TP Analyse Numérique
Antoine BRETON Arnaud DUPEYRAT
Biome B3421

Sommaire:

- Test de la qualité aléatoire de 5 générateurs
  - 1- Question 2.2.1, Test visuel
    - A- Test visuel avec 1024 valeurs.
    - B- Test visuel avec un million de valeurs.
  - 2- Question 2.2.2, Test de fréquence monobit
  - 3- Question 2.2.3, Test des runs.
  - 4- Bilan Partie Test
- Simulation d'une loi de probabilité aléatoire
  - 1 - Question 3.1 Simulation de loi Uniforme sur [0,1]
  - 2 - Question 3.2 Simulation de la loi exponentielle
- Applications aux files d'attentes
  - 1 - Question 6 File MM1
  - 2 - Question 7 Simulation File MM1 et évolution
  - 3 - Question 8 Calcul de moyennes.

Test de la qualité aléatoire de 5 générateurs

1- Question 2.2.1, Test visuel

Nous utilisons les quatre générateurs de nombre aléatoires suivant :

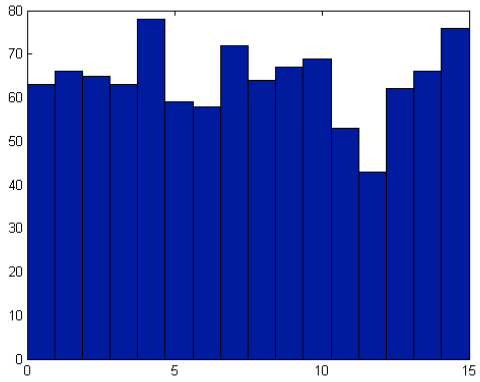
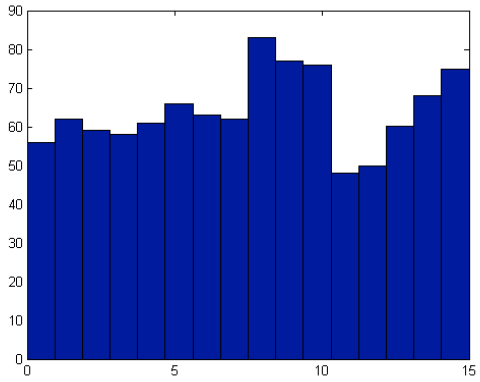
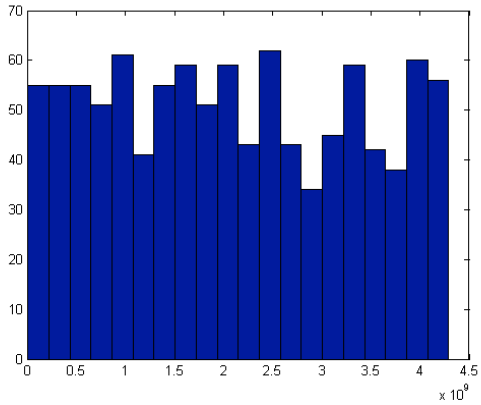
- Le générateur de Von Neumann.
- Le générateur de Mersenne-Twister.
- Le générateur AES.
- La fonction rand() du c++; On isole les 4 bits de poids forts/faibles par décalage et nous sélectionnons les bits intéressant par utilisation d'un masque 0xF, nous permettant de récupérer les 4 derniers bits.

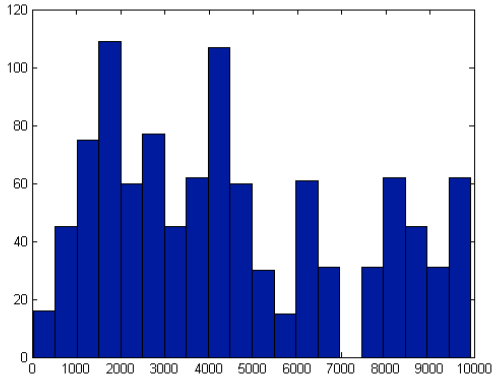
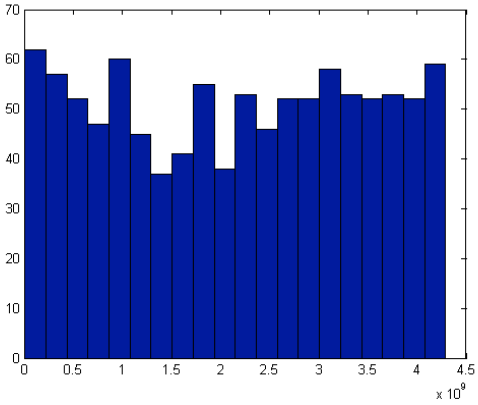
On génère pour chaque méthode 1024 nombres aléatoire. Nous représentons ensuite sous la forme d'un histogramme la répartition générale des nombres aléatoire. Nous réalisons l'histogramme à l'aide du logiciel MATLAB.

Le test visuel est considéré comme valide si la répartition semble homogène et uniforme. D'autre part pour confirmer cet à priori nous avons généré des tests produisant 1 000 000 de valeurs, si les générateurs sont effectivement aléatoire nous devrions s'approcher d'une parfaite homogénéisation des résultats.

Certains morceaux du code est dédoublé volontairement pour éviter les conversions implicites du compilateur.

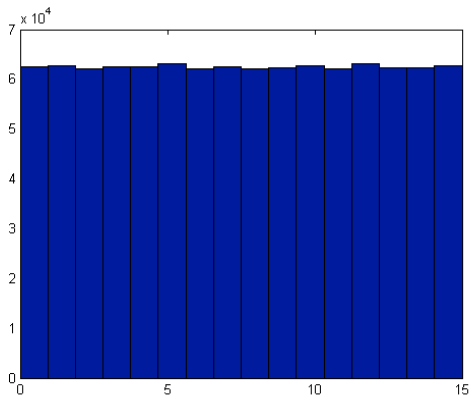
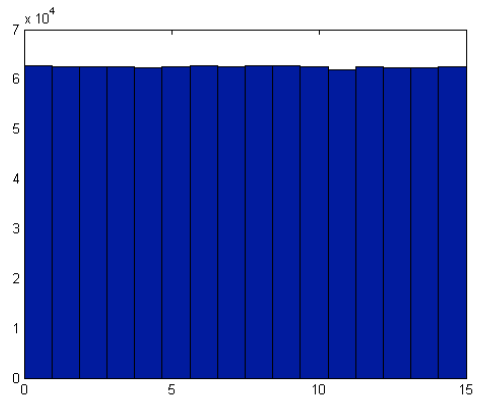
A- Test visuel avec 1024 valeurs.

Générateur rand(), bits poids faible	
Générateur rand(), bits poids fort	
Générateur Mersenne-Twister	

Générateur Von Neumann	
Générateur AES	

A travers ces histogrammes, on constate que la fonction rand() du C, le générateur de Mersenne-Twister et l'AES passent le test visuel. Le générateur de Von Neumann, par contre, échoue ce test : la distribution des nombres générés n'est pas uniforme, ce qui confirme ce qui a été dit durant le TP.

B- Test visuel avec un million de valeurs.

Générateur rand(), bits poids faible	Générateur rand(), bits poids fort
	

Nous pouvons observer une homogénéisation plus grande des résultat pour les générateurs associé à la fonction rand().
Ce qui confirme l'idée que les générateurs sont effectivement aléatoire.

2- Question 2.2.2, Test de fréquence monobit

La question 2.2.2 nous permet de mettre en application le théorème de limite centrale. Il nous est possible de vérifier que l'indépendance dans une suite de bits est vérifié.
Le test est effectué sur une suite de 1024 mots de longueurs variable dépendant des générateurs utilisés:

- Le générateur de Von Neumann, mots codés sur 16 bits.
- Le générateur de Mersenne-Twister mots codés sur 32/64 (dépend de l'architecture) bits.
- Le générateur AES mots codés sur 32 bits.
- La fonction rand() mots codés sur 4 bits.

Nous obtenons les résultats suivants:

Méthodes	**Pvaleur**
Générateur Mersenne-Twister	Variable entre 0,1 et 1
Générateur Von Neumann	0.0
Générateur AES	Approximativement 0,33
rand() : Poids fort	Approximativement 0,45
rand() : Poids faible	Approximativement 0,66

Afin que le test soit validé, il est nécessaire d'avoir une Pvaleur comprise entre 0,1 et 1.
Nous concluons alors que seul le générateur de Von Neumann ne valide pas le test de fréquence monobit.

3- Question 2.2.3, Test des runs.

La question 2.2.3 a pour but de s'intéresser à la longueur des suites successives de zéros dans la séquence de 1024 mots générer "aléatoirement".

Méthodes	Pvalue
Générateur Mersenne-Twister	Variable entre 0,1 et 1
Générateur Von Neumann	0.0
Générateur AES	Approximativement 0,72
rand() : Poids fort	Approximativement 0,17
rand() : Poids faible	Approximativement 0,13

Le critère de sélection étant que Pvalur doit être compris entre 0,1 et 1.
Nous obtenons des résultats identique aux tests précédents.

4- Bilan Partie Test

En bilan de ces trois tests, nous pouvons affirmer que le générateur de Von Neumann ne possède pas un caractère aléatoire suffisant pour être considéré comme un bon générateur aléatoire.

Nous pouvons aussi remarquer, que les tests que nous avons effectués ne semble pas suffisant pour distinguer de plus petites différences de "qualité aléatoire" entre les différentes types de générateur. En effet il est dit dans le sujet que les bits de poids faible de la fonction rand du C possède une moins bonne "qualité aléatoire".
Or les résultats de comparaison entre les bits de poids faible et les bit de poids fort sont faiblement différents et avantage le générateur associé au bit de poids faible de la fonction rand() dans certains tests.

Pour confirmer l'hypothèse avancée dans l'énoncé du TP, il serait intéressant de traiter un nombre plus important de données.

Simulation d'une loi de probabilité aléatoire

1 - Question 3.1 Simulation de loi Uniforme sur [0,1]

Notre fonction Alea() permet de ramener une loi uniforme [0,N], sur l'ensemble [0,1].
Le choix du générateur étant libre nous avons choisi le générateur associé à la fonction Mersenne-Twister.

2 - Question 3.2 Simulation de la loi exponentielle

Notre fonction Exponentielle(double lambda) permet retourner une valeur issue de la réalisation d'une loi exponentielle de paramètre lambda.

Applications aux files d'attentes

Nous considérons une file d'attente dont l'arrivée et le départ des clients suivent deux lois exponentielles de paramètres respectives lambda et mu.

1 - Question 6 File MM1

Nous avons construit la fonction **FileMM1** qui retourne les dates d'arrivée et de départ des clients sur une file d'attente en suivant une loi exponentielle.
Dans cette fonction, nous créons une structure file d'attente qui contient deux tableaux (arrivées et départs des clients) ainsi que leur taille respectives.

Voici le fonctionnement des tableaux :

- arrivee[c] = t correspond à " le client c est arrivé à l'instant t".

- depart[c] = t correspond à " le client c est parti à l'instant t".

La fonction **FileMM1** prend en paramètre une durée D. Dans notre système, nous décidons d'arrêter l'analyse quand tous les clients **arrivés** pendants durée D sont partit.
A la fin du traitement nous obtenons alors un nombre de client restant dans notre système nul.

2 - Question 7 Simulation File MM1 et évolution

Nous avons généré une file d'attente utilisant la fonction précédente avec Lambda = 0,20 et mu = ⅓ sur une durée D = 180 (minutes).

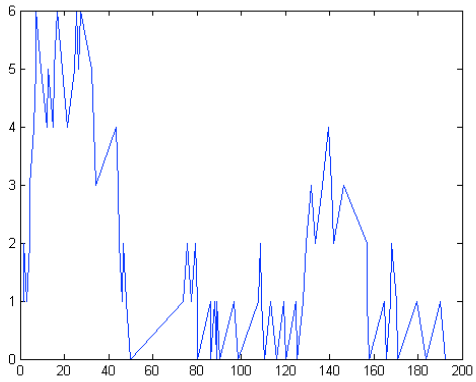
Pour afficher l'évolution du nombre de clients en fonction du temps nous utilisons une fonction "évolution(file_attente)". Le principe de cette fonction est le suivant :

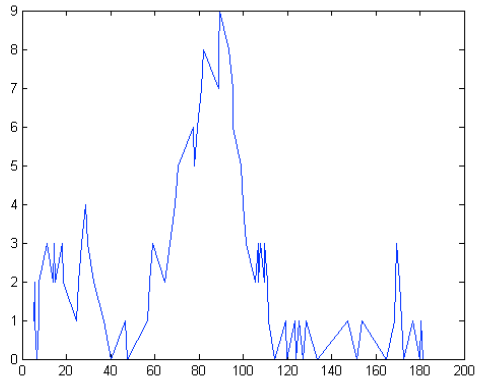
- Trier le tableau des départs (grâce à un algorithme Quick sort)

- Faire une boucle à double compteur d'indice (un pour chaque tableaux)
qui, à chaque itération après avoir comparé le minimum des deux valeurs tableaux aux indices courants, rempli un tableau "temps" avec l'instant du prochain changement au sein de notre système
Si le prochain instant le plus proche provient du tableau des départs, alors on diminue le nombre de client courant, sinon on l'augmente.

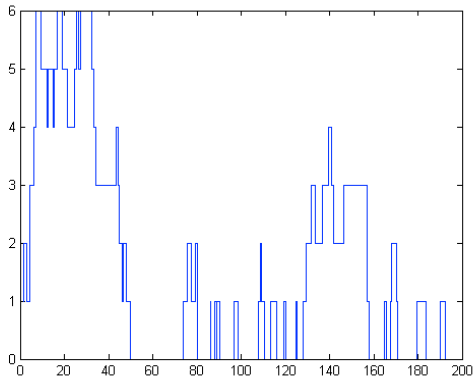
voici les résultats qui résultent ce cette simulation, effectué pour deux jeux de tests différents :

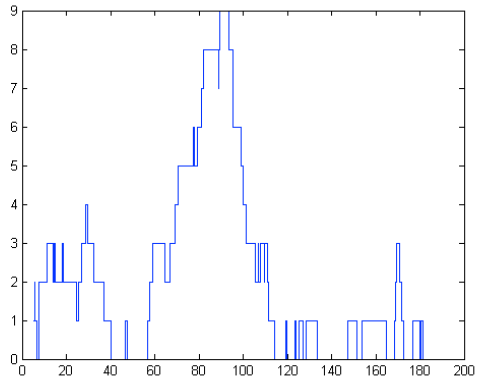
utilisation de la méthode plot





utilisation de la méthode Stairs





Les valeurs pour les tests précédents sont les suivantes :
lambda = 0,2
mu = 0,33
D = 180

3 - Question 8 Calcul de moyennes.

Pour confirmer la formule de Little, il est nécessaire de calculer différentes moyennes :

- Premièrement nous avons du calculer le nombre moyen de clients dans le système. Afin d'obtenir des résultats correctes nous avons calculer la moyenne pondéré du nombres de client par le temps passé dans l'état courant du système.
Nous obtenons alors un nombre moyen de clients du système pouvant varier de 0,5 à 1,9.

- Deuxièmement nous avons du calculer le temps moyen que le client passe dans la file d'attente. Nous obtenons des résultats variant aux alentours de 15.Ce qui semble cohérent avec la formule de Little.

Néanmoins nous n'arrivons pas à retrouver les résultats exacts de la formule de Little. Pour que nos résultats expérimentaux se rapprochent de la théorie, il est nécéssaire d'étudier les files d'attentes avec un plus grands nombres de valeurs

En théorie, on a lambda = 0.2. Donc si la formule de Little est vérifié nous avons :

<E> = lambda * t

<E> : Nombre moyen de clients dans le système.

Lambda : fréquence d'entrée.

t: : temps de présence dans le système.