Université de Liège

Projet 1 - La planche de Galton

Éléments du calcul des probabilités

Bastien Hoffmann (20161283)
Maxime Meurisse (20161278)

2e année de Bachelier Ingénieur Civil
Année académique 2017-2018

1 Méthode de Lehman & Leighton

Dans cette section, on cherche une solution exacte au problème par la construction d'un arbre de probabilités en utilisant la méthode de Lehman & Leighton.

Les variables du problème sont le choix de l'entrée par le joueur et la sortie dans laquelle la boule tombera. L'événement étudié est la probabilité que la boule atteigne une sortie particulière. Toutes les sorties étant intéressantes, on étudiera toutes les feuilles de l'arbre.

1.a Position initiale de la boule aléatoire

Dans ce cas, on suppose que le joueur choisit la position initiale de la boule au hasard et de manière équiprobable.

Pour obtenir la probabilité d'une feuille, on multiplie les probabilités se trouvant sur la branche menant à celle-ci. On additionne ensuite les probabilités des feuilles correspondantes afin d'obtenir la distribution de probabilités des sorties.

En appelant ε_1, ε_2 et ε_3 les événements tels que la boule tombe respectivement dans la sortie 1, 2 et 3, on obtient les résultats présentés dans le tableau 1.

Événement	**Probabilité**	
ε_1	11/32	0,3438
ε_2	10/32	0,3125
ε_3	11/32	0,3438

Tableau 1 – Distribution de probabilités lorsque la position initiale de la boule est aléatoire.

On constate que la probabilité de tomber dans une sortie périphérique est plus grande que celle de tomber dans la sortie centrale. Cela peut se justifier par le fait que les bords de la planche contraignent la boule à prendre une direction particulière.

1.b Position initiale de la boule fixée

Dans ce cas, on suppose que le joueur choisit toujours la rangée la plus à gauche (position 1) comme entrée. On peut d'ores et déjà supposer que la sortie 3 ne pourra être atteinte du au manque de rangées de clous pouvant l'orienter vers cette sortie.

En considérant les mêmes événements que précédemment, on obtient les résultats présentés dans le tableau 2.

Comme attendu, la sortie 3 ne peut être atteinte par aucune boule.

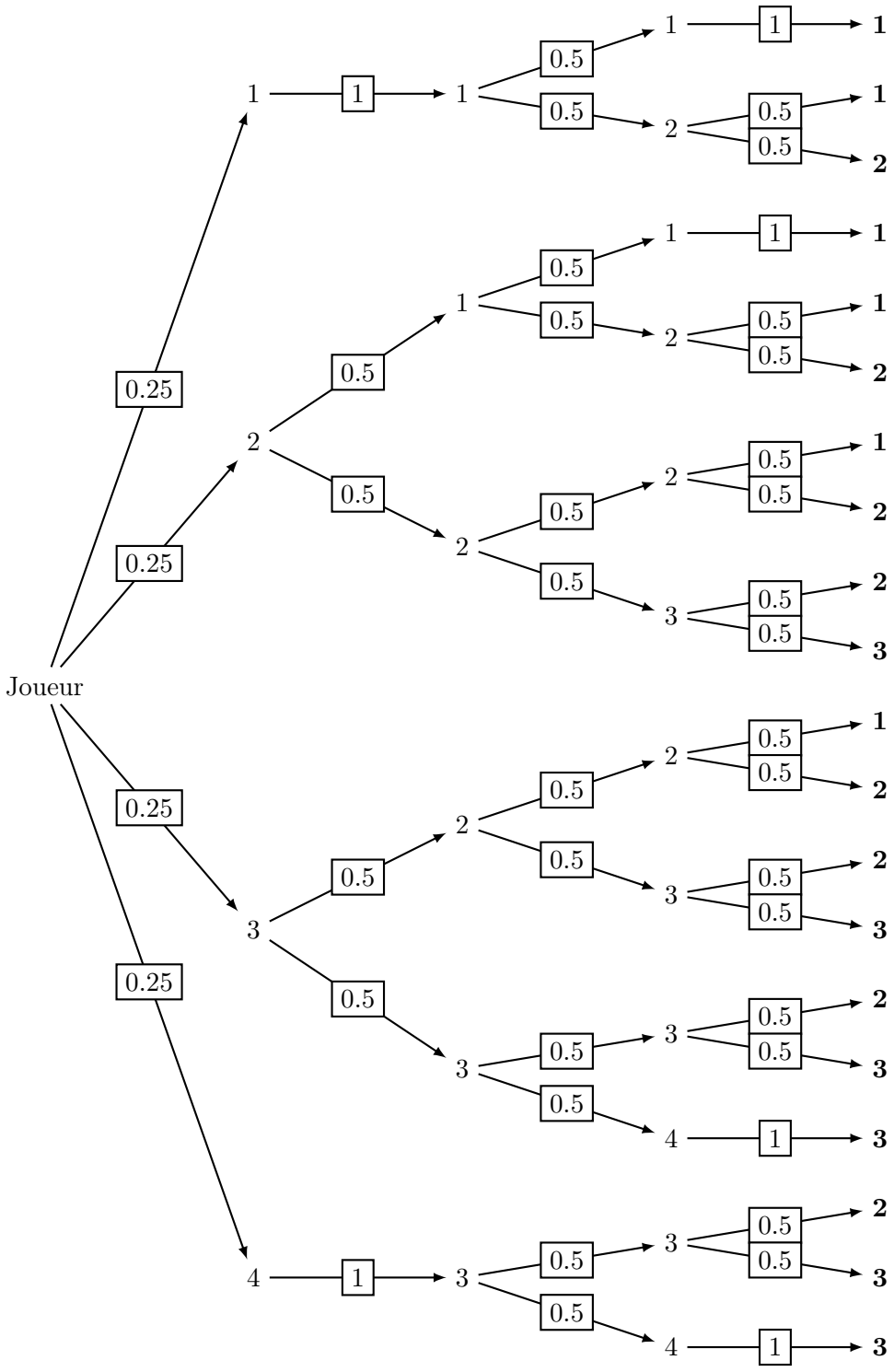

Figure 1 – Arbre de probabilités lorsque la position initiale de la boule est aléatoire.

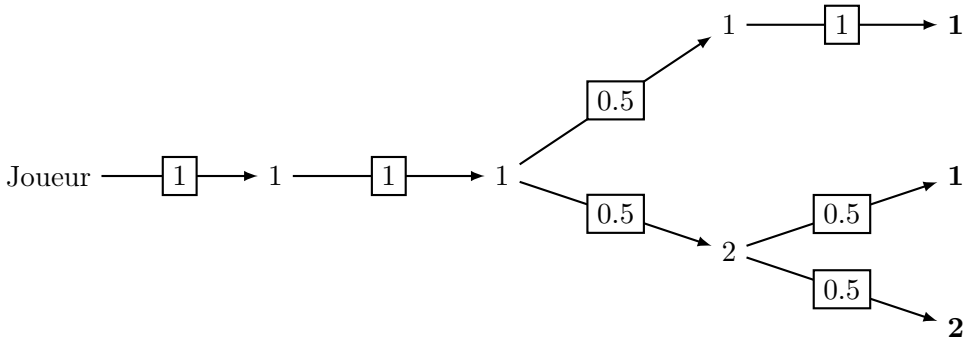

Figure 2 – Arbre de probabilités lorsque la position initiale de la boule est l'entrée la plus à gauche.

Événement	**Probabilité**	
ε_1	3/4	0,75
ε_2	1/4	0,25
ε_3	0	0

Tableau 2 – Distribution de probabilités lorsque la position initiale de la boule est l'entrée la plus à gauche.

Remarque Dans les deux cas étudiés, le second axiome de Kolmogorov est vérifié : la somme des probabilités de chaque événement vaut bien 1 .

2 Méthode de Monte Carlo

2.a Simulation pour une planche de jeu simplifiée

Les simulations sont effectuées par la fonction `GameSimulation` [1]. Celle-ci génère le vecteur des entrées (aléatoires ou non) et appelle la fonction `BallSimulation` avec ce vecteur en paramètre. Cette fonction retourne, quant à elle, pour chaque position d'entrée fournie, une position de sortie.

Les distributions de probabilités sont ensuite calculées par la fonction `GetProb` qui, pour chaque vecteur, divise le nombre de boules tombées dans une sortie x par le nombre total de boules simulées.

Ces fonctions sont appelées, avec les paramètres adéquats, par le script `Q2a.m`.

Les résultats obtenus pour deux parties, simulant chacune 10^4 boules, sont présentés dans le tableau 3.

On constate que ces résultats sont fort proches des résultats théoriques.

1. Tous les scripts et fonctions mentionnés dans ce document se trouvent en annexe.

Événement	**Position aléatoire**	**Position fixée**
ε_1	0,3392	0,7514
ε_2	0,3166	0,2586
ε_3	0,3442	0

Tableau 3 – Estimations des distributions de probabilités.

2.b Simulation pour une planche de jeu complète

À l'aide de la fonction `GetExpValue`, pour chacune des parties simulées, l'estimateur de l'espérance de gain du joueur a été calculé en faisait la moyenne, avec la fonction `mean` de Matlab, du vecteur des gains. Ce vecteur en question est obtenu en associant à chaque sortie son gain correspondant.

En effectuant 5 simulations avec le script `Q2b.m`, on constate, pour un nombre de boules inférieur à 10^3, un écart important entre les résultats correspondants de chaque simulation (tableau 4).

Nombre de boules		10^1	10^2	10^3	10^4
Espérance [€]	Simulation 1	−2,9000	0,1400	0,4170	0,4152
	Simulation 2	−3,1000	0,2900	0,4410	0,4730
	Simulation 3	3,8000	1,1900	0,6220	0,4318
	Simulation 4	0,2000	0,8600	0,4150	0,4267
	Simulation 5	−1,7000	0,2100	0,2960	0,4137

Tableau 4 – Estimations de l'espérance pour différents nombres de boules.

On observe cependant que plus le nombre de boules simulées au sein d'une partie est élevé, plus l'évaluation de l'espérance semble converger vers une valeur avoisinant 0,43 .

2.c Répétition de l'expérience

On répète 1000 fois l'expérience précédente grâce au script `Q2c.m`. Les espérances obtenues sont conservées dans un tableau, dont on a calculé, par la suite, la moyenne et la variance à l'aide des fonctions `mean` et `var` de Matlab (tableau 5).

On constate que les moyennes des estimations de l'espérance de chaque partie simulée avoisinent toutes une valeur proche de 0,47 . La précision de cette valeur s'accroît lorsque le nombre de boules simulées augmente.

La variance, quant à elle, diminue de manière significative lorsque la taille des vecteurs augmente. Il pourrait donc être intéressant de réaliser l'expérience avec 10^5 boules afin de minimiser au plus la dispersion des résultats obtenus.

Nombre de boules	10^1	10^2	10^3	10^4	10^5
Moyenne [€]	0,4200	0,4498	0,4739	0,4728	0,4712
Variance [€^{2}]	2,4984	0,2625	$2{,}7401 \times 10^{-2}$	$2{,}6899 \times 10^{-3}$	$2{,}5830 \times 10^{-4}$

Tableau 5 – Moyenne et variance des 1000 estimations de l'espérance pour différents nombres de boules.

Cependant, avec un tel nombre de boules, le temps de calcul augmente considérablement par rapport à la décimale supplémentaire obtenue. On peut donc se limiter aux résultats de 10^4 boules simulées.

2.d Détermination de l'entrée la plus avantageuse

En effectuant, avec le script `Q2d.m`, une simulation de 10^7 boules [2] pour chaque entrée, on trouve que l'entrée 6 possède l'espérance de gain la plus élevée (tableau 6).

Entrée	**Espérance** [€]
1	−0,1097
2	0,0248
3	0,3167
4	0,5700
5	0,7079
6	0,7562
7	0,7316
8	0,6478
9	0,5971

Tableau 6 – Estimation de l'espérance pour chaque entrée possible du jeu.

Cette valeur étant également supérieure à la valeur obtenue pour une entrée choisie aléatoirement, le joueur a tout intérêt à systématiquement placer sa boule dans l'entrée 6 s'il désire maximiser ses gains.

2.e Loi binomiale

On définit l'expérience comme la déviation d'une boule sur un clou. N'étant associée qu'à deux résultats possibles (la boule est déviée à droite, appelé « succès », et la boule est déviée à gauche, appelé « échec »), elle peut être assimilée à une expérience de Bernoulli.

2. Ce nombre de boules a été choisi pour son bon rapport précision des résultats/temps de calcul.

La variable aléatoire $\mathcal{X}$ est le résultat de $n = 9$ expériences consécutives. Elle suit donc une loi binomiale, dont la probabilité du succès (p) est identique à la probabilité de l'échec (q), à savoir 0,5 .

Ces expériences étant toutes indépendantes, et la probabilité du succès étant constante, l'espérance et la variance de $\mathcal{X}$ sont données respectivement par :

- $E\{\mathcal{X}\} = np = 9\frac{1}{2} = 4{,}5$
- $V\{\mathcal{X}\} = npq = 9\frac{1}{2}\frac{1}{2} = 2{,}25$

2.f Convergence vers une loi gaussienne et loi de Cauchy

Théorème de Moivre-Laplace

On définit une nouvelle variable aléatoire $\mathcal{Y}$ telle que

$$\mathcal{Y} = \mathcal{X} - E\{\mathcal{X}\}$$

En vertu du *théorème de Moivre-Laplace*, un cas particulier du *théorème central-limite*, si le nombre de sorties est suffisamment grand, on peut approcher la distribution de $\mathcal{X}$ (et donc de $\mathcal{Y}$) par une loi gaussienne.

En effet, ce théorème stipule que si $\mathcal{X}_n$ forme une suite de variables aléatoires binomiales (ce qui est le cas ici), alors

$$\frac{\mathcal{X}_n - E\{\mathcal{X}\}}{\sigma\{\mathcal{X}\}} \to \mathcal{N}(0,1)$$

La variable aléatoire $\mathcal{Y}$ n'étant pas divisée par la variance de $\mathcal{X}$, elle est approximée par une loi gaussienne centrée, mais non réduite.

Loi de Cauchy

Soit les variables aléatoires $\mathcal{Y} \sim \mathcal{N}(0,2)$, de moyenne nulle et de variance 2 , représentant le résultat d'une partie, et $\mathcal{Z}$, le rapport entre les résultats de deux parties indépendantes.

En répétant 1000 fois l'expérience demandée grâce au script `Q2f.m`, on obtient les résultats présentés dans le tableau 7.

Nombre de boules	10^2	10^3	10^4	10^5
Moyenne	0,3221	0,7114	0,1897	1,7588
Variance	$0{,}1997 \times 10^3$	$1{,}0072 \times 10^3$	$1{,}2815 \times 10^3$	$6{,}8937 \times 10^3$

Tableau 7 – Moyenne et variance des 1000 estimations de l'espérance, calculées avec une loi normale, pour différents nombres de boules.

On constate une forte divergence des résultats pour chaque nombre de boules simulé. Cela peut s'expliquer par le fait que le quotient de deux variables aléatoires réelles indépendantes suivant des lois normales standards, représenté ici par $\mathcal{Z}$, suit une loi de Cauchy. Or celle-ci n'admet ni espérance ni écart-type, et par conséquent, ni variance, d'où ces résultats qui ne semblent pas converger vers une valeur précise.

En plus

Dans le cadre de ce projet, nous avons développé une application mobile (« Galton Board Simulator », disponible sur le Play Store et l'App Store) afin de simuler de manière visuelle le problème.

Cette application n'a en aucun cas servi d'appui aux résultats présentés dans ce rapport. Elle a simplement été une occasion pour nous d'approfondir le sujet et d'améliorer nos compétences informatiques.

A Code Matlab

A.a Fonctions

```
%% Function that calculates the output position of a ball
%
% ------------------------------------------------ %
% Produced for project 1 (MATH0062-1) by Bastien Hoffmann and Maxime Meurisse
% Second year of Bachelor Civil Engineer - Academic Year 2017-2018
% ------------------------------------------------ %
%
% This function takes inputs from the grid dimensions and the starting
% positions of the balls, and returns the output positions of the balls.
%
% This function works with half-positions to avoid considering the case of
% even and odd rows separately. A half-position corresponds to a position
% which lies between two starting positions. The output positions are rounded
% down to get an integer representing the correct output position.
%
% INPUTS
%   posInput: table containing the input positions
%   nbInputs: number of possible inputs on the grid
%   nbRows: number of rows of nails on the grid
%
% OUTPUT
%   posOutput: table containing the output positions associated with each input position of the posInput table

function posOutput = BallSimulation(posInput, nbInputs, nbRows)

%% Verification of the input parameters
if(nbInputs < 1 || nbRows < 1)
    error('The parameters of the function have invalid values or dimensions.');
end

%% Initialization
posOutput = posInput;

%% Calculation of the output position for each input
tmpSize = size(posInput, 2);

for i = 1:tmpSize
    posRand = randi([0 1], [1, nbRows]); % random movements generated for each row of nail

    for j = 1:nbRows
        if(posOutput(i) == 1) % case of extreme position
            tmpPos = 1/2;
        elseif(posOutput(i) == nbInputs) % case of extreme position
```

```
44              tmpPos = -1/2;
45          else
46              tmpPos = posRand(j) - 1/2;
47          end
48
49          posOutput(i) = posOutput(i) + tmpPos;
50      end
51
52      posOutput(i) = floor(posOutput(i));
53  end
```

Listing 1 – Fonction `BallSimulation.m`.

```
1  %% Function that simulates a game
2  %
3  % ------------------------------------------- %
4  % Produced for project 1 (MATH0062-1) by Bastien Hoffmann and Maxime
       Meurisse
5  % Second year of Bachelor Civil Engineer - Academic Year 2017-2018
6  % ------------------------------------------- %
7  %
8  % This function takes as input the dimensions of the grid, the initial
9  % position and the number of simulations to be performed by vectors.
10 % It returns a cell containing the different simulation vectors.
11 %
12 % This function generates an array of inputs, of size equal to the
       size of
13 % the vector, and calls the function "BallSimulation" with this table
       of
14 % inputs in parameters. The operation is repeated for each vector.
15 %
16 % INPUTS
17 %   nbInputs: number of possible inputs on the grid
18 %   nbRows: number of rows of nails on the grid
19 %   posInit: choice of the starting position (0 = random position)
20 %   vectors: size of each vector that will contain the simulations
21 %
22 % OUTPUT
23 %   outputs: cell containing the different simulation vectors
24
25 function outputs = GameSimulation(nbInputs, nbRows, posInit, vectors)
26
27 %% Definition of variables
28 nbVectors = size(vectors, 1);
29
30 outputs = cell(1, nbVectors);
31
32 %% Verification of the input parameters
33 if(nbInputs < 1 || nbRows < 1 || posInit < 0 || posInit > nbInputs ||
       nbVectors < 1)
34     error('The parameters of the function have invalid values or
           dimensions.');
35 end
36
37 %% Calculation of the output positions for each vector
38 for i = 1:nbVectors
```

```
39      tmpSize = vectors(i, 1);
40      tmpPos = zeros(1, tmpSize);
41
42      if(posInit == 0)
43          tmpPos = randi([1 nbInputs], [1, tmpSize]);
44      else
45          tmpPos(1, :) = posInit;
46      end
47
48      outputs{i} = BallSimulation(tmpPos, nbInputs, nbRows);
49  end
50
51  end
```

Listing 2 – Fonction `GameSimulation.m`.

```
1  %% Function that calculates the expected value of a simulated game
2  %
3  % ------------------------------------------------ %
4  % Produced for project 1 (MATH0062-1) by Bastien Hoffmann and Maxime
       Meurisse
5  % Second year of Bachelor Civil Engineer - Academic Year 2017-2018
6  % ------------------------------------------------ %
7  %
8  % This function takes as input the vector of the outputs of a game and
9  % returns the expected value of this game.
10 %
11 % To obtain the expected value, this function creates a vector by
       associating
12 % with each output its gain, and then calculates the average of this
       vector.
13 %
14 % INPUT
15 %   outputs: cell containing the different simulation vectors
16 %
17 % OUTPUT
18 %   expValue: table containing the expected value of each simulation
       vector
19 %   outputsGains: cell containing the vectors built by associating
       with each output its gain
20
21 function [expValue, outputsGains] = GetExpValue(outputs)
22
23 %% Definition of variables
24 nbOutputs = size(outputs, 2);
25
26 outputsGains = cell(1, nbOutputs); % table containing the gain of
       each output
27 expValue = zeros(nbOutputs, 1);
28
29 gains = [1, -3, 5, -5, 8, -7, 7, -2, 1]; % table containing the
       winnings and losses of the game
30
31 %% Verification of the input parameters
32 if(nbOutputs < 1)
33     error('The parameters of the function have invalid values or
```

```
        dimensions.');
end

%% Calculation of the expected value of a simulated game
for i = 1:nbOutputs
    tmpSize = size(outputs{i}, 2);

    % Creation of the vector containing the gains of each output
    for j = 1:tmpSize
        outputsGains{i}(1, j) = gains(1, outputs{i}(1, j));
    end

    expValue(i, 1) = mean(outputsGains{i});
end

end
```

Listing 3 – Fonction `GetExpValue.m`.

```
%% Function that calculates the probability distribution of a
   simulated game
%
% ------------------------------------------------ %
% Produced for project 1 (MATH0062-1) by Bastien Hoffmann and Maxime
   Meurisse
% Second year of Bachelor Civil Engineer - Academic Year 2017-2018
% ------------------------------------------------ %
%
% This function takes the dimensions of the grid and the simulation
   vectors
% as inputs and returns the probability distribution of each output
   for each
% of the vectors.
%
% To obtain the probability that the ball falls into an output x, this
% function counts the number of outputs x in the vector and divides
   it by
% the number of elements in that vector.
%
% INPUTS
%   nbInputs: number of possible inputs on the grid
%   nbRows: number of rows of nails on the grid
%   outputs: cell containing the different simulation vectors
%
% OUTPUT
%   prob: cell containing the probability distribution of each
   simulation vector

function prob = GetProb(nbInputs, nbRows, outputs)

%% Definition of variables
nbVectors = size(outputs, 2);

prob = cell(1, nbVectors);

%% Verification of the input parameters
```

```
32 if(nbInputs < 1 || nbRows < 1 || nbVectors < 1)
33     error('The parameters of the function have invalid values or
           dimensions.');
34 end
35
36 %% Initialization
37 if(mod(nbRows, 2) == 0)
38     nbOutputs = nbInputs;
39 else
40     nbOutputs = nbInputs - 1;
41 end
42
43 %% Calculation of the distribution of probability of the outputs
44 for i = 1:nbVectors
45     prob{i} = zeros(nbOutputs, 1);
46
47     for j = 1:nbOutputs
48         prob{i}(j, 1) = (length(find(outputs{i}(1, :) == j))) /
               size(outputs{i}, 2);
49     end
50 end
51
52 end
```

Listing 4 – Fonction `GetProb.m`.

A.b Scripts

```
1  %% Script that simulates a game on a 4x3 grid and calculates the
       probability distribution
2  %
3  % ------------------------------------------------- %
4  % Produced for project 1 (MATH0062-1) by Bastien Hoffmann and Maxime
       Meurisse
5  % Second year of Bachelor Civil Engineer - Academic Year 2017-2018
6  % ------------------------------------------------- %
7
8  %% Clear workspace
9  clear
10
11 %% Definition of variables
12 nbInputs = 4; % number of inputs to the game
13 nbRows = 3; % number of rows of nails
14
15 posRandom = 0; % initial position of the ball (0 = random position)
16 posFixed = 1; % fixed initial position of the ball
17
18 vectors = 1e4; % size of each vector that will contain the
       simulations (example: 1e4 or [1e1; 1e2; 1e5])
19
20 %% Simulation of the game
21 outputsRandom = GameSimulation(nbInputs, nbRows, posRandom, vectors);
22 outputsFixed = GameSimulation(nbInputs, nbRows, posFixed, vectors);
23
```

```
24 %% Calculation of the probability distribution
25 probRandom = GetProb(nbInputs, nbRows, outputsRandom);
26 probFixed = GetProb(nbInputs, nbRows, outputsFixed);
27
28 %% Showing results
29 fprintf('Type "outputsRandom" to get the output positions for random
       initial positions.\n');
30 fprintf('Type "outputsFixed" to get the output positions for fixed
       initial positions.\n');
31 fprintf('Type "probRandom" to get the probability distribution for
       random initial positions.\n');
32 fprintf('Type "probFixed" to get the probability distribution for
       fixed initial positions.\n\n');
33
34 fprintf('Summary tables:\n');
35 show = table(probRandom{1}, probFixed{1});
36 show.Properties.VariableNames = {'probRandom', 'probFixed'};
37 disp(show);
38
39 %% Deleting unnecessary variables
40 clearvars vectors posFixed posRandom
```

Listing 5 – Script `Q2a.m`.

```
1  %% Script that simulates games on a 9x10 grid and calculates the
       expected values
2  %
3  % ------------------------------------------------ %
4  % Produced for project 1 (MATH0062-1) by Bastien Hoffmann and Maxime
       Meurisse
5  % Second year of Bachelor Civil Engineer - Academic Year 2017-2018
6  % ------------------------------------------------ %
7
8  %% Clear workspace
9  clear
10
11 %% Definition of variables
12 nbInputs = 9; % number of inputs to the game
13 nbRows = 10; % number of rows of nails
14
15 posInit = 0; % initial position of the ball (0 = random position)
16
17 vectors = [1e1; 1e2; 1e3; 1e4]; % size of each vector that will
       contain the simulations (example: 1e4 or [1e1; 1e2; 1e5])
18
19 %% Simulation of the game
20 outputs = GameSimulation(nbInputs, nbRows, posInit, vectors);
21
22 %% Calculation of the expected value
23 [expValue, outputsGains] = GetExpValue(outputs);
24
25 %% Showing results
26 fprintf('Type "outputs" to get the output positions for each
       vector.\n');
27 fprintf('Type "outputsGains" to get the gains for each vector.\n');
28 fprintf('Type "expValue" to get the expected value for each
```

```
    vector.\n\n');
29
30 fprintf('Summary tables:\n');
31 show = table(vectors, expValue);
32 show.Properties.VariableNames = {'vectors', 'expValue'};
33 disp(show);
34
35 %% Deleting unnecessary variables
36 clearvars posInit vectors
```

Listing 6 – Script `Q2b.m`.

```
 1 %% Script that simulates 1000 games on a 9x10 grid and calculates the
       average and variance of expected values
 2 %
 3 % ------------------------------------------- %
 4 % Produced for project 1 (MATH0062-1) by Bastien Hoffmann and Maxime
       Meurisse
 5 % Second year of Bachelor Civil Engineer - Academic Year 2017-2018
 6 % ------------------------------------------- %
 7
 8 %% Clear workspace
 9 clear
10
11 %% Definition of variables
12 nbInputs = 9; % number of inputs to the game
13 nbRows = 10; % number of rows of nails
14
15 posInit = 0; % initial position of the ball (0 = random position)
16
17 vectors = [1e1; 1e2; 1e3; 1e4; 1e5]; % size of each vector that will
       contain the simulations (example: 1e4 or [1e1; 1e2; 1e5])
18 nbVectors = size(vectors, 1);
19
20 nbExp = 1000; % number of times the experience is simulated
21
22 allExpValue = zeros(nbVectors, nbExp); % table that contains the
       expected values of the 1000 experiments
23 avgExpValue = zeros(nbVectors, 1); % table that contains the average
       of the expected values
24 varExpValue = zeros(nbVectors, 1); % table that contains the average
       of the variances
25
26 %% Repetition of the experience
27 for i = 1:nbExp
28     % Simulation of the game
29     outputs = GameSimulation(nbInputs, nbRows, posInit, vectors);
30
31     % Calculation of the expected value
32     [expValue, outputsGains] = GetExpValue(outputs);
33
34     % Saving calculated expected value
35     allExpValue(:, i) = expValue(:, 1);
36 end
37
38 %% Calculation of the average and variance of expected values
```

```
39 for i = 1:nbVectors
40     avgExpValue(i) = mean(allExpValue(i, :));
41     varExpValue(i) = var(allExpValue(i, :), 1);
42 end
43
44 %% Showing results
45 fprintf('Type "avgExpValue" to get the average of expected values for
       each vector.\n');
46 fprintf('Type "varExpValue" to get the variance of expected values
       for each vector.\n\n');
47
48 fprintf('Summary tables:\n');
49 show = table(vectors, avgExpValue, varExpValue);
50 show.Properties.VariableNames = {'vectors', 'avgExpValue',
       'varExpValue'};
51 disp(show);
52
53 %% Deleting unnecessary variables
54 clearvars allExpValue expValue i nbVectors outputs outputsGains
       posInit prob vectors
```

Listing 7 – Script `Q2c.m`.

```
1  %% Script that simulates a game on a 9x10 grid for each input and
       calculates the expected value
2  %
3  % ------------------------------------------------- %
4  % Produced for project 1 (MATH0062-1) by Bastien Hoffmann and Maxime
       Meurisse
5  % Second year of Bachelor Civil Engineer - Academic Year 2017-2018
6  % ------------------------------------------------- %
7
8  %% Clear workspace
9  clear
10
11 %% Definition of variables
12 nbInputs = 9; % number of inputs to the game
13 nbRows = 10; % number of rows of nails
14
15 vectors = 1e7; % size of each vector that will contain the
       simulations (example: 1e4 or [1e1; 1e2; 1e5])
16 nbVectors = size(vectors, 1);
17
18 expValueEachInput = zeros(nbInputs, 1); % table that contains the
       expected value of each input
19
20 %% Calculation of the expected value for each input
21 for j = 1:nbInputs
22     % Simulation of the game
23     outputs = GameSimulation(nbInputs, nbRows, j, vectors);
24
25     % Calculation of the expected value
26     [expValue, outputsGains] = GetExpValue(outputs);
27
28     % Saving calculated expected value for an input
29     expValueEachInput(j, :) = expValue(:, 1);
```

```
30 end
31
32 %% Showing results
33 fprintf('Type "expValueEachInput" to get the expected values for each
       input.\n\n');
34
35 fprintf('Summary tables:\n');
36 show = table([1; 2; 3; 4; 5; 6; 7; 8; 9], expValueEachInput);
37 show.Properties.VariableNames = {'inputs', 'expValueEachInput'};
38 disp(show);
39
40 %% Deleting unnecessary variables
41 clearvars expValue j nbVectors outputs outputsGains prob vectors
```

Listing 8 – Script `Q2d.m`.

```
1  %% Script that simulates 1000 games on a 9x10 grid with the normal
       law and calculates the average and variance of expected values
2  %
3  % ------------------------------------------------- %
4  % Produced for project 1 (MATH0062-1) by Bastien Hoffmann and Maxime
       Meurisse
5  % Second year of Bachelor Civil Engineer - Academic Year 2017-2018
6  % ------------------------------------------------- %
7
8  %% Clear workspace
9  clear
10
11 %% Definition of variables
12 mu = 0; % expected value of the random variable Y
13 sigma = sqrt(2); % standard deviation of the random variable Y
14
15 vectors = [1e2; 1e3; 1e4; 1e5]; % size of each vector that will
       contain the simulations (example: 1e4 or [1e1; 1e2; 1e5])
16 nbVectors = size(vectors, 1);
17
18 nbExp = 1000; % number of times the experience is simulated
19
20 Z = cell(1, nbVectors);
21
22 allExpValue = zeros(nbVectors, nbExp); % table that contains the
       mathematical expectations of the 1000 experiments
23 avgExpValue = zeros(nbVectors, 1); % table that contains the average
       of the expected values
24 varExpValue = zeros(nbVectors, 1); % table that contains the average
       of the variances
25
26 %% Repetition of the experience
27 for i = 1:nbExp
28     for j = 1:nbVectors
29         tmpSize = vectors(j, 1);
30
31         Z{j} = (normrnd(mu, sigma, tmpSize, 1))./(normrnd(mu, sigma,
               tmpSize, 1));
32
33         allExpValue(j, i) = mean(Z{j});
```

```
34      end
35  end
36
37  %% Calculation of the average and variance of expected values
38  for i = 1:nbVectors
39      avgExpValue(i) = mean(allExpValue(i, :));
40      varExpValue(i) = var(allExpValue(i, :), 1);
41  end
42
43  %% Showing results
44  fprintf('Type "avgExpValue" to get the average of expected values for
        each vector.\n');
45  fprintf('Type "varExpValue" to get the variance of expected values
        for each vector.\n\n');
46
47  fprintf('Summary tables:\n');
48  show = table(vectors, avgExpValue, varExpValue);
49  show.Properties.VariableNames = {'vectors', 'avgExpValue',
        'varExpValue'};
50  disp(show);
51
52  %% Deleting unnecessary variables
53  clearvars allExpValue i j Z nbVectors vectors
```

Listing 9 – Script `Q2f.m`.