

Travaux Pratiques - Probabilités

Dylan TROLES & Hugo POULIQUEN

4 mai 2016

Chapitre 1

Génération de nombres pseudo-aléatoires

1.1 Approche fréquentielle des probabilités

1.1.1 Le lancer de 5 dés

Une épreuve consiste à lancer 5 dés et à considérer la somme amenée à chaque lancer. Construire un programme qui donne :

1. Les valeurs de la somme, le nombre de sorties possibles pour chaque valeur, et la fréquence de chaque sortie.
2. Un tableau et un graphique donnant les fréquences de chaque sortie.

Pour cet exercice, nous avons fait un script python. Pour pouvoir lancer ce script, il est nécessaire d'installer la librairie matplotlib.

Listing 1.1 – Lancer_de_5_des.py

```
1  #!/usr/bin/python3
2  from random import randrange
3  import matplotlib.pyplot as plt
4
5  # Usage : La fonction roll_dice permet de lancer un nombre de des
6  # Param : nbr est le nombre de des a lancer
7  # Retour : La fonction retourne un tableau avec la valeur des des lances
8  def roll_dice(nbr):
9      throw_value = []
10     for i in range(nbr):
11         throw_value.append(randrange(1, 7))
12     return throw_value
13
14 # Usage : La fonction dice_sum effectue la somme du resultat de chaque des
15 # Param : - values
```

CHAPITRE 1. GÉNÉRATION DE NOMBRES PSEUDO-ALÉATOIRES 2

```
16 # - nbr est le nombre de des lance
17 # Retour : La fonction retourne la somme total des des
18 def dice_sum(nbr, values):
19     score = 0
20     for i in range(nbr):
21         score = score + values[i]
22     return str(score)
23
24 # Usage : La fonction possibilities calcul le nombre de possibilites pour une
25 #         somme de des donnee apparaisse
26 # Param : - la somme a traiter
27 # Retour : La fonction retourne le nombre de possibilites de combinaison de des
28 # pour cette somme
29 def possibilities(sum_values):
30     possibility = 0
31     dices = [1, 2, 3, 4, 5, 6]
32     for i in dices:
33         for j in dices:
34             for k in dices:
35                 for l in dices:
36                     for m in dices:
37                         res = i + j + k + l + m
38                         if res == sum_values:
39                             possibility += 1
40     return possibility
41
42 # Usage : La fonction possible_sum calcul le nombre de somme possible
43 # Param : - Le nombre de des a traiter
44 # Retour : La fonction retourne le nombre de somme possible lors du lancement
45 #         de ce nombre de des (ici 5 des)
46 def possible_sum(dices_nbr):
47     return str((6*dices_nbr) - 5)
48
49 # Usage : La fonction possible_combinations calcul le nombre decombinaison pour
50 #         chaque somme possible
51 # Param : - Le nombre de des a traiter
52 #         - Les valeurs des des lances
53 # Retour : La fonction retourne le nombre de combinaisons possible
54 def possible_combinations(dices_nbr):
55     values = []
56     outputs = []
57     j = dices_nbr
58     for i in range(dices_nbr, (6*dices_nbr) + 1):
59         values.append(possibilities(i))
60
61     for i in values:
```

```

62         print(str(j) + ' : ' + str(i))
63         j += 1
64         outputs.append(j)
65     return outputs, values
66
67 # Usage : La fonction frequencys calcul la frequence de chaque sortie
68 # Param : - Le nombre de des a traiter
69 #         - Les valeurs des des lances
70 # Retour : La fonction retourne les frequences de chaque sortie
71 def frequencys(dices_nbr, values):
72     freqs = []
73     k = dices_nbr
74     for i in range(len(values)):
75         frequencesres = 100*(values[i] / (6**dices_nbr))
76         freqs.append(frequencesres)
77         print(str(k) + ' : ' + str(frequencesres))
78         k += 1
79     return freqs
80
81 dices_nbr = 5 # Nombre de des
82
83 print('\nTraitement_pour : ' + str(dices_nbr) + '_des_comportant_6_faces_de_1_a_')
84 print('Nombre_de_sommes_possibles : ' + possible_sum(dices_nbr))
85 print('\n->_Lancement_des_des...')
86 print('Somme_des_des_lances : ' + dice_sum(dices_nbr, roll_dice(dices_nbr)))
87 print('\nTableau_du_nombre_de_combinaisons_possibles_pour_chaque_somme :')
88 outputs, values = possible_combinations(dices_nbr)
89 print('\nTableau_des_frequences_de_chaque_somme :')
90 freqs = frequencys(dices_nbr, values)
91
92 # CREATION DU GRAPHIQUE
93 plt.title("Frequence_d'apparition")
94 plt.plot(outputs, freqs)
95 plt.xlabel('Sorties_possibles')
96 plt.ylabel('Frequence_(%)')
97 plt.show()

```

Voici le résultat généré par le script :

```

$ python3 lancer_de_5_des.py

Traitement pour : 5 des comportant 6 faces de 1 a 6
Nombre de sommes possibles : 25

-> Lancement des des...
Somme des des lances : 21

```

Tableau du nombre de combinaisons possibles pour chaque somme :

5	:	1
6	:	5
7	:	15
8	:	35
9	:	70
10	:	126
11	:	205
12	:	305
13	:	420
14	:	540
15	:	651
16	:	735
17	:	780
18	:	780
19	:	735
20	:	651
21	:	540
22	:	420
23	:	305
24	:	205
25	:	126
26	:	70
27	:	35
28	:	15
29	:	5
30	:	1

Tableau des frequences de chaque somme :

5	:	0.01286008230452675
6	:	0.06430041152263374
7	:	0.19290123456790123
8	:	0.45010288065843623
9	:	0.9002057613168725
10	:	1.6203703703703702
11	:	2.6363168724279835
12	:	3.9223251028806585
13	:	5.401234567901234
14	:	6.944444444444445
15	:	8.371913580246913
16	:	9.452160493827162
17	:	10.030864197530864
18	:	10.030864197530864
19	:	9.452160493827162
20	:	8.371913580246913

21	:	6.944444444444445
22	:	5.401234567901234
23	:	3.9223251028806585
24	:	2.6363168724279835
25	:	1.6203703703703702
26	:	0.9002057613168725
27	:	0.45010288065843623
28	:	0.19290123456790123
29	:	0.06430041152263374
30	:	0.01286008230452675

Voici le graphe généré :

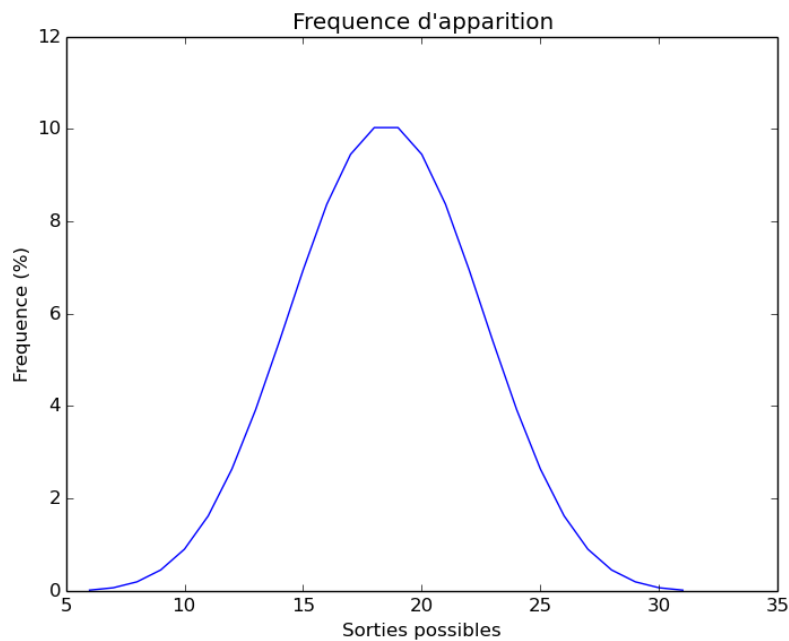


FIGURE 1.1 – Sorties possibles en lançant 5 dés différents en 1 fois

1.2 Nombres pseudo aléatoires

Exercice 1 :

1. Votre machine a une fonction RAND qui vous fournit de manière aléatoire un nombre de $[0,1[$. Donner un algorithme utilisant la fonction RAND fournissant un nombre entier entre 1 et N.

2. Pouvez-vous être satisfait de la fonction fournie par le constructeur ?

Listing 1.2 – Nombre_pseudoaleatoires_ex01.py

```

1 import random
2 import matplotlib.pyplot as plt
3
4 N = 10
5 nbr_loop = 10000
6 results = {}
7 sorties = []
8 for i in range(nbr_loop):
9     result = random.randrange(0, N + 1)
10    sorties.append(result)  # + 1 pour etre entre 1 et 1000 inclus
11    # if result in results:
12    #     counter = results[result]
13    #     counter += 1
14    #     results[result] = counter
15    # else:
16    #     results[result] = 1
17
18 print("\nGraphique_des_resultats_en_fonction_de_leur_nombre_d'apparition")
19
20
21 # nbr_apparition = []
22 # for item in results:
23 #     sorties.append(item)
24 #     nbr_apparition.append(results[item])
25
26 plt.title("Sortie_en_fonction_de_leur_nombre_d'apparition")
27 plt.hist(sorties, N)
28 plt.axis([0, N, 0, 0.20*nbr_loop])
29 plt.xlabel('Sorties')
30 plt.ylabel("Nombre_d'apparitions")
31 plt.show()

```

Réponse : Au vu du graphe, l'aléatoire de la fonction `random.randrange` donnée par le constructeur n'est pas satisfaisant. Chaque barre devrait avoir une probabilité de $\frac{1}{N}$ car nous sommes dans l'ensemble $\{1, \dots, N\}$. Dans notre exemple, $N = 10$. Pour chaque barre, $\frac{1}{N}$ devrait donc être égale à 0,10 .

Prenons la première barre, on remarque que la valeur 1 est sortie 918 fois. Calculons sa probabilité : $\frac{918}{10000} = 0.0918$

Prenons la dernière barre, on remarque que la valeur 10 est sortie 1736 fois. Calculons sa probabilité : $\frac{1736}{10000} = 0.1736$

Les probabilités d'apparitions de chaque valeur ne respecte donc pas 0.10. En conclusion, la fonction `random.randrange` n'est pas aléatoire et pour notre exemple, la valeur 10 apparait le plus souvent.

Voici le graphe généré par le script précédent :

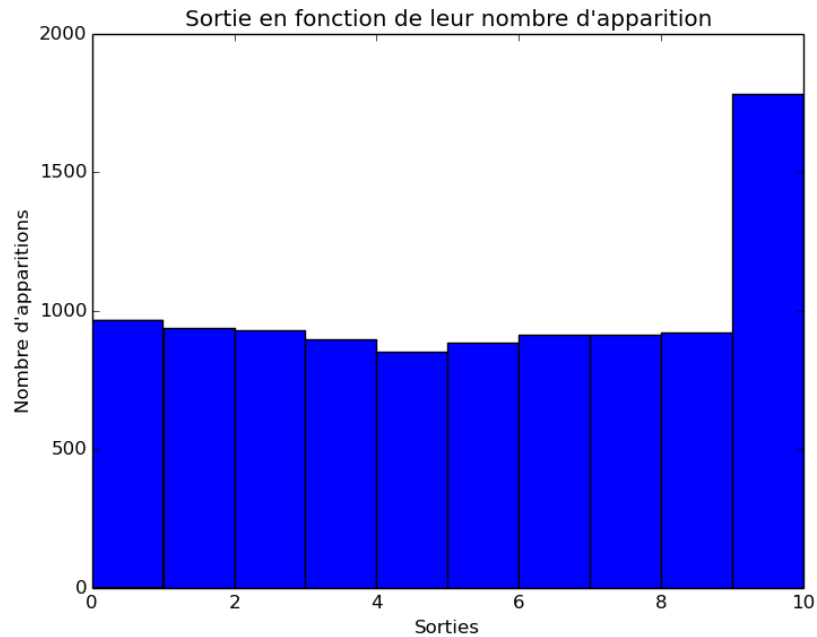


FIGURE 1.2 – Sortie en fonction de leur nombre d'apparition

Exercice 2 : Voici un programme écrit dans le langage algorithmique TestAlgo. Ce petit programme peut être amélioré.

```

algo Nombres aleatoires
var
  entier i,z;
principal
debut
  i:=1
  tantque (i<=100)
    z:=arrondi(3*aleatoire());
    afficher(z)
    i:=i+1;
  fintantque
fin

```


Que réalise ce programme ? Qu'en pensez-vous ?

Réponse :

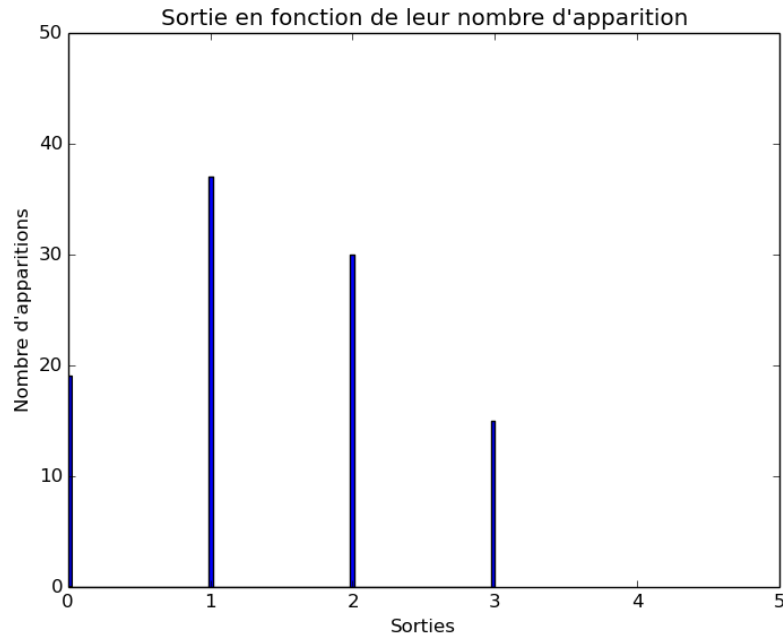


FIGURE 1.3 – Sortie en fonction de leur nombre d'apparition

Ce programme donne un nombre entier entre 0 et 3.

Au vu du graphe nous pensons que cet algorithme se rapproche d'un algorithme aléatoire mais n'en est pas un. Pour le prouver calculons la probabilité théorique de chaque valeur :

$$\frac{1}{N+1} = 0.25 \text{ avec } N = 3$$

Voici les fréquences observées à partir du graphe dans l'expérimentation 0 :

0,15
 1 : 0,32
 2 : 0,33
 3 : 0,20

Cette fois, les probabilités obtenues se rapprochent un peu plus de la probabilité théorique cependant on ne peut pas encore parler d'aléatoire car les différences entre chaque probabilités d'apparition est bien trop différente.

1.3 Génération de nombres aléatoires

1.3.1 Générateurs congruentiels linéaires

Exercice 3 : Donner la suite des nombres aléatoires et en rechercher les périodes pour :

1. $b = 3, X_0 = 7, a = 5, N = 16$
2. $b = 6, X_0 = 7, a = 5, N = 16$
3. $b = 0, X_0 = 1, a = 2, N = 17$
4. $b = 6, X_0 = 1, a = 2, N = 17$
5. $b = 0, X_0 = 3, a = 13, N = 2^7$
6. $b = 0, X_0 = 4567, a = 9749, N = 2^{17}$

Exercice 4 :

1. Écrire une fonction Scilab qui pour argument de sortie une suite $(U_n)_n \in \mathbf{N}$ de réels compris entre 0 et 1, générée par un générateur congruentiel linéaires, et pour argument d'entrée X_0 , a, b et N, les paramètres usuels d'un tel générateur.

Réponse :

Listing 1.3 – Exercice4.sce

```

1  function [Liste]=generateur_un(b,x,a,n)
2      Liste = list(); //Creation de liste
3      X = x;
4      U = X/n; //Calcul de U0
5      Liste($+1) = U; //U0 tete de liste
6      for i=1:(n-1)
7          y=a*X+b; //Calcul Xn suivant l'expression
8          X=modulo(y,n); //Enonce
9          U=X/n; //Calcul Un
10         Liste($+1) = U; //Ajout Un dans liste
11         i=i+1;
12     end
13 endfunction

```

2. En utilisant la fonction écrite en Scilab, étudiez les générateurs qui suivent : on regardera, en particulier, et si c'est possible, la période.
 - Pour $a = 25$, $b = 16$ et $N = 256 = 2^8$ et pour X_0 successifs : $X_0 = 12$; $X_0 = 11$; $X_0 = 0$
 - **Turbo-Pascal** $a = 129$, $b = 907633385$ et $N = 2^{32}$
 - **Unix** $a = 1103515245$, $b = 12345$ et $N = 2^{32}$
 - **Matlab** $a = 19807$, $b = 0$ et $N = 2^{31}-1$

1.4 Autres méthodes

1.4.1 Méthode de VON NEUMANN

Comment fonctionne cette méthode ?

- On se donne un nombre A de N chiffres
- On l'élève au carré
- On choisit comme nombre suivant le nombre de N chiffres formé par la tranche du milieu du carré obtenu
- On divise ensuite par 10^N

1. Montrer que l'on obtient bien une suite de nombres compris entre 0 et 1 (1 exclu)

Réponse : A est un nombre de N chiffres.

$$10^{N-1} \leq A < 10^N$$

On élève au carré : $10^{2N-2} \leq A^2 < 10^{2N}$

B est le nouveau nombre formé par la tranche du milieu du carré obtenu :

$$10^{N-1} \leq B < 10^N$$

On divise par 10^N : $0,1 \leq \frac{B}{10^N} < 1$

2. Donner un algorithme fournissant ces nombres

Réponse :

Listing 1.4 – VonNeumann.py

```

1  #!/usr/bin/python3
2  import matplotlib.pyplot as plt
3  import math
4
5  def VonNeumann(N, A, display):
6      if not N:
7          # Calcul de la longueur du nombre
8          N = len(str(A))
9          square = A * A
10         # Calcul de la longueur du carre du nombre
11         square_len = len(str(square))
12         part = square_len * 0.25
13         square = str(square)
14         # On sauvegarde la tranche du milieu du carre
15         middle_slice = square[int(part):int(part+N)]
16         # On divise cette tranche par 10^N
17         res = float(middle_slice) / float(10**N)
18
19         if(display == 1):
20             print( "----\nA:_:" + str(A)+ "\nN:_:" + \
21                 str(N) + "\nCarre_de_A:_:" + \
22                 str(square) + "\nTranche:_:" + \
23                 str(middle_slice) + "Resultat:_:" + \
24                 str(res)
25             )
26         return res
27
28  VonNeumann(4, 5678, 1)
29  VonNeumann(6, 123456, 1)
30
31  outputs = []
32  loop_nbrs = 100
33  N = 10
34  for i in range(loop_nbrs):
35       outputs.append(math.trunc(VonNeumann(None, i, 0)*N+1))
36
37  plt.title("Sortie_en_fonction_de_leur_nombre_d'apparition")
38  plt.hist(outputs, loop_nbrs)
39  plt.axis([0, N , 0, 0.2*loop_nbrs])

```

```

40 plt.xlabel('Sorties')
41 plt.ylabel("Nombre_d'apparitions")
42 plt.show()

```

3. Faire une étude pour $N = 4$, et $A = 5678$

Réponse :

```

A=5678
N=4
Carre de A=32239684
Tranche=2396
Resultat=0.2396

```

4. Faire une étude pour $N = 6$, et A de votre choix

Réponse :

```

A=123456
N=6
Carre de A=15241383936
Tranche=241383
Resultat=0.241383

```

5. Que penser de cette méthode?

Réponse :

Pour 100 lancers, on obtient les résultats de la figure suivante. Ces résultats montrent que l'aléatoire de VonNeumann n'uniformise pas les numéros de sortie. Si l'on compare avec la méthode du constructeur, VonNeumann est nettement plus aléatoire. On peut dire ceci car la différence du nombre d'apparition minimale et maximale n'est pas aussi importante que pour la méthode du constructeur.

Cette méthode est plus aléatoire que la méthode constructeur, mais n'est pas totalement aléatoire non plus car le nombre d'apparitions varie en fonction des sorties.

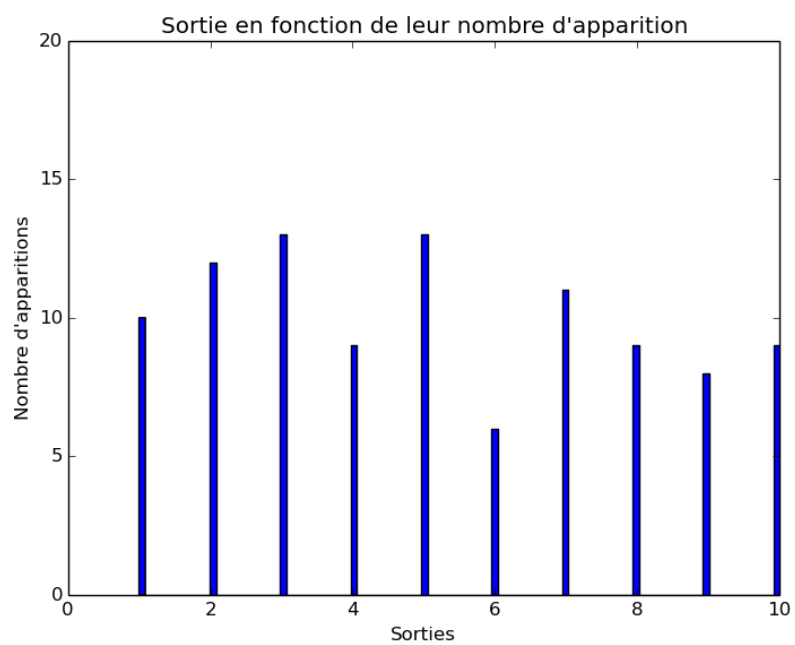


FIGURE 1.4 – Sortie en fonction de leur nombre d'apparition