Séance Python — Préparation à l'oral Agro/Véto

Séance machine de 2h — tous les exercices sont inspirés des exemples du concours 2023–2025

Organisation de la séance. La séance est divisée en deux parties.

- **Partie A** (1h – exercices guidés) : quatre thèmes fréquents à l'oral dans la partie de l'oral de l'Agro avec préparation. **A préparer sur ordinateur.**
- **Partie B** (1h – exercices flash) : des exercices courts dans le style de l'**exercice non préparé** du concours (5 à 10 min en conditions réelles).

Conseils. Testez chaque fonction avec un exemple numérique avant de passer à la suite. Nécrivez des commentaires que s'ils vous semblent nécessaires pour votre oral.

— PARTIE A : Exercices avec préparation —

Thème 1 Simulation d'une loi et estimation par la loi des grands nombres

Ex 1 :

1) Écrire une fonction `expo(lam)` qui simule une variable aléatoire suivant la loi exponentielle de paramètre $\lambda > 0$, en utilisant uniquement `rd.random()`.

   Si $U \hookrightarrow \mathcal{U}([0,1])$, alors $X = -\dfrac{1}{\lambda}\ln(1-U)$ suit une loi exponentielle de paramètre λ.

2) Écrire une fonction `minXY(lam, mu)` qui renvoie une simulation de $\min(X,Y)$ où $X \hookrightarrow \mathcal{E}(\lambda)$ et $Y \hookrightarrow \mathcal{E}(\mu)$ sont indépendantes.
   (Utiliser `expo`.)

3) Écrire une fonction `estime_E(lam, mu, N)` qui calcule une valeur approchée de $\mathbb{E}[\min(X,Y)]$ par simulation avec N réalisations.
   (On rappelle : la moyenne empirique de N réalisations approche l'espérance.)

4) On admet que $\min(X,Y) \hookrightarrow \mathcal{E}(\lambda+\mu)$. Vérifier numériquement votre résultat pour $\lambda = 1$, $\mu = 2$ et $N = 100\,000$. *(L'espérance exacte vaut $\dfrac{1}{\lambda+\mu}$.)*

Thème 2 Chaînes de Markov discrètes

Ex 2 : On considère une suite de variables aléatoires $(Y_n)_{n\in\mathbb{N}}$ à valeurs dans $\{0,1,2,3\}$, avec $Y_0 = 1$. La transition est la suivante : sachant $Y_n = k$, la variable Y_{n+1} suit la loi binomiale $\mathcal{B}\left(3, \dfrac{k}{3}\right)$.

1) Écrire une fonction `transition(k)` qui simule Y_{n+1} sachant $Y_n = k$: elle effectue 3 tirages de Bernoulli de paramètre $\dfrac{k}{3}$ et renvoie leur somme.

> Comparer `rd.random()` à $\frac{k}{3}$ pour chaque tirage. Un tirage vaut 1 si `rd.random() < k/3`, et 0 sinon.

2) Écrire une fonction `chaine(n)` qui simule $Y_0, Y_1, \ldots, Y_n$ et renvoie la liste `[Y0, Y1, ..., Yn]`.

3) Écrire une fonction `estime_E_Yn(n, N)` qui approche $\mathbb{E}[Y_n]$ par simulation de N chaînes. Vérifier numériquement que $\mathbb{E}[Y_n] = 1$ pour tout n.

4) *(Bonus)* Écrire une fonction `proba_absorption(N)` qui estime, par simulation de N chaînes, la probabilité que la chaîne atteigne 0 ou 3 avant l'étape 10.

Thème 3 Matrices et numpy

Ex 3 : 1) Écrire une fonction `matrice_K(n)` qui renvoie la matrice $K_n \in \mathcal{M}_{n+1}(\mathbb{R})$ définie par :

$$(K_n)_{i,i+1} = i \quad \text{pour } i \in \{1,\ldots,n\}, \qquad (K_n)_{j+1,j} = -n-1+j \quad \text{pour } j \in \{1,\ldots,n\},$$

tous les autres coefficients étant nuls. On représente la matrice par une **liste de listes** (sans numpy).

> Pour initialiser une matrice nulle $(n+1)\times(n+1)$, **ne pas écrire** `[[0]*(n+1)]*(n+1)` : toutes les lignes sont alors des **alias** du même objet en mémoire. Modifier `M[0][1]` modifie aussi `M[1][1]`, `M[2][1]`, etc. Utiliser à la place une double boucle `for`.

2) Vérifier pour $n = 1$ et $n = 2$ que votre fonction renvoie bien

$$K_1 = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \quad \text{et} \quad K_2 = \begin{pmatrix} 0 & 1 & 0 \\ -2 & 0 & 2 \\ 0 & -1 & 0 \end{pmatrix}.$$

3) Écrire une fonction `valeurs_propres(n)` qui utilise `numpy.linalg.eigvals` pour calculer les valeurs propres de K_n, et qui affiche le résultat arrondi à 2 décimales.
(`import numpy as np` ; convertir la liste de listes en tableau avec `np.array(M)`.)

4) Appeler `valeurs_propres(n)` pour $n \in \{1, 2, \ldots, 6\}$. Que conjecture-t-on sur les valeurs propres de K_n ?

5) Écrire une fonction `gram(famille)` qui prend en argument une liste de vecteurs (chacun étant une liste de réels) et renvoie la matrice de Gram de la famille, c'est-à-dire la liste de listes G avec $G[i][j] = \langle u_i, u_j \rangle$.
(Le produit scalaire usuel : $\langle u, v \rangle = \sum_k u_k v_k$.)

Thème 4 Polynômes représentés par des listes

Ex 4 : Dans tout cet exercice, un polynôme $P = a_0 + a_1 X + \cdots + a_n X^n$ est représenté par la liste de ses coefficients `[a0, a1, ..., an]`.

Exemples : `[1, 0, -2]` représente $1 - 2X^2$; `[3, -2, 0, 5, 0]` représente $3 - 2X + 5X^3$ dans $\mathbb{R}_4[X]$.

1) Écrire une fonction `eval_poly(P, a)` qui renvoie $P(a)$.

Méthode directe : parcourir `P` avec `enumerate` et calculer $\sum_k P[k] \cdot a^k$.

Méthode de Hörner : factoriser de l'intérieur vers l'extérieur. Par exemple pour un polynôme de degré 3 :

$$a_0 + a_1x + a_2x^2 + a_3x^3 = a_0 + x\big(a_1 + x(a_2 + x \cdot a_3)\big).$$

On initialise un accumulateur `res = P[n]` *(dernier coefficient)*, puis on parcourt les coefficients **de droite à gauche** : à chaque étape, `res = P[k] + a * res`.

2) Vérifier : `eval_poly([1, 0, -2], 3)` doit renvoyer -17.

3) Écrire une fonction `derivee(P)` qui renvoie la liste des coefficients de P'.
(Si $P = \sum a_k X^k$ alors $P' = \sum_{k\geqslant 1} k a_k X^{k-1}$.)

4) Vérifier : `derivee([3, -2, 0, 5, 0])` doit renvoyer `[-2, 0, 15, 0]`.

5) Écrire une fonction `mul_X_moins_a(P, a)` qui renvoie la liste des coefficients du polynôme $(X - a) \cdot P(X)$, où $P \in \mathbb{R}_n[X]$ est donné comme une liste de longueur $n + 1$.
($(X-a)\cdot\sum_{k=0}^{n} a_k X^k = \sum_{k=0}^{n} a_k X^{k+1} - a\sum_{k=0}^{n} a_k X^k$. Le résultat est de degré $n+1$: liste de longueur $n+2$.)

6) Écrire une fonction `poly_H(n)` qui renvoie la liste des coefficients de $H_n(X) = \prod_{i=0}^{n-1}(X - i) = X(X-1)(X-2)\cdots(X-n+1)$.
(Partir de $H_0 = [1]$ et multiplier itérativement par $(X - i)$ avec `mul_X_moins_a`.)

— PARTIE B : Exercices flash (style concours) —

Méthode de travail pour la partie B. À l'oral, avant tout codage, les candidats qui réussissent prennent le temps de dégager la logique de l'algorithme — au tableau ou à l'ordinateur, selon leur préférence. Pendant ce TD, utiliser le **brouillon** : traiter un ou deux exemples numériques à la main, identifier ce que parcourt la boucle, la condition d'arrêt, ce que renvoie la fonction. **Coder seulement ensuite** : une fois la logique claire, l'écriture est rapide et les erreurs rares.

Ex 5 : Écrire une fonction `gene(n)` qui renvoie une chaîne de n caractères formée aléatoirement et équiprobablement des caractères `'A'`, `'C'`, `'G'`, `'T'`.

Puis écrire une fonction `nbAC(g)` qui prend en entrée une chaîne formée de ces quatre caractères et renvoie le nombre d'occurrences de la sous-chaîne `'AC'`.

Exemple : `nbAC('GAGCACCCTACTTGGCGCGA')` doit renvoyer `2`.

Ex 6 : On dispose d'une liste `L` dont les éléments sont des entiers compris entre 0 et k (donné en paramètre). **L'utilisation de `max` et `count` est interdite.**

Écrire une fonction `plus_frequent(L, k)` qui renvoie l'élément le plus fréquent de `L` (ou l'un d'eux en cas d'égalité).

(Construire une liste de compteurs `freq` de longueur $k+1$ avec une boucle.)

Exemple : `plus_frequent([0, 4, 0, 1, 4], 4)` peut renvoyer `0` ou `4`.

Ex 7 : Écrire une fonction `marche(i, N)` qui simule une marche aléatoire sur $\mathbb{Z}$: le marcheur démarre en i et à chaque pas avance de $+1$ ou -1 avec probabilité $\frac{1}{2}$ chacun. La marche s'arrête après N pas et la fonction renvoie la position finale.

Puis écrire une fonction `jeu(n, a, b, i)` qui simule le jeu suivant sur un plateau de $n+1$ cases numérotées $0, 1, \ldots, n$: un pion part de la case i, se déplace aléatoirement (gauche ou droite avec probabilité $\frac{1}{2}$) et s'arrête quand il atteint la case 0 (gain a) ou la case n (gain b). La fonction renvoie le gain.

Ex 8 : Écrire une fonction `somme_cumul(L)` qui prend en entrée une liste d'entiers L et renvoie une liste M de même longueur telle que

$$M[i] = \sum_{k=0}^{i} L[k] \quad \text{pour tout indice } i.$$

Exemple : `somme_cumul([3, 1, 4, 1, 5])` doit renvoyer `[3, 4, 8, 9, 14]`.

Ex 9 : Écrire une fonction `integrale(f, a, b, N)` qui renvoie une valeur approchée de $\displaystyle\int_a^b f(t)\,\mathrm{d}t$.

Vérifier sur $\displaystyle\int_0^1 t^2\,\mathrm{d}t$ *(vous devez pouvoir calculer la valeur exacte)* avec $N = 10\,000$.

Ex 10 : Écrire une fonction `attend_PPF()` (sans argument) qui simule des lancers d'une pièce équilibrée (`1` = Pile, `0` = Face) jusqu'à l'obtention de la configuration **Pile, Pile, Face** et qui renvoie le nombre de lancers nécessaires.

À l'aide de cette fonction et d'une boucle, estimer le temps moyen d'attente de cette configuration.

Ex 11 : On appelle diviseur strict de n tout diviseur k de n avec $1 \leqslant k < n$. Un entier n est dit *parfait* si la somme de ses diviseurs stricts vaut n.

1) Écrire une fonction `s(n)` qui renvoie la somme des diviseurs stricts de n.
2) Écrire une fonction `prop_parf(N)` qui renvoie la proportion des entiers de $\{1, \ldots, N\}$ qui sont parfaits.

(`n % k` donne le reste de la division de n par k ; k divise n ssi `n % k == 0`.)

Ex 12 :

1) Écrire une fonction `election1(L)` prenant une liste `L` ne contenant qu'une ou deux valeurs distinctes, et renvoyant l'élément majoritaire (`None` en cas d'égalité).
   Exemple : `election1(['A','B','A','B','B'])` renvoie `'B'`.
2) Écrire une fonction `election2(L)` prenant une liste quelconque et renvoyant la **liste** des éléments ayant obtenu le maximum de voix (plusieurs en cas d'égalité).
   Exemple : `election2(['O','G','M','O','M'])` renvoie `['O','M']` (ou `['M','O']`).

Si vous avez terminé avant la fin de la séance, relisez vos fonctions en les commentant comme si vous les expliquiez à voix haute à un examinateur.

Prochaines séances. Il restera à traiter : la méthode de Newton et la dichotomie, la simulation des lois usuelles (géométrique, Poisson, normale), les tris de listes, les manipulations de chaînes de caractères ...