INTRODUCTION. — INFORMATIQUE ET PROBABILITÉS

L'usage de méthodes probabilistes en Informatique remonte à la naissance de cette dernière. En effet, le traitement automatique de l'information s'est au départ développé pour la résolution approchée de problèmes numériques pour lesquels des solutions théoriques ne pouvaient être exprimées ou évaluées raisonnablement et même de manière approximative à la main ou à l'aide des moyens de calcul antérieurs (règles à calcul, tables, etc.).

En particulier, trois types courants de calculs ont été très souvent abordé de manière probabiliste : calcul d'intégrales simples ou multiples (méthodes de Monte-Carlo), détermination de minima locaux ou globaux de fonctions de plusieurs variables (méthodes de recuit simulé), résolution d'équations aux dérivées partielles (simulations de solutions d'équations différentielles stochastiques). Dans ces trois exemples, le hasard est absent du problème initial ; il n'apparaît que dans la méthode de résolution.

Avec la progression de l'usage du calcul sur ordinateur et la nécessité de modéliser des systèmes de plus en plus complexes (météorologie, traffic routier, formation d'agrégats polymoléculaires, etc.), le hasard a été introduit comme une composante de certains modèles. Certains pourront penser que son introduction remplace une description plus précise des mécanismes régissant l'évolution de tels systèmes. Néanmoins, l'élément aléatoire est ici présent dès le problème initial.

1. **Matériel probabiliste de base**

Dans la plupart des situations où le hasard intervient, on a besoin d'une suite de variables aléatoires indépendantes uniformément distribuées sur l'intervalle $[0,1]$. Rappelons de quoi il s'agit.

Soit $(\Omega, \mathcal{A}, \mathbb{P})$ un espace probabilisé — la grande urne à la source de laquelle on puise le hasard. Une suite de variables aléatoires $U_n : (\Omega, \mathcal{A}, \mathbb{P}) \to [0,1]$, $n \geqslant 1$, est simplement une suite d'applications (mesurables) de l'ensemble Ω dans l'intervalle $[0,1]$:

$$\begin{aligned} U_n : \Omega &\longrightarrow [0,1] \\ \omega &\longmapsto U_n(\omega) \end{aligned}$$

(telle que pour tout intervalle $[a,b] \subset [0,1]$, $\{U_n \in [a,b]\} = \{\omega \in \Omega : U_n(\omega) \in [a,b]\} \in \mathcal{A}$). Une telle variable aléatoire U_n est uniformément distribuée sur l'intervalle $[0,1]$ si et seulement si pour tous $0 \leqslant a \leqslant b \leqslant 1$,

$$\mathbb{P}\{U_n \in [a,b]\} = \mathbb{P}\{\omega \in \Omega : U_n(\omega) \in [a,b]\} = b - a.$$

Des variables aléatoires $(U_n)_{n \geqslant 1}$ uniformément distribuées sur l'intervalle $[0,1]$ sont indépendantes si et seulement si pour tout $n \geqslant 1$, pour tous $0 \leqslant a_k \leqslant b_k \leqslant 1$, $1 \leqslant k \leqslant n$, on a

$$\mathbb{P}\{U_1 \in [a_1,b_1], \ldots, U_k \in [a_k,b_k], \ldots, U_n \in [a_n,b_n]\} = (b_1 - a_1) \times \cdots \times (b_k - a_k) \times \cdots \times (b_n - a_n).$$

Interprétation. — On tire ω dans Ω *au hasard*. On ne sait pas *a priori* (c'est-à-dire avant de savoir quel ω particulier a été tiré) quelle valeur prend la fonction U_n en ω. Dire que U_n est uniformément distribuée sur $[0,1]$, c'est dire que pour $0 \leqslant a \leqslant b \leqslant 1$, *a priori* la valeur $U_n(\omega)$ a une probabilité égale à $b - a$ d'être dans l'intervalle $[a,b]$. La condition

d'indépendance est semblable : on ne sait pas *a priori* quelles seront les valeurs du n-uplet $(U_1(\omega), \ldots, U_n(\omega))$, mais l'hypothèse d'indépendance affirme que la probabilité d'avoir ce n-uplet dans l'intervalle produit $[a_1, b_1] \times \cdots \times [a_n, b_n]$ est égale au produit des probabilités que chacune des valeurs $U_k(\omega)$ soit dans l'intervalle correspondant $[a_k, b_k]$. Si chacune des variables aléatoires U_k est uniformément distribuée sur $[0, 1]$, ce produit de probabilités est égal à $(b_1 - a_1) \times \cdots \times (b_n - a_n)$.

L'intérêt de variables aléatoires uniformément distribuées sur $[0, 1]$ réside en partie dans le principe suivant :

Principe. — *Soit π une loi de probabilité sur un ensemble raisonnable E (loi d'une variable aléatoire X à valeurs dans E) et U une variable aléatoire uniformément distribuée sur $[0, 1]$. Alors il existe une fonction $f : [0, 1] \to E$ telle que la variable aléatoire $X = f(U)$ ait pour loi π.*

Remarques. — *a*) Par ensemble raisonnable, on peut entendre : E ensemble quelconque fini ou dénombrable ; E sous-ensemble de $\mathbb{R}$, etc. Nous en verrons des exemples. C'est aussi sur des exemples que nous déterminerons une telle fonction f ou verrons si d'autres méthodes ne sont pas plus raisonnables en pratique.

b) En vertu de ce principe, on peut réaliser toute suite de variables aléatoires indépendantes $(X_n)_{n\geqslant 1}$ raisonnables — à valeurs dans des ensembles $(E_n)_{n\geqslant 1}$ raisonnables — comme une suite $(f_n(U_n))_{n\geqslant 1}$ pour $(U_n)_{n\geqslant 1}$ une suite de variables aléatoires indépendantes uniformément distribuées sur $[0, 1]$ et $(f_n)_{n\geqslant 1}$ une suite de fonctions convenablement choisies.

c) Mieux encore, on peut réaliser toute suite raisonnable de variables aléatoires $(X_n)_{n\geqslant 1}$, par exemple une chaîne de Markov à espace d'états fini ou dénombrable, à l'aide d'une telle suite $(U_n)_{n\geqslant 1}$. Simplement, la construction ne se fait pas nécessairement terme à terme si les variables aléatoires $(X_n)_{n\geqslant 1}$ ne sont pas indépendantes.

Parmi les nombreux résultats importants de la Théorie des Probabilités, on a la

Loi forte des grands nombres. — *Soit $(\xi_n)_{n\geqslant 1}$ une suite de variables aléatoires à valeurs dans $\mathbb{R}$ (ou $\mathbb{R}^d$), indépendantes, de même loi, intégrables. En notant $\mu = \mathbb{E}[\xi_n] \in \mathbb{R}$ (ou $\mathbb{R}^d$) — quantité qui est la même pour tout n par hypothèse —, on a*

$$\frac{1}{n}(\xi_1 + \cdots + \xi_n) \longrightarrow \mu \qquad \textit{quand } n \to \infty \textit{ (presque sûrement).}$$

Ce qui est remarquable c'est qu'alors que chaque moyenne partielle $(\xi_1 + \cdots + \xi_n)/n$ dépend du hasard, à la limite le hasard disparaît (presque, parce qu'en toute rigueur la convergence n'a lieu qu'avec probabilité 1). En gros, chaque ξ_n s'écarte de la même façon — ces variables aléatoires ont même loi — de leur valeur moyenne commune μ. Ce qu'il se passe est que, du fait de l'indépendance, à ω donné, les nombres (ou vecteurs) $(\xi_n(\omega))_{n\geqslant 1}$ se répartissent mutuellement selon cette loi. Ainsi, lorsqu'on prend la limite de ces moyennes partielles, on obtient nécessairement μ.

2. Générateurs aléatoires

Pour réaliser des simulations probabilistes, c'est-à-dire mettre un œuvre un modèle contenant une part d'aléatoire, il serait souhaitable de pouvoir générer des suites $(u_n)_{n\geqslant 1}$ de nombres dans $[0, 1]$ qui soient, ou représentent, une suite $(U_n(\omega))_{n\geqslant 1}$ où $(U_n)_{n\geqslant 1}$ est une suite de variables aléatoires uniformément distribuées sur $[0, 1]$. Aussi bien conceptuellement que pratiquement cela pose des problèmes.

- La suite $(u_n)_{n\geqslant 1}$ peut-elle prendre n'importe quelle valeur réelle dans $[0,1]$? Les représentations informatiques des nombres — c'est-à-dire des représentations finies — ne permettent pas d'accéder à tous les nombres réels, seulement à un nombre fini d'entre eux.
- Comment faire intervenir un hasard réel dans les choix successifs des $(u_n)_{n\geqslant 1}$? Des procédés physiques existent pour cela et se basent sur la nature quantique de la matière à petite échelle. Mais pour toute expérience scientifique — dont les simulations aléatoires — on exige la reproductibilité de son déroulement. Si un tel hasard est mis en œuvre pour une simulation, la reproductibilité risque fort d'être perdue.
- En quoi une suite particulière $(u_n)_{n\geqslant 1}$ représente-t-elle plus ou mieux une réalisation de $(U_n)_{n\geqslant 1}$ qu'une autre ? Des suites numériques même assez régulières (et même constantes) seraient pour le mathématicien tout aussi recevables (ou presque).

Ces problèmes et les solutions qui ont été proposées pour y remédier sont abondamment discutées dans

> Knuth (D. E.), *The Art of Computer Programming, volume 2, Seminumerical Algorithms*, third edition, Addison–Wesley (1998),

et dépassent largement notre propos. Nous pouvons seulement, et très sommairement, indiquer quelle est la nature de ces solutions.

- Si on ne peut pas accéder à tous les nombres réels de l'intervalle $[0,1]$, au moins peut-on considérer un « grand nombre » d'entre eux régulièrement espacés afin d'approcher autant que possible une répartition uniforme dans cet intervalle. Ceci se fait classiquement en considérant l'ensemble des entiers $\{0, 1/m, \ldots, (m-1)/m\}$ où m est un entier « grand ». Ainsi, au lieu de regarder une suite à valeurs dans $[0,1]$, on s'intéresse aux suites à valeurs dans l'ensemble à m éléments $\{0, 1, \ldots, m-1\}$.
- L'exigence de reproductibilité impose que la suite $(x_n)_{n\geqslant 1}$ à valeurs dans $\{0, 1, \ldots, m-1\}$, où $u_n = x_n/m$, soit donnée de manière algorithmique. En général, on se donne un certain entier $k \geqslant 1$, une certaine fonction $\phi : \{0, 1, \ldots, m-1\}^k \to \{0, 1, \ldots, m-1\}$, une donnée initiale $(x_1, \ldots, x_k)$, et on calcule de manière itérative $x_{n+1} = \phi(x_n, \ldots, x_{n-k+1})$. Il est à noter que la suite obtenue évolue dans un un ensemble fini (précisément $\{0, 1, \ldots, m-1\}^k$) et qu'elle reviendra nécessairement à un k-uplet déjà atteint ; alors elle bouclera, c'est-à-dire aura une évolution périodique.
- Les propriétés mathématiques que le probabiliste serait tenté de demander ne porteraient pas sur une suite particulière $(x_n)_{n\geqslant 1}$ mais sur l'ensemble des suites qu'on pourrait obtenir de cette façon. Ceci n'ayant pas d'application concrète — c'est à partir d'une seule suite $(x_n)_{n\geqslant 1}$ que se fait couramment une simulation —, c'est sur le comportement en n que se portent les exigences.

  On exige ainsi dans un premier temps que chaque k-uplet soit atteint au cours d'une période, ce qui signifie que la suite $(x_n)_{n\geqslant 1}$ passe par chaque élément de $\{0, 1, \ldots, m-1\}$ avec la même fréquence au cours de chacune de ses périodes. Ceci correspond à l'hypothèse selon laquelle chaque U_n est uniformément distribuée sur $[0,1]$.

  Pour rendre compte de l'indépendance des $(U_n)_{n\geqslant 1}$, certains critères ont été formulés sur ce que doit satisfaire de plus une telle suite de nombres $(x_n)_{n\geqslant 1}$, disons seulement qu'une certaine « imprédictibilité » est requise dans le passage d'un terme au suivant — ce qui bien sûr n'est pas, au sens strict, réalisable puisque la suite reste déterministe.

Exemple fondamental. — La classe de générateurs la plus simple est donnée, d'une part, pour $k = 1$, c'est-à-dire avec $x_{n+1} = \phi(x_n)$, et avec ϕ la fonction la plus simple qu'on puisse imaginer dans ce cadre, à savoir une fonction affine modulo l'entier maximal m :

$$x_{n+1} = (a \times x_n + b) \bmod m,$$

ainsi x_{n+1} est encore un entier compris entre 0 et $m-1$. Ce type de relation est appelée *congruence linéaire* et les propriétés d'une suite ainsi définie sont bien connues des spécialistes. Le choix des entiers a, b, et m n'est pas quelconque. Pour ne discuter que de m, ce doit être un entier grand dans le domaine de calcul de la machine ou du logiciel utilisé ; il est de plus essentiel que m soit un nombre premier de la forme 2^p-1, c'est-à-dire un nombre de Mersenne. Cette méthode est souvent appelée « méthode des congruences linéaires » et est due à D. H. Lehmer (1948).

Des logiciels de calcul scientifique ou statistique utilisent ce type de générateurs avec des choix de (a, b, m) éventuellement différents. Par exemple,

$$m = 2^{31}-1 = 2\,143\,483\,647, \qquad a = 7^5 = 16\,807, \qquad b = 0$$

est un choix qui satisfait — ou presque — les propriétés qu'on peut attendre d'un tel générateur. Remarquons que $m = 2^{31}-1$ est le plus grand nombre de Mersenne qui reste dans le domaine de calcul en entiers d'un processeur 32 bits.

Exercice optionnel. — Faire la liste des nombres de Mersenne inférieurs à 2^{32}.

Autres exemples. — Hormis les générateurs aléatoires mis en œuvres et documentés par D. E. Knuth, ou dans des logiciels de calcul scientifique réputés, il n'est pas toujours très facile d'avoir une description claire du mécanisme d'un générateur aléatoire employé dans tel logiciel ou tel langage de programmation. Par exemple, le, ou les, générateur en `gcc` ne sont que très partiellement documentés (sinon, me fournir une documentation satisfaisante). Pire, les implémentations de ces générateurs ne sont pas nécessairement identiques au fil des versions des logiciels ou des langages. Ceci peut obliger à définir son propre générateur aléatoire lorsqu'on désire obtenir un programme stable et vérifiable.

3. Travaux dirigés et pratiques

Une partie des travaux dirigés sera de nature théorique (traitée sur papier et/ou au tableau), une autre sera de nature pratique. Pour la partie pratique — effectuée sur ordinateur —, il est laissé une assez large marge de manœuvre.

Calculs et/ou simulations. — Nous n'imposons pas l'usage d'un langage de programmation particulier. Le `C` présente néanmoins l'avantage d'être assez universel pour qu'on désire le retenir. Il présente le désavantage qu'un programme doive être compilé pour être exécuté, ce qui est moins souple que le cas d'un langage interprété. Les trois exigences naturelles sont que le langage employé doit : gérer des entiers de taille suffisamment importante, des nombres sous forme flottante, des tableaux, disposer d'un générateur aléatoire (sinon, pouvoir le définir soi-même) ; être globalement compréhensible pour un néophyte, c'est-à-dire que modulo quelques courtes explications une personne ignorante de ce langage doit pouvoir comprendre ce que fait un programme ; avoir la capacité d'écrire les résultats de calculs dans un fichier annexe pour un traitement ultérieur à l'aide d'un autre logiciel.

Représentation et synthèse des résultats numériques. — On aura parfois à faire avec des listes de nombres. Celles-ci devront être lisibles par un tableur standard capable de calculer des moyennes, des variances, etc., et tracer des diagrammes statistiques courants (diagrammes en bâtons, histogrammes, etc.).

Université de Poitiers
Département de Mathématiques

FEUILLE N° 1

EXERCICE 1. — Soit $(U_i)_{i\in\mathbb{N}}$ une suite de variables aléatoires indépendantes, uniformément distribuées sur $[0,1]$. Décrire à l'aide d'une ou plusieurs variables aléatoires de la suite $(U_i)_{i\in\mathbb{N}}$ diverses procédures pour réaliser une variable aléatoire :

(i) de loi de Bernoulli $\mathcal{B}(1,p)$, $p\in[0,1]$;

(ii) discrète à n valeurs possibles $\{x_1,\dots,x_n\}$ de probabilités respectives $p_1,\dots,p_n$;

(iii) de loi binomiale $\mathcal{B}(n,p)$, $p\in[0,1]$, $n\in\mathbb{N}$ (différemment de la réponse précédente) ;

(iv) de loi uniforme $\mathcal{U}([a,b])$, $a<b$;

(v) positive de loi exponentielle $\mathcal{E}(\lambda)$, $\lambda\in\mathbb{R}_+^*$;

(vi) de loi Normale (gaussienne centrée réduite) $\mathcal{N}(0,1)$ (utiliser Φ la fonction de répartition de la loi Normale) ;

(vii) de loi gaussienne $\mathcal{N}(\mu,\sigma^2)$, $\mu\in\mathbb{R}$, $\sigma\in\mathbb{R}_+^*$.

EXERCICE 2. — Soient π une mesure de probabilité sur $\mathbb{R}$ et F sa fonction de répartition, c'est-à-dire $F(x)=\pi(]-\infty,x])$ pour tout $x\in\mathbb{R}$.

(i) Supposons que F est strictement croissante et continue sur un intervalle $]a,b[$ et vérifie $\lim_{x\downarrow a}F(x)=0$ et $\lim_{x\uparrow b}F(x)=1$. Montrer que si X est une variable aléatoire de loi π, alors la variable aléatoire $U=F\circ X$ est de loi uniforme sur $[0,1]$.

(ii) Plus généralement, définissons l'inverse généralisé F^{-1} de F par

$$F^{-1}(u)=\inf\{y\in\mathbb{R}:u\leqslant F(y)\}\qquad\text{pour tout } u\in[0,1],$$

(qui est une fonction croissante, continue à gauche et limitée à droite). Soient U une variable aléatoire de loi uniforme sur $[0,1]$ et $X=F^{-1}\circ U$. Montrer que les événements $\{X\leqslant x\}$ et $\{U\leqslant F(x)\}$ sont égaux pour tout $x\in\mathbb{R}$; en déduire que la variable aléatoire X a pour loi π.

EXERCICE 3. — Soient X et Y deux variables aléatoires indépendantes de lois $\mathcal{N}(0,1)$. Le couple (X,Y) étant considéré comme les coordonnées d'un point aléatoire du plan, on passe en coordonnées polaires : $R=\sqrt{X^2+Y^2}$, $\Theta=\arctan(Y/X)$.

(i) Montrer que R et Θ sont indépendantes et déterminer leurs lois respectives.

(ii) En déduire une méthode pratique pour simuler une variable aléatoire de loi $\mathcal{N}(0,1)$, puis, plus généralement, de loi $\mathcal{N}(\mu,\sigma^2)$.

Remarque. — La méthode présentée dans l'exercice précédent est couramment appelée « méthode polaire ». Elle a été introduite dans l'article de G. E. P. Box et Mervin E. Muller, « A Note on the Generation of Random Normal Deviates », *The Annals of Mathematical Statistics*, vol. 29 (1958), p. 610–613.

FEUILLE N° 2

1. Un générateur aléatoire

Pour commencer, on définit une procédure informatique, ou fonction, qui à chacun de ses appels retourne les valeurs successives de la suite

$$x_{n+1} = (a \times x_n + b) \bmod m,$$

avec

$$m = 2^{31} - 1 = 2\,143\,483\,647, \qquad a = 7^5 = 16\,807, \qquad b = 0$$

et x_1 à spécifier (initialisation de la graine aléatoire [*random seed*]). Les entiers a et b pourront être déclarés explicitement dans cette procédure puisqu'ils ne seront pas amenés à être changés par la suite. L'entier m pourra être utilisé par la suite. On stockera sa valeur dans une variable globale `myRandmax`.

La valeur courante de x_n sera stockée dans une variable entière globale `myCurrentRandom`. L'initialisation sera définie à l'aide d'une fonction `mySetRandom` à 1 argument entier ; cette fonction doit vérifier que l'argument est un entier compris entre 1 et $m - 1$ (la valeur initiale 0 est ici assez problématique puiqu'alors la suite obtenue stationne à 0), ou plus simplement, convenablement faire correspondre l'argument avec un tel nombre ; son rôle est d'affecter à la variable `myCurrentRandom` le nombre en question. Le calcul de la nouvelle valeur se fera avec une fonction qu'on nommera `myRandom` ; elle affectera à `myCurrentRandom` la valeur suivante dans la suite et retournera sa valeur.

Test. — Pour valider l'implémentation des fonctions qui précèdent, initialiser la graine aléatoire à 0 et afficher les 10 premières valeurs de la suite produite par la fonction `myRandom`. Comparer ses résultats avec ceux de ses voisins, se poser la question si à la vue de cette liste de nombres on peut inférer les termes suivants...

2. Lois usuelles

Pour chacunes des lois suivantes, définir une fonction utilisant `myRandom` pour tirer un nombre suivant la loi considérée, ses paramètres étant aussi ceux de la fonction en question. Exporter une liste de 100 nombres ainsi produits après réinitialisation du générateur aléatoire et choix simple de paramètres.

- Lois binomiales $\mathcal{B}(n, p)$ (fonction `myBinomial(.,.)`) ;
- Lois uniformes $\mathcal{U}(a, b)$ (fonction `myUniform(.,.)`) ;
- Lois exponentielles $\mathcal{E}(\lambda)$ (fonction `myExponential(.)`) ;
- Lois gaussiennes $\mathcal{N}(\mu, \sigma^2)$ (fonction `myGaussian(.,.)`) ;

3. Comparaisons

Réaliser les diagrammes statistiques correspondant à chacune des listes obtenues, les comparer graphiquement avec les lois de probabilité correspondantes.

FEUILLE N° 3

1. **Publication et génération de graphiques**

Sous UNIX, les outils classiques de formatage de documents ont été `troff` (`groff`, `nroff`), puis TEX et ses extensions usuelles comme LATEX. Un document (fichier où est le document à formater) LATEX s'écrit en gros sous la forme :

```
\documentclass[12pt,a4]{...}
\usepackage{...}
...

\begin{document}

\title{...}
\author{...}
\maketitle

\section*{Introduction}
...
\section{...}
...
\section{...}
...
\end{document}
```

où les « ... » sont à remplacer par ce qui convient. Le formatage du document se fait en exécutant `latex <`*document.tex*`>` (ou simplement `latex <`*document*`>`), ce qui produit un fichier `<` *document.dvi*`>` visualisable par la commande `xdvi <`*document.dvi*`>` (ou simplement `xdvi <`*document*`>`). Pour l'imprimer, exécuter `dvips <`*document.dvi*`> -P<`*imprimante*`>`, etc. Notre propos n'étant évidemment pas d'apprendre à utiliser LATEX : se référer aux diverses documentations à votre disposition (notamment en ligne) ! (On peut y préférer les traitements de textes WYSIWYG si on a du temps à perdre et qu'on se soucie peu de typographie, de stabilité, ...)

Toujours sous UNIX, il existe un certain nombre d'outils classiques pour réaliser des figures de type vectoriel : `xfig` pour le dessin à la main, `gnuplot` pour la réalisation de graphiques automatiques, METAPOST pour aller un peu plus loin... Chacun de ces programmes est capable de produire des figures au format postscript encapsulé, type de figures facilement incluables dans un document LATEX (package `epsf`, commande `\epsfbox{`*mafigure.eps*`}` par exemple).

Nos préférences personnelles vont à METAFONT/POST. Ces logiciels consistent avant tout en un langage de programmation interprété qui est largement extensible via un système élaboré de déclarations de commandes et la gestion de différents types de variables (booléennes, numériques, chaînes de caractères, dessins, tableaux multimensionnels de telles variables). Son auteur, Donald Knuth, lui a donné toutes les potentialités d'un langage de programmation standard avec des facilités pour produire des figures. Avec un peu d'effort, on peut faire avec ce langage à peu près tout et n'importe quoi. Sa principale limite native est la limitation des quantités numériques qu'il manipule : virgule fixe, précision standard égale à 1/65536,

partie entière limitée 4095, absence d'indicateurs de dépassement de calcul. Une autre limite semble être une gestion assez rigide de la mémoire qui défend l'utilisation de ce langage pour manipuler de trop grands jeux de données.

Nous n'allons pas demander d'apprendre les subtilités de METAPOST, tout au plus le faire utiliser en adaptant très légèrement des programmes déjà écrits, ou en réalisant des courbes simples.

EXERCICE. — (i) Préparer le canevas du document LaTeX (« projet » où sera fait le rapport des activités pratiques).

(ii) Récupérer les fichiers `statsmac.mp` et `info04-05-fig.mp` (sur une disquette que j'aurais fait circuler...). Le fichier `statsmac.mp` est une extension personnelle de METAPOST destinée au tracé de diagrammes statistiques courants. Son contenu est instable et absolument pas normatif. On peut, doit, ignorer ce qu'il y est écrit.

(iii) Renommer `info04-05-fig.mp`. Ce fichier permet de réaliser les diagrammes qui correspondent à la feuille n° 2, c'est-à-dire un diagramme en bâtons et trois histogrammes correspondant à quatre séries de simulations qui ont été faites. Néanmoins son contenu correspond aux essais que j'ai fait moi-même et pas nécessairement aux siens (choix des paramètres, noms de fichiers de sortie).

(iv) Éditer ce nouveau fichier (avec `emacs`, bien sûr !, qui reconnaît l'extension `.mp`). Adapter les noms des fichiers de sortie qui apparaissent dans `xdatafile(`*fichier-de-sortie*`,xx)` — le `xx` est un paramètre numérique correspondant aux ordonnées les plus hautes rencontrées en effectuant le graphique ultérieur, la commande `xdatafile` n'est qu'un raccourci pour une liste de commandes qui sont répétées plusieurs fois. Adapter les divers paramètres du programme pour correspondre à ses choix de simulation.

(v) Exécuter son fichier METAPOST (`mpost <`*fichier.mp*`>`), débugger.

(vi) Inclure les figures produites dans le document LaTeX. Exécuter le document, le convertir en postscript sans l'imprimer : `dvips <`*document.dvi*`> -o`, le visualiser avec `gv <`*document.ps*`>`. Vérifier la cohérence et la lisibilité de ses figures et éventuellement, revenir au programme METAPOST. Commencer la rédaction du rapport...

Remarque. — Ainsi que certains le pensent parfois, et de temps en temps à voix haute, il est souvent plus rapide d'implémenter des tâches complexes à l'aide de langages simples parce que fréquemment pratiqués que d'avoir à apprendre à se servir de logiciels sophistiqués, lourds, coûteux, à tort prétentieux, etc., dont on ne se servira qu'avec réticence et maladresse. Mon recours à METAPOST participe de ce type d'opinions. Mais ça reste à chacun de trouver ses outils préférés.

2. Se documenter

Nous avons évoqué LaTeX. De très nombreuses références peuvent être trouvées. De la pratique et le « LaTeX reference card » (facilement récupérable sur Internet) suffisent couramment ; des documentations spécifiques à chaque package sont souvent installées sur le système utilisé (`.../texmf/docs/...`). Pour METAPOST, une recherche rapide sur Internet conduit à la bonne référence : John HOBBY, *The* METAPOST *manual* (J. Hobby est la personne qui a porté METAFONT en une version qui produit des figures postscript, METAPOST). La seule lecture complémentaire est le METAFONTbook, second volume de *Computers and Typesetting* de D. Knuth. Néanmoins d'autres outils graphiques peuvent se montrer intéressants.

Par la suite nous n'utiliserons plus le générateur aléatoire mis en place dans la feuille n° 2 mais l'un de ceux qui sont implémentés dans le système utilisé (`rand`, `lrand48`, `random`). Voir les *man pages* associées.

FEUILLE N° 4

1. Générateurs aléatoires usuels sous Unix/Linux

Ce qui suit provient notamment de séries d'informations partielles glanées dans les différentes documentations et programmes C qui auront été trouvés et tentés d'être compris. Nous ne garantissons rien quant à sa totale exactitude ni quant à sa permanence (*voir* Introduction). Cette section n'est que documentaire, on pourra la survoler rapidement.

1.1. Le « vieux » générateur de nombres aléatoires de C

Le générateur de nombres aléatoires de C était basé sur la méthode des congruences linéaires

$$x_n = (a \times x_{n-1} + b) \bmod m, \qquad n \geqslant 1,$$

où, par défaut, le multiplicateur a est égal à 25 214 903 917, l'incrément b égal à 11 et le module $m = 2^{48}$. Ainsi, ces générateurs traitent avec des entiers longs de 48 bits. Les propriétés de tels générateurs sont bien connues (*voir*, comme d'habitude, Knuth (D. E.), *The Art of Computer Programming, volume 2, Seminumerical Algorithms*, third edition, Addison–Wesley (1998) [knuth-acp2], chap. 3), entre autres : leur période est 2^{48} et ils sont uniformément distribués dans $\{0, 1, \ldots, 2^{48} - 1\}$.

`srand48(<`*int*`>)` (précédemment `srand(<`*int*`>)`). Cette fonction initialise la valeur de la graine aléatoire à l'entier $\langle$`<`*int*`>`$\rangle \times 2^{16} + 1370$, ce dernier nombre étant arbitraire.

`lrand48()` (précédemment `rand()`). Cette fonction calcule le nombre pseudo-aléatoire suivant (sur 48 bits) et retourne ses 32 bits de poids le plus élevé, *i.e.*, un entier dans $\{0, 1, \ldots, 2^{32} - 1\}$.

Ces fonctions semblent maintenant dépréciées. Malgré tout, on doit avoir à l'esprit que ce type de générateurs aléatoires est certainement le meilleur parmi ceux dont le comportement est complètement connu. D'autres générateurs tels que le suivant ont, semble-t-il, de bonnes propriétés empiriques mais puisque peu de leurs propriétés mathématiques sont connues, il peut sembler illégitime de les utiliser. D'autre part, de nombreux logiciels professionnels utilisent toujours les suites congruentes linéaires (par exemple Sas, Maple, ...).

1.2. Le générateur aléatoire de METAFONT/POST

L'apport de Donald Knuth à « l'Art de la Programmation des Ordinateurs » est indiscutable. Dans la vie de tous les jours, il y a TEX (souvent à travers son extension LATEX), et jamais très loin — bien qu'on l'ignore fréquemment — METAFONT. Ceci ne représente que l'implication de Knuth dans l'évolution de la typographie assistée par ordinateur. D'autres domaines qu'il a très profondément abordés sont la génération de nombres aléatoires et le tri (entre autres, il suffit de feuilleter les quatre fameux volumes). Dans le langage sous-tendant METAFONT, Knuth a implémenté un générateur aléatoire qui sera décrit ci-après. Les sources de ce langage étant claires et documentées, la description du fonctionnement de ce générateur peut prétendre être exacte. L'utilisateur de METAFONT aura pu trouver anecdotique l'existence d'un tel générateur, ceux de METAPOST auront pu l'apprécier.

Le générateur de nombres aléatoires implémenté dans METAFONT/POST est celui qui est décrit dans la sous-section 3.2.2, p. 27–29, de [knuth-acp2]. Il est basé sur des suites de Fibonacci décalées (*lagged Fibonacci sequence*)

$$x_n = (x_{n-24} + x_{n-55}) \bmod m, \qquad n \geqslant 55,$$

avec $x_0, \ldots, x_1$ donnés. Le module m est égal à 2^{28} dans METAFONT/POST. Alors, la période d'une telle suite est $2^{27}(2^{55} - 1)$ qui est bien plus grande que les périodes qu'on peut obtenir habituellement avec les méthodes de congruences linéaires. Mais ses autres propriétés comme suite de nombres pseudo-aléatoires ne sont pas si bien connues. Les primitives associées à la génération de nombres aléatoires sont :

`randomseed := <`*numeric*`>`. Cette primitive est assez particulière puisque sa syntaxe lui donne l'apparence d'une assignation, même si aucune variable numérique « randomseed » n'est accessible. Son effet est l'initialisation de la ligne $x_0, \ldots, x_{54}$ basée sur la valeur numérique donné en argument. Si aucune telle assignation n'est faite, l'initialisation sera basée sur l'heure et la date. Une première ligne est calculée en tenant compte de la graine et de $x_{164}, \ldots, x_{219}$, les termes de la suite de Fibonacci décalée correspondante, et sont translatés en arrière à $x_0, \ldots, x_{54}$. On n'a pas d'accès direct à la ligne $x_0 \ldots, x_{54}$ ni à aucune de ses entrées.

`uniformdeviate <`*numeric*`>`. Cette fonction primitive retourne un nombre pseudo-aléatoire approximativement distribué uniformément sur l'intervalle réel de bornes 0 et l'argument numérique. Si chacune des entrées de la ligne $x_0, \ldots, x_{54}$ ont été utilisées, les 55 termes suivants de la suite de Fibonacci décalée sont calculés. Le dernier x_i qui n'a pas été encore utilisé sert à calculer le résultat de manière évidente.

`normaldeviate`. Cette fonction primitive sans argument retourne un nombre pseudo-aléatoire choisi par rapport à la distribution normale standard $\mathcal{N}(0, 1)$. Cela procède en une sorte de répétition d'épreuves de Bernoulli et donc appelle et rafraîchit la ligne aussi souvent que nécessaire pour obtenir une valeur acceptable (*voir* [knuth-acp2], p. 130–132).

Remarque. — Le logiciel TEX n'intègre pas de générateur aléatoire natif. Cependant un petit fichier nommé `random.tex` (`.../texmf/tex/generic/misc/random.tex` de Donald Arseneau) fournit un générateur intelligemment programmé basé sur les congruences linéaires.

1.3. Le générateur aléatoire de BSD/gcc

Le générateur de nombres aléatoires de BSD, et puis de `gcc`, sur une méthode assez semblable de celui de METAFONT/POST : les suites de Fibonacci décalées définies par

$$x_n = (x_{n-k} + x_{n-\ell}) \bmod m, \qquad n \geqslant \ell$$

où k, ℓ, m sont des entiers tels que, notamment, $k \leqslant l$. Les paramètres gouvernant ce générateur sont définis par `initstate` ou `setstate`.

`srandom(<`*int*`>)` (identique à `srand(<`*int*`>)`). Cette fonction fait le même travail que `srand48` à ceci près qu'elle calcule aussi les $\ell - 1$ termes suivants de la suite.

`random()` (identique à `rand()`). Cette fonction calcule le nombre pseudo-aléatoire suivant et retourne ses 32 bits de poids le plus élevé qui est un entier dans $\{0, 1, \ldots, 2^{32} - 1\}$.

`initstate(<`*int*`>,<`*char*`>,<`*size_t*`>)`. L'argument `<`*size_t*`>` est la taille de la pile utilisée pour ces suites, i.e., ℓ. Ses valeurs possibles sont 8, 32, 64, 128, 256. L'argument `<`*char*`>` devrait être le nom de la pile, l'argument `<`*int*`>` la valeur initiale. La pile entière est alors remplie selon une méthode compliquée.

`setstate(<`*char*`>)`. La taille de la pile étant déjà définie par `initstate`, on peut réinitialiser la pile avec `setstate`.

Franchement, il est difficile de comprendre ce qu'il y a réellement derrière ces commandes. De plus leur implémentation peut dépendre de la version d'UNIX ou de telles distribution. Nous n'utiliserons que les commandes `random`, `srandom` et la constante `RAND_MAX` — cette dernière constante étant décrite ci-après.

POINT IMPORTANT. — *Couramment en C, les entiers produits par les générateurs aléatoires sont compris entre* 0 *et* `RAND_MAX` -1 *où* `RAND_MAX` *est une constante entière définie par la bibliothèque utilisée.*

Remarque. — On accède aux fonctions externes telles que `rand`, `lrand48`, `random` et celles qui y sont associées via

```
#include <stdlib.h>
```

dans le préambule d'un programme C.

2. Avertissement

La donnée d'un générateur aléatoire correspond à celle d'une suite $(U_n)_{n\geqslant 1}$ de variables aléatoires indépendantes uniformément réparties sur $[0,1]$ et tout ce qu'on peut faire avec (voir travaux dirigés précédents).

Lorsqu'on modélise un phénomène aléatoire où le hasard n'intervient que par une unique séquence de variables aléatoires indépendantes, il n'y a aucun problème à avoir recours au générateur aléatoire à chaque fois qu'on en a besoin en cours de simulation.

En revanche, si le modèle fait intervenir plusieurs suites de variables aléatoires indépendantes, on ne peut pas les générer dans n'importe quel ordre, plus précisément, il est dangereux de faire des appels au générateur en cours de simulation puisque ces appels et les affectations des valeurs correspondantes aux différentes variables pseudo-aléatoires risque de dépendre de ce qui s'est déjà déroulé et donc des variables antérieures. On constate ainsi qu'on risque de perdre l'indépendance supposée des différentes variables. Pour s'assurer de ne pas être confronté à ce probleme, une solution simple est de générer tous les termes des suites pseudo-aléatoires dont on aura besoin dès le début de la simulation.

3. Une modélisation de files d'attentes $M/M/1$

Afin de ne pas perdre trop de temps et d'obtenir des programmes similaires, nous imposons un canevas et des variables.

Le programme C sera nommé `m-m-1.c` et son préambule commencera par :

```
/* m-m-1.c
  Compilation: gcc -lm m-m-1.c
  Exécution: ./a.out */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

La première bibliothèque est celle qui gère les entrées/sorties standard, la seconde donne accès au générateur aléatoire externe `random()` et à la fonction d'initialisation `srandom(<`*int*`>)`, la dernière permet d'accéder au logarithme népérien qui est (injustement) désigné par `log`.

```
/* Paramètres de la file d'attente */

float lambda, mu;
```

Les paramètres `lambda` (λ) et `mu` (μ) sont respectivement l'intensité d'arrivée et l'intensité de service des temps exponentiels correspondants.

```
/* Différentes quantités liées à la file d'attente */

int ni, nj, X, X_, Xmax, N = 100;
float t, t_, s, I[101], J[101];

FILE *fp;
```

La ligne `I[.]` contiendra les dates d'arrivées successives des clients, notamment ses entrées seront croissantes. Au contraire, la ligne `J[.]` contiendra les périodes successives de service, c'est-à-dire la durée occupée par chaque service, ce ne sont donc pas des dates. Les entiers `ni` et `nj` désigneront respectivement la prochaine date d'arrivée et la prochaine durée de service. Il devrait être clair qu'au cours de la simulation on aura toujours `nj <= ni`.

Le nombre flottant `t` représente l'instant présent dans la simulation, les valeurs qu'il prendra se limiteront aux instants où quelque chose se passe (arrivée d'un client/fin d'un service). Le nombre flottant `t_` contiendra la valeur précédente de `t`. L'entier `X` désignera le nombre de clients présents à la date `t` y compris celui qui est en cours de service, l'entier `X_` sera le nombre de clients à la date `t_`. L'entier `Xmax` n'aura qu'un intérêt pratique, celui de contenir la valeur maximale de `X` au cours de la simulation.

Le nombre flottant `s` pourrait représenter la date de la fin du dernier service. En fait c'est la date à partir de laquelle le serveur prend une nouvelle tâche. Ainsi, `s` pourra aussi coïncider avec le dernier instant où la file d'attente est vide (`X == 0`).

L'entier `N` et les valeurs `100` sont égaux au nombre total d'arrivées dans cette simulation (noter qu'alors les lignes `I[.]` et`J[.]` ont pour longueur maximale 101). On pourra par la suite les modifier. Le pointeur `*fp` nous permettra d'exporter le déroulement de la simulation — synthétisé par les différentes valeurs des variables `t`, `X`, `t_`, `X_` — dans un fichier externe sous une forme propre à être utilisée par un logiciel graphique. Les valeurs de `t` et `Xmax` en fin de simulation pourront se montrer utiles pour réaliser des graphiques.

```
/* Le générateur exponentiel */

float Exponential(float lambda)
{
  float x;
  if (lambda > 0)
    {
      x = -log(1-1.0*random()/RAND_MAX)/lambda;
    }
  else
    {
      x = 0; printf("le paramètre lambda doit être strictement positif !\n");
    }
  return x;
}
```

Il n'y a rien à dire sur la fonction précédente puisqu'elle aura déjà été implémentée auparavant. Simplement, nous nous servons ici du générateur aléatoire associé à `random()`. Notons cependant que ce programme sera aisément modifiable pour simuler des files $G/G/1$: il suffira de remplacer les lois exponentielles par d'autres lois.

```
Setup(float lambda_, float mu_, int Xinit)
{
  int i;
```

```
  lambda = lambda_; mu = mu_;
  X = Xinit; Xmax = Xinit + 1; t = 0; s = 0; ni = 1; nj = 1; I[0]=0;
  for (i = 1; i <= N; i++)
    {
      I[i] = Exponential(lambda) + I[i-1];
      J[i] = Exponential(mu);
    }
}
```

La fonction `Setup` doit initialiser la simulation. Il est pratique que certains paramètres soient fixés à l'aide de celle-ci : λ, μ et X_0. Ainsi il sera rapide de les modifier directement dans l'appel de cette fonction dans la fonction `main()`. Il est à noter que les lignes `I[.]` et `J[.]` auront été initialisées convenablement : la première contient les différentes dates d'arrivées séparées par des variables indépendantes de loi exponentielle de paramètre λ, la seconde simplement les périodes de service qui sont des variables indépendantes de loi exponentielle de paramètre μ.

```
NextEvent()
{
.
.
.
}
```

Nous ne précisons pas le contenu de la fonction `NextEvent()` (événement suivant). Plusieurs solutions sont envisageables. Son rôle est le suivant : à la date t, la file a une longueur X_t, et nous cherchons à déterminer ce qu'il se passe ensuite ; il faut donc préalablement stocker les anciennes valeurs de `t` et `X` dans respectivement `t_` et `X_`, déterminer ce qu'il doit se passer, modifier éventuellement `ni`, `nj`, `s`, `t`, `X`, `Xmax` en conséquence.

```
int main()
{
  srandom(1);
  Setup(1, 1, 0);
  fp = fopen("m-m-1.mp", "w");
  fprintf(fp, "lastpath := (%f, %d)\n", t, X);
  while(ni < N)
    {
      NextEvent();
      fprintf(fp, "--(%f,%d)--(%f,%d)\n", t, X_, t, X);
    }
  fprintf(fp, ";\nXmax := %d; tmax := %f;\n", Xmax, t);
  fclose(fp);
  return 0;
}
```

La fonction `main` ne fait que dérouler la simulation et écrire dans un fichier (`m-m-1.mp`) les coordonnées (en gros) du graphe à produire. La condition d'arrêt de la simulation n'est pas optimale, on pourrait l'améliorer.

Il est très important, ici, d'avoir la même initialisation du générateur aléatoire et les mêmes paramètres. En effet, toute la dynamique de la file d'attente est lue par `NextEvent()` dont on a laissé la programmation au lecteur. Avoir des paramètres identiques est une assurance pour débugger éventuellement son programme.

Ce que nous écrivons dans le fichier externe `m-m-1.mp` est la déclaration METAPOST d'un chemin en lignes brisées :

```
lastpath := (...,...)--(...,...)-- ... ---(...,...);
```

et l'indication des coordonnées maximales :

```
Xmax := ...; tmax := ...;
```

L'inclusion de ce fichier dans un programme METAPOST pourra permettre facilement de tracer la courbe d'évolution de la file d'attente. Bien entendu, le recours à d'autres logiciels est envisageable quitte à modifier en conséquence la formulation des `fprintf(fp, "...", ...)` dans la fonction `main()`, ainsi que le nom, ou le suffixe, du fichier par souci de cohérence.

4. **Visualisation des résultats**

(i) À l'aide des données contenues dans le fichier de sortie, produire le graphique retraçant le déroulement de la simulation. Inclure cette figure dans le document LaTeX du rapport et visualiser le résultat.

(ii) Il est fort possible que le graphique obtenu soit différent de graphiques obtenus par d'autres. Si les paramètres utilisés sont identiques, les graphiques doivent être semblables. Dans le cas où on observe des différences, revenir au programme C.

(iii) Faire de nouvelles simulations en faisant principalement varier λ et μ (quitte à produire de nouveaux fichiers de sortie avec de nouveaux noms, et de nouvelles figures). (On pourrait aussi faire varier la graine aléatoire initialisée par `srandom(<`*int*`>)`, mais c'est peut-être de moindre intérêt.)

(iv) Rédiger la section du rapport relative à cette simulation. Commenter ce qu'on peut observer en fonction des valeurs relatives de λ et μ.

FEUILLE N° 5

1. **Quelques propriétés des files d'attentes** $M/M/1$

Nous rappelons que pour une file d'attente de type $M/M/1$ les intervalles de temps séparant les arrivées sont des variables aléatoires de loi exponentielle de paramètre $\lambda > 0$, les durées de service sont des variables aléatoires de loi exponentielle de paramètre $\mu > 0$, et que toutes ces variables aléatoires sont indépendantes. Pour tout $t \geqslant 0$, la variable aléatoire X_t désigne la longueur de la file d'attente au temps t, client en service compris.

Le rapport $\varrho = \lambda/\mu$ est appelé intensié de trafic. On aura pu constater avec quelques simulations qu'à un changement d'échelle dans le temps près, c'est le seul paramètre qui gouverne l'évolution d'une file d'attente $M/M/1$. Un résultat important est le suivant : si $\varrho < 1$ alors la file d'attente admet une et une seule *mesure de probabilité invariante* π, ou *loi stationnaire*, et celle-ci est donnée par :

$$\pi\{n\} = (1-\varrho)\varrho^n, \qquad n \in \mathbb{N}.$$

c'est-à-dire que si X_0 a pour loi π, alors il en est de même pour tout X_t, $t \geqslant 0$.

Un autre résultat est l'ergodicité de la file d'attente lorsque $\varrho < 1$: pour toute condition initiale X_0, on a l'égalité

$$\text{pour tout } n \in \mathbb{N}, \qquad \lim_{T\to\infty} \frac{1}{T} \int_0^T \mathbb{1}_{\{X_t = n\}}\, \mathrm{d}t = \pi\{n\}.$$

c'est-à-dire que la fréquence (qui est aléatoire) du temps passé en n tend vers la mesure de $\{n\}$ pour la probabilité invariante π (qui n'est pas aléatoire). Plus généralement, pour toute fonction $f : \mathbb{N} \to \mathbb{R}$ ne croissant pas trop rapidement (en fait telle que $\sum_{n=0}^{\infty} |f(n)|\, \pi\{n\} < \infty$), en posant

$$\overline{f(X)}_T = \frac{1}{T} \int_0^T f(X_t)\, \mathrm{d}t$$

qui est une variable aléatoire, on a

$$\overline{f(X)} = \lim_{T\to\infty} \overline{f(X)}_T = \lim_{T\to\infty} \frac{1}{T} \int_0^T f(X_t)\, \mathrm{d}t = \sum_{n=0}^{\infty} f(n)\, \pi\{n\}$$

qui est déterministe.

2. **Vérifications expérimentales par simulations**

Après avoir pris garde de stocker les informations relatives aux graphiques tracés antérieurement, modifier le programme `m-m-1.c` afin de calculer les quantités $\bar{X}_T$, et $\overline{P(n)}_T$, pour $0 \leqslant n \leqslant 10$ à chaque nouvel événement arrivant à la file d'attente (arrivée ou fin de service), où

$$\bar{X}_T = \frac{1}{T} \int_0^T X_t\, \mathrm{d}t \qquad \text{et} \qquad \overline{P(n)}_T = \frac{1}{T} \int_0^T \mathbb{1}_{\{X_t = n\}}\, \mathrm{d}t.$$

L'entier N qui mesure la taille de la simulation sera porté à 1000. La taille des tableaux `I[.]` et `J[.]` sera donc portée à 1001. On stockera 1 résultat toutes les 50 arrivées (quand `ni %`

`50 == 1 && X_ <= X` par exemple, où on comprendra le choix de cette condition) dans un fichier externe afin de pouvoir les présentés sous la forme de tableaux dans le rapport. Les lignes de ce fichier pourront s'écrire

$$T \texttt{ \& } \bar{X}_T \texttt{ \& } \overline{P(0)}_T \texttt{ \& } \overline{P(1)}_T \texttt{ \& } \overline{P(2)}_T \texttt{ \& } \ldots \texttt{ \& } \overline{P(8)}_T \texttt{ \& } \overline{P(9)}_T \texttt{ \& } \overline{P(10)}_T \texttt{ \textbackslash\textbackslash}$$

afin de pouvoir être facilement exploitées dans un tableau LaTeX. La dernière ligne du tableau comportera les valeurs limites

$$\texttt{\textbackslash infty \& } \bar{X}_\infty \texttt{ \& } \pi\{0\} \texttt{ \& } \pi\{1\} \texttt{ \& } \pi\{2\} \texttt{ \& } \ldots \texttt{ \& } \pi\{8\} \texttt{ \& } \pi\{9\} \texttt{ \& } \pi\{10\} \texttt{ \textbackslash\textbackslash}$$

lorsque la loi stationnaire existe, sinon les valeurs limites si elles sont définies.

Nota bene. — Pour chaque choix de λ et μ des simulations précédentes, il faudra réaliser un tel tableau et le faire figurer dans le rapport.

Quelques indications. — Notons $T_0 = 0$ et $T_1, T_2, \ldots$ les dates successives où un nouvel événement survient (les valeur de `t` après chaque appel de la fonction `NextEvent()`). Alors pour tout $n \geqslant 1$,

$$T_n \times \overline{f(X)}_{T_n} = \int_0^{T_n} f(X_t)\,\mathrm{d}t = \sum_{k=1}^{n} f(X_{T_{k-1}})(T_k - T_{k-1}) = \overline{f(X)}_{T_{n-1}} + f(X_{T_{n-1}})(T_n - T_{n-1}).$$

puisque X_t est constant égal à $X_{T_{k-1}}$ sur $[T_{k-1}, T_k[$. Aussi après chaque appel de `NextEvent()`, suffira-t-il d'incrémenter les quantités correspondantes à l'aide des valeurs de `X_`, `t_` et `t`.

Les quantités à calculer pourront être définies comme des variables globales de type flottant :

```
float tXbar, tPbar[11];
```

Leur contenu sera de la forme

$$\int_0^{T_n} f(X_t)\,\mathrm{d}t$$

et on exportera des valeurs de la forme

$$\frac{1}{T_n}\int_0^{T_n} f(X_t)\,\mathrm{d}t.$$

avec au plus 3 chiffres derrière la virgule. Le fichier d'exportation pourra être nommé `m-m-1.tex` et on copiera après chaque simulation correcte son contenu dans le fichier LaTeX du rapport avant de modifier les paramètres λ et μ.

FEUILLE N° 6

1. Mise en forme des tableaux de résultats

Pour les tableaux de contenu mathématique ou numérique, LaTeX propose un environnement nommé `array`. Pour les sorties obtenues dans les précédentes simulations, on pourra taper dans son document LaTeX

```
$$
\begin{array}{|r|r|r|r|r|r|r|r|r|r|r|}
\hline
T & \overline X_T & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10\\
\hline
\input m-m-1.tex
\hline
\end{array}
$$
```

Ce qui suit `\begin{array}` est la description de la mise en forme du tableau (13 cases par ligne, alignées à droite, et séparées par des filets verticaux). La commande `\hline` trace un filet horizontal, la commande `&` sépare les cases du tableau et `\\` indique la fin d'une ligne. La commande `\input` <*fichier de sortie*> inclut le fichier de sortie. On pourra, devra, remplacer cette dernière par le contenu même du fichier de sortie si celui-ci s'avère correct.

Remarque. — Si le tableau s'avère trop large pour la largeur de texte de son document, on pourra :

1. modifier la largeur du document (`\textwidth=`<*dimension*> dans le préambule du document LaTeX) ;

2. réduire le corps des caractères dans le tableau (usage de `\small` ou `\footnotesize` juste avant `\begin{array}`) ;

3. diminuer l'écart entre les colonnes du tableau (`\arraycolsep=`<*dimension*> juste avant `\begin{array}`).

2. Implémentation de files d'attente $M/M/1/K$

Le type de files d'attente $M/M/1/K$ où K est un entier strictement positif se formalise de la même manière que le type $M/M/1$ à ceci près que la salle d'attente est limitée à $K-1$ places. Ainsi, il ne peut y avoir au maximum que K clients dans le système (client en service compris). Le processus des arrivées reste inchangé seulement la taille de la file d'attente est incrémenté à une date d'arrivée T si et seulement si $X_T < K$, ce qui correspond au fait qu'un client arrivant dans une salle d'attente pleine soit purement et simplement débouté.

Dans le programme, introduire la variable entière globale `K` parmi les paramètres de la file d'attente. Celui-ci sera éventuellement signé et une valeur négative ou nulle correspondra à une salle d'attente de taille infinie.

Modifier la fonction `Setup` pour y intégrer l'initialisation du paramètre `K`.

Modifier la fonction `NextEvent()` pour la prise en compte de l'éventuelle finitude de la salle d'attente.

3. Quelques propriétés des files d'attentes $M/M/1/K$

Une propriété importante des files d'attentes $M/M/1/K$ est l'existence et l'unicité pour toutes valeurs des paramètres λ, μ et K d'une mesure de probabilité invariante π (loi stationnaire).

Lorsque $\lambda \neq \mu$, c'est-à-dire $\varrho \neq 1$, celle-ci a pour expression

$$\pi\{n\} = \frac{(1-\varrho)\,\varrho^n}{1-\varrho^{K+1}}\,, \qquad n \in \{0,1,\ldots,K\}\,;$$

et si $\lambda = \mu$, c'est-à-dire $\varrho = 1$,

$$\pi\{n\} = \frac{1}{K+1}\,, \qquad n \in \{0,1,\ldots,K\},$$

qui est la mesure uniforme sur $\{0,1,\ldots,K\}$.

L'ergodicité est ici satisfaite aussi pour toutes les valeurs des paramètres.

De la même manière que dans la fiche précédente et pour un choix de K raisonnable ($K \leqslant 10$) présenter sous la forme d'un tableau l'évolution de la moyenne et des fréquences temporelles ainsi que les termes limites correspondant à la loi stationnaire pour des choix de λ et μ tels que respectivement $\lambda < \mu$, $\lambda = \mu$ et $\lambda > \mu$.

Appendice : mise en page correcte du document LaTeX

Les quatre paramètres LaTeX `\paperheight`, `\paperwidth`, `\textheight` et `\textwidth` ont un sens évident. Ce sont fréquemment eux seuls qu'on modifiera. D'autres paramètres de mise en page existent (*voir* plus bas) et ne sont pas toujours simples à régler en conséquence. Nous avons introduit un paramètre `\decalage` pour tenir compte d'une éventuelle reliure.

```
% dimensions choisies par l'utilisateur

\newdimen\decalage

\paperheight=29.7cm \paperwidth=21cm
\textheight=25.7cm \textwidth=17cm
\decalage=0cm

% déduction de \topmargin,
% \evensidemargin et \oddsidemargin

\oddsidemargin=\paperwidth
\advance\oddsidemargin by -\textwidth
\divide\oddsidemargin by 2
\advance\oddsidemargin by -1 in
\evensidemargin=\oddsidemargin
\advance\oddsidemargin by \decalage
\advance\evensidemargin by -\decalage

\topmargin=\paperheight
\advance\topmargin by -\headheight
\advance\topmargin by -\headsep
\advance\topmargin by -\textheight
\advance\topmargin by -\footskip
\divide\topmargin by 2
\advance\topmargin by -1 in
```

Université de Poitiers
Département de Mathématiques

CORRECTION DE LA FEUILLE N° 1

EXERCICE 1. — Les réponses que nous donnons ici sont cohérentes avec ce qui est développé dans l'exercice suivant. Ce ne sont pas les seules réponses possibles ; en particulier, ce ne sont pas toujours les plus immédiates.

(i) $X(\omega) = 0$ si $U_1(\omega) \leqslant 1-p$, 1 sinon. (La réponse la plus immédiate serait de poser $X(\omega) = 1$ si $U_1(\omega) \leqslant p$, 0 sinon.)

(ii) $X(\omega) = x_1$ si $U_1(\omega) \in [0, p_1]$ et $X(\omega) = x_n$ si $U_1(\omega) \in]p_1 + \cdots + p_{n-1}, p_1 + \cdots + p_n]$ pour $n \geqslant 2$. (La réponse la plus immédiate serait de poser $X(\omega) = x_n$ si $U_1(\omega) \in [p_1 + \cdots + p_{n-1}, p_1 + \cdots + p_n[$, la probabilité que U_1 soit égal à 1 étant nulle et que, pratiquement, une telle valeur ne devrait pas apparaître.)

(iii) $X = X_1 + \cdots + X_n$ où X_i est défini comme à la question (i) à l'aide de U_i. Cette réponse repose sur le fait que la somme de n variables aléatoires indépendantes de loi $\mathcal{B}(1,p)$ est de loi $\mathcal{B}(n,p)$.

(iv) $X = (b-a)U_1 + a$. La vérification de la justesse de cette réponse est immédiate. De plus elle devrait être intuitive.

(v) $X = -\ln(1-U_1)/\lambda$: le vérifier avec la fonction de répartition. (En général un début de réponse commence par regarder $\ln U_1$, puis — comme il y a un problème de signe évident — $-\ln U_1$; des petits calculs montrent qu'il faut une constante multiplicative pour obtenir le bon paramètre, ce qui conduit à regarder $-\ln U_1/\lambda$; la réponse que nous donnons est cohérente avec cette dernière mais tient compte de la direction prise par l'approche plus générale.)

(vi) $X = \Phi^{-1}(U_1)$: le vérifier avec la fonction de répartition.

(vii) $X = \sigma\Phi^{-1}(U_1) + \mu$: le vérifier avec la fonction de répartition. Ceci vient d'un fait classique concernant les lois gaussiennes : Si X est de loi $\mathcal{N}(\mu, \sigma^2)$, alors $Z = (X-\mu)/\sigma$ est de loi $\mathcal{N}(0,1)$ et réciproquement. Cette propriété ne repose que sur un changement de variables linéaire dans les intégrales correspondantes.

EXERCICE 2. — C'est très formel. À considérer plus comme un complément de cours ou culturel qu'un simple exercice.

(i) D'après les hypothèses de cette question, la restriction $F_{]a,b[}$ est un homéomorphisme sur l'intervalle $]0,1[$. Nous noterons $F^{-1} :]0,1[\to]a,b[$ son inverse qui est une fonction continue. Comme la probabilité que U_1 soit égal à 0 ou 1 est nulle, la variable aléatoire $X = F^{-1}(U_1)$ est bien définie et est à valeurs dans $]a,b[$. Soit $x \in \mathbb{R}$. Si $x \leqslant a$, $\mathbb{P}\{X \leqslant x\} = 0$, si $x \geqslant b$, $\mathbb{P}\{X \leqslant x\} = 1$, si $x \in]a,b[$, $\mathbb{P}\{X \leqslant x\} = \mathbb{P}\{F^{-1}(U_1) \in]a,x]\} = \mathbb{P}\{U_1 \in]0, F(x)]\} = F(x)$. La fonction de répartition de X étant égale à F, la loi de X est la mesure de probabilité π.

Remarque. — À titre de complément, que faire si U prend les valeurs 0 ou 1 ? Formellement, cela n'a pas d'importance puisque ces événements ont une probabilité égale à 0. Pratiquement, on pourra faire correspondre à $u = 0$, la valeur $F^{-1}(0) = a$, et à $u = 1$, la valeur $F^{-1}(1) = b$. Néanmoins, dans la formalisation de la question suivante, il conviendrait de poser $F^{-1}(0) = -\infty$ et $F^{-1}(1) = b$.

(ii) Soient F la fonction de répartition d'une loi de probabilité π sur $\mathbb{R}$,

$$F^{-1}(u) = \inf\{y \in \mathbb{R} : u \leqslant F(y)\} \qquad \text{pour tout } u \in [0,1],$$

U une variable aléatoire de loi uniforme sur $[0,1]$ et $X = F^{-1} \circ U$.

Nous souhaitons montrer que nous pour tout $x \in \mathbb{R}$, les événements $\{U \leqslant F(x)\}$ et $\{X \leqslant x\} = \{F^{-1} \circ U \leqslant x\}$. Pour cela il suffit de montrer que pour tous $x \in \mathbb{R}$, $u \in [0,1]$, on a l'équivalence $u \leqslant F(x) \iff F^{-1}(u) \leqslant x$.

Supposons $u \leqslant F(x)$. Alors $x \in \{y \in \mathbb{R} : u \leqslant F(y)\}$, et ainsi $F^{-1}(u) = \inf\{y \in \mathbb{R} : u \leqslant F(y)\} \leqslant x$, ce qui montre la première implication. Pour la réciproque, si on a $F^{-1}(u) \leqslant x$, cela veut dire que pour tout $\varepsilon > 0$, il existe $x_\varepsilon \in \{y \in \mathbb{R} : u \leqslant F(y)\}$ tel que $x_\varepsilon \leqslant x + \varepsilon$; alors $u \leqslant F(x_\varepsilon)$ et puisque F est croissante $F(x_\varepsilon) \leqslant F(x+\varepsilon)$, dont on retient simplement que $u \leqslant F(x+\varepsilon)$ pour tout $\varepsilon > 0$; en laissant tendre $\varepsilon > 0$ vers 0 et puisque F est continue à droite, on obtient $u \leqslant F(x)$, ce qui montre l'implication réciproque.

Pour conclure, il suffit de calculer la fonction de répartition de X : pour $x \in \mathbb{R}$, $F_X(x) = \mathbb{P}\{X \leqslant x\} = \mathbb{P}\{U \leqslant F(x)\} = F(x)$, ce qui montre que la loi de X est la loi π.

Remarques. — *a*) D'un point de vue pratique, la fonction de répartition F de la loi π est simplement une fonction croissante continue à droite, limitée à gauche et valant 0 en $-\infty$ et 1 en $+\infty$. Si on trace une telle fonction, la détermination de $x = F^{-1}(u)$ pour $u \in [0,1]$ consiste à placer u sur l'axe des ordonnées et y faire correspondre x sur l'axe des abscisses à l'aide du graphe de F de la façon suivante : si u est en face d'un saut de F, prendre le x où a lieu ce saut ; si u se trouve en face d'un palier du graphe de F, prendre l'extrémité inférieure (gauche) de ce palier ; sinon faire correspondre l'unique x tel que $u = F(x)$.

b) Ce calcul d'inverse généralisé correspond à ce qu'on appelle le calcul des quantiles d'une loi de probabilité : $x = F^{-1}(u)$ est le quantile d'ordre u de la loi π. Il y a cependant plusieurs déterminations possibles de quantiles, indétermination qui repose sur l'existence éventuelle de paliers de la fonction de répartition. Un choix courant est de prendre, lorsque le palier est d'amplitude finie, son point milieu, sinon son extrémité finie lorsque son amplitude est infinie. Pour la loi de $X = F^{-1}(U)$ cela ne change rien car les paliers n'étant qu'en nombre dénombrable, la probabilité que U se trouve en face de l'un d'entre eux est nulle.

Exercice 3. — Soient X, Y deux variables aléatoires réelles indépendantes et de même loi $\mathcal{N}(0,1)$; le couple (X,Y) a donc pour loi la loi produit de celles de X et de Y. On définit les variables aléatoires (R,Θ) comme étant les coordonnées polaires de (X,Y). Elles sont bien définies et à valeurs dans $\mathbb{R}_+ \times [0,2\pi[$ car l'événement $\{(X,Y) = (0,0)\} = \{R = 0\}$ est de probabilité nulle (c'est la situation où l'angle polaire est indéterminé).

(i) Soient $I \in \mathcal{B}or(\mathbb{R}_+)$, $J \in \mathcal{B}or([0,2\pi[)$ — I et J peuvent être limités aux intervalles de $\mathbb{R}_+$ et $[0,2\pi[$ respectivement —, et $C = \{(r\cos\theta, r\sin\theta) : r \in I,\ \theta \in J\}$. Alors,

$$\begin{aligned}\mathbb{P}\{R \in I,\ \Theta \in J\} = \mathbb{P}\{(X,Y) \in C\} &= \int_C \mathrm{e}^{-x^2/2}\,\mathrm{e}^{-y^2/2}\,\frac{\mathrm{d}x\,\mathrm{d}y}{2\pi} \\ &= \int_I \int_J \mathrm{e}^{-r^2/2}\,\frac{r\,\mathrm{d}\theta\,\mathrm{d}r}{2\pi} = \left(\int_I \mathrm{e}^{-r^2/2}\,r\,\mathrm{d}r\right) \times \left(\int_J \frac{\mathrm{d}\theta}{2\pi}\right).\end{aligned}$$

En particulier, en prenant $J = [0,2\pi[$, puis $I = \mathbb{R}_+$, on en déduit les lois respectives de R et de Θ :

$$\mathbb{P}\{R \in I\} = \mathbb{P}\{R \in I,\ \Theta \in [0,2\pi[\} = \int_I \mathrm{e}^{-r^2/2}\,r\,\mathrm{d}r$$

et

$$\mathbb{P}\{\Theta \in J\} = \mathbb{P}\{R \in \mathbb{R}_+,\ \Theta \in J\} = \int_J \frac{\mathrm{d}\theta}{2\pi},$$

et, ainsi, la loi de R admet la densité $r \mapsto \mathbb{1}_{\mathbb{R}_+}(r)\,r\exp(-r^2/2)$ et la loi de Θ est la loi uniforme sur $[0,2\pi[$. On constate aussi que pour tous $I \in \mathcal{B}or(\mathbb{R}_+)$ et $J \in \mathcal{B}or([0,2\pi[)$,

$$\mathbb{P}\{R \in I,\ \Theta \in J\} = \mathbb{P}\{R \in I\} \times \mathbb{P}\{\Theta \in J\},$$

ce qui signifie que R et Θ sont indépendantes.

Précisons que la variable aléatoire R^2 à valeurs dans $\mathbb{R}_+$ est de loi exponentielle de paramètre $1/2$. Pour le voir, calculons sa fonction de répartition : soit $t \geqslant 0$, alors

$$F_{R^2}(t) = \mathbb{P}\{R^2 \leqslant t\} = \mathbb{P}\{R \leqslant \sqrt{t}\} = \int_0^{\sqrt{t}} \mathrm{e}^{-r^2/2} r \, \mathrm{d}r = \left[-\mathrm{e}^{-r^2/2}\right]_0^{\sqrt{t}} = 1 - \mathrm{e}^{-t/2},$$

ce qui est bien la fonction de répartition de la loi exponentielle de paramètre $1/2$.

(ii) On part de deux variables aléatoires (U_1, U_2) indépendantes et uniformément distribuées sur $[0, 1]$. Puis on pose $R = \sqrt{-2\ln(U_1)}$ — car R^2 a pour loi la loi exponentielle de paramètre $1/2$, ce que nous avons pu vérifier précédemment — et $\Theta = 2\pi U_2$; les variables (R, Θ) sont indépendantes et sont les coordonnées polaires d'un point aléatoire (X_1, X_2) du plan ($X_1 = R\cos\Theta$ et $X_1 = R\sin\Theta$). On vient de voir que les variables (X_1, X_2) sont indépendantes et toutes deux de distribution $\mathcal{N}(0, 1)$. Ainsi, avec deux appels successifs du générateur aléatoire pour obtenir (u_1, u_2), quelques calculs élémentaires

$$x_1 = \sqrt{-2\ln(u_1)}\cos(2\pi u_2) \qquad \text{et} \qquad x_2 = \sqrt{-2\ln(u_1)}\sin(2\pi u_2),$$

on obtient (x_1, x_2) deux nombres aléatoires tirés indépendamment suivant la distribution $\mathcal{N}(0, 1)$.

CORRECTION DE LA FEUILLE N° 2

1. **Un générateur aléatoire**

Le choix de $b = 0$ pour cette méthode de congruences linéaires impose qu'on exclut la valeur 0 pour la graine aléatoire. Plus généralement, on doit exclure ici tout multiple de m sans quoi la suite obtenue stationnera à 0.

```
#include <stdio.h>
#include <stdlib.h>
/* #include <math.h> */

unsigned int myCurrentRandom = 1;
int myRandmax = 2143483647, a =16807, b=0;

mySetRandom(int seed){
  unsigned int seed_ = seed % myRandmax;
  if (seed_ == 0)
    {myCurrentRandom = 1;
    printf("Mauvaise graine remplacée par 1.\n");}
  else
    {myCurrentRandom = seed_;}
}

unsigned int myRandom(){
  myCurrentRandom=(a*myCurrentRandom+b) % myRandmax;
  /* % est l'opérateur modulo */
  return myCurrentRandom;
}
```

À la suite, nous testons le générateur aléatoire sur ses dix premiers termes.

```
main(){
  int i;
  mySetRandom(1);
  for (i = 1; i < 11; i = i+1)
    {
      printf("nombre aléatoire %d : %d\n",i,myRandom());
    }
  return 0;
}
```

Dans ce programme, on notera l'usage du type `unsigned` qui est nécessaire compte tenu qu'il arrive que les opérations faites sur les termes de la suite conduisent à mettre à 1 le bit qui indique si l'entier considéré est négatif ou positif.

En initialisant la graine à 1, on obtient pour les 10 premiers termes de la suite :

$$16807,\quad 282475249,\quad 1622647863,\quad 951787490,\quad 90650479,$$
$$1000694122,\quad 1725660967,\quad 1421206130,\quad 1898293854,\quad 1607729490.$$

À voir comme ça, ...

2. Lois usuelles

Nous poursuivons le programme précédent en loi ajoutant les fonctions destinées à simuler les lois usuelles.

```
/* info04-05.c
  Compilation: gcc -lm info04-05.c */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

unsigned int myCurrentRandom = 1;
int myRandmax = 2143483647, a = 16807, b = 0;
FILE *fp;

mySetRandom(int seed){
  unsigned int seed_ = seed % myRandmax;
  if (seed_ == 0)
    {myCurrentRandom = 1;
    printf("Mauvaise graine remplacée par 1.\n");}
  else
    {myCurrentRandom = seed_;}
}

unsigned int myRandom(){
  myCurrentRandom = (a*myCurrentRandom+b) % myRandmax;
  /* % est l'opérateur modulo */
  return myCurrentRandom;
}

int myBinomial(int n, float p){
  int i, x = 0;
  float u;
  for (i = 1; i <= n; i = i+1)
    {
      u = 1.0*myRandom()/myRandmax;
      if (u <= p)
  {
    x = x+1;
  }
    }
  return x;
}

float myUniform(float a, float b){
  float x;
  x = a+(b-a)*myRandom()/myRandmax;
  return x;
}

/* La bibliothèque <math.h> est nécessaire ici, log est le logarithme népérien. */
```

```
float myExponential(float lambda){
  float x;
  if (lambda > 0)
    {
      x = -log(1-1.0*myRandom()/myRandmax)/lambda;
    }
  else
  {
    x = 0;
    printf("le paramètre lambda doit être strictement positif !\n");
  }
  return x;
}

float myGaussian(float mu, float sigma){
  float x;
  if (sigma > 0)
    {
      /* Ici, on utilise 1-u plutot que u pour s'assurer
   de ne pas avoir un argument nul pour log. */
      x = sigma*sqrt(-2*log(1-1.0*myRandom()/myRandmax))
  *cos(6.283185307*myRandom()/myRandmax)+mu;
    }
  else
  {
    x = 0;
    printf("le paramètre sigma doit être strictement positif !\n");
  }
  return x;
}
```

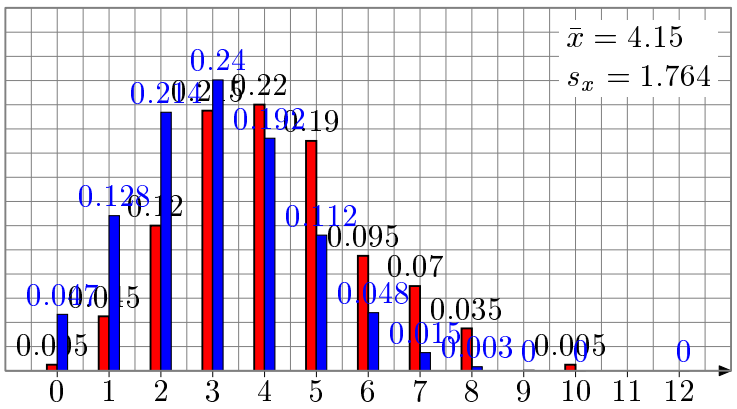
Nous adaptons la fonction `main` à notre propos, c'est-à-dire tester les différentes fonctions précédentes sur les listes de 100 nombres tirés « au hasard ».

```
main(){
  int i, imax = 100;
  mySetRandom(1);
  fp=fopen("myBinomial.dat", "w");
  for (i = 1; i <= imax; i = i+1)
    {
      fprintf(fp,"%d\n",myBinomial(12,0.333333));
    }
  fclose(fp);
  mySetRandom(1);
  fp=fopen("myUniform.dat", "w");
  for (i = 1; i <= imax; i = i+1)
    {
      fprintf(fp,"%f\n",myUniform(0,20));
    }
  fclose(fp);
  mySetRandom(1);
  fp=fopen("myExponential.dat", "w");
```

```
  for (i = 1; i <= imax; i = i+1)
    {
      fprintf(fp,"%f\n",myExponential(1));
    }
  fclose(fp);
  mySetRandom(1);
  fp=fopen("myGaussian.dat", "w");
  for (i = 1; i <= imax; i = i+1)
    {
      fprintf(fp,"%f\n",myGaussian(0,1));
    }
  fclose(fp);
  return 0;
}
```

3. **Comparaisons**

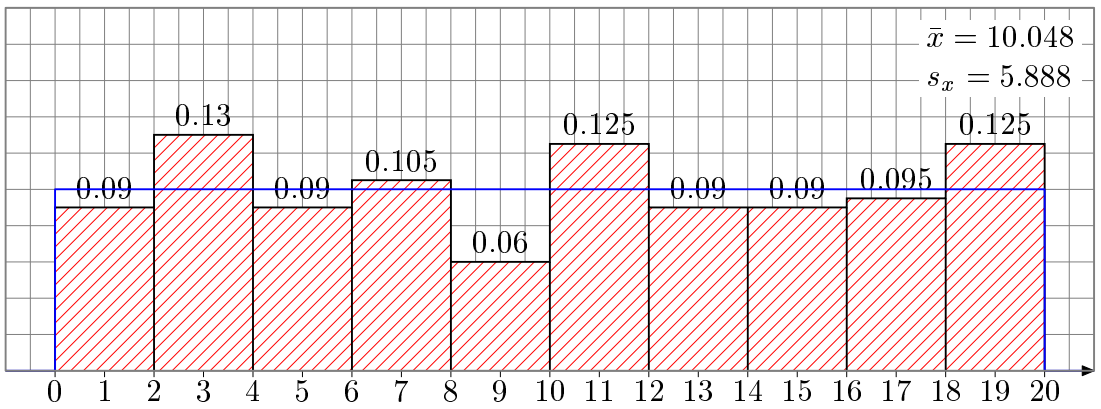
Dans la simulation suivante, nous avons tiré 100 nombres « au hasard » suivant la loi binomiale $\mathcal{B}(12, 1/3)$ — le choix de cette loi correspondant à un exercice classique de Statistique (expérience de Weldon). En rouge est tracé le diagramme en bâtons de la distribution observée, en bleu, celui de la distribution théorique. Rappelons que la loi $\mathcal{B}(12, 1/3)$ a pour moyenne $12 \times 1/3 = 4$, pour variance $12 \times 1/3 \times (1 - 1/3) = 8/3 \approx 2{,}666$ et pour écart-type, la racine carrée de la variance, approximativement 1,633.



Nous obtenons, ainsi que l'indique la figure ci-dessus des résultats assez concordants. Notons que l'usage de la loi des grands nombres — pourvu que le générateur aléatoire soit idéal — montre que la distribution observée tend vers la distribution théorique lorque la taille de l'échantillon tend vers l'infini. La convergence est néanmoins assez lente et la taille d'échantillon à laquelle nous nous sommes limité est relativement petite (100).

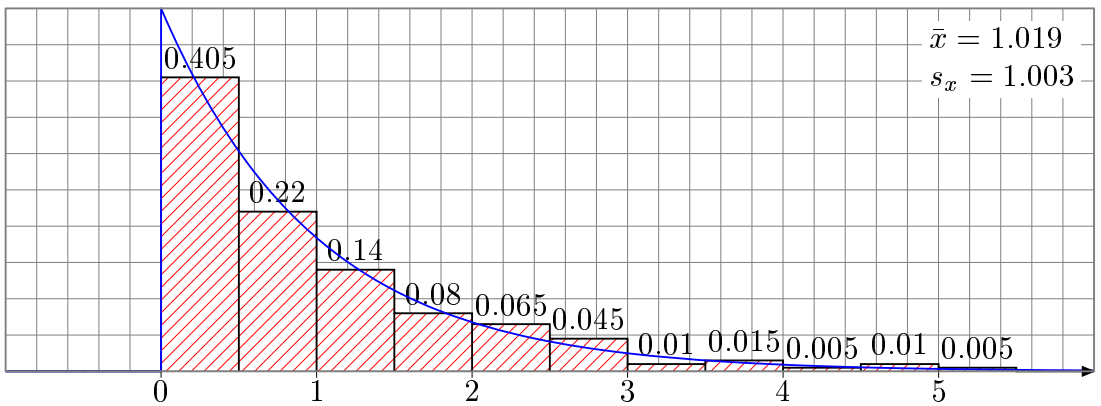
Dans la simulation suivante, nous avons tiré 100 nombres « au hasard » suivant la loi uniforme $\mathcal{U}(0, 20)$ — ce qui doit être assez évocateur pour un étudiant. En rouge est tracé un histogramme de la distribution observée, en bleu, le graphe de la densité de la distribution théorique. Rappelons que la loi $\mathcal{U}(a, b)$ a pour moyenne $(a+b)/2$ (ici égale à 10), pour variance $(b-a)^2/12$ (ici approximativement égale à 33,333, et pour écart-type, ici, approximativement

5,774).



Ce que nous obtenons par la simulation semble assez cohérent avec la distribution théorique. Notamment les moyennes et écart-types sont proches.

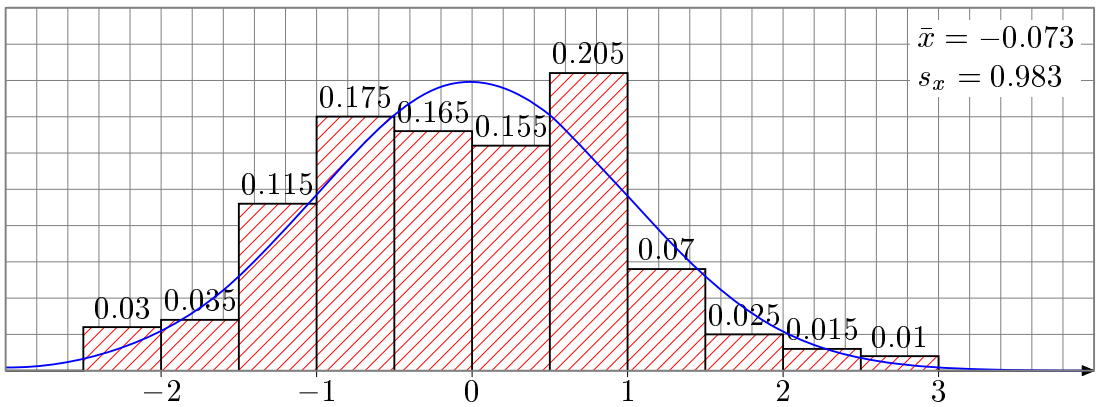
Dans la simulation suivante, nous avons tiré 100 nombres « au hasard » suivant la loi exponentielle $\mathcal{E}(1)$ — type de lois courant pour les files d'attente. En rouge est tracé un histogramme de la distribution observée, en bleu, le graphe de la densité de la distribution théorique. Rappelons que la loi $\mathcal{E}(\lambda)$ a pour moyenne $1/\lambda$ (ici égale à 1), pour variance $1/\lambda^2$ (ici égale à 1, et pour écart-type, ici, égal à 1).



Ici, tout semble concorder avec une bonne approximation. Il ne faut pas néanmoins se réjouir trop vite. Dans les simulations antérieures nous nous sommes servi exactement de la même séquence de nombres aléatoires. La forme même de la distribution assure une simulation aisée ou convaincante.

Dans la simulation suivante, nous avons tiré 100 nombres « au hasard » suivant la loi normale $\mathcal{N}(0,1)$ — type de lois courant dans les sciences expérimentales ou humaines. En rouge est tracé un histogramme de la distribution observée, en bleu, le graphe de la densité de la distribution théorique. Rappelons que la loi $\mathcal{N}(\mu,\sigma)$ a pour moyenne μ (ici égale à 0),

pour variance σ^2 (ici égale à 1, et pour écart-type σ, ici, égal à 1).



Ce que nous obtenons n'est pas très convaincant, mais ce n'est guère pire que pour le cas d'une loi uniforme.

Université de Poitiers
Département de Mathématiques

CORRECTION DE LA FEUILLE N° 3

Rien à dire.

CORRECTION DE LA FEUILLE N° 4

3. **Une modélisation de files d'attentes** $M/M/1$

Dans le programme demandé il suffisait de compléter la définition de la fonction `NextEvent()`. Nous avons écrit :

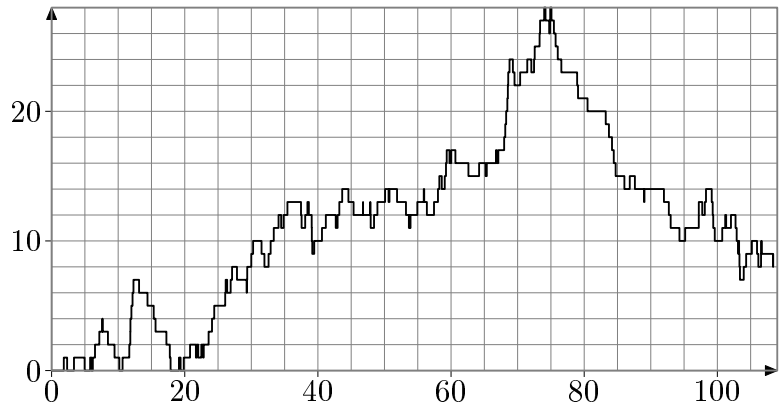
```
NextEvent()
{
  t_ = t; X_ = X;
  if (X == 0 || I[ni] < s + J[nj])
    {
      t = I[ni]; if (X == 0) s = t; X++; ni++;
      if (X > Xmax) Xmax = X;
    }
  else
    {
      s = s + J[nj]; t = s; X--; nj++;
    }
}
```

La première ligne sauvegarde les valeurs précédentes de `X` et de `t` pour un usage ultérieur. Pour comprendre ce qu'il se passe, supposons tout d'abord `X==0` : la première condition est donc satisfaite, le temps courant est maintenant la prochaine arrivée d'un client, puisque la file était vide le serveur prend en charge ce nouveau client dès la date de son arrivée, la taille de la file passe à 1 et le numéro d'arrivée suivante est incrémenté. Supposons maintenant que `I[ni] < s + J[nj]` et `X > 0`, ce qui signifie que le prochain client arrive avant que le service en cours ne soit achevé ; alors on se place à cette date d'arrivée, la longueur de la file croît de 1 et le numéro d'arrivée du client suivant est incrémenté de 1. Si maintenant Supposons maintenant que `I[ni] >= s + J[nj]` et `X > 0`, ce qui signifie que le service en cours s'achève avant ou en même temps que l'arrivée du prochain client ; dans ce cas on se place à la date de fin du service en cours, le serveur commence un nouveau service (si la file est non vide), la taille de la file décrît de 1 et le numéro de service est incrémenté. Si dans ce dernier cas a date de la prochaine arrivée coïncide avec la date de fin du service en cours, l'arrivée restera prise en compte au moins par un appel ultérieur de la fonction `NextEvent()`.

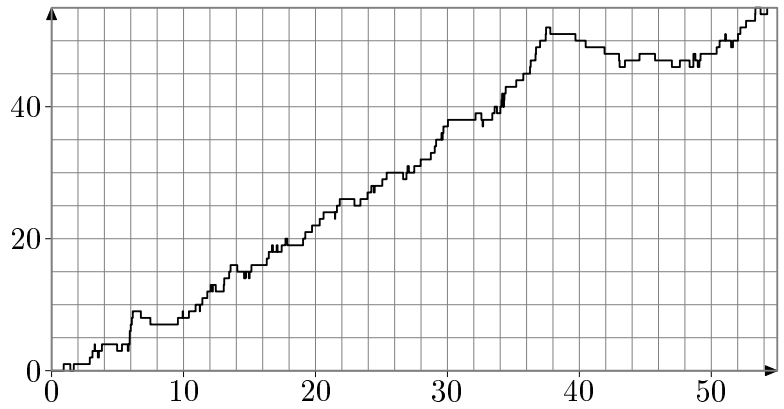
4. **Visualisation des résultats**

(i) La figure suivante montre le résultat (correct espérons-le) de la simulation demandée. La comparaison des figures obtenues avec celle-ci devrait indiquer d'éventuelles erreurs de

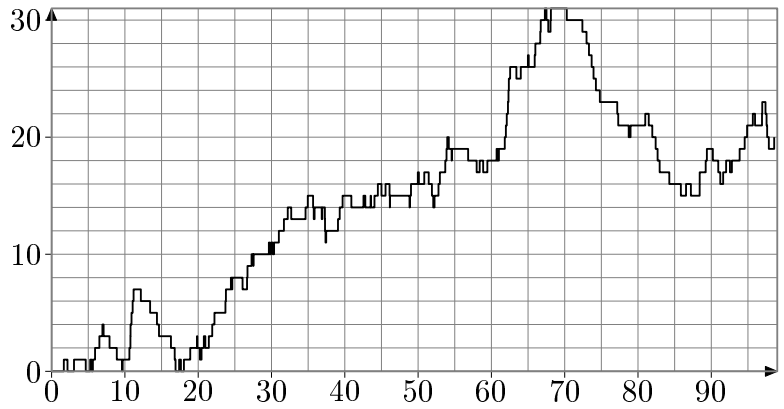
programmation.



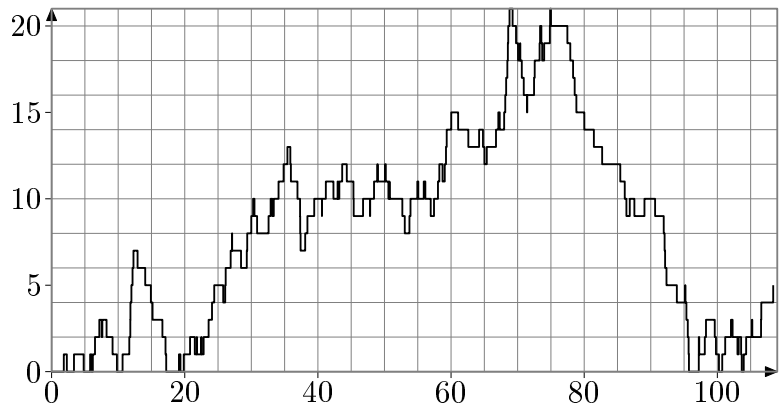
(iii) La figure suivante a été obtenue pour $\lambda = 2$ et $\mu = 1$.



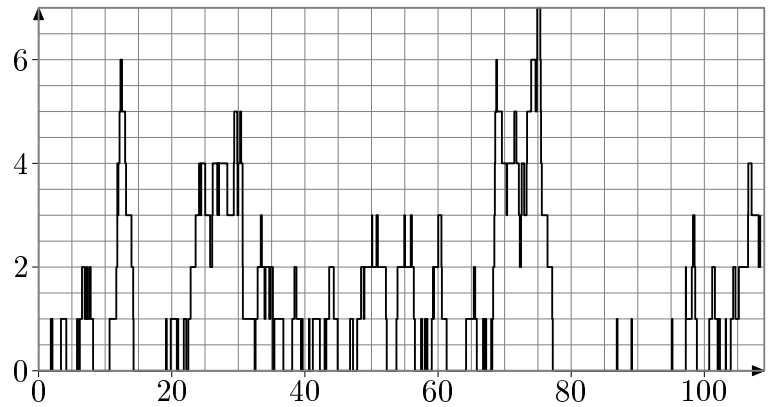
La figure suivante a été obtenue pour $\lambda = 1{,}1$ et $\mu = 1$.



La figure suivante a été obtenue pour $\lambda = 1$ et $\mu = 1{,}1$.



La figure suivante a été obtenue pour $\lambda = 1$ et $\mu = 2$.



(iv) On observe ce qu'on doit observer intuivement.

CORRECTION DE LA FEUILLE N° 5

Ici, on a pris $\lambda = \mu = 1$. La file d'attente est récurrente nulle ce qui signifie que quelque soit la condition initiale toutes les valeurs possibles de la taille de la file d'attente sont atteintes (presque sûrement), mais qu'en revanche il n'existe pas de mesure de probabilité invariante. Avec la simulation, on se rend compte que toutes les valeurs possibles sont asymptotiquement fréquentées avec une même fréquence. Comme l'ensemble des valeurs possibles est infini, cette fréquence tend donc vers 0. Bien qu'elle ne soit pas définie, l'espérance sous la loi stationnaire peut être prise infinie : c'est la limite (presque sûre) des moyennes temporelles.

T	X_T	0	1	2	3	4	5	6	7	8	9	10
48.960	6.175	0.124	0.130	0.090	0.070	0.016	0.056	0.041	0.057	0.040	0.027	0.061
108.435	11.357	0.056	0.059	0.041	0.032	0.007	0.025	0.018	0.031	0.026	0.039	0.056
168.893	10.277	0.062	0.056	0.045	0.032	0.013	0.020	0.040	0.064	0.048	0.055	0.044
221.022	8.351	0.101	0.080	0.086	0.067	0.044	0.027	0.035	0.050	0.037	0.042	0.034
257.685	8.089	0.090	0.078	0.093	0.074	0.046	0.029	0.037	0.053	0.045	0.045	0.039
314.617	8.338	0.074	0.064	0.076	0.061	0.037	0.029	0.055	0.058	0.066	0.052	0.060
363.088	9.263	0.064	0.055	0.066	0.053	0.032	0.025	0.048	0.050	0.057	0.050	0.057
410.235	9.830	0.056	0.049	0.058	0.046	0.029	0.022	0.042	0.044	0.050	0.044	0.058
460.827	9.901	0.050	0.043	0.052	0.044	0.028	0.025	0.042	0.045	0.052	0.048	0.070
508.893	9.316	0.056	0.045	0.059	0.051	0.040	0.046	0.049	0.045	0.050	0.044	0.064
553.783	9.605	0.051	0.041	0.054	0.047	0.036	0.042	0.045	0.042	0.051	0.047	0.065
592.405	10.621	0.048	0.039	0.051	0.044	0.034	0.039	0.042	0.040	0.048	0.044	0.061
647.074	11.904	0.044	0.035	0.047	0.040	0.031	0.036	0.039	0.036	0.044	0.040	0.055
696.237	12.498	0.041	0.033	0.043	0.038	0.029	0.033	0.036	0.034	0.042	0.039	0.054
750.309	12.222	0.040	0.031	0.041	0.039	0.029	0.034	0.036	0.034	0.045	0.043	0.067
800.147	11.740	0.039	0.033	0.045	0.045	0.039	0.045	0.037	0.041	0.047	0.040	0.063
860.570	11.351	0.036	0.031	0.045	0.048	0.042	0.049	0.044	0.056	0.058	0.040	0.062
900.928	11.357	0.034	0.030	0.043	0.046	0.041	0.047	0.043	0.055	0.060	0.040	0.068
942.983	12.294	0.033	0.029	0.041	0.044	0.039	0.045	0.041	0.052	0.058	0.039	0.065
995.059	13.612	0.031	0.027	0.039	0.042	0.037	0.043	0.039	0.050	0.055	0.037	0.062
∞	∞	0	0	0	0	0	0	0	0	0	0	0

Nous considérons maintenant $\lambda = 1$ et $\mu = 2$. La file d'attente est alors récurrente strictement positive : toutes les valeurs possibles sont atteintes presque sûrement, mais il existe une unique mesure de probabilité invariante (loi stationnaire). Rappelons que $\mathbb{E}^{\pi}[X_t] = \varrho/(1-\varrho)$ quand $\varrho < 1$ et π est la loi stationnaire ($\pi\{n\} = (1-\varrho)\varrho^n$, $n \in \mathbb{N}$). L'ergodicité, c'est-à-dire la convergence presque sûre des distrubtions temporelles vers la loi stationnaire est illustrée par

cette simulation de manière assez frappante.

T	X_T	0	1	2	3	4	5	6	7	8	9	10
48.960	1.205	0.420	0.284	0.111	0.075	0.080	0.025	0.005	0.000	0.000	0.000	0.000
108.435	1.199	0.480	0.196	0.142	0.070	0.066	0.031	0.010	0.004	0.000	0.000	0.000
168.893	0.957	0.525	0.229	0.121	0.053	0.042	0.020	0.006	0.003	0.000	0.000	0.000
221.022	0.836	0.550	0.245	0.109	0.042	0.032	0.015	0.005	0.002	0.000	0.000	0.000
257.685	0.840	0.528	0.267	0.119	0.039	0.029	0.013	0.004	0.002	0.000	0.000	0.000
314.617	0.841	0.516	0.277	0.124	0.039	0.027	0.012	0.003	0.001	0.000	0.000	0.000
363.088	0.866	0.511	0.268	0.129	0.048	0.029	0.010	0.003	0.001	0.000	0.000	0.000
410.235	0.844	0.516	0.273	0.122	0.050	0.027	0.009	0.003	0.001	0.000	0.000	0.000
460.827	0.815	0.521	0.277	0.120	0.046	0.024	0.008	0.002	0.001	0.000	0.000	0.000
508.893	0.821	0.522	0.270	0.122	0.053	0.024	0.007	0.002	0.001	0.000	0.000	0.000
553.783	0.851	0.513	0.270	0.122	0.058	0.027	0.008	0.002	0.001	0.000	0.000	0.000
592.405	1.007	0.499	0.261	0.117	0.059	0.029	0.013	0.010	0.007	0.006	0.000	0.000
647.074	0.966	0.510	0.260	0.115	0.055	0.027	0.012	0.009	0.006	0.005	0.000	0.000
696.237	0.962	0.516	0.256	0.110	0.057	0.027	0.015	0.009	0.006	0.005	0.000	0.000
750.309	0.957	0.517	0.253	0.109	0.061	0.028	0.014	0.008	0.005	0.004	0.000	0.000
800.147	0.951	0.517	0.256	0.109	0.059	0.026	0.015	0.008	0.005	0.004	0.000	0.000
860.570	0.936	0.518	0.256	0.114	0.057	0.025	0.014	0.008	0.005	0.004	0.000	0.000
900.928	0.984	0.504	0.260	0.116	0.057	0.028	0.018	0.010	0.005	0.004	0.000	0.000
942.983	0.989	0.498	0.261	0.121	0.058	0.027	0.017	0.009	0.005	0.004	0.000	0.000
995.059	0.998	0.499	0.259	0.122	0.056	0.026	0.017	0.009	0.006	0.004	0.001	0.000
∞	1.000	0.500	0.250	0.125	0.062	0.031	0.016	0.008	0.004	0.002	0.001	0.000

Nous considérons maintenant $\lambda = 2$ et $\mu = 1$. La file d'attente est alors transiente et il est clair que la taille tend presque sûrement vers l'infini (les durées de service ne parviennent pas à compenser la fréquence des arrivées). Il n'existe pas de mesure de probabilité invariante sur $\mathbb{N}$. En fait, la mesure invariante est la masse unité en ∞. L'espérance sous cette loi est donc tout naturellement infinie. La simulation est parfaitement illustrative de tout cela.

T	X_T	0	1	2	3	4	5	6	7	8	9	10
24.480	12.624	0.048	0.070	0.012	0.043	0.069	0.000	0.003	0.089	0.065	0.048	0.020
54.217	29.758	0.022	0.032	0.005	0.019	0.031	0.000	0.001	0.040	0.029	0.022	0.009
84.446	43.635	0.014	0.020	0.003	0.012	0.020	0.000	0.001	0.026	0.019	0.014	0.006
110.511	55.196	0.011	0.016	0.003	0.009	0.015	0.000	0.001	0.020	0.014	0.011	0.004
128.843	65.082	0.009	0.013	0.002	0.008	0.013	0.000	0.001	0.017	0.012	0.009	0.004
157.308	79.964	0.007	0.011	0.002	0.007	0.011	0.000	0.000	0.014	0.010	0.007	0.003
181.544	91.622	0.006	0.009	0.002	0.006	0.009	0.000	0.000	0.012	0.009	0.006	0.003
205.117	102.551	0.006	0.008	0.001	0.005	0.008	0.000	0.000	0.011	0.008	0.006	0.002
230.414	113.950	0.005	0.007	0.001	0.005	0.007	0.000	0.000	0.009	0.007	0.005	0.002
254.446	124.805	0.005	0.007	0.001	0.004	0.007	0.000	0.000	0.009	0.006	0.005	0.002
276.891	136.141	0.004	0.006	0.001	0.004	0.006	0.000	0.000	0.008	0.006	0.004	0.002
296.202	146.214	0.004	0.006	0.001	0.004	0.006	0.000	0.000	0.007	0.005	0.004	0.002
323.537	160.693	0.004	0.005	0.001	0.003	0.005	0.000	0.000	0.007	0.005	0.004	0.002
348.118	173.856	0.003	0.005	0.001	0.003	0.005	0.000	0.000	0.006	0.005	0.003	0.001
375.154	188.004	0.003	0.005	0.001	0.003	0.005	0.000	0.000	0.006	0.004	0.003	0.001
400.073	200.447	0.003	0.004	0.001	0.003	0.004	0.000	0.000	0.005	0.004	0.003	0.001
430.285	215.303	0.003	0.004	0.001	0.002	0.004	0.000	0.000	0.005	0.004	0.003	0.001
450.464	224.910	0.003	0.004	0.001	0.002	0.004	0.000	0.000	0.005	0.004	0.003	0.001
471.492	235.320	0.003	0.004	0.001	0.002	0.004	0.000	0.000	0.005	0.003	0.002	0.001
497.530	248.110	0.002	0.003	0.001	0.002	0.003	0.000	0.000	0.004	0.003	0.002	0.001
∞	∞	0	0	0	0	0	0	0	0	0	0	0

CORRECTION DE LA FEUILLE N° 6

1. Mise en forme des tableaux de résultats

Il n'y a rien à dire. Simplement, le format d'exportation est suffisamment général pour être intégré dans différents environnements LaTeX, ou de simples tableaux TeX par exemple.

2. Implémentation de files d'attente $M/M/1/K$

On rajoute la variable globale `K`.

```
/* Paramètres de la file d'attente */

float lambda, mu, rho;
int K;
```

On modifie la fonction `Setup`

```
Setup(float lambda_, float mu_, float K_, int Xinit)
{
  int i;
  lambda = lambda_; mu = mu_; rho = lambda/mu; K = K_;
  X = Xinit; Xmax = Xinit + 1; t = 0; s = 0; ni = 1; nj = 1; I[0]=0;
  for (i = 1; i <= N; i++)
    {
      I[i] = Exponential(lambda) + I[i-1];
      J[i] = Exponential(mu);
    }
}
```

où on note qu'on peut toujours essayer d'étendre la simulation à des files $G/G/1/K$. Puis on modifie la fonction `NextEvent()`

```
NextEvent()
{
  t_ = t; X_ = X;
  if (X == 0 || I[ni] < s+J[nj])
    {
      t = I[ni]; if (X == 0) s = t; ni++;
      if (K <= 0 || X < K)
  {
    X++;
    if (X > Xmax) Xmax = X;
  }
    }
  else
    {
      s = s + J[nj]; t = s; X--; nj++;
```

```
    }
}
```

Il faudra prendre garde dans la fonction `main()` à rajouter le paramètre manquant à `Setup` avant d'effectuer le moindre essai.

3. **Quelques propriétés des files d'attente** $M/M/1/K$

Soit K un entier strictement positif. Les files d'attente $M/M/1/K$ ont alors toujours une unique loi stationnaire. Rappelons que lorsque $\lambda \neq \mu$, c'est-à-dire $\varrho \neq 1$, celle-ci a pour expression

$$\pi\{n\} = \frac{(1-\varrho)\varrho^n}{1-\varrho^{K+1}}\,, \qquad n \in \{0,1,\ldots,K\},$$

et que son espérance (moyenne) est

$$\sum_{n=0}^{K} n \times \pi\{n\} = \frac{\varrho \times \big(K\varrho^{K+1} - (K+1)\varrho^K + 1\big)}{(1-\varrho)(1-\varrho^{K+1})}\,;$$

et si $\lambda = \mu$, c'est-à-dire $\varrho = 1$,

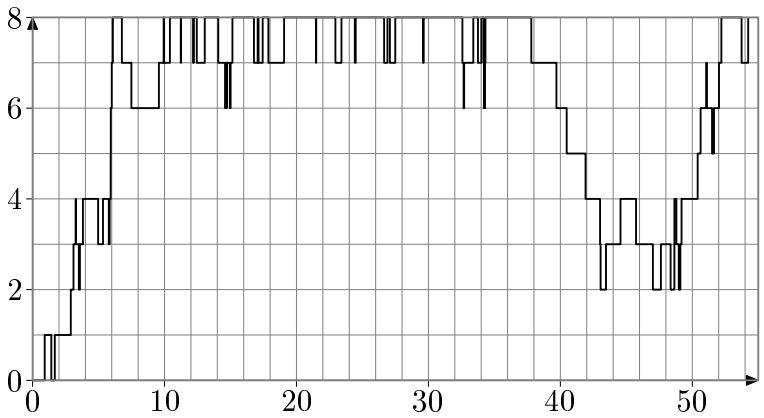
$$\pi\{n\} = \frac{1}{K+1}\,, \qquad n \in \{0,1,\ldots,K\},$$

qui est la mesure uniforme sur $\{0,1,\ldots,K\}$ et dont l'espérance est

$$\sum_{n=0}^{K} n \times \pi\{n\} = \frac{K}{2}\,.$$

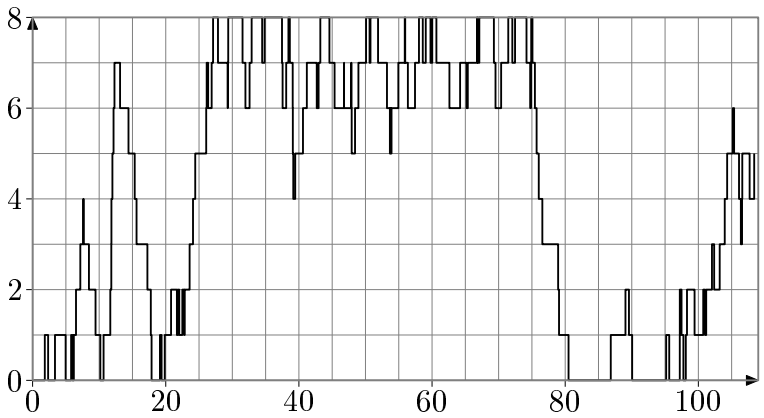
On notera que lorsque $\varrho < 1$ et K tend vers l'infini, on retrouve les expressions déjà rencontrées pour les files d'attente $M/M/1$.

Ici on a $\lambda = 2$, $\mu = 1$, $K = 8$.



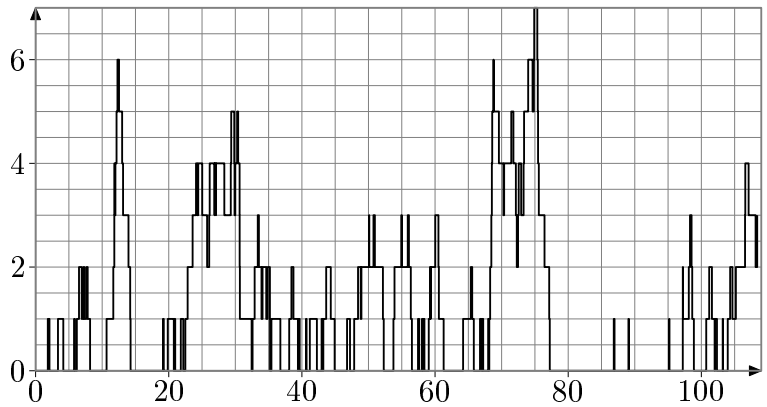
T	X_T	0	1	2	3	4	5	6	7	8
24.480	6.149	0.048	0.070	0.012	0.043	0.069	0.000	0.094	0.227	0.437
54.217	6.163	0.022	0.032	0.032	0.082	0.098	0.033	0.081	0.184	0.435
84.446	6.598	0.014	0.020	0.020	0.053	0.063	0.037	0.094	0.210	0.489
110.511	6.746	0.011	0.016	0.015	0.040	0.050	0.047	0.106	0.214	0.501
128.843	6.885	0.009	0.013	0.013	0.035	0.044	0.042	0.093	0.207	0.543
157.308	6.803	0.007	0.011	0.011	0.036	0.049	0.056	0.113	0.227	0.490
181.544	6.885	0.006	0.009	0.009	0.031	0.042	0.048	0.110	0.248	0.494
205.117	6.882	0.006	0.008	0.008	0.028	0.040	0.053	0.124	0.258	0.474
230.414	6.862	0.005	0.007	0.007	0.030	0.041	0.059	0.127	0.256	0.467
254.446	6.928	0.005	0.007	0.007	0.027	0.037	0.056	0.123	0.252	0.487
276.891	6.989	0.004	0.006	0.006	0.025	0.034	0.052	0.115	0.252	0.506
296.202	7.015	0.004	0.006	0.006	0.023	0.033	0.051	0.112	0.251	0.514
323.537	7.039	0.004	0.005	0.005	0.021	0.031	0.051	0.115	0.250	0.518
348.118	7.002	0.003	0.005	0.005	0.022	0.039	0.056	0.114	0.246	0.510
375.154	6.989	0.003	0.005	0.006	0.022	0.039	0.057	0.113	0.250	0.504
400.073	6.984	0.003	0.004	0.006	0.023	0.040	0.057	0.120	0.244	0.504
430.285	6.969	0.003	0.004	0.006	0.024	0.040	0.056	0.127	0.244	0.496
450.464	6.984	0.003	0.004	0.006	0.023	0.039	0.055	0.124	0.246	0.500
471.492	6.969	0.003	0.004	0.006	0.025	0.041	0.056	0.123	0.248	0.496
497.530	6.931	0.003	0.004	0.007	0.027	0.044	0.057	0.120	0.245	0.492
∞	7.018	0.002	0.004	0.008	0.016	0.031	0.063	0.125	0.250	0.501

Ici on a $\lambda = 1$, $\mu = 1$, $K = 8$.



T	X_T	0	1	2	3	4	5	6	7	8
48.960	4.401	0.124	0.130	0.090	0.070	0.022	0.092	0.145	0.150	0.177
108.435	4.194	0.180	0.119	0.078	0.065	0.028	0.071	0.119	0.184	0.158
168.893	4.107	0.172	0.122	0.086	0.068	0.044	0.095	0.092	0.165	0.156
221.022	3.803	0.190	0.122	0.098	0.076	0.052	0.097	0.103	0.141	0.119
257.685	3.904	0.166	0.107	0.094	0.081	0.090	0.111	0.111	0.135	0.105
314.617	3.847	0.157	0.129	0.103	0.082	0.078	0.101	0.104	0.144	0.102
363.088	4.163	0.136	0.112	0.090	0.074	0.075	0.115	0.127	0.159	0.113
410.235	4.213	0.127	0.111	0.092	0.071	0.075	0.120	0.137	0.154	0.114
460.827	4.067	0.126	0.126	0.107	0.071	0.078	0.111	0.130	0.142	0.108
508.893	3.902	0.126	0.134	0.118	0.081	0.087	0.107	0.119	0.129	0.098
553.783	3.890	0.122	0.131	0.121	0.086	0.090	0.108	0.126	0.124	0.092
592.405	4.008	0.115	0.127	0.118	0.083	0.089	0.107	0.126	0.129	0.105
647.074	4.114	0.106	0.123	0.113	0.081	0.088	0.109	0.137	0.138	0.105
696.237	4.028	0.115	0.126	0.113	0.081	0.090	0.110	0.131	0.133	0.101
750.309	4.037	0.115	0.120	0.112	0.082	0.095	0.110	0.137	0.133	0.095
800.147	3.919	0.123	0.130	0.114	0.083	0.093	0.109	0.130	0.126	0.091
860.570	3.830	0.126	0.135	0.124	0.084	0.093	0.106	0.125	0.119	0.089
900.928	3.852	0.120	0.131	0.123	0.090	0.098	0.108	0.123	0.117	0.089
942.983	3.974	0.115	0.126	0.118	0.086	0.097	0.109	0.124	0.130	0.097
995.059	4.056	0.109	0.122	0.114	0.085	0.097	0.107	0.129	0.138	0.098
∞	4.000	0.111	0.111	0.111	0.111	0.111	0.111	0.111	0.111	0.111

Ici on a $\lambda = 1$, $\mu = 2$, $K = 8$.



T	X_T	0	1	2	3	4	5	6	7	8
48.960	1.205	0.420	0.284	0.111	0.075	0.080	0.025	0.005	0.000	0.000
108.435	1.199	0.480	0.196	0.142	0.070	0.066	0.031	0.010	0.004	0.000
168.893	0.957	0.525	0.229	0.121	0.053	0.042	0.020	0.006	0.003	0.000
221.022	0.836	0.550	0.245	0.109	0.042	0.032	0.015	0.005	0.002	0.000
257.685	0.840	0.528	0.267	0.119	0.039	0.029	0.013	0.004	0.002	0.000
314.617	0.841	0.516	0.277	0.124	0.039	0.027	0.012	0.003	0.001	0.000
363.088	0.866	0.511	0.268	0.129	0.048	0.029	0.010	0.003	0.001	0.000
410.235	0.844	0.516	0.273	0.122	0.050	0.027	0.009	0.003	0.001	0.000
460.827	0.815	0.521	0.277	0.120	0.046	0.024	0.008	0.002	0.001	0.000
508.893	0.821	0.522	0.270	0.122	0.053	0.024	0.007	0.002	0.001	0.000
553.783	0.851	0.513	0.270	0.122	0.058	0.027	0.008	0.002	0.001	0.000
592.405	0.988	0.499	0.260	0.119	0.060	0.029	0.016	0.005	0.009	0.003
647.074	0.950	0.510	0.259	0.115	0.059	0.026	0.015	0.004	0.008	0.002
696.237	0.944	0.517	0.253	0.115	0.058	0.029	0.015	0.004	0.008	0.002
750.309	0.938	0.517	0.250	0.118	0.059	0.028	0.014	0.004	0.007	0.002
800.147	0.935	0.517	0.250	0.120	0.057	0.028	0.015	0.004	0.007	0.002
860.570	0.911	0.518	0.258	0.117	0.055	0.026	0.014	0.004	0.006	0.002
900.928	0.981	0.504	0.259	0.119	0.055	0.027	0.019	0.007	0.008	0.002
942.983	0.983	0.498	0.263	0.121	0.055	0.027	0.018	0.007	0.008	0.002
995.059	0.991	0.500	0.261	0.120	0.056	0.026	0.019	0.007	0.008	0.003
∞	0.982	0.501	0.250	0.125	0.063	0.031	0.016	0.008	0.004	0.002

Le programme qui suit a été remis en forme pour pouvoir facilement exporter les résultats de simulation. On notera que l'initialisation de l'exportation, l'exportation elle-même et la clôture ont été placées dans des fonctions particulières ce qui rend plus facile le passage de l'exportation de graphique à celui de la table.

```
/* m-m-1.c
  Compilation: gcc -lm m-m-1.c
  Exécution: ./a.out */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* Paramètres de la file d'attente */

float lambda, mu, rho;
int K;

/* Différentes quantités liées à la file d'attente */

int ni, nj, X, X_, Xmax, N=1000, TableMax=10;
float t, t_, s, I[1001], J[1001];

FILE *fp;

/* Vérification de l'ergodicité */

float p, tXbar, tPbar[11];

/* Le générateur exponentiel */

float Exponential(float lambda)
{
  float x;
  if (lambda > 0)
    {
      x = -log(1-1.0*random()/RAND_MAX)/lambda;
    }
  else
    {
      x = 0; printf("le paramètre lambda doit être strictement positif !\n");
    }
  return x;
}

Setup(float lambda_, float mu_, float K_, int Xinit)
{
  int i;
  lambda = lambda_; mu = mu_; rho = lambda/mu; K = K_;
  X = Xinit; Xmax = Xinit + 1; t = 0; s = 0; ni = 1; nj = 1; I[0]=0;
  for (i = 1; i <= N; i++)
    {
      I[i] = Exponential(lambda) + I[i-1];
      J[i] = Exponential(mu);
    }
}
```

```
NextEvent()
{
  t_ = t; X_ = X;
  if (X == 0 || I[ni] < s+J[nj])
    {
      t = I[ni]; if (X == 0) s = t; ni++;
      if (K <= 0 || X < K)
  {
    X++;
    if (X > Xmax) Xmax = X;
  }
    }
  else
    {
      s = s + J[nj]; t = s; X--; nj++;
    }
}

GraphSetup(char *name)
{
  fp = fopen(name, "w");
  fprintf(fp, "%%\\lambda = %.3f, \\mu = %.3f, K = %d\n", lambda, mu, K);
  fprintf(fp, "lastpath := (%f, %d)\n", t, X);
}

GraphUpdate()
{
  fprintf(fp, "--(%f,%d)--(%f,%d)\n", t, X_, t, X);
}

GraphClose()
{
  fprintf(fp, ";\nXmax := %d; tmax := %f;\n", Xmax, t);
  fclose(fp);
}

TableSetup(char *name)
{
  int i;
  fp = fopen(name, "w");
  fprintf(fp, "%%\\lambda = %.3f, \\mu = %.3f, K = %d\n", lambda, mu, K);
  fprintf(fp, "T&\\widebar X_T");
  tXbar = 0;
  if (K > 0 && K < TableMax){TableMax = K;}
  for (i = 0; i <= TableMax; i++)
    {
      tPbar[i] = 0;
      fprintf(fp, "& %d ", i);
    }
  fprintf(fp, "\\\\\n");
  fprintf(fp, "\\noalign{\\hrule}\n");
}
```

```
TableUpdate()
{
  int i;
  tXbar = tXbar + X_*(t-t_);
  if (X_ <= TableMax){tPbar[X_] = tPbar[X_] + (t - t_);}
  if (ni % 50 == 0 && X_ <= X)
    {
      fprintf(fp, "%.3f & %.3f ", t, tXbar/t);
      for (i = 0; i <= TableMax; i++){fprintf(fp, "& %.3f ",tPbar[i]/t);}
      fprintf(fp, "\\\\\n");
    }
}

TableClose()
{
  int i;
  fprintf(fp, "\\noalign{\\hrule}\n");
  if (K > 0)
    {
      if (rho == 1)
  {
    fprintf(fp, "\\infty & %.3f ", 1.0*K/2);
    p = 1.0/(K + 1);
    for (i = 0; i <= TableMax; i++)
      {
        fprintf(fp, "& %.3f ", p);
      }
  }
      else
  {
    fprintf(fp, "\\infty & %.3f ",
      rho/((1-rho)*(1-pow(rho,K+1)))
      *(K*pow(rho, K+1) - (K+1)*pow(rho, K) + 1));
    p=(1 - rho)/(1 - pow(rho, K + 1));
    for (i = 0; i <= TableMax; i++)
      {
        fprintf(fp, "& %.3f ", p);
        p = p*rho;
      }
  }
    }
  else
    {
      if (rho < 1)
  {
    fprintf(fp, "\\infty & %.3f ", rho/(1-rho));
    p=(1-rho);
    for (i = 0; i <= TableMax; i++)
      {
        fprintf(fp, "& %.3f ", p);
      p = p*rho;
```

```
      }
  }
    else
      {
  /* if (rho > 1){fprintf(fp, "\\infty & \\infty ");}
     else{fprintf(fp, "\\infty & \\hbox{non d\\'efini} ");} */
  fprintf(fp, "\\infty & \\infty ");
  for (i = 0; i <= TableMax; i++){fprintf(fp, "& 0 ");}
      }
  }
  fprintf(fp, "\\\\\n");
  fclose(fp);
}

main()
{
  srandom(1);
  N = 100;
  Setup(2, 1, 8, 0);
  GraphSetup("graph.mp");
  while(ni < N)
    {
      NextEvent();
      GraphUpdate();
    }
  GraphClose();
  return 0;
}
```