
INSEA

20

Les PRNG d’aujourd’hui:

1. Se basent sur un mélange F d'opérations (addition, multiplication, reste de division, shift de bits ...)

2. Définissent toujours une suite d'entiers périodique $x_{n+1} = F(x_n)$ avec $x_i \in [0, M]$, avec M très grand et une période très très grande (même avec 10^{12} chiffres / seconde, il faut 10^{18} années pour voir toute la séquence).

3. Définir $u_i = \frac{x_i}{M}$ permet d'obtenir des nombres qui "semblent" suivre une loi uniforme $\mathcal{U}([0,1])$.

4. Il existe plusieurs tests qui permettent de mesurer la qualité statistique d'un PRNG (corrélation, biais de sélection...)

5. On appelle x_0 *la graine (seed)* du PRNG: si on la connait, on peut déterminer toute la suite.

6. Fixer la seed permet de garantir la reproducibilité d'un programme avec des nombres (pseudo) aléatoires.

7. Les PRNG les plus connus sont:

(a) *Mersenne Twister (1997)*: utilisé par défaut `Numpy <= 1.16`.

(b) *PCG64 (2014)*: adopté par `Numpy >= 1.17` en 2019.

8. Ne doivent pas être utilisés en cryptographie (générer des clés privées / mots de passe)

Les PRNG aujourd’hui

np.random.RandomState

np.random.default_rng

Les PRNG aujourd'hui

Les PRNG d'aujourd'hui:

1. Se basent sur un mélange F d'opérations (addition, multiplication, reste de division, shift de bits ...)
2. Définissent toujours une suite d'entiers périodique $x_{n+1} = F(x_n)$ avec $x_i \in [0, M]$, avec M très grand et une période très très grande (même avec 10^{12} chiffres / seconde, il faut 10^{18} années pour voir toute la séquence).
3. Définir $u_i = \frac{x_i}{M}$ permet d'obtenir des nombres qui "semblent" suivre une loi uniforme $\mathcal{U}([0, 1])$.
4. Il existe plusieurs tests qui permettent de mesurer la qualité statistique d'un PRNG (corrélation, biais de sélection...)
5. On appelle x_0 *la graine (seed)* du PRNG: si on la connait, on peut déterminer toute la suite.
6. Fixer la seed permet de garantir la reproducibilité d'un programme avec des nombres (pseudo) aléatoires.
7. Les PRNG les plus connus sont:
   (a) *Mersenne Twister (1997)*: utilisé par défaut `Numpy <= 1.16`. np.random.RandomState
   (b) *PCG64 (2014)*: adopté par `Numpy >= 1.17` en 2019. np.random.default_rng
8. Ne doivent pas être utilisés en cryptographie (générer des clés privées / mots de passe)

INSEA

1. Pourquoi Monte-Carlo ? (Exemple de modèle hiérarchique)
2. Introduction à la méthode Monte-Carlo (historique, PRNG)
3. Algorithmes de simulation i.i.d (PRNG, transformation, rejet)
4. Méthodes MCMC (Gibbs, Metropolis)
5. Diagonstics de convergence MCMC
6. Méthodes MCMC avancées (Langevin, HMC, NUTS)

INSEA

Les PRNG permettent de générer des échantillons i.i.d $X_1, \ldots, X_n \sim \mathcal{U}([0,1])$.