Cas plus complexe: Fiabilité d'un vaccin

Pfizer annonce un vaccin efficace à 90% puis Moderna a 94.4% puis Pfizer à 95%. Qui gagne et de combien?

Cas plus complexe: Fiabilité d'un vaccin

Pfizer annonce un vaccin efficace à 90% puis Moderna à 94.4% puis Pfizer à 95%. Qui gagne et de combien?

	Placebo	Vaccinés
Pfizer	8/20500	162/20500
Moderna	5/30000	90/30000

Idée: $1 - 8/162 \approx 0.951$, $1 - 5/90 \approx 0.944$.
Ces chiffres sont ils estimés de façon précises?

Un peu de simulation sur l'efficacité des vaccins

Population de 20500 personnes

P_V = probabilité qu'une personne vaccinée tombe malade

P_N = " " non vaccinée " .

$$\text{Eff} = 1 - \frac{P_V}{P_N}$$

On mesure : N_V = nombre de personnes vaccinées tombées malades : $B(20500, P_V)$

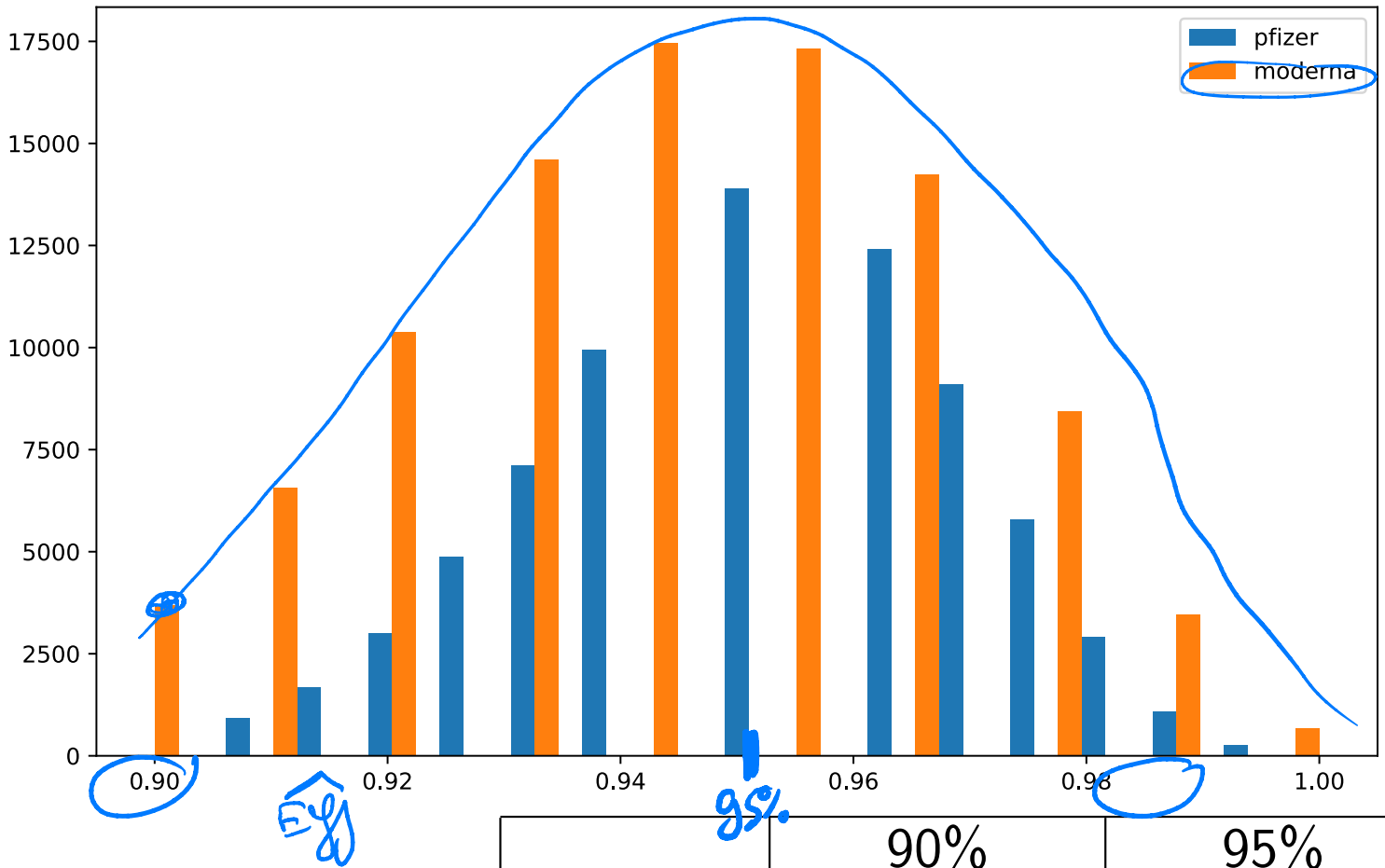
N_N = " " NV " " : $B(20500, P_N)$

on estime $\hat{P}_V = \frac{N_V}{20500}$ et $\hat{P}_N = \frac{N_N}{20500}$

Question : est-ce que $\hat{P}_V \approx P_V$ et $\hat{P}_N \approx P_N$

$$\widehat{\text{Eff}} = 1 - \frac{\hat{P}_V}{\hat{P}_N}$$

Un peu de simulation sur l'efficacité des vaccins



1000 000 tirages de n_N et n_V pour chacun des vaccins.

Eff

95%

Confidence intervals:

	90%	95%
Moderna	0.900-0.978	0.889-0.989
Pfizer	0.920-0.975	0.914-0.981

Plan du cours

1. Rappel de Probabilités
   - Événements et espaces probabilisés
   - Variables aléatoires
   - Inégalités de concentration, intervales de confiance
2. Génération d'aléatoire et simulation
   - Comment générer de l'aléatoire?
   - Générer selon une loi donnée
   - Exemple de simulation: la formule de Lindley
3. Algorithmes randomisés
   - Algorithmes "Las Vegas": Quicksort / Quickselect
   - Algorithmes "Monte Carlo"
   - Algorithmes décentralisés / apprentissage par renforcement

Quelle source d'aléatoire?

On utilise une suite de variables aléatoire *i.i.d.* (indépendantes et identiquement distribuées).

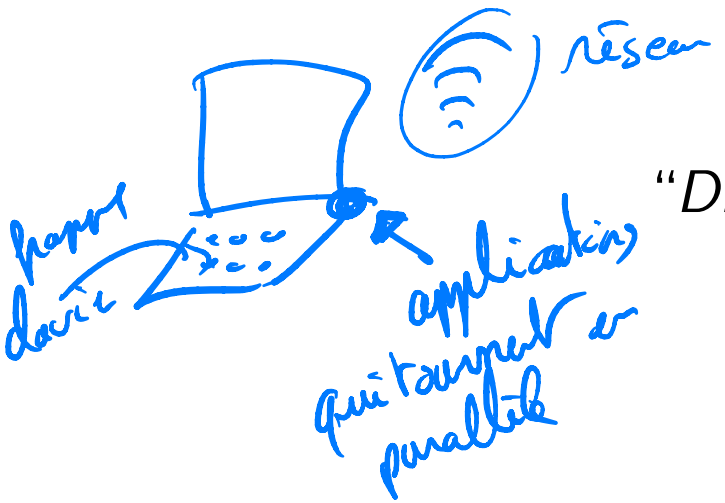
Questions:

- Comment générer une suite de v.a. $X_i \sim \mathrm{Unif}[0,1]$ indépendantes?
- Comment utiliser des variables uniformes pour en générer d'autres.

Python :

numpy.random.rand() ~ Unif([0;1])

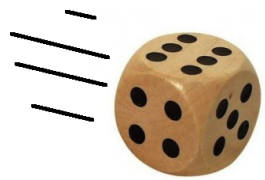
Un ordinateur est-il aléatoire?



"*Dieu ne joue pas aux dés*"

Albert Einstein

Le hasard sert à modéliser ce qu'on ne connaît pas exactement.



`/dev/random` v.s. `/dev/urandom`)

"vrai aléatoire" → peu d'aléatoire

Cryptographie

En pratique, on n'utilise des générateurs pseudo-aléatoires

Exemple: congruence linéaire

$$\begin{cases} x_0 = \text{graine} \\ x_{n+1} = a x_n \bmod m \end{cases} \qquad \in \{1, \dots, m\}$$

Sur ma machine (fonction $\mathrm{rand}()$ de la libc): $a = 16807, m = 2^{31} - 1$.

Est-ce que cela a l'air aléatoire?

$x_1, x_2, x_3, \dots$

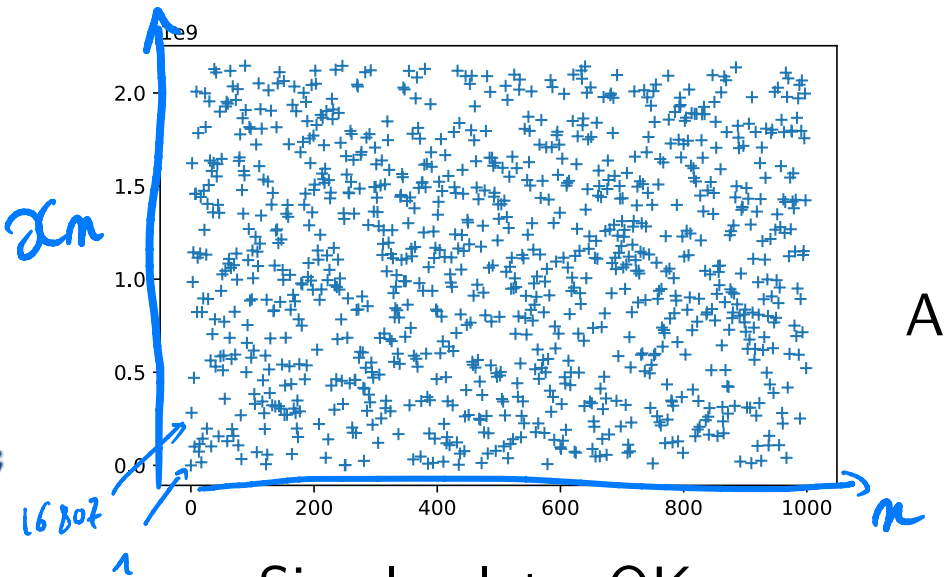
Tests basics

$x_1 = 1$

$x_{n+1} = 16807\, x_n \bmod 2^{31} - 1$

$2^{31} - 1 \approx 2.10^9$

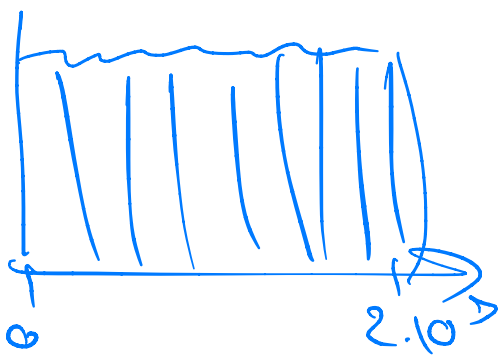
```
#include<cstdlib>
#include<iostream>

int main() {
  srand(1);
  for(int i=0;i<1000;i++)
    std::cout << rand() << "\n";
}
```



Simple plot : OK

Autocorrélation:

OK, répartition: OK.



Période d'un générateur pseudo-aléatoire

Cette source a l'air aléatoire mais:

$$x_{n+1} = ax_n \bmod m$$

$x_n = x_0$ implique que $x_{n+1} = x_1$, etc.

On appelle le plus petit n telle que $x_n = x_0$ la **période** du générateur aléatoire.

```
#include<cstdlib>
#include<iostream>

#define TWO_TO_31 2147483648

int main() {
  srand(42);
  std::cout << rand() << "\n";
  for (long long int i=0;i<TWO_TO_31-3;i++) rand();
  std::cout << rand() << "\n";
}
```

Après < 15secondes:

```
705894
705894
```

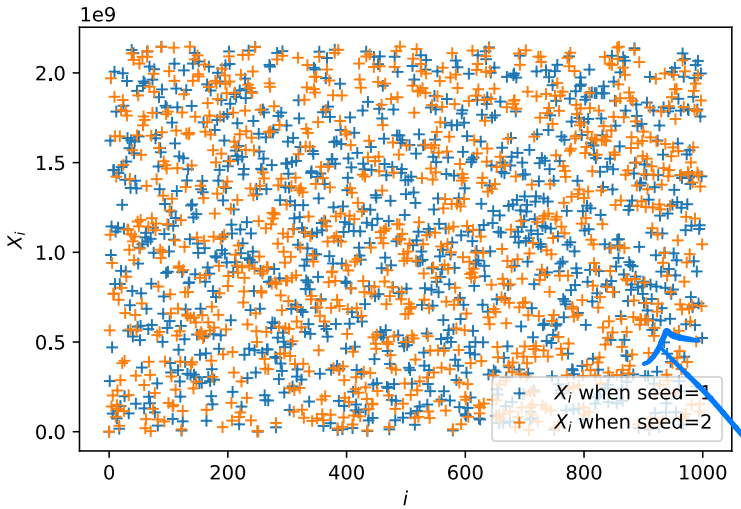
Problème des graines parallèles

$x_0 =$ graine

$x_{n+1} = a x_n \bmod m$

Graine=1

Graine=2



$x_0 = 1$ | $x_{n+1} = a x_n \bmod m$

$y_0 = 2$ | $y_{n+1} = a y_n \bmod m$

$2 x_n = y_n \bmod m$

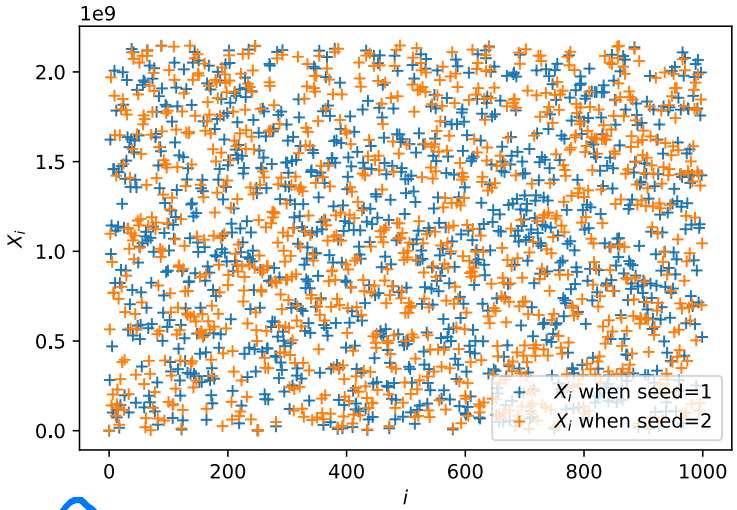
on ne le voit pas

Problème des graines parallèles

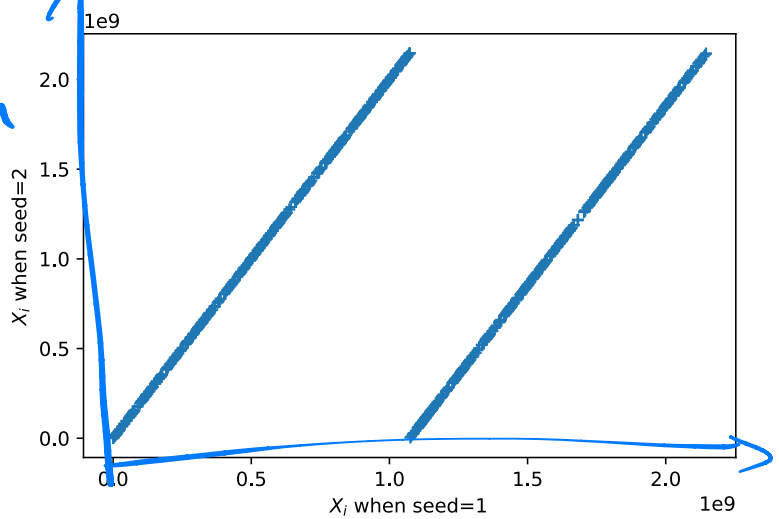
$x_0 =$ graine $\qquad x_{n+1} = ax_n \bmod m$

Graine=1

Graine=2



Graine=2



Graine=1

Take-home message

Lors de l'utilisation d'un générateur pseudo-aléatoire:

- Attention à la période
- Le générateur utilise une graine.
  - Toujours sauver la graine (pour pouvoir reproduire la même simulation)
  - Pour des réplications indépendantes, utiliser des graines différentes.

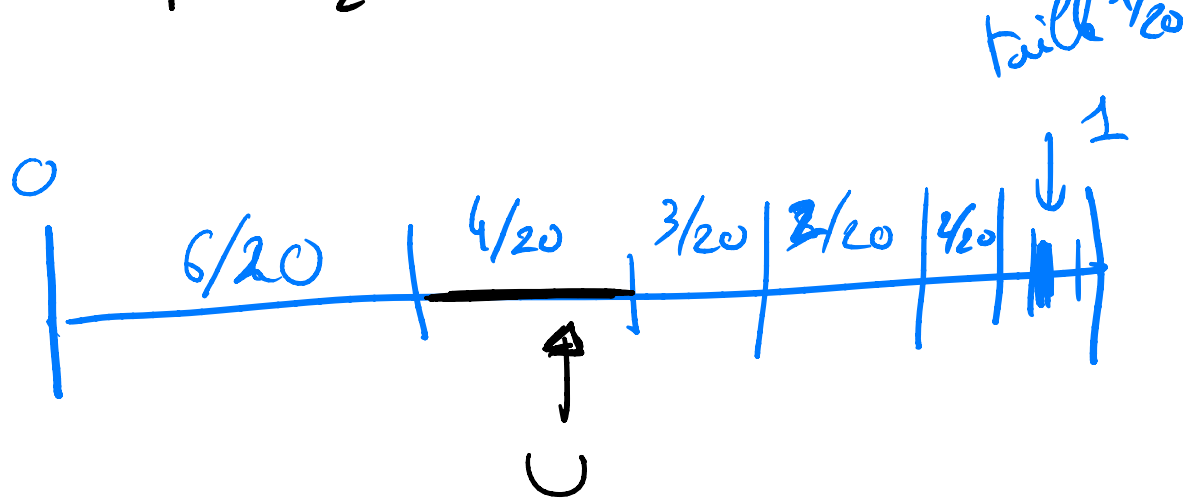
Table des matières

Comment générer une v.a. selon une distribution donnée?

On se donne $U \sim Unif([0,1])$.

Exemple 1: Comment générer X tel que $\mathbb{P}[X = i] = \pi_i$, avec
$\pi_1 = 6/20, \pi_2 = 4/20, \pi_3 = 3/20, \pi_4 = \pi_5 = 2/20, \pi_6 = \pi_7 = \pi_8 = 1/20$

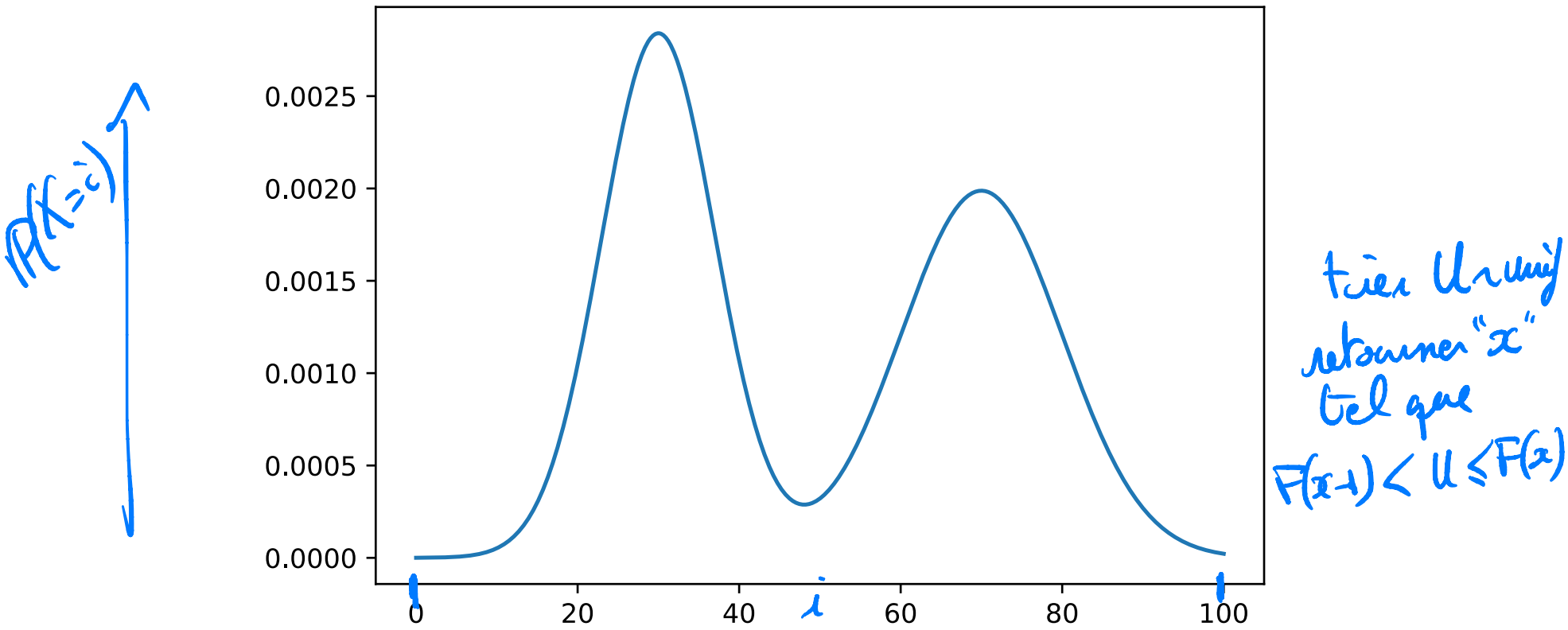
```
U = unif()
if U ≤ 6/20 :        ) arrive avec proba 6/20
    return 1
elif U ≤ 10/20 :     ) arrive avec proba 10/20 - 6/20 = 4/20
    return 2
elif U ≤ 13/20
    return 3
elif U ≤ 15/20
    return 4
elif U ≤ 17/20
    return 5
  :
```



Comment générer une v.a. selon une distribution donnée?

On se donne $U \sim Unif([0,1])$.

Exemple 2:



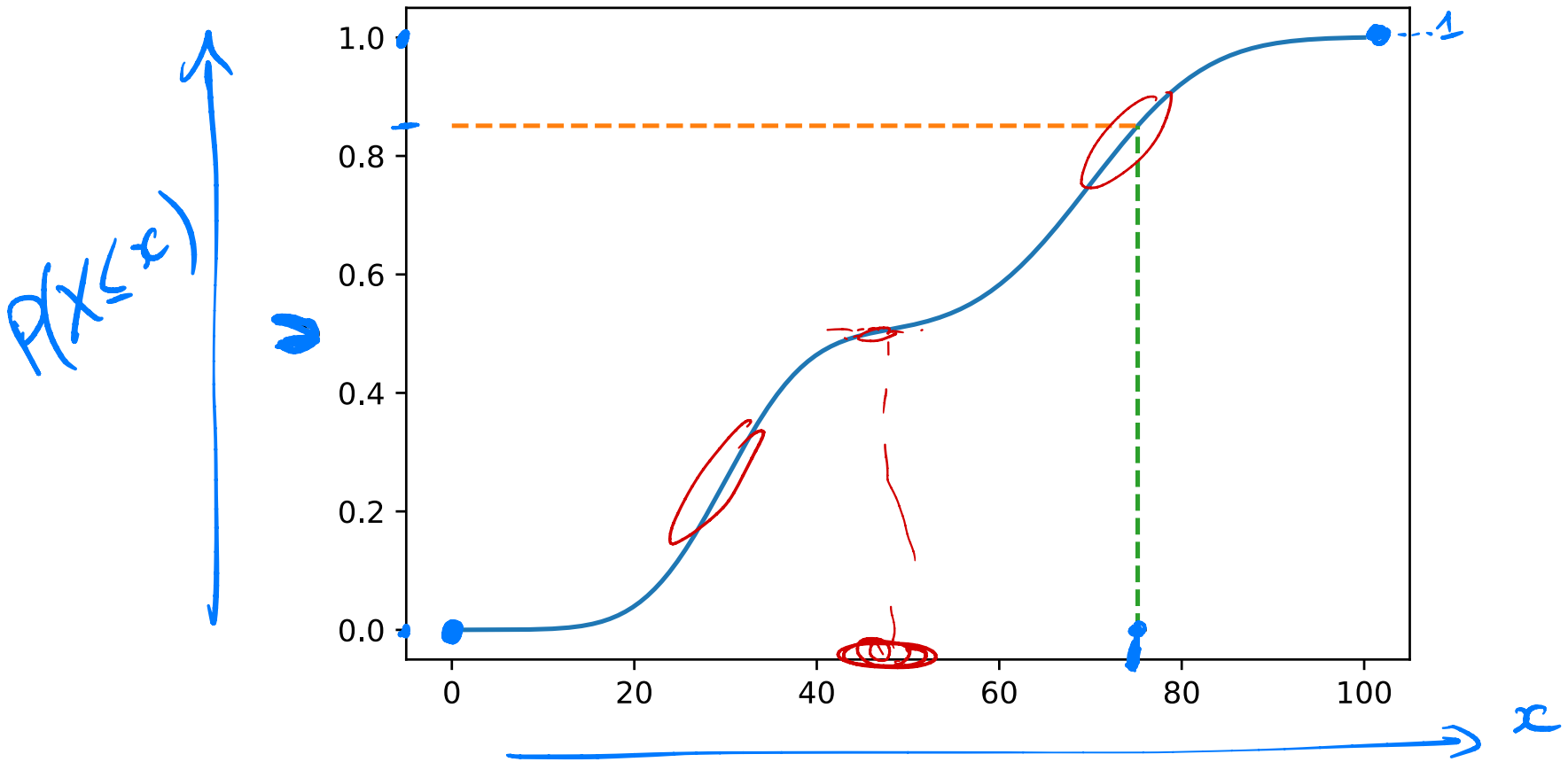
$\mathbb{P}(X=i)$

tirer Unif
retourner "x"
tel que
$F(x-1) < U \leq F(x)$

$$F(x) = \mathbb{P}(X \leq x) = \sum_{i=0}^{x} \mathbb{P}(X=i)$$

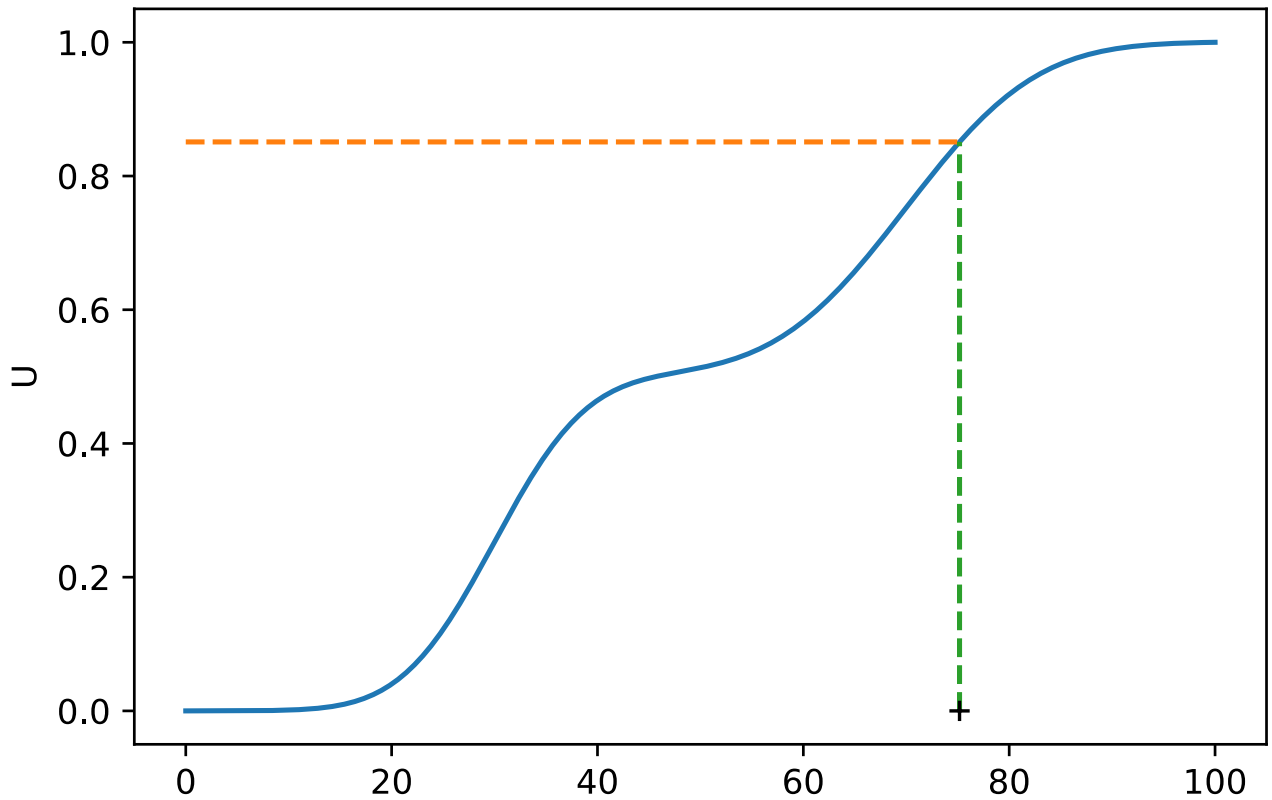
Méthode de la fonction inverse

Rappel, la fonction cumulatrice de distribution F est une fonction $F : \mathbb{R} \to [0, 1]$ telle que $F(x) = \mathbb{P}[X \leq x]$.



Méthode de la fonction inverse

Rappel, la fonction cumulatrice de distribution F est une fonction $F : \mathbb{R} \to [0, 1]$ telle que $F(x) = \mathbb{P}[X \leq x]$.

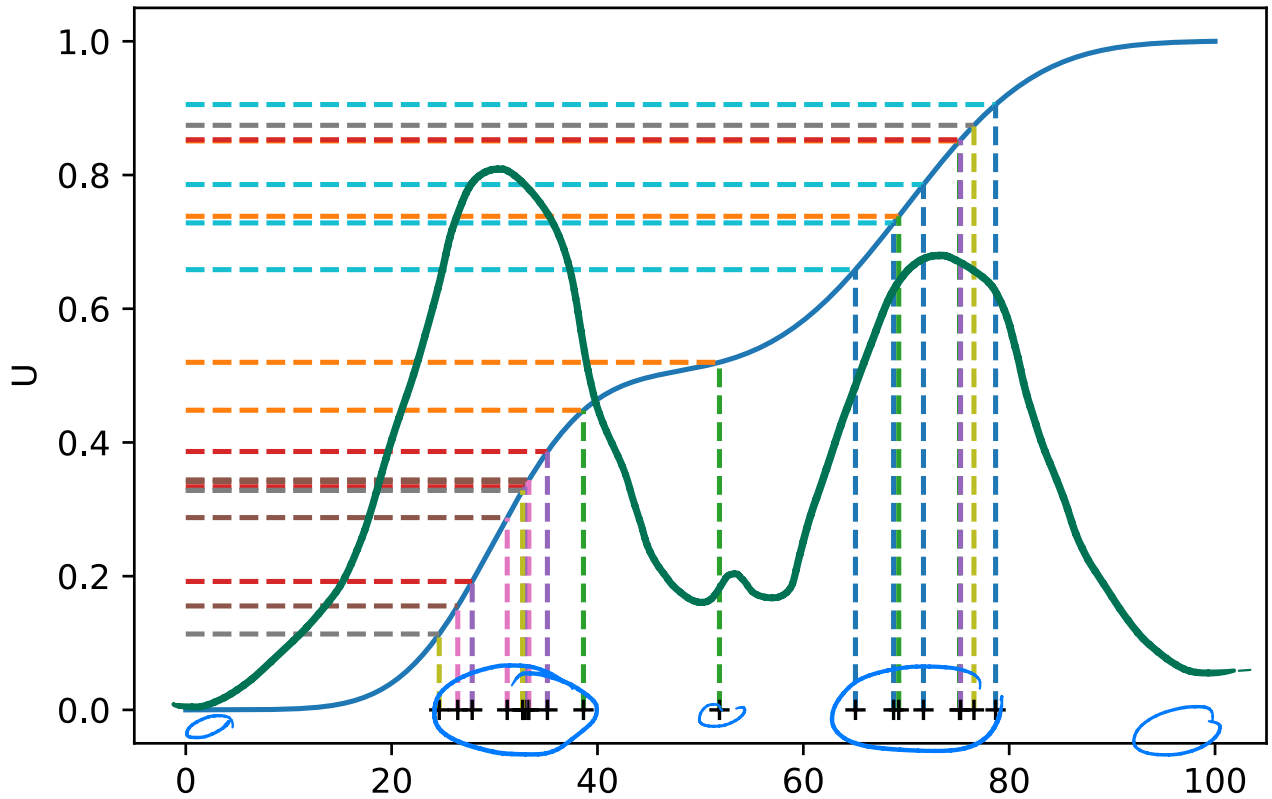


Algorithme: Générer $U \sim Unif([0, 1])$ et rendre

$$F^{-1}(U) = \sup\{x \mid F(x) \leq U\}.$$

Méthode de la fonction inverse

Rappel, la fonction cumulatrice de distribution F est une fonction $F : \mathbb{R} \to [0, 1]$ telle que $F(x) = \mathbb{P}[X \leq x]$.



Algorithme: Générer $U \sim Unif([0, 1])$ et rendre

$$F^{-1}(U) = \sup\{x \mid F(x) \leq U\}.$$

Quiz

Qu'est-ce que l'algorithme suivant génère?

```
u = rand()
if u < 0.3:
    return ( 0 )
elif( u < 0.5):
    return ( 1 )
else:
    return ( 2 )
```

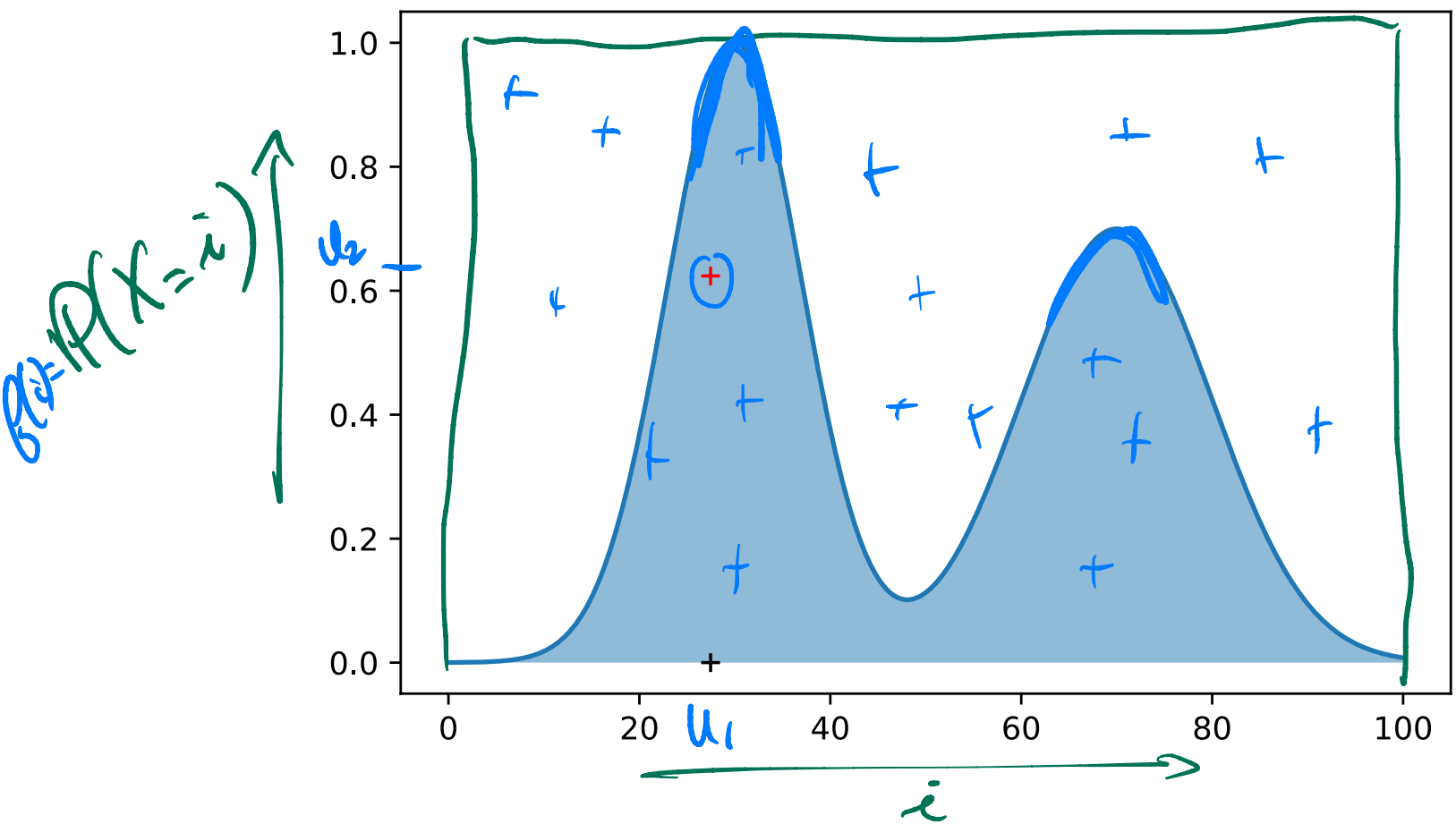
1. 0 avec probabilité 0.3, 1 avec probabilité 0.5 and 2 avec probabilité 1
2. 0 avec probabilité 0.3, 1 avec probabilité 0.2 and 2 avec probabilité 0.5
3. 0 avec probabilité 0.7, 1 avec probabilité 0.2 and 2 avec probabilité 0.1
4. Je ne sais pas.

Méthode du rejet

Soit X une variable aléatoire à valeur dans $\mathcal{X}$ et $f(x) = \mathbb{P}[X = x]$:

1. Générer U_1 sur $\mathcal{X}$ (uniformément).
2. Générer $U_2 \in [0, \max_x f(x)]$ (uniformément).
3. Si $U_2 \leq f(U_1)$, retourner U_1. Sinon revenir à l'étape 1.

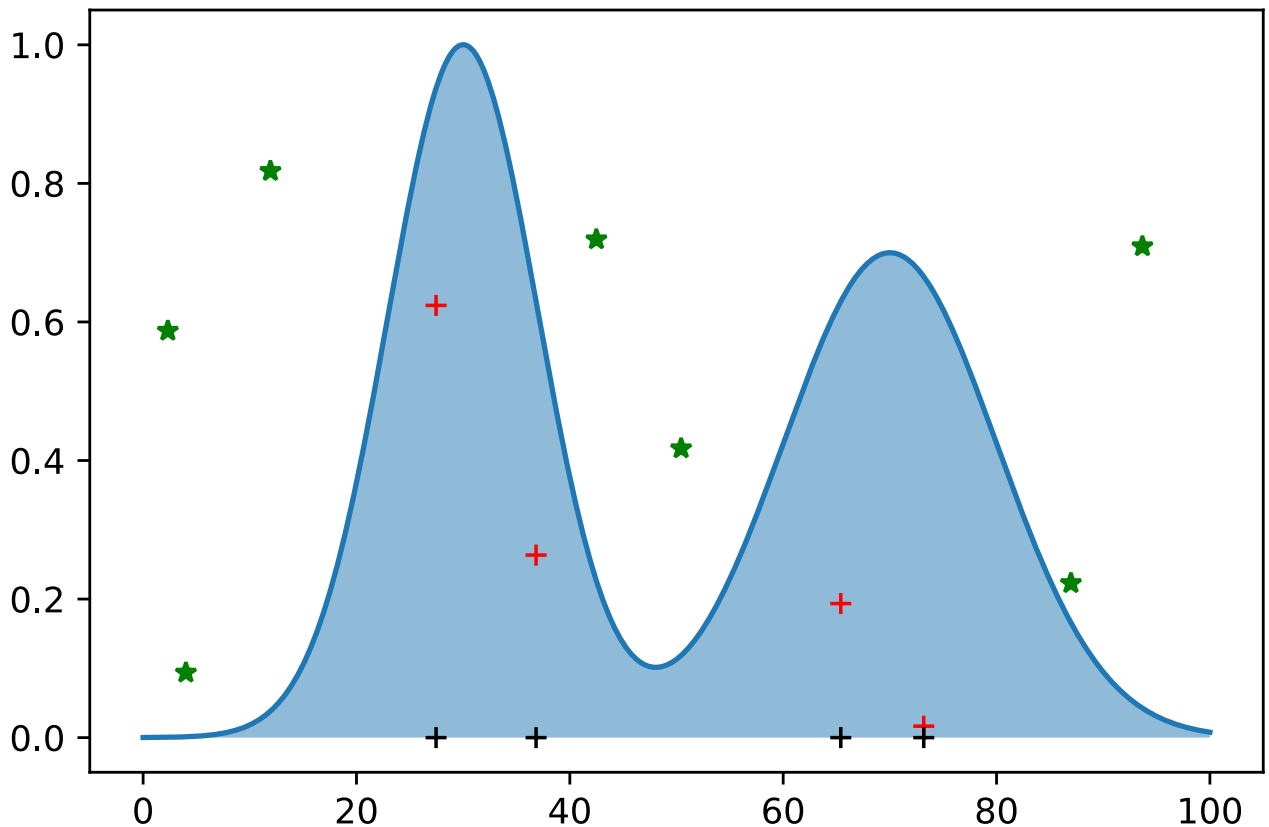
```
while True:
    u1 = rand()
    u2 = rand()
    if u2 ≤ f(u1)
        return u1
```



Méthode du rejet

Soit X une variable aléatoire à valeur dans $\mathcal{X}$ et $f(x) = \mathbb{P}[X = x]$:

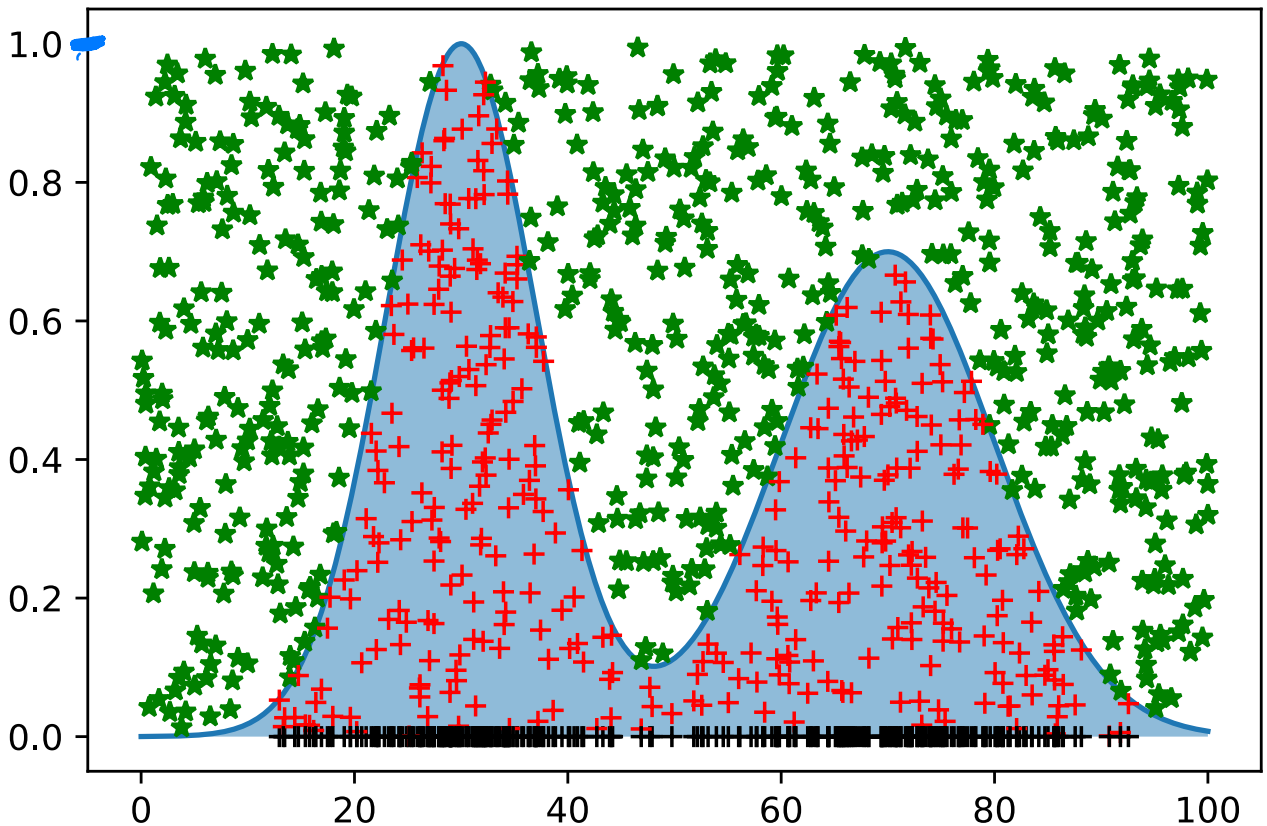
1. Générer U_1 sur $\mathcal{X}$ (uniformément).
2. Générer $U_2 \in [0, \max_x f(x)]$ (uniformément).
3. Si $U_2 \leq f(U_1)$, retourner U_1. Sinon revenir à l'étape 1.

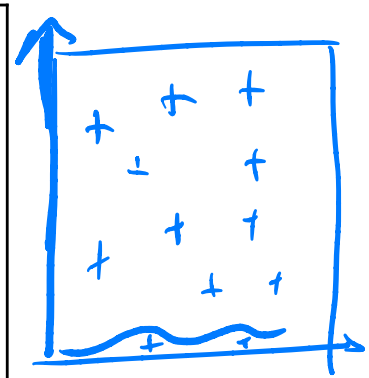


Méthode du rejet

Soit X une variable aléatoire à valeur dans $\mathcal{X}$ et $f(x) = \mathbb{P}[X = x]$:

1. Générer U_1 sur $\mathcal{X}$ (uniformément).
2. Générer $U_2 \in [0, \max_x f(x)]$ (uniformément). $U_2 \in \mathrm{Unif}[0,1]$
3. Si $U_2 \leq f(U_1)$, retourner U_1. Sinon revenir à l'étape 1.





Quiz

$0.9^2 + 0.9^2 = 2 \times 0.81 = 1.62 > 1$

Exemple 2. On génère un couple (X, Y) tiré uniformément sur le cercle centré en 0 et de rayon 1.

Les variables X et Y sont elles indépendantes?

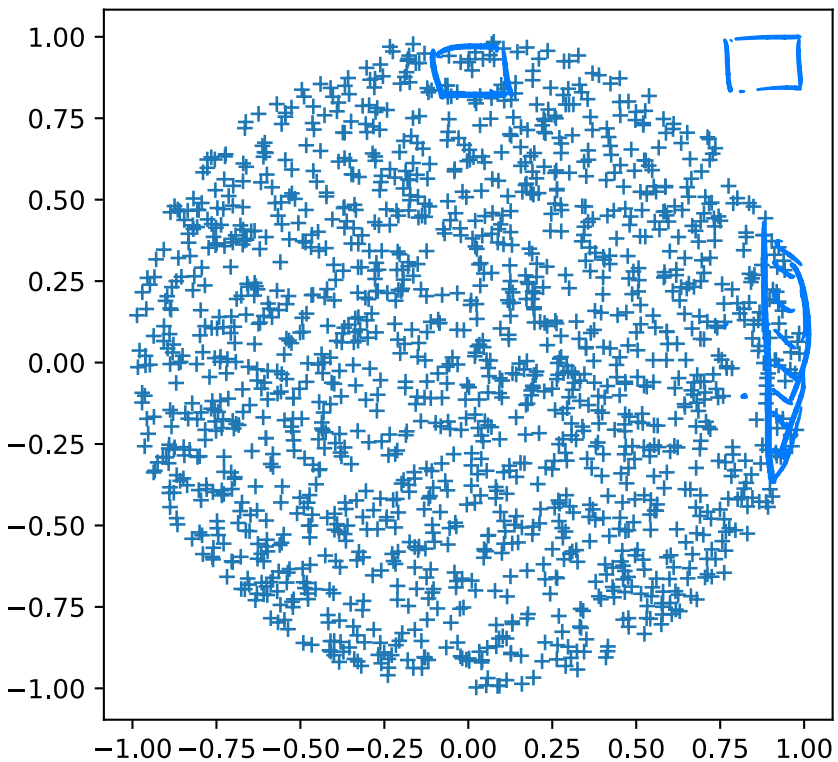
1. Oui
2. Non
3. Je ne sais pas.

$\mathbb{P}(X \geq 0.9 \text{ et } Y \geq 0.9) = 0$

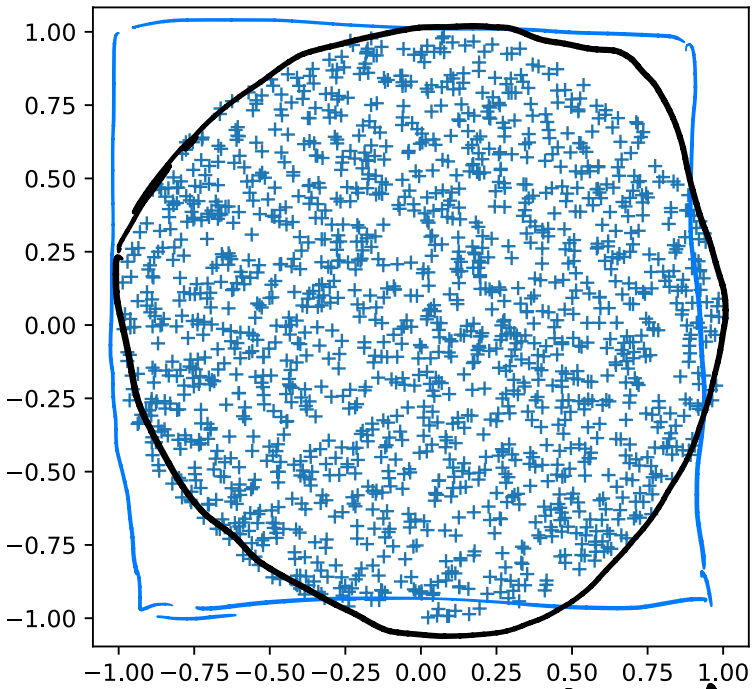
$\mathbb{P}(X \geq 0.9) > 0$

$\mathbb{P}(Y \geq 0.9) > 0$

donc $\mathbb{P}(X \geq 0.9 \text{ et } Y \geq 0.9) \neq \mathbb{P}(X \geq 0.9)\mathbb{P}(Y \geq 0.9)$



Application du rejet: comment générer une variable sur un cercle



Combien de tirage faut-il faire en moyenne?

Rep : $\frac{4}{\pi} \leq 1.33$

Faux

$r \sim \text{Unif}[0;1]$

$\theta \sim \text{Unif}[0;2\pi]$

rendre $r\cos\theta, r\sin\theta$.

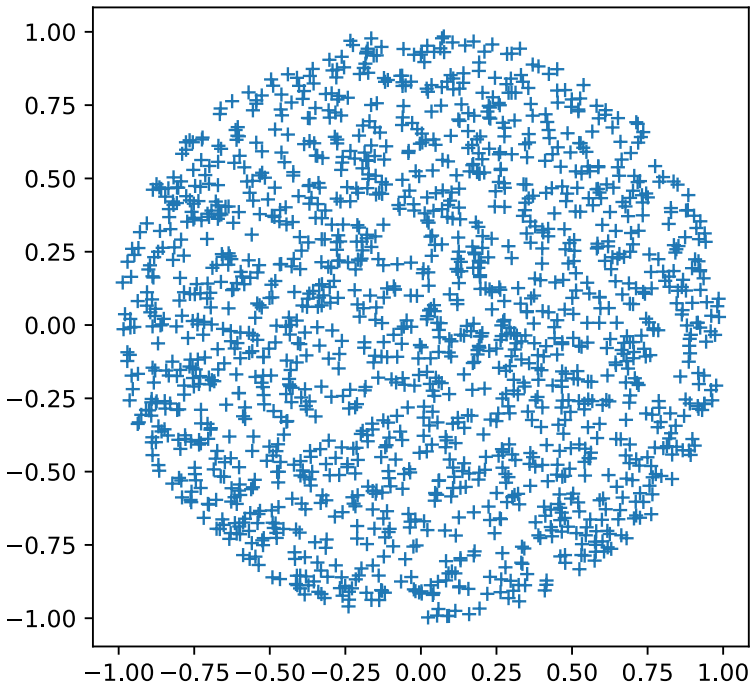
↳ non uniforme sur le cercle

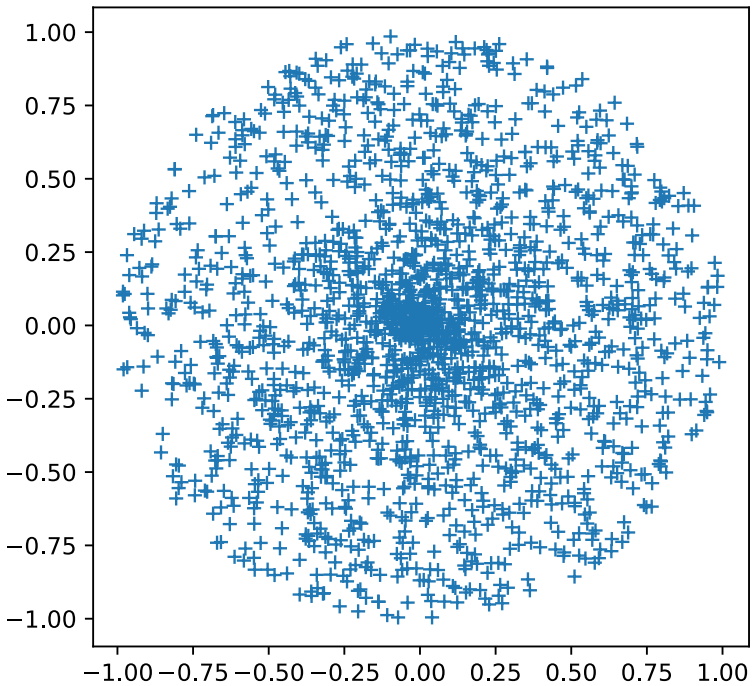
Rejet :

```
while True:
    opérer X,Y dans le carré
    si X,Y ∈ cercle:
        rendre (X,Y)
```

$\mathbb{P}(X,Y \in \text{cercle}) = \frac{\pi}{4}$

Application du rejet: comment générer une variable sur un cercle





Rejet:
Répéter

- Générer X, Y
- Tant que $X, Y \notin \text{Circle}$

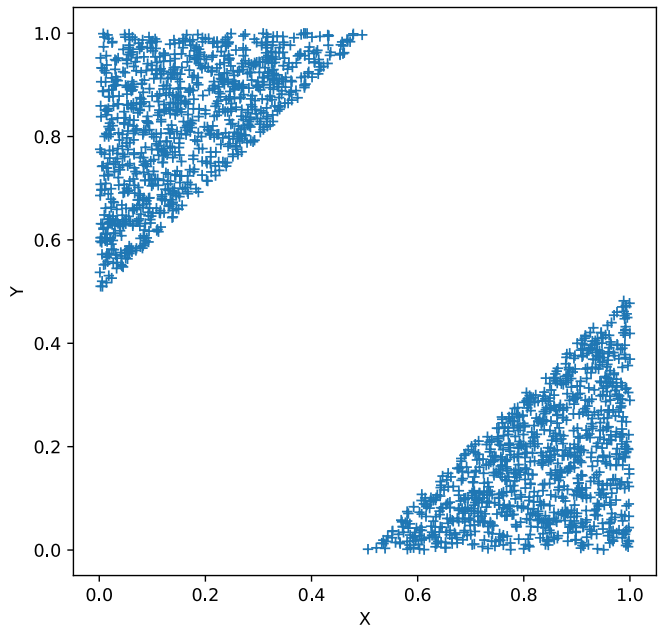
Méthode naïve (fausse):
Générer le rayon et l'angle uniformément.

Quiz

On veut générer un couple de nombre (X, Y) uniformément dans la zone bleu (*i.e.* telle que $|X - Y| > 0.5$).
L'algorithme suivante est-il correct?

- Générer $X \sim$Unif$[0, 1]$
- Faire:
- Générer $Y \sim$Unif$[0, 1]$
- Tant que $|X - Y| \leq 0.5$

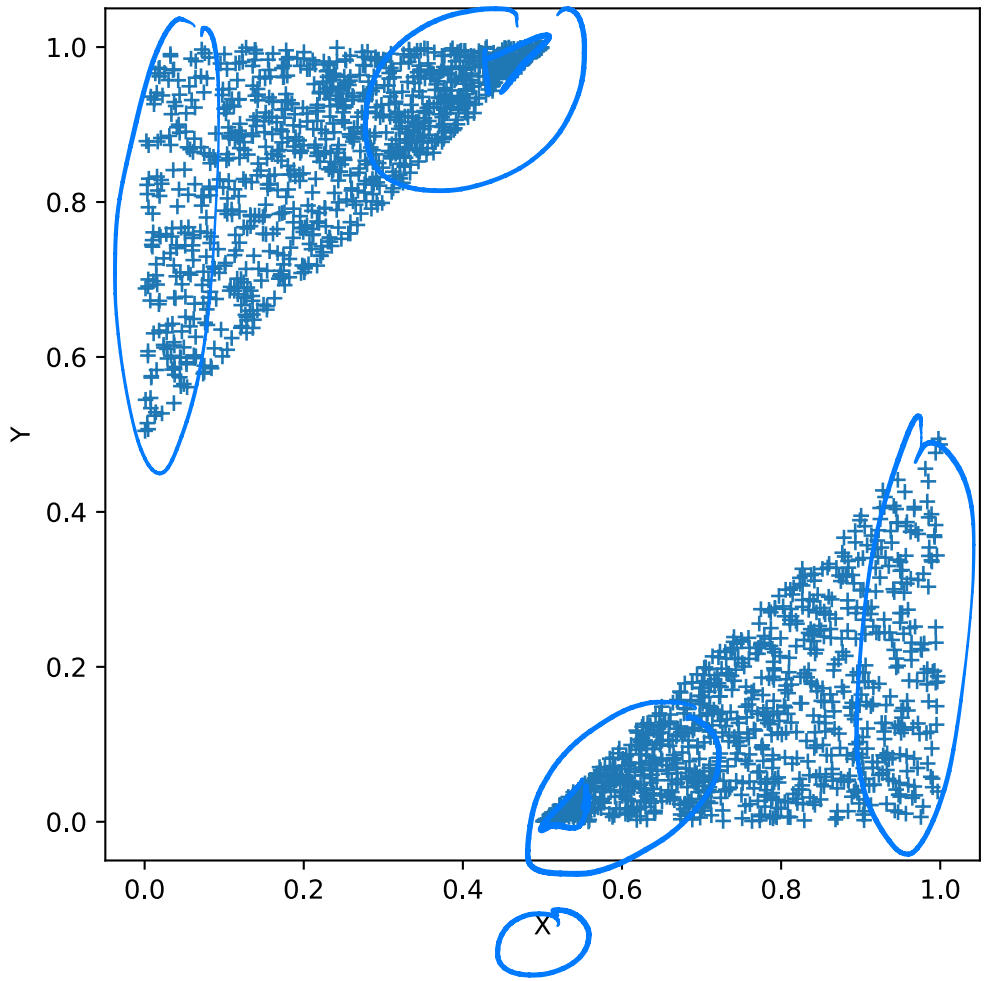
1. Oui
2. Non
3. Je ne sais pas.



Réponse: NON

Sur une simulation (10000 tirage):

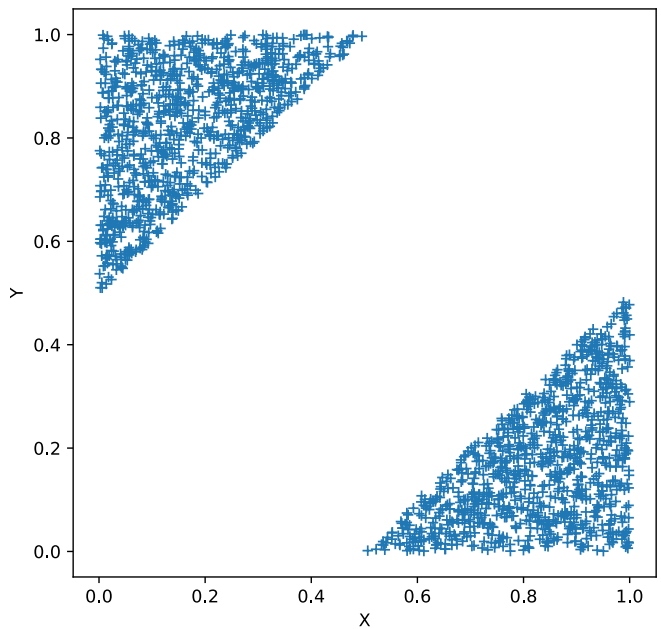
X ~ Unif [0;1]
répéter :
Y ~ Unif [0;1]
tant que
|X-Y| ≤ 0.5

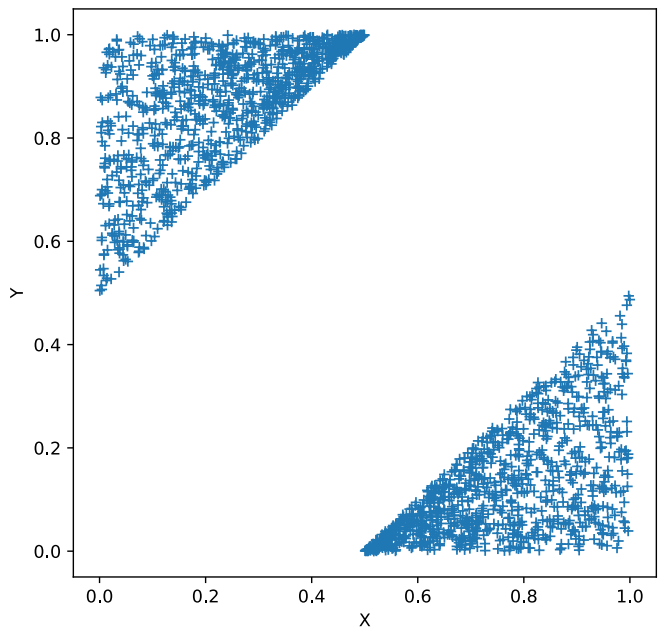


Ce n'est pas uniforme.

Pour cet exemple

Algorithme Correct	Algorithme incorrect
• Faire:	• Générer $X \sim \text{Unif}[0,1]$
• Générer $X \sim \text{Unif}[0,1]$	• Faire:
• Générer $Y \sim \text{Unif}[0,1]$	• Générer $Y \sim \text{Unif}[0,1]$
• Tant que $\lvert X - Y \rvert \leq 0.5$	• Tant que $\lvert X - Y \rvert \leq 0.5$





A retenir

- Attention, tous les générateurs pseudo-aléatoires ne sont pas bons.
- Les méthodes de rejet marchent bien.
- Il y a beaucoup de bibliothèques pour générer des nombres aléatoires: ne réinventez pas la roue.

```python
import numpy.random as rd
rd.exponential(2)              # Distribution exponentielle
rd.poisson(3)                  # Loi de poisson
rd.choice(3,p=[0.3,0.2,0.5]) # Rend 0 avec proba 0.3, 1 avec proba 0.2, 2 avec proba 0.5
```

https://docs.scipy.org/doc/numpy-1.14.0/reference/routines.random.html

Des librairies existent aussi en C, java, R, ...

Table des matières

Les algorithmes randomisés sont nombreux (et variés)

- Algorithmes "Las Vegas" : Complexité probabiliste
  - Quicksort a une **complexité moyenne** de $O(n \log n)$.
  - Les tables de hachages sont très efficaces en pratique.
- Algorithmes "Monte Carlo" : Résultat probabiliste
  - Miller-Rabin
  - Coupe minimale
- Autres:
  - Algorithmes distribués
  - Apprentissage en ligne

Table des matières

Exemple d'algorithmes randomisés : **Quicksort**

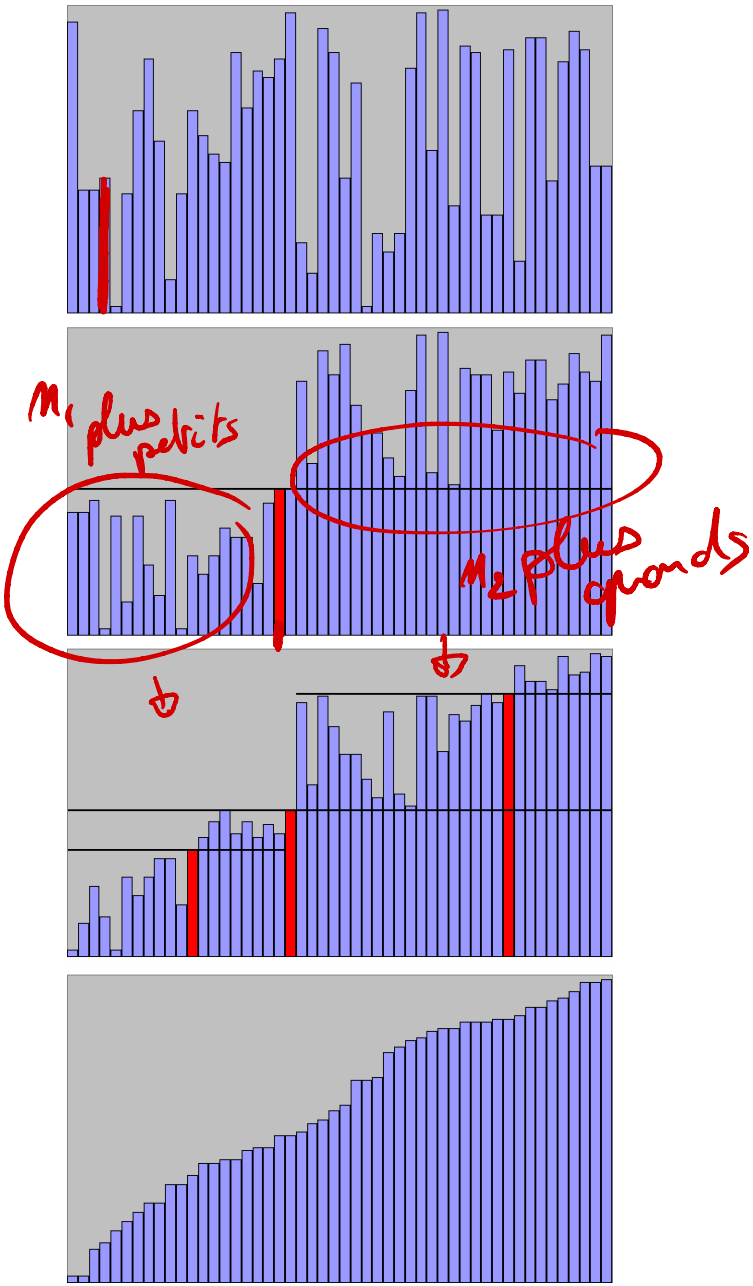
- Choisir un pivot (aléatoirement)
- Partitionner $O(n)$
- Recommencer récursivement
  - Dans les deux tas (pour quicksort)
  - Dans un des deux (pour quickselect)

complexité de pire cas en valeur de taille n

$$C(n) = O(n) + C(n_1) + C(n_2)$$

Pire cas : $0 = n_1$ ou $n_2 = 0$

$$C(n) = O(n) + C(n-1) = O(n^2)$$

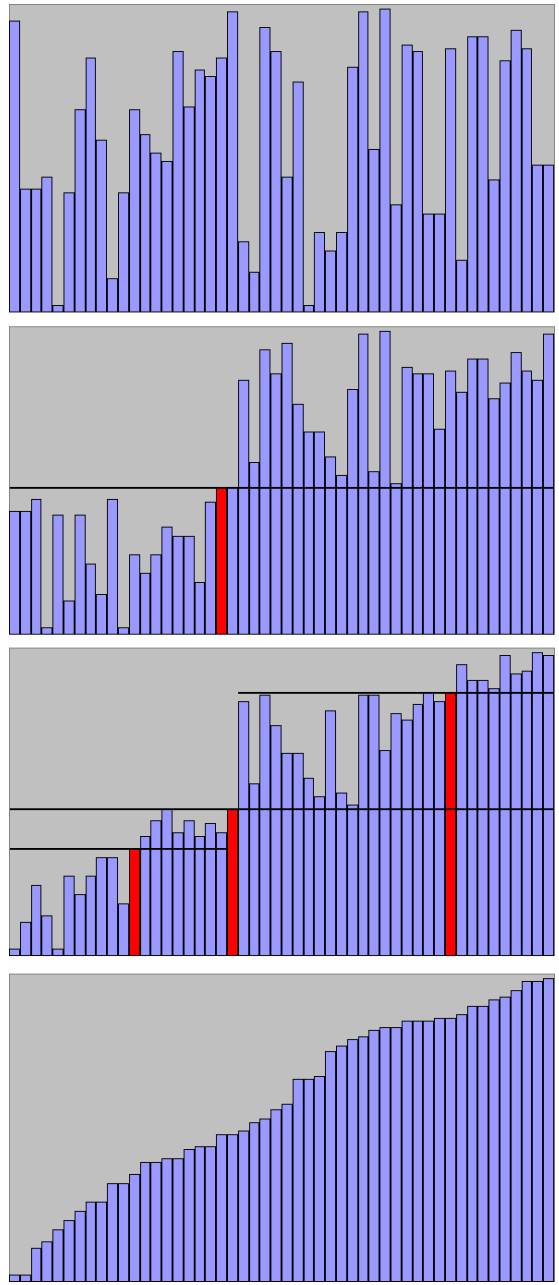


Exemple d'algorithmes randomisés : **Quicksort**

- Choisir un pivot (aléatoirement)
- Partitionner
- Recommencer récursivement
  - Dans les deux tas (pour quicksort)
  - Dans un des deux (pour quickselect)

Complexité:

- Partitionner $O(n)$
- Pire cas : $P_n : O(n^2)$.
- En moyenne : $O(n \log n)$.



Analyse de complexité: quicksort et quickselect

$$C(n) = O(n) + \sum_{i=0}^{n-1} \underbrace{\mathbb{P}(n_1 = i)}_{= \frac{1}{n}} \left[C(n_1) + C(n_2) \right]$$

$n_2 = n - n_1 - 1$ (↑ tableau des plus petits, ↑ pivot)

$$C(n) = O(n) + \frac{1}{n} \sum_{i=0}^{n-1} C(i) + C(n-i-1)$$

$$= O(n) + \frac{2}{n} \sum_{i=0}^{n-1} C(i)$$

récurrence sur $C(n)$

On peut montrer que $C(n) = O(n \log n)$

$(n+1)C(n+1) - n\,C(n)$

Quick-select (Tableau T, Entier i):

Rendre le $i^{ème}$ élément.

Choisir un pivot $T[j]$ (aléatoirement)

Partitionner T en deux tableaux :

Tpetit = éléments plus petits que $T[j]$

Tgrand = ——— grands que $T[j]$

n_1 := longueur de Tpetit.

si $i < n_1$:

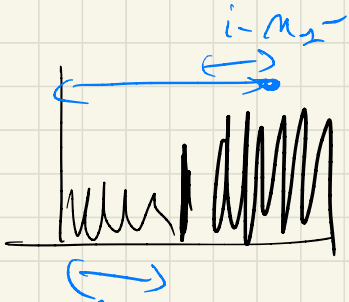
rendre Quickselect (Tpetit, i)

si $i = n_1$:

rendre $T[j]$

si $i > n_1$:

rendre Quickselect (Tgrand, $i - n_1 - 1$)



C_n = complexité moyenne de QuickSort

$$C_n = O(n) + \sum_{n_1} P(\text{pivot sort en place } n_1 \text{ et que } i < n_1)\, C_{n_1}$$

$$P(\text{pivot soit en } n_1 \text{ et que } i > n_1) \cdot C_{n-n_1-1}$$

$$= O(n) + \sum_{n_1} P(\text{pivot sort en place } n_1) \cdot \max(C_{n_1}, C_{n-n_1-1})$$

$$= O(n) + \frac{1}{n} \sum_{i=0}^{n-1} \max(C_i, C_{n-i-1})$$

$$= \underbrace{O(n)} + \frac{2}{n} \sum_{i=n/2}^{n-1} C_i$$

$$C_n = n + \frac{2}{n} \sum_{i=n/2}^{n-1} C_i$$

si $C_i \leq a \cdot i$ pour $a = 4$

$$n + \frac{2}{n} \sum_{i=n/2}^{n-1} a \cdot i$$

$$\leq n + \frac{2a}{n}\left(n^2 - \frac{n^2}{8}\right) = n + \frac{3an}{4}$$

$$\sum_{i=n/2}^{n-1} i = \sum_{i=1}^{n-1} i - \sum_{i=1}^{n/2-1} i$$

$$= \frac{n(n-1)}{2} - \frac{\frac{n}{2}\left(\frac{n}{2}-1\right)}{2}$$

$$\leq \frac{n^2}{2} - \frac{n^2}{8}$$

$\leq 4n$

Analyse de complexité: quicksort et quickselect

- Quicksort : En moyenne :

$$\begin{aligned} C_n &= n + \frac{1}{n}\sum_{i=0}^{n-1} C_i + C_{n-i-1} \\ &= n + \frac{2}{n}\sum_{i=0}^{n-1} C_i \\ &= O(n \log n) \end{aligned}$$

Analyse de complexité: quicksort et quickselect

- Quicksort : En moyenne :

$$\begin{aligned} C_n &= n + \frac{1}{n}\sum_{i=0}^{n-1} C_i + C_{n-i-1} \\ &= n + \frac{2}{n}\sum_{i=0}^{n-1} C_i \\ &= O(n \log n) \end{aligned}$$

- Quickselect :

$$\begin{aligned} C_n &= n + \frac{1}{n}\sum_{i=0}^{n-1} \max(C_i, C_{n-i-1}) \\ &= n + \frac{2}{n}\sum_{i=n/2}^{n-1} C_i \\ &= O(n) \end{aligned}$$

L'aléatoire intervient aussi dans les structures de donnée

- Table de hachage : on insère $n = \alpha m$ éléments dans une table de m éléments.
  - L'insertion se fait en temps $O(\alpha)$.
  - La recherche se fait en temps $O(\alpha)$.
  - Après n insertions, $\mathbb{P}$[une case a k éléments] $\approx \alpha^k e^{-\alpha}/k!$.

L'aléatoire intervient aussi dans les structures de donnée

- Table de hachage : on insère $n = \alpha m$ éléments dans une table de m éléments.
  - L'insertion se fait en temps $O(\alpha)$.
  - La recherche se fait en temps $O(\alpha)$.
  - Après n insertions, $\mathbb{P}[\text{une case a } k \text{ éléments}] \approx \alpha^k e^{-\alpha}/k!$.

- Soit X_n la hauteur d'un arbre binaire de recherche de taille n
  - On note $Y_n = \mathbb{E}\left[2^{X_n}\right]$. On a:

$$Y_n \leq 2\sum_{i=0}^{n-1} \frac{1}{n}(Y_i + Y_{n-i-1}) \leq 4\sum_{i=0}^{n-1} Y_i = O(n^3).$$

  - D'ou: $\mathbb{E}[X_n] \leq \log_2 \mathbb{E}\left[2^{X_n}\right] \leq 3\log_2 n + O(1)$ (Jensen)

Table des matières

Exemple d'algorithmes randomisés: **Miller-Rabin**

Let $2^s d = n - 1$, avec d impair. Choisir $a \in \{1, \ldots, n-1\}$. Si

$$(a^d \neq 1 \mod n) \text{ et } (a^{2^r d} \neq -1 \mod n \text{ pour tout } 0 \leq r \leq s-1)$$

alors retourner "n n'est pas premier". Sinon, retourner "n est premier".

Exemple d'algorithmes randomisés: **Miller-Rabin**

Let $2^s d = n - 1$, avec d impair. Choisir $a \in \{1, \ldots, n-1\}$. Si

$$(a^d \neq 1 \mod n) \text{ et } (a^{2^r d} \neq -1 \mod n \text{ pour tout } 0 \leq r \leq s-1)$$

alors retourner "n n'est pas premier". Sinon, retourner "n est premier".

Analyse:

- Si n est premier, aucun a ne vérifie cette propriété
- Si n est composé, au moins $3/4$ des a vérifient cette propriété.

Exemple d'algorithmes randomisés: **Miller-Rabin**

Let $2^s d = n - 1$, avec d impair. Choisir $a \in \{1, \ldots, n-1\}$. Si

$$(a^d \neq 1 \mod n) \text{ et } (a^{2^r d} \neq -1 \mod n \text{ pour tout } 0 \leq r \leq s-1)$$

alors retourner "n n'est pas premier". Sinon, retourner "n est premier".

Analyse:

- Si n est premier, aucun a ne vérifie cette propriété
- Si n est composé, au moins $3/4$ des a vérifient cette propriété.

Note : Le *test de fermat* $a^{n-1} = 1 \mod n$ n'est qu'un test heuristique car $2^{340} = 1 \mod 341$ (nombre de Carmichael)

Tests de primalité

- Miller-Rabin (1976 - 80) : $\texttt{PRIMES} \in \text{co} - \text{RP}$.
  Voir aussi : Solovay–Strassen (1977)
- Adleman–Huang algorithm (1992) : $\texttt{PRIMES} \in \text{RP}$

On a donc $\texttt{PRIMES} \in \text{ZPP}$.

- AKS primality test (2002) : $\texttt{PRIMES} \in \text{P}$.

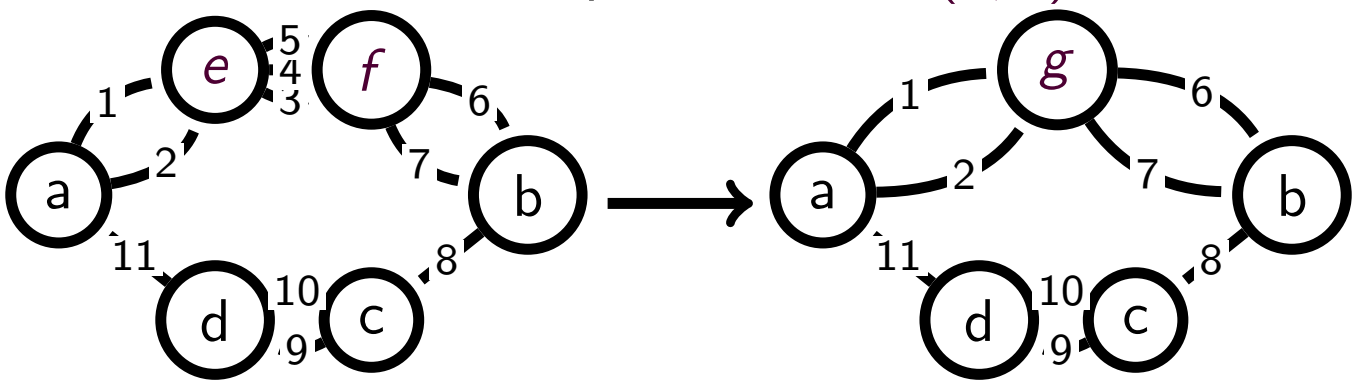
Coupe de taille minimale (algorithme de Krager)

Une coupe est un ensemble d'arêtes telles que si on les enlèves, le graphe est déconnecté.

Algorithme de Krager:

- Répéter $n-2$ fois: Choisir une arête uniformément dans le multigraphe G et "conctracter le graphe" par cette arête.

Contraction par une arête (e, f).



Analyse de l'algorithme de Krager : comment garantir l'erreur

Soit k la taille de la coupe minimale et $C_{\min}$ une coupe min. Montrer que:

- Le graphe a au moins $kn/2$ arêtes (indices: sommets isolés)

Analyse de l'algorithme de Krager : comment garantir l'erreur

Soit k la taille de la coupe minimale et $C_{\min}$ une coupe min. Montrer que:

- Le graphe a au moins $kn/2$ arêtes (indices: sommets isolés)
- Au premier tirage, on tombe sur une arête de $C_{\min}$ avec probabilité $\leq k/(kn/2) = 2/n$.
- À la $i$ème itération, le graphe a i arêtes et on tombe sur une arête de $C_{\min}$ avec probabilité $\leq 2/(n-i+1)$

Analyse de l'algorithme de Krager : comment garantir l'erreur

Soit k la taille de la coupe minimale et $C_{\min}$ une coupe min. Montrer que:

- Le graphe a au moins $kn/2$ arêtes (indices: sommets isolés)
- Au premier tirage, on tombe sur une arête de $C_{\min}$ avec probabilité $\leq k/(kn/2) = 2/n$.
- À la $i$ème itération, le graphe a i arêtes et on tombe sur une arête de $C_{\min}$ avec probabilité $\leq 2/(n-i+1)$
- D'où:

$$\mathbb{P}\left[\text{jamais choisir } C_{\min}\right] \geq 2/n^2.$$

- Comment garantir une probabilité d'erreur ε?

Analyse de l'algorithme de Krager : comment garantir l'erreur

Soit k la taille de la coupe minimale et $C_{\min}$ une coupe min. Montrer que:

- Le graphe a au moins $kn/2$ arêtes (indices: sommets isolés)
- Au premier tirage, on tombe sur une arête de $C_{\min}$ avec probabilité $\leq k/(kn/2) = 2/n$.
- À la $i$ème itération, le graphe a i arêtes et on tombe sur une arête de $C_{\min}$ avec probabilité $\leq 2/(n-i+1)$
- D'où:

$$\mathbb{P}\left[\text{jamais choisir } C_{\min}\right] \geq 2/n^2.$$

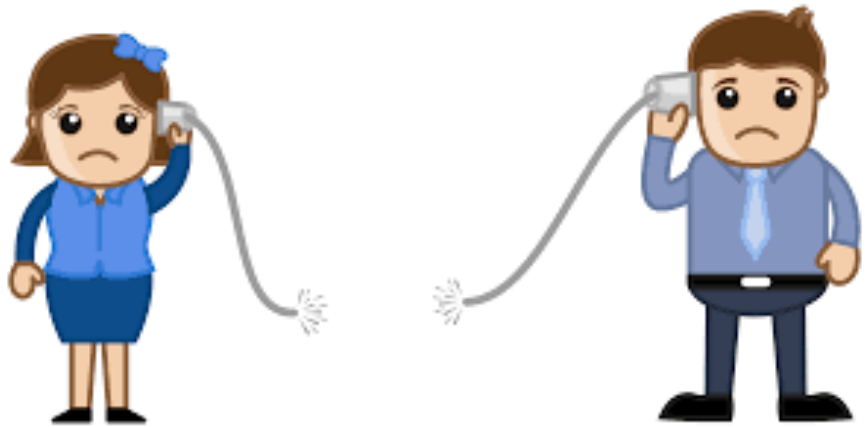
- Comment garantir une probabilité d'erreur ε?
  - Solution: répéter l'algorithme $Kn^2/2$ fois donne une erreur

$$(1-2/n^2)^{Kn^2/2} \leq e^{-K}$$

Table des matières

(Dé)synchronisation

Problème: comment décider qui parle?



Algorithmes (Slotted Aloha)

- N machines veulent communiquer sur un cannal.
- Le temps discret: $t \in \{0, 1, 2, \dots\}$.
- Au temps t, chaque machine transmet avec probabilité p.
- Si deux machines transmettent en même temps, le paquet est perdu.

Question: quel est le meilleur p?

Analyse de "Slotted Aloha"

Apprentissage en ligne