Université de Bordeaux
L2 Informatique/Maths-Informatique

Probabilités, Statistiques, Combinatoire : TM3

Simulations

L'objectif de cette feuille est de *simuler* des expériences aléatoires d'une part, et d'*observer* le résultat de ces expériences, en les répétant un grand nombre de fois, d'autre part.

Pour la génération aléatoire, on utilise le module `random` chargé avec la commande `import random` placée au début de votre fichier.

On utilisera deux fonctions du module `random` :

- `random.randint(a,b)`, un générateur aléatoire qui renvoie un nombre entier choisi au hasard (uniformément) dans $[[a, b]]$.
- `random.random()`, qui retourne un flottant compris entre 0 et 1, selon la "loi uniforme sur $[0, 1]$" : la probabilité de retourner une valeur située dans n'importe quel intervalle $[x, y]$ (avec $0 \leq x \leq y \leq 1$) est $y - x$, la longueur de l'intervalle.

Vous allez écrire beaucoup de fonctions **sans paramètres**, dont le but est de simuler une expérience aléatoire : à chaque appel de la fonction, celle-ci retournera un résultat possiblement différent (et les résultats d'appels successifs sont considérés comme indépendants ; c'est ce qui est plus ou moins garanti par les générateurs pseudo-aléatoires de la bibliothèque `random`). Il est important de comprendre la différence, pour une fonction `simulation()`, entre deux écritures :

- `simulation()` représente le résultat d'un *appel* à la fonction : celle-ci retourne une valeur, possiblement nouveau à chaque appel ;
- `simulation` (sans les parenthèses) représente la fonction elle-même ; on peut passer une fonction à une autre comme paramètre, comme dans beaucoup de langages (en `C` cela passe par un pointeur de fonction).

3.1 Première mise en jambes : Anatole et Philémon

(Inspiré du sujet de DS 2019 ; au lieu de faire des calculs, on fait des simulations)

Anatole et Philémon jouent à un jeu de hasard : ils tirent à tour de rôle à pile ou face, avec une même pièce équilibrée. Ils jouent jusqu'à ce qu'un joueur tire Pile ; ce joueur gagne la partie. C'est Anatole qui joue en premier.

1. Une fonction `SimulePileOuFace()`, qui retourne 0 (Pile) ou 1 (Face) avec probabilité 1/2 chacune, est fournie. Écrire une fonction `SimuleJeu()`, qui simule une partie du jeu décrit ci-dessus en faisant appel à `SimulePileOuFace()`, et retourne 1 si c'est Anatole qui gagne, ou 0 si c'est Philémon. On admettra que la probabilité que personne ne gagne est de 0, *i.e.* on fera comme si les parties ne pouvaient pas durer éternellement.
2. En simulant un grand nombre (quelques milliers) de parties du jeu, et en calculant la proportion de parties gagnées par chaque joueur, *deviner* la probabilité que ce soit Anatole qui gagne.

3.2 Deuxième mise en jambes : le joueur de casino

Un joueur entre dans un casino avec k euros en poche, et décide de jouer à un jeu jusqu'à ce qu'une des deux conditions suivantes soit remplie :

— soit il est ruiné (il n'a plus aucun euro)

— soit sa fortune a atteint n euros.

Les entiers k et n, avec $1 \leq k < n$, sont des paramètres du problème.

Le jeu est un jeu simple de hasard : chaque fois qu'il joue, il mise exactement 1 euro ; avec probabilité p il récupère sa mise et gagne un euro de plus (donc au total, sa fortune augmente de 1) ; avec probabilité $1-p$, il perd sa mise (sa fortune diminue de 1). Le réel p est un second paramètre du problème ; $p = 1/2$ correspond à un jeu équitable, $p < 1/2$ à un jeu favorisant le casino.

La soirée peut donc se terminer de deux façons différentes seulement (on admet que le joueur a toujours assez de temps pour atteindre soit 0, soit n euros) : soit il ressort "riche", soit il ressort "pauvre".

Écrire une fonction `SimuleCasino(p,k,n)`, qui simule une soirée du joueur et retourne 1 si le joueur "réussit" (à sortir avec n euros), ou 0 si le joueur "échoue" (il ressort sans euros). En comptant le nombre de "victoires" sur un grand nombre d'essais, *deviner* une formule donnant la probabilité de victoire en fonction de k et n, pour le cas particulier $p = 1/2$ (au moins pour le cas $k = 1$). Ici aussi, on admettra que la probabilité que la "partie" dure à l'infini est nulle.

3.3 Jeux de ballon

Nous allons simuler des matches de différents sports de balle, dans un modèle très simple.

Dans un certain nombre de sports de balle (tennis, volley, tennis de table), on joue un certain nombre de coups ; à chaque coup, un des deux joueurs (ou équipes) *sert* (met la balle en jeu) et, à la fin du coup, l'un des joueurs gagne le point ; le vainqueur du match est le résultat d'une séquence de coups, selon des règle de "comptage des points" qui varient d'un jeu à l'autre.

— Au tennis de table, celui qui remporte un coup marque un point ; le premier qui atteint 11 points avec au moins 2 points d'avance remporte un set ; le premier qui remporte un nombre prédéterminé de sets remporte le match. Dans un set, chaque joueur sert pour 2 coups consécutifs, puis c'est à son adversaire de servir ; lorsque l'on atteint 10-10, le service change tous les coups.

— Au volley, le principe est le même mais le nombre de points à remporter pour un set est de 25 points ; dans un même set, le camp qui vient de marquer sert.

— Jusqu'en 1998, la marque au volley était différente : un point n'était marqué que si le camp qui servait remportait le coup ; pour le camp adverse, remportait le coup ne lui permettait que de récupérer le service (le nombre de points pour un set était plus bas, 15 seulement).

— Au tennis, il y a trois niveaux de marque : le premier qui remporte 4 points avec 2 points d'avance, remporte un jeu ; le premier qui remporte 6 jeux avec 2 jeux d'avance, remporte un set ; le premier qui remporte un nombre prédéterminé de sets, remporte le match. Le même joueur sert pour tous les coups d'un même jeu, puis le service alterne.

Vous allez écrire des programmes qui simulent ces systèmes de marque, sous l'hypothèse

(extrêmement simplificatrice) que les différents coups d'un match sont indépendants. Dans un premier temps, on supposera qu'un joueur donné a toujours la même probabilité p de remporter chaque coup ; par la suite on affinera la simulation en introduisant deux paramètres p et q : p représentera la probabilité de remporter un coup lorsque l'on sert, q la probabilité de remporte un coup lorsque l'adversaire sert. (Normalement, $p > q$ sauf au volley, où servir est censé être un désavantage car c'est l'adversaire qui a la première occasion d'attaquer.)

1. Écrire une fonction `SimuleCoup(x)`, qui retournera le résultat d'un coup du point de vue du joueur concerné : 1 si le joueur remporte le coup, 0 sinon. Le paramètre sera la probabilité de remporter le coup pour le joueur.
2. Écrire une fonction `SimuleSetTable(x)`, qui retourne le résultat d'un set de tennis de table pour un joueur dont la probabilité de remporter chaque coup est de x. La fonction `SimuleSetTableSimplifie(x)` pourra servir d'exemple ; elle simule un set, sans la règle des deux points d'avance.
3. En simulant un grand nombre (plusieurs milliers) de sets, estimer la probabilité de remporter un set si l'on a probabilité 0.7, 0.6, 0.55, 0.51 de remporter chaque coup. Que devrait être (exactement) cette probabilité pour $x = 0.5$? Estimer aussi l'effet de la règle des deux points d'écart sur la probabilité de remporter un set.
4. Écrire de même une fonction `SimuleMatchTable(x)`, qui simule un match en 3 sets gagnants pour un joueur qui a probabilité x de remporter chaque coup. Déterminer expérimentalement la probabilité de remporter un match en fonction de la probabilité de remporter chaque coup.
5. Faire de même pour le tennis (avec une fonction pour la simulation d'un jeu, une fonction pour la simulation d'un set, une fonction pour la simulation d'un match en 3 sets gagnants).

   Déterminer expérimentalement à partir de quelle probabilité de remporter chaque balle, un joueur a 9 chances sur 10 de remporter un match en 3 sets gagnants.
6. Reprendre vos fonctions, et en écrire des versions `Bis` avec deux paramètres p et q à la place de x : p représentera la probabilité que le joueur gagne le point s'il sert, q la probabilité qu'il gagne le point si c'est son adversaire qui sert ; on considèrera que le joueur sert la première balle du match. Ajouter une fonction pour simuler l'ancienne règle du volley, où seule l'équipe qui sert peut marquer un point.
7. On peut considérer que les deux adversaires sont de force égale si on prend $q = 1 - p$ (ainsi, quelque soit le joueur qui sert, sa probabilité de remporter le coup est la même) ; typiquement on prendra $p > 1/2$ sauf pour le volley, où il est plus réaliste de prendre $p < 1/2$. En prenant $p = h + 1/2$ pour de petites valeurs de h, estimer expérimentalement (pour chaque jeu) l'avantage conféré par le fait de servir en premier.

3.4 Pour les curieux : le Monty Hall

On va s'efforcer de simuler le jeu télévisé du Monty Hall. Rappelons-en le principe : le candidat se trouve face à trois portes, dont l'une, choisie aléatoirement, cache un cadeau précieux - les deux autres cachent un "cadeau" sans valeur. Le candidat peut désigner une porte de son choix, ce après quoi l'animateur du jeu (qui sait quelle porte est la bonne) ouvre une porte et révèle ce qu'elle cache ; cette porte est toujours une porte non désignée par le

candidat, et ce n'est jamais la porte cachant le cadeau précieux (si l'animateur a le choix, on suppose qu'il choisit aléatoirement la porte à ouvrir). Le candidat doit alors faire un choix de quelle porte ouvrir – et de gagner le cadeau qu'elle cache.

La question, qui défraya la chronique, est de savoir quelle stratégie le joueur devrait adopter. Citons trois possibilités :

— il peut ouvrir la porte initialement désignée ;
— il peut choisir systématiquement d'ouvrir la porte qu'il n'a pas désignée (parmi celles qui n'ont pas été ouvertes)
— il peut choisir aléatoirement entre la porte initialement désignée, et l'autre porte non encore ouverte.

On ne va pas ici chercher à modéliser l'expérience et à analyser la probabilité de gagner le cadeau précieux ; on va plutôt programmer les différentes stratégies, et les simuler.

Vous allez donc écrire plusieurs familles de fonctions :

— `initJeu()`, qui choisit aléatoirement et retourne le numéro de la "bonne" porte (un entier de 1 à 3) ;
— `choixJoueur()`, qui représente le choix initial du joueur : elle doit retourner un entier entre 1 et 3 ;
— `animateur(bon,choix)` : représente la porte qu'ouvre l'animateur quand `bon` est le numéro de la "bonne" porte, et `choix` est le numéro de la porte désignée par le joueur ;
— `strategieJoueur(choixInitial,choixAnimateur)`, qui simule le choix du joueur quand il a initialement désigné la porte `choixInitial` et que l'animateur a ouvert la porte `choixAnimateur` ;
— `simuleJeu(iJ,cJ,a,sJ)`, qui prend en entrée les quatre fonctions précédentes (sous forme de fonctions), les utilise pour simuler le déroulement du jeu, et retourne le résultat : 1 si le joueur repart avec le cadeau de valeur, 0 sinon.

Votre travail consiste à écrire chacune des fonctions (et au moins trois versions de la fonction de stratégie du joueur), et à simuler le jeu suffisamment de fois pour trancher la polémique ! Notez qu'il est également possible de faire varier la stratégie initiale du joueur (peut-être qu'il préfère la porte 1 ?), mais aussi celle de l'animateur (à ceci près qu'il n'a jamais le droit d'ouvrir la "bonne" porte, ni celle désignée par le joueur : il n'a donc de choix que si le joueur a désigné la "bonne" porte ; mais par exemple on peut imaginer, et simuler, un animateur qui, quand il a le choix, ouvrirait toujours la porte au plus petit numéro...).