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

```python
import random
import matplotlib.pyplot as plt

dices_nbr = 5

def drawing(nbr):
    drawing = []
    for i in range(nbr):
        drawing.append(random.randrange(1, 7))
    return drawing

def sum(values, dices_nbr):
    score = 0
    for i in range(dices_nbr):
        score = score + values[i]
```

```
    return score

nbr_sum = 6*dices_nbr
values = drawing(dices_nbr)
res_sum = sum(values, dices_nbr)

def possibility(sumvalue):
    possibility = 0
    dices = [1, 2, 3, 4, 5, 6]
    for i in dices:
        for j in dices:
            for k in dices:
                for l in dices:
                    for m in dices:
                            res = i + j + k + l + m
                            if res == sumvalue:
                                possibility += 1
    return possibility

print('Traitement pour '+ str(dices_nbr) +' dÃ©s')
# Le nombre maximum d'un dÃ©s multipliÃ© par le nbr de dÃ©s moins 5
# car la plus petite combinaison c'est 5.
print('Nombre de sommes possibles : ' + str(nbr_sum - 5))
print('Nombre de combinaisons possibles : ' + str(6**dices_nbr))
print('\n-> Lancement des dÃ©s...\n')
print('Somme des dÃ©s lancÃ©s : ' + str(res_sum))

print('\nTableau du nombre de combinaison possible pour chaque somme :')
values = []
sorties = []
j = k = dices_nbr
for i in range(dices_nbr, (nbr_sum) + 1):
    values.append(possibility(i))

for i in values:
    print(str(j) + ' : ' + str(i))
    j += 1
    sorties.append(j)
print('\nTableau des frÃ©quences :\n')
frequences = []
for i in range(len(values)):
    frequencesres = 100*(values[i] / (6**dices_nbr))
    frequences.append(frequencesres)
    print(str(k) + ' : ' + str(frequencesres))
    k += 1
```

```

print('\nGraphique des frÃ©quences')

plt.title("FrÃ©quence d'apparition")

plt.plot(sorties, frequences)
plt.xlabel('Sorties possibles')
plt.ylabel('FrÃ©quence (%)')
plt.show()
```

Voici le résultat généré par le script :

1.2 Nombres pseudo aléatoires

Exercice 1 :

1. Votre machine a une fonction RAND qui vous fournit de manière aléatoire u 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_exo1.py

```
import random
import matplotlib.pyplot as plt

N = 10
nbr_loop = 10000
results = {}
sorties = []
for i in range(nbr_loop):
    result = random.randrange(0, N + 1)
    sorties.append(result)  # + 1 pour Ãªtre entre 1 et 1000 inclus
    # if result in results:
    #     counter = results[result]
    #     counter += 1
    #     results[result] = counter
    # else:
    #     results[result] = 1

print("\nGraphique des rÃ©sultats en fonction de leur nombre d'apparition")


# nbr_apparition = []
# for item in results:
#     sorties.append(item)
```

```
#        nbr_apparition.append(results[item])

plt.title("Sortie en fonction de leur nombre d'apparition")
plt.hist(sorties, N)
plt.axis([0, N, 0, 0.20*nbr_loop])
plt.xlabel('Sorties')
plt.ylabel("Nombre d'apparitions")
plt.show()
```

Voici le graphe généré par le script précédent : **Réponse :** Le script suivant nous permet de répondre à la précédente question : Nous ne pouvons pas être satisfait de la fonction random.randrange fournie par le constructeur car on voit, à l'aide du graphe, que chaque bar devrait être égale à 1 / N ici 0.10 or la première est égale à 918 / 10000 = 0.0918 **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 ?

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 $(U_n)_n \in$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.
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)
2. Donner un algorithme fournissant ces nombres
3. Faire une étude pour N = 4, et A = 5678
4. Faire une étude pour N = 6, et A de votre choix
5. Que penser de cette méthode ?

1.4.2 Méthode de HEWLET-PACKARD

Elle consiste à construire la suite $(Z_n)_n \in \mathbb{N}$ définie par

1.4.3 D'autres fonctions

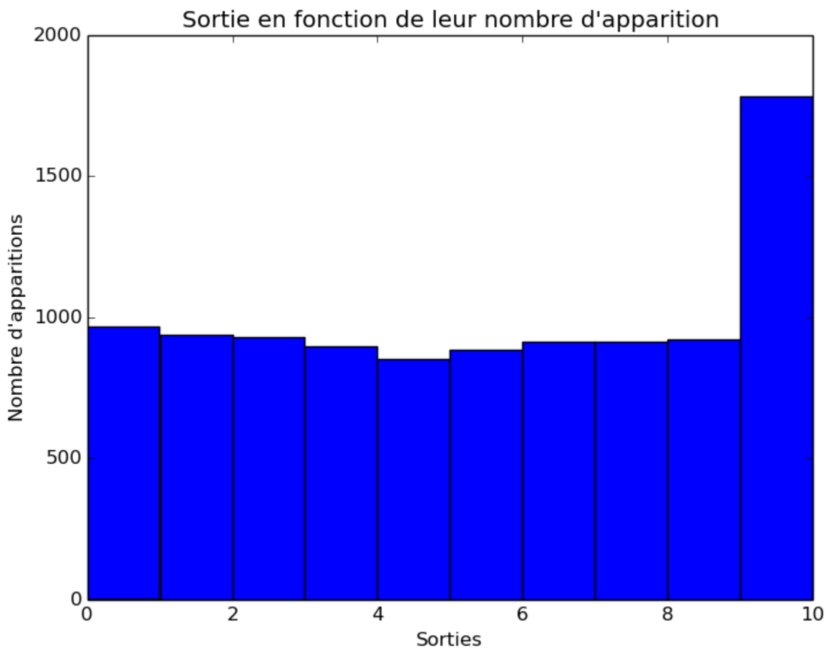

FIGURE 1.1 – Graphe