Informatique - Composition 2 – CORRIGÉ

Conseils, remarques et erreurs fréquentes

— Comme pour toute copie scientifique que vous allez rendre au concours, les résultats non mis en valeur risquent de ne pas être lus. Vous devez faciliter la lecture et la correction en écrivant lisiblement et noir ou en bleu et en utilisant des couleurs pour faire ressortir les parties importantes, vous pouvez illustrer votre raisonnement par des schémas.

— En terme de lisibilité, il vaut mieux éviter d'utiliser l'indice `i`. En effet sur certaines copies, il est difficile de distinguer les `i` et les `:`.

— Quand vous commencer à écrire une fonction ou un algorithme, vous devez avoir une idée de sa longueur et vous devez donc commencez sur une nouvelle page si vous pensez que vous n'avez pas assez de place (et en cas d'hésitation, commencez votre fonction en haut d'une page !).

— Vous devez toujours connaître le type des variables avec lesquelles vous travaillez ou maîtriser leur structure (par exemple dimension d'un tableau `numpy`).

  — Les arguments de la "fonction" `range` doivent être de type entier `int` sinon `Python` renvoie une erreur.

  — Les indices d'une liste doivent aussi être de type entier.

Python
```
N = 6/2
Ls = [k for k in range(N)]

    TypeError: 'float' object cannot be interpreted as an integer
```

Dans l'exemple ci-dessus, il faut alors utiliser une division euclidienne // pour définir `N`.

— Vous devez répondre précisément aux questions. Si dans une fonction, on vous demande de renvoyer une liste, il ne s'agit pas de renvoyer un tuplet ou tout autre objet. Les différents éléments doivent alors être mis entre crochets.

— Attention, quand on initialise une variable à une liste vide `Ls = []`, il faut ensuite ajouter les éléments soit par concaténation soit en utilisant la méthode `append`. Mais il n'est pas possible de faire une affectation simple du type `Ls[k] = 8` car l'élément d'indice `k` n'existe pas encore. De même, il faut concaténer une liste avec une liste ; ainsi `Ls += 5` renvoie un message d'erreur.

— L'utilisation d'une boucle `while` doit toujours être évitée quand c'est possible (à moins que sa terminaison ne soit rigoureusement prouvée). Dans la question 13 elle pouvait être pratique mais une boucle `for` n'était pas compliquée à mettre en place.

Commentaires spécifiques à ce sujet

1. Question bien traitée en général. Certains ont choisi de créer une fonction `moyenne` ce qui était une bonne idée et pouvait s'avérer pratique pour la suite. Attention cependant lors du calcul de la variance à ne pas recalculer la moyenne des éléments de la liste à chaque itération : il faut passer par une variable intermédiaire.

3. Quelques erreurs de calcul... Vous devez savoir faire des calculs simples sans calculatrice. Vous avez parfois manqué de précision sur la 2e partie de la question. Pour valider tous les points, vous devez être précis et synthétique. Ici le plus simple étant de revenir à la définition du ko et du kio.

4. Question plutôt bien traitée mais des maladresses dans l'initialisation du tableau. Vous devez bien lire l'énoncé. Ici, il était indispensable de créer un nouveau tableau. Vous ne pouviez pas modifier `A` (attention aux problèmes d'alias). Le plus simple ici était d'utiliser la fonction `numpy np.zeros` qui prend comme argument un tuplet. Le dernier paramètres est optionnel. Vous devez connaître la syntaxe Python au niveau de l'aide proposée.

6. Attention à bien renvoyer une liste d'indices et donc d'entiers (et non pas de flottants).

9. Plusieurs copies ont proposé un schéma explicatif de leur algorithme, ce qui a été très clairement apprécié. *A contrario*, un certain nombre de codes « originaux » mais sans aucune explication n'ont rapporté aucun point même s'ils fonctionnaient peut-être...

11. La fonction devait renvoyer un entier ce qui n'a pas toujours été respecté. En particulier, la fonction `np.floor` renvoie un flottant.

12. Globalement peu traité mais bien fait le cas échéant. Quelques codes sont inutilement compliqués car leurs auteurs oublient qu'il est possible de diviser par N tous les éléments de l'array `ls` en faisant simplement `ls/N`.

13. Les réponses sans aucune justification n'ont rapporté aucun point. Il faut quand même dénombrer les boucles imbriquées.

14. Que de réponses farfelues ! ! ! Si `z` est un array, on peut lui appliquer des opérations algébriques ou des fonctions mathématiques de numpy qui seront alors appliquées à tous les éléments.

15. Des confusions entre la dichotomie appliquée à un tableau d'entiers triés et à une fonction : dans le fond il n'y a pas beaucoup de différences mais l'emploi de l'une à la place de l'autre montre un manque de recul qu'il faut éliminer.
Par ailleurs, cette question nécessitait de faire des conversions d'unités qui ont été très rarement effectuées (sans parler de la capacité de certains à se tromper dans la conversion des μm ou des pN en m ou en N...). Rappelez vous que dans notre programme, l'informatique est un outil pour la physique, la chimie ou la SI et dans ces domaines, les grandeurs ont des unités !

17. On attendait ici la présentation du problème de Cauchy sous sa forme vectorielle.

19. La définition de la fonction à laquelle appliquer la méthode d'Euler a souvent être correcte. Par contre, la gestion du vecteur contenant les conditions initiales s'est révélée plus incertaine.

20. Question mal traitée dans l'ensemble. Beaucoup de vos réponses manquent de précision.

21. A nouveau, beaucoup de réponses manquent de précision.

24. Votre schéma doit être propre et suffisamment grand ! Indiquez proprement les itérations successives (par exemple x_0, x_1... en abscisses).

Correction

1.

```
def variance(ls):
    """IN : une liste de nombre (list)
    OUT : la variance des nombres de la liste (float)"""
    Slin = 0 # somme des éléments
    Scarre = 0 # somme des éléments au carré
    n = len(ls)
    for i in range(n):
        Slin += ls[i]/n
        Scarre += ls[i]**2/n
    return Scarre - Slin**2
```

2.

```
L1 = [1,10,20,50]
V1 = variance(L1)
```

3.

$$\text{Espace mémoire} = \underbrace{\text{nombre de pixels}}_{600\times400} \times \underbrace{\text{espace-mémoire d'un pixel}}_{1 \text{ octet}} = 240000 \text{ octets} = 240 \text{ ko}$$

— 1 ko vaut 1000 octets ;

— 1 kio vaut 2^{10} octets soit 1024 octets ;

4.

```
def seuillage(A,seuil):
    """IN : un tableau A (np.array) correspondant à l'image, la valeur de seuil (integer)
    OUT : tableau (np.array) de mêmes dimensions contenant
    des 1 là où la valeur des pixels de l'image originale est < seuil et 0 ailleurs"""
    Nligne = A.shape[0]
    Ncol = A.shape[1]
    Aseuil = np.zeros((Nligne,Ncol))
    for i in range(Nligne):
        for j in range(Ncol):
            if A[i,j] < seuil:
                Aseuil[i,j] = 1
    return Aseuil
```

5.

Le tableau contient 5 cases à 1. On somme soit les indices des lignes, soit les indices des colonnes et on en fait la moyenne en divisant par 5 :

— indice de ligne : 2 car (1+1+1+2+3)/5 = 1.6

— indice de colonne : 2 car (1+2+3+2+2)/5 = 2

0	0	0	0	0
0	1	1	1	0
0	0	**1**	0	0
0	0	1	0	0

6.

```
def pixel_centre_bille(A):
    """IN : un tableau A (np.array) correspondant à l'image seuillée
    OUT : liste des deux indices (ligne et colonne) du pixel le plus
    proche du centre de la bille"""
    ic = 0      # Initialisation de l'indice de la ligne correspondant au centre
    jc = 0      # Initialisation de l'indice de la colonne correspondant au centre
    cpt = 0     # compteur de pixels à 1 (initialisation)
    Nligne = A.shape[0]
    Ncol = A.shape[1]
    for i in range(Nligne):
        for j in range(Ncol):
            if A[i,j] == 1.:
                ic+= i
                jv+= j
                cpt += 1
    return [round(ic/cpt), round(jc/cpt)]
```

7.

```
def positions(n,seuil):
    """IN : n (integer) le nombre de photographies de la bille prises ; seuil (integer)
    OUT : la liste de ses positions dans chaque photographie en seuillant les images
    à la valeur seuil (list de list)"""
    lspos = []
    for k in range(n):
        Photo = prendre_photo()
        PhotoSeuil = seuillage(Photo,seuil)
        lspos.append(pixel_centre_bille(PhotoSeuil))
    return lspos
```

8.

```
def fluctuations(lsP,d):
    """IN : liste lsP de positions successives de la bille ; longueur d
    correspondant à un pixel (float)
    OUT : la valeur moyenne au carré (float) des déplacements quadratiques
    de la bille"""
    lsX = [elem[0] for elem in lsP]
    lsY = [elem[1] for elem in lsP]
    return (variance(lsX) + variance(lsY))*d**2
```

9.

```
def distance_maximale(ic,jc,P,Q) :
    """IN : ic, jc indices du centre de la billes (integer)
    et P, Q dimensions de l'image (integer)
    OUT : distance max entre le centre de la bille et les coins de l'image (float)
    """
    d2x = (P-1 - ic)**2     # P-1 : indice de la dernière ligne.
    if d2x < ic **2:
        d2x = ic**2
    d2y = (Q-1 - jc)**2     # Q-1 : indice de la dernière colonne
    if d2y < jc**2:
        d2y = jc**2
    return np.sqrt(d2x + d2y)
```

10.

```
def largeur_anneau(ic,jc,P,Q,n) :
    """IN : ic, jyc : position du centre de la bille (integer),
    P,Q : dimensions de l'image (integer),
    n : nombre d'anneaux (integer)
    OUT : rayon de l'anneau d'indice 0 (float)"""
    return distance_maximale(ic,jc,P,Q) / n
```

11.

```
def indice_anneau(ic,jc,P,Q,i,j,n) :
    """IN : ic, jc : position du centre de la bille (integer),
    P,Q : dimensions de l'image (integer),
    n : nombre d'anneaux (integer), i, j indices du pixel
    OUT : indice de l'anneau contenant le pixel (i, j) de type integer"""
    r0 = largeur_anneau(ic,jc,P,Q,n)     #rayon d'indice 0
    d2 =((i - ic)**2 + (j - jc)**2)**0.5 # distance du pixel i,j au centre ic,
    return int(d2 / r0) # entier le plus proche par valeur inférieure.
```

Attention, ici si on utilise la `np.floor(d2 / lb)` il faut ensuite convertir le résultat en un entier.

12.

```
def profil(A, n) :
    """IN : l'image A (array) ; nombre d'anneaux (integer)
    OUT : liste de n nombres, compris entre 0 et 1, qui donne la proportion
    de pixels blancs compris dans chaque anneau"""
    P,Q = A.shape[0],A.shape[1]
    ic, jc = pixel_centre_bille(A)
    blanc = np.zeros(n)   # initialisation  : nombre de pixels blanc dans chaque anneau
    total = np.zeros(n)   # initialisation : nombre total de pixels dans chaque anneau
    for i in range(P):
        for j in range(Q):
            a = indice_anneau(ic,jc,P,Q,i,j,n)
            total[a] += 1
            if A[i,j] != 0:
                blanc[a] += 1
    return blanc / total
```

13. La fonction `profil` fait appel aux fonctions :

- `pixel_centre_bille` qui est de complexité quadratique (car deux boucles `for` imbriquées) : on doit parcourir l'image en entier pour déterminer la position de la bille ;
- `indice_anneau` qui appelle la fonction `largeur_anneau` qui appelle elle-même la fonction `distance_maximale` qui est de complexité bornée : le nombre d'opérations et donc la durée d'exécution est indépendante des dimensions de la photo.

La fin de la fonction `profil` met en jeu deux boucles `for` impliquées.
Elle est donc globalement de **complexité quadratique** : $\mathcal{O}(p^2)$. Le nombre d'opérations sera proportionnel au nombre de pixels.

14.

```
def force(z) :
    return (K_B*T/Lp)*(1/(4*(1-z/L0)**2) - 1/4 + z/L0)
```

15. On voit sur le graphe que la force F vaut 0,1 pN entre $z = 10$ et $z = 20\mu$m : cela permet de choisir les bornes pour la dichotomie.

```
# fonction auxiliaire
def f_aux(z):
    return force(z)-0.1*10**(-12)

# fonction dichotomie
def dichotomie(f,a,b,epsilon):
    """IN : f : fonction à un argument dont on cherche le zero f(z)=0
    a et b : intervalle sur lequel on cherche la solution (float)
    epsilon : précision (float)
    OUT : solution de f(z)=0 sur le segment [a,b] à epsilon près (float)"""
    while b-a>=2*epsilon:
        c=(a+b)/2
        if f(c)*f(a)>=0:
            a=c
        else:
            b=c
    return (a+b)/2

solution = dichotomie(f_aux,10*10**{-6},25*10**{-6},0.01*10**{-12})
print(solution)
```

16.

```
def travail(F,z_1,z_2):
    """IN : la fonction F et les bornes de l'intégrale (float)
    OUT : le travail (float)"""
    N = 1000
    h = (z_2-z_1)/N
    somme = 0
    for k in range(N):
        somme += F(z_1 + k*h + h/2)
    return h*somme
```

17.

$$\frac{d}{dt}y = g(y,t) \qquad \text{avec} \quad y(0) = \begin{pmatrix} z(0) = z_0 \\ \dot{z}(0) = 0 \end{pmatrix}$$

où :

— y est le vecteur $\begin{pmatrix} z \\ \dot{z} \end{pmatrix}$;

— g vérifie la relation :

$$g\left(\begin{pmatrix} z \\ \dot{z} \end{pmatrix}, t\right) = \begin{pmatrix} \dot{z} \\ -\frac{mk_BT}{L_p}\left(\frac{1}{4(1-z/L_0)^2} - \frac{1}{4} + \frac{z}{L_0}\right) + \frac{F_0}{m}\cos\left(2\pi\frac{t}{T}\right) \end{pmatrix}$$

18. En notant Y_k l'approximation numérique de y à l'instant t_k, on peut toujours écrire :

$$Y_{k+1} = Y_k + hg(Y_k, t_k)$$

où $Y(t_k)$ est un vecteur qui vaut $\begin{pmatrix} Z_k \\ \dot{Z}_k \end{pmatrix}$ où Z_k et $\dot{Z}_k$ sont les valeurs approchées de z et $\dot{z}$ en t_k.

```python
def euler(a,b,f,y0,N):
    """Resolution de y'=f(y,t) sur l'intervalle [a,b]
    subdivisé en N segments
    y0 : vecteur conditions initiales (nd.array)
    OUT : triplet de 3 vecteurs (t,z,dz) solution de l'équation (nd.array)
    """
    h = (b-a)/N
    t = [a]      # Initialisation
    Y = [y0]     # Initialisation
    for k in range(0,N):
        newY = Y[-1] + f(Y[-1],t[-1])*h
        Y.append(newY)
        t.append(t[-1]+h)
    z = []  # va correspondre à z
    dz= []  # va correspondre à z point
    for vect in Y:
        z.append(vect[0])
        dz.append(vect[1])
    return t,z,dz
```

19.

```python
y0 = np.array([z0, 0])

def g(Y,t):
    return np.array([Y[1],-(K_B*T/(Lp*m))*(1/(4*(1-Y[0]/L0)**2) - 1/4
    + Y[0]/L0) + F0/m*np.cos(2*np.pi*t/T)])

t,z,dz = euler(0,10*T,g,y0,10/10**(-2))
```

20. La mantisse est codée sur 52 bits et permet donc de coder 2^{52} niveaux différents. Avec $2^{10} = 1024$, on trouve $2^{52} \simeq 4.10^{15}$. On peut donc obtenir au maximum une précision à 10^{-15} près en relatif. (Les bits dédiés à l'exposant donnent l'ordre de grandeur du nombre considéré).

21.

- Le choix $h = 1$ conduit à considérer l'intervalle $[0, 2x]$ comme voisinage de x !
- Le choix $h = 10^{-16}$ fait que compte tenu de la réponse à la question précédente, $1 + h = 1 - h = 1$ et conduit donc à une approximation de la dérivée systématiquement nulle !
- Diminuer la valeur de h permet de diminuer le reste du DL et donc d'améliorer la qualité théorique de l'approximation. Cependant diminuer h réduit également la précision du calcul. La valeur optimale de h dépend donc de la précision fournie pour le calcul de ϕ' et de la précision attendue pour le calcul du zéro.

  Une valeur de h de l'ordre de 10^{-7} semble un bon compromis compte tenu de la précision attendue dans la suite.

22.

```python
def derive(phi, x, h) :
    """IN une fonction, un point (float) et un pas (float)
    OUT : la dérivée première de la fonction en x"""
    return (phi(x*(1+h)) - phi(x*(1-h)))/(2*h*x)
```

23.

```python
def derive_seconde(phi, x, h) :
    """IN ue fonction, un point (float) et un pas (float)
    OUT : la dérivée seconde de la fonction en x"""
    return (derive(phi,x*(1+h),h) - derive(phi,x*(1-h),h))/(2*h*x)
```

24. Rappel : *cf.* cours.

```
def min_local(phi,x0,h):
    """IN une fonction, une abscisse initiale (float) et un pas (float)
    OUT : valeur approximée du minimum de phi"""
    un = x0
    unp1 = un - derive(phi,un,h)/derive_seconde(phi,un,h)
    while abs(derive(phi,un,h)) >= 1e-7 :
        un = unp1
        unp1 = unp1 - derive(phi,unp1,h)/derive_seconde(phi,unp1,h)
    return unp1
```