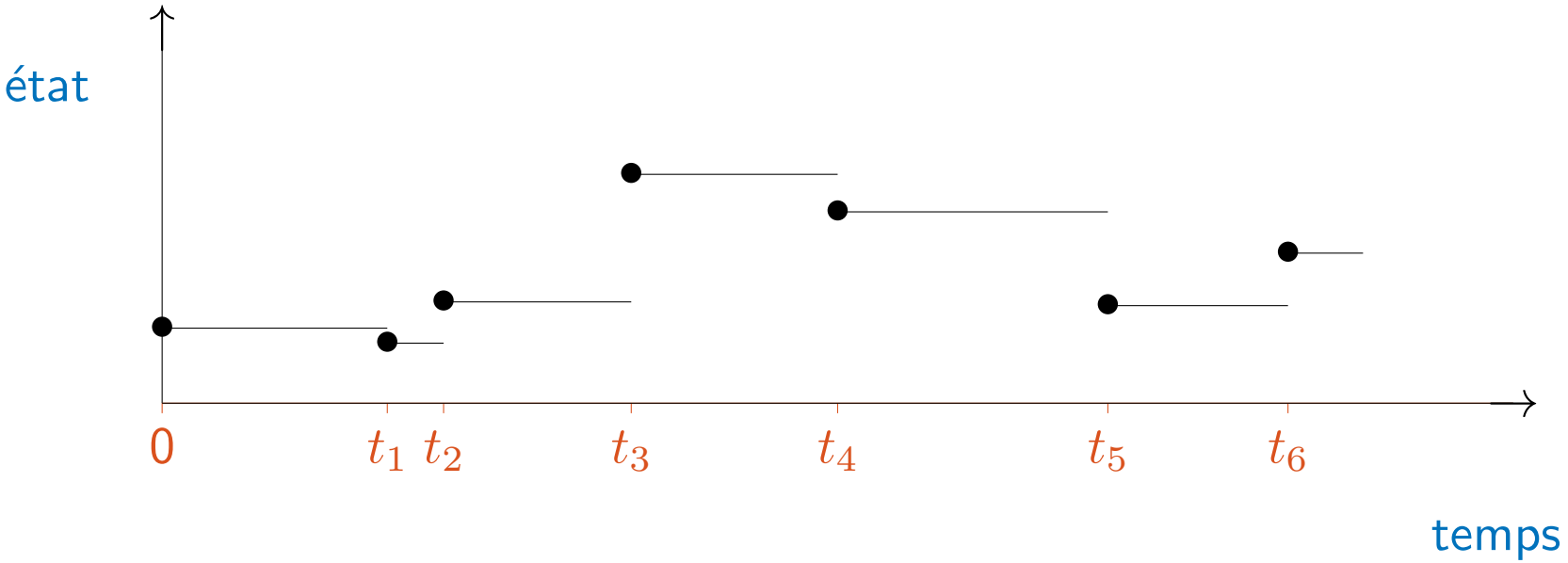
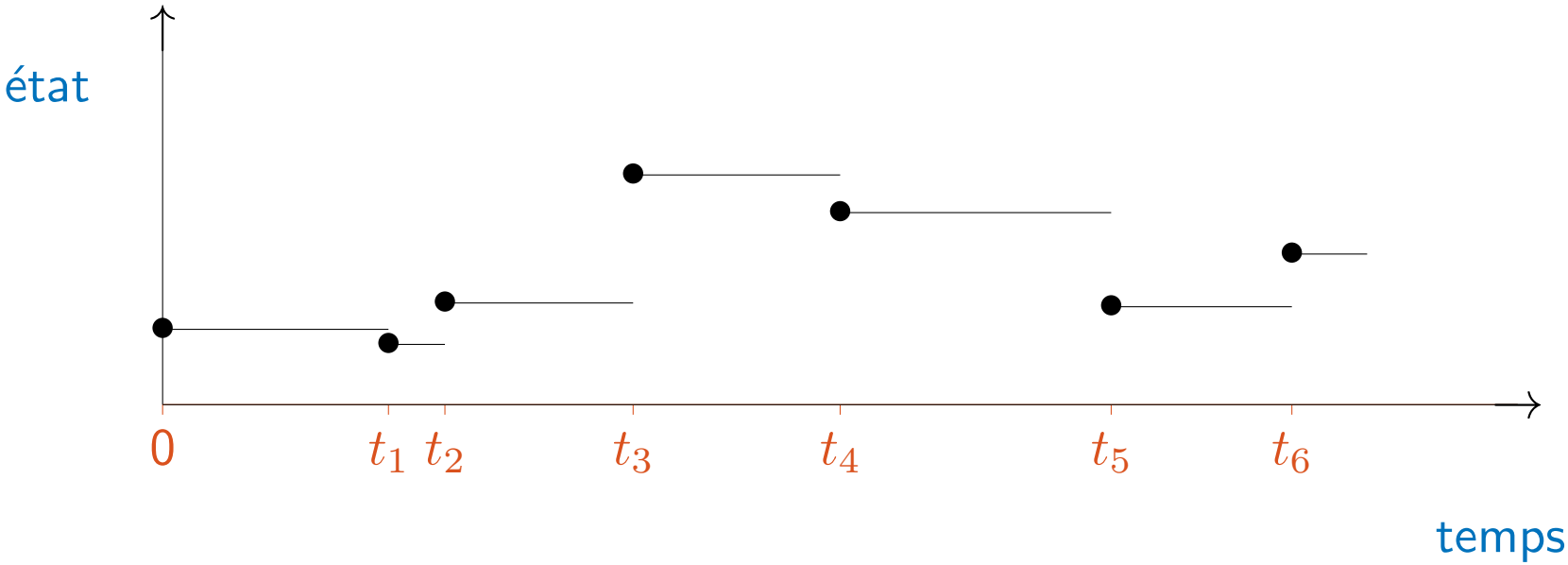
Simulation à événements discrets

Des événements $e_0, e_1, e_2, \ldots$ surviennent aux instants $0 = t_0 \leq t_1 \leq t_2 \leq \cdots$.
On note $\mathcal{S}_i$ l'état du système immédiatement après e_i.
Temps de la simulation: valeur courante de t_i.



Simulation à événements discrets

Des événements $e_0, e_1, e_2, \ldots$ surviennent aux instants $0 = t_0 \leq t_1 \leq t_2 \leq \cdots$.
On note $\mathcal{S}_i$ l'état du système immédiatement après e_i.
Temps de la simulation: valeur courante de t_i.



$(t_i, \mathcal{S}_i)$ doit contenir assez d'information pour poursuivre la simulation (sauf les valeurs des variables aléatoires générées lors des événements e_j pour $j > i$).

Exemple: Une file d'attente à un serveur.

File $GI/G/1$: Un seul serveur, les clients arrivent un à un et sont servis un à la fois, en ordre FIFO.
$S_j =$ durée de service du client j;
$A_j =$ durée entre les arrivées des clients $j-1$ et j.

Exemple: Une file d'attente à un serveur.

File $GI/G/1$: Un seul serveur, les clients arrivent un à un et sont servis un à la fois, en ordre FIFO.
$S_j =$ durée de service du client j;
$A_j =$ durée entre les arrivées des clients $j-1$ et j.
Les S_j et les A_j sont mutuellement indép. de fonc. de répart. G et F.
Le premier client arrive au temps A_1 dans un système vide.

Exemple: Une file d'attente à un serveur.

File $GI/G/1$: Un seul serveur, les clients arrivent un à un et sont servis un à la fois, en ordre FIFO.
$S_j =$ durée de service du client j;
$A_j =$ durée entre les arrivées des clients $j-1$ et j.
Les S_j et les A_j sont mutuellement indép. de fonc. de répart. G et F.
Le premier client arrive au temps A_1 dans un système vide.

On veut simuler ce système pour une durée T et calculer l'attente moyenne par client et la longueur moyenne de la file d'attente.

Exemple: Une file d'attente à un serveur.

File $GI/G/1$: Un seul serveur, les clients arrivent un à un et sont servis un à la fois, en ordre FIFO.
$S_j =$ durée de service du client j;
$A_j =$ durée entre les arrivées des clients $j-1$ et j.
Les S_j et les A_j sont mutuellement indép. de fonc. de répart. G et F.
Le premier client arrive au temps A_1 dans un système vide.

On veut simuler ce système pour une durée T et calculer l'attente moyenne par client et la longueur moyenne de la file d'attente.

Les types d'événements: arrivée, départ, fin de la simulation.

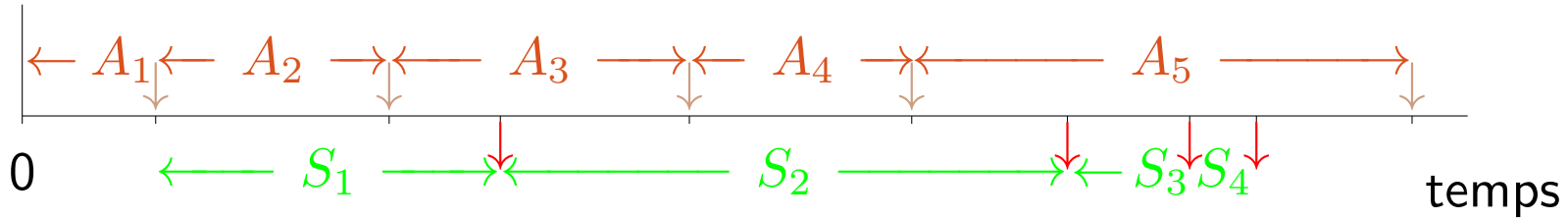
Exemple: Une file d'attente à un serveur.

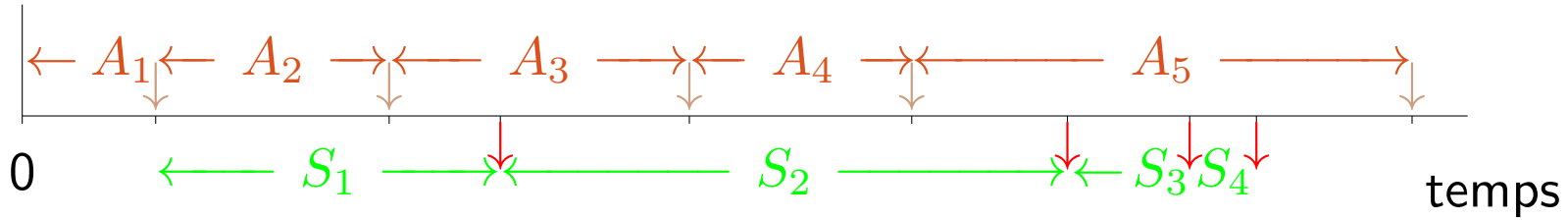
File $GI/G/1$: Un seul serveur, les clients arrivent un à un et sont servis un à la fois, en ordre FIFO.
$S_j =$ durée de service du client j;
$A_j =$ durée entre les arrivées des clients $j-1$ et j.
Les S_j et les A_j sont mutuellement indép. de fonc. de répart. G et F.
Le premier client arrive au temps A_1 dans un système vide.

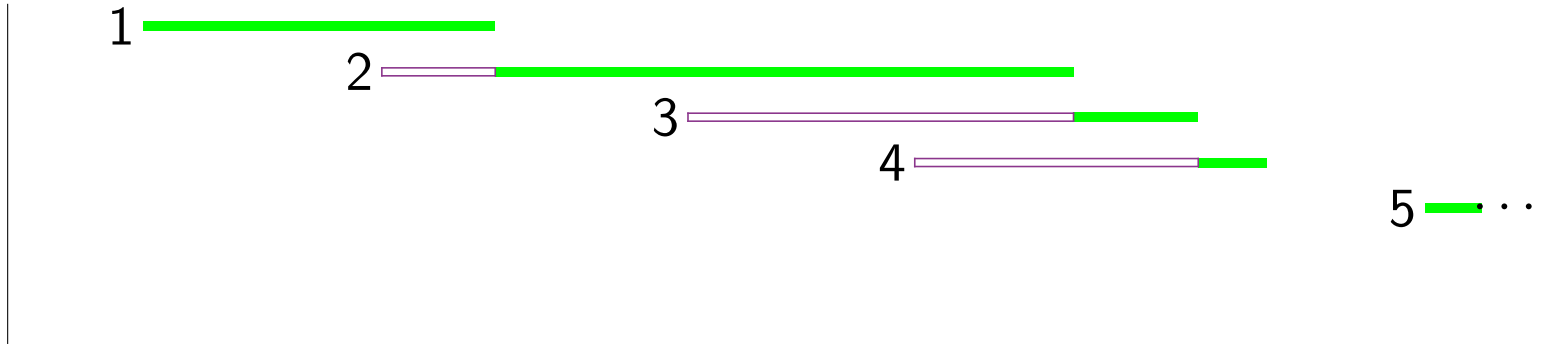
On veut simuler ce système pour une durée T et calculer l'attente moyenne par client et la longueur moyenne de la file d'attente.

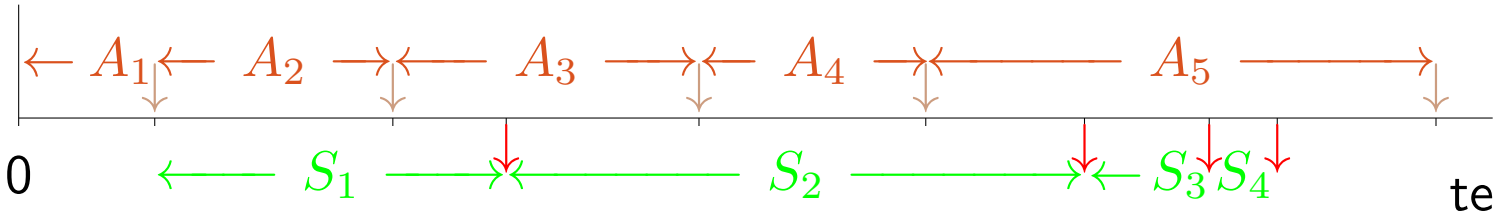
Les types d'événements: arrivée, départ, fin de la simulation.

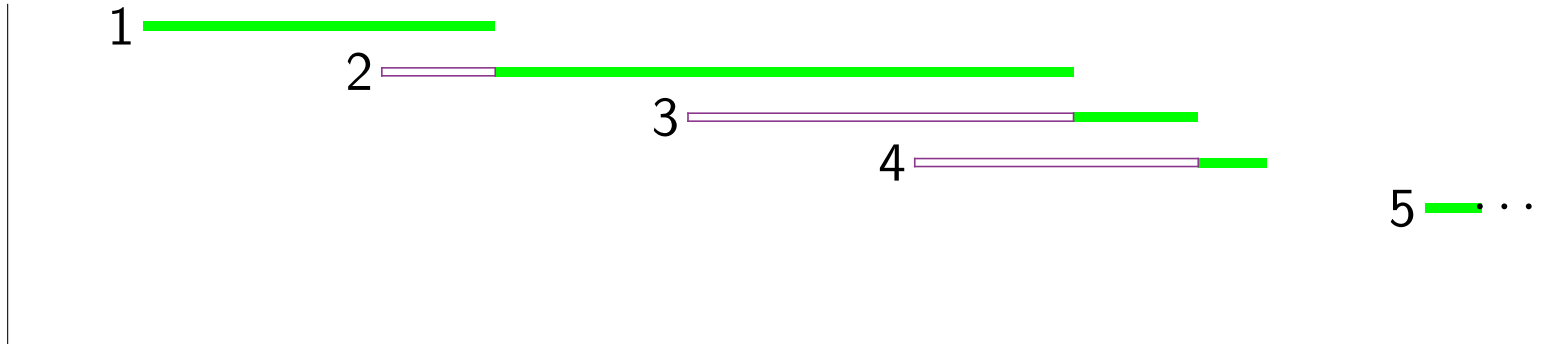
Les variables aléatoires (indépendentes) à générer:
$A_1, A_2, A_3, \ldots$ et $S_1, S_2, S_3, \ldots$.

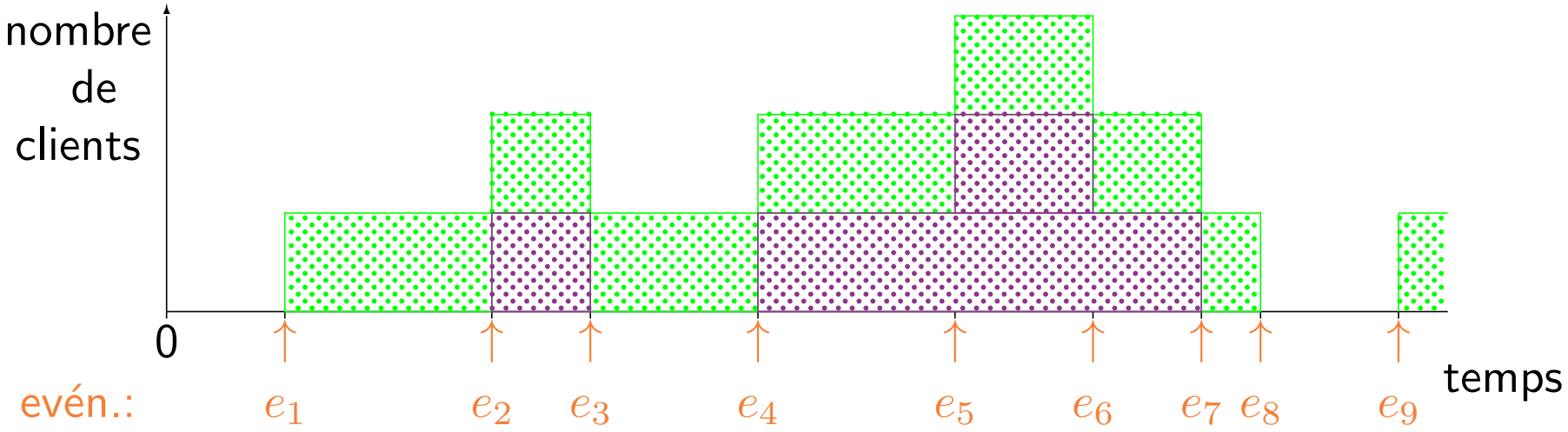

A_1
A_2
A_3
A_4
A_5
0
S_1
S_2
S_3
S_4
temps


A_1
A_2
A_3
A_4
A_5
0
S_1
S_2
S_3
S_4
temps


1
2
3
4
5


A_1
A_2
A_3
A_4
A_5
0
S_1
S_2
S_3
S_4
temps


1
2
3
4
5


nombre
de
clients
0
temps
evén.:
e_1
e_2
e_3
e_4
e_5
e_6
e_7
e_8
e_9

Notation:
W_j = temps d'attente du client j;
$Q(t)$ = longueur de la file d'attente au temps t;
$N_c(t)$ = nombre de clients ayant débuté leur service durant $[0, t]$.

Notation:
W_j = temps d'attente du client j;
$Q(t)$ = longueur de la file d'attente au temps t;
$N_c(t)$ = nombre de clients ayant débuté leur service durant $[0, t]$.

Supposons que l'on veut calculer, pour l'intervalle $[0, T]$:
(a) l'attente moyenne par client:

$$\bar{W}_{N_c(T)} = \frac{1}{N_c(T)} \sum_{j=1}^{N_c(T)} W_j;$$

Notation:
W_j = temps d'attente du client j;
$Q(t)$ = longueur de la file d'attente au temps t;
$N_c(t)$ = nombre de clients ayant débuté leur service durant $[0, t]$.

Supposons que l'on veut calculer, pour l'intervalle $[0, T]$:
(a) l'attente moyenne par client:

$$\bar{W}_{N_c(T)} = \frac{1}{N_c(T)} \sum_{j=1}^{N_c(T)} W_j;$$

(b) la longueur moyenne de la file d'attente:

$$\bar{Q}_T = \frac{1}{T} \int_0^T Q(t) dt.$$

Cette intégrale est facile à calculer (surface fuchsia).

Esquisse d'un programme de simulation à événements discrets:

Pour chaque client, on crée un objet (ou un dossier) contenant l'instant d'arrivée et la durée de service.

Esquisse d'un programme de simulation à événements discrets:

Pour chaque client, on crée un objet (ou un dossier) contenant l'instant d'arrivée et la durée de service.

Variables d'état:

- Horloge de la simulation.
- Liste des clients en attente, liste les clients en service;
- Compteurs statistiques (au besoin).
- Une liste des événements futurs prévus, par ordre chronologique (différentes implantations possibles).

Esquisse d'un programme de simulation à événements discrets:

Pour chaque client, on crée un objet (ou un dossier) contenant l'instant d'arrivée et la durée de service.

Variables d'état:

- Horloge de la simulation.
- Liste des clients en attente, liste les clients en service;
- Compteurs statistiques (au besoin).
- Une liste des événements futurs prévus, par ordre chronologique (différentes implantations possibles).

Une procédure pour chaque type d'événement:

Arrivée:

Générer A selon F et prévoir une autre Arrivée dans A unités de temps;
Créer le nouveau client et noter son instant d'arrivée;
Générer sa durée de service S selon G;
Si (serveur est occupé) **alors**
    Insérer ce nouveau client dans la liste des clients en attente;
sinon
    Insérer le client dans la liste des clients en service;
    Prévoir son départ dans S unités de temps;
Mise à jour des statistiques voulues.

Arrivée:

Générer A selon F et prévoir une autre Arrivée dans A unités de temps;
Créer le nouveau client et noter son instant d'arrivée;
Générer sa durée de service S selon G;
Si (serveur est occupé) **alors**
    Insérer ce nouveau client dans la liste des clients en attente;
sinon
    Insérer le client dans la liste des clients en service;
    Prévoir son départ dans S unités de temps;
Mise à jour des statistiques voulues.

Départ:

Enlever le client de la liste des clients en service;
Si la file d'attente n'est pas vide **alors**
    Enlever le premier client de la liste d'attente;
    L'insérer dans la liste des clients en service;
    Récupérer son S et prévoir son départ dans S unités de temps;
Mise à jour des statistiques voulues.

Arrivée:

Générer A selon F et prévoir une autre Arrivée dans A unités de temps;
Créer le nouveau client et noter son instant d'arrivée;
Générer sa durée de service S selon G;
Si (serveur est occupé) **alors**
    Insérer ce nouveau client dans la liste des clients en attente;
sinon
    Insérer le client dans la liste des clients en service;
    Prévoir son départ dans S unités de temps;
Mise à jour des statistiques voulues.

Départ:

Enlever le client de la liste des clients en service;
Si la file d'attente n'est pas vide **alors**
    Enlever le premier client de la liste d'attente;
    L'insérer dans la liste des clients en service;
    Récupérer son S et prévoir son départ dans S unités de temps;
Mise à jour des statistiques voulues.

Fin-de-la-Simulation:

Imprimer un rapport et terminer le programme.

Pour démarrer la simulation:

Initialiser les variables et les compteurs;
Prévoir la "Fin-de-la-Simulation" au temps T;
Générer A selon F et prévoir la première "Arrivée" dans A heures;
Démarrer l'exécutif.

Exécutif de la simulation:

Répéter: exécuter le prochain événement dans la liste d'événements
jusqu'à: la liste des événements prévus est vide,
ou bien un événement stoppe la simulation.

Pour démarrer la simulation:

Initialiser les variables et les compteurs;
Prévoir la "Fin-de-la-Simulation" au temps T;
Générer A selon F et prévoir la première "Arrivée" dans A heures;
Démarrer l'exécutif.

Exécutif de la simulation:

Répéter: exécuter le prochain événement dans la liste d'événements
jusqu'à: la liste des événements prévus est vide,
ou bien un événement stoppe la simulation.

Les langages et logiciels de simulation offrent des outils pour réaliser cela:

librairies dans un langage tout-usage (Java, C, C++, FORTRAN, ...),
langages de simulation (GPSS, Siman, Simscript II.5, ...),
environnements graphiques "point-click-drag" (Arena, Simul-8, Simio, ...),
environnements spécialisés pour un domaine.

Programme Java utilisant SSJ (version 2005), pour file $M/M/1$:

```
public class QueueEv {

   RandomVariateGen genArr, genServ;
   LinkedList waitList = new LinkedList ();
   LinkedList servList = new LinkedList ();
   Tally custWaits     = new Tally ("Waiting times");
   Accumulate totWait  = new Accumulate ("Size of queue");

   class Customer { double arrivTime, servTime; }

   public QueueEv (double lambda, double mu) {
      genArr = new RandomVariateGen (new MRG32k3a(),
                       new ExponentialDist(lambda));
      genServ = new RandomVariateGen (new MRG32k3a(),
                       new ExponentialDist (mu));
   }

   public void simulateOneRun (double timeHorizon) {
      Sim.init();
      new EndOfSimulation().schedule (timeHorizon);
      new Arrival().schedule (genArr.nextDouble());
      Sim.start();
   }
```

```
class Arrival extends Event {
   public void actions() {
      new Arrival().schedule (genArr.nextDouble()); // Next arrival.
      Customer cust = new Customer();  // Cust just arrived.
      cust.arrivTime = Sim.time();
      cust.servTime = genServ.nextDouble();
      if (servList.size() > 0) {       // Must join the queue.
         waitList.addLast (cust);
         totWait.update (waitList.size());
      } else {                         // Starts service.
         custWaits.add (0.0);
         servList.addLast (cust);
         new Departure().schedule (cust.servTime);
      }
   }
}
```

```
class Departure extends Event {
   public void actions() {
      servList.removeFirst();
      if (waitList.size() > 0) {
         // Starts service for next one in queue.
         Customer cust = (Customer)waitList.removeFirst();
         totWait.update (waitList.size());
         custWaits.add (Sim.time() - cust.arrivTime);
         servList.addLast (cust);
         new Departure().schedule (cust.servTime);
      }
   }
}
```

```
    class EndOfSimulation extends Event {
        public void actions() {
            Sim.stop();
        }
    }

    public static void main (String[] args) {
        QueueEv queue = new QueueEv (1.0/10.0, 1.0/9.0);
        queue.simulateOneRun (1000.0);
        System.out.println (queue.custWaits.report());
        System.out.println (queue.totWait.report());
    }
}
```

Programmation par processus:

```
public class QueueProc {

   Resource server = new Resource (1, "Server");
   RandomVariateGen genArr;
   RandomVariateGen genServ;

   public QueueProc (double lambda, double mu) {
      genArr = new RandomVariateGen (new MRG32k3a(),
                        new ExponentialDist (lambda));
      genServ = new RandomVariateGen (new MRG32k3a(),
                        new ExponentialDist (mu));
   }

   public void simulateOneRun (double timeHorizon) {
      SimProcess.init();
      server.setStatCollecting (true);
      new EndOfSimulation().schedule (timeHorizon);
      new Customer().schedule (genArr.nextDouble());
      Sim.start();
   }
```

```
    class Customer extends SimProcess {
        public void actions() {
            new Customer().schedule (genArr.nextDouble());
            server.request (1);
            delay (genServ.nextDouble());
            server.release (1);
        }
    }

    class EndOfSimulation extends Event {
        public void actions() {
            Sim.stop();
        }
    }

    public static void main (String[] args) {
        QueueProc queue = new QueueProc (1.0/10.0, 1.0/9.0);
        queue.simulateOneRun (1000.0);
        System.out.println (queue.server.report());
    }
}
```

Résultats du programme QueueProc:

```
REPORT ON RESOURCE : Server
  From time : 0.0    to time : 1000.0
                 min          max      average    std. dev.  num. obs.
  Capacity         1            1            1
  Utilisation      0            1        0.999
  Queue Size       0           12        4.850
  Wait             0      113.721       49.554       22.336         97
  Service      0.065       41.021       10.378       10.377         96
  Sojourn     12.828      124.884       60.251       21.352         96
```

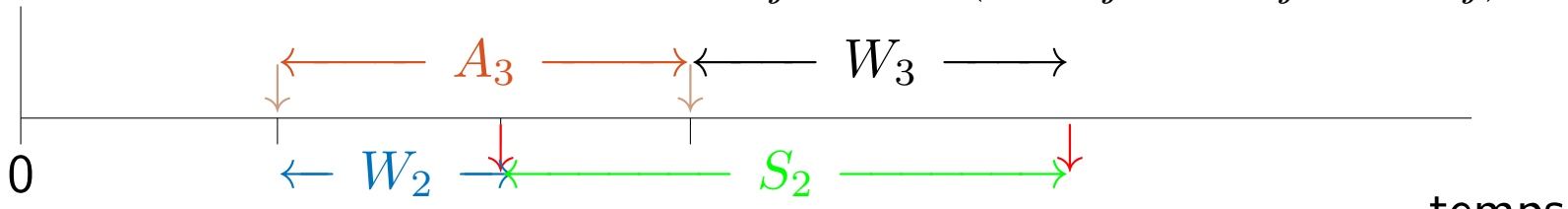
Résultats du programme QueueProc:

```
REPORT ON RESOURCE : Server
  From time : 0.0    to time : 1000.0
                 min          max      average    std. dev.  num. obs.
  Capacity         1            1            1
  Utilisation      0            1        0.999
  Queue Size       0           12        4.850
  Wait             0      113.721       49.554       22.336         97
  Service      0.065       41.021       10.378       10.377         96
  Sojourn     12.828      124.884       60.251       21.352         96
```

Attention: les 97 observations W_j ici ne sont pas indépendantes!
Si on veut calculer un intervalle de confiance pour $w_T = \mathbb{E}[\bar{W}_{N_c(T)}]$, il faut faire plusieurs répétitions indépendantes de cette simulation.

Simulation sans liste d'événements.

Récurrence de Lindley: $W_1 = 0$ et $W_j = \max(0,\ W_{j-1} + S_{j-1} - A_j)$.



Simulation sans liste d'événements.

Récurrence de Lindley: $W_1 = 0$ et $W_j = \max(0,\ W_{j-1} + S_{j-1} - A_j)$.

Ainsi, on peut facilement simuler pour un nombre fixe de clients (au lieu d'un horizon de temps T fixe):

```
public class QueueLindley {
   RandomStream streamArr  = new MRG32k3a();
   RandomStream streamServ = new MRG32k3a();

   public double simulateOneRun (int numCust, double lambda, double mu) {
      double Wj = 0.0;
      double sumWj = 0.0;
      for (int j = 2; j <= numCust; j++) {
         Wj += ExponentialDist.inverseF (mu, streamServ.nextDouble()) -
               ExponentialDist.inverseF (lambda, streamArr.nextDouble());
         if (Wj < 0.0) Wj = 0.0;
         sumWi += Wj;
      }
      return sumWj / numCust;
   }

   public static void main (String[] args) {
      System.out.println ("Average waiting time: " +
         (new QueueLindley()).simulateOneRun (100, 1.0/10.0, 1.0/9.0));
   }
}
```

Formulation comme un problème d'intégration sur $[0,1)^s$:

Formulation comme un problème d'intégration sur $[0,1)^s$:

Supposons que l'on veut estimer $\mathbb{E}[\bar{W}_{100}]$ par $\bar{W}_{100}$.
Il nous faut pouvoir calculer $W_1, \ldots, W_{100}$.
Pour cela il nous faut $S_1, A_2, S_2, \ldots, A_{99}, S_{99}, A_{100}$.
Si on pose $A_j = F^{-1}(U_{2j-2})$ et $S_j = G^{-1}(U_{2j-1})$, alors $\bar{W}_{100} = f(U_1, \ldots, U_{198})$.

Formulation comme un problème d'intégration sur $[0,1)^s$:

Supposons que l'on veut estimer $\mathbb{E}[\bar{W}_{100}]$ par $\bar{W}_{100}$.
Il nous faut pouvoir calculer $W_1, \ldots, W_{100}$.
Pour cela il nous faut $S_1, A_2, S_2, \ldots, A_{99}, S_{99}, A_{100}$.
Si on pose $A_j = F^{-1}(U_{2j-2})$ et $S_j = G^{-1}(U_{2j-1})$, alors $\bar{W}_{100} = f(U_1, \ldots, U_{198})$.

On a donc $s = 198$, i.e.,

$$\mathbb{E}[\bar{W}_{100}] = \int_{[0,1)^{198}} f(\mathbf{u}) d\mathbf{u}.$$

Formulation comme un problème d'intégration sur $[0,1)^s$:

Supposons que l'on veut estimer $\mathbb{E}[\bar{W}_{100}]$ par $\bar{W}_{100}$.
Il nous faut pouvoir calculer $W_1, \ldots, W_{100}$.
Pour cela il nous faut $S_1, A_2, S_2, \ldots, A_{99}, S_{99}, A_{100}$.
Si on pose $A_j = F^{-1}(U_{2j-2})$ et $S_j = G^{-1}(U_{2j-1})$, alors $\bar{W}_{100} = f(U_1, \ldots, U_{198})$.

On a donc $s = 198$, i.e.,

$$\mathbb{E}[\bar{W}_{100}] = \int_{[0,1)^{198}} f(\mathbf{u}) d\mathbf{u}.$$

Par contre, si on veut simuler $\bar{W}_{N_c(T)}$, le nombre de clients $N_c(T)$ est aléatoire et non borné. Dans ce cas, on a $s = \infty$.

Variation statistique

On a exécuté 8 fois la simulation dans `QueueProc`, avec $T = 1000$:

```
public static void main (String[] args) {
   QueueEv q = new QueueEv(1.0/10.0, 1.0/9.0);
   for (int i = 0; i < 8; i++)
      q.simulateOneRun (1000);
      System.out.println ("Average waiting time: " + q.custWaits.average());
      System.out.println ("Average queue length: " + q.totWaits.average());
}
```

Variation statistique

On a exécuté 8 fois la simulation dans `QueueProc`, avec $T = 1000$:

```
public static void main (String[] args) {
   QueueEv q = new QueueEv(1.0/10.0, 1.0/9.0);
   for (int i = 0; i < 8; i++)
      q.simulateOneRun (1000);
      System.out.println ("Average waiting time: " + q.custWaits.average());
      System.out.println ("Average queue length: " + q.totWaits.average());
}
```

Résultats avec $T = 1000$:

Essai	N_c	$\bar{W}_{N_c}$	$\bar{Q}_T$
1	97	49.55	4.85
2	95	27.72	2.64
3	92	44.02	4.93
4	101	27.91	2.82
5	88	27.43	2.41
6	95	60.07	8.40
7	102	20.92	2.49
8	84	24.91	2.09

Puis avec $T = 10000$:

Essai	N_c	$\bar{W}_{N_c}$	$\bar{Q}_T$
1	1033	124.52	12.87
2	997	82.93	8.27
3	998	72.40	7.31
4	1003	45.79	4.59
5	1016	82.67	8.40
6	1003	65.30	6.56
7	974	91.16	9.36
8	976	62.76	6.14

Lorsque T augmente, on note empiriquement que:
(a) la variance semble diminuer et
(b) les moyennes semblent plus élevées.

Valeurs théoriques sur horizon infini

Formule analytique:
Pour un modèle M/M/1, on connaît les moyennes pour $T \to \infty$.
Taux d'arrivée $\lambda = 1/10$. Taux de service $\mu = 1/9$.

Valeurs théoriques sur horizon infini

Formule analytique:
Pour un modèle M/M/1, on connaît les moyennes pour $T \to \infty$.
Taux d'arrivée $\lambda = 1/10$. Taux de service $\mu = 1/9$.
Intensité du trafic: $\rho = \lambda/\mu = 0.9$.

Valeurs théoriques sur horizon infini

Formule analytique:
Pour un modèle M/M/1, on connaît les moyennes pour $T \to \infty$.
Taux d'arrivée $\lambda = 1/10$. Taux de service $\mu = 1/9$.
Intensité du trafic: $\rho = \lambda/\mu = 0.9$.
Nb. moyen dans la file $q \stackrel{\text{def}}{=} \lim_{T\to\infty} \mathbb{E}[\bar{Q}_T] = \rho^2/(1-\rho) = 8.1$.

Valeurs théoriques sur horizon infini

Formule analytique:

Pour un modèle M/M/1, on connaît les moyennes pour $T \to \infty$.

Taux d'arrivée $\lambda = 1/10$. Taux de service $\mu = 1/9$.

Intensité du trafic: $\rho = \lambda/\mu = 0.9$.

Nb. moyen dans la file $q \stackrel{\text{def}}{=} \lim_{T\to\infty} \mathbb{E}[\bar{Q}_T] = \rho^2/(1-\rho) = 8.1$.

Temps moyen d'attente $w \stackrel{\text{def}}{=} \lim_{T\to\infty} \mathbb{E}[\bar{W}_{N_c(T)}] = q/\lambda = \rho^2/((1-\rho)\mu) = 81$.

Valeurs théoriques sur horizon infini

Formule analytique:
Pour un modèle M/M/1, on connaît les moyennes pour $T \to \infty$.
Taux d'arrivée $\lambda = 1/10$. Taux de service $\mu = 1/9$.
Intensité du trafic: $\rho = \lambda/\mu = 0.9$.
Nb. moyen dans la file $q \stackrel{\text{def}}{=} \lim_{T\to\infty} \mathbb{E}[\bar{Q}_T] = \rho^2/(1-\rho) = 8.1$.
Temps moyen d'attente $w \stackrel{\text{def}}{=} \lim_{T\to\infty} \mathbb{E}[\bar{W}_{N_c(T)}] = q/\lambda = \rho^2/((1-\rho)\mu) = 81$.

Comparaison avec les résultats de la simulation: biais!
Pourquoi ?

Valeurs théoriques sur horizon infini

Formule analytique:
Pour un modèle M/M/1, on connaît les moyennes pour $T \to \infty$.
Taux d'arrivée $\lambda = 1/10$. Taux de service $\mu = 1/9$.
Intensité du trafic: $\rho = \lambda/\mu = 0.9$.
Nb. moyen dans la file $q \stackrel{\text{def}}{=} \lim_{T\to\infty} \mathbb{E}[\bar{Q}_T] = \rho^2/(1-\rho) = 8.1$.
Temps moyen d'attente $w \stackrel{\text{def}}{=} \lim_{T\to\infty} \mathbb{E}[\bar{W}_{N_c(T)}] = q/\lambda = \rho^2/((1-\rho)\mu) = 81$.

Comparaison avec les résultats de la simulation: biais!
Pourquoi ?
Le système est initialement vide et $W_1 = 0$.
De plus, les clients en attente au temps T ne sont pas comptés.
Leurs temps d'attente sont en moyenne plus longs.

Valeurs théoriques sur horizon infini

Formule analytique:
Pour un modèle M/M/1, on connaît les moyennes pour $T \to \infty$.
Taux d'arrivée $\lambda = 1/10$. Taux de service $\mu = 1/9$.
Intensité du trafic: $\rho = \lambda/\mu = 0.9$.
Nb. moyen dans la file $q \stackrel{\text{def}}{=} \lim_{T\to\infty} \mathbb{E}[\bar{Q}_T] = \rho^2/(1-\rho) = 8.1$.
Temps moyen d'attente $w \stackrel{\text{def}}{=} \lim_{T\to\infty} \mathbb{E}[\bar{W}_{N_c(T)}] = q/\lambda = \rho^2/((1-\rho)\mu) = 81$.

Comparaison avec les résultats de la simulation: biais!
Pourquoi ?
Le système est initialement vide et $W_1 = 0$.
De plus, les clients en attente au temps T ne sont pas comptés.
Leurs temps d'attente sont en moyenne plus longs.

Par contre, les estimateurs sont consistants lorsque $T \to \infty$.

Échauffement de la simulation.

Supposons que l'on veut vraiment estimer w et q par Monte Carlo.
On peut diminuer le biais en recueillant les statistiques seulement à partir d'un certain temps T_0.
Les estimateurs deviennent:

$$\frac{1}{N_c(T) - N_c(T_0)} \sum_{j=N_c(T_0)+1}^{N_c(T)} W_j$$

et

$$\frac{1}{T - T_0} \int_{T_0}^{T} Q(t)dt.$$

Échauffement de la simulation.

Supposons que l'on veut vraiment estimer w et q par Monte Carlo.
On peut diminuer le biais en recueillant les statistiques seulement à partir d'un certain temps T_0.
Les estimateurs deviennent:

$$\frac{1}{N_c(T) - N_c(T_0)} \sum_{j=N_c(T_0)+1}^{N_c(T)} W_j$$

et

$$\frac{1}{T - T_0} \int_{T_0}^{T} Q(t)dt.$$

Comment choisir T_0?

Échauffement de la simulation.

Supposons que l'on veut vraiment estimer w et q par Monte Carlo.
On peut diminuer le biais en recueillant les statistiques seulement à partir d'un certain temps T_0.
Les estimateurs deviennent:

$$\frac{1}{N_c(T) - N_c(T_0)} \sum_{j=N_c(T_0)+1}^{N_c(T)} W_j$$

et

$$\frac{1}{T - T_0} \int_{T_0}^{T} Q(t)dt.$$

Comment choisir T_0?
Question très difficile en général.
Compromis entre le biais et la variance.

Dans la plupart des applications, le biais diminue comme κ_1/T et la variance comme κ_2/T.

Problème 1: variance élevée.

Problème 1: variance élevée.

Remèdes possibles:

— Plus grand nombre de répétitions; prendre la moyenne.
(Bien sûr, il faut aussi calculer un intervalle de confiance.)

— Utiliser un horizon plus long.

— Utiliser des techniques de réduction de la variance.

Problème 1: variance élevée.
Remèdes possibles:
— Plus grand nombre de répétitions; prendre la moyenne.
(Bien sûr, il faut aussi calculer un intervalle de confiance.)
— Utiliser un horizon plus long.
— Utiliser des techniques de réduction de la variance.

Problème 2: Biais dû à l'état initial.

Problème 1: variance élevée.
Remèdes possibles:
— Plus grand nombre de répétitions; prendre la moyenne.
(Bien sûr, il faut aussi calculer un intervalle de confiance.)
— Utiliser un horizon plus long.
— Utiliser des techniques de réduction de la variance.

Problème 2: Biais dû à l'état initial.
Danger: attention aux intervalles de confiance!
Améliorations possibles:
— Utiliser un horizon plus long.
— Échauffement.

Exemple: Un centre d'appels téléphonique

Ouvert durant m heures par jour.
$n_j =$ nombre d'agents disponibles durant l'heure j.

Exemple: Un centre d'appels téléphonique

Ouvert durant m heures par jour.
$n_j =$ nombre d'agents disponibles durant l'heure j.
$B =$ facteur d'achalandage pour la journée, $\sim \mathrm{Gamma}(\alpha_0, \alpha_0)$;
$\mathbb{E}[B] = 1$, $\mathrm{Var}[B] = 1/\alpha_0$.

Exemple: Un centre d'appels téléphonique

Ouvert durant m heures par jour.
$n_j =$ nombre d'agents disponibles durant l'heure j.
$B =$ facteur d'achalandage pour la journée, $\sim \mathrm{Gamma}(\alpha_0, \alpha_0)$;
$\mathbb{E}[B] = 1$, $\mathrm{Var}[B] = 1/\alpha_0$.
Arrivées des appels: processus de Poisson de taux $B\lambda_j$ durant l'heure j.

Exemple: Un centre d'appels téléphonique

Ouvert durant m heures par jour.
$n_j =$ nombre d'agents disponibles durant l'heure j.
$B =$ facteur d'achalandage pour la journée, $\sim \text{Gamma}(\alpha_0, \alpha_0)$;
$\mathbb{E}[B] = 1$, $\text{Var}[B] = 1/\alpha_0$.
Arrivées des appels: processus de Poisson de taux $B\lambda_j$ durant l'heure j.
Durées de service: i.i.d. $\text{Gamma}(\alpha, \gamma)$.

Exemple: Un centre d'appels téléphonique

Ouvert durant m heures par jour.
$n_j =$ nombre d'agents disponibles durant l'heure j.
$B =$ facteur d'achalandage pour la journée, $\sim \text{Gamma}(\alpha_0, \alpha_0)$;
$\mathbb{E}[B] = 1$, $\text{Var}[B] = 1/\alpha_0$.
Arrivées des appels: processus de Poisson de taux $B\lambda_j$ durant l'heure j.
Durées de service: i.i.d. $\text{Gamma}(\alpha, \gamma)$.
File d'attente FIFO.
Temps de patience: 0 avec prob. p, $\text{Exponentielle}(\nu)$ avec prob. $1 - p$.
Si attente $>$ patience: abandon.

Exemple: Un centre d'appels téléphonique

Ouvert durant m heures par jour.
$n_j =$ nombre d'agents disponibles durant l'heure j.
$B =$ facteur d'achalandage pour la journée, $\sim \text{Gamma}(\alpha_0, \alpha_0)$;
$\mathbb{E}[B] = 1$, $\text{Var}[B] = 1/\alpha_0$.
Arrivées des appels: processus de Poisson de taux $B\lambda_j$ durant l'heure j.
Durées de service: i.i.d. $\text{Gamma}(\alpha, \gamma)$.
File d'attente FIFO.
Temps de patience: 0 avec prob. p, $\text{Exponentielle}(\nu)$ avec prob. $1 - p$.
Si attente $>$ patience: abandon.

On veut estimer, sur horizon infini:
$w =$ temps moyen d'attente par appel.
$g(s) =$ fraction des appels attendant moins de s secondes.
$\ell =$ fraction des appels perdus par abandon.

On simule ce modèle pour n jours. Pour le jour i:
$A_i =$ nombre total d'arrivées,
$W_i =$ attente totale,
$G_i(s) =$ nombre d'appels ayant attendu moins de s secondes,
$L_i =$ nombre d'abandons.

On simule ce modèle pour n jours. Pour le jour i:
$A_i =$ nombre total d'arrivées,
$W_i =$ attente totale,
$G_i(s) =$ nombre d'appels ayant attendu moins de s secondes,
$L_i =$ nombre d'abandons.

Le nombre espéré d'arrivées par jour est $a = \mathbb{E}[A_i] = \mathbb{E}[B] \sum_{j=0}^{m-1} \lambda_j$.

On simule ce modèle pour n jours. Pour le jour i:
A_i = nombre total d'arrivées,
W_i = attente totale,
$G_i(s)$ = nombre d'appels ayant attendu moins de s secondes,
L_i = nombre d'abandons.

Le nombre espéré d'arrivées par jour est $a = \mathbb{E}[A_i] = \mathbb{E}[B] \sum_{j=0}^{m-1} \lambda_j$.
Estimateurs sans biais i.i.d., pour $i = 1, \ldots, n$:

W_i/a pour $w = \mathbb{E}[W_i]/a = \lim_{n\to\infty} \frac{\mathbb{E}[W_1+\cdots+W_n]/n}{\mathbb{E}[A_1+\cdots+A_n]/n}$,
$G_i(s)/a$ pour $g(s) = \mathbb{E}[G_i(s)]/a$,
L_i/a pour $\ell = \mathbb{E}[L_i]/a$.

On simule ce modèle pour n jours. Pour le jour i:
A_i = nombre total d'arrivées,
W_i = attente totale,
$G_i(s)$ = nombre d'appels ayant attendu moins de s secondes,
L_i = nombre d'abandons.

Le nombre espéré d'arrivées par jour est $a = \mathbb{E}[A_i] = \mathbb{E}[B] \sum_{j=0}^{m-1} \lambda_j$.
Estimateurs sans biais i.i.d., pour $i = 1, \ldots, n$:

W_i/a pour $w = \mathbb{E}[W_i]/a = \lim_{n\to\infty} \frac{\mathbb{E}[W_1+\cdots+W_n]/n}{\mathbb{E}[A_1+\cdots+A_n]/n}$,
$G_i(s)/a$ pour $g(s) = \mathbb{E}[G_i(s)]/a$,
L_i/a pour $\ell = \mathbb{E}[L_i]/a$.

On peut utiliser leurs moyennes et leurs variances empiriques pour calculer des intervalles de confiance.

On simule ce modèle pour n jours. Pour le jour i:
A_i = nombre total d'arrivées,
W_i = attente totale,
$G_i(s)$ = nombre d'appels ayant attendu moins de s secondes,
L_i = nombre d'abandons.

Le nombre espéré d'arrivées par jour est $a = \mathbb{E}[A_i] = \mathbb{E}[B] \sum_{j=0}^{m-1} \lambda_j$.
Estimateurs sans biais i.i.d., pour $i = 1, \ldots, n$:

W_i/a pour $w = \mathbb{E}[W_i]/a = \lim_{n\to\infty} \frac{\mathbb{E}[W_1+\cdots+W_n]/n}{\mathbb{E}[A_1+\cdots+A_n]/n}$,
$G_i(s)/a$ pour $g(s) = \mathbb{E}[G_i(s)]/a$,
L_i/a pour $\ell = \mathbb{E}[L_i]/a$.

On peut utiliser leurs moyennes et leurs variances empiriques pour calculer des intervalles de confiance.

Ce modèle simplifie bien sûr beaucoup la réalité.

On simule ce modèle pour n jours. Pour le jour i:
A_i = nombre total d'arrivées,
W_i = attente totale,
$G_i(s)$ = nombre d'appels ayant attendu moins de s secondes,
L_i = nombre d'abandons.

Le nombre espéré d'arrivées par jour est $a = \mathbb{E}[A_i] = \mathbb{E}[B] \sum_{j=0}^{m-1} \lambda_j$.
Estimateurs sans biais i.i.d., pour $i = 1, \ldots, n$:

W_i/a pour $w = \mathbb{E}[W_i]/a = \lim_{n\to\infty} \frac{\mathbb{E}[W_1+\cdots+W_n]/n}{\mathbb{E}[A_1+\cdots+A_n]/n}$,
$G_i(s)/a$ pour $g(s) = \mathbb{E}[G_i(s)]/a$,
L_i/a pour $\ell = \mathbb{E}[L_i]/a$.

On peut utiliser leurs moyennes et leurs variances empiriques pour calculer des intervalles de confiance.

Ce modèle simplifie bien sûr beaucoup la réalité.

Comment programmer cela?

On simule ce modèle pour n jours. Pour le jour i:
A_i = nombre total d'arrivées,
W_i = attente totale,
$G_i(s)$ = nombre d'appels ayant attendu moins de s secondes,
L_i = nombre d'abandons.

Le nombre espéré d'arrivées par jour est $a = \mathbb{E}[A_i] = \mathbb{E}[B] \sum_{j=0}^{m-1} \lambda_j$.
Estimateurs sans biais i.i.d., pour $i = 1, \ldots, n$:

W_i/a pour $w = \mathbb{E}[W_i]/a = \lim_{n\to\infty} \frac{\mathbb{E}[W_1+\cdots+W_n]/n}{\mathbb{E}[A_1+\cdots+A_n]/n}$,
$G_i(s)/a$ pour $g(s) = \mathbb{E}[G_i(s)]/a$,
L_i/a pour $\ell = \mathbb{E}[L_i]/a$.

On peut utiliser leurs moyennes et leurs variances empiriques pour calculer des intervalles de confiance.

Ce modèle simplifie bien sûr beaucoup la réalité.

Comment programmer cela?
Pour de plus gros modèles et des logiciels plus généraux: plus modulaire.

```
public class CallCenter {

   static final double HOUR = 3600.0;  // Time is in seconds.

   // Data
   // Arrival rates are per hour, service and patience times are in seconds.
   double openingTime;    // Opening time of the center (in hours).
   int numPeriods;        // Number of working periods (hours) in the day.
   int[] numAgents;       // Number of agents for each period.
   double[] lambda;       // Base arrival rate lambda_j for each j.
   double alpha0;         // Parameter of gamma distribution for B.
   double p;              // Probability that patience time is 0.
   double nu;             // Parameter of exponential for patience time.
   double alpha, gamma;   // Parameters of gamma service time distribution.
   double s;              // Want stats on waiting times smaller than s.

   // Variables
   double busyness;       // Current value of B.
   double arrRate = 0.0;  // Current arrival rate.
   int nAgents;           // Number of agents in current period.
   int nBusy;             // Number of agents occupied;
   int nArrivals;         // Number of arrivals today;
   int nAbandon;          // Number of abandonments during the day.
   int nGoodQoS;          // Number of waiting times less than s today.
   double nCallsExpected; // Expected number of calls per day.
```

```
Event nextArrival = new Arrival();           // The next Arrival event.
LinkedList waitList = new LinkedList ();

RandomStream streamB         = new MRG32k3a(); // For B.
RandomStream streamArr       = new MRG32k3a(); // For arrivals.
RandomStream streamPatience = new MRG32k3a(); // For patience times.
GammaGen genServ;   // For service times; created in constructor().

Tally statArrivals = new Tally ("Number of arrivals per day");
Tally statWaits    = new Tally ("Average waiting time per customer");
Tally statWaitsDay = new Tally ("Waiting times within a day");
Tally statGoodQoS  = new Tally ("Proportion of waiting times < s");
Tally statAbandon  = new Tally ("Proportion of calls lost");

public CallCenter (String fileName) throws IOException {
   readData (fileName);
   // genServ can be created only after its parameters are read.
   // The acceptanc/rejection method is much faster than inversion.
   genServ = new GammaAcceptanceRejectionGen (new MRG32k3a(),
                  new GammaDist (alpha, gamma));
}

// Reads data and construct arrays.
public void readData (String fileName) throws IOException {
   ...
}
```

```
class Call {                                    // A phone call.
   double arrivalTime, serviceTime, patienceTime;

   public Call() {
      serviceTime = genServ.nextDouble(); // Generate service time.
      if (nBusy < nAgents) {                    // Start service immediately.
         nBusy++;
         nGoodQoS++;
         statWaitsDay.add (0.0);
         new CallCompletion().schedule (serviceTime);
      } else {                                  // Join the queue.
         patienceTime = generPatience();
         arrivalTime = Sim.time();
         waitList.addLast (this);
      }
   }
   public void endWait() {
      double wait = Sim.time() - arrivalTime;
      if (patienceTime < wait) {            // Caller has abandoned.
         nAbandon++;
         wait = patienceTime;               // Effective waiting time.
      }
      else {
         nBusy++;
         new CallCompletion().schedule (serviceTime);
      }
      if (wait < s) nGoodQoS++;
      statWaitsDay.add (wait);
   }
}
```

```
// Event: A call arrives.
class Arrival extends Event {
   public void actions() {
      nextArrival.schedule
         (ExponentialDist.inverseF (arrRate, streamArr.nextDouble()));
      nArrivals++;
      Call call = new Call();                  // Call just arrived.
   }
}

// Event: A call is completed.
class CallCompletion extends Event {
   public void actions() { nBusy--;   checkQueue(); }
}

// Start answering new calls if agents are free and queue not empty.
public void checkQueue() {
   while ((waitList.size() > 0) && (nBusy < nAgents))
      ((Call)waitList.removeFirst()).endWait();
}

// Generates the patience time for a call.
public double generPatience() {
   double u = streamPatience.nextDouble();
   if (u <= p)
      return 0.0;
   else
      return ExponentialDist.inverseF (nu, (1.0-u) / (1.0-p));
}
```

```
// Event: A new period begins.
class NextPeriod extends Event {
   int j;     // Number of the new period.
   public NextPeriod (int period) { j = period; }
   public void actions() {
      if (j < numPeriods) {
         nAgents = numAgents[j];
         arrRate = busyness * lambda[j] / HOUR;
         if (j == 0)
            nextArrival.schedule (ExponentialDist.inverseF
                                  (arrRate, streamArr.nextDouble()));
         else {
            checkQueue();
            nextArrival.reschedule ((nextArrival.time() - Sim.time())
                                    * lambda[j-1] / lambda[j]);
         }
         new NextPeriod(j+1).schedule (1.0 * HOUR);
      }
      else
         nextArrival.cancel();  // End of the day.
   }
}
```

```
   public void simulateOneDay (double busyness) {
      Sim.init();          statWaitsDay.init();
      nArrivals = 0;       nAbandon = 0;
      nGoodQoS = 0;        nBusy = 0;
      this.busyness = busyness;

      new NextPeriod(0).schedule (openingTime * HOUR);
      Sim.start();
      // Here the simulation is running...

      statArrivals.add ((double)nArrivals);
      statAbandon.add ((double)nAbandon / nCallsExpected);
      statGoodQoS.add ((double)nGoodQoS / nCallsExpected);
      statWaits.add (statWaitsDay.sum() / nCallsExpected);
   }

   public void simulateOneDay () {
      simulateOneDay (GammaDist.inverseF (alpha0, alpha0, 8,
                                          streamB.nextDouble()));
   }

   static public void main (String[] args) throws IOException {
      CallCenter cc = new CallCenter ("CallCenter.dat");
      for (int i = 1; i <= 1000; i++)  cc.simulateOneDay();
      System.out.println ("Num. calls expected = " + cc.nCallsExpected);
      System.out.println (
         cc.statArrivals.reportAndCIStudent (0.9) +
         cc.statWaits.reportAndCIStudent (0.9) +
         cc.statGoodQoS.reportAndCIStudent (0.9) +
         cc.statAbandon.reportAndCIStudent (0.9));
   }
}
```

Valeurs aléatoires communes

Idée: pour comparer deux (ou plusieurs) systèmes semblables, utiliser les mêmes nombres aléatoires uniformes aux mêmes endroits pour tous les systèmes.

Valeurs aléatoires communes

Idée: pour comparer deux (ou plusieurs) systèmes semblables, utiliser les mêmes nombres aléatoires uniformes aux mêmes endroits pour tous les systèmes.

Supposons que $\mu_1 = \mathbb{E}[X_1]$ et $\mu_2 = \mathbb{E}[X_2]$.
On veut estimer $\mu_2 - \mu_1 = \mathbb{E}[X_2 - X_1]$.

Valeurs aléatoires communes

Idée: pour comparer deux (ou plusieurs) systèmes semblables, utiliser les mêmes nombres aléatoires uniformes aux mêmes endroits pour tous les systèmes.

Supposons que $\mu_1 = \mathbb{E}[X_1]$ et $\mu_2 = \mathbb{E}[X_2]$.
On veut estimer $\mu_2 - \mu_1 = \mathbb{E}[X_2 - X_1]$.

On simulant X_1 et X_2 avec les mêmes nombres aléatoires, on ne change pas leurs lois de probabilité individuelles, mais on peut induire une covariance positive entre les deux. On a

$$\mathrm{Var}[X_2 - X_1] = \mathrm{Var}[X_2] + \mathrm{Var}[X_1] - 2\,\mathrm{Cov}[X_1, X_2].$$

Exemple du centre d'appels.

Prenons $\alpha_0 = 10$, $p = 0.1$, $\nu = 0.001$, $\alpha = 1.0$, $\gamma = 0.01$, $s = 20$, 13 périodes, et

j	0	1	2	3	4	5	6	7	8	9	10	11	12
n_j	4	6	8	8	8	7	8	8	6	6	4	4	4
λ_j	100	150	150	180	200	150	150	150	120	100	80	70	60

(Les durées de service sont en secondes et les λ_j sont par heure.)

Exemple du centre d'appels.

Prenons $\alpha_0 = 10$, $p = 0.1$, $\nu = 0.001$, $\alpha = 1.0$, $\gamma = 0.01$, $s = 20$, 13 périodes, et

j	0	1	2	3	4	5	6	7	8	9	10	11	12
n_j	4	6	8	8	8	7	8	8	6	6	4	4	4
λ_j	100	150	150	180	200	150	150	150	120	100	80	70	60

(Les durées de service sont en secondes et les λ_j sont par heure.)

Soient $X_{1,i}$ la valeur de $G_i(s)/a$ au jour i avec cette configuration; et $X_{2,i}$ la valeur de $G_i(s)/a$ au jour i avec un agent supplémentaire pour les périodes 5 et 6.

Exemple du centre d'appels.

Prenons $\alpha_0 = 10$, $p = 0.1$, $\nu = 0.001$, $\alpha = 1.0$, $\gamma = 0.01$, $s = 20$, 13 périodes, et

j	0	1	2	3	4	5	6	7	8	9	10	11	12
n_j	4	6	8	8	8	7	8	8	6	6	4	4	4
λ_j	100	150	150	180	200	150	150	150	120	100	80	70	60

(Les durées de service sont en secondes et les λ_j sont par heure.)

Soient $X_{1,i}$ la valeur de $G_i(s)/a$ au jour i avec cette configuration; et $X_{2,i}$ la valeur de $G_i(s)/a$ au jour i avec un agent supplémentaire pour les périodes 5 et 6.

On veut estimer $\mu_2 - \mu_1 = \mathbb{E}[X_{2,i} - X_{1,i}]$.

On simule les deux configurations n fois et on estime $\mu_2 - \mu_1$ par

$$\bar{X}_{2,n} - \bar{X}_{1,n} = \frac{1}{n}\sum_{i=1}^{n}(X_{2,i} - X_{1,i}).$$

```
public class CallCenterCRN extends CallCenter {

   Tally statQoS1      = new Tally ("stats on QoS for config.1");
   Tally statDiffIndep = new Tally ("stats on difference with IRNs");
   Tally statDiffCRN   = new Tally ("stats on difference with CRNs");
   int[] numAgents1, numAgents2;

   public CallCenterCRN (String fileName) throws IOException {
      super (fileName);
      numAgents1 = new int[numPeriods];
      numAgents2 = new int[numPeriods];
      for (int j = 0; j < numPeriods; j++)
         numAgents1[j] = numAgents2[j] = numAgents[j];
   }

   // Set the number of agents in each period j to the values in num.
   public void setNumAgents (int[] num) {
      for (int j = 0; j < numPeriods; j++) numAgents[j] = num[j];
   }
```

```
public void simulateDiffCRN (int n) {
   double value1, value2;
   statQoS1.init();   statDiffIndep.init();   statDiffCRN.init();
   for (int i = 0; i < n; i++) {
      setNumAgents (numAgents1);
      streamB.resetNextSubstream();
      streamArr.resetNextSubstream();
      streamPatience.resetNextSubstream();
      (genServ.getStream()).resetNextSubstream();
      simulateOneDay();              // Simulate config 1
      value1 = (double)nGoodQoS / nCallsExpected;

      setNumAgents (numAgents2);
      streamB.resetStartSubstream();
      streamArr.resetStartSubstream();
      streamPatience.resetStartSubstream();
      (genServ.getStream()).resetStartSubstream();
      simulateOneDay();              // Simulate config 2 with CRN
      value2 = (double)nGoodQoS / nCallsExpected;
      statQoS1.add (value1);
      statDiffCRN.add (value2 - value1);

      simulateOneDay();              // Simulate config 2 indep.
      value2 = (double)nGoodQoS / nCallsExpected;
      statDiffIndep.add (value2 - value1);
   }
}
```

```
   static public void main (String[] args) throws IOException {
      int n = 1000;       // Number of replications.
      CallCenterCRN cc = new CallCenterCRN ("CallCenter.dat");
      cc.numAgents2[5]++;   cc.numAgents2[6]++;   // Config 2

      cc.simulateDiffCRN (n);
      System.out.println (
          cc.statQoS1.reportAndCIStudent (0.9) +
          cc.statDiffIndep.reportAndCIStudent (0.9) +
          cc.statDiffCRN.reportAndCIStudent (0.9));
      double varianceIndep = cc.statDiffIndep.variance();
      double varianceCRN   = cc.statDiffCRN.variance();
      // Print variance reduction factor.
      System.out.println ("Variance ratio:  " +
           PrintfFormat.format (10, 2, 3, varianceIndep / varianceCRN));
   }
}
```

```
REPORT on Tally stat. collector ==> stats on QoS for config.1
         min         max      average    standard dev.  num. obs
       0.293       1.131        0.861            0.163         1000
  90.0% confidence interval for mean: (     0.853,     0.870 )

REPORT on Tally stat. collector ==> stats on difference with IRNs
         min         max      average    standard dev.  num. obs
      -0.763       0.775      9.9E-3            0.234         1000
  90.0% confidence interval for mean: (    -0.002,     0.022 )

REPORT on Tally stat. collector ==> stats on difference with CRNs
         min         max      average    standard dev.  num. obs
      -0.013       0.101      9.9E-3            0.016         1000
  90.0% confidence interval for mean: (     0.009,     0.011 )

Variance ratio:    223.69
```

Estimation de dérivées (sensibilité)

La quantité μ que l'on veut estimer dépend d'un paramètre θ dans le modèle, $\mu = \mu(\theta)$, et on veut estimer

$$\mu'(\theta_1) = \left.\frac{\partial \mu(\theta)}{\partial \theta}\right|_{\theta=\theta_1} = \left.\frac{\partial \mathbb{E}_\theta[X(\theta)]}{\partial \theta}\right|_{\theta=\theta_1}.$$

Estimation de dérivées (sensibilité)

La quantité μ que l'on veut estimer dépend d'un paramètre θ dans le modèle, $\mu = \mu(\theta)$, et on veut estimer

$$\mu'(\theta_1) = \left.\frac{\partial \mu(\theta)}{\partial \theta}\right|_{\theta=\theta_1} = \left.\frac{\partial \mathbb{E}_\theta[X(\theta)]}{\partial \theta}\right|_{\theta=\theta_1}.$$

Différences finies:
On simule à $\theta = \theta_1$ pour obtenir un estimateur $X_1 = X_1(\theta_1)$ de $\mu(\theta_1)$,
puis on simule à $\theta = \theta_2 = \theta_1 + \delta$ pour obtenir un estimateur X_2 de $\mu(\theta_2)$,
et on estime la dérivée $\mu'(\theta_1)$ par $\Delta = (X_2 - X_1)/\delta$.

Estimation de dérivées (sensibilité)

La quantité μ que l'on veut estimer dépend d'un paramètre θ dans le modèle, $\mu = \mu(\theta)$, et on veut estimer

$$\mu'(\theta_1) = \left.\frac{\partial \mu(\theta)}{\partial \theta}\right|_{\theta=\theta_1} = \left.\frac{\partial \mathbb{E}_\theta[X(\theta)]}{\partial \theta}\right|_{\theta=\theta_1}.$$

Différences finies:
On simule à $\theta = \theta_1$ pour obtenir un estimateur $X_1 = X_1(\theta_1)$ de $\mu(\theta_1)$,
puis on simule à $\theta = \theta_2 = \theta_1 + \delta$ pour obtenir un estimateur X_2 de $\mu(\theta_2)$,
et on estime la dérivée $\mu'(\theta_1)$ par $\Delta = (X_2 - X_1)/\delta$.

Cet estimateur est biaisé, mais le biais $\beta \to 0$ quand $\delta \to 0$. On a aussi

$$\mathrm{Var}[\Delta] = \frac{\mathrm{Var}(X_2 - X_1)}{\delta^2} = \frac{\mathrm{Var}[X_1] + \mathrm{Var}[X_2] - 2\mathrm{Cov}[X_1, X_2]}{\delta^2}.$$

Si X_1 et X_2 sont indépendants, $\mathrm{Var}[(X_2 - X_1)/\delta] = \mathcal{O}(1/\delta^2) \to \infty$ quand $\delta \to 0$.

Estimation de dérivées (sensibilité)

La quantité μ que l'on veut estimer dépend d'un paramètre θ dans le modèle, $\mu = \mu(\theta)$, et on veut estimer

$$\mu'(\theta_1) = \left.\frac{\partial \mu(\theta)}{\partial \theta}\right|_{\theta=\theta_1} = \left.\frac{\partial \mathbb{E}_\theta[X(\theta)]}{\partial \theta}\right|_{\theta=\theta_1}.$$

Différences finies:
On simule à $\theta = \theta_1$ pour obtenir un estimateur $X_1 = X_1(\theta_1)$ de $\mu(\theta_1)$, puis on simule à $\theta = \theta_2 = \theta_1 + \delta$ pour obtenir un estimateur X_2 de $\mu(\theta_2)$, et on estime la dérivée $\mu'(\theta_1)$ par $\Delta = (X_2 - X_1)/\delta$.

Cet estimateur est biaisé, mais le biais $\beta \to 0$ quand $\delta \to 0$. On a aussi

$$\mathrm{Var}[\Delta] = \frac{\mathrm{Var}(X_2 - X_1)}{\delta^2} = \frac{\mathrm{Var}[X_1] + \mathrm{Var}[X_2] - 2\mathrm{Cov}[X_1, X_2]}{\delta^2}.$$

Si X_1 et X_2 sont indépendants, $\mathrm{Var}[(X_2 - X_1)/\delta] = \mathcal{O}(1/\delta^2) \to \infty$ quand $\delta \to 0$.
Idée: utiliser des valeurs aléatoires communes pour avoir $\mathrm{Cov}[X_1, X_2] > 0$. Sous certaines conditions, on peut prouver que $\mathrm{Var}[(X_2 - X_1)/\delta]$ est bornée quand $\delta \to 0$. Super!

Il faut quand même au moins d simulations pour estimer un vecteur de d dérivées.

Parfois, on peut prendre $\lim_{\delta \to 0}(X_2 - X_1)/\delta$ directement comme estimateur.
Il suffit alors d'une seule simulation pour estimer toutes les dérivées!

Parfois, on peut prendre $\lim_{\delta \to 0}(X_2 - X_1)/\delta$ directement comme estimateur.
Il suffit alors d'une seule simulation pour estimer toutes les dérivées!

Détails: Supposons que $X(\theta) = f(\theta, \mathbf{U})$ où $\mathbf{U} \sim U(0,1)^s$ et que

$$f'(\theta, \mathbf{U}) = \partial f(\theta, \mathbf{U})/\partial\theta \qquad \text{existe avec prob.1 à } \theta_1.$$

Cette dérivée stochastique $f'(\theta, \mathbf{U})$ est un estimateur sans biais de $\mu'(\theta)$ ssi

$$\mathbb{E}[f'(\theta, \mathbf{U})] \stackrel{\text{def}}{=} \mathbb{E}\left[\frac{\partial f(\theta, \mathbf{U})}{\partial\theta}\right] \stackrel{?}{=} \frac{\partial \mathbb{E}[f(\theta, \mathbf{U})]}{\partial\theta} \stackrel{\text{def}}{=} \mu'(\theta, \mathbf{U}). \tag{1}$$

Parfois, on peut prendre $\lim_{\delta \to 0}(X_2 - X_1)/\delta$ directement comme estimateur.
Il suffit alors d'une seule simulation pour estimer toutes les dérivées!

Détails: Supposons que $X(\theta) = f(\theta, \mathbf{U})$ où $\mathbf{U} \sim U(0,1)^s$ et que

$$f'(\theta, \mathbf{U}) = \partial f(\theta, \mathbf{U})/\partial\theta \qquad \text{existe avec prob.1 à } \theta_1.$$

Cette dérivée stochastique $f'(\theta, \mathbf{U})$ est un estimateur sans biais de $\mu'(\theta)$ ssi

$$\mathbb{E}[f'(\theta, \mathbf{U})] \stackrel{\text{def}}{=} \mathbb{E}\left[\frac{\partial f(\theta, \mathbf{U})}{\partial\theta}\right] \stackrel{?}{=} \frac{\partial \mathbb{E}[f(\theta, \mathbf{U})]}{\partial\theta} \stackrel{\text{def}}{=} \mu'(\theta, \mathbf{U}). \qquad (1)$$

Condition suffisante: théorème de convergence dominée. S'il existe $\delta_1 > 0$ et une v.a. Y tels que

$$\sup_{\delta \in (0, \delta_1]} \frac{|f(\theta + \delta, \mathbf{U}) - f(\theta, \mathbf{U})|}{\delta} \leq Y$$

et $\mathbb{E}[Y] < \infty$, alors (1) est valide.

Si plusieurs paramètres, le vecteur des dérivées est le gradient stochastique.

Exemple: réseau d'activités stochastique, $\mathbb{E}[T]$.

On veut estimer la dérivée de $\mathbb{E}[T]$ p.r. à chaque $\theta_j = \mu_j$ (la moyenne).
On considère un θ_j à la fois.
On écrit $T = f_j(\theta_j, \mathbf{U})$ où $\mathbf{U} = (U_1, \ldots, U_{13})$.

Exemple: réseau d'activités stochastique, $\mathbb{E}[T]$.

On veut estimer la dérivée de $\mathbb{E}[T]$ p.r. à chaque $\theta_j = \mu_j$ (la moyenne).
On considère un θ_j à la fois.
On écrit $T = f_j(\theta_j, \mathbf{U})$ où $\mathbf{U} = (U_1, \ldots, U_{13})$.

On voit que $f_j'(\theta_j, \mathbf{U}) = V_j'(\theta_j)$ si l'arc j est sur le plus long chemin, et $f_j'(\theta_j, \mathbf{U}) = 0$ sinon.

Exemple: réseau d'activités stochastique, $\mathbb{E}[T]$.

On veut estimer la dérivée de $\mathbb{E}[T]$ p.r. à chaque $\theta_j = \mu_j$ (la moyenne).
On considère un θ_j à la fois.
On écrit $T = f_j(\theta_j, \mathbf{U})$ où $\mathbf{U} = (U_1, \ldots, U_{13})$.

On voit que $f_j'(\theta_j, \mathbf{U}) = V_j'(\theta_j)$ si l'arc j est sur le plus long chemin, et $f_j'(\theta_j, \mathbf{U}) = 0$ sinon.

Si V_j suit la loi exponentielle, alors $V_j = V_j(\theta_j) = -\theta_j \ln(1 - U_j)$, $V_j'(\theta_j) = -\ln(1 - U_j)$ et

$$0 \leq \frac{f_j(\theta_j + \delta, \mathbf{U}) - f_j(\theta_j, \mathbf{U})}{\delta} \leq \frac{-\delta \ln(1 - U_j)}{\delta} = -\ln(1 - U_j) = Y_j,$$

où $Y_j \sim \text{Exponentielle}(1)$. Le théorème de convergence dominée s'applique: $f_j'(\theta_j, \mathbf{U})$ est sans biais.

Exemple: réseau d'activités stochastique, $\mathbb{E}[T]$.

On veut estimer la dérivée de $\mathbb{E}[T]$ p.r. à chaque $\theta_j = \mu_j$ (la moyenne).
On considère un θ_j à la fois.
On écrit $T = f_j(\theta_j, \mathbf{U})$ où $\mathbf{U} = (U_1, \ldots, U_{13})$.

On voit que $f_j'(\theta_j, \mathbf{U}) = V_j'(\theta_j)$ si l'arc j est sur le plus long chemin, et $f_j'(\theta_j, \mathbf{U}) = 0$ sinon.

Si V_j suit la loi exponentielle, alors $V_j = V_j(\theta_j) = -\theta_j \ln(1 - U_j)$, $V_j'(\theta_j) = -\ln(1 - U_j)$ et

$$0 \leq \frac{f_j(\theta_j + \delta, \mathbf{U}) - f_j(\theta_j, \mathbf{U})}{\delta} \leq \frac{-\delta \ln(1 - U_j)}{\delta} = -\ln(1 - U_j) = Y_j,$$

où $Y_j \sim \text{Exponentielle}(1)$. Le théorème de convergence dominée s'applique: $f_j'(\theta_j, \mathbf{U})$ est sans biais.

Si V_j suit la loi normale, alors $V_j = V_j(\theta_j) = \theta_j + (\theta_j/4)\Phi^{-1}(U_j)$ et $V_j'(\theta_j) = 1 + \Phi^{-1}(U_j)/4$. Sans biais aussi.

Exemple: réseau d'activités stochastique, $\mathbb{P}[T > x]$.

On veut maintenant estimer la dérivée de $\mathbb{P}[T > x]$ p.r. à θ_j.
L'estimateur de $\mathbb{P}[T > x]$ est $f_j(\theta_j, \mathbf{U}) = \mathbb{I}[T > x]$.
Ne peut prendre que les valeurs 0 et 1.

Exemple: réseau d'activités stochastique, $\mathbb{P}[T > x]$.

On veut maintenant estimer la dérivée de $\mathbb{P}[T > x]$ p.r. à θ_j.
L'estimateur de $\mathbb{P}[T > x]$ est $f_j(\theta_j, \mathbf{U}) = \mathbb{I}[T > x]$.
Ne peut prendre que les valeurs 0 et 1.

La dérivée $f_j'(\theta_j, \mathbf{U})$ est toujours soit 0, soit pas définie (survient avec prob. 0).
On a donc $\mathbb{P}[f_j'(\theta_j, \mathbf{U}) = 0] = 1$. Estimateur biaisé de $\mu'(\theta_j) = \partial \mathbb{P}[T > x]/\partial \theta_j$.

Ici le théorème de convergence dominée ne s'applique pas car
$\sup_{\delta>0}[f_j(\theta_j + \delta, \mathbf{U}) - f_j(\theta_j, \mathbf{U})]/\delta$ n'est pas intégrable.
Le problème vient du fait que f_j est discontinue en θ_j.

Exemple: réseau d'activités stochastique, $\mathbb{P}[T > x]$.

On veut maintenant estimer la dérivée de $\mathbb{P}[T > x]$ p.r. à θ_j.
L'estimateur de $\mathbb{P}[T > x]$ est $f_j(\theta_j, \mathbf{U}) = \mathbb{I}[T > x]$.
Ne peut prendre que les valeurs 0 et 1.

La dérivée $f_j'(\theta_j, \mathbf{U})$ est toujours soit 0, soit pas définie (survient avec prob. 0).
On a donc $\mathbb{P}[f_j'(\theta_j, \mathbf{U}) = 0] = 1$. Estimateur biaisé de $\mu'(\theta_j) = \partial \mathbb{P}[T > x]/\partial \theta_j$.

Ici le théorème de convergence dominée ne s'applique pas car
$\sup_{\delta>0}[f_j(\theta_j + \delta, \mathbf{U}) - f_j(\theta_j, \mathbf{U})]/\delta$ n'est pas intégrable.
Le problème vient du fait que f_j est discontinue en θ_j.

Plus tard on verra comment régler ce problème en remplaçant l'indicateur
$\mathbb{I}[T > x]$ par une espérance conditionnelle, continue en θ_j.

Modèles à événements discrèts.

La dérivée stochastique peut être compliquée à calculer, car une variation infinitésimale de θ peut avoir des répercussions complexes sur la suite des événements. Outils: analyse de perturbation infinitésimale (IPA).

Modèles à événements discrèts.

La dérivée stochastique peut être compliquée à calculer, car une variation infinitésimale de θ peut avoir des répercussions complexes sur la suite des événements. Outils: analyse de perturbation infinitésimale (IPA).

Pourquoi estimer les dérivées?

(a) Évaluer l'importance des différents paramètres (par ex. pour construire un méta-modèle).
(b) Intervalle de confiance qui tient compte de l'erreur d'estimation des paramètres du modèle.
(c) Évaluer l'effet du changement d'un paramètre de décision.
(d) Un estimateur du gradient est souvent requis dans les algorithmes d'optimisation.
(e) En finance, les dérivées de la valeur d'un contrat (les "Greeks") sont requises pour l'implantation de stratégies de "hedging".

Optimisation

Supposons que l'on veut optimiser certains paramètres du modèle via la simulation. Comment faire?

Optimisation

Supposons que l'on veut optimiser certains paramètres du modèle via la simulation. Comment faire?

Discussion et exemples dans le contexte du centre d'appels.

Optimisation

Supposons que l'on veut optimiser certains paramètres du modèle via la simulation. Comment faire?

Discussion et exemples dans le contexte du centre d'appels.

Par exemple, choisir le nombre et les horaires des agents pour minimiser le coût moyen à long terme sous la contrainte $g(20) \geq 0.80$.

Optimisation

Supposons que l'on veut optimiser certains paramètres du modèle via la simulation. Comment faire?

Discussion et exemples dans le contexte du centre d'appels.

Par exemple, choisir le nombre et les horaires des agents pour minimiser le coût moyen à long terme sous la contrainte $g(20) \geq 0.80$.

Problème: il restera toujours de l'incertitude.

Optimisation

Supposons que l'on veut optimiser certains paramètres du modèle via la simulation. Comment faire?

Discussion et exemples dans le contexte du centre d'appels.

Par exemple, choisir le nombre et les horaires des agents pour minimiser le coût moyen à long terme sous la contrainte $g(20) \geq 0.80$.

Problème: il restera toujours de l'incertitude.

Genre d'objectif à viser: le coût de la solution retenue dépasse celui d'une solution optimale par moins de 2%, avec probabilité $\geq 95\%$.

Formulation générale.

$$\min_{\theta \in \Theta} \alpha(\theta) \qquad \text{où} \qquad \alpha(\theta) = E_\theta[h(\theta, \mathbf{Y})],$$

$\theta =$ vecteur de variables de décision (ou configurations du système).
$\Theta =$ ensemble des solutions réalisables.
$\mathbf{Y} =$ vecteur de variables aléatoires, dont la loi peut dépendre de θ.
$h =$ fonction de coût, qui peut aussi dépendre de θ.

Formulation générale.

$$\min_{\theta \in \Theta} \alpha(\theta) \quad \text{où} \quad \alpha(\theta) = E_\theta[h(\theta, \mathbf{Y})],$$

$\theta =$ vecteur de variables de décision (ou configurations du système).
$\Theta =$ ensemble des solutions réalisables.
$\mathbf{Y} =$ vecteur de variables aléatoires, dont la loi peut dépendre de θ.
$h =$ fonction de coût, qui peut aussi dépendre de θ.

Objectif utopique:
Trouver une solution optimale θ^* et la valeur optimale $\alpha(\theta^*)$.

Formulation générale.

$$\min_{\theta \in \Theta} \alpha(\theta) \quad \text{où} \quad \alpha(\theta) = E_\theta[h(\theta, \mathbf{Y})],$$

$\theta =$ vecteur de variables de décision (ou configurations du système).
$\Theta =$ ensemble des solutions réalisables.
$\mathbf{Y} =$ vecteur de variables aléatoires, dont la loi peut dépendre de θ.
$h =$ fonction de coût, qui peut aussi dépendre de θ.

Objectif utopique:
Trouver une solution optimale θ^* et la valeur optimale $\alpha(\theta^*)$.

Objectif réaliste:
Trouver un $\hat{\theta}$ pour lequel l'écart d'optimalité $\alpha(\hat{\theta}) - \alpha(\theta^*)$ est petit (en mesure relative ou absolue).

Méthodes statistiques d'ordonnancement et de selection

Si $|\Theta|$ ne dépasse pas 20 ou 25, on peut essayer toutes les possibilités n fois pour un très grand n (pour que l'erreur d'estimation soit négligeable) et choisir celle donnant la plus petite moyenne (force brute).

Méthodes statistiques d'ordonnancement et de selection

Si $|\Theta|$ ne dépasse pas 20 ou 25, on peut essayer toutes les possibilités n fois pour un très grand n (pour que l'erreur d'estimation soit négligeable) et choisir celle donnant la plus petite moyenne (force brute).

Plus efficace: éliminer plus rapidement les configurations non compétitives.

Méthodes statistiques d'ordonnancement et de selection

Si $|\Theta|$ ne dépasse pas 20 ou 25, on peut essayer toutes les possibilités n fois pour un très grand n (pour que l'erreur d'estimation soit négligeable) et choisir celle donnant la plus petite moyenne (force brute).

Plus efficace: éliminer plus rapidement les configurations non compétitives.

On choisit un écart d'optimalité acceptable δ et un niveau de confiance $1-\alpha$, et des méthodes adaptatives déterminent quoi simuler puis retournent un $\hat{\theta}$ tel que

$$P[\alpha(\hat{\theta}) - \alpha(\theta^*) \leq \delta] \geq 1 - \alpha.$$

Méthodes statistiques d'ordonnancement et de selection

Si $|\Theta|$ ne dépasse pas 20 ou 25, on peut essayer toutes les possibilités n fois pour un très grand n (pour que l'erreur d'estimation soit négligeable) et choisir celle donnant la plus petite moyenne (force brute).

Plus efficace: éliminer plus rapidement les configurations non compétitives.

On choisit un écart d'optimalité acceptable δ et un niveau de confiance $1-\alpha$, et des méthodes adaptatives déterminent quoi simuler puis retournent un $\hat{\theta}$ tel que

$$P[\alpha(\hat{\theta}) - \alpha(\theta^*) \leq \delta] \geq 1 - \alpha.$$

Déterminer un tel $\hat{\theta}$ est parfois plus facile que d'estimer $\alpha(\theta)$ avec précision pour un θ fixé.

Paramètres continus. Si θ est un paramètre continu et α est différentiable, on peut tenter d'adapter les algorithmes d'optimisation non linéaire.

Paramètres continus. Si θ est un paramètre continu et α est différentiable, on peut tenter d'adapter les algorithmes d'optimisation non linéaire.

Approximation stochastique (méthode de descente stochastique):
choisir une suite déterministe $\{a_n,\, n \geq 0\}$ et une solution initiale θ_0;
à chaque étape n, calculer un estimateur D_n du gradient $\nabla_\theta \alpha(\theta_n)$ à la solution courante θ_n, puis aller à la solution suivante:

$$\theta_{n+1} = \theta_n - a_n D_n.$$

Paramètres continus. Si θ est un paramètre continu et α est différentiable, on peut tenter d'adapter les algorithmes d'optimisation non linéaire.

Approximation stochastique (méthode de descente stochastique):
choisir une suite déterministe $\{a_n, n \geq 0\}$ et une solution initiale θ_0;
à chaque étape n, calculer un estimateur D_n du gradient $\nabla_\theta \alpha(\theta_n)$ à la solution courante θ_n, puis aller à la solution suivante:

$$\theta_{n+1} = \theta_n - a_n D_n.$$

Si θ_{n+1} n'est pas réalisable, on peut projeter sur Θ:

$$\theta_{n+1} = \Pi_\Theta(\theta_n - a_n D_n).$$

Paramètres continus. Si θ est un paramètre continu et α est différentiable, on peut tenter d'adapter les algorithmes d'optimisation non linéaire.

Approximation stochastique (méthode de descente stochastique):
choisir une suite déterministe $\{a_n,\, n \geq 0\}$ et une solution initiale θ_0;
à chaque étape n, calculer un estimateur D_n du gradient $\nabla_\theta \alpha(\theta_n)$ à la solution courante θ_n, puis aller à la solution suivante:

$$\theta_{n+1} = \theta_n - a_n D_n.$$

Si θ_{n+1} n'est pas réalisable, on peut projeter sur Θ:

$$\theta_{n+1} = \Pi_\Theta(\theta_n - a_n D_n).$$

Plusieurs variantes: choix de l'estimateur D_n, choix de a_n, remplacer a_n par une matrice, choix de a_n adaptatif, ...

Paramètres continus. Si θ est un paramètre continu et α est différentiable, on peut tenter d'adapter les algorithmes d'optimisation non linéaire.

Approximation stochastique (méthode de descente stochastique):
choisir une suite déterministe $\{a_n,\, n \geq 0\}$ et une solution initiale θ_0;
à chaque étape n, calculer un estimateur D_n du gradient $\nabla_\theta \alpha(\theta_n)$ à la solution courante θ_n, puis aller à la solution suivante:

$$\theta_{n+1} = \theta_n - a_n D_n.$$

Si θ_{n+1} n'est pas réalisable, on peut projeter sur Θ:

$$\theta_{n+1} = \Pi_\Theta(\theta_n - a_n D_n).$$

Plusieurs variantes: choix de l'estimateur D_n, choix de a_n, remplacer a_n par une matrice, choix de a_n adaptatif, ...

Important: estimateur D_n de bonne qualité.
Beaucoup de développements en ce sens depuis 15 ans.

Optimisation d'une réalisation stochastique ("sample-path optimization"):
L'idée est d'estimer toute la fonction α par une fonction $\hat{\alpha}$, puis d'optimiser $\hat{\alpha}$ par une méthode d'optimisation déterministe classique.

Optimisation d'une réalisation stochastique ("sample-path optimization"):
L'idée est d'estimer toute la fonction α par une fonction $\hat{\alpha}$, puis d'optimiser $\hat{\alpha}$ par une méthode d'optimisation déterministe classique.
Lié: "stochastic counterpart method", "response surface methodology".

Optimisation d'une réalisation stochastique ("sample-path optimization"):
L'idée est d'estimer toute la fonction α par une fonction $\hat{\alpha}$, puis d'optimiser $\hat{\alpha}$ par une méthode d'optimisation déterministe classique.
Lié: "stochastic counterpart method", "response surface methodology".

Cas plus difficiles: nombreux optima locaux, paramètres discrets (optimisation combinatoire stochastique), etc.

Optimisation d'une réalisation stochastique ("sample-path optimization"):
L'idée est d'estimer toute la fonction α par une fonction $\hat{\alpha}$, puis d'optimiser $\hat{\alpha}$ par une méthode d'optimisation déterministe classique.
Lié: "stochastic counterpart method", "response surface methodology".

Cas plus difficiles: nombreux optima locaux, paramètres discrets (optimisation combinatoire stochastique), etc.

Méthodes: recherche aléatoire et variantes, recherche par voisinage, algorithmes de type génétique, etc.
Plusieurs sont des heuristiques; pas de preuve de convergence.
C'est ce que l'on retrouve dans les logiciels de simulation commerciaux.

Exemple: Scheduling dans un centre d'appels.

Centre d'appels recevant K types d'appels et ayant I types d'agents.

Chaque jour est divisé en P périodes. Un horaire de travail: ensemble de périodes. Il y a Q types d'horaires admissibles.

Exemple: Scheduling dans un centre d'appels.

Centre d'appels recevant K types d'appels et ayant I types d'agents.

Chaque jour est divisé en P périodes. Un horaire de travail: ensemble de périodes.
Il y a Q types d'horaires admissibles.

$c_{i,q}$ = coût d'un agent de type i ayant un horaire q.
Vecteur des coûts: $\mathbf{c} = (c_{1,1}, \ldots, c_{1,Q}, \ldots, c_{I,1}, \ldots, c_{I,Q})^{\mathsf{t}}$.

$x_{i,q}$ = nombre d'agents de type i ayant un horaire q.
Vecteur des variables de décision: $\mathbf{x} = (x_{1,1}, \ldots, x_{1,Q}, \ldots, x_{I,1}, \ldots, x_{I,Q})^{\mathsf{t}}$.

Exemple: Scheduling dans un centre d'appels.

Centre d'appels recevant K types d'appels et ayant I types d'agents.

Chaque jour est divisé en P périodes. Un horaire de travail: ensemble de périodes.
Il y a Q types d'horaires admissibles.

$c_{i,q}$ = coût d'un agent de type i ayant un horaire q.
Vecteur des coûts: $\mathbf{c} = (c_{1,1}, \ldots, c_{1,Q}, \ldots, c_{I,1}, \ldots, c_{I,Q})^{\mathsf{t}}$.

$x_{i,q}$ = nombre d'agents de type i ayant un horaire q.
Vecteur des variables de décision: $\mathbf{x} = (x_{1,1}, \ldots, x_{1,Q}, \ldots, x_{I,1}, \ldots, x_{I,Q})^{\mathsf{t}}$.

Posons $y_{i,p}$ = nombre d'agents de type i dans la période p,
et $\mathbf{y} = (y_{1,1}, \ldots, y_{1,P}, \ldots, y_{I,1}, \ldots, y_{I,P})^{\mathsf{t}} = \mathbf{A}\mathbf{x}$ où $\mathbf{A}$ est diagonale par blocs et l'élément (p, q) de chaque bloc est 1 si l'horaire q couvre la période p, 0 sinon.

Le niveau de service pour les appels de type k durant la période p est défini par

$$g_{k,p}(\mathbf{y}) = \frac{\mathbb{E}[G_{k,p}(s_{k,p})]}{\mathbb{E}[A_{k,p}]}$$

où $A_{k,p}$ est le nombre d'appels de type k arrivant durant la période p, et $G_{k,p}(s_{k,p})$ est la nombre de ceux-là répondus en moins de $s_{k,p}$ secondes, et les $s_{k,p}$ sont des constantes choisies.

Le niveau de service pour les appels de type k durant la période p est défini par

$$g_{k,p}(\mathbf{y}) = \frac{\mathbb{E}[G_{k,p}(s_{k,p})]}{\mathbb{E}[A_{k,p}]}$$

où $A_{k,p}$ est le nombre d'appels de type k arrivant durant la période p, et $G_{k,p}(s_{k,p})$ est la nombre de ceux-là répondus en moins de $s_{k,p}$ secondes, et les $s_{k,p}$ sont des constantes choisies.

On définit aussi des niveaux de service agrégés sur les périodes, ou les types d'appels, ou les deux.

Par exemple, $g_k(\mathbf{y})$ reprśente la fraction des appels répondus en moins de s_k secondes durant toute la journée, à long terme (sur une infinité de journées).

Le niveau de service pour les appels de type k durant la période p est défini par

$$g_{k,p}(\mathbf{y}) = \frac{\mathbb{E}[G_{k,p}(s_{k,p})]}{\mathbb{E}[A_{k,p}]}$$

où $A_{k,p}$ est le nombre d'appels de type k arrivant durant la période p, et $G_{k,p}(s_{k,p})$ est la nombre de ceux-là répondus en moins de $s_{k,p}$ secondes, et les $s_{k,p}$ sont des constantes choisies.

On définit aussi des niveaux de service agrégés sur les périodes, ou les types d'appels, ou les deux.

Par exemple, $g_k(\mathbf{y})$ reprśente la fraction des appels répondus en moins de s_k secondes durant toute la journée, à long terme (sur une infinité de journées).

Problème d'optimisation:

$$\begin{aligned}
\text{minimiser} \quad & \mathbf{c}^{\mathrm{t}}\mathbf{x} = \textstyle\sum_{i=1}^{I}\sum_{q=1}^{Q} c_{i,q}x_{i,q} \\
\text{sujet à} \quad & \mathbf{A}\mathbf{x} = \mathbf{y}, \\
& g_{k,p}(\mathbf{y}) \geq l_{k,p} \qquad \text{for all } k, p, \\
& g_p(\mathbf{y}) \geq l_p \qquad \text{for all } p, \\
& g_k(\mathbf{y}) \geq l_k \qquad \text{for all } k, \\
& g(\mathbf{y}) \geq l, \\
& \mathbf{x} \geq 0, \text{ et entier.}
\end{aligned}$$

Difficulté: On ne sait pas comment calculer exactement les fonctions $g_\bullet$ pour une solution donnée $\mathbf{x}$. Ces fonctions sont très complexes. En pratique, les taux d'arrivée varient dans le temps, il y a des abandons, le routage des appels est complexe, etc.

Difficulté: On ne sait pas comment calculer exactement les fonctions $g_\bullet$ pour une solution donnée $\mathbf{x}$. Ces fonctions sont très complexes. En pratique, les taux d'arrivée varient dans le temps, il y a des abandons, le routage des appels est complexe, etc.

Solution: optimisation en utilisant la simulation pour évaluer les contraintes.