Génération de nombres aléatoires et probabilités Compte rendu

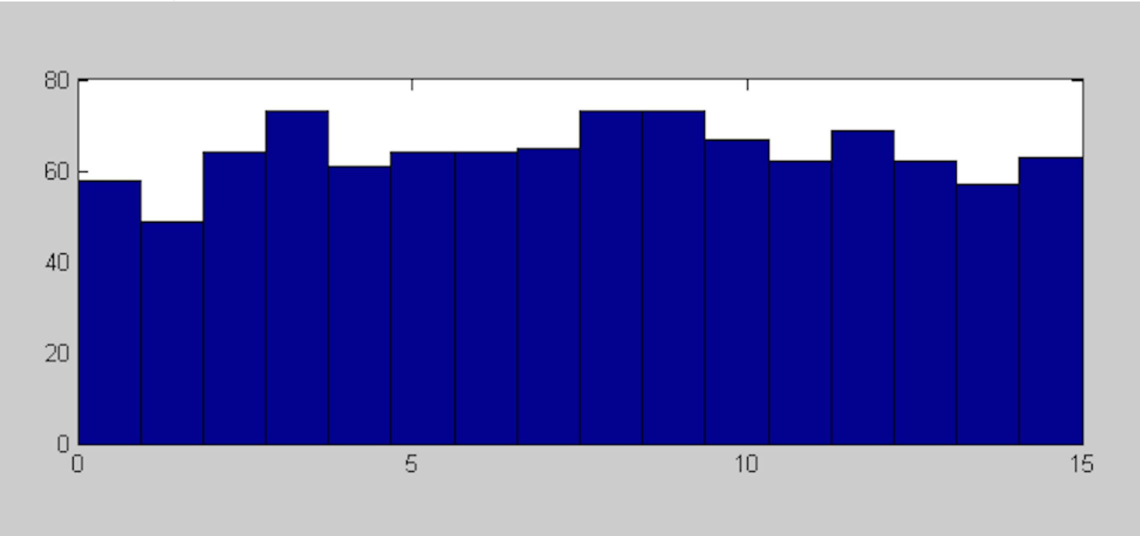
Mehdi Kitane et Amaury Courjault (B3442)

2.2.1 Test Visuel

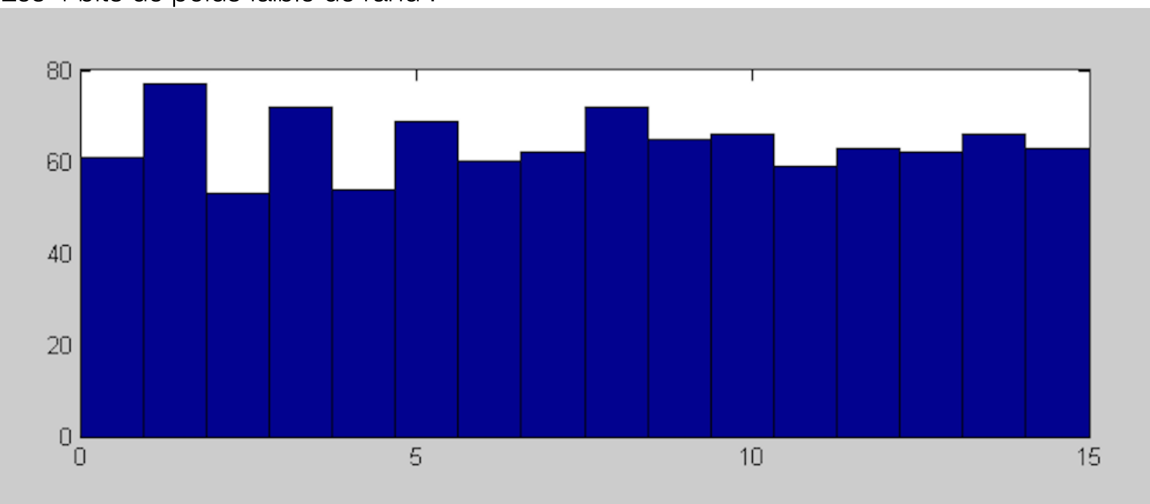
Question 1

Pour une suite de n = 1024 valeurs, voici les sorties observées.

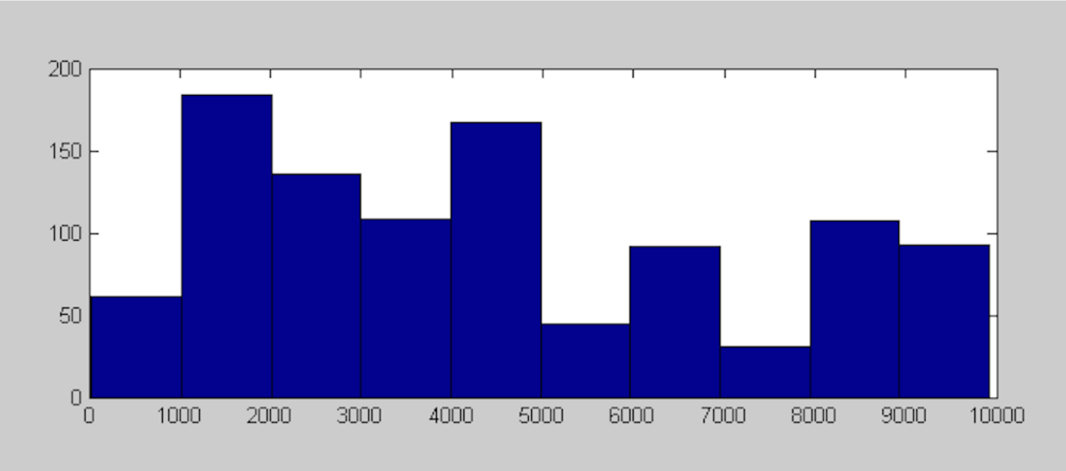
Les 4 bits de poids fort de rand :



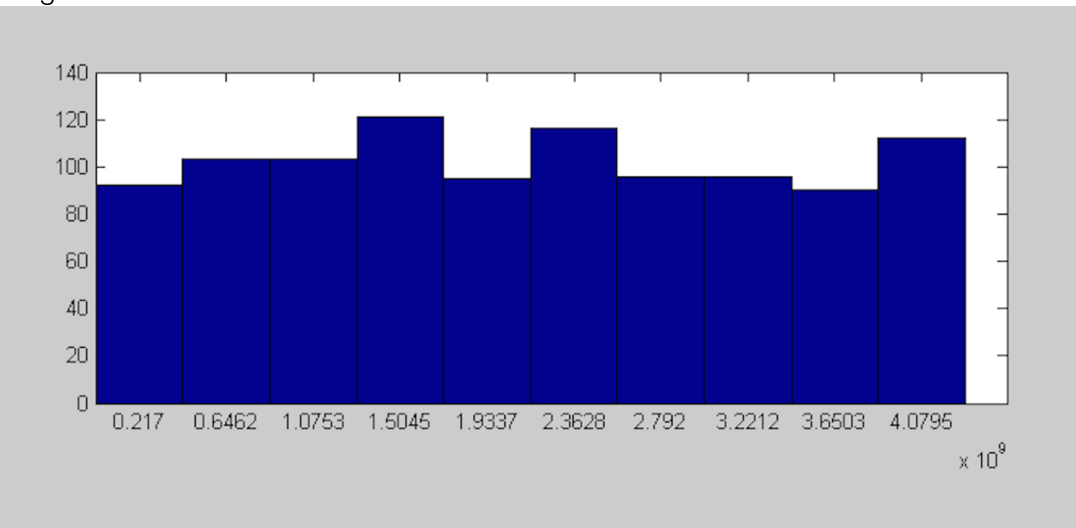
Les 4 bits de poids faible de rand :



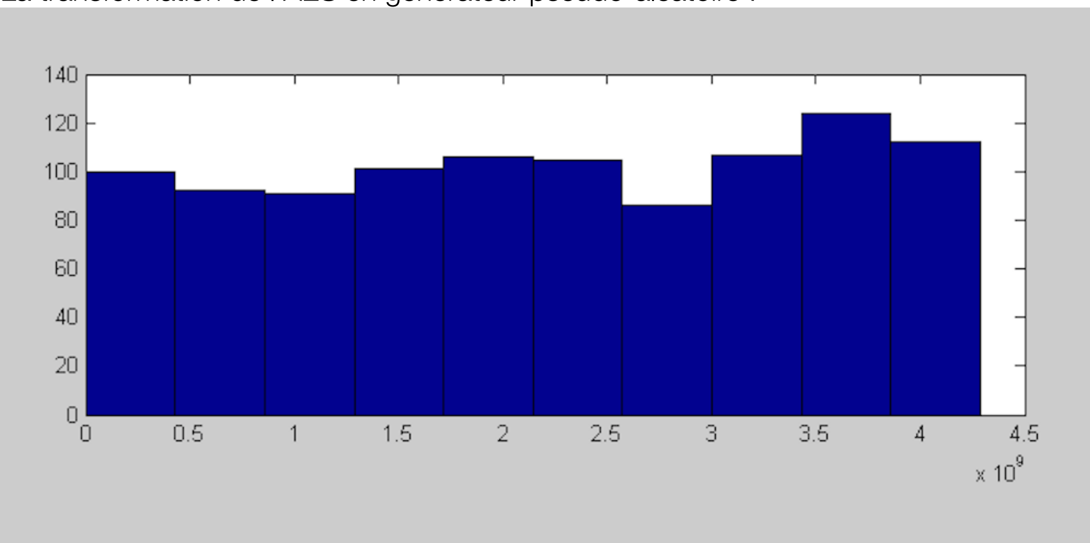
Le générateur de Von Neumann :



Le générateur Mersenne-Twister :



La transformation de l'AES en générateur pseudo-aléatoire :



On constate des fluctuations plus ou moins sur tous les graphes.
C’est normal qu’il y ait des fluctuations, mais il faut pas qu’il y’en ai trop, comme dans le Von Newmann,

AES et Mersenne sont de bons générateurs, d’après le test visuel.
Pour Mersenne-Twister, les valeurs sont autour de 100, comme prévu et sont bien réparties.
Pour Von Neumann, on a une distribution pas très uniforme, ce n’est pas un très bon générateur.

Rq : Les valeurs par défaut de hist ne sont pas adaptées a la lecture du test visuel pour le rand avec point faible et poids fort, il faut changer hist(data), par hist(data,16).

2.2.2 Test de fréquence monobits

Question 2

Rq : On a seulement ⅙ de chance d’avoir le 14eme bit « rempli » avec le générateur de Von Neumann, donc il ne faut pas trop le prendre en compte dans nos résultats de Frequency, et seulement prendre en compte les bits 0 a 13.

Mersenne-Twister :
0.833728
AES :
0.642620
Von-Neumann :
0.0000
Bit Poids Fort :
0.975070
Bit Poids Faible :
0.332670

P_{valeur} est plus petite que 0,01 que pour le test de Von-Neumann, c’est donc le seul générateur qui est d’après ce test non aléatoire.

2.3 Test des runs

Question 3

Mersenne-Twister :
0.938353
AES :
0.955942
Von-Neumann :
0.00000
Bit Poids Fort :
0.826786
Bit Poids Faible :
0.491767

P_{valeur} est plus petite que 0,01 que pour le test de Von-Neumann, c’est donc le seul générateur qui est d’après ce test non aléatoire.

3 Simulation d’une loi de probabilité exponentielle

3.1 Loi uniforme sur [0;1]

Question 4

Voir code : Méthode :

double Alea();

3.2 Loi exponentielle

Question 5

Voir code : Méthode :

double Exponentielle(double lambda);

4 Application aux files d’attentes

4.1 Files M/M/1

Question 6 :

Voir code : Méthode :

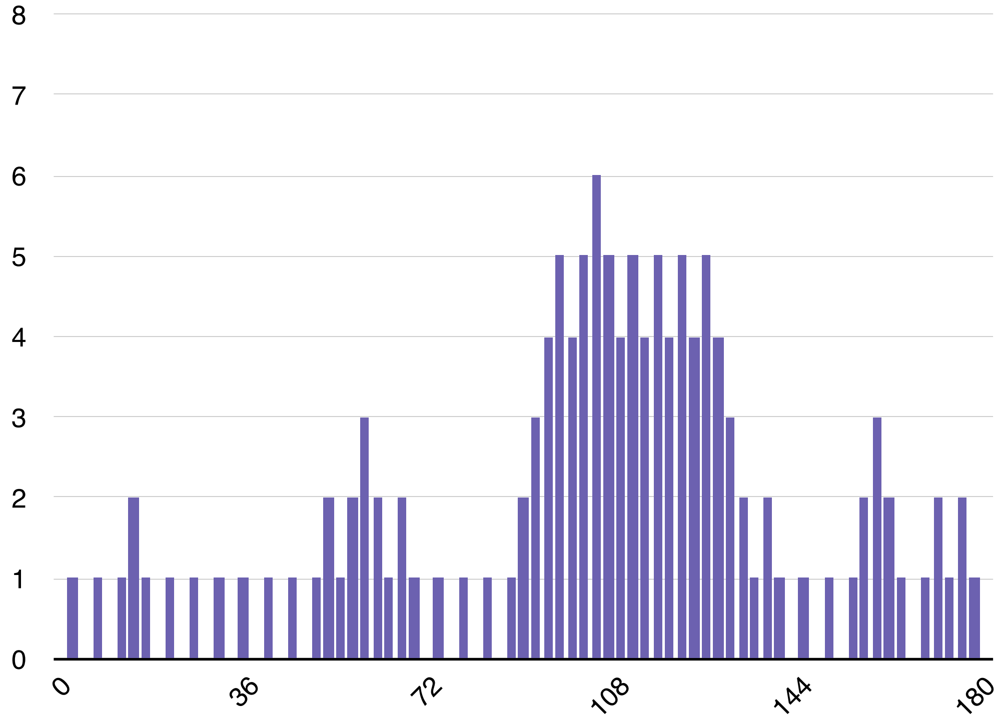
file_attente FileMM1 (double lambda, double mu, double D);

Question 7 :

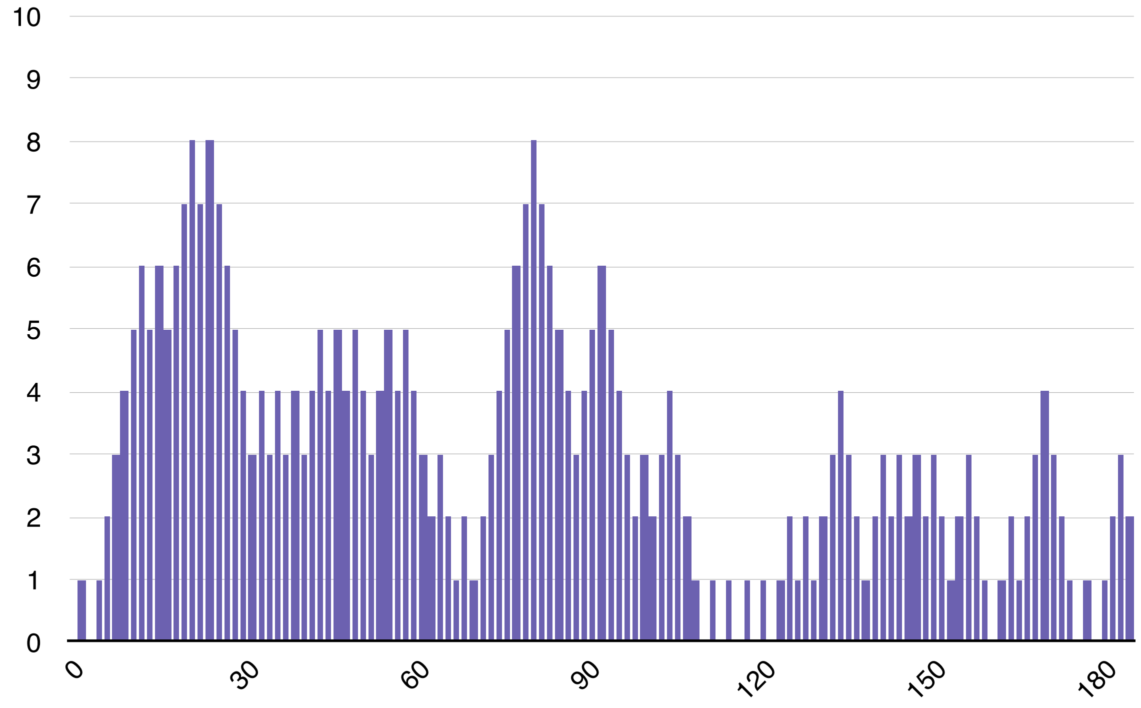
Voir code : Méthode :

evolution Calcul_evolution(file_attente a)

Evolution du nombre de clients dans le système pour λ = 0,20 μ=0,33 pendant 3h



Evolution du nombre de clients dans le système pour λ = 0,30 μ=0,30 pendant 3h



On remarque que pour 18 arrivée par heure, (Deuxième graphique) il y'a plus de gens qui attendent et plus longtemps, ce qui est logique.

Question 8 :

Pour les valeurs, λ = 0.2 et μ = 0.33
Les estimations nous donnent (Pour un lancer)
Nombre moyen de clients : 2,02
Temps de présence moyen d'un client : 9,8

Donc on retrouve bien la formule de little :

2,02 = 9,8* 0.2

On remarque que d'un lancer à un autre, nos distributions fluctuent, cependant à chaque fois ils continuent à vérifier la formule de little.

Question 9 :

Pour simuler un tel système, il suffit de remplacer μ par 2* μ quand on lance la fonction FileMM1.

Pour les valeurs, λ = 0.2 et μ = 0.33
Les estimations nous donnent (Pour un lancer)
Nombre moyen de clients : 0,49
Temps de présence moyen d'un client : 2,23

Donc on retrouve bien la formule de little :

2,23 = 0,49* 0.2

On remarque que le temps d'attente diminue bien d'à peu près 2fois par rapport à la file M/M/1, ceci est bien cohérent.

Evolution du nombre de clients dans le système pour λ = 0,30 μ=0,30 pendant 3h pour 2 serveurs