École Nationale Supérieure des Mines de Nancy
Département « Génie Industriel & Mathématiques Appliquées »
Année 2014-2015

Méthode de Monte-Carlo & Application aux Processus Aléatoires

Travail à la maison

PROBLÈME I — *L'éditeur*

Dans ce problème, on suppose connus les résultats du problème III. En particulier, on suppose qu'on dispose d'une fonction `Poisson_torpille` *(ou plus simplement* `Poisson`*) prenant un argument* $\theta \geqslant 0$ *et retournant une simulation de la loi de Poisson de paramètre* θ*.*

Ce problème concerne une maison d'édition qui souhaite s'adresser à de nouveaux investisseurs, et qui a chargé un cabinet d'audit de calculer certaines caractéristiques financières de son affaire.

Après étude statistique, les auditeurs considèrent qu'on peut modéliser le fonctionnement de cette maison d'édition sur une année de la sorte :

- *La maison d'édition reçoit* A *propositions de romans, où* A *suit une loi de Poisson de paramètre* $\Theta := 130$*.*
- *Le comité de lecture de la maison d'édition attribue à chaque proposition* ϖ *un* niveau $\ell(\varpi)$*. Le niveau de chaque proposition est indépendant, et suit une loi exponentielle de paramètre* 1*.*
- *En fonction de son niveau et du niveau des autres propositions reçues, chaque proposition est alors classée dans une catégorie qui détermine si elle sera publiée ou non, et si oui dans quelle collection :*
  - *Lorsque le niveau d'une proposition est inférieur à* $1\frac{1}{2}$*, le roman proposé est* refusé *(ce qu'on désigne par le code* `0`*).*
  - *Lorsque la proposition est de niveau supérieur à* $1\frac{1}{2}$*, mais que ce niveau est néanmoins inférieur à* 3 ou *que la proposition ne fait pas partie des 5 meilleures reçues par l'éditeur, le roman est publié en collection* standard *(code :* `1`*).*
  - *Lorsque la proposition est de niveau supérieur à* 3 *et fait partie des 5 meilleures propositions reçues, mais que son niveau est inférieur à* $4\frac{1}{2}$ ou *que ce n'est pas la meilleure proposition reçue par l'éditeur, le roman est publié en collection* supérieure *(code :* `2`*).*
  - *Sinon, càd. si le roman est de niveau supérieur à* $4\frac{1}{2}$ et *qu'il s'agit de la meilleure proposition reçue par l'éditeur, le roman est publié en* collection phare *(code :* `3`*).*

1. Je n'ai pas précisé ci-dessus si les inégalités étaient larges ou strictes. Qu'en pensez-vous ?...

Les romans que l'éditeur choisit de publier engendrent des frais d'impression et de publicité. Pour un roman publié dans la collection c, ces frais s'élèvent au montant F_c, avec :

$$F_1 := 20\,000\ €;$$
$$F_2 := 150\,000\ €;$$
$$F_3 := 1\,000\,000\ €.$$

Chaque roman publié génère ensuite des revenus, indépendamment pour chaque roman, revenus qui peuvent être modélisés par la formule :

$$e^{\alpha\ell+\beta(X-Y)}R_c,$$

où ℓ est le niveau du roman, c la collection dans laquelle il a été publiée, et X et Y sont des variables aléatoires indépendantes de loi exponentielle de paramètre 1. On donne $\alpha := 0{,}4$ et $\beta := 0{,}7$, et :

$$R_1 := 5\,000\ €;$$
$$R_2 := 25\,000\ €;$$
$$R_3 := 125\,000\ €.$$

La première partie du problème va consister à simuler le bénéfice annuel (éventuellement négatif s'il s'agit d'une perte) de l'éditeur.

2. Rappeler pourquoi on peut simuler une loi exponentielle de paramètre 1 par l'instruction « `-log(rand)` ».

3. Écrire une fonction `soumissions` qui simule le nombre de romans proposés ainsi que leurs niveaux ; cette fonction ne prendra aucun argument et renverra un vecteur-ligne de taille (aléatoire) A contenant les niveaux des romans proposés.

4. Écrire vous-mêmes une fonction `tri` qui prend en entrée un vecteur-ligne (de taille indéterminée) de réels positifs ou nuls, et qui renvoie un vecteur contenant les mêmes valeurs, mais triées de la plus grande à la plus petite, et ce en n'utilisant que des commandes élémentaires facilement transposables à n'importe quel autre langage de programmation [1]. Dans la suite de l'exercice, vous pourrez néanmoins servir plutôt d'une fonction pré-implémentée — c'est même conseillé, pour avoir un programme plus rapide.

5. Écrire une fonction `publications` qui ne prend aucun argument, simule le processus de soumission des propositions de romans, et renvoie une matrice de taille $B \times 2$, où B est le nombre (aléatoire) de romans publiés, de façon que chaque ligne de la matrice contienne, en colonne `1` la collection du roman publié et en colonne `2` son niveau.

6. Écrire une fonction `benefice` qui ne prend aucun argument et qui renvoie une simulation du bénéfice pour l'éditeur sur une année.

Maintenant qu'on sait simuler le bénéfice (qui est aléatoire), on veut déterminer quelques propriétés statistiques de celui-ci.

[1]. Autrement dit, je vous défends d'utiliser des commandes toutes faites de MATLAB qui vous mâcheraient le travail, y compris la commande permettant d'insérer un élément au milieu d'un vecteur (en décalant ceux situés après). (Bien entendu, la commande `size` pour récupérer la taille du vecteur donné en entrée est quant à elle autorisée).

7. Estimer par la méthode de Monte-Carlo l'espérance du bénéfice annuel de l'éditeur. La fonction, qui s'appellera `MC_editeur`, ne prendra aucun argument et affichera l'estimation du bénéfice annuel moyen.

8. Dans le calcul précédent, essayer de donner aussi un intervalle de confiance à 80 % (la fonction modifiée s'appellera `MC_editeur_IC`). Si je vous dis que cet intervalle de confiance n'est pas fiable, à quoi pouvez-vous le remarquer ? Et à quoi cela est-il dû, selon vous ?

9. Démontrer que, si ℓ est le niveau d'un roman publié et c la collection dans laquelle il est publié, alors le bénéfice moyen engendré par ce roman vaudra *exactement*

$$-F_c + \frac{e^{\alpha\ell} R_c}{1-\beta^2}.$$

10. En déduire une technique de conditionnement pour estimer plus précisément le bénéfice annuel moyen, où le conditionnement s'opèrera par rapport à la connaissance de combien de romans on publie, de quels niveaux, et dans quelles collections. La fonction s'appellera `MC_editeur_cond`.

11. Modifier la fonction en `MC_editeur_cond_IC` qui affiche aussi un intervalle de confiance à 80 %. Si je vous dis que cet intervalle de confiance est maintenant fiable, à quoi le voyez-vous, et comment l'expliquez-vous ? En termes de réduction de la variance, que cela signifie-t-il ?...

12. Assurez-vous, dans la mesure du possible, de la cohérence entre les résultats obtenus avec et sans conditionnement.

PROBLÈME II — *Diffusion de gaz toxique*

Dans cet exercice, on considère une expérience de chimie qui se déroule dans une pièce parallélipipédique, décrite en coordonnées cartésiennes par le pavé $[0,6] \times [0,4] \times [0,3]$ *(ces données sont exprimée en mètres). Sur tout le mur de droite (abscisse* 6*), un catalyseur a été installé sur un rectangle d'ordonnées* $[1,3]$ *et de hauteurs* $[1,2]$*. Celui-ci est conçu de sorte que, lorsqu'une molécule de gaz toxique traverse l'entrée du catalyseur, elle est instantanément désintégrée. Sur le mur de gauche, il y a une fenêtre aux jointures pas tout-à-fait étanches, que nous modéliserons pour simplifier par un trou carré dans le mur, d'ordonnées* $[2,2{,}1] \times [1{,}5,1{,}6]$ *; lorsqu'une molécule de gaz toxique traverse ce trou, elle s'échappe dans l'environnement. L'expérimentateur, situé à la position* $(5{,}5,2,1)$*, réalise une expérience qui émet des molécules de gaz toxique ; notre but est de connaitre la proportion de molécules qui sont relâchées dans l'environnement au lieu d'être détruites par le catalyseur.*

13. Faire un dessin.

Lorsqu'une molécule de gaz toxique est émise dans l'atmosphère, elle y suit ensuite une trajectoire brownienne standard. Néanmoins, la présence des murs fait que cette trajectoire sera réfléchie au niveau des murs *(en-dehors du système catalytique et du trou, s'entend) : au niveau de la simulation, cela signifie que si un pas du mouvement brownien franchit le mur, on rejette ce pas et on le refait.*

14. Écrire une fonction `diffusion` qui simule la trajectoire d'une molécule de gaz relâchée par l'expérimentateur, jusqu'à ce que celle-ci soit désintégrée par le catalyseur ou s'échappe dans l'environnement par le trou. La fonction renverra `false` dans le premier cas et `true` dans le second. Elle pourra éventuellement afficher en outre de la trajectoire de la molécule (avec la méthode de tracé du choix de l'élève), afin notamment de visualiser si le modèle est bien respecté. On prendra des pas pour le mouvement brownien qui auront un écart-type de $5\ \mathrm{cm}$ selon chaque coordonnée.

15. Déduire de la fonction précédente une fonction `MC_diffusion` qui estime par la méthode de Monte-Carlo la proportion de molécule s'échappant dans l'environnement. Cette fonction prendra en argument le nombre de simulations demandées, et reverra l'estimation de la proportion de molécules qui s'échappent, assortie d'un intervalle de confiance à 99 %.

16. Le calcul précédent est peu rapide et peu précis... Quelles pistes envisageriez-vous pour faire mieux ?...

PROBLÈME III — *Autour de la loi de Poisson*

Ce problème se propose d'étudier un peu la loi de Poisson et la façon de la simuler, notamment en vue d'applications au problème I. Dans tout cet exercice, θ désigne un réel positif quelconque, et $Poisson_\theta$ désigne (une réalisation de) la loi de Poisson de paramètre θ.

17. Montrer qu'on a pour tout $n > e\theta$ que

$$\mathbb{P}\big(Poisson_\theta = n\big) \leqslant \frac{e^{-\theta}}{\sqrt{2\pi e}},$$

et en déduire que

$$\mathbb{P}\big(Poisson_\theta > e\theta\big) \leqslant e^{-\theta}.$$

Indication — Pour la première partie de la question, on utilisera avec profit la célèbre inégalité de Stirling :

$$\forall n \in \mathbf{N} \quad n! \geqslant (2\pi n)^{1/2}(n\,/\,e)^n;$$

pour la seconde partie de la question, on pourra commencer par observer que pour tout $n > e\theta$, on a :

$$\mathbb{P}\big(Poisson_\theta = n+1\big) \leqslant \frac{1}{e}\,\mathbb{P}\big(Poisson_\theta = n\big).$$

18. Expliquer pourquoi il n'est pas raisonnable, lors d'un calcul informatique, de tenir compte d'un phénomène qui aurait un risque de se produire inférieur à 10^{-30}. En déduire que, dans le problème I, on pourra considérer sans danger que le nombre de propositions soumises à l'éditeur ne sera *jamais* plus grand que 353. Quel intérêt cela offre-t-il ?

Pour la question qui suit, on considère un processus à valeurs dans $\mathbf{N}$, indexé par le temps, issu de 0 au temps 0, qui, entre t et $t+dt$ (pour dt infinitésimal), a une probabilité dt de faire

un saut vers le haut d'amplitude 1*. C'est alors un théorème classique*[2] *que la valeur du processus en* $t = \theta$ *suit la loi* $Poisson_\theta$*.*

19. Écrire une fonction `processus_P` qui simule *exactement* le processus ci-dessus. Cette fonction prendra comme argument un temps T et simulera le processus sur $[0, T]$, affichant le graphe du processus. Déduire de la fonction `processus_P` une fonction `Poisson_naif` qui prend en argument un paramètre $\theta \geqslant 0$ et qui renvoie une simulation de la loi de Poisson de paramètre θ.

Il s'avère que la méthode ci-dessus n'est pas très efficace pour simuler des lois de Poisson de grand paramètre (disons, supérieur à 16*)... Voici un programme très efficace pour simuler une loi de Poisson de paramètre* exactement 16 *:*

```
function y = Poisson16
u = rand;
if u > .533255108612279 % Pr(y >= 16)
    if u > .990000219046895 % Pr (y >= 8)
        if u > .999906858387057 % Pr (y >= 4)
            if u > .999998086902030 % Pr (y >= 2)
                if u > .999999887464825 % Pr (y >= 1)
                    y = 0;
                    return
                else
                    y = 1;
                    return
                end
            else
                if u > .999983682399666 % Pr (y >= 3)
                    y = 2;
                    return
                else
                    y = 3;
                    return
                end
            end
        else
            if u > .998616214975237 % Pr (y >= 6)
                if u > .999599562336624 % Pr (y >= 5)
                    y = 4;
                    return
                else
                    y = 5;
                    return
                end
            else
                if u > .995993955344872 % Pr (y >= 7)
                    y = 6;
                    return
                else
                    y = 7;
                    return
                end
            end
```

[2]. En fait, normalement ce serait même plutôt la *définition* de la loi de Poisson...

```
        end
    else
        if u > .873007329933656 % Pr (y >= 12)
            if u > .956701684058134 % Pr (y >= 10)
                if u > .97801274645094 % Pr (y >= 9)
                    y = 8;
                    return
                else
                    y = 9;
                    return
                end
            else
                if u > .922603984229643 % Pr (y >= 11)
                    y = 10;
                    return
                else
                    y = 11;
                    return
                end
            end
        else
            if u > .725489076130206 % Pr (y >= 14)
                if u > .806878457539006 % Pr (y >= 13)
                    y = 12;
                    return
                else
                    y = 13;
                    return
                end
            else
                if u > .632472640234435 % Pr (y >= 15)
                    y = 14;
                    return
                else
                    y = 15;
                    return
                end
            end
        end
    end
else
    if u > .036685657816939 % Pr (y >= 24)
        if u > .187751471663162 % Pr (y >= 20)
            if u > .340656370757506 % Pr (y >= 18)
                if u > .434037576990123 % Pr (y >= 17)
                    y = 16;
                    return
                else
                    y = 17;
                    return
                end
            else
                if u > .257650854106291 % Pr (y >= 19)
                    y = 18;
                    return
                else
```

```
                y = 19;
                return
            end
        end
    else
        if u > .089226627838562 % Pr (y >= 22)
            if u > .131831965708659 % Pr (y >= 21)
                y = 20;
                return
            else
                y = 21;
                return
            end
        else
            if u > .058240927569400 % Pr (y >= 23)
                y = 22;
                return
            else
                y = 23;
                return
            end
        end
    end
else
    if u > .004105062054341 % Pr (y >= 28)
        if u > .013118562887583 % Pr (y >= 26)
            if u > .022315477981966 % Pr (y >= 25)
                y = 24;
                return
            else
                y = 25;
                return
            end
        else
            if u > .007458922829501 % Pr (y >= 27)
                y = 26;
                return
            else
                y = 27;
                return
            end
        end
    else
        if u > .001131195357155 % Pr (y >= 30)
            if u > .002188570182821 % Pr (y >= 29)
                y = 28;
                return
            else
                y = 29;
                return
            end
        else
            if u > .000567262116800 % Pr (y >= 31)
                y = 30;
                return
            end
```

```
            end
        end
    end
end
while true
    y = 31;
    % À cet endroit-là, il y a une variable que je ne fais pas figurer dans
    % le programme car on ne l'utilisera jamais directement, mais qui joue
    % un rôle essentiel dans l'algorithme: cette variable, appelée "q", est
    % initialisé à exp(-16)*16^31/(30!)/15 ; et c'est un majorant de
    % Pr(Poisson>=31).
    u = rand;
    % À cet endroit-là, il y a autre une variable que je ne fais pas
    % figurer dans le programme car on ne l'utilisera jamais directement,
    % mais qui joue elle aussi un rôle essentiel : cette variable, notée
    % "v", est définie comme valant q*u ; le point important est que, au
    % cours de la boucle, v ne changera jamais de valeur, mais restera
    % toujours égale à q*u à chaque mise à jour de q et u.
    while true
        % L'algorithme est écrit de sorte qu'à ce passage du programme, on
        % ait toujours Pr(Poisson==y) = (1-16/y)*q.
        if u > 16 / y
            return
        else
            % À cet endroit-là, q est remplacé par q*(16/y)*(y-16)/(y-15),
            % qui vaut exp(-16)*16^(y+1)/(y!)/(y-15), et qui est un
            % majorant de Pr(Poisson>=y+1).
            u = u / (16 / y) * (y - 15) / (y - 16);
            y = y + 1;
            if u > 1
                break;
                % Le "break" ci-dessus nous fait en fait repartir au début
                % de la grande boucle.
            end
        end
    end
end
end
```

20. Essayer de comprendre quelques-unes des idées du programme ci-dessus.

21. Expliquer comment combiner le programme ci-dessus et votre programme `Poisson_naif` pour simuler une loi de Poisson grand paramètre — on écrira une fonction `Poisson_torpille` ayant la même syntaxe que `Poisson_naif`, mais plus efficace.

Indication — On se rappellera avec profit que si $Poisson_\lambda$ *et* $Poisson_\nu$ *sont indépendantes, alors* $Poisson_\lambda + Poisson_\nu$ *suit la loi* $Poisson_{\lambda+\nu}$*...*