.<v>=k.extension(P)
        n=K.cardinality()
        F=list(factor(n-1))
```

```
            F=[F[i][0] for i in range(len(F))]
            cond=True
            for p in F:
                if v^((n-1)//p)==1:
                    cond=False
            if cond:
                S.append(P)
    return S
S=e42()
S,len(S)
```

On obtient 4 polynômes dans $\mathcal{E}_{4,2}$, à savoir

$$X^2+uX+u, \quad X^2+(u+1)X+u+1, \quad X^2+X+u, \quad X^2+X+u+1.$$

9. Soit $\mathbb{K}$ une extension de $\mathbb{F}_q$ de degré n. Notons φ l'indicatrice d'Euler. Le groupe $\mathbb{K}^*$ est cyclique d'ordre q^n-1, donc il possède $\varphi(q^n-1)$ générateurs. Le morphisme de Frobenius relatif à $\mathbb{F}_q$ définit une action de $\mathbb{Z}$ sur $\mathbb{K}^*$, en posant $k\cdot y = y^{q^k}$ pour $k\in\mathbb{N}$ et $y\in\mathbb{K}^*$. D'après la question 4, les $\varphi(q^n-1)$ générateurs de $\mathbb{K}^*$ consistuent une réunion d'orbites pour cette action ; chacune de ces orbites est de cardinal n, et correspond à un polynôme $P\in\mathcal{E}_{q,n}$. Précisément, on a une bijection entre $\mathcal{E}_{q,n}$ et l'ensemble des orbites de cardinal n. Donc le cardinal de $\mathcal{E}_{q,n}$ est $\frac{1}{n}\varphi(q^n-1)$. Pour $q=4$ et $n=2$ on a $\varphi(15)=(3-1)\cdot(5-1)=8$ donc $\operatorname{Card}\mathcal{E}_{4,2}=4$ ce qui est cohérent avec le résultat obtenu à la question précédente.
10. Soit $P\in\mathbb{F}_q[X]$ unitaire de degré n. Supposons qu'on sait factoriser l'entier q^n-1 ; notons $p_1,\ldots,p_r$ ses facteurs premiers (deux à deux distincts) et posons $k_j=\frac{q^n-1}{p_j}$ pour tout $j\in\{1,\ldots,r\}$. Notons aussi $k_0=q^n-1$ et considérons l'algorithme suivant : pour tout $j\in\{0,\ldots,r\}$ on calcule $\operatorname{pgcd}(P(X),X^{k_j}-1)$. Montrons que $P\in\mathcal{E}_{q,n}$ si, et seulement si, ce pgcd vaut 1 pour tout $j\in\{1,\ldots,r\}$ mais pas lorsque $j=0$. Pour cela notons $\mathbb{K}$ une extension de $\mathbb{F}_q$ de degré n. Si $P\in\mathcal{E}_{q,n}$ alors il est scindé sur $\mathbb{K}$, et ses racines sont des générateurs de $\mathbb{K}^*$ donc des racines de $X^{k_j}-1$ pour $j=0$ mais pour aucun $j\in\{1,\ldots,r\}$. Donc $\operatorname{pgcd}(P(X),X^{k_j}-1)$ est non constant pour $j=0$ (et c'est en fait P), et vaut 1 pour tout $j\in\{1,\ldots,r\}$ (en effet le pgcd peut être calculé, au choix, dans $\mathbb{F}_q[X]$ ou dans $\mathbb{K}[X]$). Réciproquement, si $P\notin\mathcal{E}_{q,n}$, ce peut être pour deux raisons. Ou bien P ne possède aucune racine non nulle dans $\mathbb{K}$, et alors $\operatorname{pgcd}(P(X),X^{k_0}-1)=1$. Ou bien il possède une racine dans $\mathbb{K}^*$ qui n'est pas un générateur de $\mathbb{K}^*$. Notons alors x une telle racine, et ω son ordre : c'est un diviseur de q^n-1, distinct de q^n-1. Donc $\frac{q^n-1}{\omega}$ est un entier strictement plus grand que 1, qui divise q^n-1 : il existe $j\in\{1,\ldots,r\}$ tel que p_j divise $\frac{q^n-1}{\omega}$, d'où ω divise k_j et $x^{k_j}=1$. Donc les polynômes P et $X^{k_j}-1$ ont une racine commune x dans $\mathbb{K}$: leur pgcd est non constant. Cela termine la preuve que l'algorithme est correct. Pour chaque valeur de j, le coût du calcul de $\operatorname{pgcd}(P(X),X^{k_j}-1)$ est en $O(n^3\log q)$ opérations arithmétiques dans $\mathbb{F}_q$ (comme à la question 3), puisque $k_j\leq q^n$. Le coût

de cet algorithme est donc en $O((r+1)n^3 \log q)$ opérations, où r est le nombre de facteurs premiers (deux à deux distincts) de l'entier q^n-1. On peut être plus explicite : on a $q^n - 1 \geq p_1 \dots p_r \geq 2^r$ donc $r = O(n \log q)$ (ce qu'on pourrait raffiner en utilisant le théorème des nombres premiers, puisqu'en fait p_j est minoré par le j-ième nombre premier si on ordonne les p_j par ordre croissant), ce qui donne finalement un coût en $O(n^4(\log q)^2)$ opérations arithmétiques dans $\mathbb{F}_q$.

11. ($\star$) On implémente l'algorithme précédent en calculant les pgcd de façon naïve dans un premier temps :

```
def test_naif(P,K):
    A.<x>=k[]
    P=A(P)
    n=P.degree()
    if P[P.degree()]!=1:
        return "Le polynôme doit être unitaire!"
    card=K.cardinality()^n-1
    F=list(factor(card))
    F=[F[i][0] for i in range(len(F))]
    if gcd(P,x**card-1)==1:
        return False
    cond=True
    j=1
    while (j<=len(F)) and cond:
           cond=cond and gcd(P,x**(card//F[j-1])-1)==1
           j=j+1
    return cond
```

et la version efficace à base d'exponentiation rapide comme expliqué en question précédente :

```
def test_efficace(P,K):
    A.<x>=K[]
    P=A(P)
    n=P.degree()
    if P[P.degree()]!=1:
        return "Le polynôme doit être unitaire!"
    card=K.cardinality()**n-1
    F=list(factor(card))
    F=[F[i][0] for i in range(len(F))]
    B.<u>=A.quotient(P)
    G=exponentiation_rapide(B(x),card).lift()
    if gcd(P,G-1)==1:
        return False
    cond=True
```

```
    j=1
    while (j<=len(F)) and cond:
        G=exponentiation_rapide(B(x),card//F[j-1]).lift()
        cond=cond and gcd(P,G-1)==1
        j=j+1
    return cond
```

On peut alors comparer les temps d'exécution et constater que la version efficace est vraiment plus efficace [4] ! On peut alors tester nos fonctions sur les exemples précédents et retrouver (heureusement !) les résultats des questions 5, 6 et 8 avec les commandes

```
k=GF(2)
A.<x>=k[]
P=x^4+x+1
test_efficace(P,k),test_naif(P,k)
P=x^4+x^2+x+1
test_efficace(P,k),test_naif(P,k)
P=x^2+x+1
k.<u>=GF(2**2,modulus=P)
for P in S:
    print(test_naif(P,k),test_efficace(P,k))
```

On donne aussi une implémentation dans le Notebook d'une nouvelle fonction permettant de lister tous les polynômes de $\mathcal{E}_{4,2}$ basée sur ce test.

4. Je vous renvoie pour cela au Notebook !