

# **Simulation**

Jean-Michel Fournéau

**PRiSM**

Université de Versailles  
45, Av. des Etats-Unis  
F-78000 Versailles  
email:jmf@prism.uvsq.fr

Partie I

## Motivation

- Construire des systèmes virtuels et les faire évoluer avec ou sans interactions avec des utilisateurs.
- Trouver de l'information utile à partir :
  - du point d'arrivée
  - de la trajectoire
  - d'un point extrême

## **Simulation Interactive**

Souvent réalisée par un hardware spécialisé

- Jeux...
- Entrainement (Aviation, Espace)
- Militaire (Wargame, DIS-HLA)

## Sans Interaction

- Simulation d'un processus  
(le temps et l'espace sont pris en compte)
- Simulation d'une variable aléatoire : Monte Carlo  
seul l'espace est pris en compte

## Méthodes de Monte Carlo

Version élémentaire

- Calculer l'intégrale d'une fonction éventuellement difficile sur une région peu régulière.
- Echantillonner des objets vérifiant une propriété, objets issus d'un ensemble fini de taille  $m$  mais où la complexité de génération et de recensement est trop importante.
- Echantillonner des objets issus d'un mécanisme de sélection qui a eu pour effet de modifier la distribution initiale

Construire un échantillon et l'utiliser pour des calculs (moyenne, étendue, mode, variance, quantiles, etc...) et des tests.

## Exemples

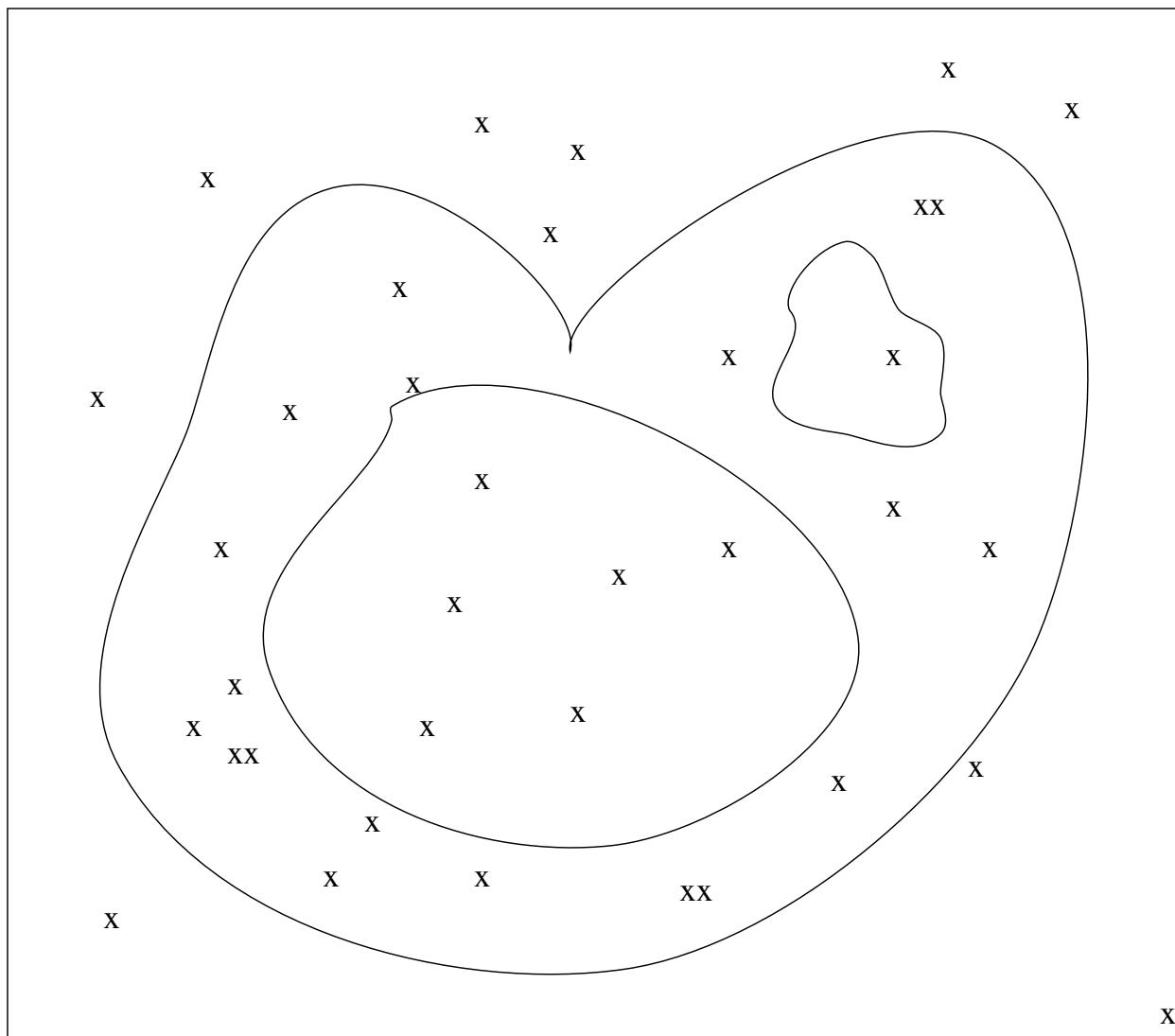
- Probabilité de déconnecter un réseau de transport par des pannes de liens
- Calculer le nombre de particules qui traversent une paroi
- Calculer la distribution du temps pour accomplir un projet de plusieurs tâches
- Construire un mot uniforme sur un ensemble de lettres avec une grammaire (génétique)

## Algorithme pour le Volume

1.  $j := 1; S := 0;$
2. tant que  $j \leq n$  faire
  - (a) Générer un point  $x^{(j)}$  uniforme sur  $[0, 1]^m$
  - (b) si  $x^{(j)} \in \mathcal{R}$  alors  $\phi = 1$  sinon  $\phi = 0$
  - (c)  $S := S + \phi; j := j + 1;$

Plus rapide à converger que la version déterministe si  $m > 2$ .

Hypothèse : la fonction  $(x^{(j)} \in \mathcal{R})$  est relativement facile à calculer.



## Fiabilité d'un réseau

On considère un réseau (sommets, arêtes). Il y a deux sommets particuliers  $s$  et  $t$  (extrémités du réseau).

- Les sommets sont fiables.
- Les arêtes peuvent tomber en panne (proba  $1 - q_i$ ).
- Pannes indépendantes.

$y_i = 1$  : arête OK (0 sinon).

$y = (y_1, \dots, y_n)$  un état du réseau.  $q$  le vecteur  $(q_1, \dots, q_n)$ .

$\phi(y)$  vaut 1 si on peut connecter  $s$  à  $t$  par le réseau lorsque la configuration de panne  $y$  se produit.

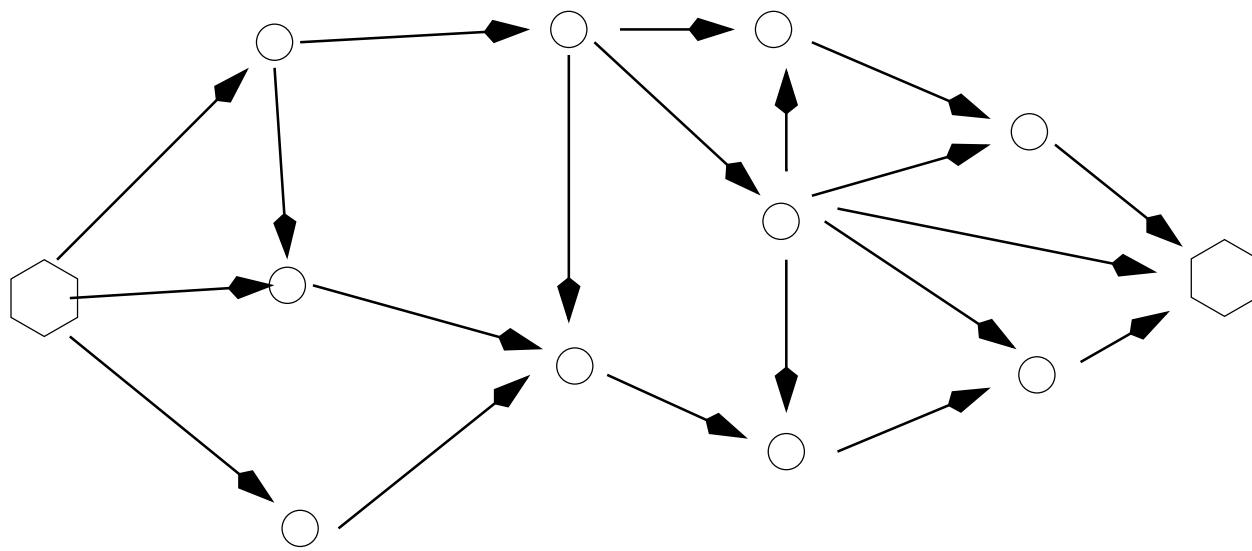
Probabilité d'une configuration de panne

$$P(y, q) = \prod_i q_i^{y_i} (1 - q_i)^{1-y_i}$$

On veut calculer  $g(q) = \sum_y \phi(y) P(y, q)$

Problème : il n'y a pas d'algorithme pour calculer  $g(q)$  en un temps polynomial (en fonction des tailles du problème).

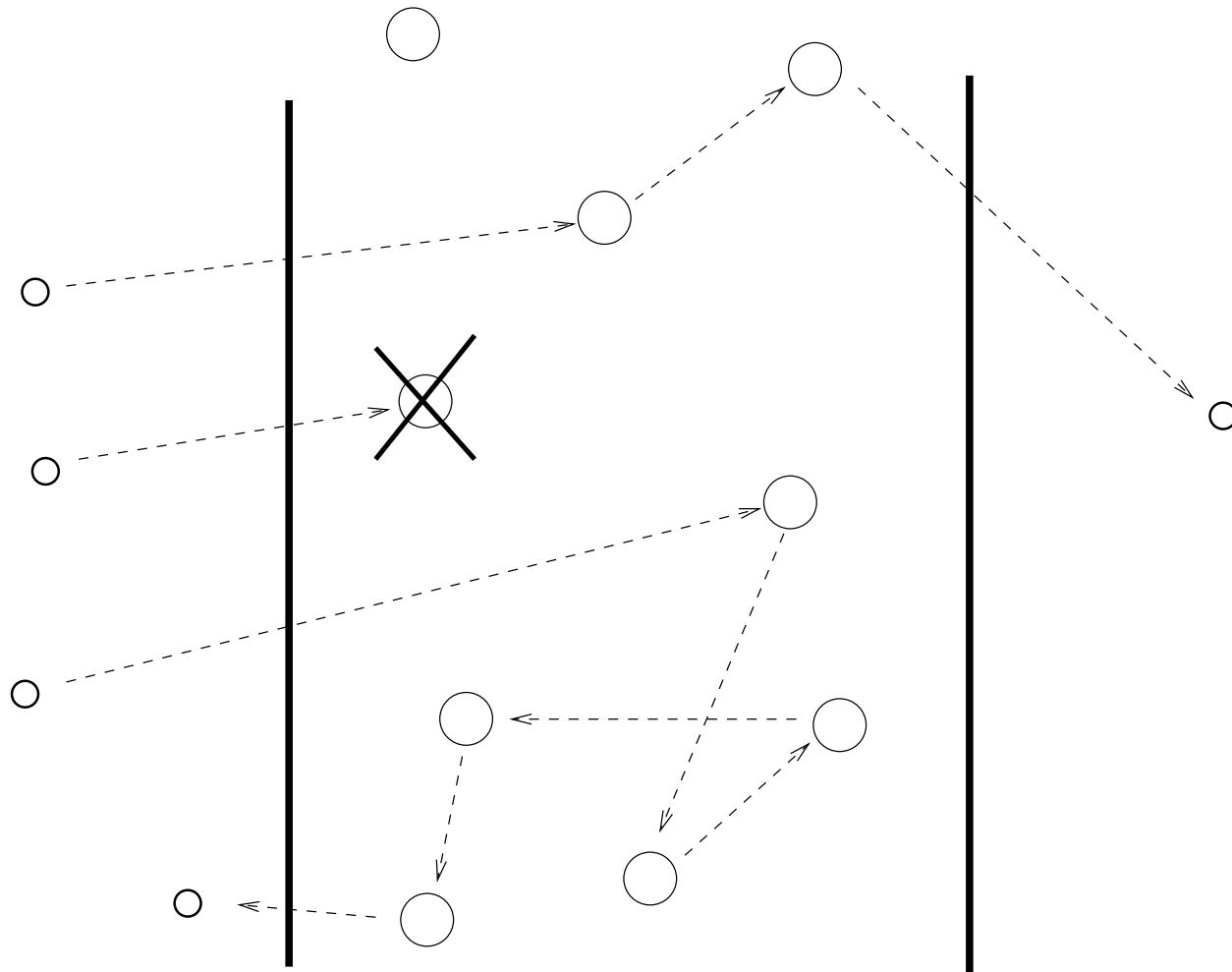
Monte Carlo : représenter le problème comme un calcul de volume



## Comptage de Particules

Compter le nombre de particules traversant une paroi caractérisée par son épaisseur et le type de choc

- On simule particule par particule
- Une particule a une position et un angle par rapport à la paroi
- La paroi est composée d'atome. Il y a interaction atome-particule.
- A chaque choc entre la particule et un atome, on peut avoir
  - destruction de la particule
  - rebond (la particule change d'angle et poursuit son chemin)
- Si suite aux chocs la particules sort de la paroi, on vérifie le côté.
- on compte les particules qui ont traversé la paroi.



## Exemple : Paroi et Particules

```
program MonteCarloPhysique(input, output);
const  etapemax= 10;lambda=2.0;
       e=3.0; {epaisseur} p=0.1; {probabilite de disparition}
       exp = 1000; {nombre d'expériences}
var   i,j,k, etape, NumSorti, NumReflechi, NumAbsorbe: integer;
      u,y,z, X : real; abs : boolean;
begin
  NumSorti:=0; NumReflechi:=0; NumAbsorbe:=0;
  for i:=1 to exp do begin
    X:=0.0; etape:=0; abs:=false;
    repeat
      etape:=etape+1;
{Calcul de la distance avant le prochain choc}
      y:=-ln(random(z))/lambda ;
{Calcul du cosinus de l'angle}
      u:=2*random(z)-1;
{Mise à jour de la distance sur l'axe des X}
      X:=X+y*u;
{si la particule est sortie, ne rien faire}
{sinon tester la disparition}
      if (X>0.0) and (X<e) then
        if (random(z)<p) then abs:=true;
      until (X<0.0) or (X>e) or abs or (etape>etapemax);
      if X<0.0 then NumReflechi:=NumReflechi+1
      else if X>e then NumSorti:=NumSorti+1
      else if abs then NumAbsorbe:=NumAbsorbe+1
    end;
end.
```

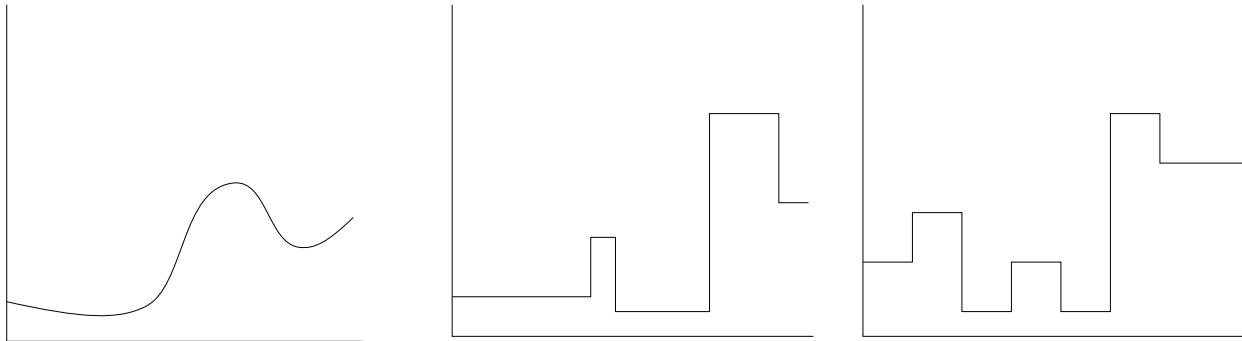
## Simulation d'un processus

Suivre une ou plusieurs *trajectoires* de manière à déterminer l'*évolution* de *systèmes complexes*.

En déduire,

- performances (débit, taux de perte, temps de réponse),
- fiabilité (temps moyen entre deux pannes),
- spécification (un état est-il atteignable ?).

- trajectoire : c'est à dire un couple : (espace, temps).
- espace : discret (ED) ou continu (EC).
- temps : 3 possibilités :
  - temps discret (TD) (clics d'une horloge),
  - temps continu (TC) (temps physique)
  - temps continu où les évolutions n'ont lieu qu'à des instants discrets (événements discrets : EvD).



**EC/TC** positions d'objets en interaction gravitationnelle.

**EC/TD** un modèle journalier de remplissage d'un barrage.

**ED/TC** le nombre de particules au cours d'une réaction atomique.

**ED/EvtD** le nombre de clients dans une file d'attente.

**ED/TD** le gain d'un joueur à chaque étape.

- systèmes complexes : systèmes dont les équations d'évolution sont difficiles, voire impossibles, à résoudre analytiquement, même si elles paraissent simples.  
exemple : Buffer de taille  $B$ , séquence d'arrivées indicées par la date, service déterministe, temps discret (durée du service), service avant les arrivées.

état du buffer =  $X_t$  : nombre de paquets dans le buffer à la date  $t$

$$X_{t+1} = \min(B, \max(0, X_t - 1) + Arr(t))$$

où  $Arr(t)$ , nombre d'arrivées entre  $t - 1$  et  $t$ .

- L'évolution complexe et en partie aléatoire. Deux raisons :
  - connaissance insuffisante des données de l'expérience
  - théorie du chaos déterministe : une petite incertitude de mesure peut avoir des conséquences importantes sur la trajectoire.

Distributions de probabilités **discrètes** ou **continues** pour représenter certaines parties du phénomène.

Problème (très délicat) de l'adéquation des distribution théoriques aux distributions empiriques et plus généralement aux spécifications incomplètes.

- évolution : deux types de résultats :
  - Transitoire : état atteint à la date  $T$ , ou durée pour arriver dans un état (Fiabilité).
  - Stationnaire : distribution du temps passé dans les différents états entre 0 et  $+\infty$  (Performance).

Problème : on a un temps de simulation fini pour estimer une moyenne sur un temps infini

On se restreint aux systèmes à espace discret, à évolution aléatoire et à événement discret ou à temps discret.

Modèles de systèmes informatiques et de réseaux

## Contenu

Quatre parties (pas nécessairement traitées dans cet ordre)

1. Informatique : langage et système de simulation (QNAP II), structures de données
2. Algorithmique pseudo-numérique : génération de nombres aléatoires uniformes
3. Statistiques : mesurer la qualité des résultats (intervalle de confiance) et des sources aléatoires, étudier la représentativité de la trajectoire, déduction et inférence, organisation des expériences
4. Probabilités : Modèles de distributions et génération.

## Plan d'une étude par simulation

1. Ecrire le simulateur
2. Tester le simulateur en contrôlant l'aléatoire
3. Choix des paramètres, Plan d'expériences
4. Analyse des résultats, Induction et inférence
5. Recommencer en 3 ou en 1...

## Ecrire un simulateur

- Langage usuel (C, C++, java, Ada) : code généré très efficace, temps de développement très long, approche très générale
- Bibliothèque dans un langage usuel : le meilleur compromis si la bibliothèque est bonne
- Langage spécialisé (simula) : temps d'apprentissage long, efficacité moyenne, approche limitée à un formalisme, pas de solveur intégré
- Système spécialisée : langage + solveur : temps de développement très court, efficacité très moyenne du code, formalisme restreint (QNAP, Stella)

# Stella

Simulation de fluides représentant des populations ou des quantités  
(équations différentielles)

- Interface graphique
- 4 types d'objet :
  - Réservoir (contient le fluide)
  - Flux (transporte du fluide, selon débit)
  - Connecteur (transporte des valeurs)
  - Calculateur (calcule un débit fonction des connecteurs)
- un fluide est infiniment divisible...
- déterministe, non linéaire
- boucles de réaction et de renforcement

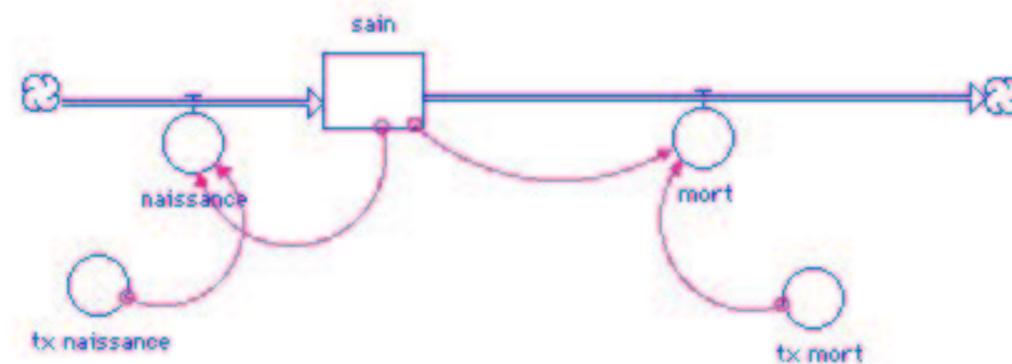
## Exemple

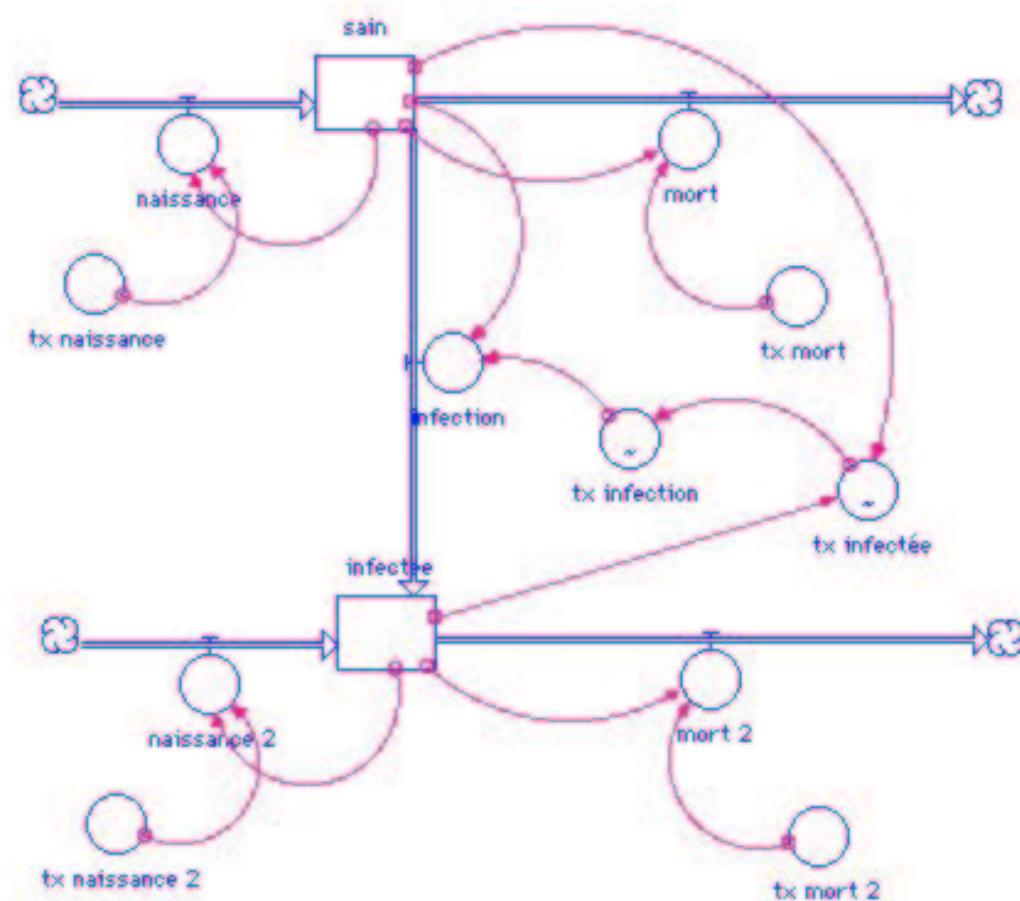
- Un modèle épidémiologique
- une population saine avec naissance et mort
- une infection dont le risque individuel est croissant en fonction de la population infectée.
- Pas de possibilité de guérison
- Une naissance dans la population infectée est infectée
- taux de mort plus important dans la population infectée

## Les objets du modèle

- 2 stocks : la population saine et la poputlation infectée
- 2 flux de naissance (calculateur+flux+taux+connecteurs),
- 2 flux de morts (calculateur+flux+taux+connecteurs),
- 1 flux d'infection

## Modèle de la population saine





## La version texte

```
infectee(t) = infectee(t - dt) + (naissance_2 + infection - mort_2) * dt
INIT infectee = 1000

naissance_2 = tx_naisance_2*infectee
infection = sain*tx_infection*0.001
mort_2 = infectee*tx_mort_2

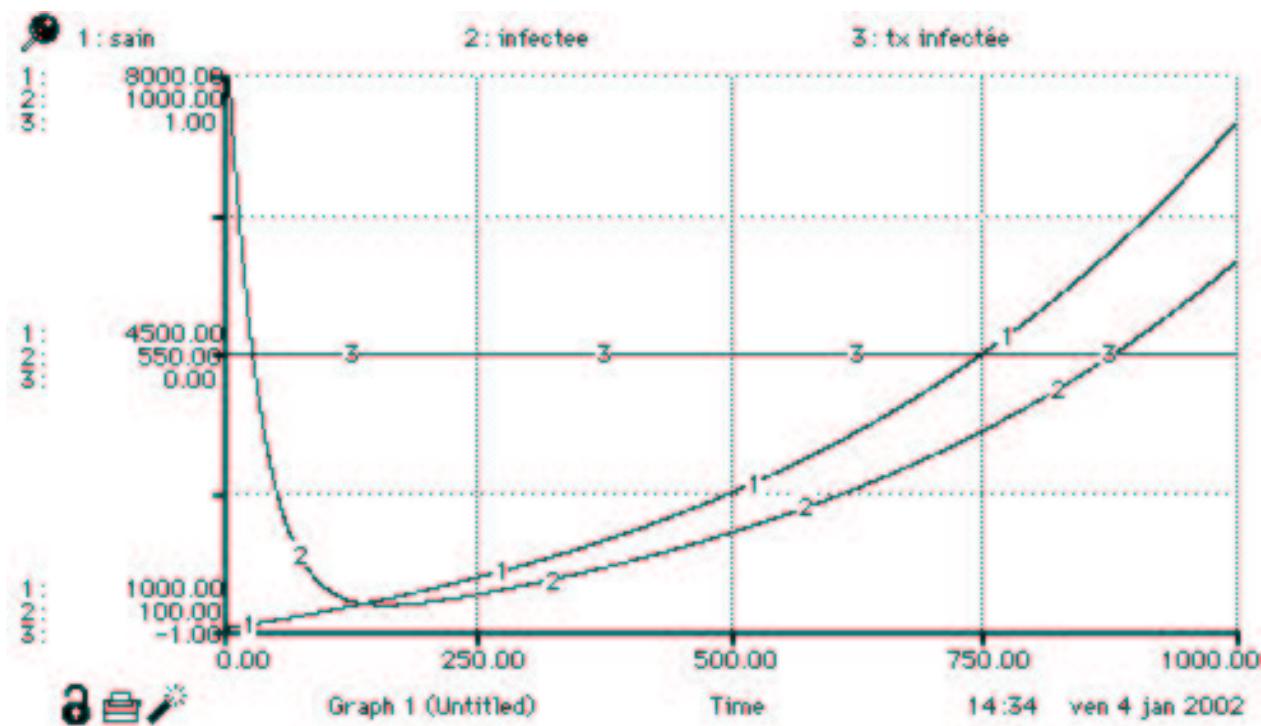
sain(t) = sain(t - dt) + (naissance - mort - infection) * dt
INIT sain = 1000

naissance = tx_naisance*sain
mort = sain*tx_mort
infection = sain*tx_infection*0.001
tx_infectee = infectee/(infectee+sain)

tx_mort = 0.005
tx_mort_2 = 0.03
tx_naisance = 0.01
tx_naisance_2 = 0.0001

tx_infection = GRAPH(tx_infectee)
(0.00, 3.00), (0.1, 4.00), (0.2, 5.00), (0.3, 6.00), (0.4, 7.00),
(0.5, 10.5), (0.6, 15.5), (0.7, 24.0), (0.8, 35.0), (0.9, 60.5),
(1, 98.0)
```

## Résultats



## **Network Simulator (NS2)**

- Simulation de réseaux
- Niveau Simple : définir une topologie, les protocoles (parmi ceux déjà codés) à l'aide de scripts.
- Expert : coder un nouveau protocole (C++)
- Initialement pour tester les protocoles Internet, puis Wireless, Réseaux Optiques, Ad-Hoc, etc

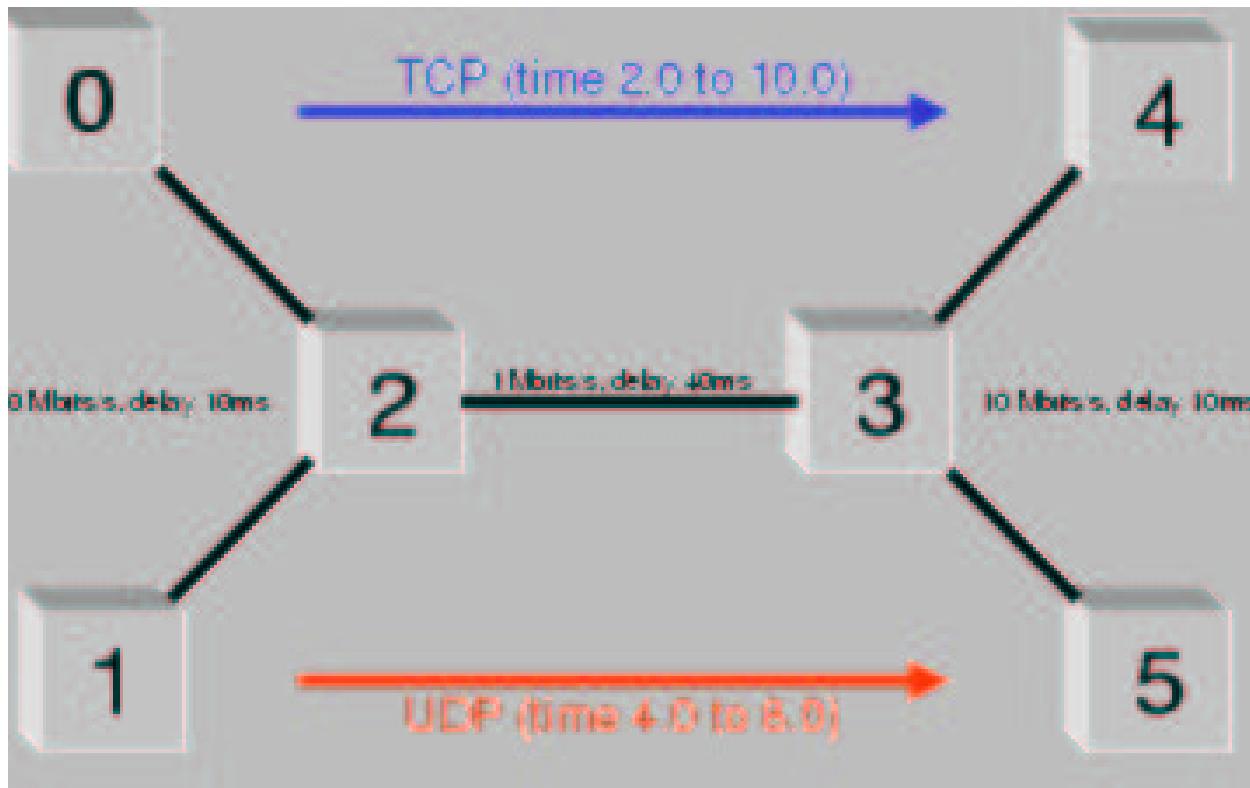


Figure 1: Topologie et Protocole

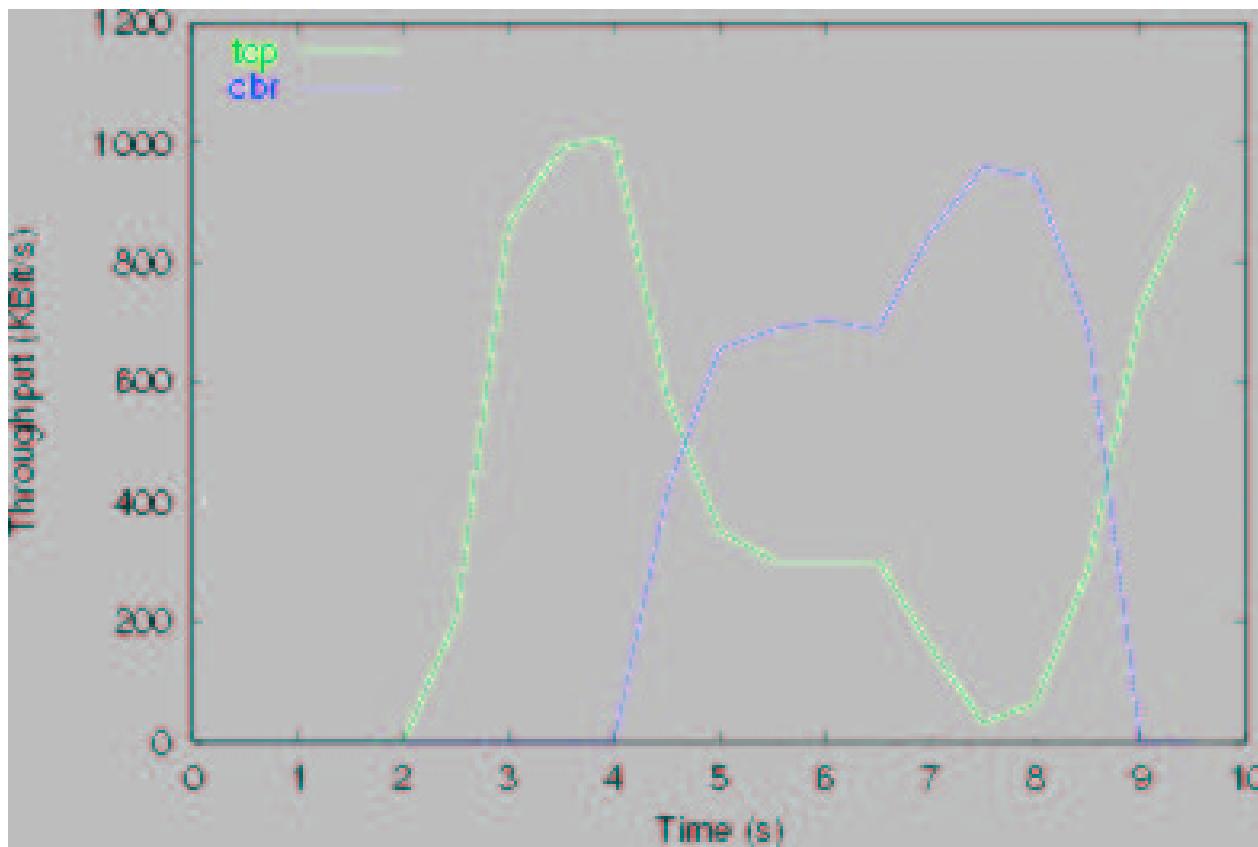


Figure 2: Résultats

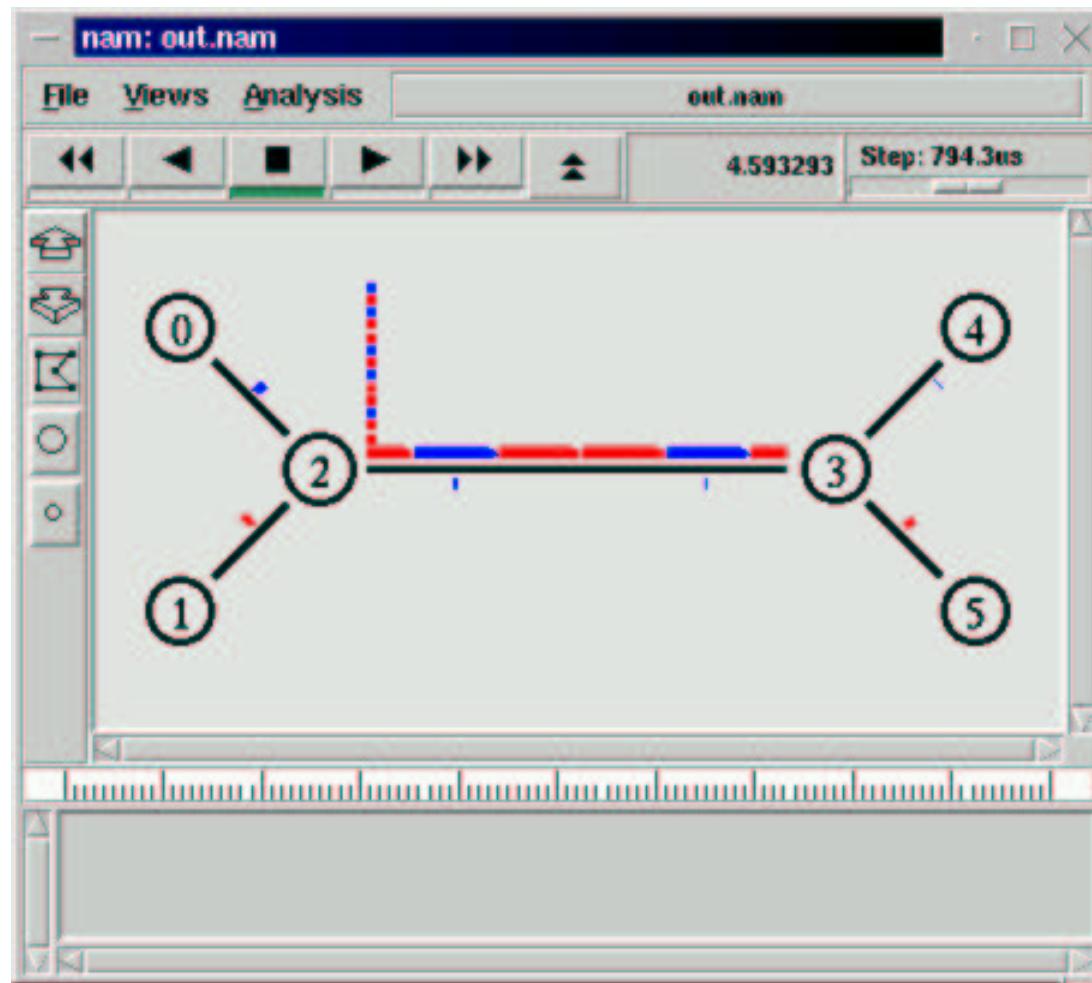


Figure 3: Nam: visualisation et animation

## En C/C++, Java

Choisir un état initial

Loop

1. Générer le contrôle aléatoire guidant la trajectoire
2. Changer d'état
3. Mettre à jour les variables d'observations statistiques (temps passé dans chaque état, nombre de franchissements de certaines transitions)
4. Vérifier la qualité de la trajectoire (intervalle de confiance)  
end Loop (sortir si la qualité des résultats est suffisante)

## Modélisation

- Déterminer la description des états
- Déterminer les règles d'évolution (transition, modification)
- Déterminer ce qui doit être mesuré

## Exemple : Joueurs

- 2 joueurs  $A$  et  $B$  jouent à pile ou face.
- Pour chaque mise, Proba  $p$  de faire gagner  $A$
- Mise de 1 unité.
- Fortune de  $A$  (resp.  $B$ ) =  $a$  (resp.  $b$ ).
- Ruine lorsque la fortune est nulle

- Etat : fortune de  $A$
- Transition :  $a \rightarrow a + 1$  avec proba  $p$  et  $a \rightarrow a - 1$  avec prob  $1 - p$
- condition de fin de la simulation : ruine ou date finale atteinte
- mesure : temps passé dans l'état  $a$ , délai avant ruine de  $A$  ou de  $B$ , probabilité de ruine de  $A$  ou de  $B$

## Rappel de Probabilités

Fonction de Répartition :  $F_X(a) = \Pr(X \leq a)$

$F$  est positive et croissante,  $F(-\infty) = 0$ ,  $F(+\infty) = 1$

- **espace continu** Densité de probabilité :

$$f(X) = \frac{dF_X}{dX}$$

Propriétés :  $f()$  est positive et  $\int_{-\infty}^{+\infty} f(x)dx = 1$

- **espace discret** Distribution de probabilité :

états possibles :  $x_1 \dots x_N$  avec les probabilités  $p_1 \dots p_N$ .

Propriétés :  $p_i \geq 0$  et  $\sum_{i=1}^N p_i = 1$

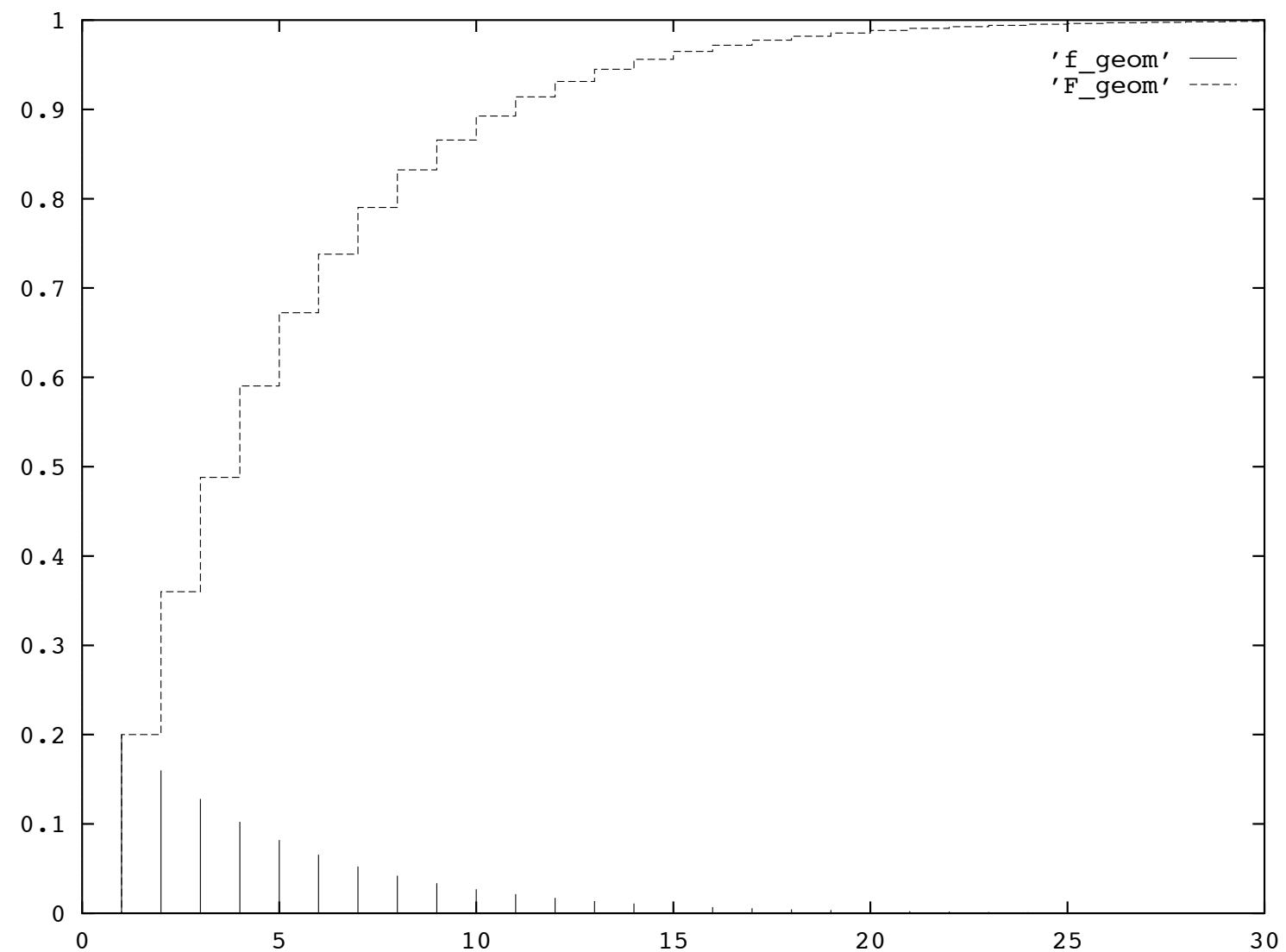


Figure 6: distribution géométrique tronquée et répartition

Espérance : ( $E[X]$ )

- Discret :  $E[X] = \sum_{i=1}^N p_i x_i$
- Continu :  $E[X] = \int_{-\infty}^{+\infty} x f(x) dx$

Variance d'une distribution :  $\text{Var}[X]$

- Discret :  $\text{Var}[X] = \sum_{i=1}^N p_i (x_i - E[X])^2$
- Continu :  $\text{Var}[X] = \int_{-\infty}^{+\infty} (x - E[X])^2 f(x) dx$

Ecart Type :  $\sigma[X] = \sqrt{\text{Var}[X]}$

Coéfficient de Variation : ( pour normaliser la variance )  $C = \frac{\sigma}{E[X]}$

Covariance : Soit deux v.a.  $X$  et  $Y$  de moyenne  $E[X]$  et  $E[Y]$

$$Cov(X, Y) = E[(X - E[X])(Y - E[Y])] = E[XY] - E[X]E[Y]$$

**Propriété 1** *Si  $X$  et  $Y$  sont indépendants alors  $Cov(X, Y)$  est nulle.*

**Propriété 2** *La réciproque est fausse.*

Malgré cela on calculera la covariance (ou plutot la corrélation) comme estimateur de l'indépendance.

Corrélation : Normalisation de la covariance entre -1 et 1

$$Cor(X, Y) = \frac{Cov[X, Y]}{\sigma[X]\sigma[Y]}$$

Autocorrélation d'ordre  $k$  : la corrélation du processus  $X_t$  avec le processus décalé de  $k$  dans le temps  $X_{t+k}$ .

Propriétés :

- Pour toutes v.a.  $X$  et  $Y$

$$E[a * X + b * Y] = a * E[X] + b * E[Y]$$

- Pour  $X$  et  $Y$  indépendantes

$$\text{Var}[a * X + b * Y] = a^2 \text{Var}[X] + b^2 \text{Var}[Y]$$

- Pour  $X$  et  $Y$  indépendantes  $E[XY] = E[X]E[Y]$

Attention, si  $X$  et  $Y$  ne sont pas indépendantes les deux dernières propriétés sont fausses.

**Erreur Courante** : Calcul d'un Débit  $E[\text{Quantite}/\text{Temps}]$  par la formule  $\frac{E[\text{Quantite}]}{E[\text{Temps}]}$ . Or le temps est en général DÉPENDANT de la quantité à transmettre...

## Trajectoire Aléatoire : représentativité, incertitude ?

- Une simulation est une visite aléatoire d'un immense espace d'états.
- On mesure le temps passé dans chaque état, le temps avant d'atteindre un état, les fréquences de passage entre groupes.
- Les états atteignable mais non visités auront des résultats faux...

Problématique des événements rares (recherche actuelle)

Multiplier les expériences afin d'estimer leur représentativité et de calculer une incertitude

Multiplication : Duplication des expériences ou Division  
“Virtuelle” d'une seule expérience

Intervalle de confiance  $[a, b]$  à q% pour  $X$ .

$$Pr(a \leq X \leq b) \geq q$$

## Base mathématique des intervalles de confiance

**Théorème 1 (Central Limite)** *La moyenne de  $n$  tirages indépendants dans une population homogène dont les caractéristiques sont une moyenne  $\mu$  et un écart type  $\sigma$  finis tend, lorsque  $n$  tend vers l'infini, vers une loi Normale de moyenne  $\mu$  et d'écart type  $\frac{\sigma}{\sqrt{n}}$ .*

Utilisation :

- une simulation = un tirage
- population : toutes les simulations du modèle

ATTENTION au problème de la finitude des  $\sigma$  : (exemple, les processus "self-similar" ou "heavy-tail" mesurés sur le WEB ou sur les systèmes de fichiers)

## Exemple de loi posant problème

- Loi puissance :  $p(x) = cx^{-\alpha}$  pour  $\alpha > 1$ .
- Espérance :  $c \sum_i ip(i) = \sum_i i^{1-\alpha}$
- Pour  $\alpha \leq 2$ ,  $E = \infty$
- Deuxième moment :  $c \sum_i i^2 p(i) = \sum_i i^{2-\alpha}$  diverge si  $\alpha \leq 3$
- Il est difficile de distinguer sur un petit échantillon entre une loi puissance et une exponentielle.

## Formule de Little

Soit  $\mathcal{Z}$  un système dans lequel arrive des objets. Ces objets demeurent un certain temps dans le système puis sortent de  $\mathcal{Z}$ .

- $\overline{N}$  est le nombre moyen d'objets dans  $\mathcal{Z}$
- $\overline{T}$  est le temps moyen passé dans  $\mathcal{Z}$  par un objet
- $\lambda$  est le taux d'arrivée en  $\mathcal{Z}$  des objets.

$$\overline{N} = \lambda \overline{T} \tag{1}$$

## Preuve de Little pour une simulation avec retour à 0

Considérons une trajectoire du système. Observons le système lorsqu'il se vide pour la  $n$ -ième fois à l'instant  $x$ . Il était vide à l'origine. Il y a eu  $m$  objets visiteurs entre 0 et  $x$ .

Soient  $a_i$  et  $d_i$  la date d'arrivée et de départ du  $i$ -ème client. Le temps de séjour du  $i$ -ème client vaut  $w_i = d_i - a_i$ .

$\bar{T}$  est la moyenne des  $w_i$ .

$$\bar{T} = \frac{\sum_{i=1}^m w_i}{m}$$

$N(t)$  est le nombre d'objets à la date  $t$ .

$$\bar{N} = \frac{\int_{t=0}^x N(t)dt}{x}$$

## Preuve de Little pour une simulation avec retour à 0 - suite

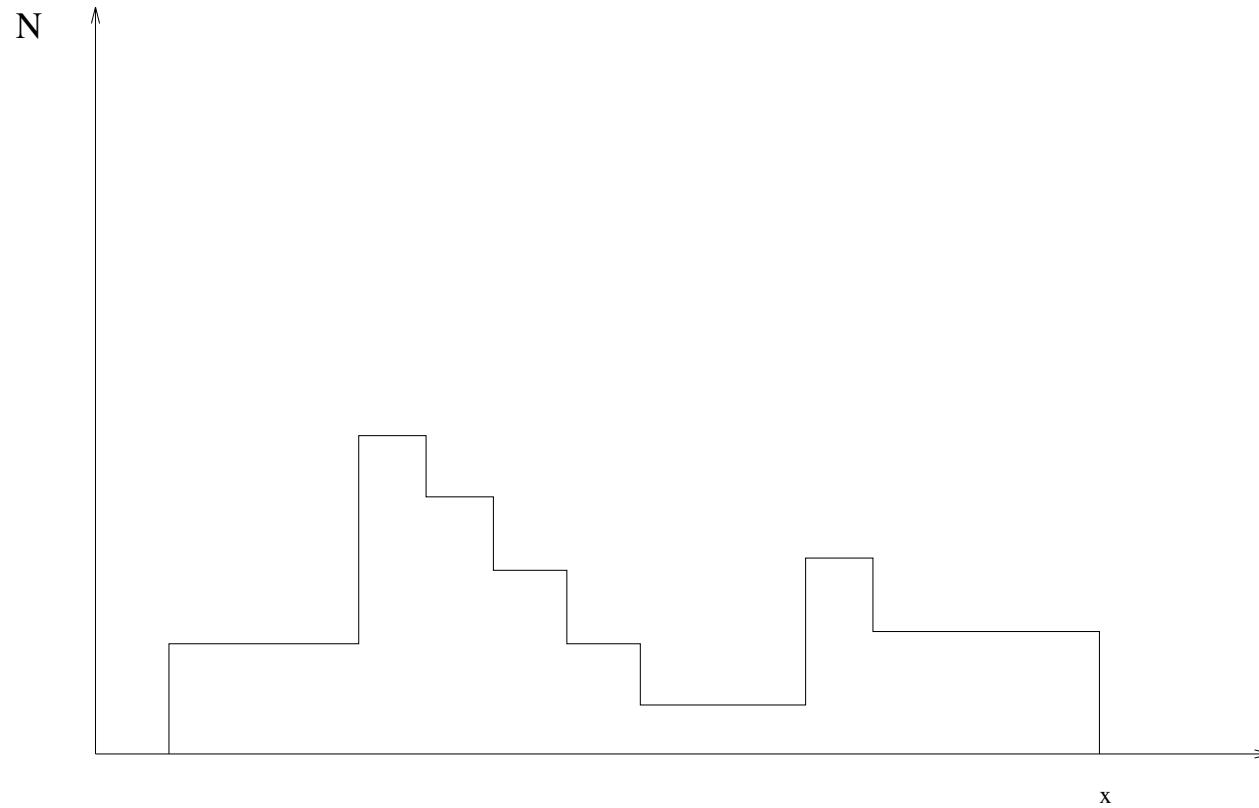


Figure 7: Trajectoire avec retour à 0

Soit  $S$  la surface engendrée par la trajectoire entre 0 et  $x$ .

## Preuve de Little pour une simulation avec retour à 0 - suite

Calcul de  $S$  dans les deux sens (vertical et horizontal)

- $S = \int_{t=0}^x N(t)dt = x \times \bar{N}$
- $S = \sum_{i=1}^m d_i - a_i = m \times \bar{T}$

Donc,  $\bar{N} = \frac{m}{x} \bar{T}$

et par définition  $\lambda = \frac{m}{x}$

## Inégalités de Markov

**Propriété 3 (Inégalité de Markov)** : Soit  $X$  une variable aléatoire sur un espace discret et soit  $f$  une fonction, on a pour toute valeur positive de  $a$

$$Pr(|f(X)| \geq a) \leq \frac{E(|f(X)|)}{a}$$

Usage en simulation :  $E(|f(X)|)$  va être estimée beaucoup plus vite que  $Pr(|f(X)| \geq a)$  si les états où  $f(X)$  est grand sont rares.

## **Génération d'entiers uniformes**

Dispositif Physique ou Algorithme :

Problème de la reproductibilité : pouvoir reproduire la séquence et donc la trajectoire (par exemple pour debugger...)

Problème du Test : pouvoir tester la séquence et ses propriétés statistiques.

**On choisit d'utiliser un algorithme...**

## Paradoxe : construction déterministe de l' aléatoire

Von Neumann (1951) : *Any one who considers arithmetical methods of reproducing random digits is, of course in a state of sin. As been pointed out several times, there is no such thing as a random number- there are only methods of producing random numbers, and a strict arithmetic procedure is not such a method.*

## **Sortir du Paradoxe**

- en simulation : réussir à un test statistique d'uniformité  
Attention, aucune garantie sur les autres propriétés
- en cryptographie : théorie de l'information

## Principe de Base

Utiliser une fonction  $f$  à  $n$  arguments pour générer la prochaine valeur. Les arguments sont les  $n$  valeurs précédentes de la série.

$$x_{n+k} = f(x_{n+k-1}, x_{n+k-2}, \dots, x_k)$$

- Il faut  $n$  valeurs pour initialiser le processus (graines, semences, seed)
- Comportement cyclique potentiel. Si on retombe sur une série de  $n$  valeurs déjà calculées, alors la suite de la série est la même.
- La longueur du cycle dépend de la fonction mais aussi des racines.

Informatique :

les entiers et des réels informatiques sont codés sur un support fini.

Conséquence :

**La séquence est finie, périodique et déterministe**

La génération de nombres uniformes est une des fonctions les plus utilisées dans la dynamique de la simulation.

Problème d'efficacité...

Employer des opérateurs simples et rapides (arithmétique entière, décalage binaire)

Compromis entre la complexité et des propriétés statistiques à vérifier.

## Contraintes sur les séquences générées

- avoir une grande longueur avant de recommencer la séquence (longueur du cycle ou période).
- réussir un test d'uniformité avec le degré de confiance choisi par l'utilisateur.
- réussir un test d'indépendance entre deux valeurs successives avec le degré de confiance choisi par l'utilisateur.
- réussir des tests variés en fonction des besoins spécifiques (distributivité dans l'espace, autocorrélation).

## **3 Principles**

- Ce n'est pas parce-que ca a l'air compliqué que c'est aléatoire....
- Chercher un algorithme récent étudié par un spécialiste...
- Effectuer des tests en cas d'utilisation "spéciale" ...

## Exemple : Générateur Idiot

$$x_{n+1} = \text{abs}(\sin(\ln(x_n)))$$

**Complexité :** fonctions transcendentales ( $\ln$  et  $\sin$ ), beaucoup plus long à calculer que "+"

**Uniformité et Distributivité :** (20 tirages  $x_0 = 2$ )

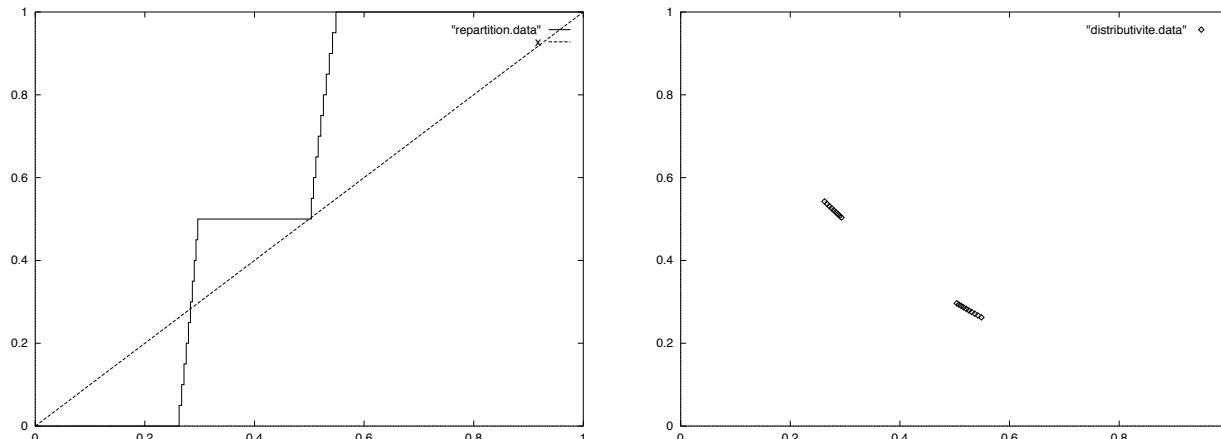


Figure 9: Uniformité et Distributivité

## Générateurs simples

- LCG : linear Congruential Generator
- Générateur à décalage sur la représentation binaire

## Générateur LCG

La fonction  $f$  est linéaire et utilise une arithmétique modulo  $m$ .

$$x_n = ax_{n-1} + b \bmod m$$

où  $a$  et  $b$  sont positifs ou nuls.  $x_n$  est un entier entre  $O$  et  $m - 1$ .  
On se ramène aux réels par division par  $m$ .

$$y_n = x_n/m$$

Donc  $y_n \in [0, 1[$ .

Attention la valeur 1 est exclue.

## Générateur à décalage

$x_n$  est représenté comme un vecteur binaire. L'itération est

$$x_{n+1} = Tx_n$$

$T$  est une matrice booléenne à diagonale constante (équivalente à un polynome)

On peut effectuer l'opération  $Tx_n$  par un circuit.

## Propriétés simples des générateurs LCG

- Période maximale  $m$ .
- Une seule racine.
- Pour faciliter les calculs, souvent  $m = 2^k$ .  
→ division et modulo = simples décalages de bits.
- si  $b$  est nul, on parle de LCG multiplicatif. Calcul plus facile.

Si  $b > 0$ , la période maximale  $m$  est obtenue si et seulement si les trois conditions suivantes sont satisfaites :

- $b$  et  $m$  sont premiers entre eux.
- chaque entier premier diviseur de  $m$  est aussi diviseur de  $a - 1$ .
- si  $m$  est multiple de 4,  $a - 1$  doit être multiple de 4.

Exemple : les générateurs de type  $m = 2^k$ ,  $a = 4c + 1$ , et  $b$  impair ont un cycle de longueur  $m$ .

La longueur du cycle n'est pas tout; la corrélation entre deux valeurs successives est très importante.

Exemples :

$$x_n = (2^{34} + 1)x_{n-1} + 1 \bmod 2^{35}$$

$$x_n = (2^{18} + 1)x_{n-1} + 1 \bmod 2^{35}$$

Le premier a une autocorrélation d'ordre 1 de 0.25 alors que le second a une autocorrélation d'ordre 1 de  $2^{-18}$ .

## LCG multiplicatif

Deux familles intéressantes :

1.  $m = 2^k$
2.  $m$  premier

$$m = 2^k$$

- Opération modulo = décalage en binaire
- période maximale  $m/4$
- période maximale atteinte pour  $a = 8i \pm 3$  pour une racine initiale impaire

$m$  premier

- Période maximale :  $m - 1$
- Impossible d'atteindre 0 (et de quitter cette valeur)
- Pour obtenir la période maximale, il faut choisir  $a$  comme racine primitive de  $m$ .

**Définition 2**  $a$  est une racine primitive de  $m$  si et seulement si  $a^n - 1$  n'est pas divisible par  $m$  pour tout  $n$  strictement inférieur

Exemple, si  $m = 31$ , on peut prendre  $a = 3$  car 3 est une racine primitive de 31.

## Problèmes d'Implémentation des générateurs LCG

1. Il faut toujours travailler sur de l'arithmétique entière exacte et non pas passer en réel et faire une troncature.
2. Vérifier que le type entier existe effectivement dans le langage et qu'il y a une arithmétique exacte associée.
3. il faut que l'opération de multiplication  $a \times x_{n-1}$  ne provoque pas d'erreurs de débordement (dépassemement de l'entier maximum).

Sinon, le codage du générateur ne vérifie pas les propriétés énoncées.

## Méthode de Schrage

Pour éliminer le problème du dépassement de l'entier maximal.

$$a \times x \bmod m = g(x) + m \times h(x)$$

$$q = (m \text{ div } a) \tag{2}$$

$$r = (m \bmod a) \tag{3}$$

$$g(x) = a \times (x \bmod q) - r \times (x \text{ div } q) \tag{4}$$

$$h(x) = (x \text{ div } q) - (a \times x \text{ div } m) \tag{5}$$

Pour tout  $x$  de 0 à  $m - 1$ , les termes apparaissant dans le calcul de  $g(x)$  sont plus petits que  $m$ .

De plus, si  $r < q$ ,  $h(x)$  vaut 1 si  $g(x)$  est négatif et 0 si  $g(x)$  est positif. Il est inutile de calculer  $h$ .

## Problèmes et Conseils

- Ne jamais utiliser la fonction random fournie par un langage non spécialisé sans l'avoir testée.
- Ne jamais utiliser 0 comme racine, si le générateur est un LCG multiplicatif, on reste coincé à 0.
- Ne jamais prendre de racines aléatoires (inconnues) ou dépendantes du système (horloge, pid). 2 Risques : non reproductibilité de la séquence et choix d'une mauvaise racine.

Ne jamais supposer que si un nombre est aléatoire, un bit de sa représentation l'est également. Ne pas se servir de la représentation binaire de  $x_n$  comme une chaîne binaire aléatoire uniforme.

Exemple,  $x_n = (8789x_{n-1} + 16384) \bmod 2^{16}$  initialisé à 1

- le bit 1 (le poids faible) est toujours 1
- le bit 2 est toujours 0
- le bit 3 suit la séquence 01
- le bit 4 suit la séquence 0110
- le bit 5 suit un cycle 11010010
- plus généralement le bit  $l$  suit un cycle de longueur  $2^{l-2}$ .

Cette propriété est toujours vraie pour les générateurs LCG avec  $m = 2^k$ . Les bits de poids faible sont "peu aléatoires".

Exemple d'erreurs provoquées par un générateur de ce type

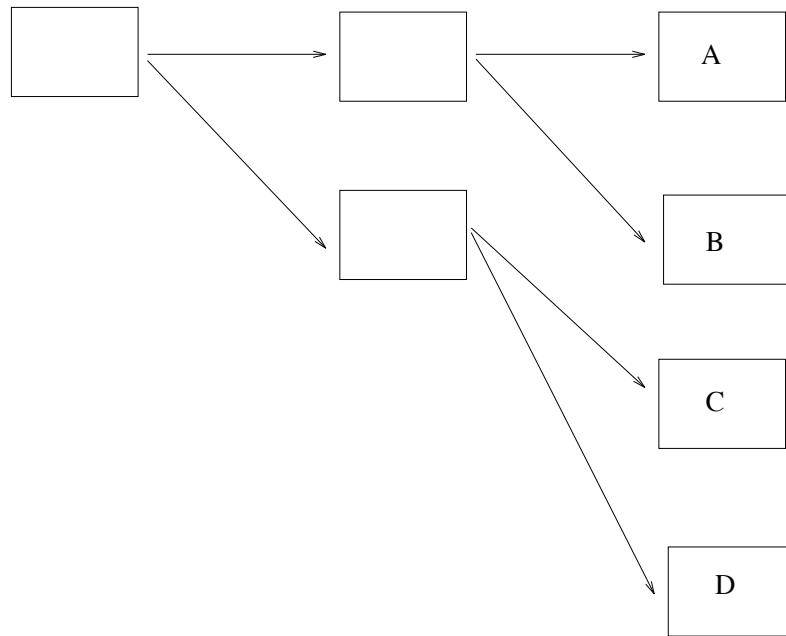


Figure 10: Réseau Multi-étages

Modèle du routage : indépendant, à chaque switch on choisit la destination avec probabilité  $1/2$ .

Conclusion du modèle: Chaque switch du 2ème étage traite  $1/4$  du trafic.

Implémentation à l'aide du générateur précédent et on utilise les bits 1 et 2 des entiers générés pour faire le routage : 0 pour sortie haute, 1 pour sortie basse.

Conclusion de la simulation : le switch  $C$  traite la totalité du trafic.

- Plus généralement, pour éviter les dépendances, éviter que les séquences ne se chevauchent :
  - déterminer la longueur de la simulation en nombre de tirages aléatoires pour chaque source ( $N$ ).
  - évaluer à part les différentes racines :  $x_0, x_N, x_{2N}$ , etc.

$$x_N = f^{(N)}(x_0)$$

$$x_{2N} = f^{(2N)}(x_0)$$

## Problème de la k-distributivité

Illustrons le problème:

- $x_n = (2^{16} + 3)x_{n-1} \text{ mod } 2^{31}$  est un générateur utilisé par IBM dans les années 60, sous le nom de RANDU.
- Il existe une version pour micro 16 bits de ce générateur  $x_n = (2^8 + 3)x_{n-1} \text{ mod } 2^{15}$ . Les deux sont à éviter....
- si on prend 3 points successifs comme coordonnées  $(x, y, z)$ , tous les points ainsi générés se devraient remplir tout le cube.

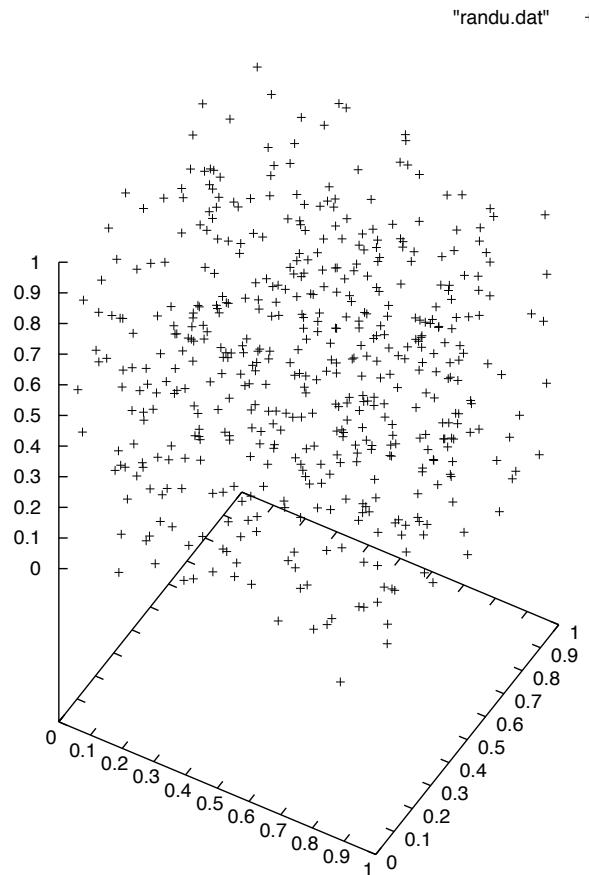


Figure 11: Triplets de points construits par RANDU

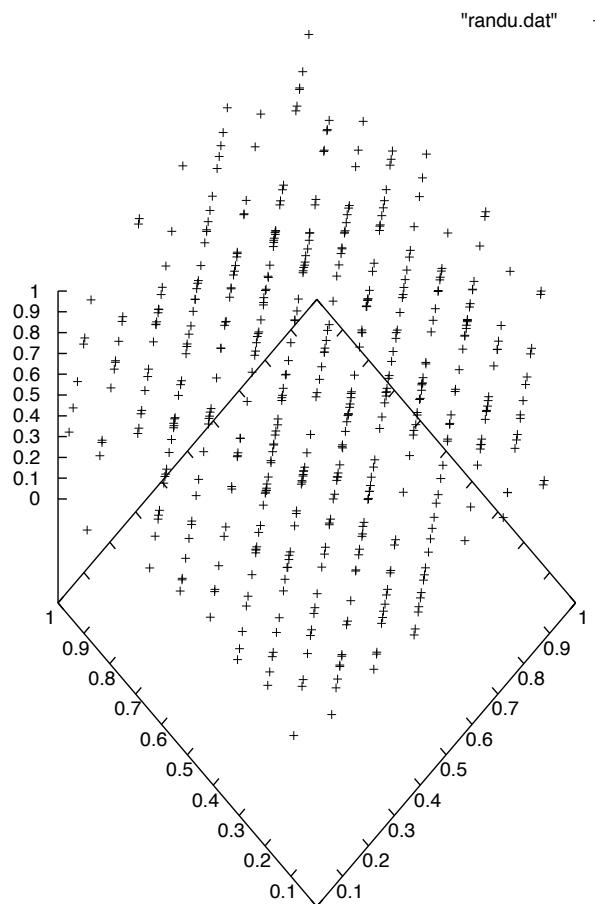


Figure 12: Triplets de points construits par RANDU, changement d'angle de vue

Tous les points ainsi générés se situent sur 15 plans et non pas dans tout le cube.

**Le problème est général à tous les générateurs LCG. Tous les  $k$ -uplets sont dans au plus  $(k!m)^{1/k}$  hyperplans parallèles.**

Illustration en dimension 2 : des droites....

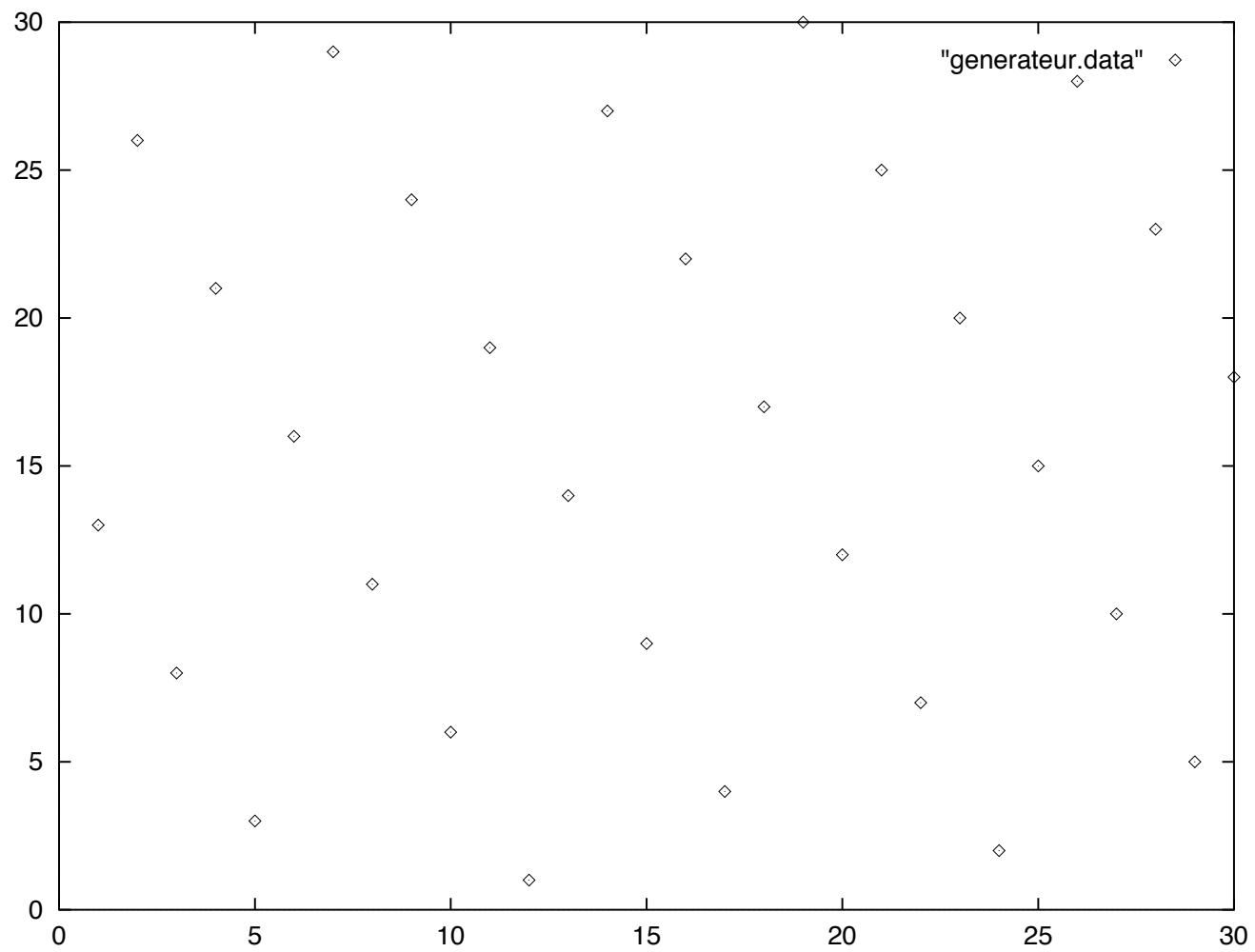


Figure 13: Pavage par un LCG

## Amélioration des LCG par combinaison

Méthode de la Table (Knuth)

On utilise deux générateurs  $x_n$  et  $y_n$ .

1. On initialise un tableau  $T$  de taille  $k$  avec  $k$  valeurs successives du générateur  $x_n$ .
2. On utilise  $y_n$  normalisé entre 1 et  $k$ .  $y_n = i$
3. On retourne  $T(i)$ ,
4. On calcule une nouvelle valeur de  $x_n$  à ranger dans  $T(i)$
5. On recommence en 2

Algorithme Kiss (Keep It Simple, Stupid) [Marsaglia, Zaman 1993]

On utilise un LCG  $I_n$ , et deux générateurs à décalage  $J_n$  et  $K_n$  pour obtenir  $x_n$ .

$$I_{n+1} = 69069I_n + 23606797 \bmod 2^{32}$$

Décalage dans la représentation binaire.

$I$  : identité,  $L$  décalage à gauche,  $R$  décalage à droite.

$$J_{n+1} = (I + L^{15})(I + R^{17})J_n \bmod 2^{32}$$

$$K_{n+1} = (I + R^{13})(I + L^{18})K_n \bmod 2^{31}$$

$$x_{n+1} = (I_{n+1} + J_{n+1} + K_{n+1})/2^{32}$$

Le générateur a une période de  $2^{95}$ .

## Code C de KISS

```
long int kiss (i, j, k)
unsigned long *i, *j, *k;
{
*j=*j^(*j<<17);
*k=(*k ^ (*k<<18) ) & 0x7fffffff;
return ((*i=69069 * (*i) + 23606797)
+ (*j^=(*j>>15))
+ (*k^=(*k>>13)));
}
```

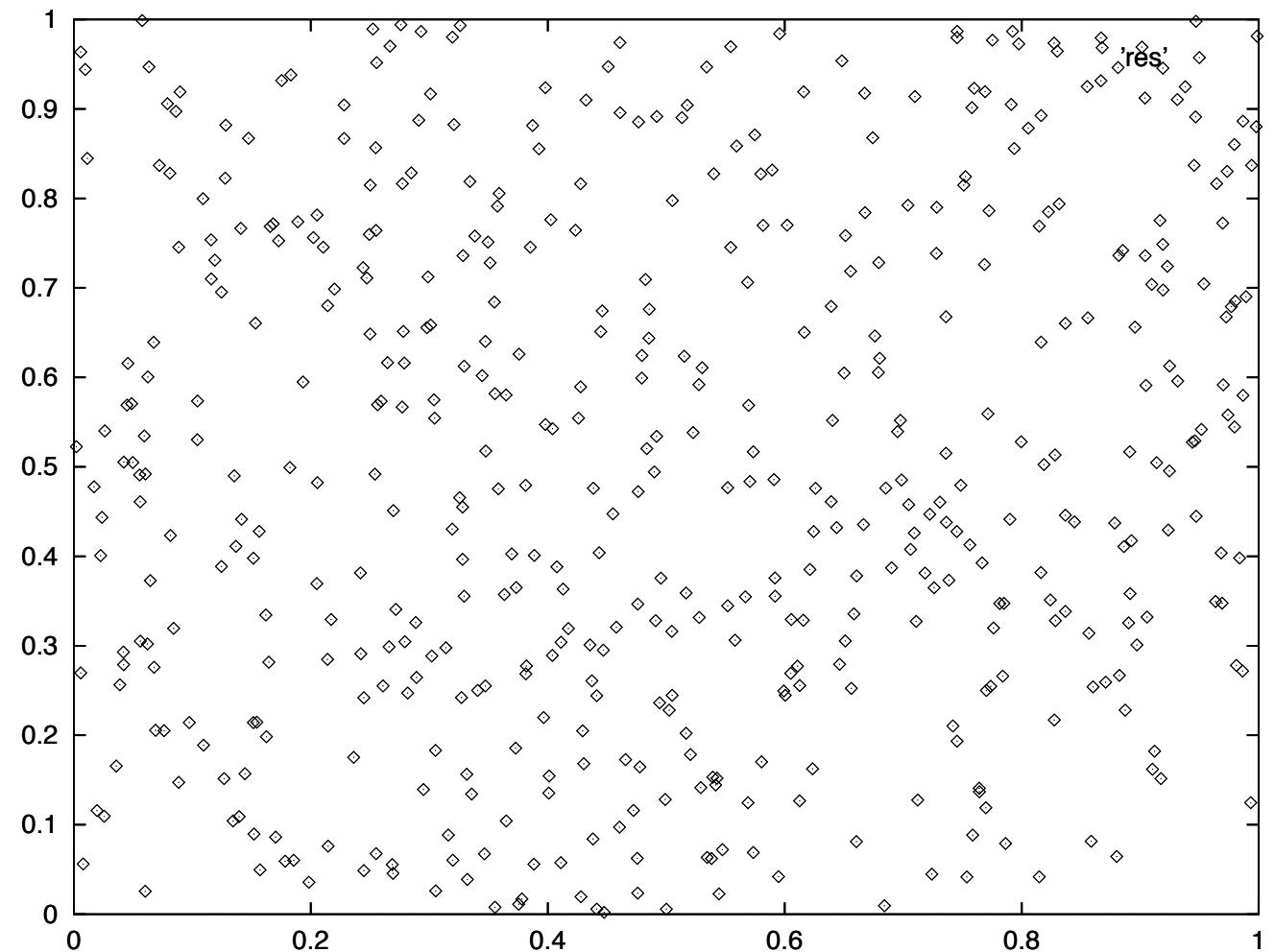


Figure 14: 500 Points générés par KISS

## Tester un générateur uniforme

1. Uniformité : application du test du  $\chi^2$
2. K-distributivité (uniformité dans l'espace), test sériel, test spectral
3. Test de 2-distributivité : simplification du test spectral
4. Test de d'indépendance (Corrélation)

## Test du $\chi^2$

Le test du  $\chi^2$  est un test d'adéquation à une distribution discrète quelconque.

On divise l'espace en  $k$  sous-intervalles, on fait  $N$  tirages qui sont répartis dans les  $k$  sous-intervalles

$$D = \sum_{i=1}^k \frac{(o_i - e_i)^2}{e_i}$$

$o_i$  est le nombre d'occurrences observées dans le  $i$ -ème sous intervalle,  $e_i$  est la quantité théorique d'occurrences compte tenu de la largeur de l'intervalle et de la distribution.

Si  $D$  est faible (idéalement  $D = 0$ ) alors, avec un certain risque, on peut accepter l'hypothèse que les deux distributions sont identiques.

Les statisticiens ont montré que  $D$  a une distribution dite du  $\chi^2$  à  $k-1$  degrés de liberté. Ce qui permet une évaluation du risque (principe du test).

Le test consiste alors

1. à calculer  $D$
2. à choisir un niveau de risque d'erreur  $\alpha : 1, 2, 5, 10 \%$
3. à consulter une table du  $\chi^2$  pour y trouver la valeur de  $\chi^2_{1-\alpha, k-1}$
4. à accepter l'hypothèse si  $D < \chi^2_{1-\alpha, k-1}$  et à rejeter l'hypothèse sinon.

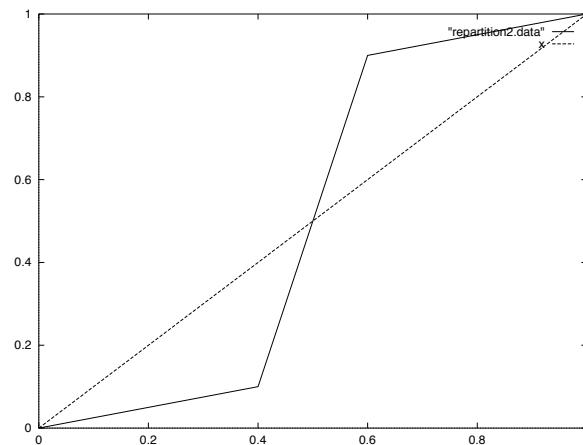
## Application à l'uniformité

Si on divise en intervalles égaux lors de la discréétisation, alors les  $e_i$  sont tous égaux.

$e_i = N/k$  (on se débrouille pour que cela tombe juste).

## Problème de la discréétisation

Une mauvaise discréétisation peut faire conclure à tort à l'uniformité



Avec 2 intervalles ( $0,5$ ), on trouve  $o_1 = 0,5 * N$  et  $o_2 = 0,5 * N$

Avec 5 intervalles de largeur  $0,2$ , on trouve :

$o_1 = 0,05 * N, o_2 = 0,05 * N, o_3 = 0,8 * N, o_4 = 0,05 * N, o_5 = 0,05 * N$

Les conclusions seront différentes dans les deux cas....

## Autres Problèmes

A cause des  $e_i$  au dénominateur, une différence entre les deux distributions a plus de poids lorsque  $e_i$  est faible que lorsque  $e_i$  est fort. On choisit, si possible, de construire des  $e_i$  de même taille.

Le test ne fonctionne que pour des grands échantillons. En pratique on conseille que tous les  $e_i$  soient plus grands que 5.

## Test de $k$ -distributivité

$k$ -distributivité : la distribution spatiale dans un espace à  $k$  dimensions de  $k$  valeurs successives du générateur.

Ces  $k$  valeurs fournissent les coordonnées d'un point dans le cube  $(0, 1)^k$ . Si les points sont répartis uniformément dans le cube, alors le générateur vérifie la propriété de  $k$ -distributivité.

Exemple : 2-distributivité :  $\forall a_1, a_2, b_1, b_2 \in [0, 1]$

$$Pr(a_1 \leq u_{n-1} \leq b_1 \text{ et } a_2 \leq u_n \leq b_2) = (b_1 - a_1) \times (b_2 - a_2)$$

## Test Sériel

Appliquer les tests d'uniformité.

1. diviser le cube  $[0, 1]^k$  en  $N^k$  cellule de taille constante.
2. générer les  $k$ -uplets et les répartir dans les cellules
3. tester l'uniformité par le test du  $\chi^2$ . Le nombre de degrés de liberté est maintenant  $N^k - 1$ .

Dans le cas de 2-distributivité, un dessin donnera une réponse rapide et souvent saisissante...

## Générer selon une distribution quelconque

A partir des générateurs de réels uniformes entre 0 et 1,

5 méthodes complémentaires :

- transformation inverse,
- rejet,
- composition,
- convolution,
- caractérisation statistique ou probabiliste

- Compromis entre la complexité (en temps) et les propriétés statistiques ou numériques.
- Tenir compte des mauvaises propriétés des générateurs uniformes
- Optimiser le temps de calcul (grand nombre d'appels à ces fonctions au cours de la simulation)

Pas de méthodes universellement meilleures...

## Coût approximatif des opérations

Heuristiques : fonction transcendante = 100 instructions simples

appel de random = 50 instructions simples

Eviter les fonctions transcendentales (log, sin, exp) et les appels à Random.

## Transformation Inverse

Idée : soit  $X$  une variable aléatoire et sa fonction de répartition  $F$ , alors  $U = F(X)$  est uniformément distribué entre 0 et 1. Il suffit donc de générer  $X$  grâce à  $F^{-1}(U)$ .

Hypothèses :  $F^{-1}$  est facilement calculée grâce à une formule analytique ou une distribution empirique.

## Exemple Analytique

Analytique :

La distribution exponentielle a pour fonction de répartition

$$F(x) = 1 - \exp(-\lambda x).$$

Inverse :  $x = \frac{-1}{\lambda} \ln(1 - u)$ .

Si  $u$  est uniforme sur  $[0, 1]$  alors  $(1 - u)$  est également uniforme sur  $[0, 1]$

$$x = \frac{-\ln(u)}{\lambda}$$

## Exemple pour une distribution empirique

On mesure les tailles de paquets sur un réseau et on obtient la distribution expérimentale suivante :

{	256 <i>octets</i>	0.4
	512 <i>octets</i>	0.25
	1024 <i>octets</i>	0.35

On calcule la fonction de répartition puis son inverse :

## Fonction de Répartition

$$\begin{cases} 0 \leq x < 256 & F(x) = 0 \\ 256 \leq x < 512 & F(x) = 0.4 \\ 512 \leq x < 1024 & F(x) = 0.65 \\ 1024 \leq x & F(x) = 1.0 \end{cases}$$

## Inverse de la Fonction de Répartition

$$F^{-1}(U) : \begin{cases} 0 \leq u < 0.4 & X = 256 \\ 0.4 \leq u < 0.65 & X = 512 \\ 0.65 \leq u < 1 & X = 1024 \end{cases}$$

## Comment optimiser

- Attention aux fonctions transcendentales.
- Une fonction de répartition d'une distribution discrète est obtenu par un tri. Employer des méthodes de tri rapides.
- Eviter de dupliquer des calculs.
- Tabuler des fonctions souvent calculées.
- Ordonner les tests par probabilité décroissante.

## Mauvaise organisation du test

- $x = \text{random};$
- if ( $x \leq 0.1$ ) return 1;
- else if ( $x \leq 0.15$ ) return 2;
- else if ( $x \leq 0.25$ ) return 3;
- else return 4;

Nombre moyen de test:  $0.1 * 1 + 0.05 * 2 + 0.1 * 3 + 0.75 * 3 = 2.75$

## Bonne organisation du test

- $x = \text{random};$
- if ( $x \leq 0.75$ ) return 4;
- else if ( $x \leq 0.85$ ) return 1;
- else if ( $x \leq 0.95$ ) return 3;
- else return 2;

Nombre moyen de test:  $0.75 * 1 + 0.1 * 2 + 0.1 * 3 + 0.05 * 3 = 1.4$

## Tabuler les calculs

Exemple de la loi de Poisson:

$$Prob(X = n) = e^{-\lambda} \frac{\lambda^n}{n!}$$

Pas d'expression simple pour la fonction de répartition et son inverse.

## Loi de Poisson

$$Prob(X = n) = \frac{\lambda}{n} Prob(X = n - 1)$$

Calcul naïf par une méthode générale d'inversion...

```
P := exp (-lambda); X:=0;  
U:=Random; F:=P;  
while (U > F) do begin  
    X:=X+1;  
    P:=P*lambda/X;  
    F:=F+P;  
end;  
return X;
```

Tabulation de la tête de la distribution qui est souvent calculée.

exemple : tableau de taille 10 pour lambda=1

$i$	0	1	2	3	4
$F_i$	0,36787944	0,73575888	0,91969861	0,98101184	0,99634015
$i$	5	6	7	8	9
$F_i$	0,99940581	0,99991675	0,99998975	0,99999887	0,99999989

Conclusions : dans 11 cas sur 100 millions on doit générer les valeurs suivantes (i.e.  $X > 9$ ).

Algorithme efficace : une recherche sur les 10 valeurs tabulées suivi de l'algorithme général pour les cas rares..

max, maxF, maxP et F ont été initialisés...

```
U:=Random;  
if U <= maxF then begin  
    X:=0;  
    while U>F(X) do X:=X+1;  
end else begin  
    X:=max; F:=maxF; P:=maxP;  
    while (U>F) do begin  
        X:=X+1;  
        P:=P*lambda/X;  
        F:=F+P;  
    end;  
end;  
return X;
```

## Rejet

Hypothèse : il existe une densité  $g(x)$  et une constante  $c$  telle que  $c g(x) \geq f(x)$  pour toute valeur de  $x$ .

La méthode consiste alors à :

1. générer  $x$  de densité  $g()$ ,
2. générer  $y$  uniforme sur  $[0, cg(x)]$ ,
3. si  $y \leq f(x)$  rendre la valeur  $x$ , sinon revenir en 1.

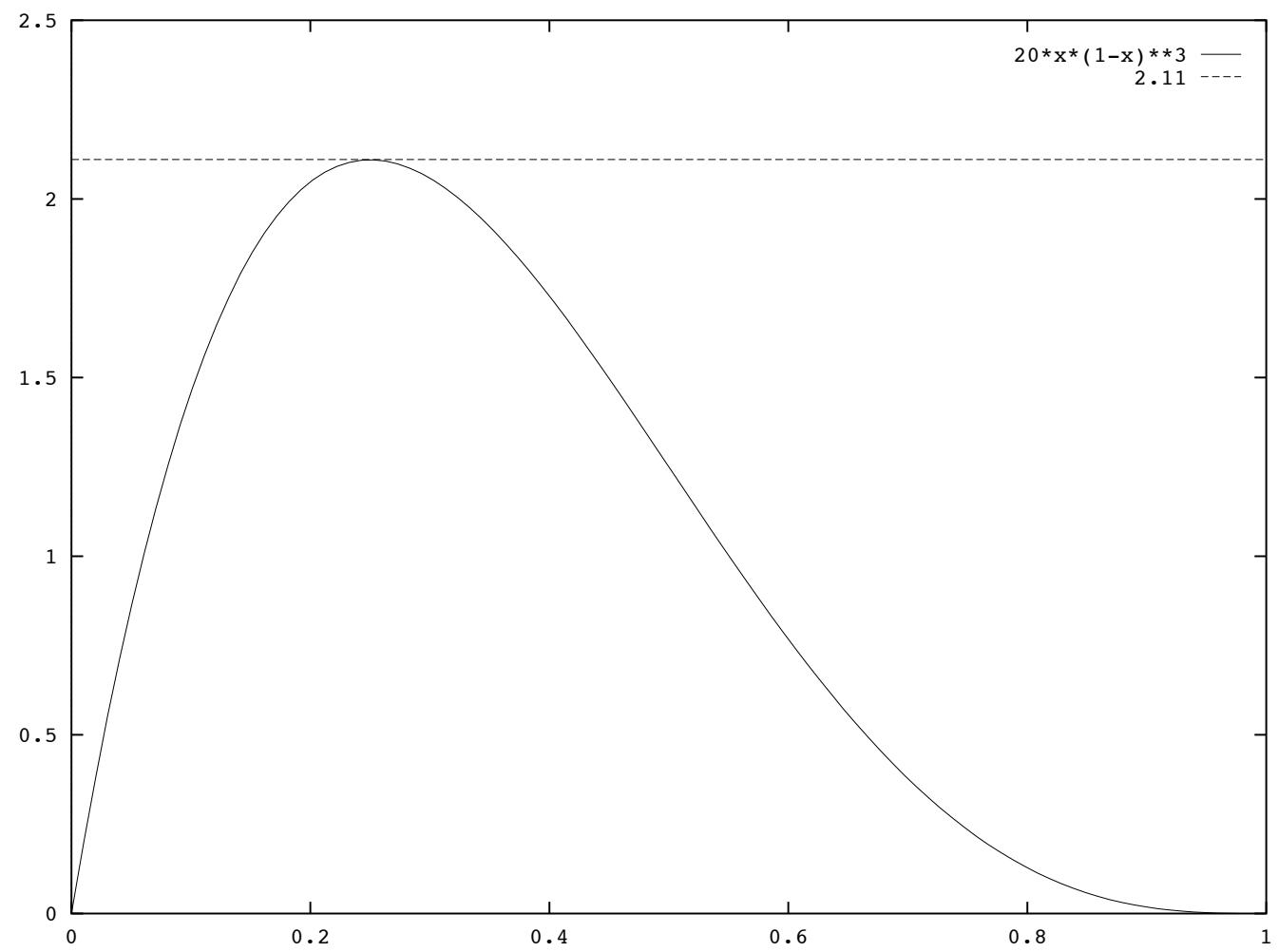


Figure 16: Exemple : méthode du rejet

## Remarques sur la méthode de rejet

- $g(x)$  doit être facile à générer
- le nombre de tentatives avant réussite est distribué selon une géométrique de ratio  $1/c$ .
- $cg(x)$  doit être très proche de  $f(x)$ . Sinon, il faudra de nombreuses tentatives avant que le test soit positif. Et la complexité de la génération sera trop élevée.
- $g(x)$  doit être une densité...  
( $g(x)$  est positive et  $\int_{-\infty}^{+\infty} g(x)dx = 1$ )

Compromis entre simplicité de  $g$  et valeur de  $c$

## Exemple : Méthode du Rejet

Exemple : Générer selon une loi Beta(2,4) dont la densité est  $f(x) = 20x(1 - x)^3$  avec  $x \in [0, 1]$ .

Lorsque  $x$  est compris entre 0 et 1, la fonction  $f(x)$  est plus petite que 2.11.

Choix possible :  $c = 2.11$  et  $g(x) = 1$  ( $g$  : uniforme sur  $[0, 1]$ ).

1. générer  $x$  uniforme sur  $[0, 1]$ ,
2. générer  $y$  uniforme sur  $[0, 2.11]$ ,
3. si  $y \leq 20x(1 - x)^3$  retourner la valeur de  $x$ , sinon revenir en 1.

Autre exemple de rejet :

On dispose d'une pièce parfaite Pile/Face. Comment simuler un dé parfait.

1. On effectue 3 jets de pièces (8 combinaisons).
2. Les combinaisons 1 à 6 sont associées respectivement aux faces 1 à 6.
3. Les combinaisons 7 et 8 sont rejetées. On revient en 1.

## Rejet avec enveloppe

Hypothèses :

- le calcul de  $f$  est complexe
- on connaît une fonction  $h$  beaucoup moins chère à évaluer et vérifiant  $h(x) \leq f(x)$ .

Amélioration en comparant d'abord avec  $h$  (pour éliminer des évaluations de  $f$ ).

1. générer  $x$  de densité  $g()$ ,
2. générer  $y$  uniforme sur  $[0, cg(x)]$ ,
3. si  $y \leq h(x)$  rendre la valeur  $x$  et fin
4. sinon si  $y \leq f(x)$  rendre la valeur  $x$  et fin, sinon revenir en 1.

## Méthode d'Alias

- pour une distribution discrète avec un nombre finie de valeurs
- une variante de la méthode de rejet
- pour éviter de passer dans une longue boucle de test par la méthode de transformation inverse.
- deux appels à la fonction RANDOM + deux vecteurs de la taille de la distribution.

## Exemple Trivial

- On souhaite générer selon la distribution  $(0.125, 0.25, 0.25, 0.375)$ .
- L'algorithme de transformation inverse serait donc :
  - $x=\text{random};$
  - if ( $x \leq 0.125$ ) return 1;
  - else if ( $x \leq 0.275$ ) return 2;
  - else if ( $x \leq 0.625$ ) return 3;
  - else return 4;

## Transformation de l'algorithme

- $y = \text{random}(1,4);$
- $x = \text{random};$
- if ( $y == 1$ ) {if ( $x \leq 0.5$ ) return 1; else return 4;}
- else if ( $y == 2$ ) return 2;
- else if ( $y == 3$ ) return 3;
- else return 4;

On ne gagne rien mais on voit la solution si on prend un peu de mémoire...

## Version optimisée

Initialisation:  $L(1) = 4, F(1) = 0.5, F(2) = 1, F(3) = 1, F(4) = 1$

- $y = \text{random}(1, 4);$
- $x = \text{random};$
- if  $(x \leq F(y))$  return  $y$ ; else return  $L(y);$

2 random, 1 test.

- Soit une distribution  $D$  à générer. Soit  $n$  le nombre de valeurs.
- On suppose que, dans un premier temps, on a construit deux vecteurs à  $n$  valeurs (un vecteur d'entier  $L()$  et un vecteur de réel entre 0 et 1,  $F()$ ) pour la distribution  $D$ .
- L' élément  $i$  des deux vecteurs représente une probabilité de  $1/n$  qui se répartit entre deux cas :  $i$  et  $L(i)$  selon la valeur  $F(i)$ .

## Algorithme de génération selon $D$

- On génère un nombre uniforme entier entre 1 et  $n$  :  $i$
- On génère un nombre uniforme réel entre 0 et 1 :  $u$
- si  $u < F(i)$  alors rendre  $i$  sinon rendre  $L(i)$ .

## Exemple

- On souhaite générer selon la distribution  $(0.1, 0.4, 0.2, 0.3)$ .
- On construit les deux vecteurs  $L = (2, 2, 4, 4)$  et  $F = (0.4, 0.0, 0.8, 0.0)$ .
- Essayons de calculer la probabilité de générer 3

$$Pr(x = 3) = Pr(i = 3)Pr(u \leq 0.8) = 0.25 * 0.8 = 0.2$$

- Et la probabilité de générer 4

$$\begin{aligned} Pr(x = 4) &= Pr(i = 3)Pr(u \geq 0.8) + Pr(i = 4)Pr(u \geq 0) \\ &= 0.25 * 0.2 + 0.25 * 1 \\ &= 0.3 \end{aligned}$$

## Algorithme de création des vecteurs

Algorithme de Walker:

1. Pour tout  $i$ ,  $L(i) = i$ ,  $F(i) = 0$ ,  $b(i) = p(i) - 1/n$
2. Pour tout  $i$  de 1 à  $n$  faire
  - (a) Soit  $c$  le minimum des  $b$  et soit  $k$  l'index du minimum
  - (b) Soit  $d$  le maximum des  $b$  et soit  $m$  l'index du maximum
  - (c) Si  $\sum_i |b(i)| \leq \epsilon$  STOP
  - (d) Sinon  $L(k) = m$ ,  $F(k) = 1 + c \times n$ ,  $b(k) = 0$ ,  $b(m) = c + d$ .

$\epsilon$  est un flottant petit (limite de précision sur les calculs en flottant).

## Exemple

- Reprenons la distribution  $(0.1, 0.4, 0.2, 0.3)$ .
- Initialement  $b$  vaut  $(-0.15, 0.15, -0.05, 0.05)$
- Au premier tour,  $c = -0.15$ ,  $k = 1$  et  $m = 2$ ,  $d = 0.15$ ,
- Donc  $L(1) = 2$   $F(1) = 1 - 0.15 * 4 = 0.4$ ,  $b(1) = 0$ ,  $b(2) = 0$
- Au second tour,  $c = -0.05$ ,  $k = 3$  et  $m = 4$ ,  $d = 0.05$ ,
- Donc  $L(3) = 4$   $F(3) = 1 - 0.05 * 4 = 0.8$ ,  $b(3) = 0$ ,  $b(4) = 0$
- L'algorithme stoppe.

## Ca peut être plus long

- Prenons la distribution  $(0.1, 0.2, 0.44, 0.26)$ .
- Initialement  $b$  vaut  $(-0.15, -0.05, 0.19, 0.01)$
- Au premier tour,  $c = -0.15$ ,  $k = 1$  et  $m = 3$ ,  $d = 0.19$ ,
- Donc  $L(1) = 3$   $F(1) = 1 - 0.15 * 4 = 0.4$ ,  $b(1) = 0$ ,  $b(3) = 0.04$
- Au second tour,  $c = -0.05$ ,  $k = 2$  et  $m = 3$ ,  $d = 0.04$ ,
- Donc  $L(2) = 3$   $F(2) = 1 - 0.05 * 4 = 0.8$ ,  $b(2) = 0$ ,  $b(3) = -0.01$
- Au troisième tour,  $c = -0.01$ ,  $k = 3$  et  $m = 4$ ,  $d = 0.01$ ,
- Donc  $L(3) = 4$   $F(3) = 1 - 0.01 * 4 = 0.96$ ,  $b(3) = 0$ ,  $b(4) = 0$
- L'algorithme stoppe avec  $L = (3, 3, 4, 4)$ , et  
 $F = (0.4, 0.8, 0.96, 0.0)$ .

## Vérification

- $L = (3, 3, 4, 4)$ , et  $F = (0.4, 0.8, 0.96, 0.0)$ .
- Par exemple pour la probabilté de générer 3 :

$$\begin{aligned} Pr(x = 3) &= Pr(i = 1)Pr(u \geq 0.4) + Pr(i = 2)Pr(u \geq 0.8) \\ &\quad + Pr(i = 3)Pr(u \leq 0.96) \\ &= 0.25 * (0.6 + 0.2 + 0.96) \\ &= 0.44 \end{aligned}$$

## Composition

Hypothèse : la fonction de répartition est la somme pondérée d'autres fonctions de répartition :

$$F(x) = \sum_{i=1}^n p_i F_i(x)$$

avec  $p_i \geq 0$  pour tout  $i$ , et  $\sum_{i=1}^n p_i = 1$ .

La méthode consiste à

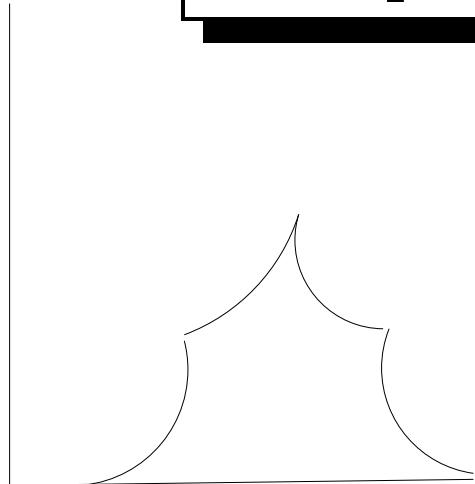
1. générer un entier  $e$  par transformation inverse de la distribution  $p_i$ ,
2. générer  $x$  selon la fonction de répartition  $F_e$ ,

Composition : un moyen général d'amélioration pour une densité ou une fonction de répartition coûteuse

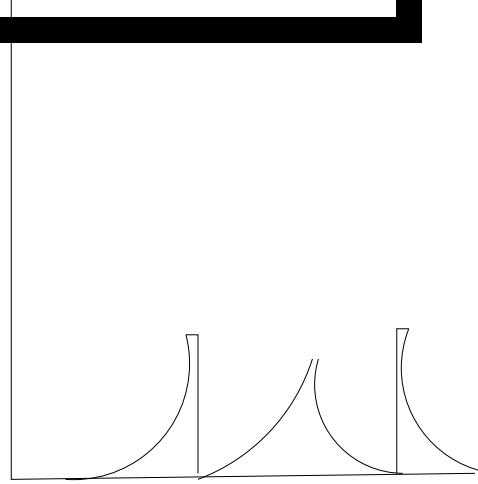
Décomposition artificielle avec des fonctions simples et 1 fonction coûteuse (i.e. : le reste).

Avec de nombreux appels, on gagne du temps chaque fois que l'on passe par les fonctions simples.

## Exemple pour une densité

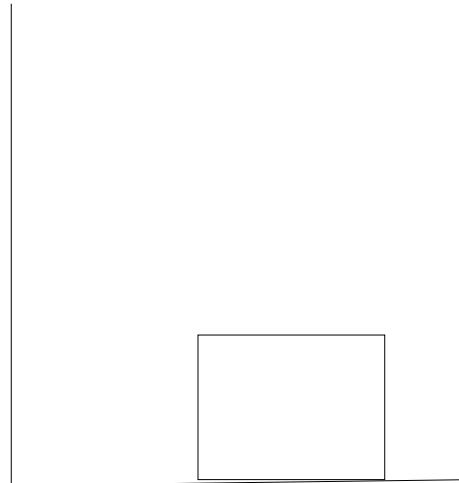


densite difficile



1 densite difficile mais peu utilisee

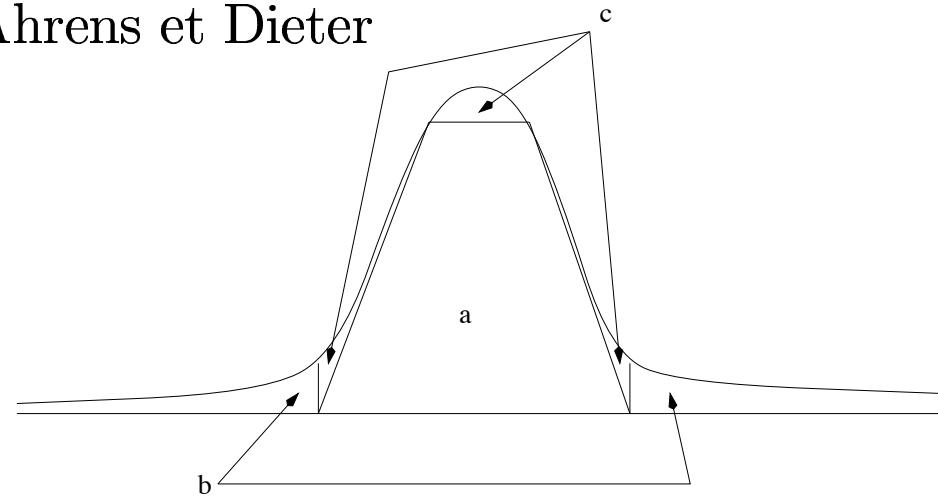
DECOMPOSITION EN :



1 densite simple (uniforme)

## Composition pour une Normale

Méthode de Ahrens et Dieter



$a = 0.9195$  (trapèze, à base d'uniformes),  $b = 0.0345$  (tail : rejet),  
 $c = 0.0460$  (reste : rejet)

## Méthode opérative

Hypothèse : lorsque la variable aléatoire à générer est obtenu à partir de  $n$  autres variables aléatoires plus faciles à obtenir et d'opérateurs arithmétiques simples.

Aussi appelé méthode de Convolution lorsque l'opérateur est une somme.

$$x = y_1 + y_2 + \dots + y_n$$

$F_x$  est la convolution des fonctions de répartition de  $y_i$ .

Méthode : On génère les  $n$  variables aléatoires  $y_i$  et on fait les opérations.

## Exemples

- Une Erlang  $k$  est la somme de  $k$  exponentielles de même paramètre.
- Une Binomiale  $B(n, p)$  est la somme de  $n$  Bernouli de taux  $p$
- Un  $\chi^2$  à  $n$  degré de liberté est la somme des carrés de  $n$  Normales centrées réduites
- La  $a$ -ième plus petite valeur parmi  $(a + b + 1)$  tirages uniformes sur  $[0, 1]$  suit une loi beta( $a, b$ ). ( $a, b$ ) entiers...
- le ratio de deux  $\chi^2$  (resp. de degré de liberté  $n$  et  $m$ ) suit une distribution  $F(n, m)$ .

## Caractérisation Statistique

Hypothèse : propriété statistique de convergence ou de décomposition.

Exemples :

- La somme d'un grand nombre de variables aléatoires indépendantes tend vers une loi normale
- Méthode de Box et Muller pour générer une loi Normale.
  1. générer  $X$  et  $Y$  uniformes et indépendants
  2.  $U = \mu + \sigma \cos(2\pi X) \sqrt{-2 \ln(Y)}$  et  
 $V = \mu + \sigma \sin(2\pi X) \sqrt{-2 \ln(Y)}$  suivent une loi Normale  $\mathcal{N}(\mu, \sigma)$  et sont indépendants.

Ne jamais employer ces deux méthodes...

- Convergence pour la première
- avec un LCG pour la seconde (transformation du système de droites)

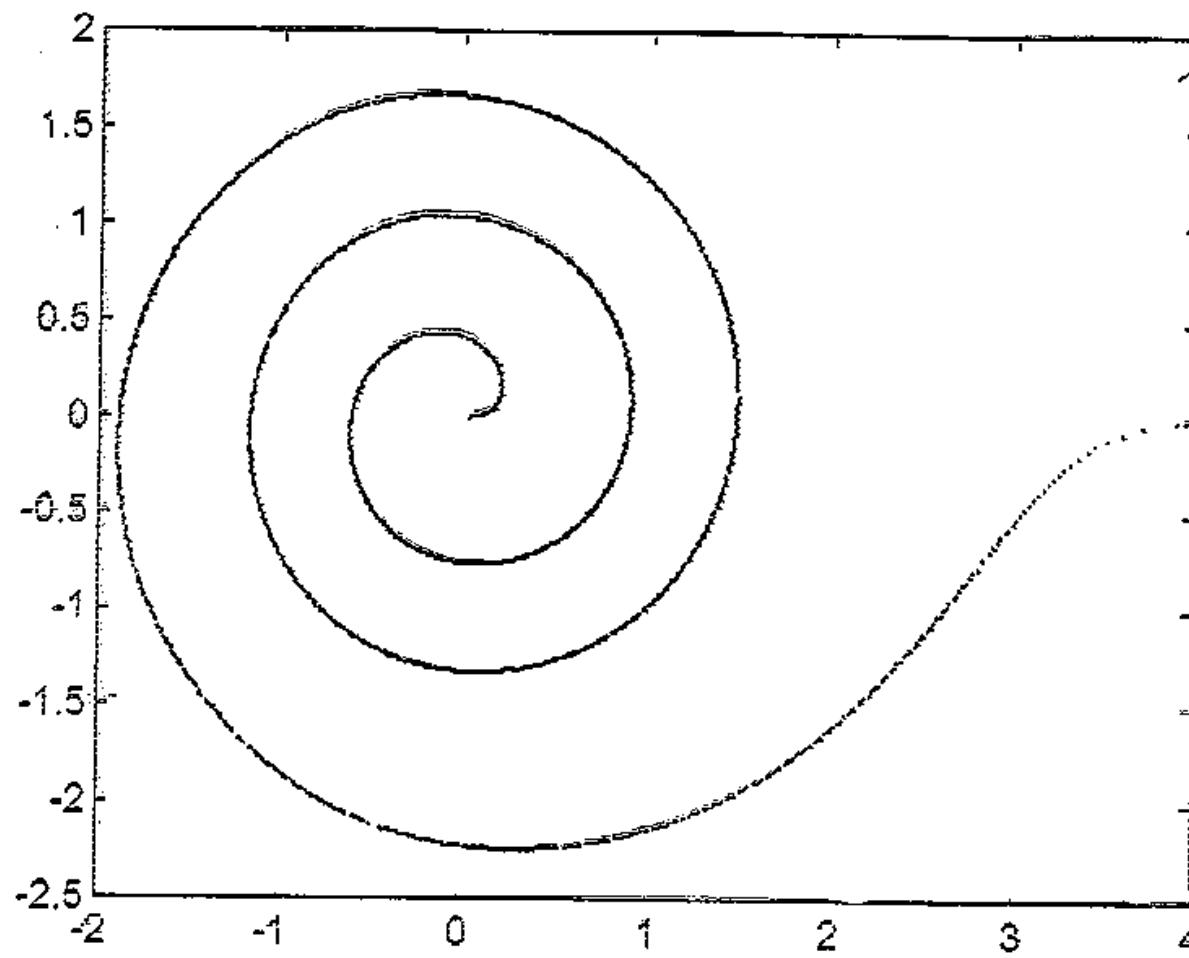


Figure 18: Box Muller et LCG: autre exemple

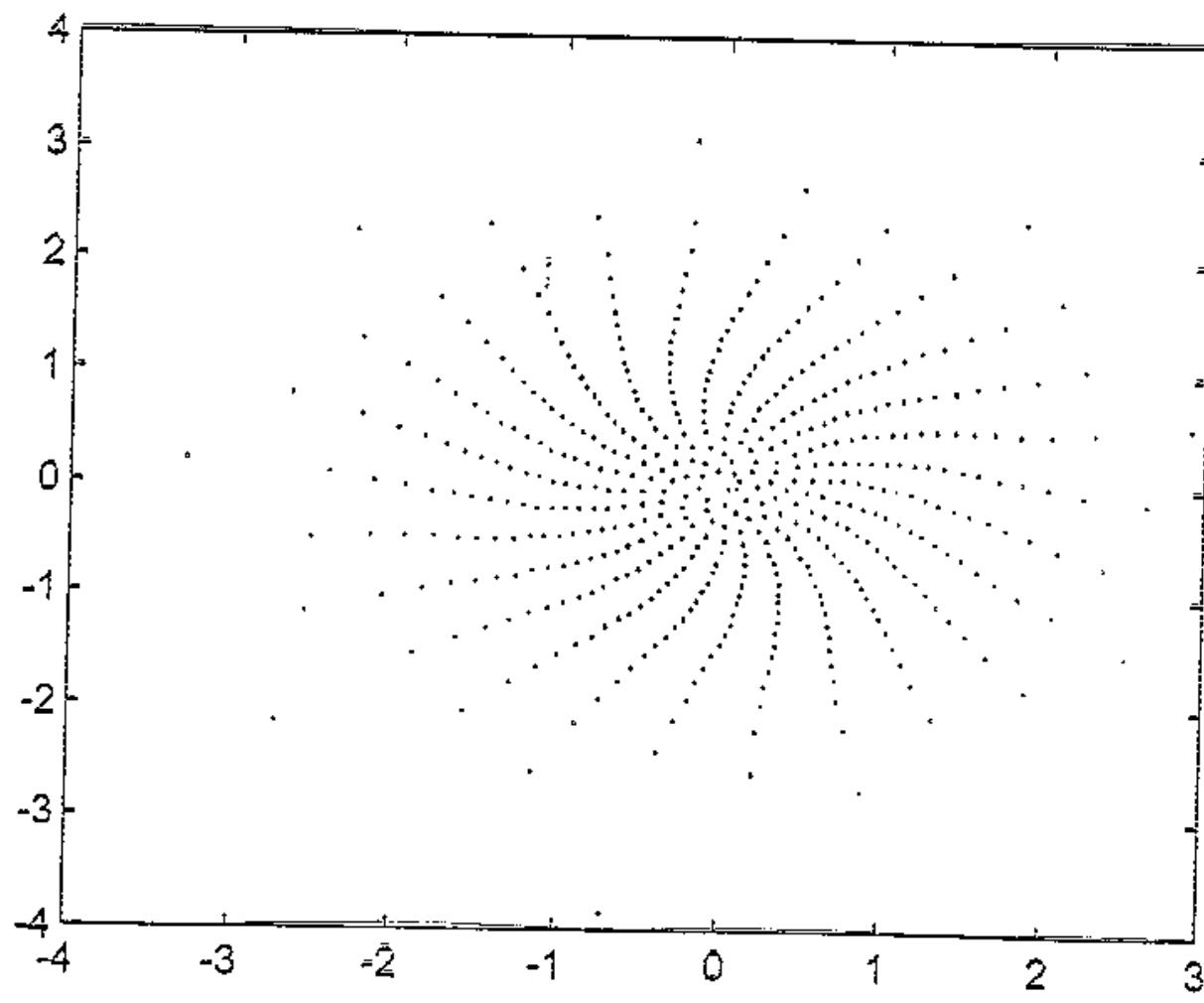


Figure 19: Box Muller et LCG: dernier exemple

# Pourquoi certaines distributions ou processus sont ils plus utilisés ?

1. Normale
2. Lognormale
3. Weibull
4. Gumbel
5. Exponentielle, Poisson
6. Bernoulli, Géométrique, Binomiale

Raisons : stabilité pour certaines opérations, propriété sans mémoire, asymptote.

## Normale

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad \text{avec} \quad -\infty \leq x \leq +\infty.$$

$\mu$  est la moyenne et  $\sigma$  l'écart type ( $\sigma \geq 0$ ).

A cause du théorème central Limite.

**Théorème 2** *La moyenne de variables aléatoires indépendantes et de variance finie est asymptotiquement distribuée selon une loi Normale.*

**Remarque 1** *Il n'est pas nécessaire que les variables aient la même distribution.*

Loi limite sur les opérations Somme

## Exponentielle

Distribution continue :

$$f(x) = \lambda e^{-\lambda x}, \quad \text{avec } 0 \leq x \leq \infty.$$

Propriété “Sans-Mémoire”.

$$P(X < t_0 + t | t_0 < X) = P(X < t)$$

L'état futur ne dépend que du présent et pas du passé.

La propriété “sans mémoire” simplifie la modélisation, l'analyse et la simulation. Le nombre d'état du système est réduit.

## Processus de Poisson

- dates réelles telles que la différence entre deux dates successives est exponentielle.
- Processus d'arrivée simple associée à la loi sans mémoire.
- résultats assymptotiques (arrivées depuis un grand nombre de sources indépendantes)

## Distribution de Poisson

Distribution discrète :

$$f(x) = e^{-\theta} \frac{\theta^x}{x!}, \quad x = 0, 1, \dots$$

Nombre d'arrivée dans l'intervalle  $[0, \theta]$  pour un processus de Poisson d'intensité 1.

## Bernoulli

Distribution discrète avec  $0 \leq p \leq 1$  :

$$f(x) = \begin{cases} 1 - p & \text{si } x = 0 \\ p & \text{si } x = 1 \\ 0 & \text{sinon} \end{cases}$$

La distribution la plus simple (pile ou face).

Routage dans les réseaux.

## Binomiale

Distribution discrète avec  $0 \leq p \leq 1$  et  $n = 1, 2, \dots$  :

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x} \quad \text{avec } x = 0, 1, \dots, n$$

Le nombre de réussites dans une suite de  $n$  tirages de Bernoulli indépendants.

1. le nombre de processeurs en marche dans une machine multi-processeurs.
2. le nombre de paquets qui ont atteint leur destination.

## Géométrique

Distribution discrète :

$$f(x) = (1 - p)^{x-1} p, \quad x = 0, 1, \dots$$

Propriété “Sans-Mémoire”, équivalent discret de l'exponentielle qui est continue.

Modélise un nombre d'essais avant le premier succès ou le premier échec :

Exemple : Nombre de paquets transmis avec succès entre deux paquets transmis avec erreurs,

## Ecrire un Simulateur dans un langage usuel

1. Ecrire un générateur Uniforme
2. Ecrire les algorithmes de transformation pour distributions "réalistes"
3. Construction des Trajectoires
  - Simulation Equationnelle : Itérer l'équation d'évolution
  - Simulation Evénementielle : Construire un échéancier temporel
4. Manipulation des Trajectoires : transitoires, calcul d'intervalles de confiance.
5. Calculs statistiques.

## Simulation Equationnelle

- Trouver l'équation d'évolution  $X_{t+1} = f(X_t)$ .
- Choisir  $X_0$
- Itérer sur l'équation

Avantage : Rapide, Efficace...

Inconvénient : Trouver l'équation  $f$  (en général très difficile).

Conclusions : chercher un peu quand même...

## Exemples

Exemple 1 :Nombre de cellules dans un buffer ATM, temps discret, arrivées de groupes de cellules selon la séquence  $A_t$ , buffer de taille  $B$ , services déterministes de durée 1. Départ avant les arrivées.

$$X_{t+1} = \min(B, \max(0, X_t - 1) + A_t)$$

Exemple 2 :Temps de séjour dans une file FIFO, séquence de dates d'arrivée  $D_t$ , séquence de durées de service  $S_t$ .

$$W_{t+1} = S_{t+1} + \max(0, D_t + W_t - D_{t+1})$$

## Simulation à Evénements Discrets

1. On a  $n$  objets.
2. Chaque objet possède une liste de transitions dans le futur : un échéancier.
3. L'événement le plus proche se réalise car il est le seul à pouvoir le faire.
4. Cette transition peut modifier la liste des transitions futures de tous les objets.
5. On fait les mises à jour des futurs potentiels et des variables d'état (construction de la trajectoire).
6. On revient en 3

## Echéancier global

- Une structure de données associées à un événement et contenant
  - la date
  - le lieu
  - le type de l'événement
  - informations liées au problème
  - informations pour gérer la structure
- Plusieurs structures de données possibles :
  - dynamiques : Heap, Liste simple ou double, Arbre
  - statiques : tableau

## Action potentielles sur l'échéancier global

1. Chercher et Oter la plus petite date (toujours)
2. Insérer un événement dans le futur (toujours)
3. Détruire un élément quelconque (pas toujours)
4. Détruire tous les éléments liés à un objet (pas toujours...)
5. Modifier les dates (pour certaines disciplines)

Attention à l'efficacité réelle : gérer l'échéancier est une des opérations les plus fréquentes.

## Liste Ordonnée

- Triée par date croissante.
- Deux pointeurs pour faire un chainage double.
- Un pointeur supplémentaire pour chainer entre eux les événements ayant lieu au même endroit
- $N$ : nombre total d'événements,  $K_i$ : nombre d'événements en  $i$
- Insertion en  $O(N)$
- Recherche du plus petit en  $O(1)$
- Détruire un événement quelconque spécifié par une date :  $O(N)$
- Détruire un événement quelconque spécifié par un lieu  $i$  :  $O(K_i)$
- Détruire tous les événements en un lieu  $i$ :  $O(K_i)$

## Heap ou Tas

- Un cas particulier d'arbre binaire éventuellement stocké dans un tableau.
- Chaque événement est un noeud de l'arbre de degré au plus deux.
- Les fils d'un noeud sont des événements de dates supérieures à la date de l'événement porté par le père.
- Insertion et Suppression doivent garantir cette propriété.
- L'élément le plus petit est le premier.

- Si la taille de l'échéancier est borné  $T$ , il est plus simple d'implémenter un tas dans un tableau
- Le plus petit élément est le premier
- Ses deux fils ont pour indice 2 et 3
- Les fils du noeud  $i$  ont pour indice  $2i$  et  $2i + 1$ .
- le père du noeud  $i$  a pour indice  $\lfloor \frac{i}{2} \rfloor$ .
- pas de pointeur pour gérer le tas
- le tableau est plein de 1 à  $N$ , vide de  $N + 1$  à  $T$ .

## Opérations

- Insertion en  $O(\log(N))$
- Recherche du plus petit en  $O(\log(N))$  (à cause de la mise à jour)
- Détruire un élément quelconque:  $O(N)$  pour le trouver,  
 $O(\log(N))$  pour mettre à jour.

## Insertion

1. On met le nouvel événement à la place  $N + 1$
2. on compare sa date à la date portée par son père
3. si elle est plus petite, c'est terminé
4. sinon on échange la place du père et du fils et on recommence en 2)

## Recherche du plus petit

1. On prend le premier élément du tableau
2. On met le dernier à la première place
3. On compare sa date à la plus petite date de ses deux fils
4. si elle est plus grande, on échange sa place avec celle du plus petit des deux fils et on recommence en 3)
5. sinon c'est terminé

## Algorithme Général

- 1** etat:=etat initial; date:=0;
- 2** dt:=CalculeDelai();
- 3** AddEvent(date+dt, FirstEvent)
- 4** while  $date < t_{max}$  do begin
  - 4.1** GetEvent(ev);
  - 4.2** olddate:=date; date:=ev.date;
  - 4.3** PerformEvent(ev);
- 5** end;

## Exemple 1 : simuler une File

- File d'attente, capacité infinie
- Arrivées exponentielles (moyenne  $\mu$ )
- Services Gaussiens (moyenne  $\lambda$ , variance  $\sigma^2$ )
- Observations : nombre de clients (distribution et moyenne)
- Critère de fin : temps = tmax;
- Intervalle de confiance sur la moyenne par duplication (à ajouter)

## Pseudo Code

```
Event = record
        date : real;
        tip : (Arrivées, Service);
end
```

Temps est un vecteur de réel

DelExp() et DelGeneral() : fonctions génération de délais aléatoires

AddEvent(date,tip) : ajouter un événement

GetEvent(event) : trouver l'événement le plus proche

tmax est le temps de simulation

- 1** Temps:=0; etat:=0; t:=0;
- 2** dt:=DelExp( $\mu$ );
- 3** AddEvent(t+dt, Arrivées)
- 4** while  $t < tmax$  do begin
  - 4.1** GetEvent(e); ot:=t; t:=e.date;
  - 4.2** Temps[etat]:=Temps[etat]+t-ot;

## **4.3** case e.type of

### **4.3.1** Service : begin

etat:=etat-1;

if etat> 0 then

AddEvent(t+DelGeneral( $\lambda$ ,  $\sigma^2$ ), Service);

end;

### **4.3.2** Arrivées : begin

etat:=etat+1;

AddEvent(t+DelExp( $\mu$ ), Arrivées);

if etat=1 then

AddEvent(t+ DelGeneral( $\lambda$ ,  $\sigma^2$ ), Service);

end;

## **4.4** end;

## **5** end;

- Une analyse de la dynamique montre qu'il y a un ou deux événements dans l'échéancier au cours de la simulation
- Inutile de faire une structure complexe dans ce cas.
- Par contre, si on augmente le nombre de files, la taille de la structure augmente (linéairement dans cet exemple).

## Exemple 2 : un réseau de deux files complexes

- Deux files d'attente, capacité infinie
- Arrivées exponentielles de l'extérieur (moyenne  $\lambda$ ) dans la première file
- Services exponentiels (moyenne  $\mu$ ) dans la file 1 et la file 2
- routage déterministe de la file 1 vers la file 2
- routage dépendant de la file 2 vers la file 1: uniquement si la file 1 est vide, sinon le client quitte le réseau
- Discipline Quantum (de pas  $qntm$ ) pour la file 2, chaque client reçoit  $qntm$  u.t. de service puis passe la main au suivant dans une liste ordonnée. A l'arrivée on rejoint la fin de la liste.
- Discipline LIFO (Last In First Out) avec conservation du service déjà acquis pour la file 1

- Critère de fin : temps = tmax;
- tmax est le temps de simulation
- Observations : nombre de clients (distribution et moyenne)

## Pseudo Code

```
tipEvent = (Arrivees, DebutService, RepriseService, FinService)
```

```
Event = record
```

```
    date : real;
```

```
    lieu : (File1,File2);
```

```
    tip : tipEvent;
```

```
    demande : real;
```

```
end
```

Temps est un tableau à deux dimensions de réel

DelExp() : fonctions génération de délais aléatoires

AddEvent(Lieu,date,tipEvent,demande) : ajouter un événement

GetEvent(event) : trouver l'événement le plus proche

LookForEvent(Lieu,tipEvent) : retourne le premier événement ayant les caractéristiques Lieu et tipEvent.

DelayAllEvent(Lieu,tipEvent,délai) : retarde de délai tous les événements correspondants aux caractéristiques Lieu et tipEvent  
etat : vecteur de deux entiers

TFinal : une date: le premier instant disponible pour le service dans la discipline RR-Quantum.

- 1** Temps:=0; etat[1]:=0; etat[2]:=0;t:=0; TFinal=0;
- 2** dt:=DelExp( $\mu$ );
- 3** AddEvent(t+dt, 1, Arrivées)
- 4** while  $t < tmax$  do begin
  - 4.1** GetEvent(e); ot:=t; t:=e.date;
  - 4.2** Temps[etat]:=Temps[etat]+t-ot;

## **4.3 case e.lieu of**

### **4.3.1 File1 : case e.tip of**

#### **4.3.1.1 Arrivées : begin**

```
AddEvent(File1,t+DelExp(λ),Arrivées,0);  
AddEvent(File1,t ,DébutService,0);  
etat(1)++;  
end;
```

#### **4.3.1.2 RepriseService : AddEvent(1,t+ev.demande,FinService,0);**

#### **4.3.1.3 FinService : begin**

```
etat(1)- -;  
AddEvent(File2,t,Arrivées,0);  
end;
```

#### **4.3.1.4 DebutService : begin**

```
ServiceDemandé=DelExp( $\mu$ );
if (etat(1)> 1) then begin
    .
    DelayAllEvent(File1,RepriseService,ServiceDemandé)
    .
    ev2=LookForEvent(File1,FinService);
    .
    AddEvent(File1,t+ServiceDemandé,RepriseService,
    .
    ev2.date-t);
end;
AddEvent(File1,t+ServiceDemandé,FinService,0);
end;
```

#### **4.3.2 File2 : case e.tip of**

##### **4.3.2.1 Arrivées : begin**

```
AddEvent(File2,t ,DébutService,0); etat(1)++;  
end;
```

##### **4.3.2.2 RepriseService : begin**

```
ServiceAccordé=Min(ev.demande,qntm);  
if ServiceAccordé=ev.demande then  
.     AddEvent(File2,TFinal,FinService,0)  
.     else AddEvent(File2,TFinal,RepriseService,  
.                   ev.demande-ServiceAccordé)  
end;
```

##### **4.3.2.3 FinService : begin**

```
etat(1)- -; if etat(1)=0 then AddEvent(File1,t,Arrivées,0);  
end;
```

#### **4.3.2.4 DebutService : begin**

ServiceDemandé=DelExp( $\mu$ );

ServiceAccordé=Min(ServiceDemandé,qntm);

if ServiceAccorde=ServiceDemandé then

.        AddEvent(File2,TFinal,FinService,0)

.        else AddEvent(File2,TFinal,RepriseService,

.              ServiceDemandé-ServiceAccordé)

end;

**4.4 end;**

**5 end;**

## Que faire des trajectoires

- Détruire la dépendance à l'état initial
- Calcul des intervalles de confiance
- Construction des expériences

Les étapes 1 et 3 doivent être traitées par des procédures compatibles

## Oter la dépendance à l'état initial

On dit aussi oter la partie transitoire ou extraire le comportement stationnaire

Problème méthodologique : estimateur d'une trajectoire infinie en utilisant un horizon fini

Solution : Oter la partie dépendante de l'état initial

Ne pas oublier de vérifier la stabilité

## Heuristiques

- Pour tester la stabilité, changer le temps de simulation et observer la variabilité des estimateurs)
- un phénomène résultant d'accumulation a moins de variabilité que les phénomènes isolés qui le composent.

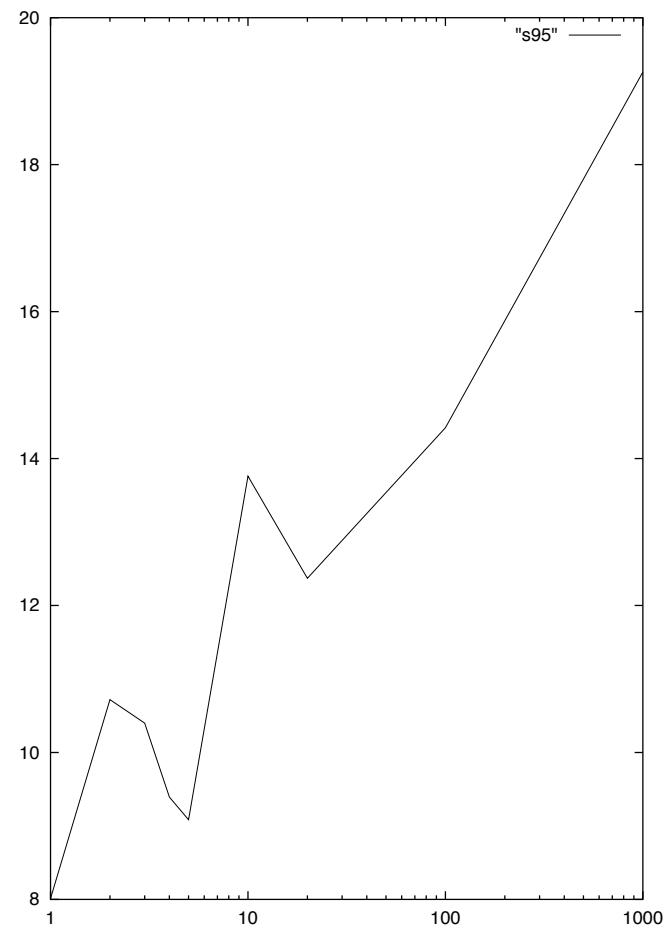


Figure 20: M/M/1; Nombre moyen de clients vs temps de simulation (logscale); charge=0,95; environ 1 million de clients

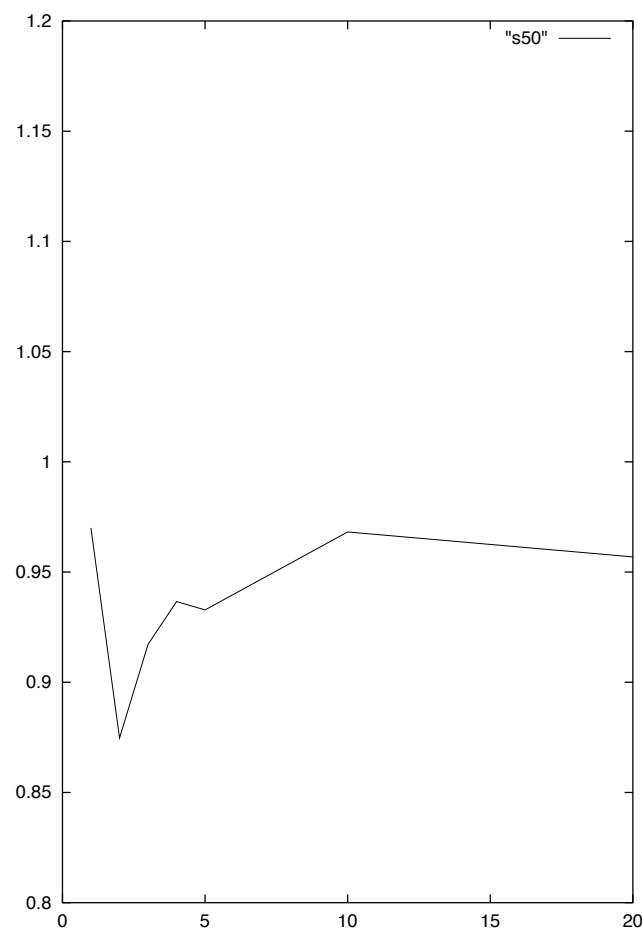


Figure 21: M/M/1; Nombre moyen de clients vs temps de simulation;  
charge=0,5; environ 500000 clients

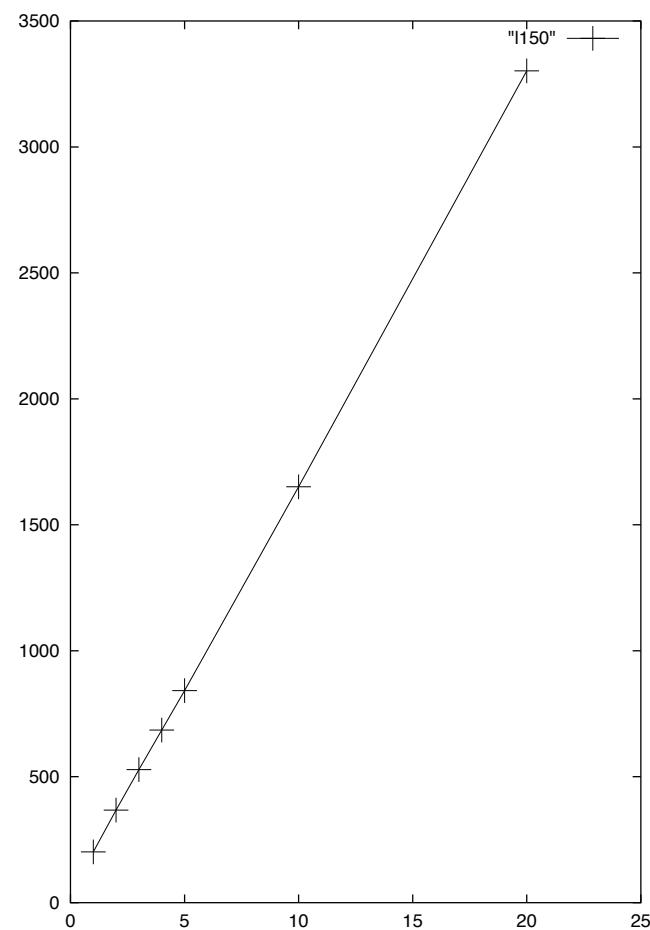


Figure 22: M/M/1; Nombre moyen de clients vs temps de simulation;  
charge=1.5; date en milliers d'ut

## Intervalles de Confiance

Problème de la Représentativité d'une expérience.

Une seule expérience : indicateur peu fiable.

Exemple : on a 1000 boules rouges et 10 boules noires dans une urne.

On fait un tirage.

On trouve une noire.

Conclusion : la probabilité de trouver une boule noire est estimée à 1.

La conclusion serait différente si on faisait plusieurs expériences

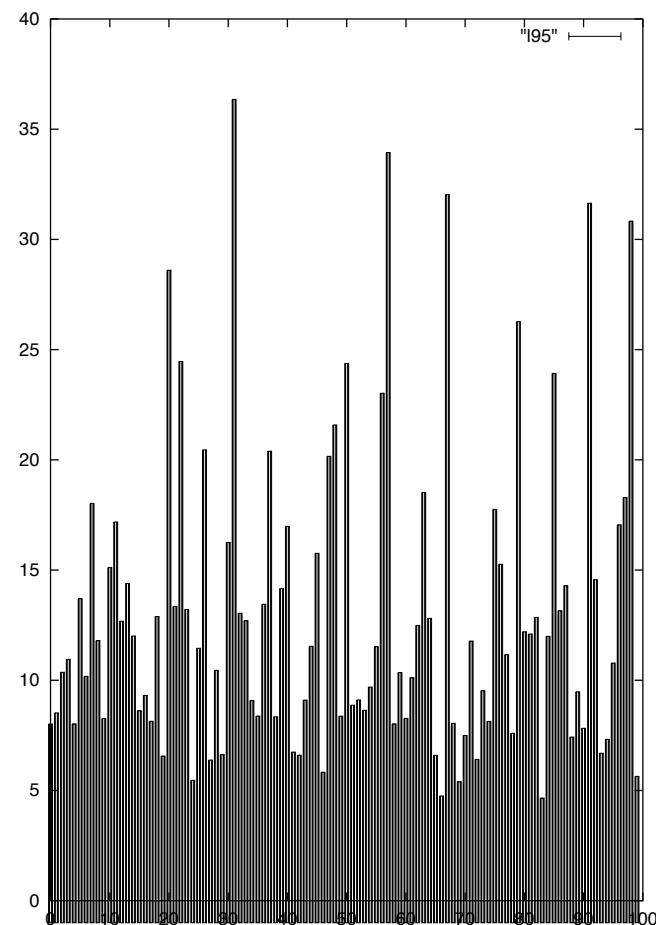
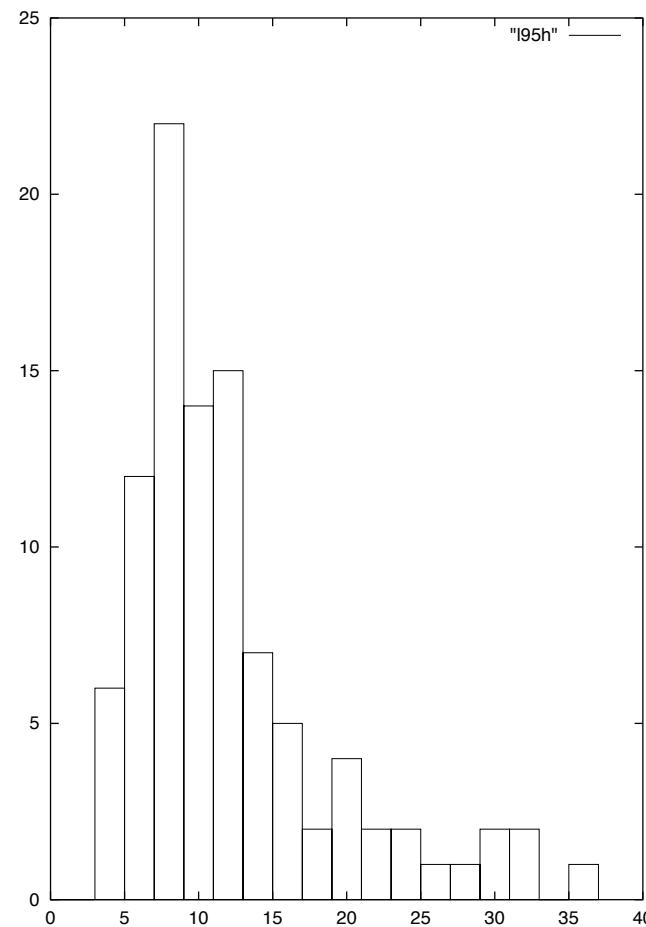


Figure 23: M/M/1; Nombre moyen de clients pour 100 simulations distinctes; charge=0,95; 1000 clients environ



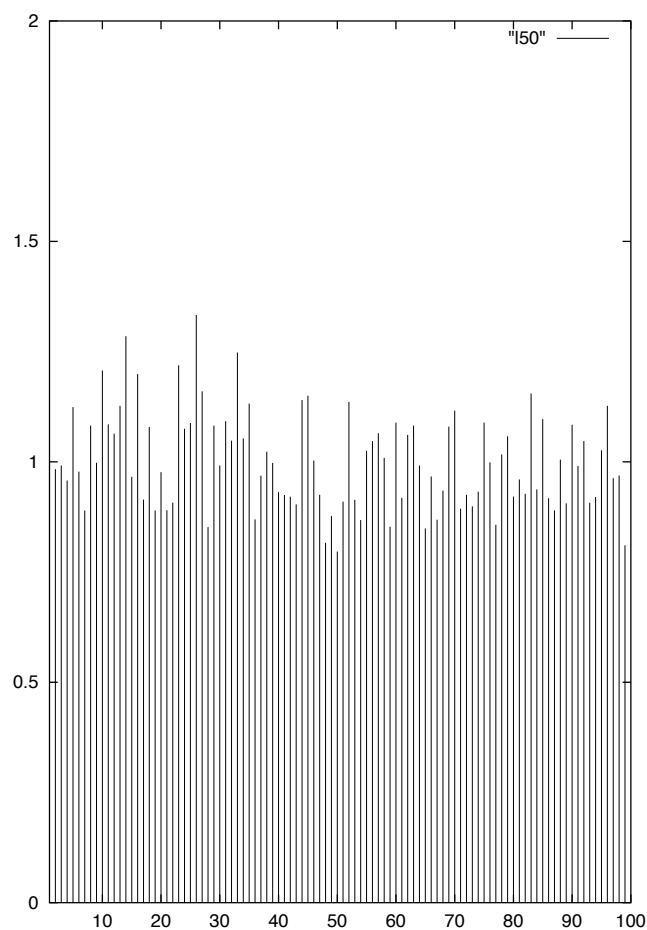


Figure 25: M/M/1; Nombre moyen de clients pour 100 simulations distinctes; charge=0,5; environ 1000 clients

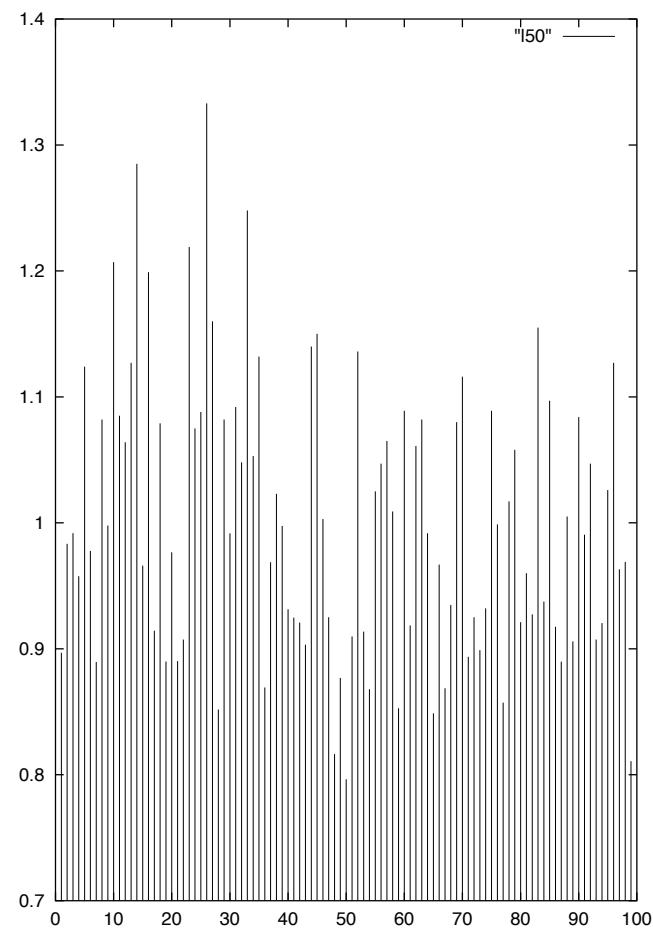


Figure 26: Différence ????

Solution : multiplier les expériences.

Idée : si les expériences sont indépendantes et correspondent à une population homogène de variance et moyennes finies, alors le théorème central limite s'applique.

Rappel : convergence de la VA “moyenne” de  $n$  expériences vers une Gaussienne  $(\mu, \sigma/\sqrt{n})$

Utilisation des Quantiles de la loi Normale (ou d'une autre distribution si le théorème ne s'applique pas à cause d'échantillons trop peu nombreux)

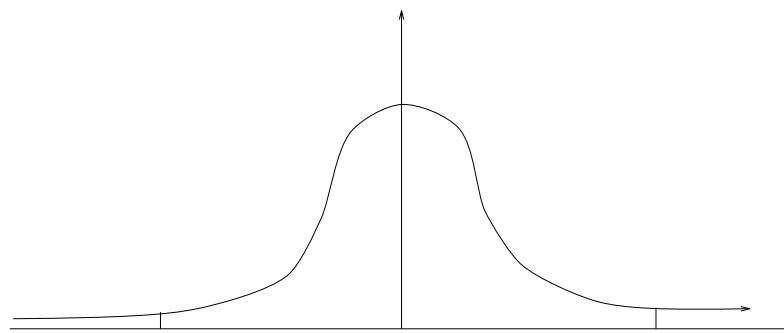
Trouver  $(a, b)$  tel que  $Pr(a \leq X \leq b) \geq 1 - \alpha$ .

$\alpha$  : typiquement 0.1 ou 0.05,

## Calcul des Intervalles de Confiance

1. pour une moyenne
2. pour une probabilité
1. avec de nombreux échantillons ( $\geq 30$ ) : assymptotique Gaussienne
2. avec peu d'échantillons mais Gaussiens : assymptotique  $t$  de Student

## Principe pour l'assymptotique Gaussienne



On mesure  $X$  moyenne et  $S$  écart type des  $n$  expériences.

On suppose la convergence vers  $\mathcal{N}(X, S/\sqrt{n})$

Quantile à  $a$  : valeur  $b$  de  $X$  tel que  $Pr(X < b) = a$

## Intervalle de Confiance pour la moyenne

1. : nombre d'échantillons  $\geq 30$

$$I = X \pm z_{1-\alpha/2} S / \sqrt{n}$$

2. : peu d'échantillons mais Gaussiens :

$$I = X \pm t_{(1-\alpha/2, n-1)} S / \sqrt{n}$$

avec  $z_{1-\alpha/2}$  quantile de la loi normale à  $(1 - \alpha/2)$ .

et  $t_{(1-\alpha/2, n-1)}$  quantile de la loi  $t$  de Student avec  $n-1$  degré de liberté

## Intervalle de Confiance pour une probabilité

Ou un ratio...

$n$  échantillons,  $n_1$  cas favorables observées.

Construire un estimateur de la proportion réelle à partir de la proportion mesurée :  $p = n_1/n$ .

Condition  $np > 10$

$$I = p \pm z_{1-\alpha/2} \sqrt{\frac{p(1-p)}{n}}$$

avec  $z_{1-\alpha/2}$  quantile de la loi normale à  $(1 - \alpha/2)$ .

## Travail sur les trajectoires

Traiter les transitoires et les intervalles de confiance

1. Moyennes par duplication
2. Méthode des blocs (batch means)
3. Régénération

## Moyenne par duplication

- Idée : on duplique les expériences
- Heuristiques : une expérience moyenne a moins de variabilité, ou ”à l'état stationnaire la moyenne est stable”
- Contrôler les générateurs pour essayer d'avoir de l'indépendance
- Heuristique : éviter les chevauchements de séquences

## **Transitoire**

$x_{i,j}$  : j-ème observation, duplication  $i$ .

$m$  duplications, expériences de longueur  $n$

On regarde la stabilité relative de la moyenne tronquée aux états "non transitoires".

1. Création d'une expérience "moyenne" :  $y_j = \frac{\sum_i x_{i,j}}{m}$
2. Calcul de la moyenne de l'expérience moyenne  $E(y) = \frac{\sum_{j=1}^n y_j}{m}$
3.  $l = 1$
4. Détruire les  $l$  premières observations, calculer la moyenne entre  $l$  et  $n$  :  $z_l = \frac{\sum_{j=l}^n y_j}{m}$
5. Calculer la variation relative  $R(l) = \frac{z_l - E(y)}{E(y)}$
6. incrémenter  $l$  et recommencer 4 et 5. Tracer  $R(l)$ . Le coude de la courbe indique la fin des transitoires. De même  $|R(l) - R(l + 1)| < \epsilon$

## Intervalles de confiance

$m$  duplications, de longueur  $n + l$ ,  $l$  longueur des transitoires (otées)

1. Calculer la moyenne de chaque expérience  $x_i = \frac{\sum_{j=l+1}^{l+n} x_{i,j}}{n}$
2. Calculer la moyenne des expériences.  $E(x) = \frac{\sum_{i=1}^m x_i}{m}$
3. Calculer la variance  $Var = \frac{\sum_{i=1}^m (x_i - E(x))^2}{m-1}$

## Problèmes et Avantages

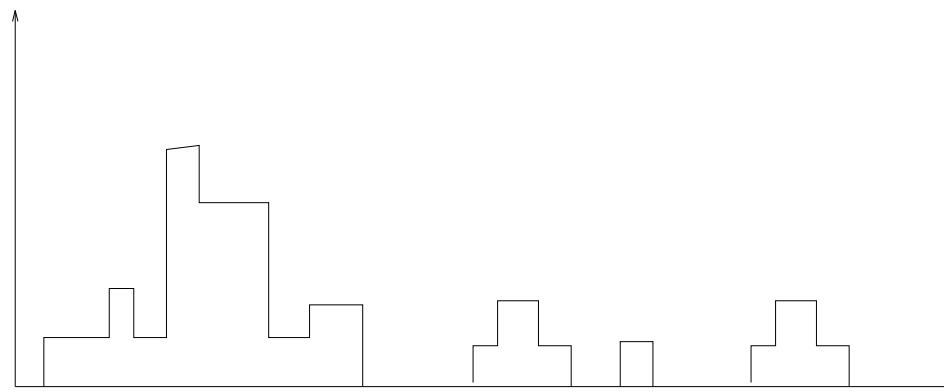
- Cout : on rejette  $ml$  points...
- Contrôle des générateurs....
- Réseaux de stations (parallélisme simple)
- On peut continuer les expériences jusqu'à convergence

## Régénération

Propriétés stochastiques : processus régénératifs.

Points de régénération : le processus perd la mémoire de son passé lorsqu'il est en ce point.

Exemple : un système d'attente vide...



## Propriétés

- Ces points délimitent des expériences indépendantes (au sens probabiliste)
- Ces trajectoires entre points de régénération sont distribuées selon l'état stationnaire.
- Transitoire : depuis le point initial jusqu'à la première régénération
- Transitoire nul si le point initial est un point de régénération

On suppose  $m$  cycles de taille  $n_1, \dots, n_m$ .

Les points sont numérotés  $x_{i,j}$ ,  $i$  numéro de cycle.

1. Calcul de la moyenne sur chaque cycle :  $x_i = \frac{\sum_{j=1}^{n_i} x_{i,j}}{n_i}$
2.  $y_i = \sum_{j=1}^{n_i} x_{i,j}$
3. Calcul de la moyenne globale :  $E(x) = \frac{\sum_i y_i}{\sum_i n_i}$
4.  $w_i = y_i - n_i E(x)$
5. Calcul de la variance des différences :  $Var = \frac{\sum_{i=1}^m (w_i)^2}{m-1}$
6. Longueur moyenne de cycle :  $\bar{n} = \frac{\sum_i n_i}{m}$

Intervalle :  $I = E(x) \pm z_{1-\alpha/2} \sqrt{\frac{Var}{(\bar{n})^2 m}}$

## Problèmes et Avantages

- Une vraie indépendance
- Prouver les points de régénération (preuve mathématique)
- Fréquence des points de régénération : dépend des paramètres de charge
- Biais : on favorise les expériences courtes

## **Comment comparer des systèmes**

- Comparaison Trajectorielle
- Comparaison Statistique
- Adéquation à une distribution

La comparaison des simples moyennes est une bêtise....

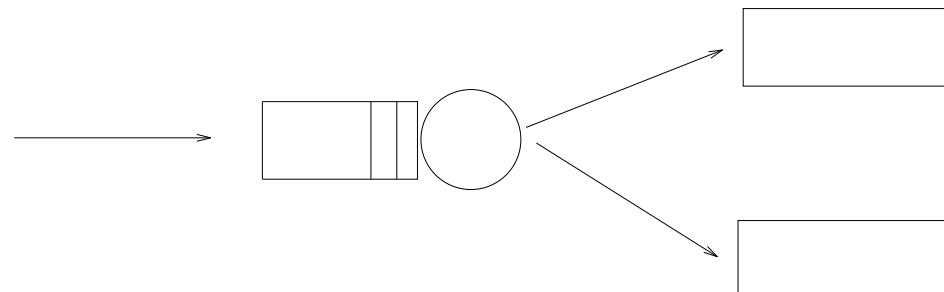
## Comparaison Trajectorielle

$X, Y$  : mesures réalisées pour deux systèmes.

**Définition 3** *La trajectoire de  $X$  est toujours supérieure à la trajectoire de  $Y$*

Attention :  $X$  et  $Y$  doivent être générées par les mêmes séquences aléatoires.

## Exemple : Nombre de Pertes



Un buffer de taille  $B$  perd trajectoriellement moins de clients qu'un buffer de taille  $B - 1$ .

Pour services déterministes :

$$\{\text{dates de perte pour } B\} \in \{\text{dates de perte pour } B-1\}$$

- Problème : Vérifier que les séquences aléatoires générant les arrivées dans les deux systèmes sont les mêmes.
- Attention : puisque plus de clients sortent dans le cas  $B$ , il y a plus de génération de nombre aléatoires pour le routage.
- Conclusions : On doit avoir deux générateurs séparés.

## Comparaison Statistique

$X_i$  et  $Y_i$  résultats expérimentaux

- Question : la différence provient-elle d'une supériorité ou du "bruit statistique"
- Réponse : théorie des tests: Construire  $X - Y$  et comparer à un "zero statistique"
- 2 constructions pour  $X - Y$  : expériences appariées ou non appariées.

Question : Quel est le "meilleur" système ?

Temps	Exp1	Exp2	Exp3	Exp4	Exp5	Exp6
Système 1	5.4	16.6	0.6	1.4	0.6	7.3
Système 2	19.1	3.5	3.4	2.5	3.6	1.7

Réponse : avec un intervalle de confiance à 90 %, ils sont statistiquement équivalents

## Expériences Apariées

**Définition 4** *On peut construire une bijection entre les mesures...*

*Bijection :*

- *Même conditions expérimentales sur les deux systèmes.*
- *Même nombre de prises de mesures sur les deux systèmes.*

Exemples corrects :

- temps pour executer la même requête sur deux SGBD
- nombre de clients dans une file pour deux ordonnancements  
(avec la même séquence d'arrivées)

Exemples faux :

- Temps entre deux pannes observées sur une durée finie (le nombre de pannes n'est pas nécessairement le même)
- Comparaison ds temps moyen de séjour dans deux files avec pertes (le nombre d'entrées n'est pas forcément le même)

## Algorithme

Idée : on construit la différence et on teste si la moyenne est statistiquement nulle avec un risque donné  $\alpha$ .

On compare  $(X_i)$  et  $(Y_i)$ ,  $i = 1..n$ .

1.  $Z_i = X_i - Y_i$ ,  $\forall i = 1..n$ .
2. On calcule  $E(Z)$ .
3. On calcule  $Var(Z)$
4. On calcule  $\mathcal{I}$  l'intervalle de confiance pour  $Z$  avec le risque  $\alpha$ .
5. Si 0 est élément de  $\mathcal{I}$  alors  $Z$  n'est pas statistiquement différent de 0 et les deux systèmes sont donc "équivalents".

Sinon le signe de  $E(Z)$  indique le meilleur système.

## Systèmes non appariés

$n_X$  et  $n_Y$  les tailles des échantillons.

1. Calcul des moyennes :  $E(X)$  et  $E(Y)$
2. Calcule des variances  $Var(X) = \frac{(\sum_i X_i)^2 - n_X E(X)^2}{n_X - 1}$ , idem pour  $Var(Y)$ .
3. Calcul de la différence  $E(X) - E(Y)$
4. Calcul de la variance de la différence  $Var = \frac{Var(X)}{n_X} + \frac{Var(Y)}{n_Y}$
5. Calcul du degré de liberté  $\nu = \frac{(Var(X)/n_X + Var(Y)/n_Y)^2}{\frac{1}{n_X+1}(\frac{Var(X)}{n_X})^2 + \frac{1}{n_Y+1}(\frac{Var(Y)}{n_Y})^2} - 2$

6. Calcul l'intervalle de confiance  $\mathcal{I}$ .

$$\mathcal{I} = E(X) - E(Y) \pm \sqrt{Var t_{1-\alpha/2,\nu}}$$

7. Si 0 est élément de  $\mathcal{I}$  alors  $X$  n'est pas statistiquement différent de  $Y$  et les deux systèmes sont donc "équivalents". Sinon le signe de  $E(X) - E(Y)$  indique le meilleur système.

## Génération d'objets complexes

- Arbres, Graphes, sous ensembles
- Génération uniforme ou selon une distribution
- Structure avec ou sans étiquettes
- méthodes constructives ou asymptotiques (marche aléatoire sur des graphes)

## Permutation $\sigma$ uniforme sur $n$ objets

Algorithme de Knuth :

1. Construire un tableau  $T$  de taille  $n$  et y ranger les nombres de 1 à  $n$
2. Pour  $i$  de 1 à  $n$  faire :
  - (a) Générer  $u$  un nombre uniforme entre 1 et  $n + 1 - i$
  - (b)  $\sigma(i)$  reçoit  $T(u)$
  - (c)  $T(u)$  reçoit  $T(n + 1 - i)$

## Marche Aléatoire sur des graphes

- Soit une chaîne de Markov en temps discret de matrice de transition  $P$ , apériodique, irréductible de taille  $n$ .
- **Définition 9 (Mesure Invariante)**  $\pi$  une mesure invariante de  $P$  ssi  $\pi P = \pi$
- $\pi(j)$  est la probabilité asymptotique d'être à l'état  $j$  partant d'un état quelconque...
- **Définition 10 (Mesure réversible)**  $\pi$  est une mesure réversible ssi  $\pi(i)P[i, j] = \pi(j)P[j, i]$ .

## Algorithme de visite d'une chaîne de Markov

$K$  étapes.

1.  $n := 0$ ; Initialiser  $X$
2. tant que  $n \leq K$ 
  - (a)  $i := X$ ;
  - (b) Choisir  $j$  selon probabilité  $P[i, j]$
  - (c)  $X := j$ ;
  - (d)  $n := n + 1$ ;

Résultat simple et très utile pour la génération :

Si  $P$  est symétrique,  $P$  admet la loi uniforme comme mesure réversible

**Marche aléatoire symétrique → génération uniforme**

## Méthode de Rejet

Si l'ensemble est inclus dans une structure de graphe, on obtient une chaîne de Markov sur l'ensemble initial avec mesure invariante uniforme en simulant la marche aléatoire symétrique sur le surensemble et en rejetant les transitions faisant sortir de l'ensemble.

## Résultat Fondamental

- Soit  $Q$  une matrice de transition telle que  
$$Q[i, j] > 0 \rightarrow Q[j, i] > 0$$
- Soit  $\pi$  une distribution quelconque de proba,
- On définit  $P = Q[i, j] \min(1, \frac{\pi(j)Q[j, i]}{\pi(i)Q[i, j]})$  si  $Q[i, j] > 0$  et 0 pour les éléments non diagonaux. La diagonale sert à la normalisation.
- Alors  $P$  admet  $\pi$  comme mesure réversible.

Pour générer des structures selon une distribution quelconque

1. construire une chaîne de Markov dont les sommets sont les structures
2. les transitions sont données par la formule précédente
3. la matrice doit être symétrique pour obtenir une loi uniforme puis visiter la chaîne pendant un temps assez long (???).

## Algorithme de Hastings Metropolis

1.  $n := 0$ ; Initialiser  $X$
2. tant que  $n \leq K$ 
  - (a)  $i := X$ ;
  - (b) Choisir  $j$  selon probabilité  $Q[i, j]$
  - (c)  $\rho = \pi(j)Q[j, i]/(\pi(i)Q[i, j])$ ;
  - (d) si ( $\rho \geq 1$ ) alors  $X := j$
  - (e) sinon si ( $\text{Random} < \rho$ ) alors  $X := j$
  - (f)  $n := n + 1$ ;

## Génération uniforme d'un stable

- soit  $G$  un graphe non orienté
- un stable est un sous ensemble des sommets tel qu'il n'y a pas dans  $G$  d'arête entre deux quelconque des ces sommets.
- Déterminer tous les stables d'un graphe est un problème difficile.
- L'algorithme simule une chaîne qui admet la loi uniforme sur  $E$  comme mesure réversible.  $E$  est l'ensemble des stables de  $G$ .
- $R$  est l'état actuel de la chaîne. C'est un stable.

1.  $m := 0; R := \emptyset$
2. Tant que  $m < K$ 
  - (a) choisir  $x$  un sommet de  $G$  (uniforme) ;
  - (b) si  $(x \in R)$  alors  $R := R - x$
  - (c) sinon si  $x$  n'est connecté à aucun sommet de  $R$  alors  
 $R := R + x$
  - (d)  $m := m + 1;$

- Idée : l'ensemble des stables est inclus dans l'ensemble des parties des sommets de  $G$ .
- Cet ensemble a une structure naturelle d'hypercube (arête entre deux parties qui ne diffèrent que par un seul sommet).
- Marche aléatoire sur l'hypercube.

## Chercher un grand stable

- Mettre une loi de proba fonction de la taille.
  - Par exemple :  $Pr(R) = \alpha \lambda^{|R|}$  avec  $\lambda > 1$  et  $\alpha$  normalisation.
1.  $m := 0; R := \emptyset$
  2. Tant que  $m < K$ 
    - (a) choisir  $x$  un sommet de  $G$  (uniforme) ;
    - (b) si  $(x \in R)$  et  $(\text{Random} < 1/\lambda)$  alors  $R := R - x$
    - (c) sinon si  $x$  n'est connecté à aucun sommet de  $R$  alors  
 $R := R + x$
    - (d)  $m := m + 1;$

Lien avec le recuit simulé... et l'optimisation.

## Bibliographie

- The art of computer systems performance analysis, Raj Jain, Wiley, 1992
- Simulation Modeling and Analysis, A. Law, D. Kelton, Mac Graw Hill.
- Méthodes statistiques à l'usage des médecins et des biologistes, D. Schwartz, Flammarion, coll. Médecine-Sciences
- Monte Carlo, concepts, algorithms, and applications, G.S. Fishman, Springer
- Simulation model design and execution : building digital worlds, P. Fishwick, Prentice Hall.