DS 2 d'informatique

Corrigé

Problème I : Une intégrale classique

1. (a) Donner une équation différentielle simple vérifiée par f.

Une équation différentielle simple vérifiée par f est $\boxed{y' = -2ty.}$

(b) Donner une condition initiale simple vérifiée par f.

Une condition initiale simple vérifiée par f est $\boxed{y(0) = 1.}$

2. Écrire une fonction `fonctionF(a, N)`, où $a > 0$ et $N \in \mathbb{N}^*$, qui renvoie la liste des valeurs approchées de f aux `N+1` points du segment $[0, a]$ découpés en `N` intervalles égaux.
On procèdera en suivant la méthode d'Euler.

```python
def fonctionF(a, N):
    h = a/N
    Y = [1]
    y = 1
    t = 0
    for _ in range(N):
        y = y + h*(-2*t*y)
        Y.append(y)
        t = t+h

    return Y
```

3. Écrire une fonction `integrale(A, N)`, où $A > 0$ et $N \in \mathbb{N}^*$, qui renvoie une valeur approchée de $\int_0^A e^{-t^2}\,\mathrm{d}t$.

```
def integrale(A, N)
    h = A/N
    Y = fonctionF(A, N)
    S = 0
    for i in range(N):
        y0 = Y[i]
        y1 = Y[i+1]
        S = S + h*(y0+y1)/2

    return S
```

4. (a) Montrer que la fonction $\begin{cases} \mathbb{R}_+ \longrightarrow \mathbb{R} \\ x \longmapsto xe^{-x} \end{cases}$ atteint son maximum en un point qu'on précisera avec une valeur qu'on précisera.

On note g la fonction en question, qui est dérivable. Si $x \geqslant 0$, on a $g'(x) = (1-x)e^{-x}$. On en déduit le tableau de signes et variations suivants :

x	0		e		$+\infty$
$g'(x)$		$+$	0	$-$	
variations de g	0	$\nearrow$	$g(e)$	$\searrow$	0

Ainsi, $\boxed{\text{la fonction } g \text{ atteint son maximum en } e\,;\text{ elle y vaut } \frac{e}{e^e}.}$

(b) En admettant que $2 < e$, montrer que $\forall x \geqslant 0,\ |xe^{-x}| \leqslant \frac{1}{2}$.

Comme $e > 2$, on a $e^e > e^2$. Et donc, $\frac{e}{e^e} < \frac{1}{e} < \frac{1}{2}$. Ainsi, on a $\boxed{\forall x \geqslant 0,\ |xe^{-x}| \leqslant \frac{1}{2}.}$

(c) Montrer que $\forall x \in \mathbb{R}_+,\ |f''(x)| \leqslant 4$.

Soit $t \geqslant 0$. On calcule $f'(t) = -2te^{-t^2}$ et donc

$$f''(t) = -2e^{-t^2} + 4t^2e^{-t^2}.$$

Donc, si $t \geqslant 0$, on a

$$\begin{aligned} |f''(t)| &\leqslant 2\underbrace{\left|e^{-t^2}\right|}_{\leqslant 1} + 4\underbrace{t^2e^{-t^2}}_{\leqslant \frac{1}{2}} \\ &\leqslant 2 + 2 = \boxed{4.} \end{aligned}$$

5. Proposez une fonction `valeurApprochee(eps)`, où `eps` > 0, qui renvoie une valeur approchée de ℓ à `eps` près.

- Notons $I_n(A)$ la valeur approchée de $\displaystyle\int_0^A e^{-t^2}\,dt$ calculée par la méthode des trapèzes.
- On a

$$\begin{aligned}|I_n(A)-\ell| &\leqslant \left|I_n(A)-\int_0^A e^{-t^2}\,\mathrm{d}t\right| + \left|\int_0^A e^{-t^2}\,\mathrm{d}t-\ell\right| \\ &\leqslant \frac{A^3}{12n^2}\times 4 + e^{-A} \\ &\leqslant \frac{A^3}{3n^2}+2^{-A}. \qquad \text{(car } e>2\text{)}\end{aligned}$$

- On définit d'abord une fonction `AConvenable(eps)` qui renvoie un entier $A \geqslant 1$ tel que $2^{-A} < \dfrac{\varepsilon}{2}$.

```
def AConvenable(eps):
    A = 1
    puissance = 1/2
    while puissance > eps/2:
        A = A+1
        puissance = puissance/2
    return A
```

- Puis, on définit une fonction `NConvenable(A, eps)` qui renvoie un entier n vérifiant $\dfrac{A^3}{3n^2} \leqslant \dfrac{\varepsilon}{2}$. Comme on a

$$\frac{A^3}{3n^2} \leqslant \frac{\varepsilon}{2} \iff n^2 \geqslant \frac{2A^3}{3\varepsilon} \iff n \geqslant \sqrt{\frac{2A^3}{3\varepsilon}},$$

on écrit :

```
import math as mt

def NConvenable(A, eps):
    x = mt.sqrt(2*A**3/(2*eps))
    return int(x)+1
```

- Finalement, voici le code proposé pour la fonction cherchée :

```
def valeurApprochee(eps):
    A = AConvenable(eps)
    N = NConvenable(A, eps)
    return integrale(A, N)
```

6. Démontrer les résultats admis (1) et (2).

- Déjà, la fonction $g : A \longmapsto \int_0^A e^{-t^2}\,\mathrm{d}t$ est croissante, puisque la fonction intégrée est $\geqslant 0$.
- Donc, $\lim\limits_{A\to+\infty} g(A)$ existe et est soit finie soit égale à $+\infty$.
- Or, si $t \geqslant 1$, on a $t^2 \geqslant t$ et donc $e^{-t^2} \leqslant e^{t}$. Donc, pour $A \geqslant 1$, on a

$$\begin{aligned}
\int_0^A e^{-t^2}\,\mathrm{d}t &= \int_0^1 e^{-t^2}\,\mathrm{d}t + \int_1^A e^{-t^2}\,\mathrm{d}t \\
&\leqslant \int_0^1 e^{-t^2}\,\mathrm{d}t + \int_1^A e^{-t}\,\mathrm{d}t \\
&= \int_0^1 e^{-t^2}\,\mathrm{d}t + (1 - e^{-A}) \\
&\leqslant \int_0^1 e^{-t^2}\,\mathrm{d}t + 1
\end{aligned}$$

- Ainsi, la fonction g est majorée. Donc, elle admet une limite finie :

$$\boxed{\text{la limite } \ell := \lim_{A\to+\infty} \int_0^A e^{-t^2}\,\mathrm{d}t \text{ existe et est finie.}}$$

- De plus, par croissance de g, pour tout $A \geqslant 0$, on a $\boxed{\int_0^A e^{-t^2}\,\mathrm{d}t \leqslant \ell.}$
- On a aussi, si $T \geqslant A \geqslant 1$, on a

$$\int_A^T e^{-t^2}\,\mathrm{d}t \leqslant \int_A^T e^{-t}\,\mathrm{d}t = e^{-A} - e^{-T}.$$

Quand on fait tendre $T \to +\infty$, on obtient

$$\lim_{T\to+\infty} \int_A^T e^{-t^2}\,\mathrm{d}t \leqslant e^{-A}.$$

- Donc, on a, si $A \geqslant 1$:

$$\begin{aligned}
0 \leqslant \ell - \int_0^A e^{-t^2}\,\mathrm{d}t &= \lim_{T\to+\infty} \int_0^T e^{-t^2}\,\mathrm{d}t - \int_0^A e^{-t^2}\,\mathrm{d}t \\
&= \lim_{T\to+\infty} \int_A^T e^{-t^2}\,\mathrm{d}t \\
&\leqslant \boxed{e^{-A}.}
\end{aligned}$$

Problème II : Enveloppes convexes dans le plan

Partie I – Questions préliminaires

1. Écrire une fonction `plusBas(tab)` qui prend en paramètre un tableau `tab` de taille $2 \times n$ et qui renvoie l'indice j du point le plus bas (c'est-à-dire de plus petite ordonnée) parmi les points du nuage P. En cas d'égalité, votre fonction devra renvoyer l'indice du point de plus petite abscisse parmi les points les plus bas.

```
def plusBas(tab):
    n = len(tab[0])
    jmin = 0
    ordmin = tab[1][jmin]
    abscis = tab[0][jmin]

    for j in range(1, n):
        if tab[1][j] < ordmin or (tab[1][j] == ordmin and tab
           [0][j] < abscis):
            jmin = j
            ordmin = tab[1][jmin]
            abscis = tab[0][jmin]

    return jmin
```

2. Sur le tableau précédent, donner le résultat du test d'orientation pour les choix d'indices suivantes :

(a) $i = 7, j = 3, k = 4$

Dans ce cas, on a $\overrightarrow{p_i p_j} \begin{pmatrix} 4 \\ 2 \end{pmatrix}$ et $\overrightarrow{p_i p_k} \begin{pmatrix} 4 \\ 5 \end{pmatrix}$. Donc

$$\frac{1}{2} \det\left(\overrightarrow{p_i p_j}, \overrightarrow{p_i p_k}\right) = \frac{1}{2} \det \begin{pmatrix} 4 & 4 \\ 2 & 5 \end{pmatrix} = \frac{4 \times 5 - 2 \times 4}{2} = \frac{12}{2} = 6 > 0.$$

Donc, $\boxed{\text{le triplet } (p_7, p_3, p_4) \text{ a une orientation positive.}}$

(b) $i = 8, j = 9, k = 10$

Dans ce cas, on a $\overrightarrow{p_i p_j} \begin{pmatrix} 1 \\ 3 \end{pmatrix}$ et $\overrightarrow{p_i p_k} \begin{pmatrix} 4 \\ 4 \end{pmatrix}$. Donc

$$\frac{1}{2} \det\left(\overrightarrow{p_i p_j}, \overrightarrow{p_i p_k}\right) = \frac{1}{2} \det \begin{pmatrix} 1 & 4 \\ 3 & 4 \end{pmatrix} = \frac{4 \times 1 - 3 \times 4}{2} = -\frac{8}{2} = -4 < 0.$$

Donc, $\boxed{\text{le triplet } (p_8, p_9, p_{10}) \text{ a une orientation négative.}}$

3. Écrire une fonction `orient(tab, i, j, k)` qui prend en paramètres le tableau `tab` et trois indices de colonnes, potentiellement égaux, et qui renvoie le résultat -1, 0, ou $+1$ du test d'orientation sur la séquence (p_i, p_j, p_k) de points de P.

```
def vecteur(tab, i, j):
    abscisses = tab[0]
    ordonnees = tab[1]
    xVecteur = abscisses[j] - abscisses[i]
    yVecteur = ordonnees[j] - ordonnees[i]

    return [xVecteur, yVecteur]

def orient(tab, i, j, k):
    PiPj = vecteur(tab, i, j)
    PiPk = vecteur(tab, i, k)
    aire = PiPj[0]*PiPk[1] - PiPj[1]*PiPk[0]
    if aire > 0:
        return 1
    elif aire < 0:
        return -1
    else:
        return 0
```

Partie II – L'algorithme du paquet cadeau

4. Décrire une réalisation en `Python` de la procédure. Elle prendra la forme d'une fonction `prochainPoint(tab, i)` qui prend en paramètre le tableau `tab` de taille $2 \times n$ ainsi que l'indice `i` du point inséré en dernier dans le paquet cadeau, et qui renvoie l'indice du prochain point à insérer.

```
def prochainPoint(tab, i):
    n = len(tab[0])
    L = [j for j in range(n) if j != i]
    imax = L[0]
    for l in L[1:]:
        if orient(tab, i, imax, l) == -1:
            imax = l

    return imax
```

5. Décrire à la main le déroulement de la procédure `prochainPoint` sur l'exemple de la figure. Plus précisément, indiquer la séquence de points de $P \setminus \{p_{10}\}$ considérés et la valeur de l'indice du maximum à chaque itération.

- On commence avec `10`.
- Puis, l'indice renvoyé par `prochainPoint(tab, 10)` est `5`.
- Puis, l'indice renvoyé est `2`.
- Puis, l'indice renvoyé est `0`.
- Puis, l'indice renvoyé est `7`.
- On considère alors qu'on a terminé.

6. Écrire une fonction `convJarvis(tab)` qui prend en paramètre le tableau `tab` de taille $2 \times n$ représentant le nuage P, et qui renvoie une liste contenant les indices des sommets du bord de l'enveloppe convexe de P, sans doublon. Le temps d'exécution de votre fonction doit être majoré par une constante fois nm, où m est le nombre de points de P situés sur le bord de $\mathrm{Conv}(P)$.

```
def convJarvis(tab):
    premierpoint = plusBas(tab)
    L = [premierpoint]
    prochain = prochainPoint(tab, premierpoint)
    while prochain != premierpoint:
        L.append(prochain)
        prochain = prochainPoint(tab, prochain)
    return L
```

7. Justifier brièvement le temps d'exécution de l'algorithme du paquet cadeau.

Déjà, la fonction `plusBas` a un coût temporel en $\mathrm{O}(\mathtt{n})$ et n'est utilisée qu'une fois.
La fonction précédente réalise une boucle de `m`tours. À l'intérieur de chaque tour de boucle, on fait appel à la fonction `prochainPoint` qui a un temps d'exécution en $\mathrm{O}(\mathtt{n})$.
Donc, $\boxed{\text{la complexité de cette fonction est en } \mathrm{O}(\mathtt{m} \times \mathtt{n}).}$

Partie III – Algorithme de balayage

8. Écrire une fonction `majES(tab, es, i)` qui prend en paramètre le tableau `tab` ainsi que la pile `es` et l'indice `i` du point à visiter, et qui met à jour l'enveloppe supérieure du sous-nuage. Le temps d'exécution de votre fonction doit être majoré par une constante fois `i`.

```
def majES(tab, es, i):
    ''' es supposée non vide '''
    p1 = pop(es)
    if isEmpty(es):    # on a qu'un élément dans la pile
        push(p1, es)
        push(i, es)
        return  # on s'arrête là

    p2 = top(es)
    while not isEmpty(es) and orient(tab, i, p1, p2) == -1:
        p1 = pop(es)    # on dépile p1
        if not isEmpty(es):
            p2 = top(es)   # on met à jour p2

    push(p1, es)   # que es soit vide ou que l'orientation soit bonne
                   # il faut remettre p1

    push(i, es)
```

9. Écrire une fonction `majEI(tab, ei, i)` qui effectue la mise à jour de l'enveloppe inférieure, avec le même temps d'exécution.

```
def majEI(tab, ei, i):
    ''' ei supposée non vide '''
    p1 = pop(ei)
    if isEmpty(ei):
        push(p1, ei)
        push(i, ei)
        return

    p2 = top(ei)
    while not isEmpty(ei) and orient(tab, i, p1, p2) == 1:
        p1 = pop(ei)
        if not isEmpty(ei):
            p2 = top(ei)

    push(p1, ei)

    push(i, ei)
```

10. Écrire maintenant une fonction `convGraham(tab)` qui prend en paramètre le tableau `tab` de taille $2 \times n$ représentant le nuage P, et qui effectue le balayage des points de P comme décrit précédemment. On supposera les colonnes du tableau `tab` déjà triées par ordre croissant d'abscisse. La fonction doit renvoyer une pile `s` contenant les indices des sommets du bord de Conv(P) triés dans l'ordre positif d'orientation, à commencer par le point p_0.

```
def convGraham(tab, n):
    es = newStack()
    ei = newStack()
    push(0, es)
    push(0, ei)
    for i in range(1, n):
        majES(tab, es, i)
        majEI(tab, ei, i)

    pop(es)    # on élimine le point le plus à droite
               # redondant dans les deux piles
    while not isEmpty(es):
        push(pop(es), ei)

    pop(ei)   # on ne rajoute pas le premier point de l'enveloppe
    return ei
```