Math ECG 2. 2022-2023

Mathématiques Appliquées - F. Gaunard
http://frederic.gaunard.com
ENC Bessières, Paris 17e.

Révisions (1)

Retour sur le programme avec Python

.

L'objectif de ces activités aux énoncés très peu détaillés est de forcer une certaine modélisation des problèmes en allant chercher dans l'ensemble du cours tous les éléments pour le faire et proposant ainsi un premier retour sur le programme tout en forçant aussi à se mettre (enfin) à Python.
Comme toute proposition de révision, il serait naïf de la croire exhaustive.

Activité 1 - Le paradoxe des anniversaires

L'objectif de cette première activité est de visualiser l'évolution de la probabilité qu'au moins deux personnes d'un groupe de n personnes fêtent leur anniversaire le même jour, en fonction du nombre d'individus dans le groupe.
On suppose, pour simplifier, qu'une année n'est constituée que de 365 jours et que les naissances sont réparties uniformément tout au long de l'année. On s'intéresse à la date mais pas à l'année (exemple: 3 Janvier).
Déterminer graphiquement avec Python à partir de quelle valeur n, la probabilité qu'au moins deux personnes soient nées le même jour dépasse $1/2$. Même question avec 99%.

Activité 2 - La ruine du joueur

José se rend au casino *Les requins de la côte* avec s euros en poche ($s \in \mathbb{N}^\star$) et l'envie de faire fortune. Après s'être vêtu de ses habits de lumière, il s'installe à une table où, à chaque partie, il gagne avec probabilité $p \in]0;1[$ un euro et perd avec probabilité $q = 1 - p$ un euro.

On note N la somme dont dispose le casino (on peut raisonnablement supposer que $s < N$). José décide qu'il arrêtera de jouer s'il devient ruiné (c'est à dire lorsque sa fortune tombe à 0) ou lorsque ce sera le cas pour le casino. On admet (même si on pourrait le montrer) que le jeu s'arrête à un moment presque sûrement.

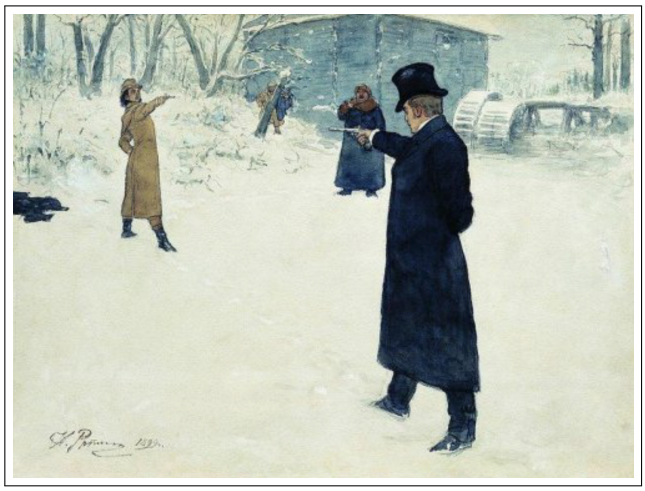
(1) Écrire une première fonction d'en-tête `def casino(s, N, p):` qui à chaque appel renvoie le graphe de l'évolution de la fortune de José jusqu'à l'arrête du processus.
(2) On introduit la variable aléatoire T qui prend la valeur 1 si José est ruiné et 0 si le casino est ruiné. Établir une conjecture quant à la loi de T.

Activité 3 - Le truel

Trois gentlemen, Mr. White, Mr. Grey et Mr. Black se retrouvent opposés dans un *truel*, où chaque adversaire tire à son tour jusqu'à ce qu'il n'en reste qu'un.

Les trois hommes n'ont pas les mêmes niveaux de tir; Mr. Black fait mouche à chaque tir alors que Mr. Grey ne touche que deux fois sur trois. Quant à Mr. White, il ne réussit son tir qu'une fois sur trois. Les hommes sont des gentlemen, ils laissent donc Mr. White tirer en premier, puis Mr. Grey et enfin Mr. Black.

Mr. Black visera toujours, tant que celui-ci est vivant, Mr. Grey, et Mr. Grey essaiera également toujours de toucher Mr. Black. Mr. White en revanche se demande s'il doit, commencer par faire tomber Mr. Black ou Mr. Grey voire tirer en l'air et les laisser s'entretuer, donnant ainsi lieu à trois stratégies, numérotées 1, 2 et 3.



Duel between Onegin and Lenski, Ilya Repin, 1899. Pushkin Museum, Moscou. Domaine public.

Comparer les trois stratégies avec Python, *via* simulation d'un grand nombre de duels pour chacune des trois stratégie et émettre une conjecture sur celle la plus intéressante à suivre pour Mr. White.

Activité 4 - Matrices aléatoires

On considère deux variables aléatoires X et Y (définies sur un même espace probabilisé) et la matrice

$$N = \begin{pmatrix} 1 & X \\ 1 & Y \end{pmatrix}.$$

On s'intéresse à l'estimation, grâce à Python, de la probabilité que N soit inversible.
Essayer de répondre à cette question dans le cas où

(i) X et Y sont deux lois géométriques indépendantes de même paramètre p;
(ii) X et Y sont deux lois binomiales indépendantes de mêmes paramètres n et p;
(iii) X est une loi uniforme sur $[\![1, n]\!]$ et, sachant $[X = k]$, Y est une loi uniforme sur $[\![1; k]\!]$.

Activité 5 - Tri et pioche sans remise

Une urne contient n boules numérotées de 1 à n et indiscernables au toucher. On effectue successivement k tirages **sans remise** dans cette urne (avec $1 \leq k \leq n$). On note X le deuxième plus petit numéro obtenu à l'issue de l'expérience.

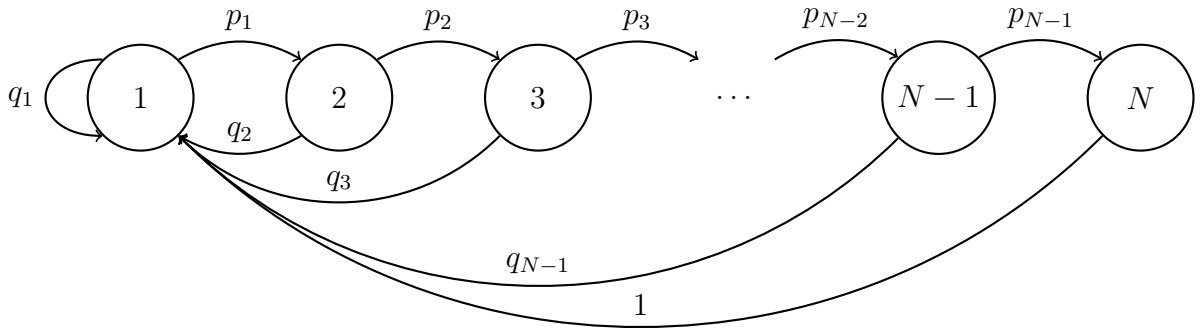
(1) Écrire une fonction d'en-tête `def selection(U):` qui prend en argument une liste de valeurs `U` et renvoie un élément `k` sélectionné dans cette liste mais aussi la nouvelle liste `L` des valeurs restantes.
(2) Écrire alors une fonction permettant de simuler X.

Activité 6 - Le *Rogue-like*

Soit N un entier supérieur ou égal à 2.
On considère un joueur (ou une joueuse) qui joue à un *rogue-like*. Le rogue-like est un sous-genre de jeu vidéo dans lequel le joueur explore un donjon infesté de monstres qu'il doit combattre pour progresser vers la dernière salle, contenant le *boss final* et la promesse de récompenses épiques. Une des caractéristiques du rogue-like est que toute mort est définitive, ce qui oblige le joueur à recommencer du début s'il souhaite continuer à jouer. Nous considérons dans la suite un joueur soumis aux règles suivantes :

- Le joueur commence au *niveau* 1 et a pour objectif d'atteindre le niveau N afin d'y affronter le boss final et de le vaincre.
- A chaque niveau, le joueur effectue un unique *combat* qu'il peut soit gagner soit perdre.
  Plus précisément :
  - pour tout $i \in [\![1, N-1]\!]$, on note $p_i \in]0,1[$ la probabilité que le joueur gagne le combat effectué au niveau i, et on pose $q_i = 1 - p_i$. On suppose que $p_i \geq p_{i+1}$.
  - si le combat mené au niveau $i \in [\![1, N-1]\!]$ est gagné, le joueur passe au niveau supérieur, sinon, le joueur meurt et retourne au niveau 1.
  - quelque soit l'issue du combat effectué au niveau N, le joueur recommence au niveau 1 ensuite.
- On appelle *partie* toute séquence de jeu commençant au niveau 1 et se terminant soit par la mort du joueur, soit par la victoire du joueur contre le boss final.
- Le joueur enchaîne indéfiniment les parties, même en cas de victoire contre le boss final.

On représente cette chaîne de Markov à N états par le graphe probabiliste :



On admet que toutes les v.a sont définies sur un même espace probabilisé. On note :

- X_n la v.a. égale au numéro du niveau où se trouve le joueur lors de son $n-$ième combat. En particulier, $X_1 = 1$.
- Y_j la v.a. égale au niveau maximal atteint lors de la $j-$ième partie jouée.
- Z la variable aléatoire égale au numéro de la partie où le joueur combat le boss final pour la première fois.
- T la variable aléatoire égale au numéro du premier combat effectué contre le boss final.

(1) Écrire une première fonction `def Etape(P,a):` qui prend en argument une liste de probabilités `P` $= [p_1, p_2, ..., p_{N-1}]$ et un entier a donnant l'état de la chaîne à un instant donné et qui renvoie l'état à l'instant suivant.
(2) En déduire l'écriture d'une fonction permettant de simuler X_n, puis une autre permettant de simuler Z et une dernière permettant de simuler T.
(3) Écrire un programme permettant d'observer que, dans le cas où toutes les probabilités sont égales (*i.e.* pour tout i, $p_i = p$), on a

$$E(T) \sim \frac{1}{(1-p)p^{N-1}}, \quad N \to +\infty.$$

Activité 7 - Jurassic Park

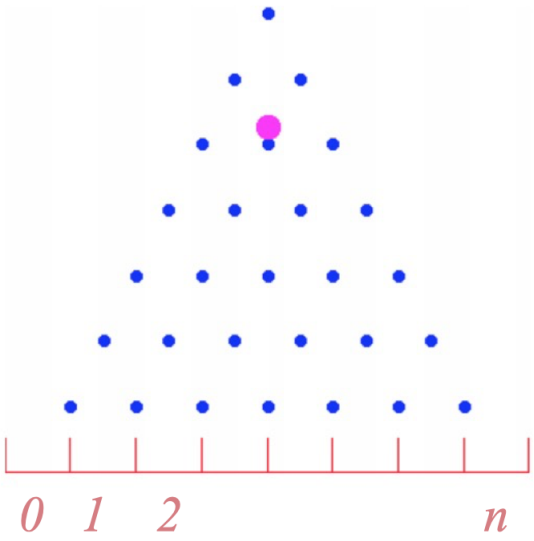
Une équipe d'archéologues travaille sur des ossements d'un même spécimen de dinosaure, retrouvés sur différents sites, au quatre coins du monde. Afin de mieux comprendre l'anatomie de ce sympathique animal, ils procèdent aux mesures de la longueur (en cm) de leur humérus et de leur fémur. Les données collectées sont enregistrées dans un fichier nommé dino.csv.

(1) Importer le fichier de données sous forme d'un tableau noté `data`.
(2) Écrire une commande qui permet de récupérer la longueur du fémur du dinosaure avec l'humerus le plus grand.
(3) Représenter le nuage de points du couple statistique. Proposer une approximation de la longueur du fémur en fonction de la longueur de l'humérus.

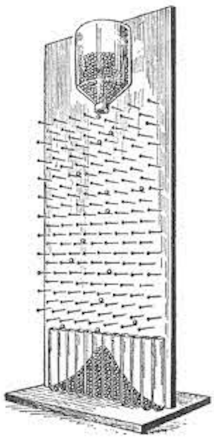
Activité 8 - Planche de Galton

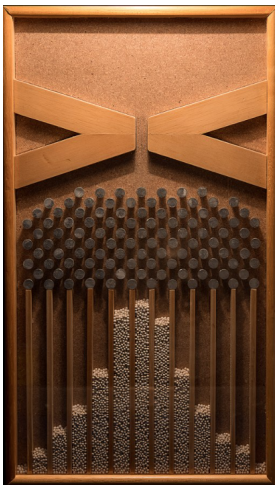
Des clous sont plantés sur la partie supérieure d'une planche, en n rangées. Chaque rangée de clous contient un clou de plus que la précédente; la première en contient un, la $n-$ième exactement n.

On lâche une bille au "sommet" de la pyramide de clou, de telle sorte que chaque fois qu'elle rencontre un clou, celle-ci passe aléatoirement et uniformément soit à droite soit à gauche pour chaque rangée de clous.
Dans la partie inférieure de la planche, $n+1$ colonnes, numérotées de gauche à droite de 0 à n, sont disposées pour "récolter" les billes lachées.



On lâche une bille et on note X_n la variable correspondant au numéro de la colonne dans laquelle se trouve la bille après avoir franchi les rangées de clous.





Proposer un programme permettant de simuler la chute de 1000 billes sur une planche de Galton à n rangées de clous. Le résultat devra permettre d'observer ou d'illustrer une convergence en loi.

Activité 9 - Théorème de Zeckendorf

On rappelle que la suite de Fibonacci (F_n) est la suite de nombre réels définis par

$$\begin{cases} F_0 &= 0 \\ F_1 &= 1 \\ F_{n+2} &= F_{n+1} + F_n \end{cases}$$

Les premiers termes de la suite sont $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55...$

Théorème

Le **Théorème de Zeckendorf** dit la chose suivante.
Pour tout entier non nul n, il existe un unique entier k et un unique $k-$uplet d'entiers $(c_1, .., c_k)$ vérifiant

- $c_1 \geq 2$
- $c_i + 1 < c_{i+1}$

tels que

$$n = \sum_{i=1}^{k} F_{c_i}$$

Cette décomposition de n en somme de nombres de la suite de Fibonacci est appelée *décomposition de Zeckendorf de n.*

Par exemple $17 = 13 + 3 + 1 = F_7 + F_4 + F_2$ donc $k = 3$ et $(c_1, c_2, c_3) = (2, 4, 7)$.

Le but de cet exercice est d'écrire un programme qui renvoie cette décomposition pour un nombre n pris en argument.

(1) Écrire une fonction `def fibo(k):` qui renvoie le terme F_k de la suite (F_n).

(2) Écrire une fonction `def tri(L):` qui renvoie la liste des termes de `L` triés par ordre croissant.

(3) Écrire une fonction `def recherche(x, L):` qui prend en argument un réel x et une liste `L` (déjà triée dans l'ordre croissant, de première terme inférieur ou égal à x et de dernier terme strictement supérieur à x) et qui renvoie le plus grand élément de la liste inférieur ou égal à x.

(4) Écrire une fonction `def Zeckendorf(n):` qui renvoie la décomposition de Zeckendorf de n. *On pourra commencer par écrire la liste des nombres de Fibonacci inférieurs à n (ainsi que le premier strictement supérieur) et faire une boucle descendante sur cette liste.*