Université de Liège

Projet 1 - Jeu de Hasard

Éléments du Calcul des Probabilités

François Rozet (s161024)
Jules Remes (s162964)

2ème année de Bachelier Ingénieur civil

Année académique 2017 - 2018

1 Lehman & Leighton

(a) Position initiale aléatoire

Si le joueur choisit aléatoirement la position initiale de la balle, on peut supposer que ses quatre choix possibles sont équiprobables. On suppose aussi que, à chaque bifurcation, la direction prise par la bille est choisie aléatoirement et de façon équiprobable.

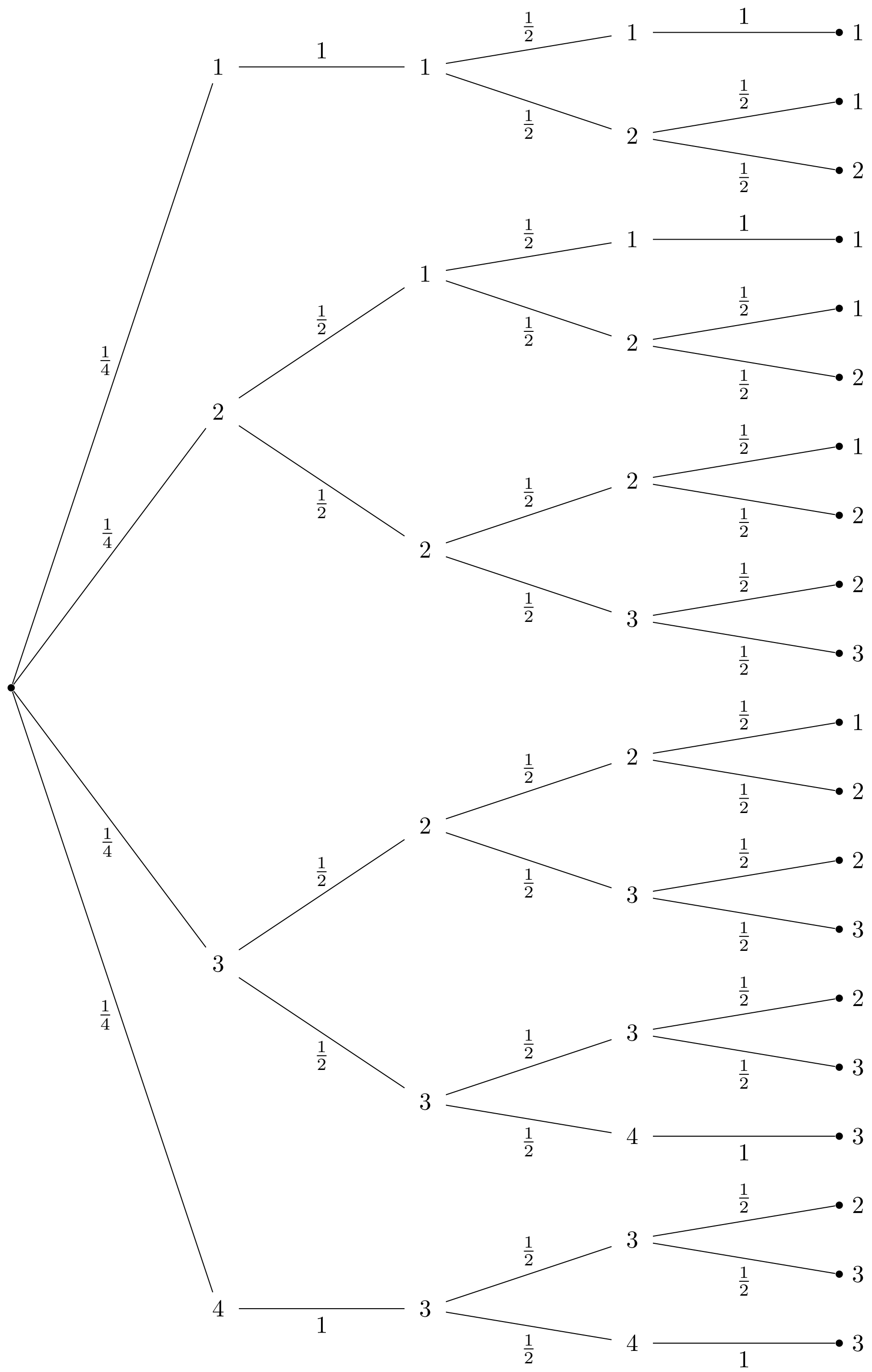

Figure 1 – Arbre de probabilité pour un départ aléatoire.

En considérant les collections d'évènements ε_1, ε_2 et ε_3 qui rassemblent, respectivement, les chemins donnant lieux aux sorties 1, 2 et 3, on obtient les probabilités d'occurrence données dans le tableau 1.

$P(A \in \varepsilon_1)$	$P(A \in \varepsilon_2)$	$P(A \in \varepsilon_3)$
$\frac{11}{32}$	$\frac{10}{32}$	$\frac{11}{32}$
0,343 75	0,3125	0,343 75

Tableau 1 – Distribution de probabilité théorique pour un départ aléatoire.

(b) Position initiale fixée

Lorsque le joueur choisit toujours la position initiale la plus à gauche, c.-à-d. 1, l'arbre se réduit à une seule de ses branches principales.

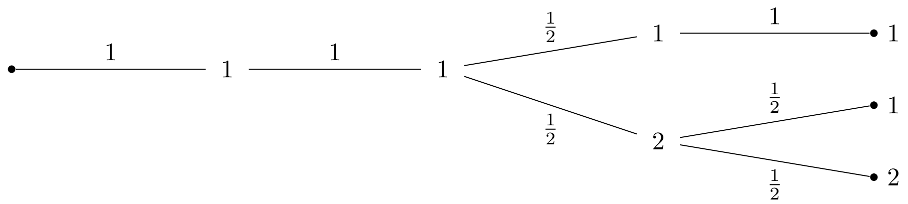

Figure 2 – Arbre de probabilité pour le départ 1.

Et, pour les mêmes collections que précédemment, on trouve les résultats contenus dans le tableau 2.

$P(A \in \varepsilon_1)$	$P(A \in \varepsilon_2)$	$P(A \in \varepsilon_3)$
$\frac{3}{4}$	$\frac{1}{4}$	0
0,75	0,25	0

Tableau 2 – Distribution de probabilité théorique pour le départ 1.

On peut remarquer que, dans les deux cas étudiés, la somme des probabilités d'occurrence associées aux trois collections d'évènements vaut 1, vérifiant ainsi le second axiome de Kolmogorov.

2 Monte Carlo

(a) Simulation simplifiée et estimation de probabilité

Dans le script `Q2a.m`[1], on génère, en premier lieu, un vecteur d'entrées `in`. Dans le cas où la position initiale est aléatoire, il est créé à partir de la fonction `randi` et dans l'autre cas, ses éléments sont fixés à 1 par la fonction `ones`.

Ensuite, la fonction `doSim`, prenant en argument le vecteur `in` et les paramètres du problème, simule une sortie pour chacune des entrées. Ce vecteur de sorties permet à la fonction `getProb` de calculer les fréquences d'occurrences relatives de ces dernières et donc de construire l'estimateur de la distribution de probabilité demandée.

Les résultats obtenus, renseigné dans tableau 3, en exécutant le script sont relativement proches des valeurs théoriques.

Taille $= 10^4$	$P(A \in \varepsilon_1)$	$P(A \in \varepsilon_2)$	$P(A \in \varepsilon_3)$
(a)	0,3394	0,3191	0,3415
(b)	0,7559	0,2441	0

Tableau 3 – Estimations de distributions de probabilité.

Suite à plusieurs essais, on note qu'augmenter la taille du vecteur d'entrées permet d'améliorer la précision de la méthode.

Taille $= 10^5$	$P(A \in \varepsilon_1)$	$P(A \in \varepsilon_2)$	$P(A \in \varepsilon_3)$
(a)	0,3440	0,3143	0,3417
(b)	0,7491	0,2509	0

Tableau 4 – Estimations de distributions de probabilité.

(b) Simulation complète et estimation de l'espérance

Ici, et pour chaque taille du vecteur d'entrées, les sorties générées par la fonction `doSim` sont utilisées par la fonction `getGain` pour déterminer le gain qu'elles génèrent séparément.

Ensuite, la fonction `mean` calcule la moyenne de ces gains pour l'utiliser en tant qu'estimateur de l'espérance.

Après avoir lancé le script `Q2b.m` à trois reprises, on observe des variations significatives dans les résultats qu'il retourne et qui sont fournis dans la table 5.

1. Les scripts, et les fonctions qu'ils utilisent, sont disponibles dans l'annexe.

	Taille	10^1	10^2	10^3	10^4
Test 1		1,200	0,280	0,506	0,522
Test 2	**Espérance** [€]	−2,100	0,870	0,432	0,540
Test 3		1,600	0,400	0,330	0,402

Tableau 5 – Estimations de l'espérance en variant la taille de l'échantillon.

Bien que les valeurs semblent converger vers un nombre avoisinant 0,5 lorsque la taille du vecteur augmente, les variations qu'elles subissent restent trop grande pour qu'elles soient représentatives.

(c) Étude de la variance de l'espérance

Il s'agit, pour cette question, de répéter l'expérience précédente `nbrRepeat` (c.-à-d. 10^3) fois. Ensuite, pour chaque taille du vecteur de sorties, les moyenne et variance des `nbrRepeat` estimations de l'espérance sont déterminées respectivement grâce aux fonctions `mean` et `var`.

Taille	10^1	10^2	10^3	10^4	10^5
Moyenne [€]	0,436	0,465	0,479	0,470	0,472
Variance [€^{2}]	2,656	$2{,}907 \times 10^{-1}$	$2{,}666 \times 10^{-2}$	$2{,}703 \times 10^{-3}$	$2{,}699 \times 10^{-4}$

Tableau 6 – Étude des estimations de l'espérance.

On observe à partir des valeurs calculées par le script `Q2c.m`, présentées dans le tableau 6, que les moyennes respectives des espérances estimées sont plutôt proches les unes des autres. Au contraire, la variance semble diminuer de la même façon que la taille du vecteur augmente.

Cependant, pour la taille 10^4, si on approxime l'erreur moyenne par l'écart type, c.-à-d. par la racine de la variance, on trouve qu'elle est de l'ordre d'un dixième de l'espérance, ce qui n'est pas négligeable.

$$\delta \simeq \sigma = \sqrt{2{,}703 \times 10^{-3}} = 5{,}200 \times 10^{-2}$$

Dès lors, il est parfaitement envisageable de choisir une taille de 10^5, voir même plus, si on souhaite augmenter notre précision et que le temps de calcul ne nous rebute pas.

On peut calculer facilement que l'erreur moyenne subira une diminution d'un facteur $\sqrt{10}$ lorsque on multipliera la taille par 10.

(d) Détermination de la position initiale d'espérance maximale

Cette expérience réutilise le code écrit pour la question 2b aux exceptions près que les entrées ne sont plus aléatoires et que le changement du nombre de simulations est remplacé par le changement de la position fixée.

Les résultats du script `Q2d.m`, donné dans le tableau 7, indiquent que la position initiale dont on peut espérer le gain le plus élevé est la 6ème.

Position	1	2	3	4	5	6	7	8	9
Espérance [€]	−0,109	0,022	0,318	0,572	0,707	0,755	0,731	0,647	0,601

Tableau 7 – Estimations de l'espérance en variant la position initiale.

(e) Expériences de Bernoulli et loi binomiale

Chaque déviation de la bille peut être considérée comme une expérience de Bernoulli. Ainsi, suivre son chemin le long de la planche revient à étudier la répétition de 9 expériences de Bernoulli.

Si la réussite de ces dernières est associée à l'évènement « déviée à droite » dont la probabilité d'occurrence est $\frac{1}{2}$, la loi suivie par la variable aléatoire $\mathcal{X}$ est définie par la loi binomiale $\mathcal{B}\left(9, \frac{1}{2}\right)$.

Dès lors, l'espérance et la variance de $\mathcal{X}$ sont facilement déterminables :

$$E\left\{\mathcal{X}\right\} = 9\frac{1}{2} = \frac{9}{2} \qquad V\left\{\mathcal{X}\right\} = 9\frac{1}{2}\left(1 - \frac{1}{2}\right) = \frac{9}{4}$$

(f) Théorème de Moivre-Laplace et loi de Cauchy

Théorème de Moivre-Laplace

En toute généralité, si $\mathcal{X}_n$ est définie par la loi binomiale $\mathcal{B}\left(n, p\right)$, on sait que :

$$E\left\{\mathcal{X}_n\right\} = np = \mu \qquad V\left\{\mathcal{X}_n\right\} = np\left(1-p\right) = \sigma^2$$

Or, d'après le théorème de Moivre-Laplace,

$$\begin{aligned} & \left(\frac{\mathcal{X}_n - np}{\sqrt{np\left(1-p\right)}}\right) \xrightarrow{\mathcal{L}} \mathcal{N}(0,1) \\ \Rightarrow \qquad & \mathcal{X}_n \xrightarrow{\mathcal{L}} \mathcal{N}\left(np, np\left(1-p\right)\right) \end{aligned}$$

On peut donc approximer la loi binomiale $\mathcal{B}\left(n, p\right)$ par la loi gaussienne (normale) $\mathcal{N}\left(\mu, \sigma^2\right)$ lorsque n augmente. La variable $\mathcal{Y}$ est, quant à elle, approchable par la loi normale $\mathcal{N}\left(0, \sigma^2\right)$.

En pratique, cette approximation est généralement considérée comme acceptable lorsque la variance de $\mathcal{X}_n$ est supérieure ou égale à 10.

$$V\left\{\mathcal{X}_n\right\} = np\left(1-p\right) \geq 10$$

Dans notre cas, p valant $\frac{1}{2}$, la condition est remplie lorsque n est supérieur ou égal à 40.

Loi de Cauchy

Le principe de cette question est le même que celui de la question 2c et, dès lors, leur implémentations est très semblable. On commence par simuler 10^{k} fois la variable $\mathcal{Z}$ comme le rapport de deux instances de la fonction `normrnd` c.-à-d. le rapport de deux variables aléatoires indépendantes suivant une loi normale.

Ensuite, on détermine l'estimateur de l'espérance en calculant la moyenne de ses 10^{k} simulations et on recommence tout ceci 10^3 fois. À partir de ces valeurs, on détermine la moyenne et la variance de l'espérance estimée.

Cette méthode, implémentée dans le script `Q2f.m` pour différentes valeurs de k, donne les résultats du tableau 8.

Taille	10^2	10^3	10^4	10^5
Moyenne [€]	1,281	−0,7325	−0,0202	−2,2852
Variance [€^{2}]	$1{,}020 \times 10^4$	$4{,}740 \times 10^2$	$1{,}403 \times 10^2$	$7{,}117 \times 10^3$

Tableau 8 – Étude des estimations de l'espérance de $\mathcal{Z}$.

On remarque que l'espérance et la variance de $\mathcal{Z}$ ne convergent pas lorsque la taille des vecteurs de sorties augmente.

L'explication de ce phénomène se trouve dans le fait que la variable aléatoire $\mathcal{Z}$ est définie comme le rapport de deux variables aléatoires indépendantes $\mathcal{Y}_1$ et $\mathcal{Y}_2$ suivant des lois gaussiennes $\mathcal{N}(0,2)$.

$$\begin{aligned}\mathcal{Z} &= \frac{\mathcal{Y}_1}{\mathcal{Y}_2} \\ &= \frac{\mathcal{Y}_1}{\sqrt{2}}\frac{\sqrt{2}}{\mathcal{Y}_2}\end{aligned}$$

Cependant, $\frac{\mathcal{Y}_1}{\sqrt{2}}$ suit, ainsi que $\frac{\mathcal{Y}_2}{\sqrt{2}}$, une loi normale $\mathcal{N}(0,1)$. Dès lors, $\mathcal{Z}$ peut être définie par le rapport de deux lois gaussiennes centrées-réduites, c.-à-d. par une loi de Cauchy qui n'admet pas d'espérance et, à fortiori, de variance.

A Scripts

```
%% Parameters

nbrPos = 4;
nbrNail = 3;
nbrOut = nbrPos - mod(nbrNail, 2);
nbrSim = 10^4;

%% Code

tic
in = randi([1, nbrPos], 1, nbrSim); %Genere des entrees aleatoires
probHazard = getProb(doSim(in, nbrPos, nbrNail), nbrOut);

in = ones(1, nbrSim); %Genere des entrees constantes
probOne = getProb(doSim(in, nbrPos, nbrNail), nbrOut);
toc

%% Display

disp(table((1:nbrOut)', probHazard', probOne', 'VariableNames',
    {'Position', 'probHazard', 'probOne'}));

clearvars -except probHazard probOne;
```

Listing 1 – `Q2a.m`

```
%% Parameters

nbrPos = 9;
nbrNail = 10;
gain = [1 -3 5 -5 8 -7 7 -2 1];
nbrTry = 4;

%% Code

nbrSim = zeros(1, nbrTry);
esperance = zeros(1, nbrTry);

tic
for i = 1:nbrTry %Pour chaque taille

    nbrSim(i) = 10^i; %Calcule le nombre de simulations
    in = randi([1, nbrPos], 1, nbrSim(i)); %Genere des entrees
        aleatoires
    esperance(i) = mean(getGain(doSim(in, nbrPos, nbrNail), gain));

end
toc

%% Display

disp(table(nbrSim', esperance', 'VariableNames', {'Simulations',
```

```
    'Esperance'}));

clearvars -except nbrSim esperance;
```

Listing 2 – Q2b.m

```
%% Parameters

nbrPos = 9;
nbrNail = 10;
gain = [1 -3 5 -5 8 -7 7 -2 1];
nbrTry = 4;
nbrRepeat = 10^3;

%% Code

nbrSim = zeros(1, nbrTry);
moyenne = zeros(1, nbrTry);
variance = zeros(1, nbrTry);

esperance = zeros(1,nbrRepeat);

tic
for i = 1:nbrTry %Pour chaque taille

    nbrSim(i) = 10^i; %Calcule le nombre de simulations
    in = randi([1, nbrPos], nbrRepeat, nbrSim(i)); %Genere des
        entrees aleatoires

    parfor j = 1:nbrRepeat %Calcule #nbrRepeat fois l'esperance
        esperance(j) = mean(getGain(doSim(in(j,:), nbrPos, nbrNail),
            gain), 2)';
    end

    moyenne(i) = mean(esperance); %Calcule la moyenne des esperances
    variance(i) = var(esperance, 1); %Calcule la variance des
        esperances

end
toc

%% Display

disp(table(nbrSim', moyenne', variance', 'VariableNames',
    {'Simulations', 'Moyenne', 'Variance'}));

clearvars -except nbrSim moyenne variance;
```

Listing 3 – Q2c.m

```
%% Parameters

nbrPos = 9;
nbrNail = 10;
```

```
gain = [1 -3 5 -5 8 -7 7 -2 1];
nbrSim = 10^7;

%% Code

esperance = zeros(1, nbrPos);

tic
parfor i = 1:nbrPos %Pour chaque position initiale

    in = ones(1, nbrSim) * i; %Genere des entrees constantes
    esperance(i) = mean(getGain(doSim(in, nbrPos, nbrNail), gain));

end
toc

[value, index] = max(esperance); %Cherche le maximum

%% Display

disp(table((1:nbrPos)', esperance', 'VariableNames', {'Position',
    'Esperance'}));

disp(table(index, value, 'VariableNames', {'Optimale', 'Esperance'}));

clearvars -except esperance index value;
```

Listing 4 – `Q2d.m`

```
%% Parameters

mu = 0;
sigma = sqrt(2);
nbrTry = 4;
nbrRepeat = 10^3;

%% Code

nbrSim = zeros(1, nbrTry);
moyenne = zeros(1, nbrTry);
variance = zeros(1, nbrTry);

esperance = zeros(1, nbrRepeat);

tic
for i = 1:nbrTry %Pour chaque taille

    nbrSim(i) = 10^(i + 1); %Calcule le nombre de simulations

    matmu = ones(nbrSim(i), nbrRepeat) * mu; %Matrice esperance
    matsigma = ones(nbrSim(i), nbrRepeat) * sigma; %Matrice ecart-type

    %Genere les valeurs de la v.a. Z
    Z  = normrnd(matmu, matsigma) ./ normrnd(matmu, matsigma);

```

```
    esperance = mean(Z); %Calcule les esperances

    moyenne(i) = mean(esperance); %Calcule la moyenne des esperances
    variance(i) = var(esperance, 1); %Calcule la variance des esperances
end
toc

%% Display

disp(table(nbrSim', moyenne', variance', 'VariableNames', {'Simulations', 'Moyenne', 'Variance'}));

clearvars -except nbrSim moyenne variance;
```

Listing 5 – `Q2f.m`

B Fonctions

Lors de l'écriture des fonctions, nous avons fait le choix de ne pas les rendre robustes, c.-à-d. que nous supposons leurs entrées valides et ne les vérifions pas. Ce choix permet de garder les scripts rapides malgré les nombreuses simulations effectuées.

```
function out = doSim(in, nbrPos, nbrNail)
%% doSim

%   Simule une partie pour chaque position initiale contenue dans la
    matrice #in et renvoie les positions finales dans la matric #out.

%% Parameters

%   #nbrPos est le nombre de positions que la bille peut prendre.
%   #nbrNail est le nombre de bifurcations effectuees par la bille.

%% Code

    out = in;
    totalLength = numel(in);

    add = randi([0 1], totalLength, nbrNail); %Matrice des chemins

    for i = 1:totalLength %Pour chaque element de #in
        for j = 1:nbrNail
            if (out(i) == 1) %Condition a la bordure gauche
                add(i, j) = 1;
            elseif (out(i) == nbrPos) %Condition a la bordure droite
                add(i, j) = 0;
            end
            out(i) = out(i) + add(i, j) - 1 / 2;
        end
    end

    out = floor(out);

end
```

Listing 6 – `doSim.m`

```
function prob = getProb(mat, nbrPos)
%% getProb

%   Calcule et renvoie #prob, le vecteur des frequences d'occurence
    des sorties contenues dans la matrice #mat.

%% Parameters

%   #nbrPos est le nombre de positions que la bille peut prendre.

%% Code

```

```
    totalLength = numel(mat);

    prob = zeros(1, nbrPos);

    for i = 1:totalLength
        prob(mat(i)) = prob(mat(i)) + 1; %Compte les occurences
    end

    prob = prob / totalLength; %Pondere sur la taille de l'echantillon

end
```

Listing 7 – `getProb.m`

```
function matGain = getGain(mat, gain)
%% getGain

%   Calcule et renvoie #matGain, la matrice des gains assosciee a la matrice des sorties #mat.

%% Parameters

%   #gain contient les gains associes respectivement aux sorties possibles.

%% Code

    matGain = mat;
    totalLength = numel(mat);

    for i = 1:totalLength
        matGain(i) = gain(mat(i)); %Associe un gain a chaque sortie
    end

end
```

Listing 8 – `getGain.m`