UNIVERSITÉ DE BORDEAUX

Licence CPBx

UE : J1CP100I – Informatique
durée : 1h30

Devoir Surveillé du 9 janvier 2014

Les documents sont interdits. Vous pouvez utiliser toutes les fonctions du langage python.
Les questions accompagnées d'un ♠ sont difficiles. Nous vous conseillons fortement de les traiter en dernier.
Les questions accompagnés d'un ⊗ sont hors programme pour un élève de première année !
Le barême est de 2 points par question. La note finale est le min entre 20 et le total de vos points.

L'objectif de cette partie est de proposer un algorithme pour calculer rapidement la somme : $\sum_{k=0}^{n} k^d$, où d est un entier positif ou nul en utilisant la formule :

$$\sum_{k=0}^{n} k^d = \left(\frac{\partial^d}{\partial x^d} \frac{e^{(n+1)x} - 1}{e^x - 1} \right) (0).$$

où $\frac{\partial^d}{\partial x^d} f$ est la dérivée $d^{\text{ième}}$ de f.

1. Écrivez une fonction $somme(n, d)$, non optimisée, qui prend en paramètres deux entiers n et d et qui renvoie le résultat de la somme $\sum_{k=0}^{n} k^d$.

Pour pouvoir améliorer ce calcul, nous allons avoir besoin de manipuler des polynômes. Pour cela, nous allons coder les polynômes $P = a_0 + a_1.x + a_2.x^2 + \cdots + a_n.x^n$ par leurs listes de coefficients $[a_0, a_1, a_2, \ldots, a_n]$.

Par exemple, le polynôme $P = 1 + 3x^2 - 5x^4$ est codé par $[1, 0, 3, 0, -5]$.

2. Écrivez une fonction $evaluer\_polynome(P, x)$ qui prend en paramètres un polynôme P codé sous forme de liste et un réel x et qui renvoie l'évaluation de P en x. Par exemple, $evaluer\_polynome([1, 0, 2], 3)$ renvoie le réel 19 car $1 + 0 \times 3 + 2 \times 3^2 = 19$.
3. Écrivez une fonction $scalaire\_polynome(P, s)$ qui prend en paramètres un polynôme P et un réel s et qui renvoie le polynôme $s \times P$ écrit sous forme de liste. Par exemple, $scalaire\_polynome([1, 0, 2, -1], 2)$ renvoie $[2, 0, 4, -2]$.
4. Écrivez une fonction $deriver\_polynome(P)$ qui prend en paramètre un polynôme P et qui renvoie la dérivée de P. Par exemple, $derivee\_polynome([3, -1, 2, 1])$ renvoie $[-1, 4, 3]$.
5. Écrivez une fonction $deriver\_nieme\_polynome(P, n)$ qui prend en paramètres un polynôme P et qui renvoie le polynôme $\frac{\partial^n}{\partial x^n} P$ qui est la dérivée $n^{\text{ième}}$ de P.
6. Écrivez une fonction $decalage\_polynome(P, d)$ qui prend en paramètres un polynôme $P = a_0 + \cdots + a_n x^n$ et un entier d et qui renvoie le polynôme $P \times x^d$ si d est positifs ou nul et $\left(a_{-d} x^{-d} + \cdots + a_n x^n\right) x^d$ si d est négatif. Par exemple, $decalage\_polynome([1, 0, 2, 3], 2)$ renvoie $[0, 0, 1, 0, 2, 3]$, alors que $decalage\_polynome([1, 0, 2, 3], -2)$ renvoie $[2, 3]$.
7. Écrivez une fonction $somme\_polynome(P, Q)$ qui prend en paramètres deux polynômes P et Q codés sous forme de listes et qui renvoie le polynôme $P + Q$, codé sous forme de liste. Par exemple, $somme\_polynome([1, 0, 3], [1, -1, 2, 4])$ renvoie $[2, -1, 5, 4]$ et $somme\_polynome([1, -1], [2, 1])$ donne $[3, 0]$.
8. Écrivez une fonction $produit\_polynome(P, Q)$ qui prend en paramètres deux polynômes P et Q, codés sous forme de listes, et qui renvoie le polynôme $P \times Q$ codé sous forme de liste. Par exemple, $produit\_polynome([1, 0, 1], [1, 1, 2])$ renvoie $[1, 1, 3, 1, 2]$. Vous pourrez utiliser les fonctions $decalage\_polynome(P, d)$ et $scalaire\_polynome(P, s)$ pour implémenter cette fonction.
9. Écrivez une fonction $puissance\_polynome(P, d)$ qui prend en paramètres un polynôme et un entier et qui renvoie le polynôme P^d écrit sous forme de liste.
10. Écrivez une fonction $composition\_polynome(P, Q)$ qui prend en paramètres deux polynômes P et Q et qui renvoie la composition de P avec Q : $P \circ Q(x) = P(Q(x))$.

Nous allons maintenant remplacer l'évaluation des fonctions en 0 par l'évaluation des polynômes en 0. C'est à dire, qu'à toute fonction f, on va approcher f par un polynôme P, de degré d, de sorte que, pour tout entier $k \leq d$, $\frac{\partial^k}{\partial x^d} f(0) = \frac{\partial^k}{\partial x^d} P(0)$, et de sorte que $f(x) \approx P(x)$ pour toutes valeurs de x proches de 0. On dit alors que P est le développement limité en 0 de f d'ordre d.

Le développement limité en 0 de l'exponentielle est :	Celui de $1/(1-x)$ est :
$e^x \approx \sum_{k=0}^{d} \frac{1}{k!} x^k$	$\frac{1}{1-x} \approx \sum_{k=0}^{d} x^k$

11. Écrivez une fonction *geometrique_polynome*(d) qui renvoie un polynôme de degré d qui est le développement limité d'ordre d en 0 de $1/(1-x)$. Par exemple, *geometrique_polynome*(4) renvoie $[1, 1, 1, 1, 1]$.
12. Écrivez une fonction *exponentielle_polynome*(d) qui renvoie un polynôme de degré d qui est le développement limité d'ordre d en 0 de l'exponentielle. Par exemple, *exponentielle_polynome*(4) renvoie $[1, 1, \frac{1}{2}, \frac{1}{6}, \frac{1}{24}]$.
13. Écrivez une fonction *g_polynome*(d) qui renvoie un polynôme qui est le développement limité d'ordre d en 0 de $g(x) = (1 + x - e^x).x^{-1}$.

Revenons à la formule initiale que nous souhaitons implémenter :

$$\sum_{k=0}^{n} k^d = \left(\frac{\partial^d}{\partial x^d} \frac{e^{(n+1)x} - 1}{e^x - 1} \right)(0). \tag{1}$$

14. Commencez par démontrer la formule ci-dessus (la formule 1).

Soit $s(x)$ la fonction définie par :

$$s(x) = \frac{\partial^d}{\partial x^d} \frac{e^{(n+1)x} - 1}{e^x - 1} = \frac{\partial^d}{\partial x^d} \frac{\left(e^{(n+1)x} - 1\right) x^{-1}}{1 - g(x)} \tag{2}$$

15. ⊗ Montrez que pour obtenir un calcul exact de $\sum_{k=0}^{n} k^d$, il faut approcher $\frac{1}{1-x}$ et $g(x)$ par des polynômes de degré d, et $e^{(n+1)x}$ par un polynôme de degré $d+1$. Montrez aussi que $g(0) = 0$ (cette condition est nécessaire pour utiliser le développement limité de $\frac{1}{1-x}$ dans le terme droit de l'équation 2).
16. Proposez un fonction *somme*$1(n, d)$ qui calcule $\sum_{k=0}^{n} k^d$ à l'aide de la formule 1 écrite en utilisant la forme du terme droit de la formule 2 et en prenant en compte les remarque de la question 15.
17. La fonction *somme*$1(n, d)$ ne renvoie pas une valeure exacte. Pourquoi ?
18. Écrivez une fonction *test*$1(nmax, dmax, err)$ qui renvoie vrai si *somme*$1(n, d)$ est égale à *somme*(n, d) à une erreur *err* près, pour tout entier n compris entre 0 et $nmax$ exclu, et pour tout entier d compris entre 0 et $dmax$ exclu.
19. Proposez (sans l'implémenter) une solution pour que *somme*1 renvoie une valeur exacte. Discutez très rapidement des avantages et inconvénients de la solution que vous proposez.
20. ⊗ Pourquoi, lorsque n devient grand devant d, la fonction *somme*$1(n, d)$ est plus rapide que la fonction *somme*(n, d) ?

En fait, on sait que $\sum_{k=0}^{n} k^d$ est un polynôme de degrés $d+1$ en n. Nous allons donc maintenant, déterminer, pour un entier d donné, le polynôme qui calcule cette somme.

21. ♠ Démontrer, à partir de l'équation 1, que

$$\sum_{k=0}^{n} k^d = \frac{1}{d+1} \sum_{j=1}^{d+1} \left(C_{d+1}^{j} \cdot \left(\frac{\partial^{d+1-j}}{\partial x^{d+1-j}} \frac{1}{1 - g(x)} \right)(0) \right) .(n+1)^j \qquad \text{où} \qquad C_j^i \text{ est le binomial } \frac{j!}{i!(j-i)!}.$$

On rappele que $g(x) = (1 + x - e^x).x^{-1}$.

22. Proposez une fonction *somme*2*_polynome*(d) qui renvoie un polynôme P de sorte que $P(n+1) = \sum_{k=0}^{n} k^d$.
23. Proposez un fonction *somme*$2(n, d)$ qui calcule $\sum_{k=0}^{n} k^d$ à l'aide de la fonction *somme*2*_polynome*(d).
24. ⊗ Quel est l'interêt d'utiliser *somme*$2(n, d)$ plutôt que *somme*$1(n, d)$?

Correction

1.
```
def somme(n,d):
    res = 0
    for k in range(n+1):
        res = res + k**d
    return res
```

2.
```
def evaluer_polynome(P,x):
    res = 0
    v = 1
    for k in range(len(P)):
        res = res + P[k] * v
        v = v * x
    return res
```

3.
```
def scalaire_polynome(P,s):
    res = []
    for i in range( len(P) ):
        res.append( P[i]*s )
    return res
```

4.
```
def deriver_polynome(P):
    res = decalage_polynome( P, -1 )
    for i in range( len(res) ):
        res[i] = res[i]*(i+1)
    return res
```

5.
```
def deriver_nieme_polynome(P,n):
    res = [ P[i] for i in range(len(P)) ]
    for i in range(n):
        res = deriver_polynome( res )
    return res
```

6.
```
def decalage_polynome(P,d):
    if d < 0 :
        return P[-d:]
    return [0 for i in range(d)] + P
```

7.
```
def copier_polynome(P,n):
    res = []
    for i in range( len(P) ):
        res.append( P[i] )
    for i in range( len(P), n ):
        res.append( 0 )
    return res

def somme_polynome(P,Q):
    res = copier_polynome( P, len(Q) )
    Q = copier_polynome( Q, len(P) )
    for k in range( len(res) ):
        res[k] = res[k] + Q[k]
    return res
```

8.
```
def produit_polynome(P,Q):
    res = []
    for d in range( len(P) ):
        res = somme_polynome(
            res,
            scalaire_polynome(
                decalage_polynome(Q,d),
                P[d]
            )
        )
    return res
```

9.
```
def puissance_polynome(P,d):
    res = [1]
    for i in range( d ):
        res = produit_polynome(res, P)
    return res
```

10.
```
def composition_polynome(P,Q):
    res = []
    for i in range( len(P) ):
        res = somme_polynome(
            res,
            scalaire_polynome(
                puissance_polynome(Q,i),
                P[i]
            )
        )
    return res
```

11.
```
def geometrique_polynome(d):
    return [ 1 for i in range(d+1) ]
```

12.
```
def factorielle(n):
    res = 1
    for i in range(1,n+1):
        res = res * i
    return res

def exponentielle_polynome(d):
    res = []
    for i in range( d+1 ):
        res.append(1.0/factorielle(i))
    return res
```

13.
```
def g_polynome(d):
    res = [1,1]
    res = somme_polynome(
        res,
        scalaire_polynome(
            exponentielle_polynome(d+1),
            -1
        )
    )
    res = decalage_polynome( res, -1 )
    return res
```

14. On reconnait dans $\frac{e^{(n+1)x}-1}{e^x-1}$ le calcul d'une somme géométrique de raison e^x. Ainsi,

$$\begin{aligned}\frac{\partial^d}{\partial x^d}\frac{e^{(n+1)x}-1}{e^x-1} &= \frac{\partial^d}{\partial x^d}\sum_{k=0}^{n} e^{k.x} \\ &= \sum_{k=0}^{n} k^d e^{k.x}.\end{aligned}$$

Si l'on évalue l'expression précédente en 0, on obtient,

$$\frac{\partial^d}{\partial x^d}\frac{e^{(n+1)x}-1}{e^x-1}(0) = \sum_{k=0}^{n} k^d.$$

15. Soit f_1 une fonction. Pour obtenir un développement limité (DL) d'ordre 0 de $\frac{\partial^d}{\partial x^d} f_1$, il faut un DL d'ordre d de f_1.

Soit f_2 une fonction, pour avoir un DL d'ordre d de $f_2 x^{-1}$ il faut un DL d'ordre $d+1$ pour f_2.

Soit f_3 et f_4 deux fonctions. Pour obtenir un DL d'ordre d de $f_3 f_4$, il faut des DLs d'ordre d pour f_3 et pour f_4.

Soit f_5 et f_6 deux fonctions telles que $f_6(0) = 0$. Pour obtenir un DL d'ordre d de $f_5 \circ f_6$, il faut des DLs d'ordre d pour f_5 et f_6.

Posons maintenant $f_5 = \frac{1}{1-x}$, $f_6 = g(x)$, $f_2(x) = e^{(n+1)x}-1$, $f_1 = f_3.f_4$, $f_3 = f_5 \circ f_6$ et $f_4 = f_2.x^{-1}$. Pour pouvoir calculer les DLs précédents, il faut vérifier que $f_6(0) = g(0) = 0$. Pour cela, il suffit de faire un DL de g d'ordre 0 :

$$\begin{aligned} g(x) &= (1+x-e^x).x^{-1} \\ &= (1+x-(1+x+o(x)))\,x^{-1} \\ &= o(1). \end{aligned}$$

Comme $\sum_{k=0}^{n} k^d = \frac{\partial^d}{\partial x^d} f_1(0)$, pour calculer $\sum_{k=0}^{n} k^d$, nous devons obtenir un DL d'ordre 0 pour $\frac{\partial^d}{\partial x^d} f_1$. Nous en déduisons qu'il faut déterminer des DLs d'ordre d pour $g(x)$ et $\frac{1}{1-x}$ et un DL d'ordre $d+1$ pour $e^{(n+1)x}$.

16.
```
def somme1(n,d):
    tmp1 = decalage_polynome(
      somme_polynome(
        composition_polynome(
            exponentielle_polynome(d+1),
            [0,n+1]
        ), [-1]
      ),
      -1
    )
    tmp2 = composition_polynome(
        geometrique_polynome(d),
        g_polynome(d)
    )
    res = deriver_nieme_polynome(
        produit_polynome( tmp1, tmp2 ),
        d
    )
    return evaluer_polynome(res,0)
```

17. Cet algorithme renvoie TOUJOURS un calcul EXACT. Bien que l'on approche certaines fonctions par des polynômes, bien que pour des valeures proches de 0 leur évaluation est approximative, si on choisi d'évaluer ces polynômes en exactement 0, on obtient toujours une valeure exacte ! Ce n'est donc pas l'utilisation des polynômes qui pose problème. En fait, c'est l'utilisation des fractions rationnelles dans le développement limité de l'exponentille qui fait appraître des calculs approximatifs. En effet, sous python, le calcul des fractions rationnelles n'est pas exact car le résultat est stocké à l'aide de réels codé par le type `float`.

18.
```
def test1( nmax, dmax, erreur=0.0001 ):
    for n in range(0,nmax):
        for d in range(0,dmax):
            di = somme(n,d)-somme1(n,d)
            if abs(di) > erreur :
                return False
    return True
```

19. Un solution consiste à ne pas utiliser les réèls de python, et a implémenter les fractions rationnelles par des couples d'entiers pour pouvoir faire des calculs exacts (il n'y a pas de limite de taille pour les entiers en Python). Dans ce cas, il faut réimplémenter l'addition, la multiplication et tous les opérateurs usuels sur les réels.

Dans la pratique, on peut utiliser, pour faire cela, le module 'fractions' qui est présent dans les bibliothèques de Python.

Il suffit alors de remplacer l'implémentation de *exponentielle_polynome*(d) par

```
from fractions import Fraction

def exponentielle_polynome(d):
    res = []
    for i in range( d+1 ):
        res.append(
            Fraction(1,factorielle)
        )
    return res
```

et de remplacer la dernière ligne de *somme*1(n,d) par :

```
    return int(evaluer_polynome(res,0))
```

Une autre solution consiste à arrondir le résultat de *somme*1(n,t) aux entiers. Cette solution est plus rapide à mettre en oeuvre et à s'exécuter, mais elle donnera des résultats faux lorsque n et d prendront des valeures trop grandes.

20. La fonction *somme*1(n,d) utilise des fonctions dont le nombre d'opérations dépend uniquement du degrés des polynômes utilisés. Or les degrés des polynômes utilisés dépendent uniquement de d. Le nombre d'opérations réalisées par *somme*1(n,d) ne dépend donc pas de n. Lorsque n est grand devant d, la fonction *somme*1(n,d) réalise beaucoup moins d'opérations que la fonction *somme*(n,d) dont le nombe d'opérations est un $O(nd)$.

21. Soit $f \in \mathcal{C}^{d+1}$, alors

$$\begin{aligned}\frac{\partial^{d+1}}{\partial x^{d+1}}(xf) &= \sum_{j=0}^{d+1} C_{d+1}^{j} \frac{\partial^j}{\partial x^j} x \frac{\partial^{d+1-j}}{\partial x^{d+1-j}} f \\ &= x\frac{\partial^{d+1}}{\partial x^{d+1}} f + (d+1)\frac{\partial^d}{\partial x^d} f \\ \frac{\partial^{d+1}}{\partial x^{d+1}}(xf)(0) &= (d+1)\frac{\partial^d}{\partial x^d} f(0)\end{aligned}$$

Comme

$$\frac{e^{(n+1)x}-1}{e^x-1} = \sum_{k=0}^{n} e^{k.x}$$

est $\mathcal{C}^{d+1}$ sur $\mathbb{R}$, on peut utiliser la formule précédente en posant $f = \frac{e^{(n+1)x}-1}{e^x-1}$ pour obtenir :

$$\begin{aligned}\frac{\partial^d}{\partial x^d}\frac{e^{(n+1)x}-1}{e^x-1}(0) &= \frac{1}{d+1}\frac{\partial^{d+1}}{\partial x^{d+1}}\frac{x\left(e^{(n+1)x}-1\right)}{e^x-1}(0) \\ &= \frac{1}{d+1}\frac{\partial^{d+1}}{\partial x^{d+1}}\frac{e^{(n+1)x}-1}{1-g(x)}(0)\end{aligned}$$

Si l'on développe cette dernière expression, la somme $\sum_{k=0}^{n} k^d$ devient

$$\frac{1}{d+1}\sum_{j=0}^{d+1} C_{d+1}^{j} \frac{\partial^j}{\partial x^j}\left(e^{(n+1)x}-1\right)\frac{\partial^{d+1-j}}{\partial x^{d+1-j}}\frac{1}{1-g(x)}(0)$$

L'évaluation en 0 de $\frac{\partial^j}{\partial x^j}\left(e^{(n+1)x}-1\right)$ vaut 0 si $j=0$ et $(n+1)^j$ sinon. Ainsi, la somme $\sum_{k=0}^{n} k^d$ est égale à

$$\frac{1}{d+1}\sum_{j=1}^{d+1} C_{d+1}^{j} \frac{\partial^{d+1-j}}{\partial x^{d+1-j}}\frac{1}{1-g(x)}(0)(n+1)^j.$$

22.
```
def binomial( n,i ):
    nu = factorielle(n)
    de = factorielle(i)*factorielle(n-i)
    return nu/de

def somme2_polynome(d):
    res = [0]
    for j in range(1,d+2):
        pol = deriver_nieme_polynome(
            composition_polynome(
              geometrique_polynome(d+1-j),
              g_polynome(d+1-j)
            ), d+1-j
        )
        coef = evaluer_polynome(pol,0)
        coef = coef*binomial(d+1,j)/(d+1)
        res.append( coef )
    return res
```

23.
```
def somme2( n,d ):
    P = somme2_polynome(d)
    return evaluer_polynome( P, n+1 )
```

24. Lorsque l'on souhaite calculer plusieurs fois $\sum_{k=0}^{n} k^d$ pour la même valeur de d, mais pour des valeurs de n différentes, il n'est pas nécessaire de recalculer *somme*2*_polynome*(d) car il ne dépend pas de n. Pour pouvoir réutiliser les calculs réalisés précédement, on peut utiliser un dictionnaire et implémenter *somme*2(n,d) de la façon suivante :

```
precalcul_polynome = {}

def somme2( n,d ):
    if d in precalcul_polynome :
        P = precalcul_polynome[d]
    else :
        P = somme2_polynome(d)
        precalcul_polynome[d] = P
    return evaluer_polynome( P, n+1 )
```

Ainsi, pour des calculs utilisant la même valeur de d, la complexité de *somme*2(n,d), après la première exécution, devient identique à la complexité de *evaluation_polynome*$(P,n+1)$ qui est en $O(d)$.