DS 9 – Mathématiques

Mercredi 12 Juin 2024

BCPST 1A

Durée de l'épreuve : 2 heures 30 minutes

La qualité de la présentation de le copie sera prise en compte pour une part importante de la note finale.

L'usage de document est interdit ainsi que celui de la calculatrice. Les téléphones portables doivent être éteints.

Le sujet est composé de 3 exercices de mathématiques. Le deuxième exercice comporte plusieurs questions d'informatique qui seront évaluées indépendamment.

Important : Une notice Python est disponible en fin de document, que vous pouvez utiliser librement.

Exercice 1 (Étude d'une application linéaire)**.** On considère la fonction

$$\varphi : \begin{array}{l} \mathbb{R}^3 \longrightarrow \mathbb{R}^3 \\ (x,y,z) \longmapsto (2y-z, 3x-2y, -2x+2y+z) \end{array}.$$

et on note $B = (e_1, e_2, e_3)$ la base canonique de $\mathbb{R}^3$.

1. Montrer que φ est une application linéaire.
2. Déterminer $A = \mathrm{Mat}_B(\varphi)$, la matrice associée à φ dans la base B.
3. Déterminer une base et la dimension de $\mathrm{Ker}\,\varphi$. φ est-elle injective ? Justifier.
4. Déterminer une base et la dimension de $\mathrm{Im}\,\varphi$. φ est-elle surjective ? Justifier.
5. On définit les vecteurs de $\mathbb{R}^3$ suivants

$$u = (1,1,1), \quad v = (4,3,-2), \quad w = (2,-3,2)$$

et on note $C = (u, v, w)$ la famille constituée de ces 3 vecteurs.

(a) Démontrer que C est une base de $\mathbb{R}^3$.

(b) Déterminer la matrice $D = \mathrm{Mat}_C(\varphi)$.

(c) On note $P = \mathrm{Mat}_{C,B}(\mathrm{Id}_{\mathbb{R}^3})$ la matrice de l'application identité dans les bases C et B. Justifier que

$$P = \begin{pmatrix} 1 & 4 & 2 \\ 1 & 3 & -3 \\ 1 & -2 & 2 \end{pmatrix}.$$

(d) Montrer que P est inversible et que

$$P^{-1} = \frac{1}{30}\begin{pmatrix} 0 & 12 & 18 \\ 5 & 0 & -5 \\ 5 & -6 & 1 \end{pmatrix}.$$

(e) Prouver que $A = PDP^{-1}$.

(f) Montrer que pour tout $n \in \mathbb{N}$, $A^n = PD^nP^{-1}$ et en déduire l'expression de A^n pour tout $n \in \mathbb{N}$.

Exercice 2 (Une marche aléatoire)**.** On considère un crabe se déplaçant sur sur la droite réelle en effectuant une série de déplacements horizontaux de longueurs ± 1, tous les Δt instants. Ce processus est appelé une marche aléatoire.

A chaque déplacement, ce crabe peut se déplacer de 1 unité vers la gauche avec la probabilité $1-p$ ou de 1 unité vers la droite avec la probabilité p, où $p \in [0,1]$ est un paramètre fixé. Les déplacements sont supposés deux à deux indépendants.

On suppose qu'à l'instant initial le crabe est en $x = 0$. On note S_n la position du crabe au temps $n\Delta t$ c'est-à-dire après n déplacements.

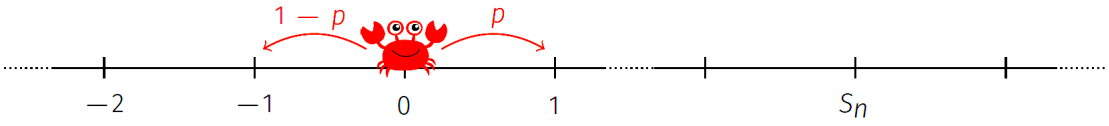

FIGURE 1 – Un crabe se déplaçant sur la droite réelle par translation.

Remarque. Pour tout $n \in \mathbb{N}$, S_n est une variable aléatoire. S_0 suit la loi certaine égale à 0.

1. Donner la loi de S_1.
2. Donner la loi de S_3. On pourra s'aider d'un arbre de probabilités.
3. Soit X_n la variable qui compte le nombre de déplacements vers la droite du crabe au temps $n\Delta t$. Quelle est la loi de X_n ? Justifier.
4. Exprimer S_n en fonction de X_n pour $n \in \mathbb{N}$.
5. Soit $n \in \mathbb{N}$. Déduire de ce qui précède :
   (a) L'espérance de S_n. Pour quelle(s) valeur(s) de p la variable S_n est-elle centrée ? Ce résultat vous semble-t-il cohérent ?
   (b) L'ensemble des valeurs pouvant être prises par S_n ainsi que sa loi.
   (c) La variance de S_n.
6. On cherche maintenant à simuler le déplacement du crabe à l'aide du langage Python.
   (a) Écrire une fonction `bernoulli(p)` prenant en argument $p \in [0,1]$ et renvoyant une réalisation d'une variable aléatoire suivant une loi de Bernoulli de paramètre p.
   (b) A l'aide de la question précédente, écrire une fonction `binomiale(n,p)` prenant en argument $n \in \mathbb{N}$ et $p \in [0,1]$ et renvoyant une réalisation d'une variable aléatoire suivant une loi binomiale de paramètres n et p.
   (c) Utiliser les questions précédentes pour proposer une fonction `deplacement(n, p)` qui prend en argument le nombre n de déplacements du crabe, la probabilité $p \in [0,1]$ et qui renvoie une réalisation de S_n, la position du crabe après n déplacements.
   (d) A l'aide de la fonction `deplacement(n, p)`, proposer un script Python qui calcule une estimation de $E(S_n)$.
   (e) Soit $m \in \mathbb{N}$. On note Z_m le nombre de sauts nécessaires pour atteindre ou dépasser la m-ième marche. Écrire un script qui simule une réalisation de la variable aléatoire Z_m.

Exercice 3 (Urne de Polya)**.** Une urne contient initialement une boule blanche et une boule noire. On répète n fois l'expérience suivante : on tire une boule de l'urne, on la remet et on ajoute une boule de la même couleur. Soit $k \in \{1,\dots,n\}$.

1. Quel est le nombre de boules dans l'urne après la k-ème exprérience ?
2. Pour tout $k \in \mathbb{N}$, on note N_k la variable aléatoire donnant le nombre de boules noires dans l'urne après la k-ème expérience.
   (a) Donner la loi de N_1.
   (b) Quelle est la loi de N_2 ? (*indication : on pourra considérer avantageusement les événements* $(N_1 = 1)$ *et* $(N_1 = 2)$ *et montrer qu'ils forment un système complet d'événements*).
   (c) Montrer par récurrence et en s'inspirant de la question précédente que pour tout $k \in \mathbb{N}$, la variable aléatoire suit N_k une loi uniforme sur $\{1,2,\dots,k+1\}$.

FIN

PYTHON
AGRO-VETO
2023

Listes

`[]` -------- Créer une liste vide
`[a]*n` ----- Créer une liste avec n fois l'élément `a`
`L.append(a)` Ajoute l'élément `a` à la fin de la liste `L`
`L1 + L2` --- Concatène les deux listes `L1` et `L2`
`len(L)` ---- Renvoie le nombre d'éléments de la liste `L`
`L.pop(k)` -- Renvoie le $k^{\text{ème}}$ élément de la liste `L` et l'enlève de `L`
`L.remove(a)` Enlève une fois la valeur `a` de la liste `L`
`max(L)` ---- Renvoie le plus grand élément de la liste `L`
`min(L)` ---- Renvoie le plus petit élément de la liste `L`
`sum(L)` ---- Renvoie la somme de tous les éléments de la liste `L`

Numpy

`import numpy as np`
`np.array()` ------------- Transforme une liste en matrice `numpy`
`np.linspace(a,b,n)` ----- Crée une matrice ligne de n valeurs uniformément réparties entre a et b (inclus)
`np.zeros([n,m])` -------- Crée la matrice nulle de taille $n \times m$
`np.eye(n)` -------------- Crée la matrice identité de taille n
`np.diag(L)` ------------- Crée la matrice diagonale dont les termes diagonaux sont les éléments de la liste `L`
`np.transpose(M)` -------- Renvoie la transposée de M
`np.dot(M,P)` ------------ Renvoie le produit matriciel MP
`np.sum(M)` -------------- Renvoie la somme de tous les éléments de M
`np.prod(M)` ------------- Renvoie le produit de tous les éléments de M
`np.max(M)` -------------- Renvoie le plus grand élément de M
`np.min(M)` -------------- Renvoie le plus petit élément de M
`np.shape(M)` ------------ Renvoie dans un couple le format de la matrice M
`np.size(M)` ------------- Renvoie le nombre d'éléments de M

Numpy.linalg

`import numpy.linalg as la`
`la.inv(M)` -------------- Renvoie l'inverse de la matrice M si elle est inversible
`la.eigvals(M)` ---------- Renvoie la liste des valeurs propres de M
`la.eig(M)` -------------- Renvoie un couple `L,P` où `L` est la liste des valeurs propres de M et `P` la matrice de passage associée
`la.matrix_rank(M)` ------- Renvoie le rang de M

Random

`import random as rd`
`rd.random()` ------- Simule une réalisation d'une variable $X \hookrightarrow \mathcal{U}([0,1])$
`rd.randint(a,b)` --- Simule une réalisation d'une variable $X \hookrightarrow \mathcal{U}(\llbracket a,b \rrbracket)$
`rd.gauss(0,1)` ----- Simule une réalisation d'une variable $X \hookrightarrow \mathcal{N}(0,1)$
`rd.choice(L)` ------ Choisit aléatoirement un élément de la liste `L`

Math

`import math as m`
`m.atan(x)` ------- Renvoie $\arctan(x)$
`m.floor(x)` ------ Renvoie $\lfloor x \rfloor$
`m.factorial(n)` -- Renvoie $n!$ si $n \in \mathbb{N}$
`m.sqrt(x)` -- Renvoie $\sqrt{x}$ si $x \geq 0$
`m.log(x)` --- Renvoie $\ln(x)$ si $x > 0$
`m.exp(x)` --- Renvoie e^x

Logique

`a == b` ---- Teste l'égalité « $a = b$ »
`a != b` ---- Teste « $a \neq b$ »
`a < b` ----- Teste « $a < b$ »
`a <= b` ---- Teste « $a \leq b$ »
`a > b` ----- Teste « $a > b$ »
`a => b` ---- Teste « $a \geq b$ »
`not A` ----- Renvoie la négation de A
`A and B` --- Renvoie « A et B »
`A or B` ---- Renvoie « A ou B »
`True` ------ Constante booléenne « Vrai »
`False` ----- Constante booléenne « Faux »

Matplotlib.pyplot

`import matplotlib.pyplot as plt`
`plt.plot(X,Y,'+-r')` ---- Génère la courbe des points définis par les listes `X` et `Y` (abscisses et ordonnées) avec les options :
- symbole : `'.'` point, `'o'` rond, `'h'` hexagone, `'+'` plus, `'x'` croix, `'*'` étoile, ...
- ligne : `'-'` trait plein, `'--'` pointillé, `'-.'` alterné, ...
- couleur : `'b'` **bleu** , `'r'` **rouge** , `'g'` **vert** , `'c'` **cyan** , `'m'` **magenta** , `'k'` **noir** , ...

`plt.bar(X,Y)` ---------- Génère l'histogramme des points définis par les listes `X` et `Y` (abscisses et ordonnées)
`plt.axis('equal')` ------ Rend le repère orthonormé
`plt.xlim(xmin,xmax)` ---- Fixe les bornes de l'axe des abscisses
`plt.ylim(ymin,ymax)` ---- Fixe les bornes de l'axe des ordonnées
`plt.show()` ------------ Affiche le graphique

Cette liste est non exhaustive. Les candidats sont libres d'utiliser les commandes de leur choix.