M1 IM-MF/MPA **LOO**
Université Côte d'Azur **2023-24**

Rattrapage Python

Consignes:

- Nommer les fichiers comme demandé dans les énoncés
- Faites des sauvegardes régulières de votre travail
- Envoyer les fichiers python *.py ou *.ipynb dans un mail aux adresses suivantes: simon.girel@univ-cotedazur.fr, gilles.scarella@univ-cotedazur.fr.

Tous les documents sont autorisés mais votre travail doit être personnel.

Les exercices sont indépendants et peuvent être traités dans n'importe quel ordre.

Le script de vos fichiers *.py ou *.ipynb doit contenir l'importation de tous les packages et modules nécessaires à sa compilation.

Durée: 2h15

---

1 Probabilités et marche aléatoire [5 points]

Le code devra être écrit dans le fichier ***Marche.py***.

On introduit ci-dessous la matrice $P \in \mathbb{M}_n(\mathbb{R})$, qui décrit les probabilités de transition d'une marche aléatoire sur un graphe circulaire à n noeuds.

$$P = \begin{pmatrix} \alpha & \frac{1-\alpha}{2} & 0 & \cdots & \cdots & 0 & \frac{1-\alpha}{2} \\ \frac{1-\alpha}{2} & \alpha & \frac{1-\alpha}{2} & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \cdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \frac{1-\alpha}{2} & \alpha & \frac{1-\alpha}{2} \\ \frac{1-\alpha}{2} & 0 & \cdots & \cdots & 0 & \frac{1-\alpha}{2} & \alpha \end{pmatrix}$$

On trouve donc la valeur α sur la diagonale de P et la valeur $(1-\alpha)/2$ sur sa sous-diagonale, sa sur-diagonale, et ses coefficients $P_{0,n-1}$ et $P_{n-1,0}$.

1. Définir la fonction python *MarcheAleat* prenant en argument un entier n et un réel *alpha* et renvoyant la matrice P de taille $n \times n$. On pourra utiliser plusieurs boucles *for*.
2. Créer la matrice $P = MarcheAleat(n = 6, alpha = 1/2)$.

3. Créer un vecteur $X0$, de taille 6, dont les coefficients sont tirés aléatoirement selon la loi uniforme $\mathcal{U}_{[0,1]}$.

4. Renormaliser $X0$ pour que la somme de ses coefficients soit égale à 1. On pourra utiliser la fonction *numpy.sum(V)* qui renvoie la somme des éléments d'un vecteur V.

5. Créer le tableau X de taille 6×30 ne contenant que des 0

6. À partir de $X0$, on construit la suite $(X_k)_{k\geqslant 0}$ définie par la relation de récurrence $\forall k \geqslant 0,\ X_{k+1} = PX_k$. Remplir la première colonne de X avec le vecteur $X0$ puis remplir la matrice X de sorte que, pour tout $1 \leqslant j \leqslant 29$, sa colonne d'indice j contienne le vecteur X_j. Pour vérification, tous les coefficients de la dernière colonne de M devraient être proche de 0.166.

2 Processus aléatoire de naissance et de mort - résolution d'EDO

Le code devra être écrit dans le fichier ***Processus.py***.

Classe ProcessM [3,5 points]

Dans cet exercice, on définit la classe *ProcessM*, qui possède trois attributs :

- c : un réel strictement positif appelé taux de croissance intrinsèque.
- L : un réel strictement positif appelé capacité de charge.
- X : une liste d'entiers naturels.

Cette classe permet de construire des processus markoviens de naissance et mort en temps discret dont la croissance est caractérisée par les paramètres c et L et dont les valeurs seront stockées dans la liste X. Plus précisément, on calculera une itération du processus de la manière suivante :

$$\forall i \geqslant 0,\ X[i+1] = \begin{cases} X[i] + 1 \text{ si } u_i < \dfrac{L}{L + X[i] - 1}, \\ X[i] - 1 \text{ sinon,} \end{cases} \tag{1}$$

où u_i est tiré selon la loi uniforme $\mathcal{U}_{[0,1]}$.

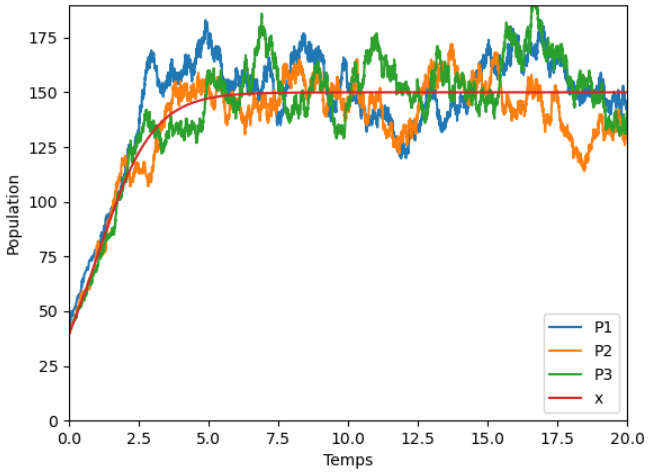

Figure 1: Exemple de figure à obtenir

1. Créer la classe *ProcessM* et définir son constructeur, qui prend trois arguments en plus de *self*. Ce constructeur permet de définir les attributs c, L et X. On donnera à ces attributs les valeurs par défaut $c = 1$, $L = 100$ et $X = [10]$.

2. Définir la méthode *iteration* qui, à partir du dernier élément de la liste X et de la relation (1), calcule le terme suivant du processus et l'ajoute à la liste X.

3. Redéfinir l'opérateur <= afin que, si P et Q sont deux instances de la classe *ProcessM*, $P <= Q$ retourne *True* si le dernier élément de l'attribut X de P est inférieur ou égal au dernier élément de l'attribut X de Q, et *False* sinon.

4. Définir l'opérateur d'affichage de la classe *MProcess* qui permettra d'afficher les attributs c et L d'une instance ainsi que la longueur de son attribut X comme ci-dessous :

```
>>> P=ProcessM(3.234,6.324,[1,2,3,2]) ; print(P)
Processus de Markov en temps discrets de longueur 4 et de
paramètres c= 3.23 et L=6.32.
```

Héritage de classe [3,5 points]

On introduit désormais la classe *ProcessMCont* qui permet de construire des processus de Markov à temps continu.

5. Créer la classe *ProcessMCont*, qui hérite de la classe *ProcessM* et définir son constructeur, qui prend quatre arguments en plus de *self* : les trois premiers sont ceux de la classe ProcessM et le quatrième argument définit l'attribut T, par défaut initialisé à $[0]$. L'attribut T contient dans une liste la suite des temps auxquels le processus change de valeur, cette suite est définie par

$$\forall i \geqslant 0,\ T[i+1] = T[i] - \frac{L\log(v_i)}{cX[i](L+X[i]-1)} \tag{2}$$

où v_i est tiré selon la loi uniforme $\mathcal{U}_{[0,1]}$.

6. Définir la méthode *iterationTmax* qui prend en argument (en plus de *self*) un réel positif $Tmax$ et qui, à partir des relations (1-2) complète les listes X et T jusqu'à ce que le dernier temps de la liste T soit strictement supérieur à $Tmax$.

7. Créer trois instances $P1$, $P2$ et $P3$ de la classe *ProcessMCont* ayant les mêmes attributs respectifs $c = 1$, $L = 150$, $X = [40]$ et $T = [0]$ puis appeler la méthode *iterationTmax* sur chaque instance, avec le même paramètre $Tmax = 20$.

8. Tracer sur la même figure les trois processus stochastiques définis par les instances $P1$, $P2$ et $P3$ en utilisant les attributs T et X en abscisse et en ordonnée respectivement.

Résolution d'équation différentielle [2 points]

On se propose enfin de comparer les processus générés et la solution de l'équation différentielle

$$\begin{cases} x'(t) = cx(t)\left(1 - \frac{x(t)}{L}\right). \\ x(0) = 40, \end{cases} \tag{3}$$

9. Résoudre en utilisant *scipy.integrate.odeint* l'EDO (3) sur l'intervalle de temps $[0, 20]$ avec les paramètres $c = 1$ et $L = 150$. L'intervalle $[0, 20]$ sera discrétisé en un tableau de 100 points.

10. Sur la même figure qu'en question 8, tracer la solution x de l'équation différentielle (3)

11. Limiter l'affichage de l'axe des abscisses (resp. des ordonnées) à l'intervalle $[0, 20]$ (resp. $[0, 190]$) et ajouter le label 'Temps' (resp. 'Population') sur l'axe. Ajouter également une légende permettant d'identifier chaque courbe x, $P1$, $P2$ et $P3$. Votre figure complète devrait ressembler à la Figure 1.

3 Lois de probabilité et matplotlib [6 points]

Dans cet exercice, on considère trois variables aléatoires X_1, X_2, X_3 telles que

- X_1 suit $\mathcal{E}(1/4)$ (loi exponentielle d'espérance 4)
- X_2 suit $\mathcal{U}(4, 13)$ (loi uniforme sur $[4, 13]$)
- X_3 suit $\mathcal{P}(8)$ (loi de Poisson de paramètre $\lambda = 8$)

L'objectif est d'obtenir la figure 2 en utilisant un subplot de matplotlib. Cette figure contient sur la première ligne trois nuages de points correspondant respectivement aux tirages de X_1, X_2 et X_3. La deuxième ligne du subplot contient les histogrammes de X_1, X_2 et X_3 normalisés par la taille de l'échantillon, c'est à dire leurs densités empiriques.

Si vous n'arrivez pas obtenir la figure demandée, essayez de vous en rapprocher le plus possible.

Dans un fichier ***Proba.py*** :

1. Définir un tableau numpy D, de taille 17, discrétisant l'intervalle $[-1, 15]$.
2. Définir une matrice M, tableau numpy à 200 lignes et 3 colonnes dont la $i-$ème colonne contient 200 réalisations indépendantes de X_i pour $i = 1, 2, 3$.
3. Définir une liste L à trois éléments contenant des chaînes de caractères correspondant aux titres des figures de la première ligne du *subplot*. Par exemple, pour X_1, on utilise la moyenne et l'écart-type empiriques calculés à partir de l'échantillon de X_1 contenu dans M, pour obtenir un titre de la forme suivante.

   ```
   Loi exponentielle - moyenne 4.30 - ec. type. 4.39
   ```

   On utilisera les éléments de L pour le titre de l'affichage des nuages de points (première ligne du subplot).
4. Ecrire le code permettant d'obtenir le subplot demandé (vous pouvez utiliser deux boucles for, une pour chaque ligne du subplot). Pour l'axe des ordonnées des nuages de points, utiliser les valeurs min et max du tableau D. Pour les histogrammes de la seconde ligne, on utilisera D pour définir la largeur des barres. Pour l'affichage des titres, on utilisera la syntaxe suivante (pour limiter la taille du texte, on pourra utiliser la syntaxe `plt.title(L[i], fontsize=6)`).
5. Ecrire le code permettant de d'afficher la densité théorique des lois continues $\mathcal{E}(1/4)$ et $\mathcal{U}(4, 13)$. L'affichage devra être superposé aux histogrammes précédemment tracés, en trait continu rouge. On choisira un maillage suffisamment fin pour que la courbe ait l'air continue (par exemple 100 points).

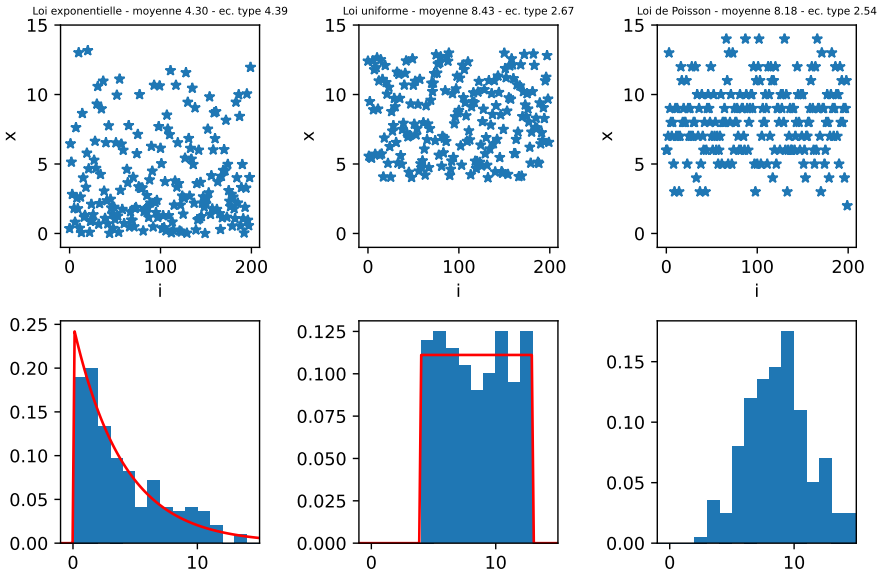

Figure 2: Exemple de rendu pour l'exercice 3 pour $\mathcal{E}(1/4)$, $\mathcal{U}(4,13)$ et $\mathcal{P}(8)$