Exercices types Algorithmique
Oral Mathématiques et Algorithmique
Banque PT

Ces exercices portent sur les semestres 1 et 2 ainsi que le paragraphe 3.2 du programme d'informatique commune des classes préparatoires scientifiques ainsi que les outils numériques et mathématiques de base, mentionnés en annexe des programmes de physique-chimie de la filière PTSI/PT.

Ils peuvent être de longueur et de difficulté inégales. L'examinateur saura, bien sûr, adapter sa notation selon le sujet.

Lors de l'épreuve, un memento sera mis à disposition du candidat, sous forme de feuilles à côté de l'ordinateur. Ce memento sera disponible début novembre à l'adresse : `https://www.banquept.fr/spip.php?article237`

Exercice

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

---

1. Soit l'entier `n = 1234`. Quel est le quotient, noté `q`, dans la division euclidienne de `n` par 10 ? Quel est le reste ? Que se passe-t-il si on recommence la division par 10 à partir de `q` ?
2. Écrire la suite d'instructions calculant la somme des cubes des chiffres de l'entier 1234.
3. Écrire une fonction `somcube`, d'argument `n`, renvoyant la somme des cubes des chiffres du nombre entier `n`.
4. Trouver tous les nombres entiers inférieurs à 1000 égaux à la somme des cubes de leurs chiffres.
5. En modifiant les instructions de la fonction `somcube`, écrire une fonction `somcube2` qui convertit l'entier `n` en une chaîne de caractères permettant ainsi la récupération de ses chiffres sous forme de caractères. Cette nouvelle fonction renvoie toujours la somme des cubes des chiffres de l'entier `n`.

Exercice

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

On cherche à calculer une valeur approchée de l'intégrale d'une fonction donnée par des points dont les coordonnées sont situées dans un fichier.

1. Le fichier `"test.csv"`, situé dans le sous-répertoire `data` du répertoire de travail, contient une quinzaine de lignes selon le modèle suivant :

   ```
   0.0;1.00988282142
   0.1;1.07221264497
   ```

   Chaque ligne contient deux valeurs flottantes séparées par un point-virgule, représentant respectivement l'abscisse et l'ordonnée d'un point. Les points sont ordonnés par abscisses croissantes.

   Ouvrir le fichier en lecture, le lire et construire la liste `LX` des abscisses et la liste `LY` des ordonnées contenues dans ce fichier.

2. Représenter les points sur une figure.

3. Les points précédents sont situés sur la courbe représentative d'une fonction f. On souhaite déterminer une valeur approchée de l'intégrale I de cette fonction sur le segment où elle est définie.

   Écrire une fonction `trapeze`, d'arguments deux listes `y` et `x` de même longueur `n`, renvoyant :

   $$\sum_{i=1}^{n-1} (\mathtt{x}_i - \mathtt{x}_{i-1}) \, \frac{\mathtt{y}_i + \mathtt{y}_{i-1}}{2} \, .$$

   `trapeze(LY,LX)` renvoie donc une valeur approchée de l'intégrale I par la méthode des trapèzes.

4. En utilisant la méthode d'intégration numérique `trapz` de la sous-bibliothèque `scipy.integrate` *du langage Python*, retrouver la valeur approchée de l'intégrale I.

Exercice

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

Soit un entier naturel n non nul et une liste `t` de longueur n dont les termes valent 0 ou 1. Le but de cet exercice est de trouver le nombre maximal de 0 contigus dans `t` (c'est-à-dire figurant dans des cases consécutives). Par exemple, le nombre maximal de zéros contigus de la liste `t1` suivante vaut 4 :

`i`	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
`t1[i]`	0	1	1	1	0	0	0	1	0	1	1	0	0	0	0

1. Écrire une fonction `nombreZeros(t,i)`, prenant en paramètres une liste `t`, de longueur n, et un indice i compris entre 0 et $n-1$, et renvoyant :

   $$\begin{cases} 0, \text{ si } \texttt{t}[i] = 1 \\ \text{le nombre de zéros consécutifs dans } \texttt{t} \text{ à partir de } \texttt{t}[i] \text{ inclus, si } \texttt{t}[i] = 0 \end{cases}$$

   Par exemple, les appels `nombreZeros(t1,4)`, `nombreZeros(t1,1)` et `nombreZeros(t1,8)` renvoient respectivement les valeurs 3, 0 et 1.

2. Comment obtenir le nombre maximal de zéros contigus d'une liste `t` connaissant la liste des
   `nombreZeros(t,i)` pour $0 \leqslant i \leqslant n-1$?
   En déduire une fonction `nombreZerosMax(t)`, de paramètre `t`, renvoyant le nombre maximal de 0 contigus d'une liste `t` non vide. On utilisera la fonction `nombreZeros`.

3. Quelle est la complexité de la fonction `nombreZerosMax` construite à la question précédente ?

4. Trouver un moyen simple, toujours en utilisant la fonction `nombreZeros`, d'obtenir un algorithme plus performant.

Exercice

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

Soient n un entier naturel strictement positif et p un réel compris entre 0 et 1.
On considère X et Y deux variables aléatoires à valeurs dans $\mathbb{N}$ sur un espace probabilisé donné.
X suit une loi de Poisson de paramètre $\lambda = np$ et Y suit une loi binomiale de paramètres (n, p).

1. Définir une fonction `Px`, d'arguments k, n et p, renvoyant la valeur de $P(X = k)$.
   $k!$ (*factorielle* k) *s'obtient par* `factorial(k)` *en Python* (*bibliothèque* `math`).
   Déterminer, pour $n = 30$ et $p = 0.1$, la liste des valeurs de $P(X = k)$ pour $k \in \mathbb{N}$, $0 \leq k \leq 30$.
2. Définir une fonction `Py`, d'arguments k, n et p, renvoyant la valeur de $P(Y = k)$.
   On pourra utiliser `comb` *de la sous-bibliothèque* `scipy.misc` *en Python.*
   Déterminer, pour $n = 30$ et $p = 0.1$, la liste des valeurs de $P(Y = k)$ pour $k \in \mathbb{N}$, $0 \leq k \leq 30$.
3. Soit $k \in \mathbb{N}$. On rappelle que, sous certaines conditions sur n et p, la probabilité $P(Y = k)$ peut être approchée par $P(X = k)$. Déterminer une fonction `Ecart` d'arguments n et p, renvoyant le plus grand des nombres $|P(Y = k) - P(X = k)|$, pour $0 \leq k \leq n$.
4. Soit e un réel strictement positif. Déterminer une fonction `N`, d'arguments e et p, renvoyant le plus petit entier n tel que `Ecart`(n, p) soit inférieur ou égal à e.
5. Faire l'application numérique dans les quatre cas suivants :
   — $p = 0.075$ avec $e = 0.008$ et $e = 0.005$.
   — $p = 0.1$ avec $e = 0.008$ et $e = 0.005$. Interpréter le dernier résultat.

Exercice

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

---

1. On considère le code Python de la fonction `d` suivante :

```
def d(n):
    L=[1]
    for nombre in range(2,n+1):
        if n%nombre==0:
            L.append(nombre)
    return L
```

   Quel est le résultat de l'appel `d(4)` ? Puis de l'appel `d(10)` ?

   Que fait la fonction `d` ?

2. Un *diviseur non-trivial* d'un entier `n` est un diviseur de `n` différent de `1` et de `n`. Écrire une fonction `DNT`, d'argument `n`, renvoyant la liste des diviseurs non-triviaux de l'entier `n`.

3. Écrire une fonction `sommeCarresDNT`, d'argument `n`, renvoyant la somme des carrés des diviseurs non-triviaux de l'entier `n`.

4. Écrire la suite des instructions permettant d'afficher tous les nombres entiers inférieurs à 1000 et égaux à la somme des carrés de leurs diviseurs non-triviaux. Que peut-on conjecturer ?

Exercice

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

Soit `n` un entier vérifiant $\texttt{n} \leqslant 26$. On souhaite écrire un programme qui code un mot en décalant chaque lettre de l'alphabet de `n` lettres.
Par exemple pour $n = 3$, le décalage sera le suivant :

avant décalage	a	b	c	d	...	...	x	y	z
après décalage	d	e	f	g	...	...	a	b	c

Le mot `oralensam` devient ainsi `rudohqvdp`.

1. Définir une chaîne de caractères contenant toutes les lettres dans l'ordre alphabétique (caractères en minuscule).
2. Écrire une fonction `decalage`, d'argument un entier `n`, renvoyant une chaîne de caractères contenant toutes les lettres dans l'ordre alphabétique, décalées de `n`, comme indiqué ci-dessus.
3. Écrire une fonction `indices`, d'arguments un caractère `x` et une chaîne de caractères `phrase`, renvoyant une liste contenant les indices de `x` dans `phrase` si `x` est une lettre de `phrase` et une liste vide sinon.
4. Écrire une fonction `codage` d'arguments un entier `n` et une chaîne de caractères `phrase`, renvoyant `phrase` codé avec un décalage de `n` lettres.
5. Comment peut-on décoder un mot codé ?

Exercice

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

On pose $M = 20$ et $m = 10$.
À un nombre c quelconque, on associe la suite $(u_n)_{n\geqslant 0}$ définie par

$$u_0 = 0 \quad \text{et} \quad u_{n+1} = u_n^2 + c \ \text{ pour } n \geqslant 0.$$

S'il existe, on note k le plus petit entier tel que l'on ait $0 \leqslant k \leqslant m$ et $|u_k| > M$.
On définit alors la fonction f par

$$f : c \mapsto \begin{cases} k & \text{s'il existe} \\ m+1 & \text{sinon.} \end{cases}$$

1. Donner le code définissant la fonction f.
2. Tracer l'allure de la courbe représentative de la fonction f sur $[-2; 2]$.
3. Construire le tableau des valeurs $f(x + \mathrm{i}\, y)$ où x prend 101 valeurs comprises entre -2 et 0.5 et y prend 101 valeurs entre -1.1 et 1.1. *On rappelle que le nombre complexe* i *est représenté par* `1j`. *Par exemple, le complexe* $1 + 2\,\mathrm{i}$ *est représenté par* `1+2j`.
4. Tracer l'image que code ce tableau à l'aide des fonctions `imshow` et `show` de la sous-bibliothèque `matplotlib.pyplot`.
   Quels paramètres peut-on modifier pour obtenir une meilleure résolution ?

Exercice

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

Soit N un entier naturel non nul. On cherche à trier une liste L d'entiers naturels strictement inférieurs à N.

1. Écrire une fonction `comptage`, d'arguments L et N, renvoyant une liste P dont le k-ième élément désigne le nombre d'occurences de l'entier k dans la liste L.
2. Utiliser la liste P pour en déduire une fonction `tri`, d'arguments L et N, renvoyant la liste L triée dans l'ordre croissant.
3. Tester la fonction `tri` sur une liste de 20 entiers inférieurs ou égaux à 5, tirés aléatoirement.
4. Quelle est la complexité temporelle de cet algorithme ? La comparer à la complexité des tri usuels.

Exercice

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

---

1. Deux paramètres b et w valant respectivement 0.5 et 6.0, définir trois fonctions d'une variable t renvoyant des couples :

$$\begin{cases} p : t \mapsto (\ \cos(t) + b\,\cos(w\,t) & ,\ \sin(t) + b\,\sin(w\,t) &) \\ v : t \mapsto (\ -\sin(t) - b\,w\,\sin(w\,t) & ,\ \cos(t) + b\,w\,\cos(w\,t) &) \\ a : t \mapsto (\ -\cos(t) - b\,w^2\,\cos(w\,t) & ,\ -\sin(t) - b\,w^2\,\sin(w\,t) &) \end{cases}$$

   Vérifier ces fonctions sur un exemple.

   $p(t) = (x(t), y(t))$ désigne la position dans le plan d'une masse ponctuelle mobile au cours du temps, $v(t) = (x'(t), y'(t))$, sa vitesse, et $a(t) = (x''(t), y''(t))$, son accélération.

2. Construire la liste `L` des points $p(t)$, pour t variant de $-\pi$ à π avec un pas de discrétisation δt vérifiant $\delta t = 0.01\,\pi$.

3. Faire tracer dans le plan muni d'un repère orthonormal la ligne polygonale reliant les points $p(t)$ de la liste `L`.

4. Définir puis tester la fonction c d'une variable t qui renvoie le couple des coordonnées du centre de courbure donné par :

$$c(t) = \big(\ x(t) - d\,y'(t)\ ,\ y(t) + d\,x'(t)\ \big)\ \text{ où } d = \frac{x'(t)^2 + y'(t)^2}{x'(t)\,y''(t) - y'(t)\,x''(t)}\ .$$

5. Rajouter sur le graphique précédent la ligne décrite par les centres de courbure, avec la même discrétisation en temps.

6. Calculer la longueur de la ligne polygonale reliant les points $p(t)$, pour différents pas de discrétisation δt. Observer l'évolution de cette longueur lorsque δt diminue.

Exercice

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

On considère un pendule amorti, initialement en position verticale et qu'on lance avec une vitesse angulaire initiale v_0. L'angle $\theta(t)$ est alors solution de l'équation différentielle :

$$\begin{cases} \theta''(t) = -\alpha\,\theta'(t) - \beta\,\sin(\theta(t)) \\ \theta(0) = 0, \quad \theta'(0) = v_0. \end{cases} \tag{1}$$

On définit la fonction $f : \mathbb{R}^2 \to \mathbb{R}$ par $f(x, y) = -\alpha y - \beta \sin(x)$.
Afin de trouver une solution approchée sur l'intervalle $[0, T]$ de (1) on découpe $[0, T]$ en N intervalles de taille $h = T/N$ et on calcule les valeurs des deux suites (a_n) et (b_n) définies par les relations

$$\begin{cases} a_0 = 0 \\ b_0 = v_0 \end{cases} \qquad \text{et} \qquad \forall n \geqslant 0 \qquad \begin{cases} a_{n+1} = a_n + h\,b_n \\ b_{n+1} = b_n + hf(a_n, b_n) \end{cases} \tag{2}$$

Nous admettrons que a_n est une valeur approchée de la solution $\theta(t_n)$ au temps $t_n = nh$. Dans la suite on fixe $\alpha = 0.8$ et $\beta = 1.1$.

1. Définir une fonction `f` d'arguments `x` et `y` qui renvoie $-0.8\,y - 1.1\sin(x)$.
2. Définir une fonction `U` d'un argument `t` qui renvoie $5\sqrt{\frac{2}{47}}\,\sin\left(\sqrt{\frac{47}{2}}\,\frac{t}{5}\right)e^{-\frac{2t}{5}}$. Tracer sa courbe representative sur l'intervalle $[0, 25]$.
3. Écrire une fonction `SolEDO` de trois arguments `v0`, `T` et `N`, qui renvoie la liste $(a_n)_{0\leqslant n\leqslant N}$ donnée par l'algorithme (2).
4. Tracer le nuage de points $(t_n, a_n)_{0\leqslant n\leqslant N}$ pour `v0`=1, `T`=25 et différentes valeurs de `N` que vous choisirez.
5. Vers quelle courbe semble se rapprocher le nuage de points ?
6. Reprendre la question précédente avec `v0`=4. Comment expliquer la différence de comportement en temps long des solutions obtenues ?
7. Déterminer une valeur approchée à 10^{-4} près de la valeur de $v_0 \in [1, 4]$ pour laquelle ce changement de comportement a lieu.

Exercice

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

On appelle sous-liste d'une liste $T = [t_0, \cdots, t_{n-1}]$ de longueur n toute liste de la forme $[t_{i_0}, \cdots, t_{i_k}]$ d'indices croissants ($0 \leqslant i_0 < i_1 < \cdots < i_k < n$). Cette sous-liste est dite *croissante* si ses éléments sont rangés par ordre croissant : $t_{i_0} \leqslant t_{i_1} \leqslant \cdots \leqslant t_{i_k}$.

1. Chercher à la main une sous-liste croissante de `[15,2,14,22,3,45]` de longueur maximale. Y a-t-il unicité ?
2. Écrire une fonction `glouton` d'argument une liste `T` renvoyant une sous-liste croissante de `T` construite de manière gloutonne : on prend le premier élément de `T`, puis le suivant s'il convient, et ainsi de suite.

   Vérifier que `glouton([15,2,14,22,3,45])` renvoie `[15,22,45]`.

On propose dans les questions suivantes un algorithme efficace pour *calculer la longueur maximale d'une sous-liste croissante.*

On construit une nouvelle liste $A = [a_0, \cdots, a_{n-1}]$ de la manière suivante :

$$a_i = \begin{cases} 1 & \text{si } t_j > t_i \text{ pour tout } j < i, \\ 1 + \max\limits_{j<i:\, t_j \leqslant t_i} a_j & \text{sinon.} \end{cases}$$

3. Calculer à la main la liste A obtenue pour la liste de la question 1.
4. Écrire une fonction `longueurs` d'argument T qui renvoie la liste A correspondante.
5. Écrire une fonction `longueurmax` d'argument T renvoyant la longueur maximale d'une sous-liste croissante de T.

   Vérifier que `longueurmax([15,2,14,22,3,45])` renvoie `[2,14,22,45]`.
6. Discuter de la complexité de votre fonction `longueurmax`.
7. Peut-on modéliser et résoudre ce problème d'un point de vue algorithmique dans un graphe particulier ?

Exercice

Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.

Dans un graphe non orienté $G = (V, E)$, la *coupe* définie par un sous-ensemble non vide $X \neq V$ de sommets est l'ensemble des arêtes ayant exactement une extrémité dans X et une dans $V \setminus X$. Une arête sera représentée par la liste de ses deux extrémités, et une coupe sera représentée par la liste des arêtes qui la composent. Les graphes seront représentés par listes d'adjacences.
Exemple : Dans le graphe G représenté par `[[1,2,4],[0],[0,4],[4],[0,2,3]]`, le sommet 0 est voisin des sommets 1, 2, 4, et la coupe définie par $\{0, 2\}$ est représentée par `[[0, 1], [0, 4], [2, 4]]`.

1. Écrire une fonction `coupe` d'argument un graphe `G` et un ensemble de sommets `X`, et qui renvoie la liste des arêtes de la coupe définie par `X`.
2. Importer la fonction `mystere` et la tester par les lignes de code :

```
from algo_code import mystere
print(mystere(4))
```

   La fonction `mystere` prend en entrée un entier `n` strictement positif. En la testant pour diverses valeurs, décrire ce qu'elle renvoie [1].
3. Écrire une fonction `liste_coupes` d'argument un graphe `G` qui renvoie la liste des coupes du graphe. On pourra utiliser la fonction `mystere`.
4. Écrire une fonction `coupe_min` d'argument un graphe `G` qui renvoie le nombre d'arêtes minimum dans une coupe du graphe.
   Vérifier que `coupe_min([[1,2,4],[0],[0,4],[4],[0,2,3]])` renvoie `1`.
5. En retirant du graphe ci-dessus une arête bien choisie, trouver un graphe `H` tel que `coupe_min(H)` renvoie `0`.
   Écrire une fonction `est_connexe` d'argument un graphe `G` qui renvoie `True` si le graphe est connexe, et `False` sinon.
6. Discuter la complexité de la fonction `est_connexe`, et éventuellement proposer une méthode plus efficace.

1. mystere(n) renvoie tous les sous-ensembles de $\{1, \ldots, n\}$ sauf $\emptyset$ et $\{1, \ldots, n\}$