



PRÉPASIMPLE

prepasimple.fr

ESSEC

Mathématiques 2

1992

Corrigé de mathématiques approfondies

Mathématiques approfondies — Prépa ECG

Annales de concours · Corrigé

PARTIE 1

On note $\mathbb{R}_m[X]$ l'espace des fonctions polynômes réelles de degré inférieur ou égal à m , en incluant le polynôme nul. Sa dimension est $m + 1$.

a. Endomorphisme et noyau

Si P est une fonction polynomiale, $X \mapsto P(X + 1) - P(X)$ est encore une fonction polynomiale réelle. De plus, pour $P, R \in \mathbb{R}[X]$ et $\alpha, \beta \in \mathbb{R}$:

$$\Delta(\alpha P + \beta R) = \alpha \Delta P + \beta \Delta R.$$

Ainsi Δ est un endomorphisme de $\mathbb{R}[X]$. Tout polynôme constant appartient à son noyau.

Méthode 1 — Examiner le degré

Si P n'est pas constant, écrivons son degré $d \geq 1$ et son coefficient dominant $a_d \neq 0$. Le binôme donne $(X + 1)^d - X^d = dX^{d-1}$ plus des termes de degré inférieur à $d - 1$. Les différences issues des termes de degré inférieur à d de P ont un degré inférieur à $d - 1$. Donc :

$$\deg(\Delta P) = d - 1, \quad \text{son coefficient dominant est } da_d \neq 0.$$

Par conséquent ΔP ne peut pas être nul. Le noyau est donc exactement l'ensemble des polynômes constants :

$$\text{Ker } \Delta = \text{Vect}(P_0) = \mathbb{R}_0[X].$$

Méthode 2 — Utiliser les racines d'un polynôme

Si $P \in \text{Ker } \Delta$, alors $P(x+1) = P(x)$ pour tout réel x . En particulier, par récurrence, $P(m) = P(0)$ pour tout entier $m \geq 0$. Le polynôme $P - P(0)$ possède une infinité de racines distinctes ; un polynôme non nul n'en possède qu'un nombre fini. Donc $P - P(0)$ est nul, et P est constant.

La première preuve précise aussi l'effet de Δ sur le degré ; la seconde montre directement qu'une fonction polynomiale de période 1 est constante.

b. Différences des polynômes P_k

Pour $k = 0$, $\Delta P_0 = 0$. Pour $k = 1$, $P_1(X) = X$, donc $\Delta P_1 = 1 = P_0$.

Pour $k \geq 2$, les produits $P_k(X+1)$ et $P_k(X)$ possèdent le facteur commun $X(X-1)\cdots(X-k+2)$. On factorise sans diviser par ce facteur, qui peut s'annuler :

$$P_k(X+1) - P_k(X) = \frac{X(X-1)\cdots(X-k+2)}{k!} [(X+1) - (X-k+1)].$$

La différence entre crochets vaut k , d'où :

$$\Delta P_k = P_{k-1} \quad \text{pour } k \geq 1, \quad \Delta P_0 = 0.$$

En itérant, chaque application de Δ abaisse l'indice de 1 jusqu'à P_0 , puis donne le polynôme nul. Ainsi, pour $j, k \in \mathbb{N}$:

$$\Delta^j P_k = \begin{cases} P_{k-j} & \text{si } 0 \leq j \leq k, \\ 0 & \text{si } j > k. \end{cases}$$

Le cas $j = 0$ restitue bien P_k , conformément à la convention de l'énoncé.

c. Base et formule de décomposition

Méthode 1 — Utiliser les degrés échelonnés

Pour chaque k , P_k est de degré k et de coefficient dominant $1/k!$. Si une combinaison linéaire

$$\sum_{k=0}^n a_k P_k$$

est nulle et qu'au moins un coefficient est non nul, soit r le plus grand indice de coefficient non nul. Cette combinaison a un terme de degré r de coefficient $a_r/r! \neq 0$, contradiction.

La famille est donc libre. Elle possède $n + 1$ éléments dans $\mathbb{R}_n[X]$, de dimension $n + 1$; elle en est une base.

Méthode 2 — Évaluer aux entiers successifs

Pour $m \in \{0, \dots, n\}$, on a $P_k(m) = 0$ lorsque $k > m$ et $P_m(m) = 1$. Si

$$\sum_{k=0}^n a_k P_k = 0,$$

l'évaluation en 0 donne $a_0 = 0$. L'évaluation en 1 donne ensuite $a_1 = 0$, puis l'évaluation en 2 donne $a_2 = 0$. Par récurrence, lorsque $a_0, \dots, a_{m-1}$ sont nuls, l'évaluation en m donne $a_m = 0$.

La famille est libre et, par le même argument de dimension, constitue une base. La première méthode utilise les coefficients dominants; la seconde exploite une structure triangulaire des valeurs aux entiers.

Détermination des coordonnées

Écrivons

$$P = \sum_{k=0}^n a_k P_k.$$

Pour $j \in \{0, \dots, n\}$, la question b. donne :

$$\Delta^j P = \sum_{k=j}^n a_k P_{k-j}.$$

Or $P_0(0) = 1$ et $P_r(0) = 0$ pour tout $r \geq 1$. L'évaluation en 0 conserve donc uniquement le terme d'indice $k = j$, ce qui donne $(\Delta^j P)(0) = a_j$. Ainsi :

$$P(X) = \sum_{k=0}^n (\Delta^k P)(0) P_k(X).$$

d. Surjectivité et primitive discrète normalisée

Méthode 1 — Construire explicitement un antécédent

Pour un polynôme quelconque P de degré au plus m , écrivons

$$P = \sum_{k=0}^m a_k P_k.$$

Posons :

$$Q = \sum_{k=0}^m a_k P_{k+1}.$$

D'après b.,

$$\Delta Q = \sum_{k=0}^m a_k P_k = P.$$

Tout polynôme admet donc un antécédent : Δ est surjectif. Comme $P_{k+1}(0) = 0$ pour tous les indices, ce Q vérifie $Q(0) = 0$.

Si P est de degré exactement n , son coefficient a_n dans la base est non nul. Le polynôme Q construit est alors de degré $n + 1$, car son terme de plus haut degré provient de $a_n P_{n+1}$.

Si deux polynômes Q et R satisfont $\Delta Q = \Delta R = P$ et $Q(0) = R(0) = 0$, leur différence appartient à $\text{Ker } \Delta$. Elle est donc constante et vaut 0 à l'origine ; elle est nulle. Cela prouve l'unicité.

Méthode 2 — Utiliser le théorème du rang en dimension finie

Pour $m \geq 0$, posons $F_{m+1} = \{Q \in \mathbb{R}_{m+1}[X] : Q(0) = 0\}$. Ce sous-espace possède la base $(X, X^2, \dots, X^{m+1})$ et sa dimension est $m + 1$.

La restriction de Δ à F_{m+1} prend ses valeurs dans $\mathbb{R}_m[X]$: le degré diminue d'une unité pour un polynôme non constant. Son noyau est nul, car un polynôme constant qui s'annule en 0 est nul. Le théorème du rang donne donc un rang $m + 1$, égal à la dimension de $\mathbb{R}_m[X]$. Cette restriction est un isomorphisme.

Elle fournit pour chaque $P \in \mathbb{R}_m[X]$ un unique antécédent normalisé. Pour P de degré n , cet antécédent est de degré $n + 1$, puisque $\deg(\Delta Q) = \deg Q - 1$. L'unicité parmi tous les polynômes suit encore du noyau de Δ .

La première méthode donne une formule constructive ; la seconde explique l'existence et l'unicité par un isomorphisme entre espaces de même dimension.

En particulier, l'unique polynôme Q_n est donné par :

$$Q_n(X) = \sum_{k=0}^n (\Delta^k(X^n))(0) P_{k+1}(X).$$

e. Sommes de puissances et calcul de Q_1, Q_2, Q_3

Pour tout entier $r \geq 1$, la définition donne $r^n = Q_n(r+1) - Q_n(r)$. La somme est télescopique :

$$S_n(p) = \sum_{r=1}^p [Q_n(r+1) - Q_n(r)] = Q_n(p+1) - Q_n(1).$$

Comme $n \geq 1$, $0^n = 0$. En évaluant l'identité de définition de Q_n en 0, on obtient $Q_n(1) - Q_n(0) = 0$. Puisque $Q_n(0) = 0$, il vient $Q_n(1) = 0$. Donc :

$$S_n(p) = Q_n(p+1).$$

Méthode 1 — Décomposer dans la base des P_k

Pour $n = 1$, $X = P_1(X)$. Donc $Q_1 = P_2$:

$$Q_1(X) = \frac{X(X-1)}{2}, \quad S_1(p) = \frac{p(p+1)}{2}.$$

Pour $n = 2$, $2P_2(X) = X^2 - X$, donc $X^2 = P_1 + 2P_2$. L'antécédent normalisé est :

$$Q_2 = P_2 + 2P_3 = \frac{X(X-1)}{2} + \frac{X(X-1)(X-2)}{3} = \frac{X(X-1)(2X-1)}{6}.$$

Ainsi :

$$S_2(p) = \frac{p(p+1)(2p+1)}{6}.$$

Pour $n = 3$, $6P_3(X) = X^3 - 3X^2 + 2X$. En remplaçant X^2 par $P_1 + 2P_2$, on obtient $X^3 = P_1 + 6P_2 + 6P_3$. Ainsi :

$$Q_3 = P_2 + 6P_3 + 6P_4.$$

En factorisant $X(X-1)$:

$$Q_3(X) = X(X-1) \left[\frac{1}{2} + (X-2) + \frac{(X-2)(X-3)}{4} \right] = \frac{X^2(X-1)^2}{4}.$$

On conclut :

$$S_3(p) = \frac{p^2(p+1)^2}{4}.$$

Méthode 2 — Identifier les coefficients dans la base usuelle

La condition $Q_n(0) = 0$ impose un terme constant nul. Pour $n = 1$, écrivons

$Q_1 = a_2X^2 + a_1X$. Alors $\Delta Q_1 = 2a_2X + a_2 + a_1$. L'identité $\Delta Q_1 = X$ donne $a_2 = 1/2$ et $a_1 = -1/2$, ce qui redonne $Q_1 = X(X-1)/2$.

Pour $n = 2$, écrivons $Q_2 = a_3X^3 + a_2X^2 + a_1X$. En développant :

$$\Delta Q_2 = 3a_3X^2 + (3a_3 + 2a_2)X + (a_3 + a_2 + a_1).$$

L'identification avec X^2 donne successivement $a_3 = 1/3$, $a_2 = -1/2$, $a_1 = 1/6$. On retrouve $Q_2 = X(X-1)(2X-1)/6$.

Pour $n = 3$, écrivons $Q_3 = a_4X^4 + a_3X^3 + a_2X^2 + a_1X$. Le binôme donne :

$$\Delta Q_3 = 4a_4X^3 + (6a_4 + 3a_3)X^2 + (4a_4 + 3a_3 + 2a_2)X + (a_4 + a_3 + a_2 + a_1).$$

L'identification avec X^3 donne $a_4 = 1/4$, puis $a_3 = -1/2$, $a_2 = 1/4$ et $a_1 = 0$. Donc $Q_3 = X^2(X - 1)^2/4$.

En évaluant chaque polynôme en $p + 1$, cette seconde méthode retrouve les trois sommes précédentes. La base des P_k simplifie l'inversion de Δ ; l'identification dans la base usuelle est directe pour de petits degrés.

PARTIE 2

Les variables aléatoires considérées dans cette partie ont un support fini. Toutes leurs espérances et variances existent. Les fonctions génératrices sont des polynômes réels, définis pour tout réel t ; seuls des produits et des sommes finis seront utilisés. Dans ces polynômes, le monôme constant t^0 vaut 1, y compris lorsque $t = 0$.

1. Fonction génératrice d'une variable aléatoire

1. (a) Loi de Bernoulli

La loi de Bernoulli de paramètre λ donne $P(X = 0) = 1 - \lambda$ et $P(X = 1) = \lambda$. Par définition :

$$G_X(t) = 1 - \lambda + \lambda t, \quad t \in \mathbb{R}.$$

1. (b) Loi binomiale

Si $X \sim \mathcal{B}(n, \lambda)$, alors $P(X = k) = \binom{n}{k} \lambda^k (1 - \lambda)^{n-k}$ pour $0 \leq k \leq n$. La formule du binôme donne :

$$G_X(t) = \sum_{k=0}^n \binom{n}{k} (\lambda t)^k (1-\lambda)^{n-k} = (1-\lambda + \lambda t)^n.$$

Cette égalité est valable pour tout réel t .

2. Fonction génératrice d'une somme de variables aléatoires indépendantes

2. (a) Somme de deux variables indépendantes

Question adaptée au programme actuel : on précise $n \geq 2$ pour que X_2 soit bien présente dans la famille.

Méthode 1 — Espérance d'un produit

Pour une variable entière positive ou nulle à support fini, le transfert donne $G_X(t) = E(t^X)$, avec la convention $t^0 = 1$ précédente. Les variables t^{X_1} et t^{X_2} sont des fonctions respectives de deux variables indépendantes ; elles sont donc indépendantes. Elles ont un support fini et sont intégrables. Par factorisation de l'espérance :

$$G_{Y_2}(t) = E(t^{X_1+X_2}) = E(t^{X_1}t^{X_2}) = E(t^{X_1})E(t^{X_2}) = G_{X_1}(t)G_{X_2}(t).$$

Méthode 2 — Calculer la loi de la somme

Pour chaque entier m entre 0 et $p_1 + p_2$, les couples (a, b) tels que $a + b = m$ réalisent des événements disjoints. L'indépendance donne :

$$P(Y_2 = m) = \sum_{\substack{0 \leq a \leq p_1, \\ 0 \leq b \leq p_2 \\ a+b=m}} P(X_1 = a)P(X_2 = b).$$

En insérant cette formule dans la définition de G_{Y_2} et en réorganisant les sommes finies :

$$G_{Y_2}(t) = \sum_{a=0}^{p_1} \sum_{b=0}^{p_2} P(X_1 = a)P(X_2 = b)t^{a+b} = \left(\sum_{a=0}^{p_1} P(X_1 = a)t^a \right) \left(\sum_{b=0}^{p_2} P(X_2 = b)t^b \right).$$

On retrouve le même produit. La première méthode est courte et exploite le transfert ; la seconde explicite le regroupement des couples dans la loi de la somme.

2. (b) Somme de n variables mutuellement indépendantes

On raisonne par récurrence. Pour $n = 1$, $G_{Y_1} = G_{X_1}$. Supposons le résultat établi pour une somme de $n - 1$ variables, avec $n \geq 2$.

Par indépendance mutuelle et par le lemme des coalitions, $Y_{n-1} = X_1 + \dots + X_{n-1}$ est indépendante de X_n . La question précédente et l'hypothèse de récurrence donnent :

$$G_{Y_n}(t) = G_{Y_{n-1}}(t)G_{X_n}(t) = \prod_{k=1}^n G_{X_k}(t).$$

Cette égalité est donc établie pour tout $n \geq 1$ et tout $t \in \mathbb{R}$.

2. (c) Retrouver la fonction génératrice d'une Bernoulli

La formule de 1. (b) est valable pour tout entier $n \geq 1$. En la spécialisant à $n = 1$, la loi $\mathcal{B}(1, \lambda)$ est une loi de Bernoulli de paramètre λ , puisque ses deux probabilités sont $1 - \lambda$ en 0 et λ en 1. Elle a donc pour fonction génératrice :

$$G_X(t) = (1 - \lambda + \lambda t)^1 = 1 - \lambda + \lambda t.$$

On retrouve exactement le résultat de 1. (a), en respectant les renvois figurant dans l'énoncé.

3. Étude d'un cas particulier

3. (a) Moments des lois uniformes et de la somme

Méthode 1 — Transfert et sommes de puissances

Pour $1 \leq k \leq n$, X_k est uniforme sur $\{0, \dots, k-1\}$. La partie 1 donne :

$$E(X_k) = \frac{1}{k} \sum_{r=0}^{k-1} r = \frac{1}{k} \frac{(k-1)k}{2} = \frac{k-1}{2}.$$

$$E(X_k^2) = \frac{1}{k} \sum_{r=0}^{k-1} r^2 = \frac{1}{k} \frac{(k-1)k(2k-1)}{6} = \frac{(k-1)(2k-1)}{6}.$$

Ces identités comprennent $k=1$, où $X_1=0$ et les sommes sont nulles. Par Koenig-Huygens :

$$V(X_k) = \frac{(k-1)(2k-1)}{6} - \frac{(k-1)^2}{4} = \frac{(k-1)[2(2k-1) - 3(k-1)]}{12} = \frac{k^2-1}{12}.$$

Méthode 2 — Ramener à une loi uniforme usuelle

La variable $T_k = X_k + 1$ est uniforme sur $\{1, \dots, k\}$. Les moments de cette loi usuelle sont $E(T_k) = (k+1)/2$ et $V(T_k) = (k^2-1)/12$. La transformation affine donne $E(X_k) = E(T_k) - 1 = (k-1)/2$ et $V(X_k) = V(T_k) = (k^2-1)/12$.

Cette seconde méthode est immédiate lorsque les moments de la loi uniforme usuelle sont disponibles; la première les retrouve à partir des sommes établies dans la partie 1.

Par linéarité de l'espérance :

$$E(Y_n) = \sum_{k=1}^n \frac{k-1}{2} = \frac{n(n-1)}{4}.$$

Les X_k sont mutuellement indépendantes et de variance finie ; la variance de leur somme est la somme des variances :

$$V(Y_n) = \sum_{k=1}^n \frac{k^2 - 1}{12} = \frac{1}{12} \left[\frac{n(n+1)(2n+1)}{6} - n \right].$$

Or $(n+1)(2n+1) - 6 = 2n^2 + 3n - 5 = (n-1)(2n+5)$. Donc :

$$V(Y_n) = \frac{n(n-1)(2n+5)}{72}.$$

Pour $n = 1$, $Y_1 = 0$, et les deux moments sont bien nuls.

3. (b) Récurrence sur les coefficients

La variable X_{n+1} est uniforme sur $\{0, \dots, n\}$, donc :

$$G_{X_{n+1}}(t) = \frac{1 + t + \dots + t^n}{n+1}.$$

L'indépendance et 2. (b) donnent :

$$G_{n+1}(t) = \frac{1 + t + \dots + t^n}{n+1} G_n(t) = \frac{1}{(n+1)!} (1 + t + \dots + t^n) \sum_{r=0}^{n(n-1)/2} c_{n,r} t^r.$$

Le coefficient de t^k dans le produit est la somme des $c_{n,r}$ pour lesquels $r = k - j$, avec $0 \leq j \leq n$. Avec la convention sur les coefficients hors support :

$$c_{n+1,k} = \sum_{j=0}^n c_{n,k-j} = c_{n,k} + c_{n,k-1} + \cdots + c_{n,k-n}.$$

Cette formule vaut pour $0 \leq k \leq n(n+1)/2$ et s'étend à tout entier k avec les mêmes conventions. Le degré maximal est augmenté de n , conformément à $n(n-1)/2 + n = n(n+1)/2$.

3. (c) Coefficients entiers et somme

Méthode 1 — Récurrence

Pour $n = 1$, $X_1 = Y_1 = 0$ presque sûrement, donc $G_1(t) = 1$ et $c_{1,0} = 1$. Les autres coefficients valent 0 par convention.

Si les $c_{n,k}$ sont des entiers naturels, la récurrence de b. exprime chaque $c_{n+1,k}$ comme une somme finie de tels entiers. Les coefficients sont donc des entiers naturels pour tout $n \geq 1$.

Enfin

$$G_n(1) = \sum_k P(Y_n = k) = 1,$$

donc :

$$\sum_{k=0}^{n(n-1)/2} c_{n,k} = n!.$$

Méthode 2 — Dénombrer les suites de valeurs

Chaque n -uplet $(x_1, \dots, x_n)$ satisfaisant $0 \leq x_r \leq r - 1$ a, par indépendance, la probabilité

$$\prod_{r=1}^n (1/r) = 1/n!.$$

Il y a

$$\prod_{r=1}^n r = n!$$

tels n -uplets.

Le nombre de ces n -uplets dont la somme vaut k est donc $n!P(Y_n = k) = c_{n,k}$. Il s'agit d'un entier naturel. Les classes correspondant aux différentes sommes k partitionnent l'ensemble des n -uplets; la somme de leurs cardinaux vaut $n!$.

La première méthode utilise la récurrence des polynômes; la seconde donne une interprétation combinatoire des coefficients.

PARTIE 3

Code adapté en Python. Les indices mathématiques $1, \dots, n$ sont conservés : dans les tableaux Python, la case d'indice 0 n'est pas utilisée. Les boucles `range(1, n)` et `range(1, n - k + 1)` parcourent respectivement $1, \dots, n - 1$ et $1, \dots, n - k$, comme les boucles du sujet original.

La fonction de tri utilisée est la suivante; elle ne nécessite aucun import :

Python

```
def echanger(s, j):
    temp = s[j]
    s[j] = s[j + 1]
```

`s[j + 1] = temp`

```
def tri_bulles(s, n):
    for k in range(1, n):
        for j in range(1, n - k + 1):
            if s[j] > s[j + 1]:
                echanger(s, j)
```

Les codes de tri et de calcul des inversions présentés ici ont été exécutés sur toutes les permutations pour $1 \leq n \leq 7$. Les résultats concordent avec les démonstrations ci-dessous ; ces contrôles finis ne remplacent pas les preuves générales.

1. Exemples de mise en œuvre de l'algorithme

Pour $n = 3$, les comparaisons ont lieu dans l'ordre $(k, j) = (1, 1), (1, 2), (2, 1)$. Les tableaux ci-dessous donnent l'état initial puis chaque état obtenu après un appel effectif de `echanger` ; une comparaison sans échange ne crée pas de ligne supplémentaire.

Tableau initial (1, 2, 3)

Étape	$s[1]$	$s[2]$	$s[3]$
État initial	1	2	3

Nombre total d'appels : 0. Le tableau est déjà trié.

Tableau initial (2, 3, 1)

Étape	$s[1]$	$s[2]$	$s[3]$
État initial	2	3	1
Échange avec $(k, j) = (1, 2)$	2	1	3
Échange avec $(k, j) = (2, 1)$	1	2	3

Nombre total d'appels : 2.

Tableau initial (3, 1, 2)

Étape	$s[1]$	$s[2]$	$s[3]$
État initial	3	1	2
Échange avec $(k, j) = (1, 1)$	1	3	2
Échange avec $(k, j) = (1, 2)$	1	2	3

Nombre total d'appels : 2.

Tableau initial (1, 3, 2)

Étape	$s[1]$	$s[2]$	$s[3]$
État initial	1	3	2
Échange avec $(k, j) = (1, 2)$	1	2	3

Nombre total d'appels : 1.

Tableau initial (3, 2, 1)

Étape	$s[1]$	$s[2]$	$s[3]$
État initial	3	2	1
Échange avec $(k, j) = (1, 1)$	2	3	1
Échange avec $(k, j) = (1, 2)$	2	1	3
Échange avec $(k, j) = (2, 1)$	1	2	3

Nombre total d'appels : 3.

Tableau initial (2, 1, 3)

Étape	$s[1]$	$s[2]$	$s[3]$
État initial	2	1	3
Échange avec $(k, j) = (1, 1)$	1	2	3

Nombre total d'appels : 1.

2. Action de l'algorithme

2. (a) Première remontée du maximum

Code adapté en Python. La boucle parcourt $j = 1, \dots, n - 1$ et échange les cases adjacentes seulement lorsqu'elles sont dans l'ordre décroissant.

Notons a_i la valeur initiale de $s[i]$. Après la comparaison d'indice j , la case $s[j + 1]$ contient $\max(a_1, \dots, a_{j+1})$.

En effet, pour $j = 1$, l'échange éventuel place le maximum de a_1, a_2 en case 2. Si, après l'étape $j - 1$, $s[j]$ contient le maximum de $a_1, \dots, a_j$, la case $s[j + 1]$ n'a pas encore été touchée et contient a_{j+1} . La comparaison et l'échange éventuel placent donc en case $j + 1$ le maximum de ces $j + 1$ valeurs. Cela établit la propriété par récurrence.

Pour $n \geq 2$, après la dernière comparaison, $s[n] = \max(a_1, \dots, a_n) = n$. Si $n = 1$, la boucle est vide et l'unique case contient déjà $1 = n$. Dans tous les cas :

$$s[n] = n.$$

2. (b) État final

Après le premier passage de la boucle extérieure, le maximum n est en case n . Il ne sera plus déplacé, car les passages suivants ne comparent que des cases situées avant lui.

Plus généralement, supposons qu'après $k - 1$ passages, les cases $n - k + 2, \dots, n$ contiennent leurs indices respectifs et que les cases $1, \dots, n - k + 1$ contiennent les

valeurs $1, \dots, n - k + 1$ dans un ordre quelconque. Le passage d'indice k balaie précisément ce préfixe ; par le raisonnement de a., il place son maximum $n - k + 1$ en case $n - k + 1$, sans modifier les cases déjà fixées.

L'invariant est donc préservé. Après $n - 1$ passages, toutes les cases $2, \dots, n$ sont correctes et la valeur restante en case 1 est 1. Pour $n = 1$, aucun passage n'est nécessaire. Conclusion :

$$s[k] = k \quad \text{pour tout } 1 \leq k \leq n \text{ en fin d'algorithme.}$$

3. Complexité de l'algorithme

3. (a) Nombre de comparaisons

Pour un passage extérieur d'indice k , la boucle intérieure effectue $n - k$ comparaisons entre valeurs du tableau. Ce nombre ne dépend pas de leur résultat. Par la somme arithmétique établie dans la partie 1 :

$$C_n = \sum_{k=1}^{n-1} (n - k) = \sum_{r=1}^{n-1} r = \frac{n(n-1)}{2}.$$

Il s'agit du nombre de tests $s[j] > s[j + 1]$, et non du nombre des tests de contrôle des boucles. Pour $n = 1$, il vaut 0.

3. (b) Nombres extrêmes d'échanges

Il y a au plus un échange par comparaison, et le nombre d'échanges est positif ou nul. Il est donc compris entre 0 et $n(n-1)/2$.

Le minimum 0 est atteint pour $s = (1, 2, \dots, n)$: toutes les comparaisons sont fausses et le tableau demeure inchangé.

Le maximum est atteint pour $s = (n, n-1, \dots, 1)$. En effet, le premier passage échange n successivement avec chacun des $n-1$ nombres suivants et produit $(n-1, n-2, \dots, 1, n)$.

Le préfixe restant est encore décroissant. Au passage suivant, $n - 1$ est échangé avec les $n - 2$ valeurs qui le suivent dans ce préfixe. En poursuivant par récurrence, chaque comparaison produit un échange : le passage k en effectue $n - k$.

Donc :

$$N_{\min} = 0, \quad N_{\max} = \frac{n(n-1)}{2}.$$

Pour $n = 1$, les deux extrêmes coïncident à 0.

4. Nombre d'appels de la fonction `echanger`

4. (a) Effet d'un échange adjacent

Supposons $s[j] = a > b = s[j+1]$. L'échange transforme le couple adjacent (a, b) en (b, a) .

Le couple de positions $(j, j+1)$ était une inversion et cesse de l'être : sa contribution diminue de 1.

Pour une position $i < j$, la contribution totale des deux couples (i, j) et $(i, j+1)$ est $\mathbf{1}_{\{s[i]>a\}} + \mathbf{1}_{\{s[i]>b\}}$. Après l'échange, ces deux termes sont simplement permutés ; leur somme est conservée.

Pour une position $i > j+1$, la contribution totale des deux couples (j, i) et $(j+1, i)$ est $\mathbf{1}_{\{a>s[i]\}} + \mathbf{1}_{\{b>s[i]\}}$, également conservée.

Enfin, les couples ne faisant intervenir ni j ni $j+1$ sont inchangés. Le nombre total d'inversions diminue donc exactement de 1 à chaque échange.

D'après 2. (b), le tableau final est croissant et possède 0 inversion. Si le nombre d'appels est $N(s)$, on a donc $0 = V_n(s) - N(s)$. Ainsi :

$$N(s) = V_n(s).$$

Cette identité explique pourquoi des tableaux de même taille peuvent subir des nombres d'échanges différents alors que leur nombre de comparaisons est identique.

4. (b) Inversions des six tableaux

On compte, pour chaque case k , les valeurs strictement plus grandes situées à sa gauche.

Tableau s	$U_1(s)$	$U_2(s)$	$U_3(s)$	$V_3(s)$
(1, 2, 3)	0	0	0	0
(2, 3, 1)	0	0	2	2
(3, 1, 2)	0	1	1	2
(1, 3, 2)	0	0	1	1
(3, 2, 1)	0	1	2	3
(2, 1, 3)	0	1	0	1

Par exemple, dans (2, 3, 1), aucune valeur avant 3 n'est plus grande que lui, tandis que les deux valeurs 2 et 3 sont plus grandes que 1. Dans (3, 1, 2), 3 est plus grand que chacune des deux valeurs suivantes, d'où les deux contributions 1 et 1. Les autres lignes s'obtiennent par le même comptage direct.

Les valeurs de V_3 coïncident avec les nombres d'échanges calculés en 1., conformément à 4. (a).

4. (c) Programme de calcul des inversions

Question adaptée au programme actuel. Code adapté en Python : la fonction conserve les indices $1, \dots, n$ et utilise une case 0 inutilisée dans les deux tableaux.

Python

```
import numpy as np
```

```
def inversions(s, n):
    U = np.zeros(n + 1, dtype=int)
    V = 0
    for k in range(1, n + 1):
        for i in range(1, k):
            if s[i] > s[k]:
                U[k] = U[k] + 1
        V = V + U[k]
    return U, V
```

Le tableau U est initialisé à zéro. Pour un k fixé, $\text{range}(1, k)$ parcourt exactement les indices $1 \leq i < k$. Chaque comparaison vraie ajoute 1 à $U[k]$, qui contient donc $U_k(s)$ après la boucle intérieure. La variable V cumule ces valeurs et contient finalement $V_n(s)$.

Pour $k = 1$, la boucle intérieure est vide : $U[1]$ reste bien nul. Le programme ne modifie pas s et ne lit jamais $s[0]$.

Exemple exécuté :

Python

```
import numpy as np

s = np.array([0, 3, 2, 1], dtype=int)
U, V = inversions(s, 3)
```

On obtient $U = \text{array}([0, 0, 1, 2])$ et $V = 3$. Le premier zéro de U correspond à la case inutilisée, et les trois suivants sont les valeurs de U_1, U_2, U_3 . Aucun argument probabiliste n'est déduit de cette seule exécution.

5. Bijectivité de l'application $s \mapsto U(s)$

5. (a) Ensemble des valeurs possibles

Pour un k fixé, il y a exactement $k - 1$ positions avant la case k . Le nombre $U_k(s)$ de celles qui contiennent une valeur plus grande est donc un entier satisfaisant :

$$0 \leq U_k(s) \leq k-1, \quad \text{soit } 0 \leq U_k(s) < k.$$

En particulier $U_1(s) = 0$. Ainsi $U(s) = (U_1(s), \dots, U_n(s))$ appartient à Ω_n .

5. (b) Reconstruction de droite à gauche

Toutes les valeurs $1, \dots, n$ autres que $s[n]$ sont placées avant la dernière case. Le nombre de valeurs plus grandes que $s[n]$ est donc exactement $n - s[n]$. Ainsi :

$$u_n = U_n(s) = n - s[n], \quad s[n] = n - u_n.$$

Une fois cette valeur déterminée, on la retire de l'ensemble des valeurs disponibles. Il reste $n - 1$ valeurs qui occupent les cases $1, \dots, n - 1$.

Plus généralement, supposons les cases $k + 1, \dots, n$ déjà reconstruites. Notons $a_1 < a_2 < \dots < a_k$ les k valeurs encore disponibles, qui doivent occuper les cases $1, \dots, k$. Si $s[k] = a_r$, les $k - r$ valeurs disponibles plus grandes que a_r sont toutes dans les cases précédentes. Par conséquent $U_k(s) = k - r$.

L'identité $U_k(s) = u_k$ impose donc :

$$s[k] = a_{k-u_k}.$$

Cette valeur existe et est unique, puisque $0 \leq u_k < k$ implique $1 \leq k - u_k \leq k$. On retire ensuite cette valeur et on continue avec $k - 1$. Cela détermine successivement $s[n - 1], \dots, s[1]$.

Par exemple, pour $(u_1, u_2, u_3) = (0, 0, 2)$, on choisit $s[3] = 3 - 2 = 1$. Parmi les valeurs restantes $2 < 3$, on choisit $s[2] = 3$, puisque $u_2 = 0$. Il reste $s[1] = 2$: le tableau reconstruit est $(2, 3, 1)$.

5. (c) Existence et unicité d'un tableau pour chaque code

Méthode 1 — Construire l'application réciproque

Soit un n -uplet admissible $(u_1, \dots, u_n) \in \Omega_n$. Partons des valeurs disponibles $1, \dots, n$ et effectuons la reconstruction de b , de $k = n$ jusqu'à $k = 1$.

À chaque étape, $s[k]$ est bien défini et retiré des valeurs disponibles. Aucune valeur n'est utilisée deux fois et toutes les valeurs sont finalement placées : le tableau construit appartient à $\mathcal{S}_n$.

Lorsque la valeur de $s[k]$ est choisie, il reste exactement u_k valeurs disponibles plus grandes qu'elle. Elles seront toutes placées dans les cases $1, \dots, k - 1$. Les valeurs déjà placées sont à droite de k et n'interviennent pas dans U_k . On obtient donc $U_k(s) = u_k$ pour chaque k . Cela prouve la surjectivité.

La question b. impose une unique valeur à chaque étape de toute reconstruction d'un tableau de ce code ; deux tableaux possédant le même code sont donc identiques. Cela prouve l'injectivité et la bijectivité.

Méthode 2 — Injectivité et égalité des cardinaux

La reconstruction de b . montre déjà l'injectivité. D'autre part, $\mathcal{S}_n$ contient les $n!$ permutations des entiers $1, \dots, n$. Dans Ω_n , la coordonnée u_k possède exactement k valeurs possibles, choisies sans contrainte entre les coordonnées. Ainsi :

$$|\Omega_n| = \prod_{k=1}^n k = n! = |\mathcal{S}_n|.$$

Une application injective entre deux ensembles finis de même cardinal est surjective : son image a le cardinal du domaine, donc le cardinal de tout le codomaine. L'application est donc bijective.

La première méthode donne explicitement le tableau correspondant à chaque code ; la seconde déduit la surjectivité d'un dénombrement. Dans les deux cas, pour tout code admissible, le nombre de tableaux qui le réalisent vaut exactement 1.

6. Lois des variables aléatoires $U_1, U_2, \dots, U_n$

6. (a) Loi conjointe

L'espace $\mathcal{S}_n$ comporte $n!$ tableaux équiprobables. D'après la bijection précédente, pour chaque $(u_1, \dots, u_n) \in \Omega_n$, un unique tableau réalise toutes les égalités $U_k = u_k$. Donc :

$$P(U_1 = u_1, \dots, U_n = u_n) = \frac{1}{n!}.$$

Pour un n -uplet non admissible, cette probabilité est nulle.

6. (b) Lois marginales et indépendance mutuelle

Fixons $k \in \{1, \dots, n\}$ et $u_k \in \{0, \dots, k-1\}$. Dans Ω_n , si la coordonnée d'indice k est fixée, les autres coordonnées peuvent être choisies de

$$\prod_{r \neq k} r = n!/k$$

façons. Chaque code correspond à un tableau de probabilité $1/n!$. Donc :

$$P(U_k = u_k) = \frac{n!/k}{n!} = \frac{1}{k}.$$

Ainsi U_k suit la loi uniforme sur $\{0, 1, \dots, k-1\}$. En particulier $U_1 = 0$ presque sûrement.

Pour tout code admissible :

$$P(U_1 = u_1, \dots, U_n = u_n) = \frac{1}{n!} = \prod_{k=1}^n \frac{1}{k} = \prod_{k=1}^n P(U_k = u_k).$$

Si une coordonnée est hors de son support, la probabilité conjointe et le produit des probabilités marginales sont tous deux nuls. La factorisation vaut donc pour toutes les valeurs possibles. Par la caractérisation de l'indépendance des variables discrètes par leurs probabilités ponctuelles, les variables $U_1, \dots, U_n$ sont mutuellement indépendantes.

6. (c) Moments du nombre d'échanges et équivalents

Notons N_n le nombre aléatoire d'appels de echanger. D'après 4. (a) :

$$N_n = V_n = \sum_{k=1}^n U_k.$$

Les U_k possèdent exactement les lois des X_k de la partie 2 et sont mutuellement indépendantes. Les résultats de II.3. (a) donnent donc :

$$E(N_n) = \frac{n(n-1)}{4}, \quad V(N_n) = \frac{n(n-1)(2n+5)}{72}.$$

On peut aussi les relire directement comme

$$\sum_{k=1}^n (k-1)/2$$

et

$$\sum_{k=1}^n (k^2 - 1)/12;$$

l'indépendance est nécessaire pour la seconde somme de moments. Pour $n = 1$, $N_1 = 0$ et les deux formules donnent 0.

Pour les équivalents, on calcule les quotients lorsque n tend vers l'infini :

$$\frac{E(N_n)}{n^2/4} = 1 - \frac{1}{n} \rightarrow 1.$$

$$\frac{V(N_n)}{n^3/36} = \frac{(n-1)(2n+5)}{2n^2} = \left(1 - \frac{1}{n}\right) \left(1 + \frac{5}{2n}\right) \rightarrow 1.$$

Donc :

$$E(N_n) \sim \frac{n^2}{4}, \quad V(N_n) \sim \frac{n^3}{36}.$$

Relevé des adaptations

- II.2. (a) : ajout de la condition $n \geq 2$ pour que les deux variables X_1 et X_2 figurent dans la famille. Le cas $n = 1$ est conservé dans la formule générale de II.2. (b).
- III, algorithme introductif et questions 1., 3. (b), 4. (a), 6. (c) : code adapté en Python et nom de la fonction d'échange harmonisé. Le comportement du tri et les indices $1, \dots, n$ sont conservés ; une case d'indice 0 inutilisée est explicitement prévue.
- III.2. (a) : boucle adaptée en Python ; `range(1, n)` parcourt exactement les indices $1, \dots, n - 1$.
- III.4. (c) : demande de programme convertie en Python, avec spécification des tableaux et des valeurs renvoyées ; la création d'un tableau d'entiers par `np.zeros(..., dtype=int)` est expliquée dans le sujet.