



PRÉPASIMPLE

prepasimple.fr

EDHEC

Mathématiques

2017

Corrigé de mathématiques approfondies

Mathématiques approfondies — Prépa ECG

Annales de concours · Corrigé

Exercice 1

1) a) Code adapté en Python. Question adaptée au programme actuel.

Le tableau `np.arange(1, n + 1)` contient les entiers de 1 à n , car la borne supérieure de `arange` est exclue. L'élévation de x aux puissances de ce tableau est effectuée coefficient par coefficient. On somme ensuite ces puissances :

Python

```
import numpy as np

def f(x, n):
    y = np.sum(x ** np.arange(1, n + 1))
    return y
```

Les paramètres satisfont $x \in [0, 1]$ et $n \geq 1$. La fonction renvoie

$$\sum_{k=1}^n x^k = f_n(x).$$

Le code a été exécuté et comparé à une somme calculée par une boucle pour plusieurs couples (x, n) .

b) Code adapté en Python. Question adaptée au programme actuel.

Si $x \neq 1$, la formule de la somme géométrique finie donne :

$$f_n(x) = x \sum_{k=0}^{n-1} x^k = \frac{x(1 - x^n)}{1 - x}.$$

Pour $x = 1$, chacun des n termes vaut 1, donc $f_n(1) = n$. La deuxième déclaration est :

Python

```
def f(x, n):
    if x == 1:
        y = n
    else:
        y = x * (1 - x**n) / (1 - x)
    return y
```

La branche `if` évite toute division par 0. Les valeurs renvoyées ont été contrôlées par comparaison avec la première version, `y` compris pour $x = 0$ et $x = 1$.

- 2) La fonction f_n est polynomiale, donc continue sur $[0, 1]$. On a $f_n(0) = 0$ et $f_n(1) = n \geq 1$. Le théorème des valeurs intermédiaires assure l'existence d'une solution.

Pour $x \in [0, 1]$:

$$f'_n(x) = \sum_{k=1}^n kx^{k-1} \geq 1.$$

Ainsi f_n est strictement croissante sur $[0, 1]$, ce qui assure l'unicité. Il existe donc un unique $\alpha_n \in [0, 1]$ tel que $f_n(\alpha_n) = 1$.

Comme $f_n(0) = 0$, on a $\alpha_n > 0$. Pour $n = 1$, $f_1(x) = x$, donc $\alpha_1 = 1$; pour $n \geq 2$, $f_n(1) = n > 1$, donc $\alpha_n < 1$.

- 3) a) La relation entre deux sommes consécutives donne :

$$f_{n+1}(\alpha_n) = f_n(\alpha_n) + \alpha_n^{n+1} = 1 + \alpha_n^{n+1} > 1.$$

Or $f_{n+1}(\alpha_{n+1}) = 1$. Par stricte croissance de f_{n+1} , on en déduit $\alpha_{n+1} < \alpha_n$. La suite est donc strictement décroissante, en particulier décroissante comme demandé.

b) Tous les α_n sont positifs. La suite est décroissante et minorée par 0 ; le théorème de la limite monotone assure sa convergence vers un réel $\ell \in [0, 1]$.

4) a) L'équation $f_2(x) = 1$ s'écrit $x + x^2 = 1$, soit $x^2 + x - 1 = 0$. Les deux solutions réelles sont $(-1 - \sqrt{5})/2$ et $(-1 + \sqrt{5})/2$. Seule la seconde appartient à $[0, 1]$:

$$\alpha_2 = \frac{\sqrt{5} - 1}{2}.$$

Comme $1 < \sqrt{5} < 3$, on a $0 < \alpha_2 < 1$.

b) Pour $n \geq 2$, la décroissance donne $0 < \alpha_n \leq \alpha_2 < 1$. En prenant les puissances positives $n + 1$:

$$0 \leq \alpha_n^{n+1} \leq \alpha_2^{n+1}.$$

La suite géométrique de raison α_2 tend vers 0. Par encadrement :

$$\lim_{n \rightarrow +\infty} \alpha_n^{n+1} = 0.$$

Il serait insuffisant d'utiliser seulement $\alpha_n < 1$; l'encadrement par une constante strictement inférieure à 1 justifie la limite malgré la variation de la base.

c) Pour $n \geq 2$, $\alpha_n \neq 1$. La formule de 1) b) permet donc d'écrire :

$$1 = \frac{\alpha_n(1 - \alpha_n^n)}{1 - \alpha_n}.$$

En multipliant par $1 - \alpha_n$, il vient $1 - \alpha_n = \alpha_n - \alpha_n^{n+1}$, donc :

$$2\alpha_n = 1 + \alpha_n^{n+1}.$$

D'après b), le terme de droite tend vers 1. Ainsi :

$$\lim_{n \rightarrow +\infty} \alpha_n = \frac{1}{2}.$$

Cette identité donne aussi, pour $n \geq 2$, $0 < \alpha_n - \frac{1}{2} \leq \frac{1}{2}\alpha_2^{n+1}$.

5) Code adapté en Python. Question adaptée au programme actuel.

En arithmétique exacte, après l'itération d'indice k , la variable x vaut $k/1000$. La boucle s'arrête au premier entier k tel que $f_n(k/1000) \geq 1$. Puisque f_n est strictement croissante, cela équivaut à $k/1000 \geq \alpha_n$.

L'arrêt existe au plus tard pour $k = 1000$, puisque $f_n(1) = n \geq 1$. De plus, la boucle a été exécutée au moins une fois puisque $f_n(0) = 0$. À la dernière valeur précédente, on avait encore $f_n((k-1)/1000) < 1$, donc :

$$\frac{k-1}{1000} < \alpha_n \leq \frac{k}{1000}.$$

Le résultat affiché est ainsi une approximation de α_n par excès à moins de 10^{-3} :

$$0 \leq x - \alpha_n < 10^{-3}.$$

C'est la plus petite valeur de la grille $\{0, 0.001, \dots, 1\}$ supérieure ou égale à α_n . Si α_n appartient à la grille, la valeur affichée lui est égale.

Le code complet, utilisant la première version de f , est :

Python

```
import numpy as np

def f(x, n):
    return np.sum(x ** np.arange(1, n + 1))

n = int(input('entrer la valeur de n : '))
k = 0
x = 0.0
while f(x, n) < 1:
    k = k + 1
    x = k / 1000
print(x)
```

Le compteur entier évite l'accumulation des erreurs liée à des additions répétées de 0.001. Les évaluations de f restent néanmoins des calculs à virgule flottante : la borne démontrée concerne l'algorithme en arithmétique exacte. Pour de grands n , une différence inférieure à la précision de la machine peut être arrondie à 0 lors de la comparaison.

Code exécuté pour $n = 1, 2, 3, 10$: les résultats affichés sont respectivement 1.0, 0.619, 0.544 et 0.501.

Exercice 2

1) a) Pour tout réel x , on a :

$$\{M_n \leq x\} = \bigcap_{i=1}^n \{X_i \leq x\}.$$

Les X_i étant indépendantes, ces événements sont indépendants. Leur fonction de répartition commune est 0 pour $x < 0$, x pour $0 \leq x \leq 1$, et 1 pour $x > 1$. Ainsi :

$$F_{M_n}(x) = \begin{cases} 0 & \text{si } x < 0, \\ x^n & \text{si } 0 \leq x \leq 1, \\ 1 & \text{si } x > 1. \end{cases}$$

Cette fonction est continue sur $\mathbb{R}$, y compris en 0 et en 1, et de classe C^1 sauf éventuellement en ces deux points. La caractérisation des variables à densité par leur fonction de répartition montre que M_n est à densité.

- b)** On dérive la fonction de répartition sur chacun des intervalles ouverts et on choisit arbitrairement la valeur 0 aux extrémités :

$$f_{M_n}(x) = \begin{cases} nx^{n-1} & \text{si } 0 < x < 1, \\ 0 & \text{sinon.} \end{cases}$$

Cette fonction est positive et son intégrale vaut $\int_0^1 nx^{n-1} dx = 1$. C'est bien une densité de M_n .

- c)** Puisque $0 \leq M_n \leq 1$ presque sûrement, les variables M_n et M_n^2 sont bornées et admettent des espérances. Par transfert :

$$E(M_n) = n \int_0^1 x^n dx = \frac{n}{n+1}, \quad E(M_n^2) = n \int_0^1 x^{n+1} dx = \frac{n}{n+2}.$$

d) Méthode 1 — Inégalité de Markov

La variable $W_n = (M_n - 1)^2$ est positive et bornée. L'inégalité de Markov appliquée au seuil $\varepsilon^2 > 0$ donne :

$$P((M_n - 1)^2 \geq \varepsilon^2) \leq \frac{E((M_n - 1)^2)}{\varepsilon^2}.$$

Par linéarité de l'espérance et les moments calculés en c) :

$$E((M_n - 1)^2) = \frac{n}{n+2} - \frac{2n}{n+1} + 1 = \frac{2}{(n+1)(n+2)}.$$

Ainsi, pour tout $\varepsilon > 0$:

$$P((M_n - 1)^2 \geq \varepsilon^2) \leq \min \left(1, \frac{2}{(n+1)(n+2)\varepsilon^2} \right).$$

Cette méthode réutilise les moments et fournit une majoration valable pour tout seuil sans calculer exactement la probabilité.

Méthode 2 — Calcul exact par la fonction de répartition

Comme $M_n \in [0, 1]$ presque sûrement, $|M_n - 1| = 1 - M_n$. Pour $0 < \varepsilon < 1$:

$$P((M_n - 1)^2 \geq \varepsilon^2) = P(M_n \leq 1 - \varepsilon) = (1 - \varepsilon)^n.$$

Pour $\varepsilon = 1$, cette probabilité vaut $P(M_n = 0) = 0$, et pour $\varepsilon > 1$, elle est nulle. Ainsi :

$$P((M_n - 1)^2 \geq \varepsilon^2) = \begin{cases} (1 - \varepsilon)^n & \text{si } 0 < \varepsilon < 1, \\ 0 & \text{si } \varepsilon \geq 1. \end{cases}$$

Le calcul exact est plus précis, mais il nécessite de connaître la fonction de répartition de M_n .

- e) Pour tout $\varepsilon > 0$, les événements $\{|M_n - 1| \geq \varepsilon\}$ et $\{(M_n - 1)^2 \geq \varepsilon^2\}$ sont égaux. Le majorant de Markov tend vers 0 lorsque $n \rightarrow +\infty$; par encadrement :

$$\lim_{n \rightarrow +\infty} P(|M_n - 1| \geq \varepsilon) = 0.$$

C'est exactement la convergence en probabilité de M_n vers la constante 1 :

$$M_n \xrightarrow{P} 1.$$

La probabilité que le maximum s'écarte de 1 d'au moins une quantité fixée tend donc vers 0.

2) a) Code adapté en Python. Question adaptée au programme actuel.

Le tableau X contient une simulation des n variables uniformes indépendantes. Son maximum simule M_n ; on applique ensuite la transformation $n(1 - M_n)$:

Python

```
import numpy as np
import numpy.random as rd

def f(n):
    X = rd.random(n)
    Y = n * (1 - np.max(X))
    return Y
```

La variable renvoyée appartient à $[0, n]$ dans le modèle probabiliste. Le code a été exécuté et ses sorties ont été contrôlées pour plusieurs valeurs de n .

b) Code adapté en Python. Question adaptée au programme actuel.

Les histogrammes ont des formes très proches : une fréquence importante dans la première classe, puis une décroissance rapide. On conjecture que :

$$Y_n \xrightarrow{\mathcal{L}} Y, \quad Y \sim \mathcal{E}(1).$$

Les scripts complets permettant la comparaison sont :

Python

```
import numpy as np

import numpy.random as rd

import matplotlib.pyplot as plt

# Script (1)

e = rd.exponential(scale=1.0, size=10000)

s = np.linspace(0, 10, 11)

plt.hist(e, bins=s, weights=np.ones(10000) / 10000,
         color='white', edgecolor='black')

plt.show()

# Fonction et script (2)

def f(n):

    X = rd.random(n)

    return n * (1 - np.max(X))

n = int(input('entrez la valeur de n : '))

Y = np.zeros(10000)

for k in range(10000):

    Y[k] = f(n)

s = np.linspace(0, 10, 11)

plt.hist(Y, bins=s, weights=np.ones(10000) / 10000,
         color='white', edgecolor='black')

plt.show()
```

La loi exponentielle NumPy est paramétrée par `scale`, qui vaut l'inverse du taux : un taux de 1 correspond ici à `scale=1.0`. La boucle effectue 10000 appels utilisant de nouveaux tirages indépendants ; chaque sortie est rangée dans une case de `Y`.

Les deux scripts ont été exécutés, le second avec $n = 1000$. Les graphiques simulés varient d'une exécution à l'autre ; les figures du sujet reproduisent les données graphiques du PDF d'origine. La ressemblance observée suggère la convergence en loi, mais elle ne constitue pas une démonstration.

- 3) a) On a $0 \leq Y_n \leq n$ presque sûrement. Pour $x < 0$, $F_{Y_n}(x) = 0$; pour $x \geq n$, $F_{Y_n}(x) = 1$.

Pour $0 \leq x < n$, en utilisant la transformation décroissante et l'absence d'atomes de M_n :

$$F_{Y_n}(x) = P(n(1 - M_n) \leq x) = P\left(M_n \geq 1 - \frac{x}{n}\right) = 1 - \left(1 - \frac{x}{n}\right)^n.$$

Ainsi :

$$F_{Y_n}(x) = \begin{cases} 0 & \text{si } x < 0, \\ 1 - \left(1 - \frac{x}{n}\right)^n & \text{si } 0 \leq x < n, \\ 1 & \text{si } x \geq n. \end{cases}$$

En particulier, $F_{Y_n}(0) = 0$.

- b) Si $x = 0$, la limite est $0 = 1 - e^0$. Fixons désormais $x > 0$. Pour tout $n > x$, la formule centrale de a) s'applique et $1 - x/n > 0$.

Le développement limité $\ln(1 - u) = -u + o(u)$ lorsque $u \rightarrow 0$ donne :

$$n \ln\left(1 - \frac{x}{n}\right) \rightarrow -x.$$

En exponentiant, $(1 - x/n)^n \rightarrow e^{-x}$. Par conséquent :

$$\forall x \geq 0, \quad \lim_{n \rightarrow +\infty} F_{Y_n}(x) = 1 - e^{-x}.$$

La limite est calculée pour chaque x fixé ; ce n'est pas une approximation uniforme supposée sur un intervalle variable.

- c) Pour $x < 0$, toutes les fonctions de répartition valent 0. Avec b), pour tout réel x , leur limite est donc :

$$F_Y(x) = \begin{cases} 0 & \text{si } x < 0, \\ 1 - e^{-x} & \text{si } x \geq 0. \end{cases}$$

C'est la fonction de répartition d'une loi exponentielle de taux 1, continue sur tout $\mathbb{R}$. La convergence des fonctions de répartition en tous ses points de continuité donne :

$$Y_n \xrightarrow{\mathcal{L}} Y, \quad Y \sim \mathcal{E}(1).$$

Le résultat conjecturé est ainsi démontré.

Exercice 3

- 1) a) Les coefficients hors diagonale de J valent 1. Pour obtenir $-n$ sur la diagonale, il faut retrancher $n + 1$ à chaque coefficient diagonal. Ainsi :

$$A = J - (n + 1)I.$$

Chaque coefficient de J^2 est la somme de n termes égaux à 1, donc $J^2 = nJ$. Puisque I commute avec J :

$$A^2 = J^2 - 2(n + 1)J + (n + 1)^2I = -(n + 2)J + (n + 1)^2I.$$

b) En remplaçant J par $A + (n + 1)I$ dans cette expression :

$$A^2 = -(n + 2)A - (n + 1)I.$$

Le polynôme $Q(X) = X^2 + (n + 2)X + n + 1 = (X + 1)(X + n + 1)$ est donc un polynôme annulateur de A .

Si $AX = \lambda X$ pour un vecteur $X \neq 0$, alors $Q(A)X = Q(\lambda)X = 0$, d'où $Q(\lambda) = 0$. Les seules valeurs propres possibles sont :

$$\lambda = -1 \quad \text{ou} \quad \lambda = -(n + 1).$$

Ce raisonnement repose sur le polynôme annulateur calculé explicitement, sans aucun résultat supplémentaire de réduction matricielle.

c) Méthode 1 — Inverse obtenu par le polynôme annulateur

La relation précédente donne :

$$A(A + (n + 2)I) = -(n + 1)I.$$

Comme $n + 1 \neq 0$ et que les deux facteurs commutent, la matrice

$B = -\frac{1}{n + 1}(A + (n + 2)I)$ vérifie $AB = BA = I$. Ainsi A est inversible, avec :

$$A^{-1} = -\frac{A + (n + 2)I}{n + 1} = -\frac{J + I}{n + 1}.$$

Cette méthode donne l'inverse en même temps que l'inversibilité.

Méthode 2 — Noyau de la matrice

Supposons $AX = 0$ et posons

$$S = \sum_{i=1}^n x_i,$$

où $X = {}^t(x_1, \dots, x_n)$. L'égalité $AX = JX - (n+1)X = 0$ donne $S - (n+1)x_i = 0$ pour tout i . En sommant ces n égalités, $nS - (n+1)S = 0$, donc $S = 0$. On en déduit $x_i = 0$ pour tout i .

Le noyau est réduit à $\{0\}$; pour une matrice carrée, le théorème du rang assure alors l'inversibilité. Cette seconde méthode s'appuie directement sur le système linéaire.

- 2) La base $(\varepsilon_0, \dots, \varepsilon_n)$ est orthonormale. Le carré de la norme de u est donc la somme des carrés de ses coordonnées :

$$\|u\|^2 = \frac{1}{n+1} \sum_{k=0}^n \|\varepsilon_k\|^2 = \frac{n+1}{n+1} = 1.$$

Ainsi $\|u\| = 1$. On utilisera aussi :

$$\forall i \in \llbracket 0, n \rrbracket, \quad \langle \varepsilon_i, u \rangle = \frac{1}{\sqrt{n+1}}.$$

- 3) a) Posons $c = \sqrt{(n+1)/n}$ et $a = 1/\sqrt{n+1}$. Alors $e_i = c(\varepsilon_i - au)$, avec $\langle \varepsilon_i, u \rangle = a$ et $\|u\| = 1$. En développant le carré de la norme :

$$\|e_i\|^2 = c^2(\|\varepsilon_i\|^2 - 2a\langle \varepsilon_i, u \rangle + a^2\|u\|^2) = \frac{n+1}{n}(1 - a^2) = 1.$$

Par conséquent, $\|e_i\| = 1$ pour tout i .

b) Pour $i \neq j$, l'orthonormalité donne $\langle \varepsilon_i, \varepsilon_j \rangle = 0$. Ainsi :

$$\langle e_i, e_j \rangle = c^2(0 - a\langle \varepsilon_i, u \rangle - a\langle u, \varepsilon_j \rangle + a^2\|u\|^2) = c^2(-a^2) = -\frac{1}{n}.$$

c) Pour tout i , on calcule :

$$\langle e_i, u \rangle = c(\langle \varepsilon_i, u \rangle - a\|u\|^2) = c(a - a) = 0.$$

Ainsi chaque e_i est orthogonal à u et appartient à $F = (\text{Vect}(u))^\perp$.

d) Comme u est non nul et $\dim E = n + 1$, on a $\dim F = n$. Il suffit donc de montrer que les n vecteurs $e_1, \dots, e_n$ sont libres.

Supposons

$$\sum_{i=1}^n a_i e_i = 0.$$

En prenant le produit scalaire avec chaque e_j , pour $1 \leq j \leq n$, on obtient :

$$a_j - \frac{1}{n} \sum_{i \neq j} a_i = 0.$$

La matrice de ce système est $G = (\langle e_i, e_j \rangle)_{1 \leq i, j \leq n}$, de diagonale 1 et de coefficients hors diagonale $-1/n$. Avec la matrice de la question 1), on a $G = -A/n$.

La matrice A est inversible par 1) c), donc G l'est également. Le seul vecteur de coefficients solution est nul : $a_1 = \dots = a_n = 0$. La famille est libre et comporte n vecteurs de F , dont la dimension est n ; c'est une base de F .

4) a) Question adaptée au programme actuel : la définition de « forme bilinéaire symétrique » est explicitée dans le sujet.

Fixons $y \in F$. Pour $x, z \in F$ et $\alpha, \beta \in \mathbb{R}$, la bilinéarité du produit scalaire donne :

$$f(\alpha x + \beta z, y) = \sum_{k=0}^n (\alpha \langle x, e_k \rangle + \beta \langle z, e_k \rangle) \langle y, e_k \rangle - \frac{n+1}{n} (\alpha \langle x, y \rangle + \beta \langle z, y \rangle).$$

En regroupant, $f(\alpha x + \beta z, y) = \alpha f(x, y) + \beta f(z, y)$. La même opération dans la seconde variable montre sa linéarité lorsque la première est fixée. Ainsi f est bilinéaire.

Enfin, la commutativité du produit réel et la symétrie du produit scalaire donnent $f(x, y) = f(y, x)$. C'est donc une forme bilinéaire symétrique.

b) Si $i = j$, le terme d'indice $k = i$ vaut 1 et les n autres termes valent $1/n^2$. Ainsi :

$$f(e_i, e_i) = 1 + \frac{n}{n^2} - \frac{n+1}{n} \times 1 = 0.$$

Si $i \neq j$, les termes d'indices $k = i$ et $k = j$ valent chacun $-1/n$. Les $n - 1$ autres termes valent $1/n^2$. Par conséquent :

$$f(e_i, e_j) = -\frac{2}{n} + \frac{n-1}{n^2} - \frac{n+1}{n} \left(-\frac{1}{n}\right) = \frac{-2n + n - 1 + n + 1}{n^2} = 0.$$

Dans les deux cas, $f(e_i, e_j) = 0$.

c) Méthode 1 — Bilinéarité et base de F

La famille $(e_1, \dots, e_n)$ est une base de F . Écrivons

$$x = \sum_{i=1}^n a_i e_i$$

et

$$y = \sum_{j=1}^n b_j e_j.$$

Par bilinéarité :

$$f(x, y) = \sum_{i=1}^n \sum_{j=1}^n a_i b_j f(e_i, e_j) = 0,$$

puisque tous les termes $f(e_i, e_j)$ sont nuls. En revenant à la définition de f :

$$\sum_{k=0}^n \langle x, e_k \rangle \langle y, e_k \rangle = \frac{n+1}{n} \langle x, y \rangle.$$

Cette méthode suit l'organisation du sujet : une forme bilinéaire est déterminée par ses valeurs sur les couples de vecteurs d'une base.

Méthode 2 — Coordonnées dans la base orthonormale initiale

Pour $x, y \in F$, on a $\langle x, u \rangle = \langle y, u \rangle = 0$. La définition de e_k donne alors :

$$\langle x, e_k \rangle = \sqrt{\frac{n+1}{n}} \langle x, \varepsilon_k \rangle, \quad \langle y, e_k \rangle = \sqrt{\frac{n+1}{n}} \langle y, \varepsilon_k \rangle.$$

Ainsi :

$$\sum_{k=0}^n \langle x, e_k \rangle \langle y, e_k \rangle = \frac{n+1}{n} \sum_{k=0}^n \langle x, \varepsilon_k \rangle \langle y, \varepsilon_k \rangle = \frac{n+1}{n} \langle x, y \rangle.$$

La dernière égalité est la formule du produit scalaire en base orthonormale. Cette preuve éclaire directement le rôle de la projection sur $u^\perp$ dans la construction des e_k .

d) On prend $y = x$ dans c). On obtient

$$\sum_{k=0}^n \langle x, e_k \rangle^2 = \frac{n+1}{n} \|x\|^2,$$

puis :

$$\|x\|^2 = \frac{n}{n+1} \sum_{k=0}^n \langle x, e_k \rangle^2.$$

Problème

Partie 1 : étude d'une application définie sur $\mathbb{R}_n[X]$

1) Pour chaque $k \in \llbracket 0, n \rrbracket$, la dérivation k fois est une opération linéaire, et $P^{(k)}$ est un polynôme de degré au plus n , ou le polynôme nul. La somme $\varphi(P)$ appartient donc à $\mathbb{R}_n[X]$.

Pour tous $P, Q \in \mathbb{R}_n[X]$ et $a, b \in \mathbb{R}$:

$$\varphi(aP + bQ) = \sum_{k=0}^n (aP^{(k)} + bQ^{(k)}) = a\varphi(P) + b\varphi(Q).$$

Ainsi φ est un endomorphisme de $\mathbb{R}_n[X]$.

2) a) Le polynôme $e_0 = 1$ a toutes ses dérivées d'ordre au moins 1 nulles. Ainsi $\varphi(e_0) = e_0$.

Comme $e_0 \neq 0$, le réel 1 est une valeur propre de φ , avec vecteur propre associé e_0 .

b) Pour $j \in \llbracket 1, n \rrbracket$, les dérivées d'ordre supérieur à j du polynôme X^j sont nulles. On a :

$$\varphi(e_j) - e_j = \sum_{k=1}^j \frac{j!}{(j-k)!} X^{j-k}.$$

Chaque monôme a un degré inférieur ou égal à $j-1$. Par conséquent,

$$\varphi(e_j) - e_j \in \mathbb{R}_{j-1}[X].$$

c) D'après b), l'image de e_j est une combinaison de $e_0, \dots, e_j$, avec coefficient 1 devant e_j . La matrice de φ dans $\mathcal{B}$ est donc triangulaire supérieure et tous ses coefficients diagonaux sont égaux à 1.

Les valeurs propres d'une matrice triangulaire sont ses coefficients diagonaux. La seule valeur propre de φ est donc 1, comme en a).

Pour préciser les coefficients, si les indices des lignes et colonnes vont de 0 à n , la matrice A vérifie :

$$A_{i,j} = \begin{cases} \frac{j!}{i!} & \text{si } 0 \leq i \leq j \leq n, \\ 0 & \text{si } i > j. \end{cases}$$

d) Méthode 1 — Matrice triangulaire

La matrice triangulaire de φ possède uniquement des coefficients diagonaux non nuls ; elle est donc inversible. Ainsi φ est un automorphisme de $\mathbb{R}_n[X]$.

Méthode 2 — Conservation du coefficient dominant

Si $P \neq 0$, la somme des dérivées d'ordre au moins 1 a un degré strictement inférieur à celui de P . Elle ne peut annuler son coefficient dominant, donc $\varphi(P) \neq 0$. L'application est injective et, en dimension finie avec espace de départ égal à l'espace d'arrivée, elle est bijective.

3) a) Pour $P \in \mathbb{R}_n[X]$, on a $P^{(n+1)} = 0$. En utilisant la linéarité des dérivées :

$$\varphi(P - P') = \sum_{k=0}^n (P^{(k)} - P^{(k+1)}) = P - P^{(n+1)} = P.$$

La somme est télescopique.

b) Comme φ est bijective et $\varphi(P - P') = P$, l'unique antécédent de P est $P - P'$. Ainsi :

$$\forall P \in \mathbb{R}_n[X], \quad \varphi^{-1}(P) = P - P'.$$

Les images des vecteurs de la base sont $\varphi^{-1}(e_0) = e_0$ et, pour $1 \leq j \leq n$, $\varphi^{-1}(e_j) = e_j - j e_{j-1}$. La matrice M de φ^{-1} est donc triangulaire supérieure à deux diagonales :

$$M = \begin{pmatrix} 1 & -1 & 0 & \cdots & 0 \\ 0 & 1 & -2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & 1 & -n \\ 0 & \cdots & \cdots & 0 & 1 \end{pmatrix}.$$

Autrement dit, $M_{j,j} = 1$ pour $0 \leq j \leq n$, $M_{j-1,j} = -j$ pour $1 \leq j \leq n$, et tous les autres coefficients sont nuls.

c) Code adapté en Python. Question adaptée au programme actuel.

Le tableau `M` construit la matrice de φ^{-1} . En Python, les indices commencent à 0 ; le coefficient situé sur la ligne de e_{k-1} et la colonne de e_k est donc `M[k - 1, k]`. La matrice cherchée de φ est l'inverse de `M` :

Python

```
import numpy as np

n = int(input('entrez la valeur de n : '))

M = np.eye(n + 1)

for k in range(1, n + 1):
    M[k - 1, k] = -k

A = np.linalg.inv(M)

print(A)
```

L'inversion est justifiée par l'automorphisme établi en 2) d). La boucle traite les entiers 1 à n inclus. Le code a été exécuté pour $n = 1, 2, 3, 5$ et ses coefficients ont été comparés à la formule $A_{i,j} = j!/i!$ pour $i \leq j$.

Pour $n = 2$, il affiche :

$$A = \begin{pmatrix} 1 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}.$$

Partie 2 : étude d'une autre application définie sur $\mathbb{R}_n[X]$

4) a) Fixons $x \in \mathbb{R}$ et $k \in \mathbb{N}$. La fonction $t \mapsto t^k e^{-t}$ est continue sur tout segment. La borne inférieure x , même négative, ne présente donc aucune singularité.

Pour $t \rightarrow +\infty$, la croissance comparée donne $t^k e^{-t/2} \rightarrow 0$. Il existe donc $T > \max(x, 0)$ tel que, pour $t \geq T$:

$$0 \leq t^k e^{-t} \leq e^{-t/2}.$$

L'intégrale de la majorante exponentielle sur $[T, +\infty[$ converge. Par comparaison et

intégration sur le segment $[x, T]$, l'intégrale $\int_x^{+\infty} t^k e^{-t} dt$ converge absolument.

La justification reste valable pour $k = 0$.

b) Écrivons

$$P(t) = \sum_{j=0}^n a_j t^j.$$

L'inégalité triangulaire donne :

$$|P(t)e^{-t}| \leq \sum_{j=0}^n |a_j| |t|^j e^{-t}.$$

Chaque terme du membre de droite est intégrable sur $[x, +\infty[$ par a). La somme étant finie, l'intégrale de la majorante converge. Ainsi l'intégrale $\int_x^{+\infty} P(t)e^{-t} dt$ est absolument convergente.

5) a) La fonction $-e^{-t}$ est une primitive de e^{-t} . Ainsi :

$$\int_x^{+\infty} e^{-t} dt = \lim_{A \rightarrow +\infty} (-e^{-A} + e^{-x}) = e^{-x}.$$

b) Méthode 1 — Intégration par parties et récurrence

Posons $B_k(x) = \int_x^{+\infty} t^k e^{-t} dt$. Pour $k = 0$, la formule est celle de a), puisque $0! = 1$ et la somme vaut 1.

Pour $k \geq 1$, sur le segment $[x, A]$, une intégration par parties donne :

$$\int_x^A t^k e^{-t} dt = x^k e^{-x} - A^k e^{-A} + k \int_x^A t^{k-1} e^{-t} dt.$$

Comme $A^k e^{-A} \rightarrow 0$ et que les intégrales impropres convergent d'après 4), le passage à la limite donne :

$$B_k(x) = x^k e^{-x} + k B_{k-1}(x).$$

Supposons la formule établie au rang $k - 1$. Alors :

$$B_k(x) = x^k e^{-x} + k(k-1)! e^{-x} \sum_{i=0}^{k-1} \frac{x^i}{i!} = k! e^{-x} \sum_{i=0}^k \frac{x^i}{i!}.$$

La récurrence établit, pour tout $k \in \mathbb{N}$ et tout $x \in \mathbb{R}$:

$$\int_x^{+\infty} t^k e^{-t} dt = k! \left(\sum_{i=0}^k \frac{x^i}{i!} \right) e^{-x}.$$

Cette méthode construit la formule et fournit une relation utile pour calculer successivement les intégrales.

Méthode 2 — Vérification d'une primitive

Définissons le polynôme

$$S_k(t) = k! \sum_{i=0}^k t^i / i!.$$

Pour $k \geq 1$, sa dérivée vaut

$$k! \sum_{i=0}^{k-1} t^i / i!,$$

d'où $S_k - S'_k = t^k$. Cette dernière relation reste vraie pour $k = 0$, car $S_0 = 1$ et $S'_0 = 0$.

En dérivant un produit :

$$\frac{d}{dt}(-e^{-t}S_k(t)) = e^{-t}(S_k(t) - S'_k(t)) = t^k e^{-t}.$$

Ainsi $-e^{-t}S_k(t)$ est une primitive. Chaque monôme de $S_k(t)$ multiplié par e^{-t} tend vers 0 lorsque $t \rightarrow +\infty$. L'évaluation de cette primitive aux bornes donne $B_k(x) = e^{-x}S_k(x)$, soit la même formule.

Cette seconde méthode vérifie directement une expression proposée, sans itérer les intégrations par parties.

6) a) Code adapté en Python. Question adaptée au programme actuel.

Le tableau j contient les réels $1, 2, \dots, k$. Alors `np.prod(j)` calcule $k!$ et `np.cumprod(j)` contient les factorielles $1!, 2!, \dots, k!$. Il faut donc définir u comme le tableau des $x^i/i!$, pour $1 \leq i \leq k$, puis ajouter le terme d'indice 0, qui vaut 1.

Python

```
import numpy as np

k = int(input('entrez la valeur de k : '))
x = float(input('entrez la valeur de x : '))
j = np.arange(1, k + 1, dtype=float)
p = np.prod(j)
u = x**j / np.cumprod(j)
s = p * (1 + np.sum(u)) * np.exp(-x)
print(s)
```

Les puissances et la division sont effectuées coefficient par coefficient. Les dénominateurs sont les factorielles, toutes strictement positives.

Pour $k = 0$, j et u sont vides ; le produit vide vaut 1 et la somme vide vaut 0. Le script renvoie donc e^{-x} , conformément à 5) a). Pour $x = 0$, le terme d'indice 0 reste

explicitement égal à 1, et les autres termes sont nuls.

L'option `dtype=float` évite un calcul des factorielles dans un type entier de taille limitée. Le programme effectue néanmoins une évaluation numérique : pour des valeurs très grandes de k ou très négatives de x , des dépassements de capacité ou des pertes de précision restent possibles.

Code exécuté pour $k = 0, 1, 2, 5, 21$ et plusieurs valeurs de x , positives, nulles ou négatives. Les résultats ont été comparés à une intégration numérique indépendante.

b) Code adapté en Python. Question adaptée au programme actuel.

Pour $A > x$, on effectue le changement de variable affine $t = u + x$, avec $dt = du$. Il donne :

$$\int_x^A t^k e^{-t} dt = e^{-x} \int_0^{A-x} (u+x)^k e^{-u} du.$$

Le membre de droite est absolument convergent lorsque $A \rightarrow +\infty$: $(u+x)^k$ est un polynôme en u , donc 4) b) s'applique sur $[0, +\infty[$. On obtient ainsi :

$$\int_x^{+\infty} t^k e^{-t} dt = e^{-x} \int_0^{+\infty} (u+x)^k e^{-u} du.$$

Si $Z \sim \mathcal{E}(1)$, sa densité est $u \mapsto e^{-u}$ pour $u > 0$. Par transfert, avec intégrabilité absolue assurée par 4) :

$$\int_x^{+\infty} t^k e^{-t} dt = e^{-x} E((Z+x)^k).$$

On remplace cette espérance par la moyenne de 100000 réalisations indépendantes :

Python

```
import numpy as np

import numpy.random as rd

x = float(input('entrez la valeur de x : '))
k = int(input('entrez la valeur de k : '))
Z = rd.exponential(scale=1.0, size=100000)
s = np.exp(-x) * np.mean((Z + x)**k)
print(s)
```

Le tableau Z contient 100000 exponentielles indépendantes de taux 1. La transformation $z \mapsto (z + x)^k$ est appliquée à chaque tirage, puis `np.mean` calcule leur moyenne.

Pour justifier la méthode, la variable $(Z + x)^k$ admet une variance finie : son carré $(Z + x)^{2k}$ est un polynôme dont l'intégrale contre la densité exponentielle converge par 4). La loi faible des grands nombres assure donc que la moyenne de ces transformations converge en probabilité vers leur espérance lorsque le nombre de simulations augmente.

Code exécuté pour $k = 0$ et pour $(x, k) = (1, 2)$. Dans le premier cas, le résultat vaut e^{-x} sans fluctuation ; dans le second, l'estimation a été comparée à la valeur exacte $5e^{-1}$. Une sortie Monte-Carlo reste une approximation aléatoire, sans précision déterministe garantie par le seul nombre de tirages.

7) a) Fixons

$$P = \sum_{j=0}^n a_j X^j.$$

Les intégrales convergent d'après 4) b). Par linéarité d'une somme finie et la formule de 5) b), pour tout $x \in \mathbb{R}$:

$$\psi(P)(x) = e^x \sum_{j=0}^n a_j \int_x^{+\infty} t^j e^{-t} dt = \sum_{j=0}^n a_j j! \sum_{i=0}^j \frac{x^i}{i!}.$$

Cette expression est un polynôme en x de degré au plus n . Ainsi $\psi(P)$ appartient bien à $\mathbb{R}_n[X]$.

Pour tous $P, Q \in \mathbb{R}_n[X]$ et $a, b \in \mathbb{R}$, la linéarité des intégrales convergentes donne $\psi(aP + bQ) = a\psi(P) + b\psi(Q)$. Par conséquent, ψ est un endomorphisme de $\mathbb{R}_n[X]$.

b) La question a) montre déjà que $F = \psi(P)$ est polynomiale, donc de classe C^1 sur $\mathbb{R}$.

Pour obtenir la relation demandée, posons $H(x) = \int_x^{+\infty} P(t)e^{-t} dt$.

La relation de Chasles permet d'écrire, pour tout réel x , avec la convention usuelle d'intégration orientée lorsque $x < 0$:

$$H(x) = \int_0^{+\infty} P(t)e^{-t} dt - \int_0^x P(t)e^{-t} dt.$$

Le premier terme est une constante finie. L'intégrande du second est continue sur $\mathbb{R}$; le théorème fondamental du calcul intégral donne donc $H'(x) = -P(x)e^{-x}$.

Comme $F(x) = e^x H(x)$, la dérivation du produit conduit à :

$$F'(x) = e^x H(x) - P(x) = F(x) - P(x).$$

Ainsi :

$$F - F' = P.$$

Cette justification utilise une primitive sur un segment et n'invoque aucune dérivation sous une intégrale impropre.

c) Si $P \in \text{Ker}(\psi)$, alors $F = \psi(P)$ est le polynôme nul, donc $F' = 0$. La relation

$$F - F' = P \text{ entraîne } P = 0.$$

Le noyau de ψ est donc réduit à $\{0\}$: ψ est injective. Comme elle est un endomorphisme de l'espace de dimension finie $\mathbb{R}_n[X]$, elle est bijective et constitue un automorphisme.

La relation de b) précise aussi son inverse : pour $Q \in \mathbb{R}_n[X]$, $\psi^{-1}(Q) = Q - Q'$.

8) a) L'égalité propre est $\psi(P) = \lambda P$, avec $P \neq 0$ et $\lambda \neq 0$. En appliquant 7) b) à $F = \lambda P$, on obtient :

$$\lambda P - \lambda P' = P.$$

Comme λ est non nul, on peut diviser et réarranger :

$$P' = \frac{\lambda - 1}{\lambda} P.$$

b) Supposons $\lambda \neq 1$. Alors le coefficient $c = (\lambda - 1)/\lambda$ est non nul.

Si $\deg P = d \geq 1$, on a $\deg P' = d - 1$, alors que $\deg(cP) = d$; l'égalité $P' = cP$ est impossible. Si P est constant non nul, $P' = 0$ tandis que $cP \neq 0$, ce qui est également impossible.

Ainsi nécessairement $\lambda = 1$. Enfin, 0 n'est pas une valeur propre puisque ψ est un automorphisme par 7) c). Le seul réel pouvant être valeur propre de ψ est donc 1.

c) Pour le polynôme constant $P = 1$, la question 5) a) donne :

$$\psi(1)(x) = e^x \int_x^{+\infty} e^{-t} dt = e^x e^{-x} = 1.$$

Le polynôme 1 est non nul, donc 1 est effectivement valeur propre. Avec b), on conclut :

$$\text{Sp}(\psi) = \{1\}.$$

9) a) Méthode 1 — Égalité des inverses

Les questions 3) b) et 7) c) donnent, pour tout $Q \in \mathbb{R}_n[X]$:

$$\varphi^{-1}(Q) = Q - Q' = \psi^{-1}(Q).$$

Les deux automorphismes ont donc le même inverse ; en prenant l'inverse de cette application bijective, on obtient $\varphi = \psi$.

Cette méthode relie directement les résultats des deux parties du problème.

Méthode 2 — Égalité sur la base canonique

Pour $j \in \llbracket 0, n \rrbracket$, la formule de 5) b) donne :

$$\psi(e_j)(x) = j! \sum_{i=0}^j \frac{x^i}{i!}.$$

De son côté, la somme des dérivées de X^j est :

$$\varphi(e_j) = \sum_{k=0}^j \frac{j!}{(j-k)!} X^{j-k} = \sum_{i=0}^j \frac{j!}{i!} X^i.$$

Les images de chaque vecteur de la base canonique sont donc égales. Les applications étant linéaires, elles sont égales sur tout $\mathbb{R}_n[X]$. Cette preuve fait apparaître leurs coefficients communs.

b) Supposons $P(t) \geq 0$ pour tout $t \geq a$. Fixons $x \geq a$. Pour tout $t \geq x$, on a alors $P(t)e^{-t} \geq 0$. L'intégrale est convergente par 4) b), et la positivité donne :

$$\int_x^{+\infty} P(t)e^{-t} dt \geq 0.$$

Comme $e^x > 0$, $\psi(P)(x) \geq 0$. Par a), $\psi(P) = \varphi(P)$, d'où :

$$\forall x \geq a, \quad \sum_{i=0}^n P^{(i)}(x) = e^x \int_x^{+\infty} P(t)e^{-t} dt \geq 0.$$

La conclusion porte sur la somme des dérivées ; elle n'exige pas que chaque dérivée soit positive séparément.

Relevé des adaptations

Exercice 1, questions 1) a), 1) b) et 5) : code adapté en Python, avec bornes de tableaux et syntaxe des conditions conservant le comportement mathématique. À la question 5), le pas 0.001 est construit à partir d'un compteur entier afin d'éviter les additions flottantes répétées et de conserver la grille jusqu'à 1.

Exercice 2, questions 2) a) et 2) b) : code adapté en Python ; les tirages uniformes utilisent `rd.random` et les exponentielles utilisent `scale=1.0`. Les histogrammes utilisent des poids 1/10000 pour représenter les fréquences dans dix classes de largeur 1. La convention Python des extrémités des classes est précisée ; elle est équivalente presque sûrement pour les lois continues considérées.

Exercice 2, question 2) b) : la figure est retranscrite en SVG dans l'énoncé à partir de l'original, avec les valeurs mesurées. Les axes ont été calibrés sur les graduations 0 et 10 horizontalement, 0 et 0.7 verticalement. Les dix classes de chaque panneau sont conservées ; la précision des dernières barres est limitée par le document source.

Exercice 3, question 4) a) : la définition de forme bilinéaire symétrique est fournie, afin de demander une vérification élémentaire de linéarité et de symétrie sans supposer connu un cours général sur les formes bilinéaires.

Problème, questions 3) c), 6) a) et 6) b) : code adapté en Python, avec indices commençant à 0, inversion par `np.linalg.inv` et exponentielles de taux 1 simulées avec

scale=1.0. À la question 6) a), les syntaxes `np.prod` et `np.cumprod`, la convention du produit vide et l'option `dtype=float` sont précisées ; le code traite également $k = 0$.

Points restant à vérifier

Exercice 2, question 2) b) : les effectifs exacts et les hauteurs exactes des dernières classes, au-delà de 6, ne peuvent pas être reconstitués avec certitude à partir de l'image. Les barres correspondantes sont reproduites à la résolution graphique disponible ; une hauteur confondue avec l'axe ne signifie pas que l'effectif est exactement nul. Cette limite n'affecte ni la conjecture ni sa démonstration à la question 3).