Centre Universitaire de AbdElhafid Boussouf, Mila
2ème Année Master Intelligence artificielle et ses applications
Année universitaire : 2023/2024
Matière: **Modélisation et simulation**

CHAPITRE IV:
SIMULATION À ÉVÈNEMENTS DISCRETS

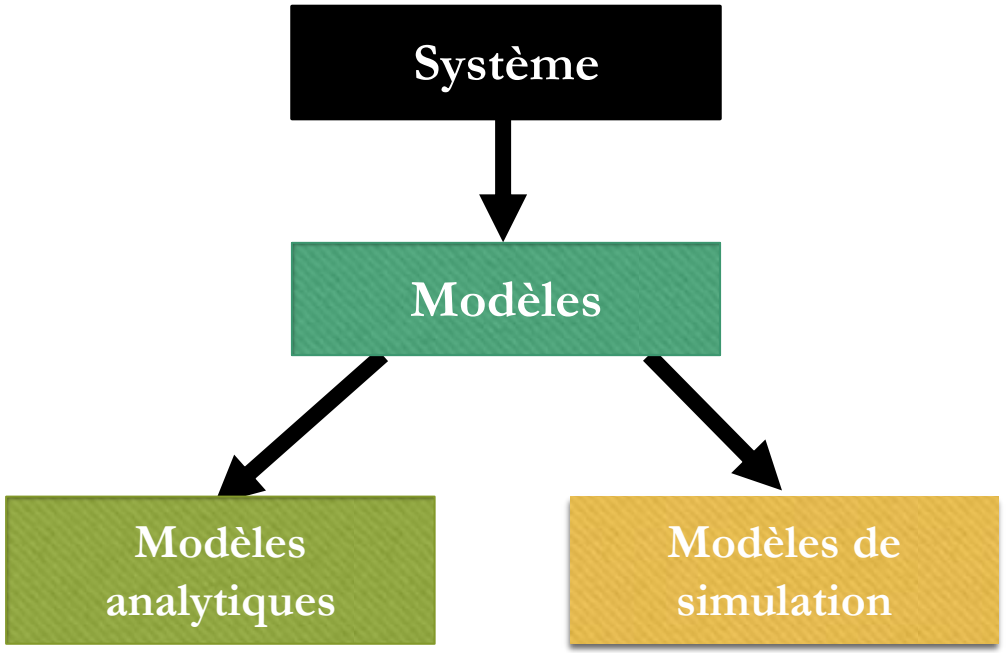
Responsable de la matière: DR. SADEK BENHAMMADA

1. Introduction: Limites des modèles analytiques des files d'attente

- Les modèles analytiques présentés dans le chapitre III sont basés sur les hypothèses:
  - Les arrivées des clients sont distribués selon une loi de Poisson,
  - Les durées de service sont distribué selon une loi exponentielle
- Ces hypothèses ne sont pas toujours valables dans les systèmes du monde réel, ce qui peut conduire à des prédictions inexactes du modèle.
- Les modèles analytiques des files d'attente s'applique particulièrement aux systèmes qui sont dans un état stable, ce qui signifie que les taux d'arrivée et de service sont relativement constants:
  - Exemples de systèmes étudiés à l'aide de modèles analytiques :
    - Centres d'appels (call center)
    - Chaînes de production en usine (montage de véhicules, etc.)
    - Salles d'urgence des hôpitaux (à l'exception des périodes de pointe)
  - Exemples de systèmes avec des taux d'arrivée et/ou de service variables
    - Trafic routier: les taux d'arrivée de véhicules peuvent varier en fonction des heures de la journée, des jours de la semaine, etc.
    - Réseaux de télécommunications: les taux d'arrivée et de service de demandes de service peuvent varier en fonction des heures de la journée, des événements spécifiques, etc.

1. Introduction: Modèles analytiques vs. Modèles de simulation

- La simulation s'applique là où les modèles analytiques ne sont pas adaptés,
- La simulation peut être défini comme un processus de conception et de développement de modèles informatiques qui imitent sur un ordinateur l'évolution d'un système réel dans le temps.



1. Introduction: Modèles analytiques vs. Modèles de simulation

Caractéristique	Modèles analytiques	Modèles de simulation
Représentation	Équations mathématiques	Code informatique
Hypothèses	Simplifiées, idéalisées, irréalistes	Plus réalistes
Applicabilité	Limité aux systèmes avec des relations mathématiques bien définies	Peut être appliqué à une plus large gamme de systèmes : systèmes complexes, imprévisibles ou difficiles à décrire de manière analytique
Méthode de résolution	Méthodes analytiques	Simulations numériques
Solution	Les Méthodes analytiques peuvent fournir des solutions mathématiques exactes	La simulation numérique donne des solutions approximatives
Temps de calcul	Rapide, car les solutions sont obtenues de manière analytique	Peut être plus lent en raison de la simulation numérique
Avantages	Rapide, efficace et précis pour les systèmes simples	Polyvalents : peuvent être appliqués sur des systèmes simples et complexes
Inconvénients	Applicabilité limitée, peut nécessiter des hypothèses simplificatrices	Peut être coûteux en calcul, Ne donnent pas de solutions exactes

2. Simulation à événements discrets

- **La simulation à événements discrets (Discrete-event simulation)** est la modélisation d'un système tel qu'il évolue dans le temps par une représentation dans laquelle les **variables d'état** changent instantanément à des points de temps distincts.
- Ces points dans le temps sont ceux auxquels un **événement** se produit,

Bien que la simulation à événements discrets puisse théoriquement être réalisée à l'aide de calculs manuels, la quantité de données qui doivent être stockées et manipulées pour la plupart des systèmes du monde réel impose que les simulations à événements discrets soient effectuées sur un ordinateur.

2. Simulation à événements discrets

- Les éléments clés d'une simulation à événements discrets sont: les variables qui définissent **l'état du système**, les **événements**, et **l'horloge**.

1. **État du système**
   - Un ensemble de variables qui décrivent l'état du système (System state) à un instant t

   ***Exemple** : les variables d'état d'une file d'attente : Le nombre de clients dans le système, le nombre de clients en attente, les temps d'attentes des clients, l'état des serveurs (libre ou occupé), etc.*

2. **Evénement (Event):** est une occurrence instantanée qui peut **changer** l'**état** du système.

   ***Exemple** : Les événements qui changent l'état d'une file d'attente : « Arrivée d'un client » , « Début de service » , « Départ d'un client ».*

3. **Horloge (Clock):** Dans un modèle de simulation, la variable qui conserve la valeur actuelle du temps simulé l'**horloge** de simulation **(clock):**
   - L'horloge est représentée par une variable réelle (l'unité du temps n'est pas représentée pour simplifier le codage)
   - Généralement aucune relation entre le temps simulé et le temps nécessaire pour exécuter une simulation sur l'ordinateur.

2. Simulation à événements discrets

2.2. Mécanismes d'avancement du temps

- Trois principales approches pour faire avancer l'horloge de simulation :

1. **La simulation à temps discret (Discret time simulation)**

2. **La simulation à temps continu (Continuous time simulation)**

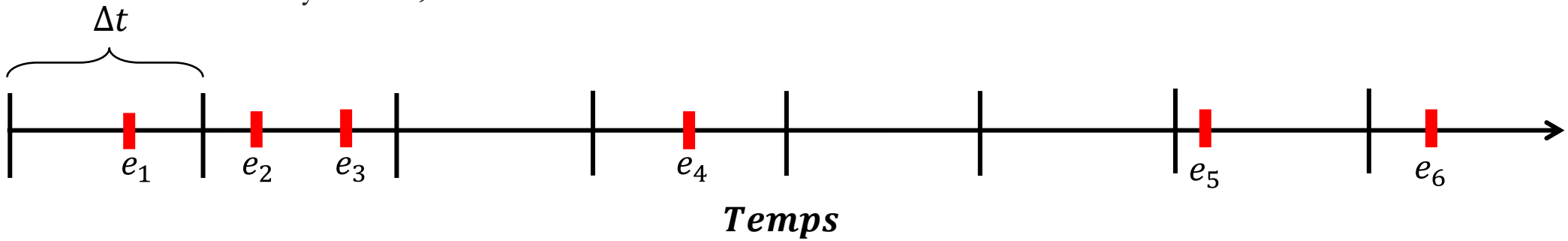
3. **La simulation à évènements discrets (Discret events simulation)**

2.2.1. Simulation à temps discrets

- Découpage du temps à des intervalles constants (Δt) (seconde, minutes, heures, jours, etc.),
- L'horloge de simulation avance par des pas de temps fixes Δt .
- À chaque pas de temps Δt, les calculs sont effectués pour mettre à jour l'état du système.

Cette approche est simple mais pose des problèmes si le système évolue de manière irrégulière.

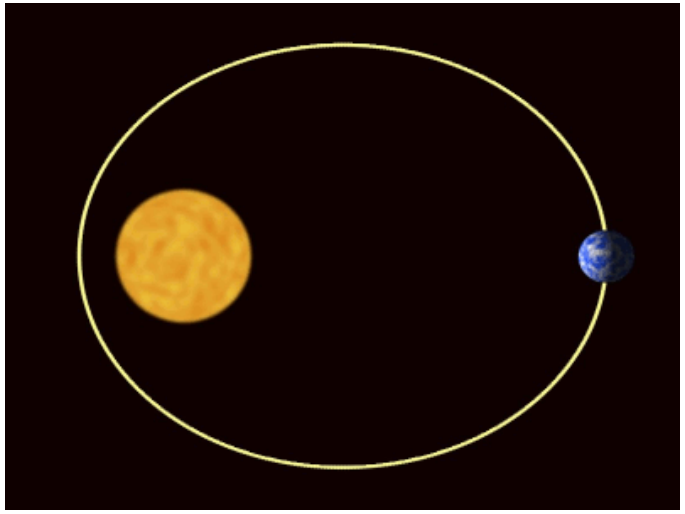
- Pendant de nombreux pas du temps, il n'y a pas de changement dans l'état du système et, par conséquent, de nombreux calculs sont inutiles.
- Difficulté de déterminer la valeur de Δt: il existe souvent une grande variation des durées d'activité au sein d'un système,

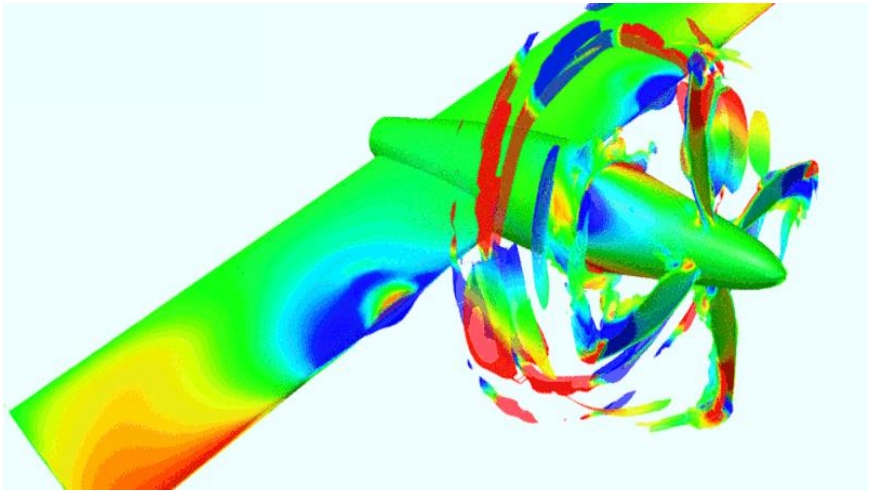


L'approche de simulation à événements discrets résout ces deux problèmes.

2.2.2. Simulation à temps continu

- La simulation continue (Continuous simulation) convient aux systèmes dans lesquels les états peuvent changer en permanence. Exemple: Mouvement d'une planète, Simulation de fluides (les liquides, les gaz et les plasmas).
- Le système est modélisé par une fonction continue ou un système de fonctions continues qui modélisent l'évolution des variables du système en fonction du temps,
- La simulation continue est souvent se formalise sous forme d'équations différentielles.
- La simulation informatique ne peut pas modéliser les changements continus d'état. Par conséquent, l'approche de simulation continue se rapproche du changement continu en prenant de petits pas de temps discrets (Δt).





Dans la Simulation à évènements discrets, l'horloge de simulation est initialisée à zéro, et les temps d'occurrence des événements futurs sont déterminés, puis:

1. L'horloge avance jusqu'au temps d'occurrence de l'événement le plus imminent (le prochaine évènement).
2. L'état du système est mis à jour, et les instants des événements futurs sont déterminées,
3. Le processus se répète pour chaque événement jusqu'à ce qu'une condition d'arrêt prédéfinie soit finalement satisfaite.

e_i: Evènement i

2. Simulation à événements discrets

2.3. Méthode en trois phases (Three-phase method)

- La méthode en trois phases est une approche de simulation à évènements discrets
- La méthode en trois phases classe les événements en deux types:

1. ***Événements B (Bound or Booked):***
    - Il s'agit des évènements qui se produisent certainement à un moment donné.
    - Les évènements B sont planifiés pour s'exécuter à un moment prévisible.
    - En général, les événements B concernent les arrivées ou l'achèvement d'une activité.
    - Exemple, dans un modèle de file d'attente: les **arrivées de clients**, et leurs **départs** (les **fins de services des clients**) sont des évènement B.
2. ***Événements C (Conditional):***
    - Les évènements C sont exécutées sous des conditions et ne peuvent pas être planifiées sur une liste d'événements puisqu'elles dépendent des états d'autres entités du système ou de la disponibilité des ressources système.
    - En général, les événements C se rapportent au début d'une activité
    - Par exemple : le début du service dans un système de file d'attente (qui dépend de la présence d'un client en attente et d'un serveur inactif)

Méthode en trois phases (three-phase method)

Début

Initialiser la simulation :

Etat initial du système

Horloge du système

Evénements initiaux

TantQue (La simulation n'est pas terminée)

Phase A: Déterminer le temps du prochain événement et avancer l'horloge à cet instant,

Phase B: Exécuter les événements B planifiés à cet instant,

Phase C: Exécuter les événements C dont les conditions sont vérifiées,

Fin TantQue

Retourner les résultats de simulation

Fin

2. Simulation à événements discrets

2.3. Méthode en trois phases (Three-phase method)

Exemple: Considérons une file d'attente avec un seul serveur. Le service s'arrete lorsque tous les clients sont servis.

1. Utilisez les listes ci-dessous des durées d'inter-arrivée et de service pour une simulation manuelle à l'aide de la méthode en trois phases.
2. Calculer: le temps moyen de séjour dans le système ($\bar{T}$), le temps d'attente moyen dans la file d'attente ($\bar{T}_Q$), le temps d'attente moyen dans la file d'attente ($\bar{T}_S$), le nombre moyen de clients dans le système ($\bar{N}$), le nombre moyen de clients en attente ($\bar{N}_Q$), et le taux d'utilisation du serveur ρ.

Client	Durées inter-arrivées	Temps d'arrivées	Durées de service	Début de service	Temps de départ
1	0,4		2		
2	1,2		0,7		
3	0,5		0,2		
4	1,7		1,1		
5	0,2		3,7		
6	1,6		0,6		
7	0,2		3,9		
8	1,4		4,1		
9	1,9		5,2		

Evènement	Temps	N	Q	Etat du serveur

2. Simulation à événements discrets

2.3. Méthode en trois phases (Three-phase method)

N: le nombre de clients dans le système

Q: le nombre de clients en attente

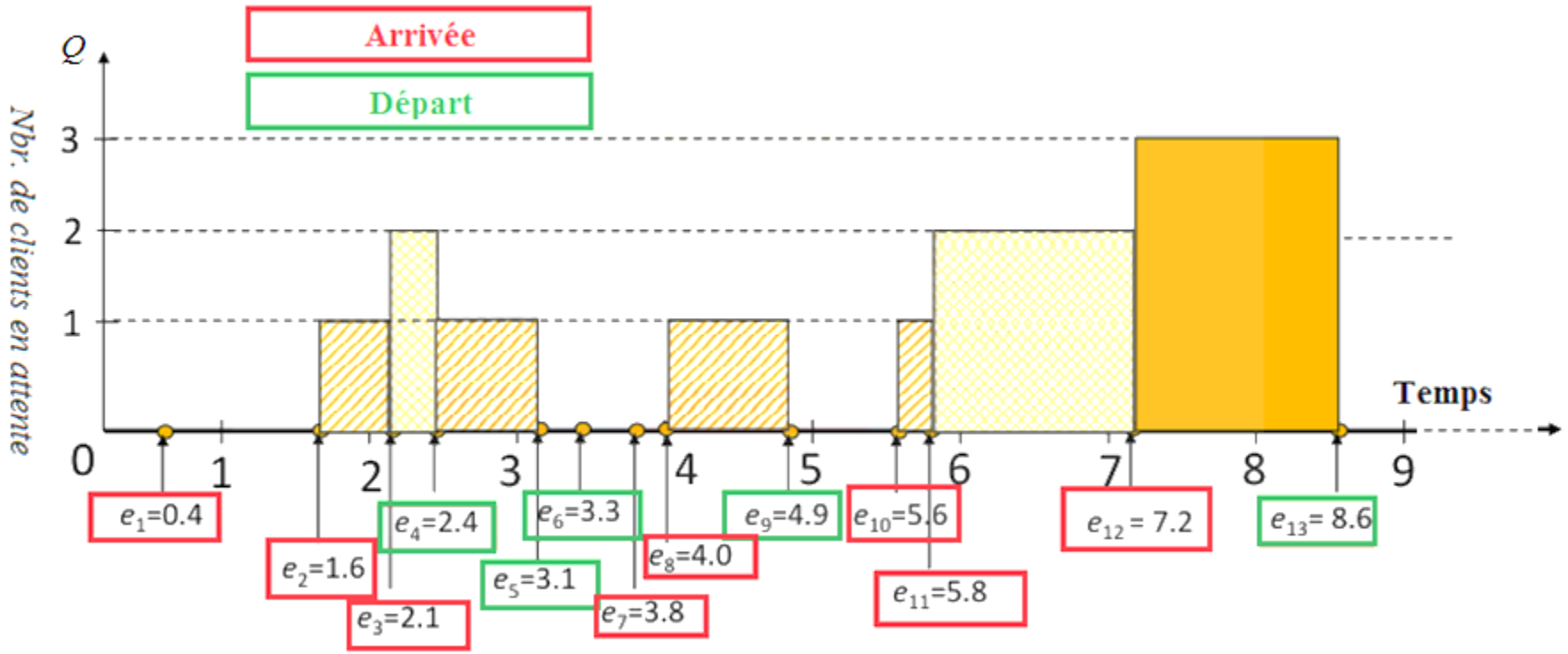
Client	Durées inter-arrivées	Temps d'arrivées	Durées de service	Début de service	Temps de départ
1	0,4	0,4	2	0,4	2,4
2	1,2	1,6	0,7	2,4	3,1
3	0,5	2,1	0,2	3,1	3,3
4	1,7	3,8	1,1	3,8	4,9
5	0,2	4	3,7	4,9	8,6
6	1,6	5,6	0,6	8,6	9,2
7	0,2	5,8	3,9	9,2	13,1
8	1,4	7,2	4,1	13,1	17,2
9	1,9	9,1	5,2	17,2	22,4

Evènement	Temps	N	Q	Etat du serveur
Arrivée	0,4	1	0	**Occupé**
Arrivée	1,6	2	1	**Occupé**
Arrivée	2,1	3	2	**Occupé**
Départ	2,4	2	1	**Occupé**
Départ	3,1	1	0	**Occupé**
Départ	3,3	0	0	**Libre**
Arrivée	3,8	1	0	**Occupé**
Arrivée	4	2	1	**Occupé**
Départ	4,9	1	0	**Occupé**
Arrivée	5,6	2	1	**Occupé**
Arrivée	5,8	3	2	**Occupé**
Arrivée	7,2	4	3	**Occupé**
Départ	8,6	3	2	**Occupé**
Arrivée	9,1	4	3	**Occupé**
Départ	9,2	3	2	**Occupé**
Départ	13,1	2	1	**Occupé**
Départ	17,2	1	0	**Occupé**
Départ	22,4	0	0	**Libre**

2. Simulation à événements discrets

2.3. Méthode en trois phases (Three-phase method)

Evènement	Temps	N	Q
Arrivée	0,4	1	0
Arrivée	1,6	2	1
Arrivée	2,1	3	2
Départ	2,4	2	1
Départ	3,1	1	0
Départ	3,3	0	0
Arrivée	3,8	1	0
Arrivée	4	2	1
Départ	4,9	1	0
Arrivée	5,6	2	1
Arrivée	5,8	3	2
Arrivée	7,2	4	3
Départ	8,6	3	2
Arrivée	9,1	4	3
Départ	9,2	3	2
Départ	13,1	2	1
Départ	17,2	1	0
Départ	22,4	0	0



Trajectoire du modèle de simulation (nombre de clients en attente en fonction du temps)

2.3. Méthode en trois phases (Three-phase method)

- Soient:
  - A_i le temps d'arrivée du client i
  - S_i le temps du début de service du client i, et
  - D_i le temps du départ du client i
  - C: $le\ nombre\ de\ clients\ servis$

- Le temps que le client i passe dans la file d'attente: $S_i - A_i$
- Le temps que le client i passe dans le système: $D_i - A_i$

Le temps moyen d'attente: $\bar{T}_Q = \frac{\sum_{i=1}^{C}(S_i - A_i)}{C}$

Le temps moyen de séjour dans le système: $\bar{T} = \frac{\sum_{i=1}^{C}(D_i - A_i)}{C}$

2.3. Méthode en trois phases (Three-phase method)

Soient:

- $t_0 = 0$: le temps de début de simulation
- $t_1, t_2, \ldots, t_{n-1}$: les temps des évènements d'arrivée et de départ
- t_n: le temps de la fin de la simulation
- N_k le nombre de clients en attente durant l'intervalle $[t_{k-1}, t_k]$
- Q_k le nombre de clients en attente durant l'intervalle $[t_{k-1}, t_k]$

- Le nombre moyen de clients dans le système $\bar{N}$:

$$\bar{N} = \sum_{k=0}^{n} \frac{N_k(t_k - t_{k-1})}{t_n}$$

- Le nombre moyen de clients en attente $\bar{N}_Q$:

$$\bar{N}_Q = \sum_{k=0}^{n} \frac{Q_k(t_k - t_{k-1})}{t_n}$$

- Le taux d'utilisation du serveur:

$$\rho = \sum_{N_k>0} \frac{(t_k - t_{k-1})}{t_n} = 1 - \sum_{N_k=0} \frac{(t_k - t_{k-1})}{t_n}$$

Le temps moyen d'attente:

$$\bar{T}_Q = \frac{\sum_{i=1}^{C}(S_i - A_i)}{C} = \frac{(0{,}4-0{,}4)+(2{,}4-1{,}6)+(3{,}1-2{,}1)+(3{,}8-3{,}8)+(4{,}9-4)++(8{,}6-5{,}6)+(9{,}2-5{,}8)+(13{,}1-7{,}2)+(17{,}2-9{,}1)}{9} = 2{,}57$$

Le temps moyen de séjour dans le système:

$$\bar{T} = \frac{\sum_{i=1}^{C}(S_i - A_i)}{C} = \frac{(2{,}4-0{,}4)+(3{,}1-1{,}6)+(3{,}3-2{,}1)+(4{,}9-3{,}8)+(8{,}6-4)++(9{,}2-5{,}6)+(13{,}1-5{,}8)+(17{,}2-7{,}2)+(22{,}4-9{,}1)}{9} = 4{,}96$$

Client	Durées inter-arrivées	Temps d'arrivées	Durées de service	Début de service	Temps de départ
1	0,4	0,4	2	0,4	2,4
2	1,2	1,6	0,7	2,4	3,1
3	0,5	2,1	0,2	3,1	3,3
4	1,7	3,8	1,1	3,8	4,9
5	0,2	4	3,7	4,9	8,6
6	1,6	5,6	0,6	8,6	9,2
7	0,2	5,8	3,9	9,2	13,1
8	1,4	7,2	4,1	13,1	17,2
9	1,9	9,1	5,2	17,2	22,4

2. Simulation à événements discrets

2.3. Méthode en trois phases (Three-phase method)

- Le nombre moyen de clients dans le système:

$$\bar{N} = \sum_{k=0}^{n} \frac{N_k(t_k - t_{k-1})}{t_n - t_0}$$

$$= \frac{1(1{,}6 - 0{,}4) + 2(2{,}1 - 1{,}6) + 3(2{,}4 - 2{,}1) + 2(3{,}1 - 2{,}4) + \cdots + 1(22{,}4 - 17{,}2)}{22{,}4} \approx 1{,}99$$

- Le nombre moyen de clients en attente:

$$\bar{N}_Q = \sum_{k=0}^{n} \frac{Q_k(t_k - t_{k-1})}{t_k - t_{k-1}}$$

$$= \frac{0(1{,}6 - 0{,}4) + 1(2{,}1 - 1{,}6) + 2(2{,}4 - 2{,}1) + 1(3{,}1 - 2{,}4) + \cdots + 1(17{,}2 - 13{,}2)}{22{,}4} \approx 1{,}03$$

Le taux d'utilisation du serveur:

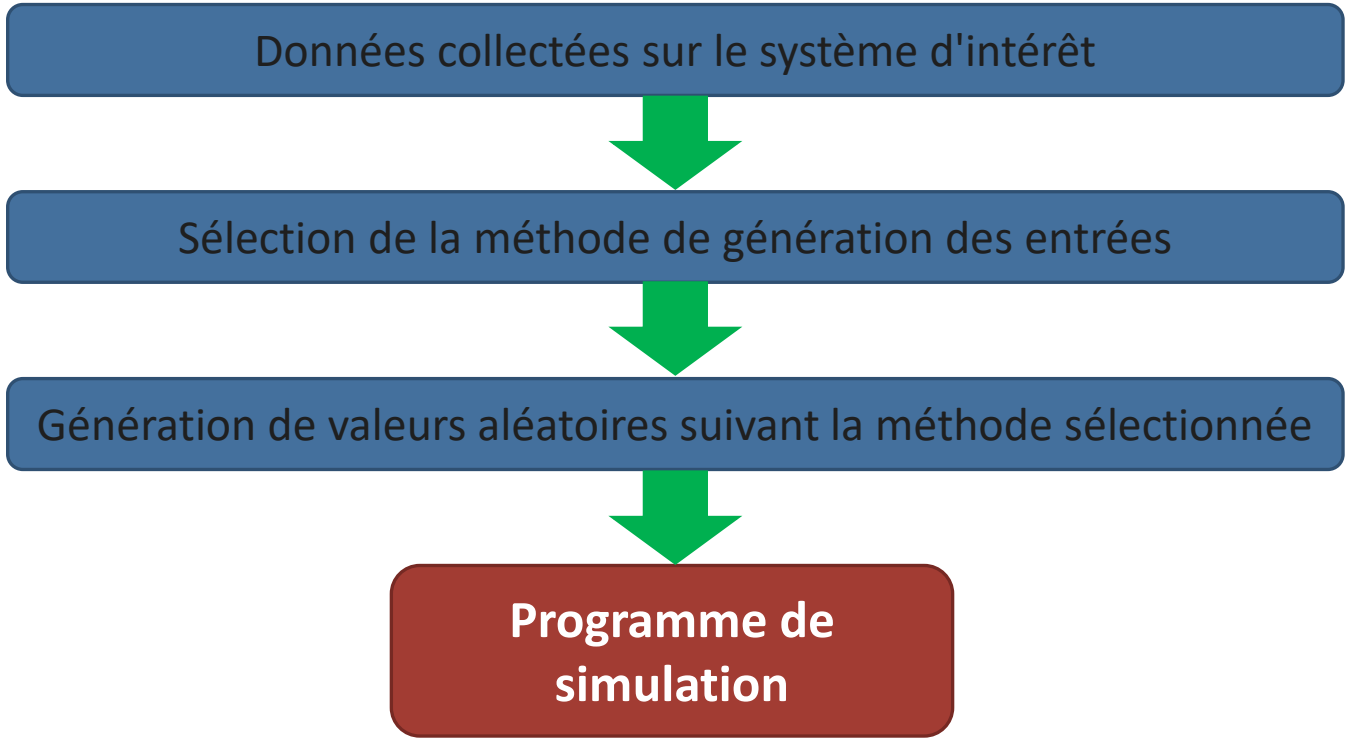
$$\rho = 1 - \sum_{N_k=0} \frac{(t_k - t_{k-1})}{t_n} = 1 - \frac{(0{,}4 - 0) + (3{,}8 - 3{,}3)}{22{,}4} = 1 - \frac{0{,}9}{22{,}4} \approx 0{,}96$$

Evenement	Temps	N	Q
Arrivée	0,4	1	0
Arrivée	1,6	2	1
Arrivée	2,1	3	2
Départ	2,4	2	1
Départ	3,1	1	0
Départ	3,3	0	0
Arrivée	3,8	1	0
Arrivée	4	2	1
Départ	4,9	1	0
Arrivée	5,6	2	1
Arrivée	5,8	3	2
Arrivée	7,2	4	3
Départ	8,6	3	2
Arrivée	9,1	4	3
Départ	9,2	3	2
Départ	13,1	2	1
Départ	17,2	1	0
Départ	22,4	0	0

3. Génération des entrées de simulation

3.1. Introduction

- Pour la simulation des évènements, tels que les arrivées et les départs de clients, nous avons besoins de générer des valeurs aléatoires suivant une méthode donnée.
- Une correspondance étroite entre les entrées des modèles de simulation et le véritable mécanisme probabiliste sous-jacent associé au système réel est nécessaire pour des analyses de simulation réussies.
- La question est de savoir comment modéliser un élément probabiliste tel que le processus d'arrivée ou les temps de service étant donné un ensemble de données collectées sur le système d'intérêt.



3. Génération des entrées de simulation

- Les méthodes de génération de variables aléatoires, transforment les nombres aléatoires de distribution uniforme $X \sim \mathcal{U}(0,1)$ pour générer des théoriques (normale, exponentielle, etc.) ou empiriques
- L'approche moderne consiste à utiliser un ordinateur pour générer successivement des nombres ***pseudo-aléatoires*** de distribution uniforme $X \sim \mathcal{U}(0,1)$,
- Les nombres **pseudo-aléatoires** constituent une séquence de valeurs qui, bien qu'elles soient générées de manière déterministe, ont toutes les apparences d'être des variables aléatoires indépendantes uniformes $X \sim \mathcal{U}(0,1)$.
- La plupart des langages de programmation permettent de générer des valeurs ***pseudo-aléatoires*** distribuées selon la loi uniforme.
- Différentes méthodes sont utilisées pour générer des nombres pseudo-aléatoires : ***Linear Congruential Generator , Feedback Shift Register Generators, Mersenne Twister, etc.***

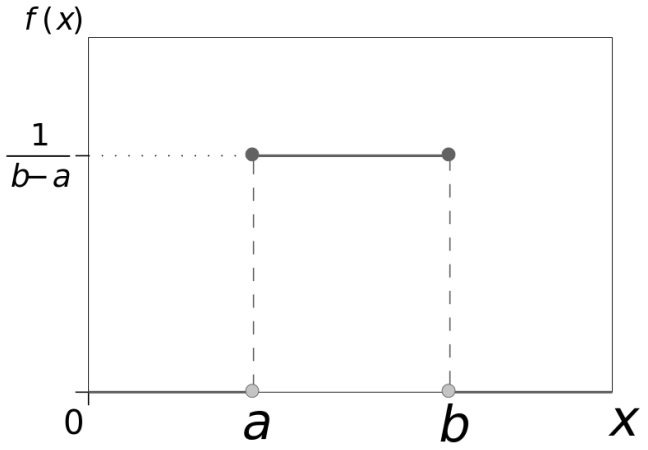
Rappel: La loi uniforme

- Soit X une variable aléatoire continue qui prend ses valeurs sur une intervalle $[a, b]$ où $a < b$. Si X suit un loi uniforme: $X \sim \mathcal{U}(a, b)$, s'il est uniformément distribuée sur l'intervalle $[a, b]$.

La fonction de densité est donnée par: $f(x) = \frac{1}{b-a}$

La fonction de répartition: $F(x) = P(X \leq x) = \frac{x-a}{b-a}$ $\quad a \leq x \leq b$

$$E(X) = \frac{a+b}{2}$$

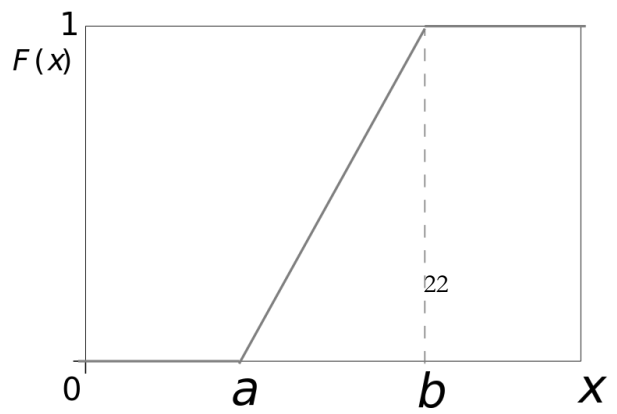


Soit U une variable aléatoire continue qui suit une la uniforme sur l'intervalle $[0,1]$: U $\sim \mathcal{U}(0,1)$

La fonction de densité de U est donnée par: $f(x) = 1$

La fonction de répartition de U : $F(x) = P(U \leq x) = x$

$$E(X) = \frac{1}{2}$$



3. Génération des entrées de simulation

3.2. Génération de nombres pseudo-aléatoires

3.2.2. Générateur congruentiel linéaire

- **Générateur congruentiel linéaire (*Linear Congruential Generator*)** est l'une des méthodes les plus appliquées pour générer des nombres pseudo-aléatoires,
- Le **générateur congruentiel linéaire** commence par une valeur initiale x_0, appelée **semence (seed en anglais)**, puis calcule de manière récursive les valeurs successives $x_n, n \geq 1$ par la formule:

$$x_n = (ax_{n-1} + c)\ mod\ m$$

  - $\boldsymbol{m},\ \boldsymbol{m > 0}$ est **le module**
  - $\boldsymbol{a},\ \boldsymbol{0 < a \leq m}$ est le **multiplicateur**
  - $\boldsymbol{c},\ \boldsymbol{0 < c \leq m}$ est **l'incrément**
  - $x_0,\ \boldsymbol{0 < x_0 \leq m}$ est **la semence ou valeur initiale (seed en anglais)**

- Les entiers x_n générés prennent des valeurs de 0 à $m - 1$, alors,
- Pour calculer le nombre uniforme pseudo-aléatoire u_n, pris comme une approximation de la valeur d'une variable aléatoire uniforme $U \sim Unif\ [0,1[$, nous utilisons la formule :

$$u_n = \frac{x_n}{m}$$

Exemple : Considérons un LCG avec des paramètres ($m = 8, a = 5, c = 1, x_0 = 5$).

Calculez les neuf premières valeurs dex_i et u_i .

Formule du générateur congruetiel linéaire: $x_n = (5x_{n-1} + 1)\ mod\ 8$

$x_1 = (5 \times 5 + 1)\ mod\ 8 = 2 \Rightarrow\ u_1 = \frac{x_1}{m} = \frac{2}{8} = 0.25$

$x_2 = (5 \times 2 + 1)\ mod\ 8 = 3 \Rightarrow\ u_1 = \frac{x_2}{m} = \frac{3}{8} = 0.375$

$x_3 = (5 \times 3 + 1)\ mod\ 8 = 0 \Rightarrow\ u_1 = \frac{x_{13}}{m} = \frac{0}{8} = 0$

$x_4 = (5 \times 0 + 1)\ mod\ 8 = 1 \Rightarrow\ u_1 = \frac{1}{8} = 0.125$

$x_5 = (5 \times 1 + 1)\ mod\ 8 = 6 \Rightarrow\ u_1 = \frac{6}{8} = 0.75$

$x_6 = (5 \times 6 + 1)\ mod\ 8 = 7 \Rightarrow\ u_1 = \frac{7}{8} = 0.875$

$x_7 = 4\ \Rightarrow\ u_1 = 0.5$

$x_8 = 5\ \Rightarrow\ u_1 = 0.625$

$x_9 = 2 \Rightarrow\ u_1 = 0.25$

Exemple : Considérons un LCG avec des paramètres ($m = 8, a = 5, c = 1, x_0 = 5$).

Calculez les neuf premières valeurs dex_i et u_i .

Formule du générateur congruetiel linéaire: $x_n = (5x_{n-1} + 1)\ mod\ 8$

$x_1 = (5 \times 5 + 1)\ mod\ 8 = 2 \Rightarrow\ u_1 = \frac{x_1}{m} = \frac{2}{8} = 0.25$

$x_2 = (5 \times 2 + 1)\ mod\ 8 = 3 \Rightarrow\ u_1 = \frac{x_2}{m} = \frac{3}{8} = 0.375$

$x_3 = (5 \times 3 + 1)\ mod\ 8 = 0 \Rightarrow\ u_1 = \frac{x_{13}}{m} = \frac{0}{8} = 0$

$x_4 = (5 \times 0 + 1)\ mod\ 8 = 1 \Rightarrow\ u_1 = \frac{1}{8} = 0.125$

$x_5 = (5 \times 1 + 1)\ mod\ 8 = 6 \Rightarrow\ u_1 = \frac{6}{8} = 0.75$

$x_6 = (5 \times 6 + 1)\ mod\ 8 = 7 \Rightarrow\ u_1 = \frac{7}{8} = 0.875$

$x_7 = 4\ \Rightarrow\ u_1 = 0.5$

$x_8 = 5\ \Rightarrow\ u_1 = 0.625$

$x_9 = 2 \Rightarrow\ u_1 = 0.25$

Période du LCG

- Soient $x_0, x_1, x_2, \ldots$ la séquence générée par un LCG de paramètres (m , a , c , x_0). La **période** d'un générateur congruentiel linéaire (LCG) est le plus petit entier positif n tel que : $x_0 = x_n$
- La période correspond à la longueur de la séquence avant qu'elle ne commence à se répéter: Si $x_0 = x_n$ alors: $x_1 = x_{n+1}$, $x_2 = x_{n+2}, etc$
- Une **propriété importante d'un LCG** est qu'il a une **période longue**, aussi proche que possible du module m.
- Si la période du LCG est égale à $\boldsymbol{m}$, on dit que le LCG a une **période complète (full period)**.

Théorème (Conditions de période complète LCG). Un LCG de paramètres (m , a , c , x_0) a une période complète si et seulement si les trois conditions suivantes sont remplies :

1. $\mathbf{m}$ et $\boldsymbol{c}$ doivent être premiers entre eux (Le seul entier positif qui divise à la fois m et c est 1).
2. $\mathbf{a - 1}$ doit être divisible par tous les facteurs premiers de $\boldsymbol{m}$. Autrement dit, si $\boldsymbol{q}$ est un nombre premier qui divise $\boldsymbol{m}$, alors q doit diviser $(\boldsymbol{a - 1})$,
3. **Si 4** divise $\boldsymbol{m}$, alors 4 devrait diviser $(\boldsymbol{a - 1})$. Autrement dit, si $\boldsymbol{m}$ est un multiple de **4**, alors $(\boldsymbol{a - 1})$ doit également être un multiple de **4**.

Exemple

- Soit le LCG de paramètres ($m = 8, a = 5, c = 1, x_0 = 5$). Vérifier s'il doit ou non avoir une période complète.
- Pour appliquer le théorème, vous devez vérifier si chacune des trois conditions est vraie pour le générateur.

Condition 1 : c et m n'ont pas de facteurs communs autres que 1:

- Les facteurs de $m = 8$ sont (1, 2, 4, 8), puisque $c = 1$ (de facteur 1), donc, la condition 1 est vérifiée.

Condition 2 : $(a - 1)$ est un multiple de tout nombre premier qui divise m.

- Les nombres premiers q, qui divisent $m = 8$ sont $(q = 1, 2)$.
- $(a - 1) = 4, q = 1$ divise 4 et $q = 2$ divise 4. Ainsi, la condition 2 est vraie.

Condition 3 : si 4 divise m, alors 4 devrait diviser $(a - 1)$.

- 4 divise $m = 8$. De plus, 4 divise $(a - 1) = 4$. Ainsi, la condition 3 est vérifiée.

Le LCG de paramètres ($m = 8, a = 5, c = 1, x_0 = 5$) a une **période complète**

3. Génération des entrées de simulation

Choix des constantes a, c et m

Les constantes a, c et m doivent être choisies pour satisfaire trois critères :

1. Pour toute valeur initiale x_0, la séquence résultante a « l'apparence » d'être une séquence d'une variable aléatoire uniforme $X \sim \mathcal{U}(0,1)$.
2. Pour toute valeur initiale x_0, le nombre de variables pouvant être générées avant le début de la répétition est important. (après un nombre fini de valeurs générées (au plus m),
3. Les valeurs peuvent être calculées efficacement sur un ordinateur numérique.

Dans les générateurs congruents linéaires (LCG), l'utilisation du modulo $\boldsymbol{m = 2^n}$ est courante pour des raisons liées à l'efficacité de calcul
En fait, le modulo 2^n est particulièrement efficace à calculer avec des opérations de bits : $a \bmod 2^n = a \gg n$
Le calcul en utilisant des opérations de bits est très rapide sur les ordinateurs.

Exemples :

$$37 \bmod 2^2 = 37 \gg 2 = (100101)_2 \gg 2 = (01)_2 = 1$$
$$37 \bmod 2^3 = 37 \gg 3 = (100101)_2 \gg 3 = (101)_2 = 5$$
$$57 \bmod 2^4 = 57 \gg 4 = (111001)_2 \gg 4 = (1001)_2 = 9$$

3. Génération des entrées de simulation

3.2. Génération de nombres pseudo-aléatoires

3.2.3. Générateurs à registre à décalage à rétroaction (FSRG)

Les Générateurs à registre à décalage à rétroaction (Feedback Shift Register Generators : FSRG) sont une classe d'algorithmes de génération de nombres pseudo-aléatoires basés sur des registres à décalage (shift registers) et des opérations de rétroaction (feedback). :

- **Registre à décalage (*Shift Register*)** : Un registre à décalage est un registre binaire dans lequel les bits sont déplacés d'une position à chaque itération. Le contenu du registre est décalé d'une position vers la gauche (ou la droite).
- **Rétroaction (*Feedback*)** : La rétroaction consiste à appliquer une opération logique (souvent le *XOR*) entre certains bits du registre à décalage et à utiliser le résultat comme entrée pour le bit le plus à gauche du registre à décalage lors de la prochaine itération.
- **Graine** (Seed) : Comme de nombreux générateurs de nombres pseudo-aléatoires, les FSR Generators nécessitent une valeur initiale appelée "seed". La seed détermine l'état initial du registre à décalage et, par conséquent, la séquence de nombres générée.
- **Période** : La période d'un FSR Generator dépend de la longueur du registre à décalage et des opérations de rétroaction spécifiques utilisées. Si le générateur atteint un état précédemment visité, il commencera à répéter la séquence, et la période est le nombre maximal d'itérations avant qu'une telle répétition ne se produise.

3. Génération des entrées de simulation

3.2. Génération de nombres pseudo-aléatoires

3.2.3. Générateurs à registre à décalage à rétroaction (FSRG)

Exemple :

- Le registre est initialisé avec la valeur 1101.
- Supposons que nous ayons un générateur de nombres pseudo-aléatoires à registre à décalage à rétroaction génératif (LFSR) à 4 bits :
- À chaque étape, nous utilisions une opération de rétroaction *XOR* entre le premier et le quatrième bit pour générer un nouveau bit.
- Le bit le plus à droite du registre est supprimé (décalage à droite), et le nouveau bit devient le nouveau bit le plus à gauche.
- Le tableau ci-dessous montre les étapes de génération des nombres aléatoires :

Étape	Registre	Rétroaction	Bit généré
0	1101	$1 \oplus 1$	0
1	0110	$0 \oplus 0$	0
2	0011	$0 \oplus 1$	1
3	1001	$1 \oplus 1$	0
4	0100	$0 \oplus 0$	0
5	0010	$0 \oplus 0$	0
6	0001	$0 \oplus 1$	1
7	1000	$1 \oplus 0$	1
8	1100	$1 \oplus 0$	1

3. Génération des entrées de simulation

3.2. Génération de nombres pseudo-aléatoires

3.2.5. Mersenne Twister

- Mersenne Twister est une variante du générateur de nombres pseudo-aléatoires à registre à décalage à rétroaction génératif (LFSR), qui utilise une rétroaction complexe conçue pour produire une séquence de nombres aléatoires avec une bonne uniformité et une période très longue.

- La version la plus couramment utilisée de l'algorithme Mersenne Twister est le **MT19937** avec une très longue période de $\mathbf{2^{19937} - 1}$. Cela signifie qu'il générera une séquence de $\mathbf{2^{19937} - 1}$ nombres avant de commencer à se répéter.

- Le **MT19937** le générateur de nombres pseudo-aléatoires par défaut du langage Python

3. Génération des entrées de simulation

3.2. Génération de nombres pseudo-aléatoires

3.2.5. Génération de Nombres pseudo-aléatoires dans les Langages de Programmation

- **En Java**, la génération de nombres pseudo-aléatoires se fait par la classe `java.util.Random`
- **Java.util.Random** est un générateur de nombres pseudo-aléatoires congruentiels linéaires de paramètres :

$$\boldsymbol{m = 2^{48}} \quad \boldsymbol{a} = 5\text{DEECE66D}_{\mathbf{16}} = 25214903917 \quad \boldsymbol{c = 11}$$

- **Quelques méthode de la classe `java.util.Random`**

Méthode	Description
`nextDouble()`	Renvoie nombre réel pseudo-aléatoire uniformément distribuée dans l'intervalle [0 , 1[à partir de la séquence de ce générateur de nombres aléatoires.
`nextInt(int n)`	Renvoie un entier pseudo-aléatoire uniformément distribuée dans l'intervalle [0 , n[.
`nextBoolean()`	Renvoie un booléen aléatoire.
`setSeed (long seed)`	Définit la graine du générateur de nombres aléatoires.
`getSeed()`	Renvoie la graine du générateur de nombres aléatoires.

```java
// Exemple
import java.util.Random;
public class RandomExample {
    public static void main(String[] args) {
              Random random = new Random(); // Création d'une instance de la classe Random
              int a = random.nextInt(100); // Génération d'un nombre entier pseudo-aléatoire entre 0 (inclus) et 100 (exclus)
             double x = random.nextDouble(); // Génération d'un nombre réel pseudo-aléatoire entre 0.0 (inclus) et 1.0 (exclus)
             boolean bool = random.nextBoolean(); // Génération d'une valeur boooléenne aléatoire (true ou false)
    }
}
```

3. Génération des entrées de simulation

3.2. Génération de nombres pseudo-aléatoires

3.2.5. Génération de Nombres pseudo-aléatoires dans les Langages de Programmation

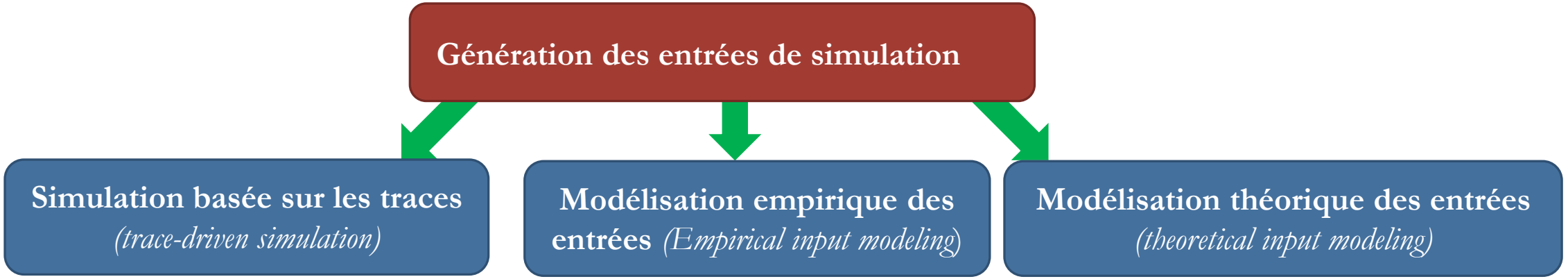
- **Python** offre une bibliothèque intégrée appelée **`random`** pour la génération de nombres aléatoires. Voici quelques fonctions de la bibliotheque random :
- Python utilise l'algorithme ***Mersenne Twister*** comme générateur de base.
- ***Mersenne Twister*** produit des flottants a une période de 2^{19937}-1.

Quelques fonctions de la bibliothèque **`random`**

Fonction	Description
`random()`	Retourne un nombre décimal pseudo-aléatoire dans l'intervalle [0.0, 1.0[.
`randint(a, b)`	Génère un entier pseudo-aléatoire dans l'intervalle [a, b].
`uniform(a, b)`	Génère un nombre décimal pseudo-aléatoire dans l'intervalle [a, b).
`seed(a)`	Initialise le générateur de nombres pseudo-aléatoires avec une graine `a`. Si `a` n'est pas spécifié, l'heure système est utilisée.
…	…

```
# Exemple
import random
x = random.random() # Génération d'un nombre réel dans l'intervalle [0,1[
random.seed(42) # Initialiser le générateur avec une graine spécifique
a = random.randint(0, 10) # Générer un entier aléatoire dans l'intervalle [0,10[
bool = random.choice([True, False]) # Générer un booléen aléatoire
```

- Trois approches pour modéliser les entrées :
  1. **La simulation basée sur les traces** *(trace-driven simulation)*
  2. **La modélisation empirique des entrées** *(Empirical input modeling)*
  3. **La modélisation théorique des entrées** *(theoretical input modeling)*



- Pour bien illustrer ces différentes approches, prenons l'exemple suivant : Soient le tableau suivant qui représente les données collectées des temps inter-arrivées et de services des 10 premiers clients d'un guichet automatique bancaire (GAB).

Numéro d'observation (k)	1	2	3	4	5	6	7	8	9	10
Temps inter-arrivées $\{A_k\}$	121	13	87	36	7	236	8	33	152	67
Temps de service $\{X_k\}$	56	51	73	65	84	58	62	69	44	66

3.3.1 Simulation basée sur des traces (Trace-driven simulation)

- Dans cette approche, les données réelles collectées auprès du système réel sont utilisées directement dans la simulation.
- Par exemple, les temps inter-arrivée de clients collectés dans une file d'attente, peuvent être utilisé directement pour générer les arrivées dans un modèle de simulation. De même, les temps de service réels peuvent être utilisés pour simuler les temps de service.

Avantages : Capture les dépendances complexes et les corrélations entre les variables d'entrée.

Limites:

- Nécessite de grandes quantités de données, qui ne sont pas toujours disponibles.
- Les données réelles peuvent être limitées et/ou ne couvrent pas toutes les situations possibles, ce qui peut entraîner des imprécisions potentielles dans le modèle de simulation.

3.3.2. Modélisation empirique des entrées

Dans la modélisation empirique des entrées, les variables aléatoires pour la simulation sont générées directement à partir des données collectées.

Avantages :

- La modélisation empirique des entrées peuvent être utilisée lorsque les données réelles sont limitées, mais suffisantes pour estimer une distribution empirique.

Limites :

- La précision du modèle d'entrée dépend fortement de la qualité et de la représentativité des données collectées
- Les données peuvent pas capturer toutes les complexités du système,

- Deux méthodes pour générer des variables aléatoires à partir de données collectées :
  - **La modélisation non paramétrique,**
  - **La transformation inverse de la fonction de répartition empirique.**

Dans la modélisation empirique des entrées, les variables aléatoires pour la simulation sont générées directement à partir des données collectées.

Avantages :

- La modélisation empirique des entrées peuvent être utilisée lorsque les données réelles sont limitées, mais suffisantes pour estimer une distribution empirique.

Limites :

- La précision du modèle d'entrée dépend fortement de la qualité et de la représentativité des données collectées
- Les données peuvent pas capturer toutes les complexités du système,

- Deux méthodes pour générer des variables aléatoires à partir de données collectées :
  - **La modélisation non paramétrique,**
  - **La transformation inverse de la fonction de répartition empirique.**

- Les valeur de la variable aléatoire x sont échantillonnées à plusieurs reprises à partir des données collectées $\{X_k, k = 1, \dots, n\}$ avec une probabilité uniforme $1/n$.

Algorithme de modélisation d'entrée non paramétrique:

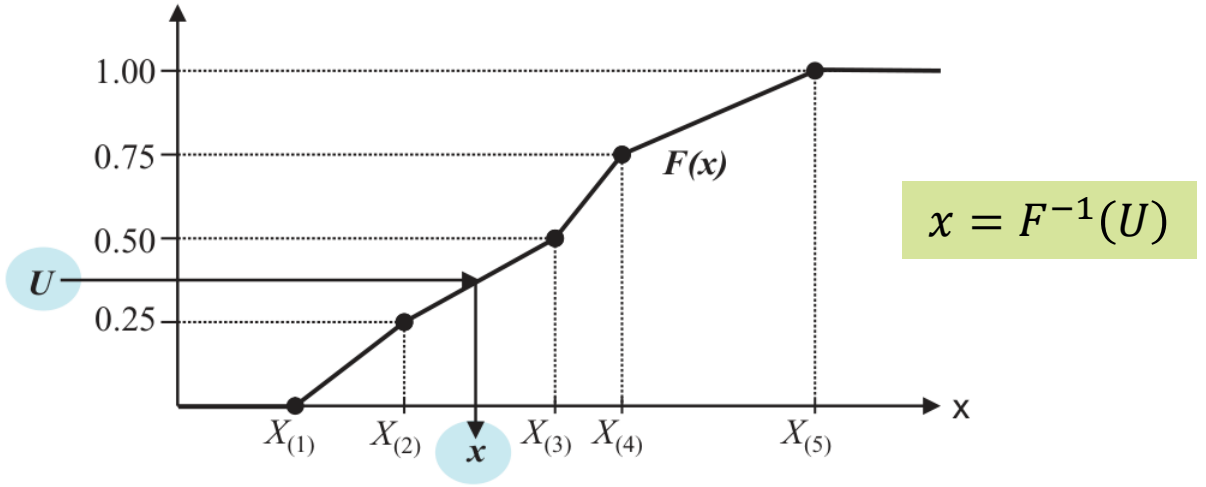
1. Générer un nombre aléatoire uniforme $u \sim \mathcal{U}[0,1[$.
2. Calculer $P = n \times U$ et l'indice $k = [P] + 1$; // $[P]$ est la partie entière de P .
3. Retourner $x = X_k$

- ***Exemple:*** Générer un temps de service en utilisant la méthode non paramétrique à partir des données de temps de service $\{X_k\}$ données dans le tableau.

1. Générer un nombre aléatoire uniforme $u \sim \mathcal{U}(0,1)$. Soit $u = 0{,}36988$
2. $P = n \times u = 10 \times 0{,}36988 = 3{,}6988$, l'indice $k = [P] + 1 = 3 + 1 = 4$
3. Retourner $x = X_4 = 65$ comme temps de service empirique à utiliser dans la simulation.

Numéro d'observation (k)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Temps inter-arrivées $\{A_k\}$	121	13	87	36	7	236	8	33	152	67
Temps de service $\{X_k\}$	56	51	73	**65**	84	58	62	69	44	66

- Soit $\{X_{(k)}, k = 1, \dots, n\}$ les données d'échantillon ordonnées dans un ordre croissant,
- La fonction de répartition empirique de X est $F\left(X_{(k)}\right) = \frac{k-1}{n-1}$.
- Application de la méthode de transformation inverse pour générez entrées x (Générer $u \sim \mathcal{U}(0,1)$, puis calculer $F^{-1}(u)$



$x = F^{-1}(U)$

Algorithme de transformation inverse de la fonction de répartition empirique :

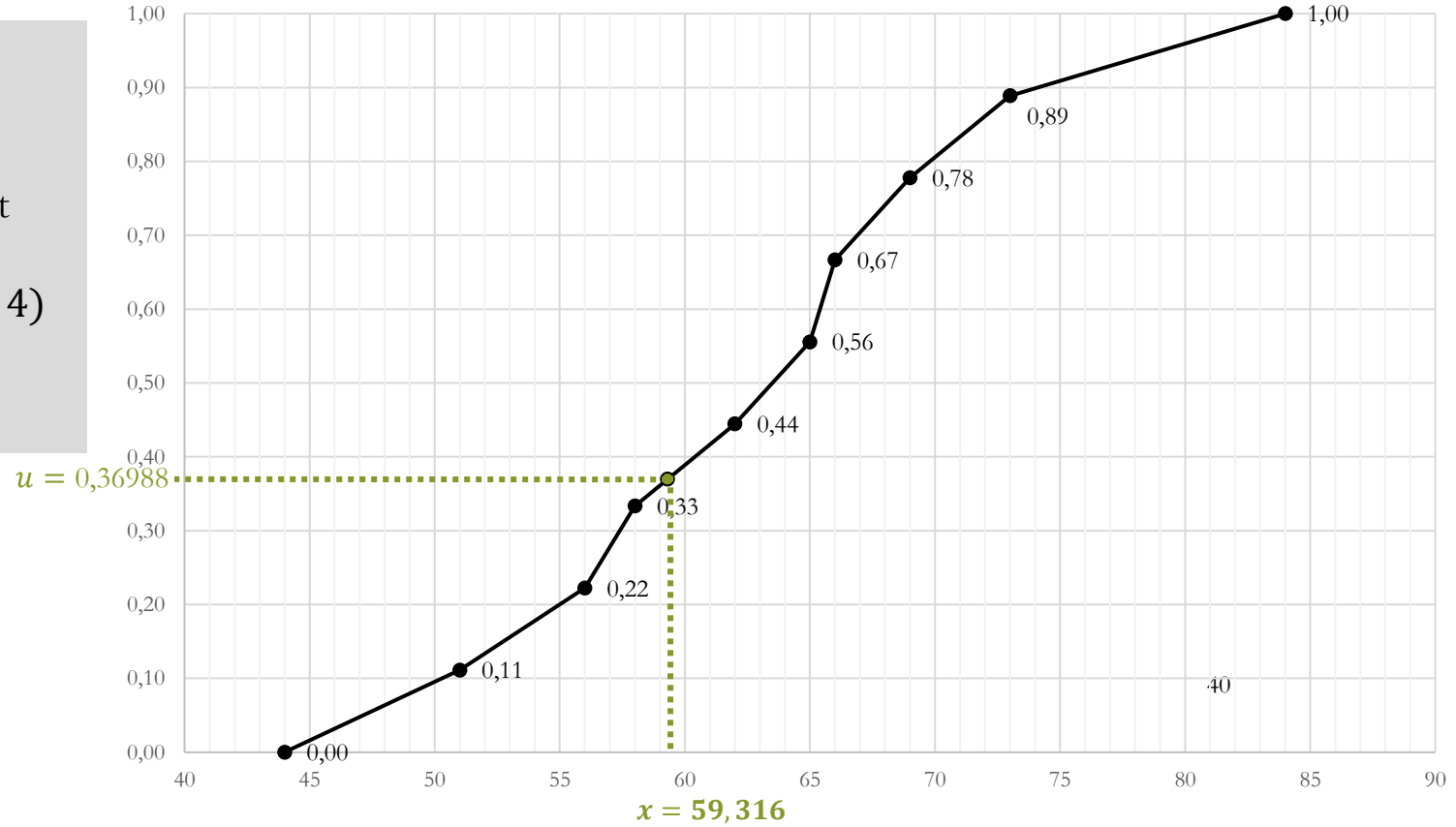
1. Générer un nombre aléatoire uniforme $U \sim \mathcal{U}(0,1)$.
2. Calculer $P = (n-1) \times U$ et $j = [P] + 1$; // [P] est la partie entière de P .
3. Retourner $x = X_{(j)} + (P + 1 - j) \times \left(X_{(j+1)} - X_{(j)}\right)$

- ***Exemple.*** Le tableau montre les de temps de service ordonnées dans un ordre croissant $\{X_{(k)}\}$

Indice d'ordre croissant (k)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Temps de service $\{X_{(k)}\}$	**44**	**51**	**56**	**58**	**62**	**65**	**66**	**69**	**73**	**84**
F($X_{(k)}$)	0	0,111	0,222	0,333	0,444	0,556	0,667	0,778	0,889	1,000

Génération d'un temps de service

1. Générer un nombre aléatoire uniforme $u \sim \mathcal{U}(0,1)$. Soit $u = 0{,}36988$
2. $P = (n-1) \times u = 9 \times 0{,}36988 = 3{,}328929$ et $j = [P] + 1 = 4$;
3. Retourner le temps de service: $x = X_{(4)} + (P + 1 - 4) * (X_{(4+1)} - X_{(4)}) = 58 + 0{,}328929 \times (62 - 58) = 59{,}316$



- ***Exercice.*** Appliquer la transformation inverse de la fonction de répartition empirique pour générer des temps inter-arrivées, à partir du nombre aléatoire uniforme $u = 0{,}74381276$

Indice d'ordre croissant (k)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Temps de service $\{X_{(k)}\}$	**44**	**51**	**56**	**58**	**62**	**65**	**66**	**69**	**73**	**84**
F($X_{(k)}$)	0	0,111	0,222	0,333	0,444	0,556	0,667	0,778	0,889	1,000

- Générer un nombre aléatoire uniforme $u \sim \mathcal{U}(0,1)$. Soit $u = u = 0{,}74381276$

1. $P = (n-1) \times u = 9 \times 0{,}74381276 = 6{,}69431484$ et $j = [P] + 1 = 7$;
2. Retourner le temps de service:

$$x = X_{(7)} + (P + 1 - 7) \times (X_{(7+1)} - X_{(7)}) = 87 + 0{,}69431484 \times (121 - 87) = 110{,}606705$$

3.3.3. Modélisation théorique des entrées (Theoretical input modeling)

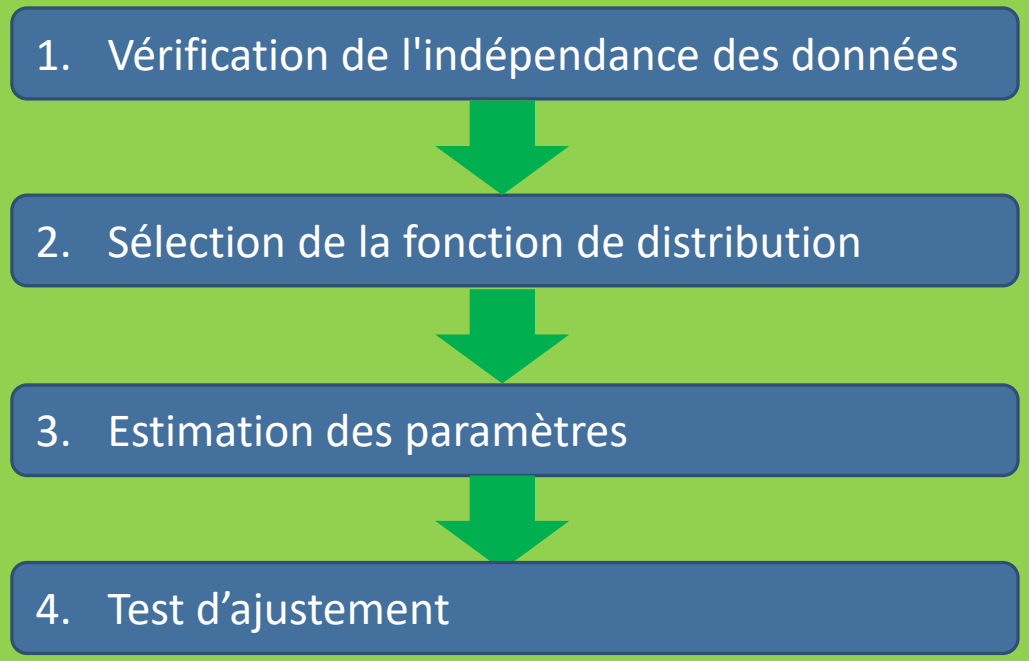
- La modélisation théorique des entrées (*theoretical input modeling*), consiste à sélectionner une loi de distribution théorique appropriée, puis, estimer les paramètres de sa fonction de de répartition à partir des données réelles, finalement des variables aléatoires sont générées à partir de la fonction de répartition ajustée.

Avantages : La modélisation théorique des entrées est adaptée

- Lorsque les données réelles collectées sont insuffisantes pour estimer des distributions empiriques,
- Lorsqu'il est possible de trouver une distribution théorique qui correspond bien au comportement du système.

Limites: Les hypothèses des distributions théoriques peuvent ne pas refléter pleinement les complexités du système réel, ce qui entraîne des imprécisions potentielles dans les résultats de simulation.

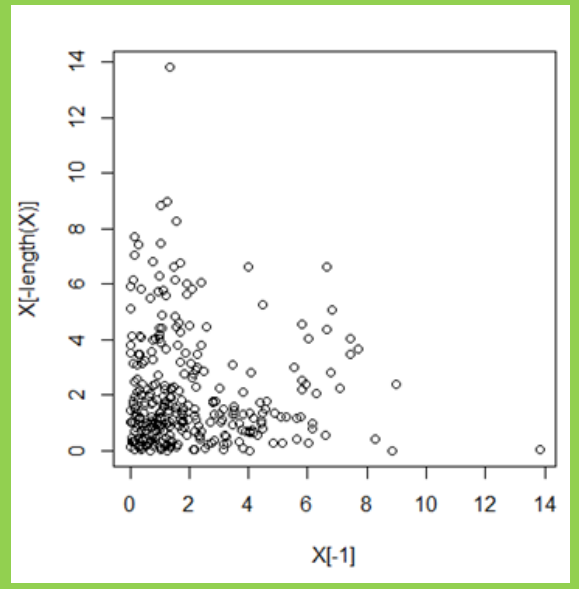
- L'ajustement de la distribution est un processus d'estimation statistique classique qui se déroule en 4 étapes:



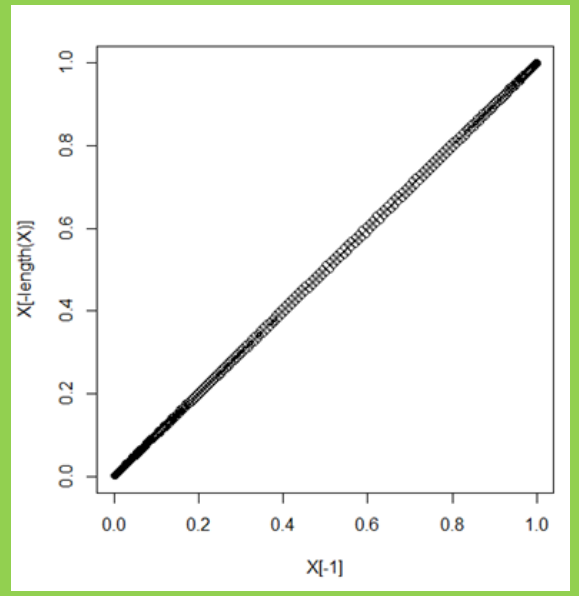
1. **Vérification de l'indépendance des données:** La première étape de la modélisation théorique des entrées consiste à vérifier si les données obtenues sont indépendantes.

Tracer le diagramme de dispersion:

- Pour les données $X_1, X_2 \ldots X_n$ **répertoriés dans l'ordre temporel de la collecte**, les paires (X_i, X_{i+1}) pour $i = 1 \sim n - 1$ sont tracées sur un système de coordonnées (X_i comme valeur de x et X_{i+1}comme valeur de y).
- Si les points tracés sont dispersés au hasard, on peut conclure que les données sont indépendantes.



Exemple de données indépendants



Exemple de données dépendantes

1. Vérification de l'indépendance des données
2. Sélection de la fonction de distribution
3. Estimation des paramètres
4. Test d'ajustement

2. Sélection de la fonction de distribution

La deuxième étape consiste à sélectionner une fonction de distribution candidate appropriée sur la base **d'une justification théorique** et en observant **les propriétés statistiques (moyenne, écart type, skewness, kurtosis, etc.), les graphiques de densité de probabilités et de la fonction de répartition empirique.**

Distributions	Applications
Exponentielle (θ)	Modélisation des **temps inter-arrivées** des « clients » qui se produisent à un taux constant ;
Erlang (k, θ)	Modélisation des **temps inter-arrivées** des « clients » ;
Weibull(α, β)	Modélisation des **temps inter-arrivées** des « clients » qui se produisent à un taux qui augmente ou diminue dans le temps ; – Lorsque $\beta > 1$, le taux augmente avec le temps (phase d'usure), – $\beta = 1$ correspond à un taux constant (distribution exponentielle), – Lorsque $0 < \beta < 1$, le taux de diminue avec le temps.
Uniforme (a, b)	Modélisation des **temps de services** lorsque seule la plage [a, b] des durées de service est fournie.
Triangular (a, b, c)	Modélisation des **temps de services** si le mode c est également donné en plus de la plage [a, b].
Bêta (α, β)	Modélisation des **temps de services** avec une plage finie. La distribution bêta standard $Y \sim Beta(\alpha, \beta)$ a une plage unitaire [0, 1]. Alors la variable aléatoire bêta X avec une plage générale $[a, b]$ peut être obtenue à partir de Y comme suit : $X = a + Y(b - a)$
Normale (μ, σ)	Modélisation des **temps de services si** la distribution des temps de services est symétrique
Lognormal (μ, σ)	Modélisation des **temps de services si** la distribution des temps de services est asymétrique vers la droite (asymétrie négative), ils sont générés à partir de la distribution log-normale.

1. Vérification de l'indépendance des données
2. Sélection de la fonction de distribution
3. Estimation des paramètres
4. Test d'ajustement

3. Estimation des paramètres

- La troisième étape consiste à estimer les paramètres de la distribution sélectionnée.
- L'estimateur du maximum de vraisemblance (Maximum likelihood estimation : MLE) est le choix préféré pour l'estimation des paramètres,
- D'autres méthodes peuvent être utilisées lorsque le MLE n'a pas une forme simple. Par exemple, la méthode des moments (Method of moment)

Distributions	Méthode d'estimation des paramètres	Estimateurs
$Exponentielle\ (\theta)$	Méthode du maximum de vraisemblance	$\hat{\lambda} = \frac{1}{\sum_{i=1}^{n} x_i} = \frac{1}{\bar{x}}$
$Erlang\ (k,\theta)$ $Gamma\ (k,\theta)$	Méthode des moments, Méthode du maximum de vraisemblance	$m_1 = \frac{1}{n}\sum_{i=1}^{n} x_i = \bar{x}, m_2 = \frac{1}{n}\sum_{i=1}^{n} {x_i}^2$ $\hat{\lambda} = \frac{1}{\hat{\theta}} = \frac{m_1}{(m_2 - {m_1}^2)}$, $\hat{k} \cong \frac{{m_1}^2}{(m_2 - {m_1}^2)}$
$Weibull(\alpha, \beta)$	Méthode des moments, Méthode du maximum de vraisemblance	
$Bêta\ (\alpha, \beta)$	Méthode des moments	$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$ $\hat{\alpha} = \hat{x}\left(\frac{\hat{x}(1-\hat{x})}{s^2} - 1\right)$, $\hat{\beta} = (1-\hat{x})\left(\frac{\hat{x}(1-\hat{x})}{s^2} - 1\right)$
$Normale\ (\mu, \sigma)$ $Lognormal\ (\mu, \sigma)$	Méthode du maximum de vraisemblance	$\hat{\mu} = \sum_{i=1}^{n} x_i = \bar{x}$, $\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$

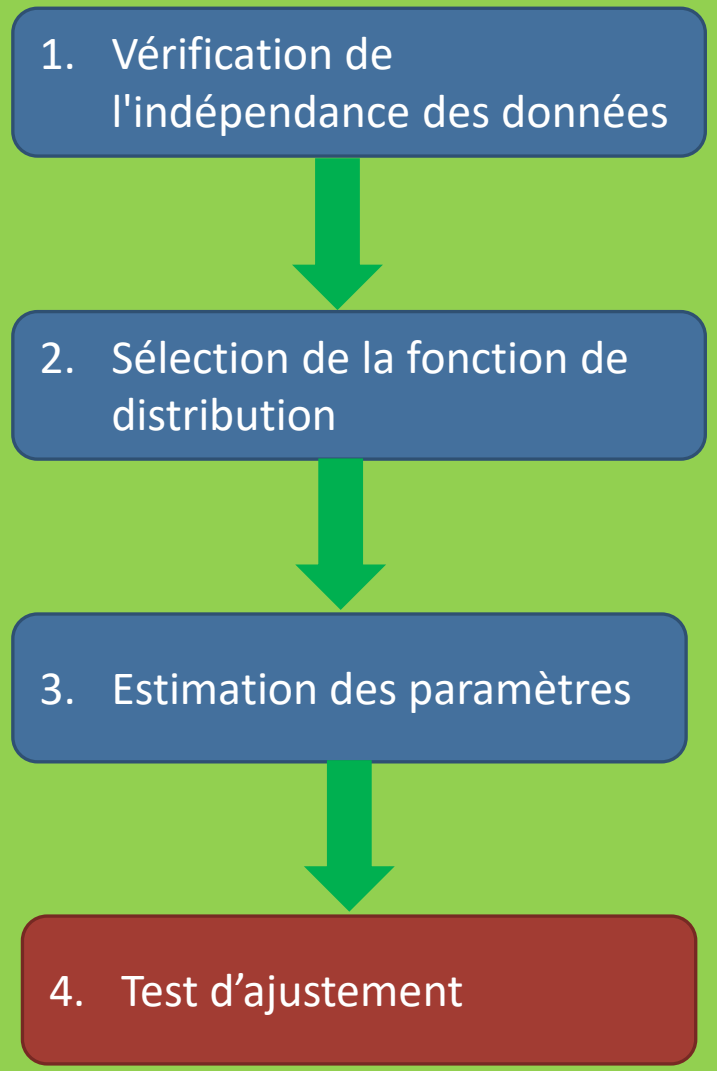
1. Vérification de l'indépendance des données
2. Sélection de la fonction de distribution
3. Estimation des paramètres
4. Test d'ajustement

4. Test d'ajustement

La quatrième étape de l'ajustement de la distribution théorique évalue l'adéquation du modèle à l'aide d'un test d'ajustement

Différents test d'ajustement :

- *Test du khi*-deux χ^2,
- est de Kolmogorov-Smirnov,
- Test d'Anderson-Darling.



- Après avoir ajusté une distribution théorique pour chaque type de données d'entrée, la phase finale de la modélisation d'entrée consiste à **générer des données pour la simulation**.
- Lorsque la fonction de distribution a une fonction inverse connue, la **méthode de transformation inverse** est le choix.
- Autrement, des méthodes spéciales de génération de variables aléatoires peuvent être utilisées.
- Le tableau résume les méthodes de génération de variables aléatoires.

Distributions	Méthodes de génération
$Exponentielle\ (\theta)$	Inverse-transform
$Erlang\ (k, \theta)$	Convolution of exponential
$\mathbf{Weibull}(\alpha, \beta)$	Inverse-transform
$Triangulaire\ (a, b, c)$	Composition method
$Bêta\ (\alpha, \beta)$	Acceptance–rejection
$Normale\ (\mu, \sigma)$	Box & Muller method
$Lognormal\ (\mu, \sigma)$	Conversion of normal variate

Théorème :

- Soit X une variable aléatoire de fonction de répartition F, donc $F(x) = P(X \leq x)$
- Soit F^{-1} la fonction réciproque (Inverse) de F: $F(x) = y \Leftrightarrow F^{-1}(y) = x$
- Soit $U \sim \mathcal{U}(0,1)$, alors la variable aléatoire $F^{-1}(U)$ a pour fonction de répartition F . Autrement dit: $F^{-1}(U)$ suit la même loi que X.

Démonstration

Soit G la fonction de répartition de $F^{-1}(U)$:

$G(x) = P(F^{-1}(U) \leq x)$

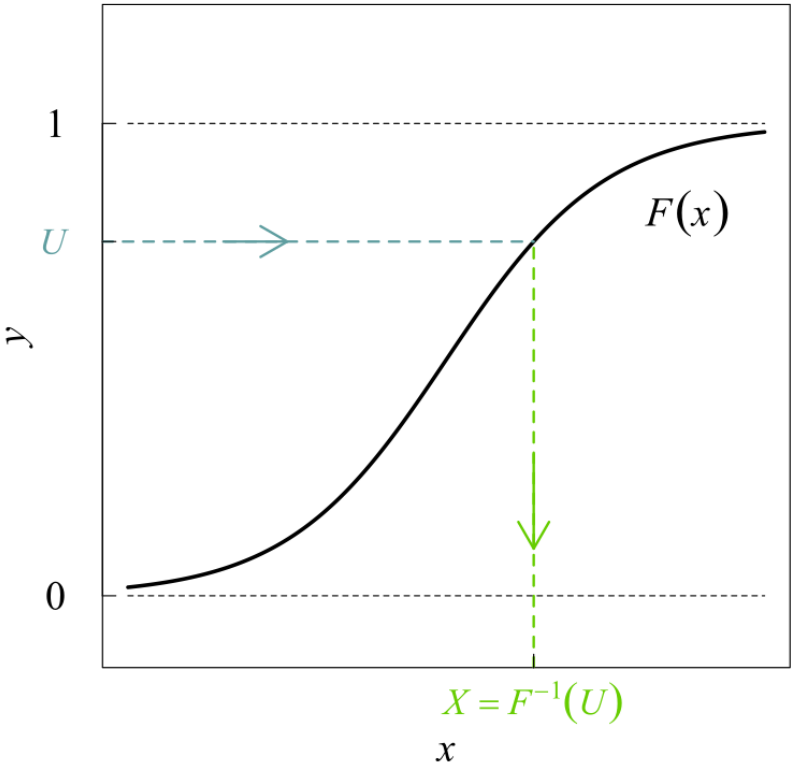
F est une fonction de répartition, donc F est strictement croissante, donc:

$G(x) = P\left(F\left(F^{-1}(u)\right) \leq F(x)\right)$

$F\left(F^{-1}(U)\right) = U$, donc: $G(x) = P\left(U \leq F(x)\right)$

U $\sim \mathcal{U}(0,1)$, donc:

$G(x) = P\left(U \leq F(x)\right) = F(x)$



- La méthode de transformation inversée consiste générer des valeurs d'une variable aléatoire X continue de fonction de répartition F, pour laquelle F^{-1} est connue, à partir des échantillons de la loi uniforme U $\sim \mathcal{U}$ (0,1).

- L'algorithme de transformation inversée peut être résumé comme suit:

1: Générer une valeur u de U $\sim \mathcal{U}$ (0,1)
2: Calculer $x = F^{-1}(u)$
3: Retourner x

Exemple: La loi exponentielle

Soit X une variable aléatoire qui suit une loi exponentielle de paramètre λ: X $\sim Exp\ (\lambda)$

La fonction de répartition de X: $F(x) = P(X \leq x) = 1 - e^{-\lambda x}$

Donc: $F^{-1}(x) = -\frac{\ln(1-x)}{\lambda}$

Si X $\sim Exp\ (\lambda)$, et U $\sim \mathcal{U}\ (0,1)$, nous avons: $-\frac{\ln(1-U)}{\lambda} \sim Exp\ (\lambda)$

Remarquons que $(1 - U) \sim \mathcal{U}\ (0,1)$, donc: $-\frac{\ln(U)}{\lambda} \sim Exp\ (\lambda)$

- Algorithme pour générer des valeurs d'une variable aléatoire X $\sim Exp\ (\lambda)$:

1: Générer une valeur u_i de U $\sim \mathcal{U}\ (0,1)$
2: Calculer $x_i = -\frac{\ln(u_i)}{\lambda}$
3: Retourner x_i

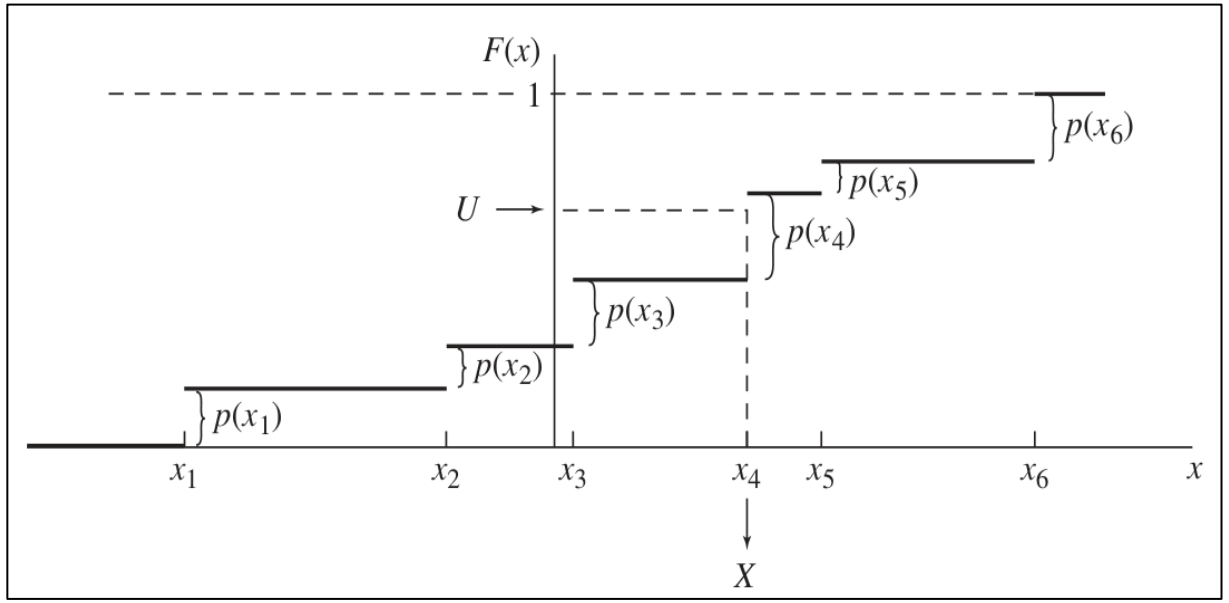
Considérons une variable aléatoire discrète non négative X, qui peut prendre les valeurs $x_0, x_1, x_2, \ldots, x_i, \ldots, x_n$

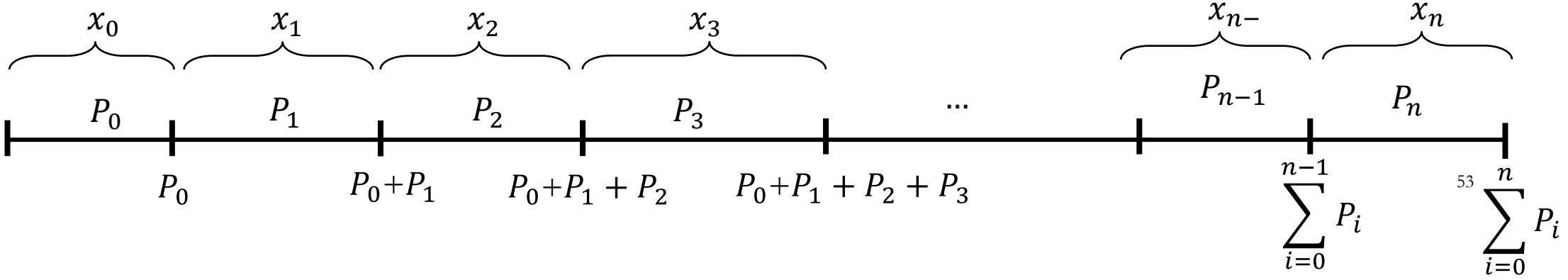
- La fonction une fonction de masse de X $f(x) = P(X = x_i) = P_i, x_i \geq 0.$
- La fonction cumulative (de répatition) $F(X) = P(X \leq x_i) = \sum_{j=0}^{i} P_i$

La fonction $F^{-1}(U)$, U $\sim \mathcal{U}$ (0,1), est explicitement donnée par :

$$F^{-1}(u) = \begin{cases} x_0 & Si\ 0 < u < F(x_0) \\ x_i \text{ Si } F(x_i) < u \leq F(x_{i+1}) \end{cases}$$

$$F^{-1}(u) = \begin{cases} x_0, & Si\ 0 \le u < P_0 \\ x_1, & Si\ \ P_0 \le u < P_0 + P_1 \\ x_2, & Si\ P_0 + P_1 \le u < P_0 + P_1 + P_2 \\ x_3, & SiP_0 + P_1 + P_2 \le u < P_0 + P_1 + P_2 + P_3 \\ & \dots \end{cases}$$





Exemple

Soit une variable aléatoire X telle que:

$P(X = 1) = P_1 = 0{,}20$

$P(X = 2) = P_2 = 0{,}15$

$P(X = 3) = P_3 = 0{,}25$

$P(X = 4) = P_4 = 0{,}40$

Algorithme pour générer la distribution de X

Générer U $\sim \mathcal{U}(0,1)$ et faire ce qui suit :

```
Si (U < 0,20)alors
        X = 1
Sinon
    Si (U < 0,35)alors
            X = 2
    Sinon
        Si (U < 0,60) alors
                X = 3
        Sinon
                X = 4
        FinSi
    FinSI
FinSI
```

- Pour certaines distributions théoriques, la variable aléatoire X peut être exprimée comme une somme d'autres variables aléatoires qui sont IID.
- Nous supposons qu'il existe des variables aléatoires IID $Y_1, Y_2, \ldots, Y_m$, tel que la somme $Y_1 + Y_2 + \cdots + Y_m$ à la même distribution que X ; nous écrivons donc $X = Y_1 + Y_2 + \cdots + Y_m$
- Soit F la fonction de répartition de X et G la fonction de répartition de Y_i. L'algorithme de génération de la variable X :

(1) Générez $Y_1, Y_2, \ldots, Y_m$ IID chacun avec sa fonction de distribution G.
(2) Retour $X = Y_1 + Y_2 + \cdots + Y_m$

- La méthode d'acceptation-rejet utilise une fonction majorante $g(x)$ de la fonction de densité $f\ (x)$ pour laquelle nous souhaitons générer des variables aléatoires.
- La fonction majorante $g(x)$ doit avoir les propriétés suivantes :
  - (1) $g(x) \geq f\ (x)$;
  - (2) $m = \int g(x)\, dx < \infty$;
  - (3) la variable aléatoire $Y \sim g(x) / m$ est facilement générée.
- La méthode d'acceptation-rejet pour générer une variable aléatoire $X \sim f\ (x)$ peut être résumée comme suit :

(1) Générer Y ayant une fonction de densité $g(x) / m$;
(2) Générer $U \sim U(0,1)$, indépendant de Y ;
(3) Si $U \leq f(Y) / g(Y)$ alors retournez $X = Y$, sinon revenez à l'étape (1).

Algorithmes de génération de variables aléatoires

Uniforme : $\mathbf{X \sim Uniform\ (a, b)}$

La fonction de répartition d'un variable aléatoire $\mathbf{X \sim \boldsymbol{Uniform}\ (\boldsymbol{a}, \boldsymbol{b})}$est facilement inversée. Ainsi, nous pouvons utiliser la méthode de transformation inverse pour générer X :

Algorithme pour générer des valeurs d'une variable aléatoire $\mathbf{X \sim \boldsymbol{Uniform}\ (\boldsymbol{a}, \boldsymbol{b})}$:
(1) Générer une valeur $U \sim \mathcal{U}\ (0,1)$
(2) Calculer $X = \mathrm{a} + (\mathrm{b} - \mathrm{a})\mathrm{U}$
(3) Retourner X

Exponentielle $\mathbf{X \sim Exp\ (\lambda)}$

- Comme expliqué précédemment, la fonction de répartition de la loi exponentielle a une fonction inverse connue. L'algorithme basé sur la transformation inverse pour la génération d'une variable $\mathbf{X \sim \boldsymbol{Exp}\ (\boldsymbol{\lambda})}$ et le suivant :

Algorithme pour générer des valeurs d'une variable aléatoire exponentielle : $\mathbf{X \sim \boldsymbol{Exp}\ (\boldsymbol{\lambda})}$:
(1) Générer une valeur $\mathrm{U} \sim \mathcal{U}\ (0,1)$
(2) Calculer $X = -\frac{\ln(\mathrm{U})}{\lambda}$
(3) Retourner X

Algorithmes de génération de variables aléatoires

Erlang $\mathbf{X \sim Erlang\ (k, \theta)}$

- Une variable aléatoire Erlang-k $X \sim Erlang\ (k, \theta)$ de moyenne θ est définie comme $X = \sum Y_i$, pour $i = 1 \sim k$, où les Y_i sont des variables aléatoires exponentielles indépendantes et identiquement distribuées (IID) de moyenne θ/k.

- $X \sim Erlang\ (k, \theta)$ peut être générée par la somme de k variables aléatoires exponentielles IID (convolution methode).

Algorithme pour générer des valeurs d'une variable aléatoire Erlang : $X \sim Erlang\ (k, \theta)$

(1) Générer une valeur $U, U_2, \ldots, U_k$ distribuées suivant la loi $\mathcal{U}\ (0,1)$

(2) Calculer $x_i = \frac{-\theta}{k} \ln\left(\prod_{i=1}^{k} U_i\right)$

(3) Retourner x_i

Weibull (α, β)

- La fonction de distribution de $Weibull\ (\alpha, \beta)$ est facilement inversée, donc la méthode de transformation inverse est appliquée pour générer des variables distribuées suivants cette loi.

Algorithme pour générer des valeurs d'une variable aléatoire Weibull : $X \sim$ Weibull (α, β)

(1) Générer une valeur $u_1, u_2, \ldots u_k$, de U $\sim \mathcal{U}\ (0,1)$

(2) Calculer $X = \beta(-\ln U)^{1/\alpha}$

(3) Retourner X

Algorithmes de génération de variables aléatoires

X ~ Beta (α, β) : Acceptance-rejection technique

- Les méthodes de de transformation inverse et de convolution ne s'appliquent pas à la distribution bêta Beta (α, β), la méthode d'acceptation-rejet est utilisée pour générer une variable aléatoire bêta.
- Appliquer la méthode d'acceptation-rejet pour la génération d'une variable aléatoire bêta n'est pas un problème trivial, et il existe un grand nombre de méthodes de génération de variables aléatoires bêta.
- Ci-dessous la méthode de base pour générer une variable aléatoire bêta standard présentée dans Cheng [1978].

0. Initialization:
   - $A = \alpha + \beta$;
   - *If* $\min(\alpha, \beta) \le 1$ *then* $B = \max(\alpha^{-1}, \beta^{-1})$ *else* $B = \sqrt{(A-2)/(2\alpha\beta - A)}$;
   - $C = \alpha + B^{-1}$;
1. Generate: $U_1 \sim U(0,1) \& U_2 \sim U(0,1)$.
2. Set: $V = B \log[U_1/(1-U_1)]$; $W = \alpha \cdot e^{v}$.
3. If $\{A \cdot \log[A/(\beta+W)] + C \cdot V - \log 4\} < \log(U_1^2 U_2)$ then go to step (1); // rejection.
4. $Y = W/(\beta + W)$.
5. Return $X = Y(1+Y)$

Remarque :

La distribution bêta standard $Y \sim Beta(\alpha, \beta)$ a une plage unitaire [0, 1]. Alors la variable aléatoire bêta X avec une plage générale $[a, b]$ peut être obtenue à partir de Y comme suit : $X = a + X(b-a)$.

Normale (μ , σ^2)

- La méthode Box & Muller est une méthode utilisée pour générer une variable aléatoire normale :

1. Generate $U_1 \sim U(0,1)$ & $U_2 \sim U(0,1)$.
2. Compute $Z_1 = (-2 \ln U_1)^{1/2} \cos(2\pi U_2)$; $Z_2 = (-2\ln U_1)^{1/2} \sin(2\pi U_2)$.
3. Return $X_1 = \hat{\mu} + \hat{\sigma} \cdot Z_1$; $X_2 = \hat{\mu} + \hat{\sigma} \cdot Z_2$.

Lognormal (μ , σ^2)

- Une propriété particulière de la distribution lognormale, à savoir que si Y ~ Nromal(μ , σ^2) alors e^Y ~ Lognoromal(μ , σ^2) . L'algorithme pour générer X ~ Lognoromal(μ , σ^2) :

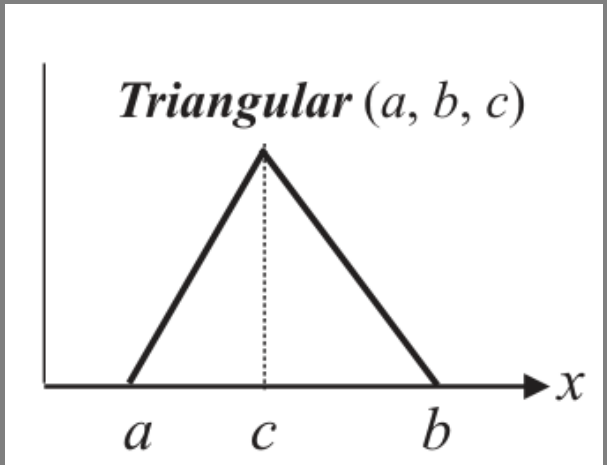
(1) Générez Y ~ Nromal(μ , σ^2)
(2) Retournez $X = e^Y$

$X \sim Triangulaire\ (a, b, c)$

Si le mode c est également donné en plus de la plage $[a, b]$, les temps de service peuvent être échantillonnés à partir d'une distribution triangulaire:

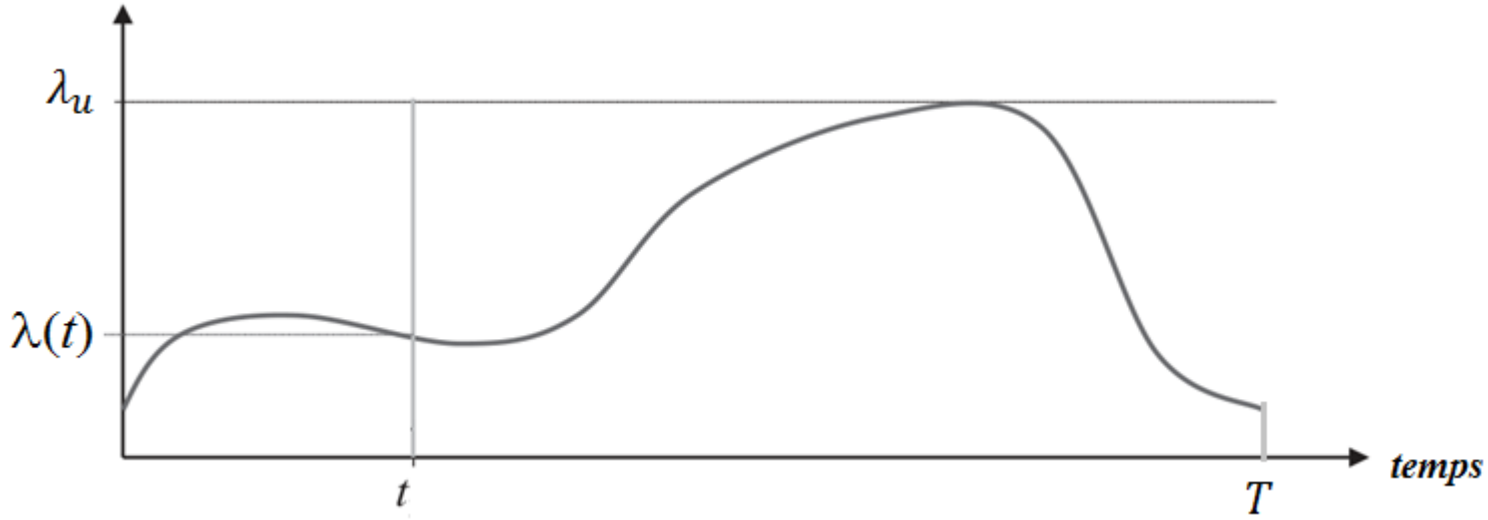
Génération des temps de service à partir d'une distribution triangulaire $X \sim Triangulaire\ (a, b, c)$:

1. $p = (c - a)/(b - a)$
2. Générer $U_1 \sim \mathcal{U}\ (0,1)$, $U_2 \sim \mathcal{U}\ (0,1)$
3. Si $U_1 \leq p$, alors $X = a + (c - a)\sqrt{U_2}$, $Sinon\ X = c + (b - c)(1 - \sqrt{1 - U_2})$



Génération des temps inter-arrivées pour des taux d'arrivée variables: *Thinning method*

- Supposons que les temps entre les arrivées sont distribués de manière exponentielle, mais que le taux d'arrivée $\lambda(t)$ évolue dans le temps.
- Ce processus d'arrivée est appelé ***processus de Poisson non stationnaire (non homogène)***, de **fonction d'intensité $\boldsymbol{\lambda(t)}$,**
- Les processus d'arrivées ***non homogène*** est fréquent dans de nombreux systèmes de services, entre autres: L'arrivée des clients dans une banque, les cafétérias, l'arrivée de supporters dans un stade de football, l'arrivée d'e-mails sur un serveur de messagerie, etc.



Génération des temps inter-arrivées pour des taux d'arrivée variables: *Thinning method*

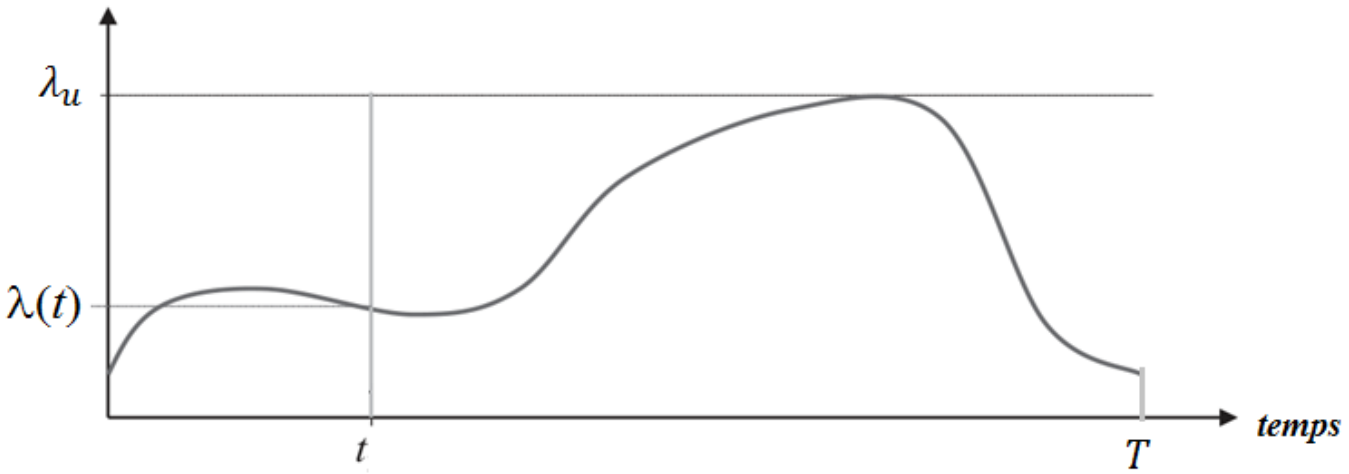
- La méthode la plus populaire pour générer pour générer des temps inter-arrivées non homogènes est la méthode par amincissement « ***Thinning method*** » ,

Théorème.

Considérons un processus de Poisson non homogène d'intensité $\tilde{\lambda}$. $\forall t \in [0,T], 0 \leq \lambda(t) \leq \lambda_u$, $//\lambda_u = max_{t\in[0,T]}\,\lambda(t)$

Soient $t_0, t_1,\ldots,t_n \in [0,T]$ des valeurs aléatoires qui représentent les temps d'évennements du processus de Poisson d'intensité λ_u.

Supposons que $\forall t_i \in [0,T]$, on supprime le point t_i avec la probabilité $1 - \frac{\lambda(t_i)}{\lambda_u}$, alors les points qui restent forment un processus de Poisson non homogène d'intensité $\lambda(t)$.



Génération des temps inter-arrivées pour des taux d'arrivée variables: *Thinning method*

Algorithme de la méthode Thinning:

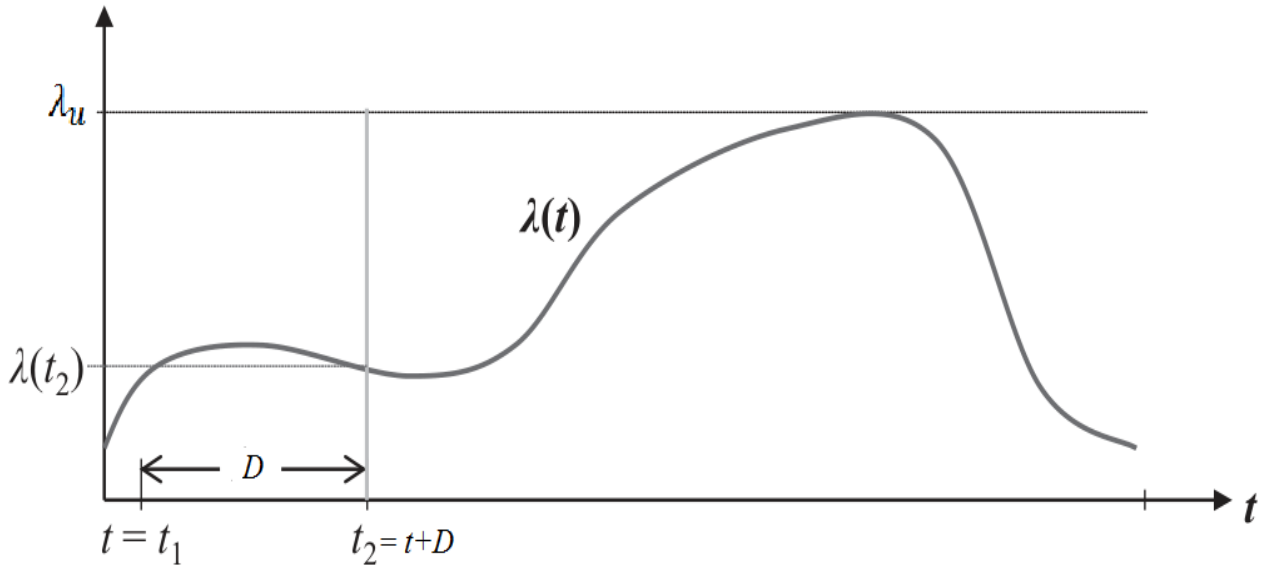
t: *Temps de l'horloge*

$\lambda(t)$: *Fonction d'intensité du processus de Poisson non homogène*

λ_u: $max_{t\in[0,T]}\,\lambda(t)$

1. Générer $U_1 \sim \mathcal{U}\,(0,1)$
2. $D \leftarrow -\left(\frac{1}{\lambda_u}\right) ln\, U_1$
3. $t \leftarrow t + D$
4. Générer $U_2 \sim \mathcal{U}\,(0,1)$
5. Si $U_2 \leq \frac{\lambda(t)}{\lambda_u}$ retourner t

   Sinon, Revenir à l'étape 1



Génération de variables aléatoire dans le langage Python

- La bibliothèque standard `random` de Python propose des fonctions pour générer des nombres distribués suivant différentes loi théoriques.

Fonction	Description
random()	Retourne un nombre aléatoire uniforme dans l'intervalle [0.0, 1.0).
random.uniform(a, b)	Retourne un nombre aléatoire uniforme dans l'intervalle [a, b).
random.Expovariate (lambda)	Retourne un nombre aléatoire avec une distribution de probabilité exponentielle, de parametre lamda
random.triangular(low,hgh, mode)	Retourne un nombre aléatoire dans l'intervalle [low, high], avec une probabilité de pic à la valeur du mode.
random.betavariate(alpha, beta)	Retourne un nombre aléatoire dans l'intervalle [0.0, 1.0) avec une distribution de probabilité bêta de paramètres alpha, beta .
random.gammavariate(alpha, beta)	Retourne un nombre aléatoire avec une distribution de probabilité gamma de paramètres de forme et d'échelle, alpha et beta.
random.lognormvariate(mu, sigma)	Retourne un nombre aléatoire avec une distribution de probabilité log-normale.
random.normalvariate(mu, sigma)	Retourne un nombre aléatoire avec une distribution normale de moyenne mu et d'écart-type sigma.
random.weibullvariate(alpha,beta)	Retourne un nombre aléatoire en virgule flottante avec une distribution de probabilité de Weibull spécifiée.
….	

Génération de variables aléatoire dans le langage Python

```
import random
print("random():", random.random()) # random()
a, b = 1, 10
print(f"uniform({a}, {b}):", random.uniform(a, b)) # random.uniform(a, b)
lambda_param = 2.0 # random.expovariate(lambda)
print(f"expovariate({lambda_param}):", random.expovariate(lambda_param))
low, high, mode = 0, 10, 5# random.triangular(low, high, mode)
print(f"triangular({low}, {high}, {mode}):", random.triangular(low, high, mode))
alpha, beta = 2, 5 # random.betavariate(alpha, beta)
print(f"betavariate({alpha}, {beta}):", random.betavariate(alpha, beta))
alpha, beta = 2, 2 # random.gammavariate(alpha, beta)
print(f"gammavariate({alpha}, {beta}):", random.gammavariate(alpha, beta))
mu, sigma = 0, 0.1 # random.lognormvariate(mu, sigma)
print(f"lognormvariate({mu}, {sigma}):", random.lognormvariate(mu, sigma))
mu, sigma = 0, 1# random.normalvariate(mu, sigma)
print(f"normalvariate({mu}, {sigma}):", random.normalvariate(mu, sigma))
alpha, beta = 2, 1 # random.weibullvariate(alpha, beta)
print(f"weibullvariate({alpha}, {beta}):", random.weibullvariate(alpha, beta))
```

Bibliographie

1. Fishman, G. S. (2001). *Discrete-event simulation: modeling, programming, and analysis* (Vol. 537). New York: Springer.
2. Choi, B. K., & Kang, D. (2013). *Modeling and simulation of discrete event systems*. John Wiley & Sons.
3. Ross, S. M. (2022). *Simulation*. Academic Press.
4. Law, A. M., Kelton, W. D., & Kelton, W. D. (2007). *Simulation modeling and analysis* (Vol. 3). New York: Mcgraw-hill.
5. Rossetti, M. D. (2015). *Simulation modeling and Arena*. John Wiley & Sons.
6. Zeigler, B. P., Muzy, A., & Kofman, E. (2018). *Theory of modeling and simulation: discrete event & iterative system computational foundations*. Academic press