Corrigé du partiel de M.A.O. Calcul Formel

Lundi 16 avril 2018
Master 1 M.F.A., Orsay

1. Pour calculer u_{n+d} on effectue d multiplications et $d-1$ additions, soit $O(d)$ opérations arithmétiques dans $\mathbb{K}$.

2. Pour chaque entier i compris entre d et N on effectue $O(d)$ opérations d'après la question 1. Au total, cela fait donc $O(d(N-d))$ opérations arithmétiques dans $\mathbb{K}$, soit $O(dN)$ (ce qui revient au même car on suppose N/d très grand).

3. ($\star$)

   Dans cette question de Sage comme dans les suivantes, il ne faut surtout pas calculer les entiers u_n car ils deviennent énormes. Il faut immédiatement considérer l'image de cette suite dans $\mathbb{Z}/13\mathbb{Z}$, et calculer systématiquement dans ce corps fini. Le programme suivant donne la réponse : 17.

```
d = 3
K = GF(13)
a = (2, 3, -5)
u = (0, 2, 3)

def u_list(N):
        aa = vector(K, a)
        res = [K(z) for z in u]
        while len(res) <= N:
            res.append(aa.dot_product(vector(res[-d:])))
        return(res)

def count_zeros(N):
        u = u_list(N)
        return(u[:N+1].count(K.zero()))

count_zeros(100)
```

4. On raisonne par récurrence sur $n-i$; si $n-i=0$ le résultat est trivial. Pour déduire la formule avec n de celle avec $n-1$, il suffit de constater que la relation de récurrence (1) s'écrit $(u_n,\dots,u_{n+d-1}) = (u_{n-1},\dots,u_{n+d-2})M$ par choix de la matrice compagnon M.

5. Il suffit de multiplier la relation précédente à droite par le vecteur colonne comportant un 1 en première position, et des zéros partout ailleurs.

6. On calcule M^N par exponentiation rapide dans $M_d(\mathbb{K})$, ce qui coûte $O(\log N)$ multiplications dans $M_d(\mathbb{K})$. Chacune de ces multiplications coûte $O(d^3)$ opérations arithmétiques dans $\mathbb{K}$, car pour calculer chacun des d^2 coefficients de la matrice produit il faut faire $O(d)$ opérations arithmétiques dans $\mathbb{K}$. Finalement, le calcul de M^N (qui donne instantanément V_N) coûte $O(d^3 \log N)$ opérations arithmétiques dans $\mathbb{K}$. Pour en déduire u_N il suffit d'appliquer la question 5 : le coût de cette étape, $O(d)$, est négligeable devant le coût du calcul de V_N. Finalement, calculer u_N par cette méthode coûte donc $O(d^3 \log N)$ opérations arithmétiques dans $\mathbb{K}$. Cet algorithme est donc très efficace si d est petit, ou éventuellement de l'ordre de $\log N$; mais si d est de l'ordre de $\sqrt{N}$ par exemple alors on va voir dans la suite qu'on peut faire mieux.

7. ($\star$) Le programme suivant renvoie la réponse : 8.

```
def question_7(N):
        # construction de la matrice compagnon
        M = matrix(K, 3)
        for i in xrange(d-1):
            M[i+1,i] = 1
        M[:,-1] = vector(K, a)
        # utilisation
        uu = vector(K, u)
        return (uu * M^N)[0]

question_7(200)
```

8. Par construction on a $P(x) = 0$. En multipliant par x^n, on en déduit que la suite (u_n) définie par $u_n = x^n$ pour tout $n \geq 0$ vérifie la relation (1). Stricto sensu on n'est pas dans le contexte du préambule, car les termes de cette suite appartiennent à $\mathbb{K}[X]/(P)$ qui n'est pas forcément un corps puisqu'on ne suppose pas P irréductible. Cependant, les résultats démontrés précédemment restent valables dans ce cadre, et la question 5 (avec $i = 0$) permet de conclure directement.

9. Deux éléments r et s de $\mathbb{K}[X]/(P)$ sont représentés respectivement par deux polynômes R et S, à coefficients dans $\mathbb{K}$, de degrés inférieurs ou égaux à $d - 1$. Pour calculer le produit rs, on calcule le reste dans la division euclidienne de RS par P. Le calcul de RS coûte $O(d^2)$ opérations arithmétiques dans $\mathbb{K}$ par la méthode naïve, et celui du reste aussi. Donc le calcul de rs coûte $O(d^2)$ opérations arithmétiques dans $\mathbb{K}$ (et multiplier R et S par l'algorithme de Karatsuba ne permettrait pas d'améliorer significativement ce coût).

10. On calcule x^N par exponentiation rapide dans l'anneau $\mathbb{K}[X]/(P)$. Le coût est cette fois de $O(\log N)$ multiplications dans $\mathbb{K}[X]/(P)$, ce qui représente $O(d^2 \log N)$

opérations arithmétiques dans $\mathbb{K}$ d'après la question précédente. La question 8 montre que connaître x^N revient à connaître V_N. La question 6 avec $i = 0$ fournit alors u_N moyennant $O(d)$ opérations. Finalement, le coût de cet algorithme est donc $O(d^2 \log N)$ opérations arithmétiques dans $\mathbb{K}$.

11. ($\star$)

```
def question_11(N):
        aa = vector(K, a)
        uu = vector(K, u)
        R.<X> = K[]
        P = X^len(aa) - R(list(aa))
        S.<x> = R.quotient(P)
        return uu.dot_product(vector((x^N).list()))
```

12. La formule habituelle du produit de deux polynômes donne

$$Q_n = \Big(\sum_{j=0}^{2d-2} u_{2d-2-j} X^j\Big)\Big(\sum_{k=0}^{d-1} v_k^{(n)} X^k\Big) = \sum_{\ell=0}^{3d-3} q_\ell^{(n)} X^\ell$$

avec

$$q_\ell^{(n)} = \sum_{\substack{j+k=\ell \\ 0\le j\le 2d-2 \\ 0\le k\le d-1}} u_{2d-2-j} v_k^{(n)}.$$

Pour ℓ compris au sens large entre $d-1$ et $2d-2$, cette somme est indexée par k allant de 0 à $d-1$ puisqu'alors $j = \ell - k$ est automatiquement compris entre 0 et $2d-2$. Pour ces valeurs de ℓ on a donc

$$q_\ell^{(n)} = \sum_{k=0}^{d-1} u_{2d-2-\ell+k} v_k^{(n)}.$$

On a donc $q_{2d-2-i}^{(n)} = \sum_{k=0}^{d-1} u_{i+k} v_k^{(n)}$ pour tout $i \in \{0, \dots, d-1\}$. D'après la question 5 (dans laquelle on remplace n par $n+i$), il vient immédiatement $q_{2d-2-i}^{(n)} = u_{n+i}$.

13. L'algorithme consiste à calculer les termes u_d, u_{d+1}, …, u_N par blocs de d termes consécutifs. Quitte à augmenter N légèrement, on peut supposer que c'est un multiple de d. On commence par calculer u_d, u_{d+1}, …, u_{2d-2}, ce qui par hypothèse ne coûte que $O(d \log d)$ opérations arithmétiques dans $\mathbb{K}$. Ensuite, pour k allant de 1 à N/d, on calcule x^{kd} en effectuant le produit dans $\mathbb{K}[X]/(P)$ de $x^d = a_0 + a_1 x + \ldots + a_{d-1} x^{d-1}$ par $x^{(k-1)d}$ (qu'on a calculé à l'étape précédente de la boucle) ; la question 8 montre que connaître x^{kd} revient à connaître V_{kd}. Cela permet alors de calculer $Q_{kd}(X)$ et, en utilisant la question 12, d'en déduire le bloc de d termes consécutifs u_{kd}, u_{kd+1}, …, $u_{kd+(d-1)}$. A chaque passage dans la boucle on effectue un produit dans $\mathbb{K}[X]/(P)$

et un produit de deux polynômes de degré $\leq 2d$ dans $\mathbb{K}[X]$; par hypothèse cela ne coûte que $O(d \log d)$ opérations arithmétiques dans $\mathbb{K}$. Pour calculer les termes de la suite jusqu'à u_N il suffit de N/d passages dans la boucle. Le coût total de ce calcul est donc $O(N \log d)$ opérations arithmétiques dans $\mathbb{K}$.

14. Par rapport à la question 2, on est passé de $O(Nd)$ à $O(N \log d)$. Si d est petit (par exemple dans le cas de la suite de Fibonacci) on n'a presque rien gagné. En revanche pour d grand (par exemple proche de $\sqrt{N}$), le gain est substantiel. On ne peut pas espérer beaucoup mieux car il semble difficile d'imaginer un algorithme qui utilise moins d'une opération arithmétique par terme qu'il renvoie : passer en-dessous de $O(N)$ ne semble pas réaliste.

15. ($\star$)

```
def fast_u(N):
        N = Integer(N)
        q, r = N.quo_rem(d)
        aa = vector(K, a)
        R.<X> = K[]
        uu = R(u_list(2*d-2)[::-1])
        P = X^d - R(list(aa))
        S.<x> = R.quotient(P)
        res = []
        xd = x^d
        xkd = S.one()
        for k in range(q+1):
            Vkd = xkd.list()
            Qkd = uu * R(Vkd)
            res[:] = [Qkd[2*d-2-j] for j in range(d)]
            xkd *= xd
        return(res[r])
```

16. Notons Δ l'application linéaire qui à toute suite $(u_n)_{n\geq 0}$ associe la suite $(v_n)_{n\geq 0}$ définie par $v_n = u_{n+1} - u_n$ pour tout $n \geq 0$. Si $u_n = P(\alpha n + \beta)$ et $(v_n) = \Delta((u_n))$, alors on a pour tout n :

$$v_n = P(\alpha(n+1) + \beta) - P(\alpha n + \beta) = Q(\alpha n + \beta)$$

où $Q(X) = P(X + \alpha) - P(X)$. En notant $\varphi : \mathbb{K}[X] \to \mathbb{K}[X]$ l'application qui à P associe $P(X + \alpha) - P(X)$, on a $\deg \varphi(P) = \deg P - 1$ pour tout P non nul (puisque $\alpha \neq 0$), si bien que $\varphi^{\deg P+1} P = 0$. Cela montre que $\Delta^{\deg P+1}((u_n)) = 0$ lorsque $u_n = P(\alpha n + \beta)$ avec $P \neq 0$.

Notons maintenant τ le décalage, qui à toute suite $(u_n)_{n\geq 0}$ associe la suite $(v_n)_{n\geq 0}$ définie par $v_n = u_{n+1}$; on a alors $\Delta = \tau - \mathrm{Id}$. La formule du binôme de Newton

donne, pour $u_n = P(\alpha n + \beta)$ avec $P \neq 0$ de degré $d-1$:

$$\sum_{i=0}^{d} (-1)^{d-i} \binom{d}{i} \tau^i((u_n)) = 0.$$

C'est exactement la relation de récurrence (1) avec $a_i = (-1)^{d+1-i}\binom{d}{i}$ pour tout $i \in \{0, \dots, d-1\}$ car cela s'écrit

$$u_{n+d} = -\sum_{i=0}^{d-1} (-1)^{d-i} \binom{d}{i} u_{n+i}.$$

Notamment cette relation ne dépend pas de P, mais seulement de son degré (et même plus précisément d'un majorant de son degré).