Cours SEP
Simulation pour l'Evaluation des Performances

Niveau Master 1

RN - RS

Prof. BOUALLOUCHE Louiza

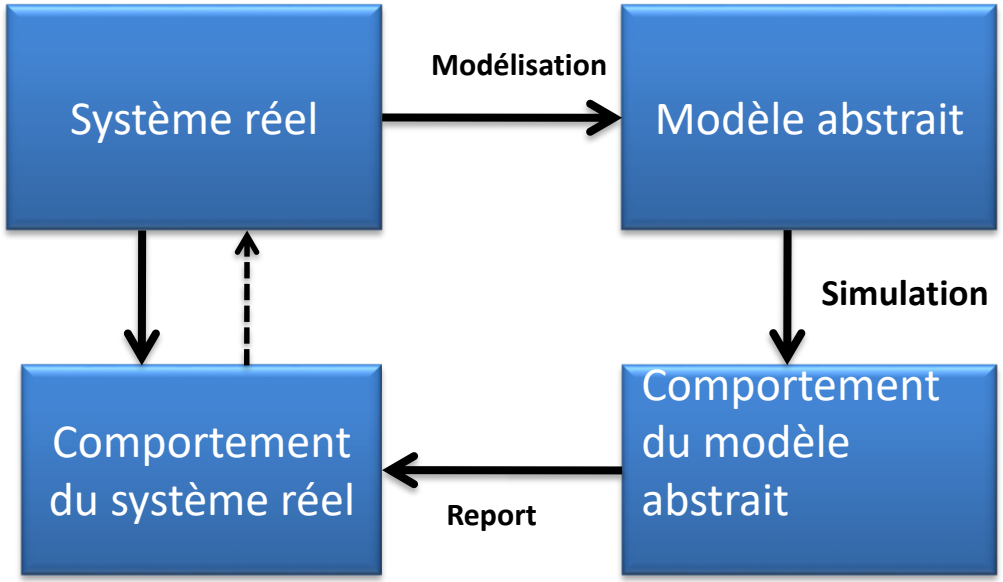
Bouallouche.medjkoune@univ-bejaia.dz

Département Informatique / Université de Béjaïa

Programme

1. Introduction à la Simulation : Cycle et étapes
2. Simulation à évènements discrets : Algorithme intuitif
3. Fonctions de génération de variables aléatoires
4. Analyse opérationnelle et calcul des métriques
5. Exemples de simulation de systèmes et réseaux

1. Introduction à la Simulation : Cycle et étapes



Cycle de simulation

Étapes de Simulation

1. Construction du modèle de simulation ;
2. Implémentation du modèle ;
3. Création des expériences de simulation ;
4. Validation du modèle de simulation ;
5. Exécution du simulateur et analyse de données (data mining).

– Le modèle sera valide s'il permet à un analyste ou un décideur de prendre, à propos du système donné, les mêmes décisions que celles qu'il aurait prises s'il avait pu procéder à une expérimentation directe.

– Les résultats d'études faites par simulation auront d'autant plus de crédibilité s'ils sont présentés indépendamment de la nature aléatoire des lois utilisées et sur une analyse appropriée des résultats de la simulation.

3. Simulation à évènements discrets : Algorithme intuitif

Il s'agit de prendre en compte, dans la modélisation des tâches actives, les seuls instants où un événement se produit et à concentrer l'activité des tâches simulées sur ces instants là.

Les variables décrivant un état ne changent qu'à un nombre fini de points sur l'axe des temps. Ces points sont les instants où se passent les événements.

Exemple

Pour une station file d'attente, les événements généralement considérés sont :

- l'arrivée d'un client à la station et
- la prise en charge d'un client par le serveur ou
- la sortie du client de la station.

Types de simulateurs

- ***Simulateurs orientés événements***. On a une vue centrée sur les objets manipulés par les acteurs du système (programme dans une CPU, paquet au niveau d'un nœud de communication, ...). Il s'agit de gérer l'échéancier où les événements sont consignés avec l'heure de leur activation et la procédure qui doit être appelée à ce moment là. Lors de l'exécution de ces procédures de nouveaux événements peuvent être créés et insérés dans l'échéancier.

- ***Simulateurs orientés processus.***

  L'unité conceptuelle est la tâche active. Elle est programmée comme un processus activé par chaque événement se rapportant à cette tâche. Ces systèmes ont une vue centrée sur les acteurs eux mêmes (CPU, nœud, ...).

Simulateurs orientés événements

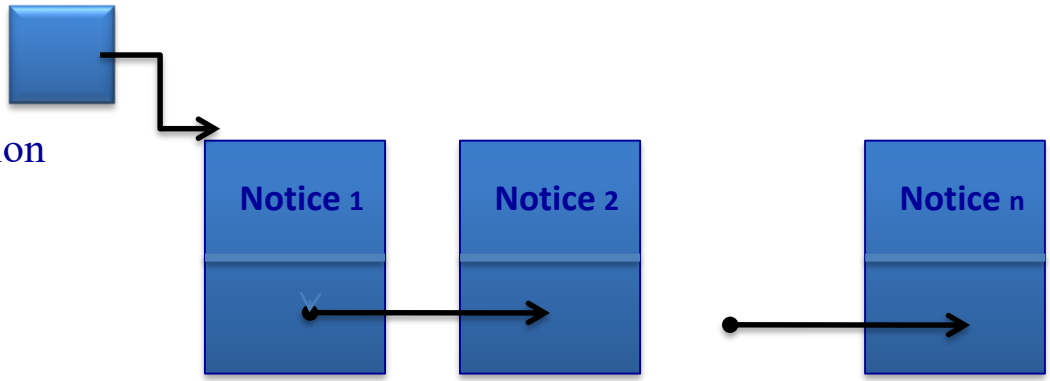
Un événement est décrit par deux attributs constituant la notice d'événement.

Notice d'événement

Le simulateur lui même a deux attributs :

- une horloge qui fournit l'heure de simulation (aussi l'heure d'activation d'un événement)

- un échéancier contenant des notices d'événements .



Échéancier

Algorithme de simulation

Procédure d'activation d'un événement

```
Procédure Pi() ;
{
    Exécuter les opérations correspondantes à l'événement ;
    Mettre à jour les variables statistiques ;
    Créer les événements qui résultent (révoqués) de ce traitement ;
    Insérer (supprimer) leurs notices dans l'échéancier ;
}
FIN.
```

Procédure de simulation (Algorithme principal)

```
Procédure Simulation() ;
{
  Initialiser échéancier, horloge de simulation, variables ;
  Créer premiers événements, insérer leurs notices dans échéancier ;
  While (non-fin-de-simulation et échéancier non vide)
  {
    Retirer la notice en tête de l'échéancier ;
    Mettre à l'heure l'horloge de la simulation ;
     Activer la procédure correspondante ;
  }
  Calculer les valeurs statistiques et faire le rapport de l'expérience ;
  Afficher les résultats
}
FIN.
```

Exemple : Simulation d'une file d'attente

On peut considérer seulement deux événements :

- **Ea** : **« Arrivée d'un client dans la file »** et
- **Es**, : « **Fin de service ou départ d'un client** ».

Pa et **Ps** les **procédures** correspondantes.

```
Procédure  simulation () ;
{
   while   (non-fin-de-simulation)
    {
       obtenir prochain événement ;
       If Ea {
          Pa
        }
        else {
         Ps} ;
    }
   Affichage des résultats ;
}
Fin.
```

4. Fonctions de génération de variables aléatoires

Il s'agit de la Génération des données d'entrée, pseudo aléatoires d'un simulateur, on parle de génération de la charge synthétique.

❖ Génération de nombres aléatoires

Il s'agit de générer, par ordinateur, des n.a. statistiquement indépendants et répartis de manière uniforme sur [0,1] (n ≈≈> U[0, 1]) appelés nombre ***pseudo-aléatoires***.

Méthode séquentielle

Elle consiste à générer des nombres entiers *Xn* compris entre 0 et *m*.

$$U_n = x_n/m \Longrightarrow U_n \rightsquigarrow U[0,1]$$

La séquence des Xn est déterminée comme suit :

$$x_n = \begin{cases} x_0 (\text{valeur de départ}) \\ f(x) \end{cases}$$

Il existe au plus *m* nombres différents ⇒ elle est périodique.

Conditions requises pour la génération des n.a.

- La périodicité doit être la plus grande possible ($\Longleftrightarrow$ Uniformité)
- Les nombres doivent satisfaire les tests d'indépendance statistiques,
- Le calcul doit être simple et efficace.

La fonction f est de la forme $f(x) = (ax + c)mod(m)$, où
m : module $m > 0$ ($m = 2^{31}$ si la taille du mot mémoire $= 32bits$)
a : multiplicateur $0 < a < m$
c : incrément $0 \leq c < m$
x_0 : valeur de départ $0 \leq x_0 < m$.

Les bons candidats (générateurs) sont :

$$x_i = (5^{15}x_{i-1} + 1)mod2^{35},$$
$$x_i = (314159269x_{i-1} + 453806245)mod2^{31}$$
$$x_i = 16807x_{i-1}mod(2^{31} - 1)\ \text{(reconnu meilleur générateur)}\ ...$$

❖ Générateurs de variables aléatoires

Il s'agit d'engendrer une variable aléatoire (v.a.) X suivant une certaine loi à partir des lois plus simples (loi uniforme) et en se basant sur des techniques connues.

Transformation inverse (ou technique par inversion)

Cette technique permet de générer des valeurs de X de fonction de répartition F(x) désirée (continue et strictement croissante).

Il s'agit de générer un nombre aléatoire uniforme U(U ∈ U[0, 1]) puis de calculer $x = F^{-1}(U).$

En effet, la $P(X \leq x) = P(F^{-1}(U) \leq x) = P(U \leq F(x)) = F(x).$

Fonction de génération par la transformation Inverse

Supposons que l'on dispose d'une fonction ***Uniforme()*** qui génère des nombres U[0, 1].

Et soit $Finv = F^{-1}$

Algorithme qui génère des valeurs de X de fonction de répartition F(x).

```
type function generer () ;
{
U=uniforme() ;
  X=Finv(U) ;
return X;
}
```

Exemple

Soit X une v.a suivant une loi exponentielle : $X \rightsquigarrow \exp(\lambda)$

La fonction de répartition de X $F(x) = 1 - e^{-\lambda x} \Rightarrow x = -\frac{\ln(1-U)}{\lambda}$, $U \in [0,1]$

```
real  function  generer-Exp();
{
X= - ln (Uniforme ( ) ) / lambda;  / / U et 1-U ont la même fonction-répartition
Return X
}
```

Si X = durée de traitement d'une tâche,

h[i] **:** l'instant de fin de traitement de la tâche ***i.***

On pourrait connaître la séquence des instants de fin de traitement des tâches

h[i+1]= h[i] + generer-Exp()

Méthode de convolution

Cette méthode s'applique lorsque la v.a. X est la somme de plusieurs variables données.

$$X = Y_1 + Y_2 + ... + Y_k$$

L'algorithme de génération d'une valeur de X consiste à :

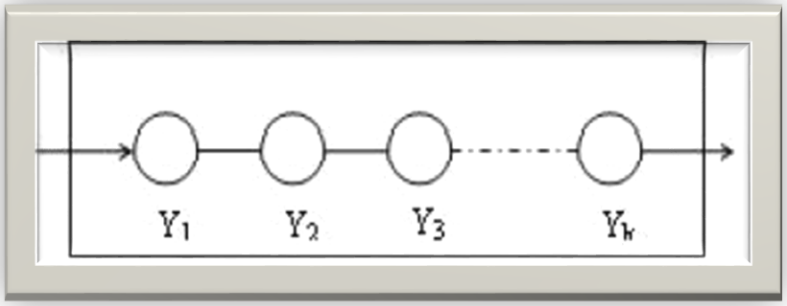
1. générer les Y_i (selon leurs lois de distribution) ;
2. calculer $X = Y_1 + Y_2 + ... + Y_k$;
3. retourner X;

Exemple.

Soit $X \rightsquigarrow Erlang_k(\lambda)$. $X = Y_1 + Y_2 + ... + Y_k$

Où $Y_i \rightsquigarrow \exp(\lambda)$

Schématiquement, X est représentée par k serveurs fictifs en série



Loi Erlang-k

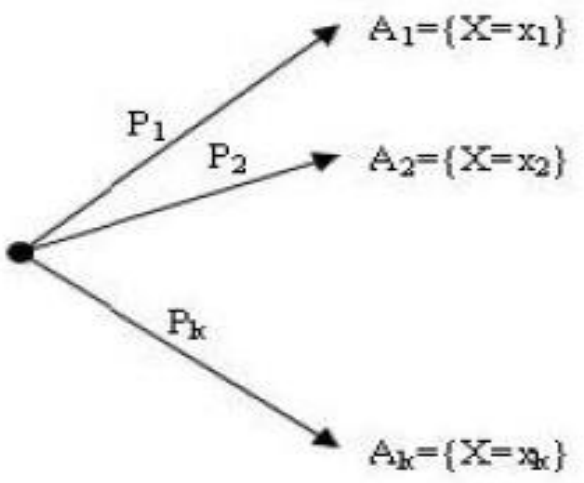
$$Y_i = -\frac{Ln(U_i)}{\lambda}, U_i \rightsquigarrow U[0,1]. \qquad X = -\frac{\sum_{i=1}^{k} Ln(U_i)}{\lambda} = -\frac{Ln\prod_{i=1}^{k}(U_i)}{\lambda}$$

```
real function generer-Erlangk ();
{ X=1 ; i=1 ;
    While i ≤ k
    {
        U=Uniforme() ;
        X=X*U ; i++ ;
    }
    X = − ln(X)/Lambda ;
    return X ;
}
```

Génération de v.a. discrètes (ou aiguillage probabiliste)

Soit X une v.a. discrète prenant l'ensemble fini de valeurs $x_1, x_2, \ldots, x_k$.

$$P(X = x_i) = P(x_i) = P_i, \text{ avec } \sum_{i=1}^{k} P_i = 1.$$
$$F(x) = P(X \leq x) = \sum_{x_i \leq x} P(x_i).$$



Algorithme.
- Générer $U \in U[0,1]$
- Trouver le plus petit $j / U \leq F(x_j)$

Aiguillage probabiliste à k sorties

Méthode de génération d'une sortie dans l'aiguillage probabiliste

On cumule les probabilités $P_1, P_2, ...P_k$ et on obtient

$$P_1^c = P_1$$
$$P_2^c = P_1 + P_2 = P_1^c + P_2$$
$$P_3^c = P_1 + P_2 + P_3 = P_2^c + P_3, \ ...$$
$$P_i^c = P_{i-1}^c + P_i, \ ...$$
$$P_k^c = P_{k-1}^c + P_k = 1$$

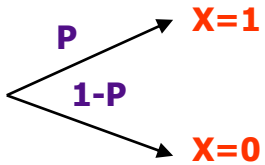
On tire un n.a. $U \in [0,1]$, puis si $P_{j-1}^c < U \leq P_j^c \Rightarrow$ on se dirige vers la sortie $A_j \Leftrightarrow X = x_j$

```
Int function aiguillage () ;

{  j=1; U= Uniforme();
While U >  Pc [ j ]
      j++;
return j;   // X = x [ j ]
}
```

Exemple

Soit X une v.a suivant une loi Bernoulli(p) définie par
P1 = P(X=1) = p et P2 = P(X=0) = 1-p.



On réalise un aiguillage à 2 sorties.

```
Int function Bernoulli () ;

{ if Uniforme() ≤ p then return 1 else return 0;
}
```

Application

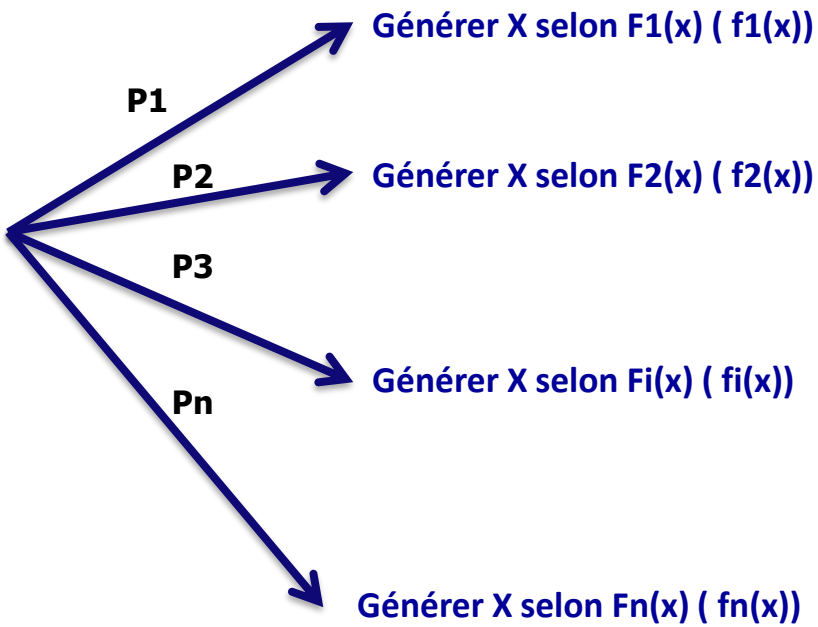
Soit X une v.a suivant une loi de Poisson définie par

$$Pr(X = k) = P_k = e^{-\lambda}\frac{\lambda^k}{k!}, k = 0, 1, 2, ...$$

NB. Il est important de calculer les Pk et de limiter l'ensemble des valeurs de X d'une manière efficace

Méthode de composition

Elle s'applique lorsque $F(x)$, resp. $f(x)$, peut s'exprimer sous la forme $F(x) = \sum_{j=1}^{\infty} P_j F_j(x)$, resp. $f(x) = \sum_{j=1}^{\infty} P_j f_j(x)$.




```
real function composition();
{ i = 1; U = Uniforme();
    While U > Pc[i]
        i++;
    générer X suivant F_i(x) (ou suivant f_i(x));
    return X;
}
```

Génération par des méthodes mixtes

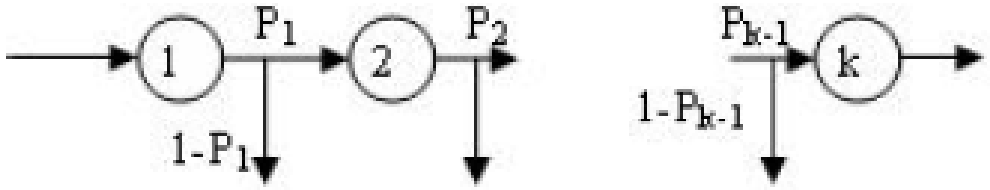
≡ Combinaison des méthodes simples (connues) et de l'aiguillage probabiliste ⟹ Lois Sophistiquées.

Exemples très simples (vus) :

1° loi ***Erlang-k*** (voir méthode de convolution), où on a k serveurs fictifs en série.

2° loi ***hyper-exponentielle-k*** (voir méthode de composition où les Fi(x) sont des exponentielles).

Exemple de loi plus sophistiquée : loi ***Cox-k***, dont chaque serveur est exponentiel et schématisée comme suit :




```
real function  cox-k () ;

{  j = 1;  X = - ln(Uniforme()) / lambda[1];
 while ( Uniforme()  ≤  P[ j ] )
   {  X = X - ln(Uniforme()) / lambda[j] ;
      j++;
   }

Return X ;
}
```

NB. On met P[k] = −1 et on éviterait ainsi le test sur le nombre d'étapes.

Méthode de rejet (ou d'acceptation)

Soit à générer une v.a. X de densité *f(x) bornée (f(x) ≤ B pour tout X)*

Si f(x) ≠ 0, a < x ≤ b, on peut faire ce changement de variable

$$0 \leq y \leq \frac{x-a}{b-a} \leq 1.$$

$$0 \leq f^*(y) = \frac{1}{B} f(a + (b-a)y) \leq 1.$$

Générer (x, f(x)) revient à générer (y, f*(y))

La méthode est :

- générer U1, U2 € U[0,1]
- si U2 ≤ f*(U1) ⟶ on accepte le couple et donc y = U1 ou x=a+U1*(b-a)

Fonction de génération par rejet

```
real function  rejet () ;

{  Do
        U1 = Uniforme() ; U2= (Uniforme();

  While ( U2 > f*(U1) )

  X =  a + U1*(b-a) ;

  Return X ;

}
```

5. Analyse opérationnelle et calcul des métriques

Cas d'un système simple

Mesures élémentaires retenues:

– Tamx : durée de simulation ;
– A : nombre total d'arrivées de requêtes ;
– D : nombre total de départs de requêtes ;
– X(n) : durée cumulée pendant laquelle le système contenait n requêtes ;
– Nmax : nombre maximum de requêtes dans le système.

A partir de ces mesures élémentaires, nous pouvons dégager les ***métriques de performance*** suivantes :

ds : Débit du système en sortie (nombre moyen de requêtes traitées par unité de temps) .

$$ds = D/Tmax.$$

Nm : Nombre moyen de requêtes dans le système. $Nm = \frac{\sum_{i=1}^{Nmax} iX(i)}{Tmax}.$

T : Temps moyen de réponse du système. $T = \frac{\sum_{i=1}^{Nmax} iX(i)}{D}.$

U : Taux d'utilisation ou d'occupation du système. $U = \frac{Tmax - X(0)}{Tmax}.$

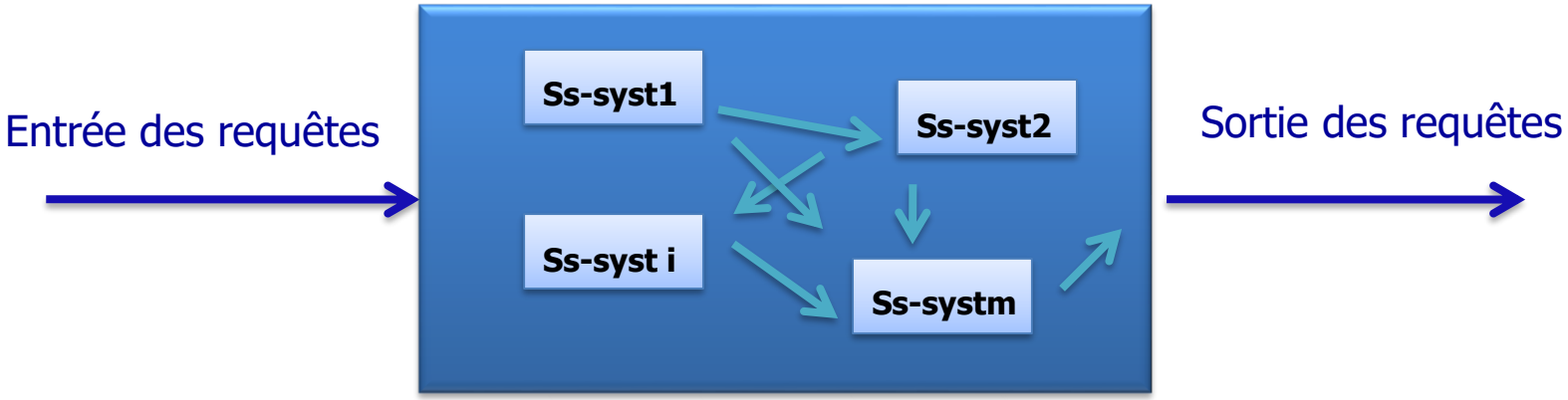
De ces formules nous obtenons la formule de Little opérationnelle.

$$Nm = ds * T.$$

L'équilibre opérationnelle du système conduit à l'observation : A ≈D

Indication. $\sum i * X(i) = \sum n_i \Delta(i)$, où $\Delta(i)$ est la durée entre deux changements d'état ($i-1$ et i) et n_i est nombre de requêtes se trouvant dans le système pendant la durée $\Delta(i)$.

Cas d'un système contenant *m* sous systèmes simples



Mesures élémentaires retenues:
– Tamx : durée de simulation ;
– D : nombre total de départs de requêtes ;
– D_i : nombre total de départs de requêtes du sous-système$_i$;
– $X_i(n)$: durée cumulée pendant laquelle le sous-système$_i$ contenait n requêtes ;
– Nmax : nombre maximum de requêtes dans le système.

On obtient les ***métriques de performance*** suivantes.

- dsi : Débit du sous-systèmei en sortie dsi = Di/Tmax.
- Ui : Taux d'utilisation du sous-systèmei $U_i = \frac{Tmax - X_i(0)}{Tmax}$.
- Si : Durée moyenne de traitement d'une requête par le sous-systèmei $S_i = \frac{Tmax - X_i(0)}{D_i}$.
- ei : Nombre moyen de visites au sous-systèmei par requête $e_i = \frac{D_i}{D}$.
- Nmi : Nombre moyen de requêtes dans le sous-systèmei $Nm_i = \frac{\sum_{j=1}^{Nmax} jX_i(j)}{Tmax}$.
- Ti : Temps moyen de réponse du sous-systèmei . $T_i = \frac{\sum_{j=1}^{Nmax} jX_i(j)}{D_i}$
- ds : Débit du système en sortie ds = dsi/ei
- Nm : Nombre moyen de requêtes dans le système $Nm = \sum_{i=1}^{m} Nm_i$.
- T : Temps moyen de réponse du système : $T = \sum_{i=1}^{m} e_i T_i$.

La formule de Little est donnée : $Nm = ds * T$.

6. Exemples de simulation de systèmes et réseaux

Application à la simulation d'une file d'attente

On peut considérer uniquement 2 événements : **Ea** : **« arrivée d'un client dans la file »** et **Es** : « **fin de service ou départ d'un client** ». **Pa** et **Ps** les **procédures** correspondantes.

```
Procédure simulation () ;
{
   while (non-fin-de-simulation)
    {
       If Ea {
          Pa
        }
        else {
          Ps} ;
    }
   Calcul des statistiques et
Affichage des résultats ;
}
FIN.
```

Dans la Ps, il faut tester si la file est vide; dans ce cas il faut bloquer le serveur pour éviter qu'une fin de service ne s'exécute.

If file-vide { bloquer-serveur};

Dans Pa, il faut vérifier si le serveur est bloqué; dans ce cas il faut le libérer.

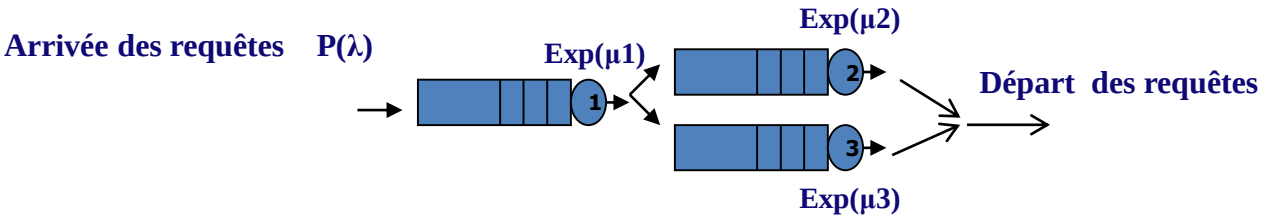
If serveur-bloqué Then libérer-serveur;

Les variables utilisées :

- Horloge : date de la simulation (qui est aussi la date de réalisation d'un événement).
- n : nombre de clients dans le système.
- Ta : instant de réalisation de l'événement Ea (date de l'arrivée d'un client).
- Ts : instant de réalisation de l'événement Es (date de la fin de service d'un client)
- b : temps cumulé pendant lequel le système est actif.
- tb : date de réactivation du serveur.
- Hp : date de réalisation de l'événement précédent (valeur précédente de Horloge).
- $x = \sum_{i=1}^{Nmax} iX(i) = \sum n_j \Delta(j)$

```
Procédure Simulation M/M/1() ;
{Lambda = 1/500; mu = 1/200; Tmax = 30000; n = 0; x = 0; D = 0;
b = 0; tb = 0; Horloge = 0; Hp = 0; ta = 0; ts = Tmax;
   while Horloge < Tamx
   {
      If Ta ≤ Ts
      {Horloge = ta;
         x = x + n * (Horloge − Hp);
         n++;
         ta = Horloge − ln(Uniforme())/Lambda;
         Hp = Horloge;
         If (n == 1)        /  libérer serveur
            {ts = Horloge − ln(Uniforme())/mu;
            tb = Horloge};
      }
      Else
      {Horloge=ts;
         x = x + n * (Horloge − Hp);
         n − −;
         D++;
         Hp = Horloge;
         If (n == 0)        //    bloquer serveur;
            { ts = Tmax;
            b = b + (Horloge − tb); }
         Else ts = Horloge − ln(Uniforme())/mu;
      }
   }
   T=x/D; Nm=x/Tmax; ds=D/Tmax; U=b/Tmax;
   Print (T,Nm,ds,U);
}
```

Application à la simulation d'un modèle de RFA



On considère les 4 événements significatifs suivants:

– Ea : "l'événement d'arrivée dans le réseau (via la station1)."
– Es1 : "l'événement de fin de service dans la station1".
– Es2 : "l'événement de fin de service dans la station2".
– Es3 : "l'événement de fin de service dans la station3".

```
Procédure Simulation reseau() ;
{Initialisations;
   while Horloge < Tamx
   { Horloge = min(ta, ts1, ts2, ts3)
      If Horloge = Ta
         { x[1] = x[1] + n[1] * (Horloge − Hp1);
         n[1] + +;
         Ta = Horloge − ln(Uniforme())/Lambda;
         Hp1 = Horloge;
         If (n[1] == 1)
            { ts1 = Horloge − ln(Uniforme())/mu1;
            tb1 = Horloge; }
         }
      If Horloge = Ts1
            { x[1] = x[1] + n[1] * (Horloge − Hp1);
            n[1] − −;
            D[1] + +;
            Hp1 = Horloge;
            If (n[1] == 0)
               { ts1 = Tmax; b[1] = b[1] + (Horloge − tb1); }
            Else ts1 = Horloge − ln(Uniforme())/mu1;
```

```
// réaliser aiguillage à 2 sorties;
 If (Uniforme() <= P)      // Entrée dans stations2;
    { x[2] = x[2] + n[2] * (Horloge − Hp2);
    n[2] + +;
    Hp2 = Horloge;
    If (n[2] == 1)      // libérer serveur;
       { ts2 = Horloge − ln(Uniforme())/mu2;
       tb2=Horloge; }
    }
 Else // Entrée dans stations3;
    { x[3] = x[3] + n[3] * (Horloge − Hp3);
    n[3] + +;
    Hp3 = Horloge;
    If (n[3] == 1)      // libérer serveur;
       {ts3 = Horloge − ln(Uniforme())/mu3;
       tb3=Horloge; }
    }
```

```
    If Horloge = Ts2
          { x[2] = x[2] + n[2] * (Horloge - Hp2);
          n[2] - -;;
          D[2] + +;
          Hp2 = Horloge;
          If (n[2] == 0)      / /   bloquer serveur;
              { ts2 = Tmax; b[2] = b[2] + (Horloge - tb2); }
          Else ts2 = Horloge - ln(Uniforme())/mu2;
       }
    If Horloge = Ts3
          { x[3] = x[3] + n[3] * (Horloge - Hp3);
          n[3] - -;
          D[3] + +;
          Hp3 = Horloge;
          If (n[3] == 0)      / /   bloquer serveur;
              { ts3 = Tmax; b[3] = b[3] + (Horloge - tb3); }
          Else ts3 = Horloge - ln(Uniforme())/mu3;
       }
  } / / fin while
  D = D[2] + D[3]; Nm = 0, T = 0;
  for (i = 1; i < 4; i + +)
      { T[i] = x[i]/D[i]; Nm[i] = x[i]/Tmax; ds[i] = D[i]/Tmax;
      U[i] = b[i]/Tmax; e[i] = D[i]/D; Nm = Nm + Nm[i];
      T = T + e[i] * T[i] }
  ds = D/Tmax;
}
```

Application à la simulation d'une CMTD

Soit X une CM à espace d'états finis E = 1, 2, ..., n de matrice de transition
$P = [Pij]_{nXn}$. $P_{ij} = Pr(X_n = j / X_{n-1} = i)$ et $\Sigma^n_{j=1} P_{ij} = 1, \quad i = 1, 2, ..., n$

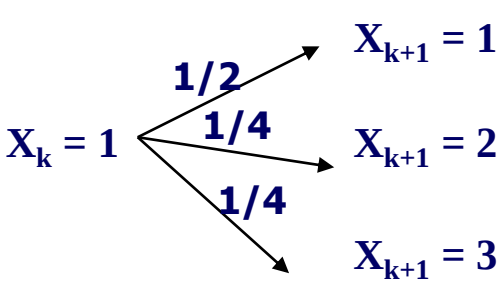
Chaque ligne de la matrice de transition représente une probabilité discrète pour *i* fixée, que l'on peut générer en utilisant la fonction *aiguillage()*.

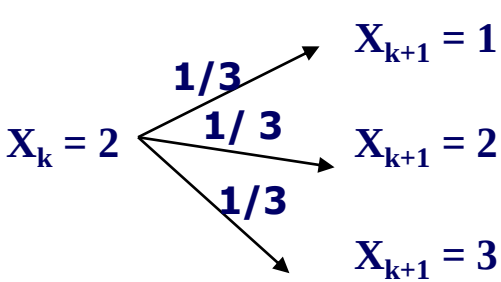
Simuler X $\leftrightarrow$ calculer $X_1, X_2, \ldots, X_n$ en démarrant à partir de X_0.

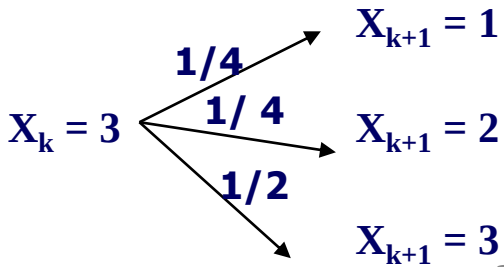
Exemple. Soit une CMTD X (E = 1, 2, 3) et P sa matrice de transition.

$$P = \begin{pmatrix} 1/2 & 1/4 & 1/4 \\ 1/3 & 1/3 & 1/3 \\ 1/4 & 1/4 & 1/2 \end{pmatrix}$$

Si







Simuler X ←→ calculer $X_1, X_2, \ldots, X_n$ en démarrant à partir de X_0.

$$P_c = \begin{pmatrix} 1/2 & 3/4 & 1 \\ 1/3 & 2/3 & 1 \\ 1/4 & 1/2 & 1 \end{pmatrix}$$

On suppose que $X_0 = 1$;

U= Uniforme() si U <= ½ alors $X_1 = 1$
sinon si U <= ¾ alors $X_1 = 2$
sinon $X_1 = 3$

On détermine de la même façon X2 et ainsi de suite, ...

En simulant la CMTD X, on peut obtenir les mesures de performance suivantes:

- Nombre de passages dans chaque état (visites de chaque état);
- Probabilité stationnaire ou d'état ;

Données :

N[1..100] : tableau qui contient le nombre de visites de chaque état
(**N[i]** : nombre de visites de l'état **i**).

P[1..100] : tableau qui contient les probabilités d'état ;
(P[i] = probabilité d'être à l'état i ou encore $P[i] = \lim_{n\to\infty} P(X_n = i)$.

```
Algorithm simul-CMTD ;
    {
    for (i = 1; i < 101; i++) N[i] = 0;
    i = 1; time = 0; \\ time : horloge;
    While (time <   tmax)
        {
        N[i]++;
        U = Uniforme();
        j = 1; \\ on cherche l'état prochain
        While (U > P_c[i, j]) j++;
        i = j;
        time++;
       }
    for(i = 1; i < 101; i++) P[i] = N[i]/tmax;
    }
```

Simulation de la CMTD de M/M/1

On distingue 2 types de sauts :

+1 correspond à l'arrivée d'un client avec une probabilité $P_{ii+1} = \lambda/(\mu + \lambda)$;

-1 correspond à une fin de service avec une probabilité $\mu/(\mu + \lambda)$;

Mesures de performance:

– le nombre de visites de chaque état (N[i]= nombre de visites de l'état i);

– la probabilité stationnaire ou d'état (P[i] = probabilité de l'état i) ;

– le nombre moyen de clients nb = $\Sigma i * P[i]$.

```
Algorithm simul-CM-M/M/1 ;
{
for (i = 1; i < 21; i++) N[i] = 0;
etat = 0; time = 0 ;
While (time < tmax)
    {
   U = Uniforme() ; N[etat]++;
   If U <= Lambda/(lambda + mu)
     {
       etat++;
       Else If etat! = 0   etat--;

     }
   time++;          }
nb = 0;
for (i = 1; i < 21; i++)
{ P[i] = N[i]/Tmax; nb = nb + i * P[i] }
Print (nb);      }
```