TP5 - Python - Noté

SUJET B

Consignes:

- Nommer les fichiers comme demandé dans les énoncés
- Si vous utilisez jupyter ou cocalc, 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, avec le sujet [M1 LOO] TP noté python.

  Vous pouvez aussi envoyer vos fichiers dans une archive (au format .tar, .zip, .rar, .gz). N'oubliez pas de fichiers!

Tous les documents sont autorisés mais votre travail doit être personnel. Vous ne pouvez pas changer de sujet sous peine d'obtenir 0.

Les exercices sont indépendants et peuvent être traités dans n'importe quel ordre. Le barême est indicatif et non définitif.

Tous les exemples demandés, mêmes basiques, doivent figurer dans votre code. S'il manque des exemples, vous n'aurez pas la totalité des points même si vos fonctions sont justes.

Durée: 2h15 (3h si tiers temps)

---

1 Modèle de Leslie [7pts]

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

1. Définir une fonction *MatLeslie* qui prend en argument V et B, deux tableaux *numpy* (qui contiendront des *floats* strictement positifs).

   Lors de l'appel de la fonction, B est supposé contenir un élément de plus que V. Si cette condition est respectée, la fonction renverra la matrice de Leslie suivante (où v_i et b_i sont les coefficients de V et B) :

$$M = \begin{pmatrix} b_0 & b_1 & \dots & \dots & b_n \\ v_0 & 0 & \dots & \dots & 0 \\ 0 & v_1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & v_{n-1} & 0 \end{pmatrix} \tag{1}$$

Si la condition n'est pas respectée, *Leslie* affichera "V est de taille n, donc B doit être de taille $n+1$" (où n est à remplacer par la longueur du V passé en argument) et ne renverra rien.

2. On pose $q = 11$. Créer la matrice $M = MatLeslie(V, B)$ où V et B sont deux tableaux *numpy* de tailles respectives $q-1$ et q, respectivement initialisés avec des tirages aléatoires de lois χ^2_3 (loi du chi2 à 3 degrés de liberté) et $\mathcal{E}(2)$ (loi exponentielle d'espérance $1/2$). Si vous n'y parvenez pas, remplissez V et B de n'importe quels coefficients > 0 et continuez.

   On considère la suite de vecteurs suivante

$$Y_0 \in \mathbb{R}^q, \forall n \geq 0,\ Y_{n+1} = MY_n \tag{2}$$

3. Créer un tableau *numpy* $Y0$ de longueur q dont les coefficients sont tirés aléatoirement selon la loi uniforme $\mathcal{U}_{0,30}$ (si vous n'y parvenez pas, utilisez n'importe quels coefficients > 0 et continuez).

4. On pose $p = 50$. Créer un tableau *numpy* Y à q lignes et p colonnes dont tous les coefficients sont des zéros à l'exception de la colonne 0, qui contient $Y0$.

5. A partir de la relation de récurrence (**??**) initialisée par $Y_0 = Y0$, remplir les colonnes du tableau Y afin que $\forall i = 1, ..., p-1$, la i-ème colonne de Y contienne le vecteur Y_i.

6. On obtient les valeurs propres et vecteurs propres de M grâce au code suivant (*values* contient les valeurs propres et *vectors* les vecteurs)

   ```
   values , vectors = scipy.linalg.eig(M)
   ```

   Créer la variable d correspondant au module de la plus grande valeur propre de M (on pourrait montrer que $d \in \mathbb{R}^+_*$).

7. On peut montrer que d est de multiplicité géométrique 1 et que ses vecteurs propres sont réels. On peut accéder à un vecteur propre associé grâce au code

   ```
   vectors[:,np.argmax(values)].real
   ```

   Créer le tableau P qui contiendra l'unique vecteur propre associé à d dont la somme des coefficients soit égale à 1.

8. Créer le vecteur S de taille p tel que $\forall 0 \leq i \leq p-1$, $S[i]$ soit égal à la somme des éléments de la $i-$ème colonne de Y.

9. On peut montrer que $S[i+1]/S[i]$ tend vers d lorsque i tend vers l'infini et que $Y_i/S[i]$ tend vers P. Sur une première figure, afficher $S[i]$ en fonction de i. Légender les axes. Un exemple est visible sur la Figure **??**, attention toutefois : les figures dépendent des tirages aléatoires précédents ! Vous pouvez obtenir des croissances/décroissances exponentielles, et des convergences plus ou moins rapides.

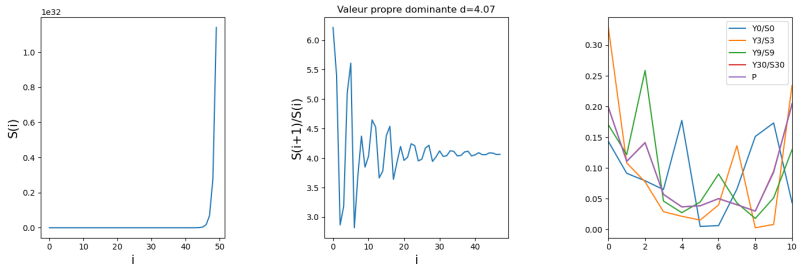

Figure 1: Exemple de figure à obtenir

10. Dans une nouvelle figure ou un subplot (au choix), afficher $S[i+1]/S[i]$ en fonction de i pour $i = 0, ..., p-2$. Légender les axes et ajouter le titre suivant : "taux d'accroissement et valeur propre dominante d=..." où la valeur de d devra apparaître (exemple Figure **??**).

11. Dans une nouvelle figure ou un subplot, afficher $Y_i/S[i]$ pour $i = 0, 3, 9$ et 49. Afficher également P sur le même graphe. On prendra soin d'ajouter une légende pour identifier chaque courbe. Normalement, Y_{49} et P sont proches, voire confondus (exemple Figure **??**).

2 Variables aléatoires, intégration, méthode du rejet [7pts]

Le code de cet exercice devra être écrit dans le fichier ***RejetB.py***.

2.1 Loi normale [2pts]

1. Définir un tableau *numpy* N contenant un échantillon de 300 réalisations indépendantes d'une variable aléatoire de loi normale de moyenne 6 et d'écart-type 2.5.

2. Calculer et afficher moyenne, médiane et variance empiriques de l'échantillon précédent sous la forme suivante (les valeurs doivent apparaître comme des *floats* à 2 décimales) :

   ```
   Cet échantillon a une moyenne E=..., une variance V=...
   et une médiane M=...
   ```

3. En utilisant la fonction subplot de *matplotlib.pyplot*, reproduire (à fluctuations aléatoires près) la Figure **??** dans laquelle le premier graphe est le nuage de points correspondant à l'échantillon N en utilisant le marqueur rond. Le deuxième graphe correspond à la densité de la loi et le troisième à sa fonction de répartition. Pour l'axe des abscisses des deux derniers

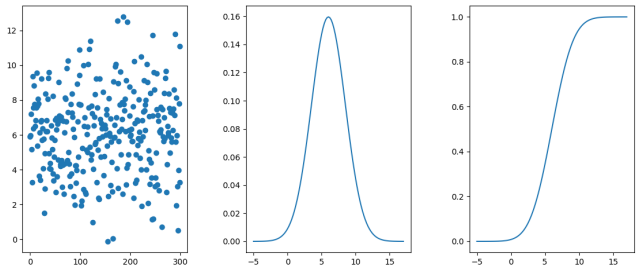

Figure 2: Figure à obtenir

graphes, on utilisera une discrétisation de l'intervalle $[-5, 17]$ avec un pas $h = 0.2$. Si les subplots se chevauchent, on pourra améliorer l'affichage en ajoutant la commande suivante (*plt* désigne *matplotlib.pyplot)* :

```
plt.tight_layout()
```

2.2 Loi à support compact [5pts]

Dans la suite de l' exercice on souhaite construire des fonctions python permettant de simuler des réalisation d'une variable aléatoire X de densité f à support compact.

Indication : dans toute la suite de cet exercice, la notation g sera toujours utilisée pour désigner une fonction $\mathbb{R} \to \mathbb{R}$, positive, continue par morceaux et qui n'est pas la fonction nulle. a et b seront toujours utilisés pour désigner deux *floats* et on supposera $a < b$.

1. Créer une fonction python *DensiteCompacte* qui prend en argument g, a et b. Elle renverra la fonction f, appelée densité de probabilité associée à la fonction g sur $[a, b]$, et définie par

$$f(s) = \begin{cases} 0 & \text{si } s < a \\ 0 & \text{si } s > b \\ g(s) / \left(\int_a^b g(x) \mathrm{d}x \right) & \text{si } a \leq s \leq b \end{cases}$$

   Le calcul numérique de l'intégrale pourra se faire avec la méthode *quad* de *scipy.integrate*.

   Attention, on ne demande pas de renvoyer l'évaluation de f en quelque(s) point(s) de l'intervalle $[a, b]$ mais bien la fonction f (un objet python de type *function*). Pour cela, on sera amené à définir la fonction f dans le corps de la fonction *DensiteCompacte*. De plus, afin que la fonction f renvoyée puisse prendre un vecteur comme argument, la fonction *DensiteCompacte* ne renverra pas directement f mais une version "vectorisée" de f, pour cela on utilisera `return np.vectorize(f)` (si *numpy* est importé sous le nom *np*) et non `return f`.

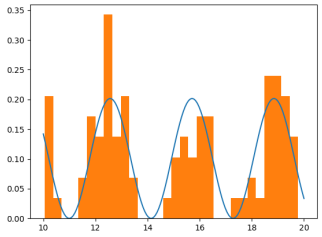

Figure 3: Figure à obtenir

2. Créer la fonction python *Maximum* qui prend en argument g, a et b. Cette fonction construit une discrétisation régulière de l'intervalle $[a, b]$ en 100 points et renvoie la valeur maximum prise par la fonction g sur cette discrétisation.

3. On veut mettre en place l'algorithme du rejet, qui permet de simuler la réalisation d'une variable aléatoire de densité donnée f. L'algorithme du rejet procède ainsi

   (a) Tirer u_1 et u_2, deux réalisation indépendantes de loi $\mathcal{U}_{[0,1]}$.

   (b) Poser $x = a + (b - a)u_1$ et $y = u_2 \max_{a \leq s \leq b} f(s)$.

   (c) Si $f(x) \geq y$, alors x est une réalisation d'une variable aléatoire de densité f. Sinon, revenir à l'étape (a).

   Créer une fonction *AlgoRejet* qui prend en argument g, a et b. Cette fonction détermine la densité f associée à g sur $[a, b]$ et renvoie une réalisation d'une variable aléatoire de densité f obtenue par l'algorithme du rejet (le maximum de f étant estimé par la fonction *Maximum*). On pourra penser à utiliser une boucle *while*

4. Créer deux *floats* $a1 < b1$ et une fonction $g1$ (positive, continue par morceaux, non identiquement nulle) de votre choix. Grâce à votre fonction *DensiteCompacte*, créer la densité $f1$ associée à $g1$ sur $[a1, b1]$.

5. Créer une discrétisation régulière I de l'intervalle $[a1, b1]$ en 150 points. Tracer le graphe de $f1$ sur la discrétisation I.

6. Grâce à votre fonction *AlgoRejet*, créer un tableau *numpy* $T1$ contenant un échantillon de 90 réalisations indépendantes d'une variable aléatoire de densité $f1$ et tracer, sur la même figure qu'à la question précédente, la densité empirique de cet échantillon sous la forme d'un histogramme à 30 barres (*bins*) dont l'aire doit être égale à 1. Un exemple est proposé pour $g = cos^2$ sur $[10, 20]$ sur la figure **??**.

3 Résolution d'EDO - Processus aléatoire de naissance et de mort [6pts]

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

3.1 Résolution d'équation différentielle [1.5pts]

Soit l'équation différentielle ordinaire autonome

$$\begin{cases} y'(t) = 3y(t) - \frac{3y(t)^2}{120}. \\ y(0) = y_0, \end{cases} \tag{3}$$

1. Résoudre en utilisant *scipy.integrate.odeint* l'EDO (**??**) sur l'intervalle de temps $[0, 25]$ avec $y_0 = 7$. L'intervalle $[0, 25]$ sera discrétisé en un tableau de 100 points.

2. Tracer la solution y de l'équation différentielle (**??**). Limiter l'affichage de l'axe des abscisses (resp. des ordonnées) à l'intervalle $[0, 25]$ (resp. $[0, 150]$) et ajouter le label 'Temps' (resp. 'Population') sur l'axe des abscisses (resp. ordonnées). On doit obtenir la Figure **??**.

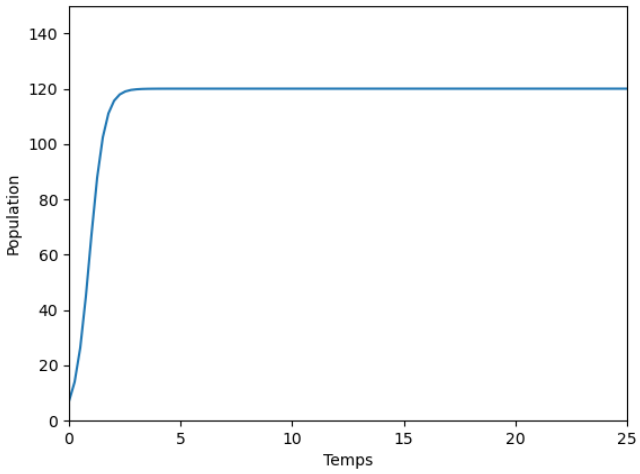

Figure 4: Exemple de figure à obtenir

3.2 Classe NaissMort [4.5pts]

On considère la classe *NaissMort*, qui possède deux attributs :

- C : un réel strictement positif appelé capacité de charge.

- X : une liste d'entiers naturels (ou un tableau *numpy*).

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

$$\forall i \geq 0,\ X[i+1] = \begin{cases} X[i]-1 \text{ si } u_i < \dfrac{X[i]-1}{C+X[i]-1}, \\ X[i]+1 \text{ sinon,} \end{cases} \tag{4}$$

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

1. Créer la classe *NaissMort* et définir son constructeur, qui prend deux arguments en plus de *self* : C et X . Ce constructeur permet de définir les attributs C et X avec les valeurs par défaut $C = 120$ et $X = [7]$.

2. Définir la méthode *iteration* qui, à partir du dernier élément de la liste X et de la relation (**??**), calcule le terme suivant du processus et l'ajoute à la liste X. *iteration* ne prend pas d'autre argument que *self*. Pour ajouter un terme à une liste L, afin de tenir compte du caractère mutable de celle-ci, il est recommandé d'éviter d'utiliser l'opérateur `L+=...` et de lui préférer la syntaxe `L=L+...`

3. Redéfinir l'opérateur $>=$ afin que, si $P1$ et $P2$ sont deux instances de la classe *NaissMort*, $P1 >= P2$ retourne *True* si le dernier élément de l'attribut X de $P1$ est supérieur ou égal au dernier élément de l'attribut X de $P2$, et *False* sinon.

4. Définir l'opérateur d'affichage de la classe *NaissMort* qui permettra d'afficher l'attribut C d'une instance ainsi que la longueur de son attribut X comme ci-dessous :

   ```
   >>> P=NaissMort(6.32,[1,2]) ; print(P)
   Processus de naissance et mort en temps discret de longueur 2 et de
   paramètre C=6.32.
   ```

5. Créer deux instances $P1$ et $P2$ de la classe *NaissMort* ayant les mêmes attributs respectifs $C = 120$ et $X = [7]$. À l'aide d'une boucle, exécutez 100 fois la méthode *iteration* sur $P1$ et sur $P2$.

6. Testez l'opérateur d'affichage sur $P1$, puis afficher le résultat de la comparaison de $P1$ et $P2$ avec l'opérateur $>=$.