Les sorties aléatoires

Aldo GONZALEZ LORENZO*

15 octobre 2016

1 Introduction

Imaginez un pigeon bourré sur une corde. À chaque seconde, le pigeon se déplace un pas à gauche ou à droite. Comme il est bourré, il ne suit aucune logique : il fait son choix aléatoirement. Et puis, quand il repasse par sa position initiale, il retrouve sa bouteille et il arrête de bouger.

Afin de pouvoir faire quelques calculs, disons que la position initiale du pigeon est

$$x_0 = 0.$$

À chaque fois, il peut aller à droite ou à gauche, c'est-à-dire,

$$x_{i+1} = x_i + 1 \qquad \text{ou}$$
$$x_{i+1} = x_i - 1$$

Par exemple,

$$\begin{aligned} x &= (x_0, x_1, x_2, \ldots, x_6) \\ &= (0, 1, 2, 1, 2, 1, 0) \end{aligned}$$

On dit que x, la suite de positions du pigeon, est une *sortie aléatoire*.

> **Exercice :** Faites une sortie aléatoire. Utilisez une pièce pour choisir droite ou gauche à chaque fois.

On peut étudier plusieurs choses sur les sorties aléatoires. Par exemple, est-ce qu'on s'éloigne beaucoup de l'origine ? Repasse-t-on souvent par un point où on a déjà été ? Quelle est la position la plus fréquentée ?
On s'intéresse ici à la *durée* des sorties aléatoires, c'est-à-dire le nombre de pas qu'on fait avant de retourner à l'origine. Par exemple la durée de la sortie aléatoire dans l'exemple précédent est 6.

> **Exercice :** Quelle était la durée de la sortie aléatoire de l'exercice précédent ?

*aldo.gonzalez-lorenzo@univ-amu.fr

On se demande alors qu'elle est l'espérance (la valeur moyenne théorique) de la durée. Plutôt que de la calculer théoriquement ou de le faire à la main, faisons un programme.

Exercice : Écrivez un programme qui fait une sortie aléatoire, affiche les positions et affiche la durée. Par exemple,

```
Positions : -1 -2 -1 0
Durée = 4
```

Si on lance un dé, la probabilité d'avoir le numéro 1 est 1/6. Du coup, si on lance le dé 6000 fois, on devrait avoir plus ou moins 1000 fois le numéro 1. Disons qu'on a eu 954. Si on divise 954/6000 on obtient une valeur proche de 1/6. De plus, si on lance le dé 10 000 ou 100 000 fois, on obtiendra des valeurs encore plus proches de 1/6. Essayez vous-même (avec un programme en C) si vous ne me croyez pas.

Cette notion est la *loi des grands nombres*, qui dit que la *moyenne empirique* (la proportion de 1s obtenus) converge vers l'*espérance* (la moyenne théorique, 1/6 dans ce cas) quand le nombre d'expériences est grand.

Cela ne vous donne pas envie de savoir quelle est l'espérance (moyenne théorique) de la durée d'une sortie aléatoire ?

Exercice : Exécutez votre programme 10 fois et calculez la moyenne des valeurs obtenues.

Je m'attendais à une petite valeur. La moitié des sorties retournent à l'origine après 2 pas (dans le premier pas on se décale d'une position, et puis on a une chance sur deux de revenir), et par un raisonnement semblable (mais plus compliqué) on déduit que la probabilité de revenir après 4 pas est 10/16. Avant d'aller trop loin, faisons plus d'expériences

Exercice : Adaptez votre programme pour faire 100 sorties aléatoires et donner la moyenne de la durée. Je vous conseille de ne plus afficher les positions et de mettre le code qui fait la sortie aléatoire dans une fonction `int randWalk()`.

Si vous avez eu la même chance que moi, vous avez eu une moyenne assez grande. Elle devrait converger vers une valeur (l'espérance) si on incrémente le nombre d'expériences, n'est-ce pas ?

Exercice : Calculez la moyenne de la durée pour 1000, 10 000, 100 000 et 1 000 000 expériences. Vous pouvez écrire une fonction `double meanRandWalk(int n)` pour cela, où l'argument `n` est le nombre d'essais.

Vous devez maintenant être convaincu(e) que la moyenne de la durée ne converge pas. Pourquoi ? À mon avis il n'y a qu'une seule possibilité : elle est infinie !

2 Retourne-t-on toujours à l'origine ?

Voyez-vous le paradoxe ? Si en moyenne le nombre de pas avant de retourner à l'origine est infini, ça veut dire que souvent on n'y retourne jamais. Pourtant, on a déjà fait beaucoup de sorties aléatoires et on a toujours fini par revenir.

Je vous propose de changer un peu l'approche. On va faire une variante de la fonction `randWalk()` qui simule une sortie aléatoire bornée à n pas, où n est un argument de la fonction. Si on n'est pas retourné à l'origine avant n pas, alors on s'arrête et on renvoie -1.

> **Exercice :** Faites-le. Le prototype de votre nouvelle fonction est `int boundedRandWalk(int n)`, puisqu'on fait des sorties bornées.

Cette fonction est plus « prudente » que `randWalk()`, puisqu'on commence à soupçonner qu'elle ne s'arrête pas toujours. Un programme qui ne s'arrête pas pose toujours problème. Du coup, en utilisant `boundedRandWalk(int n)` on est certain de finir.

Sur mon ordinateur j'ai fait 1000 sorties bornées à 10 pas et j'ai trouvé que seulement 772 ont réussi à finir. Pourtant, cela ne veut pas forcément dire qu'elles étaient des sorties infinies.

> **Exercice :** Calculez quel pourcentage de sorties aléatoires bornées à 10 pas retournent à l'origine. Faites 10 000, 100 000 et 1 000 000 (ou même plus) essais. Est-ce que la moyenne converge ?

Sur mon ordinateur il semble que oui. Cela semble converger vers le 75,4%. C'est un pourcentage modeste, j'imagine que ça augmente quand on permet plus de pas.

> **Exercice :** Calculez le pourcentage des sorties aléatoires bornées à 100, 1000, 10 000, 100 000 et 1000 000 pas. Vous pouvez créer une fonction `double proportionBoundedRandWalk(int n)` pour cela. Qu'est-ce qui se passe ?

D'accord. En fait je suis sûr de cela, puisque c'est un théorème connu. Une sortie aléatoire retourne à l'origine *avec probabilité 1*. C'est-à-dire, si on se permet de faire des sorties aléatoires bornées à n pas, la probabilité de retourner à l'origine tend vers 1 quand $n \to \infty$. Notez par contre que cela ne veut pas dire que *toutes* les sorties aléatoires retournent à l'origine, il y en a beaucoup qui

ne le font pas. Par contre, c'est un nombre négligeable par rapport à toutes les sorties possibles.

Je peux vous donner un autre exemple dans ce style. Si on choisit aléatoirement un réel dans l'intervalle $[0, 1]$, il sera un rationnel avec probabilité 0. Pourtant, ce n'est pas du tout impossible d'en trouver un : il y en a beaucoup, même une infinité. Ces notions font partie de la *théorie de la mesure*.

3 D'autres dimensions

Attention, cette section contient des surprises.

On peut bien imaginer une sortie aléatoire en deux dimensions (2D). Le pigeon se réveille dans la position $x_0 = (0, 0)$ et à chaque fois il se déplace vers l'est, l'ouest, le nord ou le sud. C'est à dire,

$$\begin{aligned} x_{i+1} &= x_i + (1, 0) && \text{ou} \\ x_{i+1} &= x_i + (-1, 0) && \text{ou} \\ x_{i+1} &= x_i + (0, 1) && \text{ou} \\ x_{i+1} &= x_i + (0, -1) \end{aligned}$$

Exercice : Ecrivez la fonction `int boundedRandWalk2D(int n)`. Faites quelques essais en affichant les positions pour vérifier qu'elle marche correctement.

J'ai l'intuition que, puisqu'on peut se déplacer maintenant sur un plan, on va se perdre plus facilement et on va donc retourner à l'origine plus rarement.

Exercice : Calculez le pourcentage des sorties aléatoire 2D bornées à 10, 100, 1000, ... pas. Étudiez si les moyennes sont plus petites que pour le cas 1D.

Intéressant. Essayez maintenant en 3D.

4 Retour au cas unidimensionnel

Vu qu'on a écrit la fonction `proportionBoundedRandWalk(int n)`, utilisons-la pour voir comment la moyenne des durées des sorties bornées à n pas varie quand on change n.

Exercice : Créez une fonction `double meanBoundedRandWalk(int n)` qui lance 1 000 000 fois `boundedRandWalk(n)` et calcule la moyenne des durées. Attention, il faut compter seulement les sorties qui finissent avant n pas, donc celles qui renvoient un nombre positif. Si vous ne voulez pas fixer le nombre d'essais à 1 000 000, laissez-le comme argument. La fonction sera alors `double proportionBoundedRandWalk(int n, int m)`, où m dénote le nombre d'essais.

Calculez donc la moyenne des durées des sorties aléatoire bornées à 10, 100, 1000, 10 000, ... avec la fonction `meanBoundedRandWalk(int n)`. Je trouve que les moyennes sont de plus en plus grandes (pas très étonnant, vu qu'on se permet de plus en plus de pas). Par contre, cela nous donne une nouvelle vision sur le problème : les sorties aléatoires (non bornées, comme au début) sont en fait des sorties aléatoires bornées à ∞ pas. Donc, la moyenne des durées est aussi ∞.

On n'a pas démontré ce résultat, mais je crois que vous êtes au moins plus convaincu(e) maintenant. Sinon, on n'a qu'à trouver la formule qui donne l'espérance de la durée d'une sortie aléatoire bornée à n pas et voir si cela tend vers l'infini quand $n \to \infty$. La section suivante fait presque ça.

5 Assez d'expériences, quelques vrais calculs des probabilités

Cette section est plus compliquée que les autres. On va deviner théoriquement les valeurs qu'on a estimé expérimentalement, tout en utilisant l'ordinateur au lieu de notre imagination mathématique.

Si on ne peut faire que 2 pas, il y a 4 sorties possibles :

$$x = (1, 2)$$
$$x = (1, 0)$$
$$x = (-1, 0)$$
$$x = (-1, -2)$$

Il y a 2 sorties qui retournent à l'origine et 2 qui ne le font pas. Alors, la probabilité de retourner à l'origine est

$$\mathbb{P}\{\text{retourner à l'origine après 2 pas}\} = \frac{2}{4} = 0.5$$

Aussi, l'espérance de la durée d'une sortie bornée à 2 pas est 2, puisque toutes les sorties qui retournent à l'origine font 2 pas. Formellement,

$$\mathbb{E}\{\text{durée d'une sortie aléa. bornée à 2 pas}\} = 2$$

Passons maintenant à 4 pas. Parmi les 16 ($= 2^4$) sorties possibles, il y en a 10 qui retournent à l'origine, dont

— 8 qui durent 2 pas : ce sont celles qui commencent par $(1, 0, \ldots)$ ou $(-1, 0, \ldots)$

— 2 qui durent 4 pas : $(1, 2, 1, 0)$ et $(-1, -2, -1, 0)$

Alors,

$$\mathbb{P}\{\text{retourner à l'origine après 4 pas}\} = \frac{10}{16} = 0,625$$

$$\mathbb{E}\{\text{durée d'une sortie aléa. bornée à 4 pas}\} = 2 \cdot \frac{8}{10} + 4 \cdot \frac{2}{10} = 2,4$$

Le calcul de cette probabilité est facile, mais comprenez-vous le calcul de l'espérance ? La durée vaut "2" 8 fois sur 10 et "4" les autres 2 fois, donc en moyenne elle vaut 2,4. Elle est plus proche de "2" (qui est plus probable) que de "4". Si

cela n'est pas clair encore, réfléchissez-y.

Notez que la probabilité et l'espérance se sont incrémentées quand on est passé de 2 à 4 pas. Je vous propose de faire maintenant ces calculs pour un n quelconque. Appelons

$P(n) =$ « probabilité de retourner à l'origine après n pas »
$E(n) =$ « espérance de la durée d'une sortie aléa. bornée à n pas »

On pourrait essayer de le faire à la main, mais ça n'a pas l'air facile. Essayons donc avec l'ordinateur. Pour un n donné, il faudra faire toutes les sorties possibles de n pas (il y en a 2^n) et :

— Pour $P(n)$, on compte combien retournent à l'origine et on le divise par le nombre total de sorties (2^n)
— Pour $E(n)$, on somme toutes les durées et on les divise par le nombre de sorties qui retournent à l'origine.

Implémenter ceci est plus compliqué. Je vais vous guider.

Nous pouvons encoder une sortie aléatoire avec un vecteur de zéros et de uns :

$$(1, 2, 3, 2, 3, 2, 1, 0) \longleftrightarrow (1, 1, 1, 0, 1, 0, 0, 0)$$

Quand on incrémente la position, on écrit 1 ; quand on la décrémente, on écrit 0. Du coup, pour parcourir toutes les sorties de taille n, il faut parcourir tous les vecteurs de zéros et de uns de taille n. Et cela revient à écrire les nombres 0, 1, …, $2^n - 1$ en base binaire, n'est pas ? Essayons avec $n = 3$:

0	000	(-1, -2, -3)
1	001	(-1, -2, -1)
2	010	(-1, 0, -1)
3	011	(-1, 0, 1)
4	100	(1, 0, -1)
5	101	(1, 0, 1)
6	110	(1, 2, 1)
7	111	(1, 2, 3)

On peut donc parcourir tous les nombres dans l'intervalle $[0, 2^n - 1]$, les transformer en base binaire dans un vecteur de taille n et transformer celui-ci en sortie. Encore mieux, on peut faire une fonction qui prend un nombre en base binaire et l'incrémente d'un.

Exercice : Faites une fonction `int next(int a[], int size)` qui prend en nombre binaire écrit dans un tableau de taille n et l'incrémente. La fonction retourne `1` au moins que `a` soit égal à `[1, 1, ..., 1]`.

Solution :

```
int next(int a[], int size)
{
```

```
    int i = 0;
    a[i]++;
    while(a[i] > 1)
    {
        if(i+1 < size)
        {
            a[i] = 0;
            i++;
            a[i]++;
        }
        else
            return 0;
    }
    return 1;
}
```

Avec une telle fonction il n'est pas difficile de calculer, pour un n donné, $P(n)$ et $E(n)$.

Exercice : Faites la fonction `double probabilityBoundedRandWalk(int n)` qui calcule $P(n)$.

J'ai obtenu les valeurs suivantes :

n	$P(n)$
0	$0/1 = 0$
1	$0/2 = 0$
2	$2/4 = 0,5$
3	$4/8 = 0,5$
4	$10/16 = 0,625$
5	$20/32 = 0,625$
6	$44/64 = 0,6875$
7	$88/128 = 0,6875$
8	$186/256 = 0,7265625$
9	$372/512 = 0,7265625$
10	$772/1024 = 0,75390625$

Quand j'obtiens une suite de nombres, je la cherche tout de suite sur The On-Line Encyclopedia of Integer Sequences. Je n'ai rien trouvé quand j'ai tapé les numérateurs :

$$0, 0, 2, 4, 10, 20, 44, 88, 186, 372, 772$$

Par contre, quand j'ai cherché les termes pairs j'ai retrouvé la suite A068551, d'où j'ai déduit que

$$P(2n) = \frac{4^n - \binom{2n}{n}}{2^{2n}} = 1 - \frac{\binom{2n}{n}}{4^n}$$

Je suis sûr qu'un(e) bon(ne) mathématicien(ne) pourra démontrer que

$$\lim_{n\to\infty} \left(1 - \frac{\binom{2n}{n}}{4^n}\right) = 1$$

On peut faire pareil pour $E(n)$.

> **Exercice :** Faites la fonction `double expectedMeanBoundedRandWalk(int n)` qui calcule $E(n)$.

Maintenant j'ai obtenu

n	$P(n)$
2	$4/2 = 0,5$
3	$8/4 = 0,5$
4	$24/10 = 0,625$
5	$48/20 = 0,625$
6	$120/44 = 0,6875$
7	$240/88 = 0,6875$
8	$560/186 = 0,7265625$
9	$1120/372 = 0,7265625$
10	$2520/772 = 0,75390625$

De nouveau, j'ai trouvé les numérateurs pairs dans la suite A002011. Après quelques manipulations je suis arrivé à

$$\forall n > 1, E(2n) = \frac{4(2n-1)\binom{2n-1}{2n}}{4^n - \binom{2n}{n}}$$

Je soupçonne que cela tends vers l'infini.

6 Conclusions

On a réussi à trouver les formules pour $P(n)$ et $E(n)$ sans aucun raisonnement ! Il ne reste qu'à démontrer qu'elles sont justes et que $\lim_{n\to\infty} P(n) = 1$ et $\lim_{n\to\infty} E(n) = \infty$.

Voici mes conclusions :

1. Une sortie aléatoire bornée à n pas consiste à déplacer un point sur $\mathbb{Z}$ aléatoirement n fois .
2. Une sortie aléatoire (tout court) est une sortie aléatoire bornée à ∞ pas. Donc elle n'est pas du tout bornée.
3. La proportion de sorties aléatoires bornées à n pas qui retournent à l'origine tend vers 1 quand $n \to \infty$
4. En même temps, la moyenne du premier retour à l'origine tend vers ∞ quand $n \to \infty$.
5. Puisqu'il y a des ∞s dans cette histoire, on ne peut déduire rien de logique et on trouve un paradoxe. Dommage.

J'ai essayé avec ce projet de reproduire les pensées d'un scientifique qui étudie un problème. Je l'ai fait avec une approche expérimentale, où on devine la formule et après on essaie de la démontrer (ce que je n'ai pas fait).

Si vous voulez continuer à travailler sur ce sujet, je vous propose de faire tout ça pour deux et trois dimensions. Bon courage !