Centrale Informatique MP-PC-PSI 2019 — Corrigé

Ce corrigé est proposé par William Aufort (professeur en CPGE) ; il a été relu par Cyril Ravat (professeur en CPGE) et Benjamin Monmege (enseignant-chercheur à l'université).

Ce sujet s'intéresse à l'élasticité de l'ADN, plus particulièrement à la relation qui existe entre une force exercée sur un brin d'ADN et l'allongement de ce brin. Cette relation permettrait d'avancer dans la compréhension de processus biologiques, comme la réplication de l'ADN.

- La première partie comporte seulement trois questions d'implémentation de fonctions utilitaires, dont deux sont explicitement au programme d'informatique pour tous.
- La deuxième partie introduit le protocole expérimental permettant d'évaluer l'intensité de la force de traction exercée artificiellement sur un brin d'ADN auquel on a accroché une bille aimantée, ainsi que l'allongement du brin. Cette partie est principalement consacrée à des algorithmes de traitement d'images (représentées par des tableaux Numpy à deux dimensions).
- Une fois les mesures effectuées, on souhaite maintenant les faire correspondre à un modèle mathématique. La troisième partie étudie le « modèle du ver », qui donne la relation entre force de traction et allongement sous la forme d'une formule mathématique faisant intervenir des caractéristiques du brin. Cette partie propose de déterminer ces caractéristiques à partir des observations effectuées et d'une fonction prédéfinie en Python, puis explore les bases de l'algorithme de minimisation sous-jacent pour des problèmes à une ou deux dimensions.
- Enfin, la dernière partie propose un second modèle, probabiliste cette fois, permettant de simuler l'allongement d'un brin d'ADN soumis à une force donnée en paramètre.

Les parties 2, 3 et 4 sont indépendantes, et le sujet couvre une grande proportion du programme d'informatique pour tous (listes et tableaux Numpy, représentation des nombres, ingénierie numérique et récursivité). La plupart des questions sont abordables dès la première année. Enfin, dans chaque partie la dernière question est plus délicate, sans être vraiment difficile.

Indications

Partie I

2 Attention à ne pas utiliser la fonction `moyenne` n fois.

3 L'utilisation d'une fonction récursive semble adaptée.

Partie II

5 Pour arrondir à l'entier le plus proche, on pourra utiliser la fonction `round`.

7 Commencer par calculer le barycentre des positions de la liste, puis dans un deuxième temps la moyenne des déplacements quadratiques. Ne pas oublier d'utiliser correctement le paramètre `t`.

8 L'énoncé suggère d'utiliser des fonctions intermédiaires. On peut commencer par déterminer le rayon extérieur du plus grand anneau en fonction de la position du centre de la bille. Puis pour chaque pixel de l'image, on cherchera à exprimer le numéro de l'anneau dans lequel il se trouve.

Partie III

11 Dans la fonction `curve_fit`, la fonction `f` ne doit pas avoir la température `T` comme argument. Il faut également extraire `xdata` et `ydata` à partir du tableau à deux colonnes `Fz`. Attention à l'ordre.

12 Le nombre de chiffres significatifs dépend uniquement de la mantisse.

13 Pour $h = 10^{-16}$, utiliser la question 12 pour savoir comment seront représentés les nombres $1 + h$ et $1 - h$ dans ce codage.

15 Le premier argument de `derive` est une fonction, il faut donc être précautionneux si on veut utiliser `derive` deux fois de suite.

16 Il faut appliquer la méthode de Newton à ϕ' et non à ϕ.

17 Le système se déduira directement des équations des plans tangents considérés.

18 Deux stratégies sont possibles : utiliser les applications partielles, ou effectuer le calcul à la main, avec des formules semblables à celle utilisée à la question 14.

19 On pourra séparer de la fonction `min_local_2D` le calcul de la matrice $\mathrm{J}(x, y)$. Pour ce calcul, commencer par définir les fonctions g_x et g_y, puis réutiliser la fonction `grad`.

Partie IV

20 Générer des nombres aléatoires dans $[\,0\,;1\,[$, puis utiliser une transformation affine pour obtenir des angles dans l'intervalle $[\,-\pi\,;\pi\,[$.

22 La question demande de créer une nouvelle conformation. Attention également aux indices manipulés.

24 Utiliser une liste comme une file pour conserver uniquement les 500 derniers allongements. On pourra commencer par initialiser les 500 premiers allongements, puis utiliser une boucle `while` pour la phase de convergence.

I. Fonctions utilitaires

1 Il suffit d'additionner tous les éléments de la séquence à l'aide d'une boucle, sans oublier de diviser par la longueur de la séquence pour obtenir la moyenne.

```
def moyenne(X):
    somme = 0
    for i in range(len(X)):
        somme += X[i]
    return somme/len(X)
```

> On peut aussi parcourir la séquence sans utiliser d'indice avec un itérateur :
>
> ```
> def moyenne(X):
>     somme = 0
>     for x in X:
>         somme += x
>     return somme/len(X)
> ```

2 On utilise la formule de König-Huygens rappelée dans l'énoncé : on calcule d'abord la moyenne des carrés comme précédemment, à laquelle on soustrait le carré de la moyenne.

```
def variance(X):
    s = 0
    for x in X:
        s += x**2
    return s/len(X) - moyenne(X)**2
```

> Une autre stratégie consiste à utiliser la définition de la variance. Attention dans ce cas à utiliser la fonction `moyenne` une seule fois avant la boucle, afin d'éviter une complexité en $O(n^2)$.
>
> ```
> def variance(X):
>     s = 0
>     m = moyenne(X)
>     for x in X:
>         s += (x - m)**2
>     return s/len(X)
> ```

3 Pour pouvoir explorer l'ensemble des éléments de la séquence imbriquée, le plus simple est d'écrire une fonction `somme` récursive. Si l'argument `M` de la fonction `somme` est un composant élémentaire, c'est-à-dire un nombre réel (que l'on peut détecter avec `isinstance(M, numbers.Real)`), on peut le renvoyer directement. Dans le cas contraire, `M` est une séquence imbriquée : on calcule alors la somme des éléments qui la compose à l'aide d'une boucle et de la fonction `somme` appliquée à chacun de ces éléments.

> L'énoncé ne suppose pas que le module `numbers` a été importé, ce qui est pourtant nécessaire pour pouvoir utiliser `numbers.Real`.

```
def somme(M):
    if isinstance(M, numbers.Real):
        return M
```

```
    else:
        s = 0
        for x in M:
            s += somme(x)
        return s
```

Dans le code précédent, le `else` est facultatif, du fait de la présence du `return` dans le cas où `M` est un nombre réel. On pourrait donc l'enlever, ainsi que le niveau d'indentation induit dans tout le bloc suivant, ce qui peut rendre le code plus agréable à lire.

II. Mesures expérimentales

4 On commence par initialiser le tableau demandé à l'aide de la fonction `np.zeros`. Puis on parcourt le tableau initial à l'aide d'une double boucle pour mettre à jour les pixels dont la valeur est strictement inférieure au seuil, les autres étant déjà associés à la valeur 0 dans l'image seuillée.

```
def seuillage(A, seuil):
    S = np.zeros(A.shape, int)
    for i in range(A.shape[0]):
        for j in range(A.shape[1]):
            if A[i, j] < seuil:
                S[i, j] = 1
    return S
```

Pour récupérer les dimensions du tableau en argument, on a utilisé ici l'attribut `A.shape`, qui était donné en annexe. Dans ce type d'épreuve, il est primordial de bien lire l'annexe avant de commencer le sujet, pour identifier notamment des fonctions pouvant être utiles.

Une autre possibilité pour extraire les dimensions consiste à utiliser la fonction `len`. En effet, `len(A)` renvoie le nombre de lignes du tableau `A` (identique donc à `A.shape[0]`), et `len(A[0])` renvoie le nombre de colonnes de `A` (identique à `A.shape[1]`).

Enfin, une dernière solution consiste à utiliser le fait que la plupart des opérations usuelles (arithmétiques et logiques) sur les tableaux Numpy agissent élément par élément. On pourrait par exemple écrire :

```
def seuillage(A, seuil):
    return 1 * (A < seuil)
```

En effet, `A < seuil` renvoie un tableau Numpy de mêmes dimensions que `A`, contenant les booléens résultats des comparaisons des éléments de `A` avec `seuil`. Puis, le produit `1 * (A < seuil)` permet de renvoyer un tableau dans lequel chacun des booléens précédents a été multiplié par `1`, ce qui a pour effet de transformer les `True` en 1 et les `False` en 0. La fonction répond donc bien à la question posée.

5 On commence par extraire dans deux listes les abscisses et les ordonnées des pixels blancs de l'image, ce qui permettra de calculer les moyennes correspondantes avec la fonction `moyenne` de la question 1. Enfin on utilise la fonction `round`, rappelée dans l'annexe, pour arrondir chacune des deux valeurs obtenues à l'entier le plus proche.