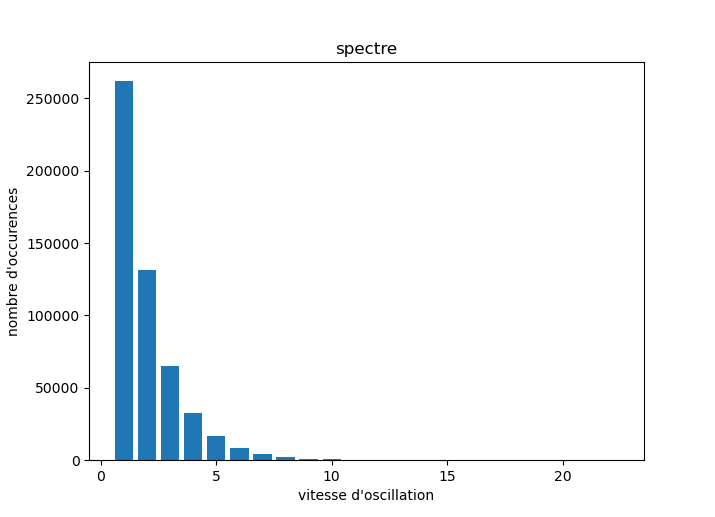
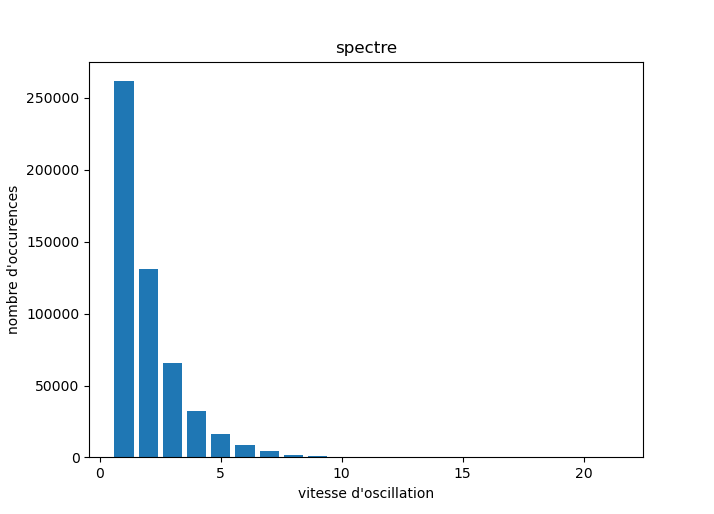
**Partie 1 : PRNG**

La génération de nombres aléatoires par ordinateur se fait grâce à des algorithmes, comme par exemple la méthode de Von Neumann ou encore l’algorithme Mersenne Twister. Cependant, un algorithme est une suite d’opérations prédéfinies. Il est donc déterministe et son résultat l’est aussi, ce qui est en opposition avec la génération de nombres aléatoires puisqu’il faudrait que le résultat soit complètement imprévisible. On a donc nommé les nombres issus de ces générateurs des nombres pseudo-aléatoires, d’où le nom Pseudo Random Number Generator (PRNG).

Afin de tester la qualité d’un PRNG, nous avons programmé en python un programme qui nous donne une séquence de bits, assez longue (2^20) pour avoir des résultats significatifs. La bibliothèque random s’appuie sur l’algorithme de Mersenne Twister, qui a été développé par Makoto Matsumoto et Takuji Nishimura en 1997, et plus particulièrement la variante MT19937. Elle a notamment l’avantage d’avoir une période de 219937-1 et d’être plus rapide que la plupart des PRNG de même qualité. Cependant, il présente plusieurs défauts, comme celui de la complexité linéaire ou encore de ne pas avoir passé avec succès l’algorithme de test TestU01.

Pour tester la qualité de Mersenne Twister, nous avons donc mesuré la fréquence d’apparition, la vitesse d’oscillation, qui correspond au nombre de bits dans le même état à la suite avant l’apparition d’un bit dans l’état opposé, et le spectre de plusieurs séquences.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test n° | 1 | 2 | 3 | 4 | 5 |
| Fréquence des 1 | 49.991 | 49.983 | 50.042 | 49.993 | 50.011 |
| Vitesse d’oscillation moyenne | 2.000 | 2.001 | 1.999 | 2.001 | 2.002 |



On remarque tout d’abord que les résultats sont très similaires pour les trois tests que nous avons effectués. On reconnait notamment la loi normale sur le spectre, ce qui nous permet de déduire que plus on a un nombre de bits pareils à la suite élevé, plus la probabilité de rencontrer le bit opposé augmente. Or cela ne devrait pas arriver sur un tirage aléatoire car on ne devrait pas pouvoir calculer la probabilité de la valeur du prochain bit en fonction des valeurs précédentes.

Nous avons cherché quel était la plus grande suite de bits répétée dans la séquence. Par soucis de rapidité de notre programme, nous avons réduit la taille de la séquence à 2^10 bits. Nous obtenons les résultats suivants.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test n° | 1 | 2 | 3 | 4 | 5 |
| Taille de la plus grande séquence répétée | 17 | 18 | 22 | 16 | 18 |