D.S. 2

Autour des nombres premiers : Mines 2019

1 Préliminaires

Question 1

```
from math import floor, ceil, log, sqrt
log(0.5)
```

Ou, si on veut conserver l'origine des fonctions

```
import math as m
m.log(0.5)
```

Question 2

```
def sont_proches(x,y):
    atol = 1e-5
    rtol = 1e-8
    return abs(x-y) <= atol + abs(y) *rtol
```

Question 3

```
def mystere(x, b):
    if x < b:
        return 0
    else:
        return 1 + mystere(x/b, b)
```

De la récursivité !

```
def mystere(1001, 10) = 1 + mystere(100.1, 10)
                      = 2 + mystere(10.01, 10)
                      = 3 + mystere(1.001, 10)
                      = 3
```

Question 4

On constate que, si $b^p \leqslant x < b^{p+1}$ alors $b^{p-1} \leqslant x < b^p$ et, par récurrence,
`mystere(x, b) = p + mystere(x/b**p, b)`.
Or $x.b^{-p} < p$ donc `mystere(x, b) = p`.
On aurait donc `mystere(x, b) = floor(log(x)/log(b))`.
MAIS ceci n'est vrai que pour $p \geqslant 0$, c'est-à-dire $x \geqslant 1$.
Pour $x \leqslant 1$, la fonction renvoie 0 : une des erreurs de l'énoncé ?

Question 5

```
pas = 1e-5

x2 = 0
for i in range(100000):
    x1 = (i+1)*pas
    x2 = x2 + pas

print("x1 : ", x1)
print("x2 : ", x2)

x1 : 1.0
x2 : 0.9999999999980838
```

Le premier commentaire qui vient à l'esprit est qu'on ne comprend pas l'utilité de faire 100000 produits alors qu'on n'utilise que le dernier.
Bien sur, la réponse attendue est que les erreurs d'arrondi s'accumulent, ici lors de la somme.
Tous les calculs flottants sont faux.

2 Génération de nombre premiers

2.1 Approche systématique

Question 6
Une mémoire vive de 4 Go, soit 4×8 Giga bits permet d'enregistrer un tableau de 1 Giga cases de 32 bits, c'est-à-dire 10^9 cases (ou 2^30 si on considère que 1 Ko = 1024 octets).

Question 7
Comme il n'y a que deux valeurs pour un booléen, on peut le coder par un seul bit.
On pourrait alors enregistrer une liste 32 fois plus longue.

Question 8
Pour marquer faux les multiples de i, on peut utiliser un pas de i dans le `range` : on commence à $2i$ et on finit au pire à N,. Ne pas oublier le décalage de -1 dans le tableau

```
def erato_iter(N):
    liste_bool=[True]*N
    liste_bool[0]=False # 1 n'est pas premier
    for i in range(2, floor(sqrt(N))+1): #Attention à ne pas
        exclure racine de N dans le cas d'un carré parfait.
        if liste_bool[i-1]:
            for k in range(2*i, N+1, i):
                liste_bool[k-1]=False
    return liste_bool
```

Question 9
- La création de la liste de booléens demande N opérations
- Le parcourt de la boucle pour i et le teste demandent moins de $\sqrt{N}$ opérations.
- Pour chaque nombre premier $p \leqslant \sqrt{N}$ on fait moins de $\frac{N}{p}$ affectations.

On obtient un total d'opérations majoré par $C(N) = N + \sqrt{N} + \sum_{p\leqslant\sqrt{N}} \frac{N}{p} = N + \sqrt{N} + N\sum_{p\leqslant\sqrt{N}} \frac{1}{p}$.
La propriété admise donne $N\sum_{p\leqslant\sqrt{N}} \frac{1}{p} \sim N\ln\big(\ln(\sqrt{N})\big) \sim N\ln\big(\ln(N)\big)$.
Comme ce terme domine les autres, la complexité est de l'ordre de $N\ln\big(\ln(N)\big)$.

Question 10
Si l'écriture de N a m chiffres (en base 10) alors $N \leqslant 10^{m+1}$.
L'équivalent ci-dessus donne $C(N) \leqslant A.N.\ln\big(\ln(N)\big)$ donc
$C(N) \leqslant A.10.10^m \ln\big((m+1)\ln(10)\big) = A'.10^m\big(\ln(m+1) + \ln(10)\big)$.
Comme on a $\ln(m+1) \sim \ln(m)$ et $\ln(m)$ domine la constante $\ln(10)$,
on trouve un majorant de l'ordre de $10^m.\ln(m)$.

2.2 Génération rapide de nombres premiers

Question 11
L'énoncé est particulièrement obscur : il semble ne pas tenir compte de x_0 ce qui ne produirait que des nombres pairs, ce qui est particulièrement génant si on cherche des nombres premiers.
On va sommer à partir de x_0 : si les x_i sont impairs on ajoute toutes les puissances de 2
donc $A = \sum_{i=0}^{N-1} 2^i = 2^N - 1$.

Question 12
Les indications de l'énoncé sont (encore) mal écrites : on doit utiliser les 7 décimales du temps, il faut donc les calculer et les convertir en entier.
On doit

```
def bbs(N):
    p1 = 24375763
    p2 = 28972763
    M=p1*p2
    h = time.time()
    decimales = h - floor(h)
    xi=floor(decimales*10**7)
    for i in range(N):
        if xi%2 == 1:
            A = A + 2**i
        xi=(xi**2)%M
    return A
```

On aurait pu simplifier les calculs en ajoutant une variable qui prend la valeur de 2^i, il suffit de la multiplier par 2 à chaque étape plutôt que de calculer une puissance. Cependant Python calcule les puissances d'un entier avec un algorithme rapide.

Le sujet comporte une autre erreur : avec le calcul proposé on a $x_0^2 \leqslant M$ donc $x_1 = x_0^2$ a la même parité que x_0. On aura toujours les deux premiers bits de A égaux, on ne construit que la moitié des entiers possibles.

Question 13
On commence par une fonction de test de quasi-primalité.

```
def quasi_premier(p):
    prem = True
    for a in [2, 3, 5, 7]:
        if a**(p-1) % p != 1:
            prem = False
    return prem
```

On peut alors chercher un nombre. Le générateur aléatoire du sujet est tellement mal écrit qu'on aurait pu utiliser le bon vieux `randint`.

```
def premier_rapide(n_max):
    N = mystere(n_max, 2)
    p = bbs(N)
    while p < 2 or not quasi_premier(p):
        p = bbs(N)
    return p
```

Question 14

```
def stats_bbs_fermat(N, nb):
    erreurs=[]
    premier = erato_iter(N)
    for k in range(nb):
        p = premier_rapide(N)
        if not premier[p-1]:
            erreurs.append(p)
    return erreurs, len(erreurs)/nb
```

3 Compter les nombre premiers

3.1 Utilisation du crible

Question 15

```
def Pi(N):
    premier = erat0_iter(N)
    res=[0]*N
    pin=0
    for n in range(1, N+1):
        if premier[n - 1]:
            pin = pin + 1
        res[n-1] = [n, pin]
    return res
```

Question 16

```
def verif_Pi(N):
    pi = Pi(N)
    res=True
    for n in range(5393, N+1):
        if n/(log(n)-1) >= pi[n-1][1]:
            res=False
    return res
```

3.2 Calcul d'intégrale

Question 17

La complexité est linéaire en le nombre d'intervalles, $n = \dfrac{b-a}{\varepsilon}$.
L'énoncé signale plus loin que ce quotient est entier.

Question 18

La complexité est toujours linaire : on fait un nombre constant d'opération pour chaque pas.

Question 19

```
def inv_ln_rect_d(a, b, pas):
    n = floor((b-a)/pas)
    somme=0
    for k in range(1, n+1):
        somme = somme + 1/log(a+k*pas)
    return somme*pas
```

Question 20

```
def li_d(x, pas):
    if x == 1:
        return -float("infinity")
    elif x > 1:
        return inv_ln_rect_d(0, 1-pas, pas)
             + inv_ln_rect_d(1+pas, x, pas)
    else:
        return inv_ln_rect_d(0, x, pas)
```

3.3 Analyse des résultats

Question 21

Au "voisinage de 1.4", comme dit l'énoncé, la fonction s'annule. Il semble normal que l'écart relatif, qui divise l'écart par la valeur, n'ait pas de sens (il tend vers l'infini en valeur absolue).

Question 22

On peut noter que le sujet, prudent, à utilisé les sommes de Riemann à droite ; en effet la fonction admet un prolongement par continuité en 0, de valeur 0, mais l'inverse du log donnerait une erreur. Par contre la fonction à intégrer n'est pas intégrable sur $[0;1[$ ni sur $]1;x]$. Ici encore il n'est pas illogique que les calculs donnent des résultats aberrants.
La partie $[0;1-\varepsilon]$ calcule la fonction jusqu'à la valeur $1-\varepsilon$,
la partie $[1+\varepsilon;x]$ calcule la fonction depuis la valeur $1+2\varepsilon$.
Le terme $\varepsilon.f(1-\varepsilon)$ est équivalent à -1 et il n'est pas "équilibré" par un terme symétrique d'où l'écart de 1 constaté.

Question 23

On peut, simplement, utiliser les rectangles à gauche pour la partie $[1+\varepsilon;x]$.

3.4 Calcul de série

Question 24

```
def li_dev(x):
    xx = log(x)
    if xx <= 0:
        return False
    else :
        ei_avant = 0.577215664901 + log(xx)
        ei = ei_avant + xx # valeur de ei(1)
        n=1
        fact = 1 # On calcule la factorielle par
           multiplications
        puissance = xx # On calcule les x**n
        while n <= MAXIT and not sont_proches(ei, ei_avant):
            n = n + 1
            fact = n*fact
            puissance = puissance*xx
            ei_avant = ei
            ei = ei + puissance/n/fact
        if n > 100:
            return False
        else:
            return ei
```

4 Analyse des performances

Question 25
Plusieurs enregistrements ont la même valeur pour le champ `nom`, ce ne peut pas être une clé primaire.

Question 26

1. Une requête simple avec agrégations.

```
SELECT COUNT(*) AS nb_ordi, AVG(RAM) AS RAM_moyenne
FROM ordinateurs
```

2. La difficulté est qu'il faut exclure une partie des résultats.
   On utilise la différence ensembliste : `EXCEPT`.

```
SELECT nom FROM ordinateurs
   EXCEPT
SELECT teste_sur FROM fonction
WHERE  algorithme="rectangles" and nom = "li"
```

   On n'a alors plus besoin de jointure.

3. Le tri est vu en TP mais n'est pas au programme ...

```
SELECT f.algorithme, o.nom, o.ram, o.gflops
FROM fonctions AS f JOIN ordinateurs AS o ON f.teste_sur = o.
   nom
WHERE f.nom = "Ei"
ORDER BY f.temps_exec DESC
```