TRAVAUX PRATIQUES DE PROBABILITÉS, GÉNÉRATEURS ALÉATOIRES ET SIMULATIONS DE VARIABLES ALÉATOIRES

1. **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 $\{0, 1/m, \dots, (m-1)/m\}$ 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 $\{0, 1, \dots, m-1\}$.
- L'exigence de reproductibilité impose que la suite $(x_n)_{n\geqslant 1}$ à valeurs dans $\{0,1,\dots,m-1\}$, 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 $\phi : \{0,1,\dots,m-1\}^k \to \{0,1,\dots,m-1\}$, une donnée initiale $(x_1,\dots,x_k)$, et on calcule de manière itérative $x_{n+1} = \phi(x_n,\dots,x_{n-k+1})$. Il est à noter que la suite obtenue évolue dans un un ensemble fini (précisément $\{0,1,\dots,m-1\}^k$) 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 premier de Mersenne[1] (on démontre qu'alors p est nécessairement premier). 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 car la valeur 0, puisque $b = 0$, poserait clairement problème pour une telle suite — les propriétés qu'on peut attendre d'un tel générateur. Remarquons que $m = 2^{31} - 1$ est le plus grand nombre premier de Mersenne qui reste dans le domaine de calcul en entiers d'un processeur 32 bits.

Pour Maple, il semblerait que le générateur aléatoire standard soit basé sur la méthode des congruences linéaires avec

$$m = 10^{12} - 11, \qquad a = 427\,419\,669\,081, \qquad b = 0.$$

Hélas !, il est toujours très fréquent que l'implémentation des générateurs aléatoires dans des logiciels commerciaux ou non soit insuffisamment documentée, ce qui donne une impression de « boîte noire » incompatible avec le calcul scientifique. Il semblerait que la commande

```
rand()
```

retourne un entier entre 0 et $m - 1 = 10^{12} - 12$ déterminé par congruence linéaire. Une variable entière nommée `_seed` contient la dernière valeur calculée et fixe donc les valeurs suivantes (affecter 0 comme valeur à `_seed` ne donne pas des résultats très intéressants). La commande `rand()` peut avoir un argument non vide : se reporter à la documentation en ligne (`?rand;`).

Exercice 1. — (i) Définir une fonction Maple basée sur les explications précédentes :

1. Marin Mersenne (1588–1648), moine français.

```
> m := 10^12-11; a := 427419669081; b := 0;
> myseed := 1;
> myrand := proc() global a, b, m, myseed;
  myseed := modp(a*myseed+b, m);
  return myseed;
  end proc;
```

Comparer son comportement avec la fonction `rand()` (il faut bien sûr initialiser `_seed` et `myseed` avec les mêmes valeurs et comparer les résultats d'exécutions successives de `rand();` `myrand();`). Nous a-t-on menti ?

(ii) Vérifier que $m = 10^{12} - 11$ est premier (`isprime(m);`). Est-ce un nombre premier de Mersenne ? (Utiliser la fonction `log[2](.)` pour voir si $m + 1$ est de la forme 2^n.) Sinon, quelle justification simple pourrait-on trouver pour un tel choix ?

Remarques. — *a*) Dans la définition de la procédure `myrand()` les quantités `a`, `b`, `m` et `myseed` ont une valeur globale : ce sont les valeurs préalablement connues de `a`, `b`, `m` et `myseed` qui sont utilisées et la nouvelle valeur de `myseed` est prise en compte pour la suite.

b) Lorsqu'on sait qu'un générateur aléatoire est basé sur la méthode des congruences linéaires, il n'est pas difficile de trouver la valeur de b : il suffit d'initialiser la graine à 0 et de faire un appel au générateur ($(a \times 0 + b) \bmod m = b \bmod m = b$). Puis on peut retrouver la valeur de a en initialisant la graine à 1 (en supposant $a + b \leqslant m$). Pour m, on peut toujours faire une longue liste de nombres premier de Mersenne, puis faire tourner le générateur un certain nombre de fois afin d'identifier la ou les bornes supérieures plausibles, et enfin faire quelques essais pour s'assurer d'avoir trouvé le bon module m.

2. **Simulations de variables aléatoires**

Les trois exercices qui suivent ne font pas l'objet d'une utilisation de l'ordinateur. Ils ont été déjà abordés en travaux dirigés et ne sont là qu'à titre de mémoire.

Exercice 2 (rappel). — 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 $\{x_1, \ldots, x_n\}$ de probabilités respectives $p_1, \ldots, p_n$;

(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 3 (rappel). — 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 4 (rappel). — 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.

Exercice 5. — Illustrer numériquement les exercices de rappel précédents en se servant des fonctions Maple `rand()`. Il esra conseillé de charger le module `stats` à l'aide de la commande `with(stats)` et de se reporter à l'aide en ligne pour trouver des moyens d'obtenir des représentation graphiques (diagrammes en bâtons, histogrammes).

3. **Autour des nombres normaux**

Si $b \geqslant 2$ est un entier, un nombre x est normal en base b si les fréquences d'apparition des chiffres $\{0,1,\dots,b-1\}$ dans son écriture en base b sont (asymptotiquement) égales ; x est normal ou absolument normal s'il est normal dans toute base $b \geqslant 2$. Puisque la normalité ne dépend que du comportement asymptotique de la partie fractionnaire, la notion peut être prolongée à tout $x \in \mathbb{R}$. Il en est de même du résultat de Borel : les nombres normaux sont de mesure de Lebesgue pleine dans $\mathbb{R}$.

On sait exhiber des nombres normaux dans une base donnée. Par exemple,

$$0.010101010101\ldots = \sum_{n=1}^{\infty} \frac{1}{4^n} = \frac{1}{3}$$

est normal en base 2 ; de même

$$0.012345678901234567890123456789\ldots = \sum_{n=1}^{\infty} \frac{123\,456\,789}{10^{10n}} = \frac{123\,456\,789}{10^{10}-123\,456\,789}$$

est normal en base 10. Ces deux exemples sont rationnels et ne sont clairement pas normaux en base 3 pour le premier et en base $10^{10}-123456789$ pour le second. D'ailleurs, pour tout rationnel x donné, il y a toujours une base dans laquelle x n'est pas normal.

Il a été démontré que le nombre (de Champernowne)

$$0.123456789101112131415\ldots = 0.1\text{-}2\text{-}3\text{-}4\text{-}5\text{-}6\text{-}7\text{-}8\text{-}9\text{-}10\text{-}11\text{-}12\text{-}13\text{-}14\text{-}15\ldots$$

est normal base 10 et transcendant (et donc irrationnel), mais on ne sait pas à jour s'il est absolument normal. Il y a de nombreux nombres non rationnels usuels dont on ignore s'ils sont normaux, même dans une base donnée : $\sqrt{2}$, $\sqrt{3}$, … π, e, $\ln 2$, …

Pour faire quelques tests de l'éventuelle normalité de certains nombres, nous proposons la procédure Maple suivante :

```
> restart;
> normality := proc(x, b, n) local fx, i, j, frequency;
  Digits := ceil(n*ln(b)/ln(10));
  for i from 0 to b-1 do frequency[i] := 0.0 end do;
  fx := frac(evalf(x));
  for i from 1 to n do
  fx := b*fx;
  j := floor(fx);
  frequency[j] := frequency[j]+1;
  fx := frac(fx);
  end do;
```

```
Digits := 10;
for i from 0 to b-1 do print(i, frequency[i]/n) end do;
end;
```

Cela définit la commande

`normality(x, b, n)`

dont les trois arguments sont respectivement le nombre dont on veut regarder le développement, la base, le nombre de chiffres à prendre en compte.

Exercice 6. — Saisir la définition de la procédure précédente (`<shift+entrée>` permet d'obtenir un retour de ligne sans interprétation de la ligne de commande). Puis exécuter la procédure `normality` pour différents nombres « intéressants » en faisant varier le nombre de décimales à prendre en compte ainsi que la base de développement. Notez vos impressions ou commentaires.

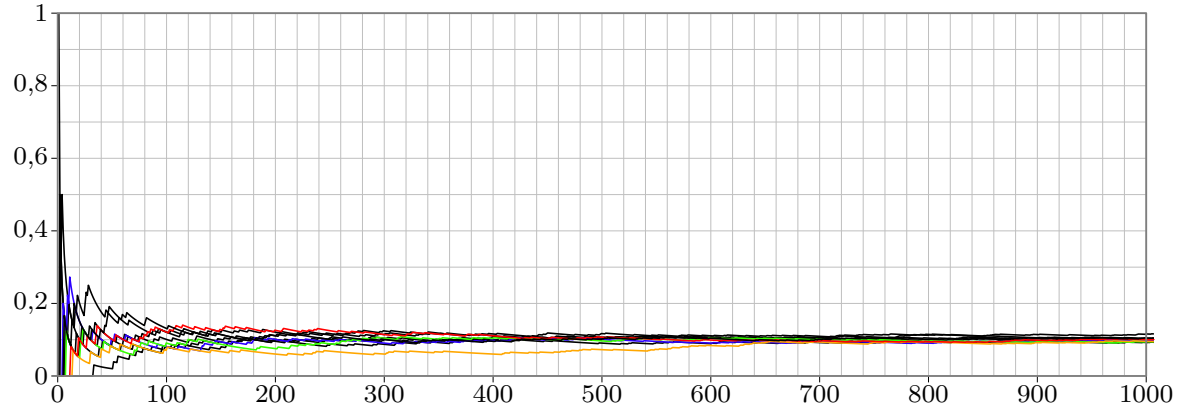
Remarques. — *a*) La normalité ne repose que sur les fréquences d'apparition des nombres dans le développement, mais peut-être pas sur l'imprédictibilité de leur suite (il semble qu'en 2002 un nombre absolument normal, « calculable » — et donc dont les suites des termes de ses développements soient prévisibles — ait été découvert). Le sujet reste assez mystérieux et est à cheval entre la Théorie des Nombres, le Calcul des Probabilités, et, depuis ces dernières années, la Calculabilité.

b) Examinons la syntaxe de la procédure `normality`. On commence par assigner à `normality` le type d'une procédure de trois variables muettes `x`, `b` et `n`. Comme l'utilisateur de cette procédure sait quels sont les types des variables à employer (`float`, `integer` et `integer`), il est inutile de spécifier ces types. Ensuite, on déclare les variables qui seront locales à cette procédure qui ne sont que des intermédiaires de calcul : `fx` est essentiellement la partie fractionnaire qu'on développe, `i` une variable de boucle, `j` le terme courant du développement, `frequency` le tableau où sont enregistrées les fréquences d'apparition des différents chiffres entre 0 et $b-1$ (ses entrées sont initialisées à `0.0` afin de s'assurer qu'elles soient traitées comme des nombres flottants).

La quantité `Digits` désigne le nombre de décimales à prendre en compte lors de l'évaluation de nombres en virgule flottante. Les diverses expressions en nombres flottants sont en base 10 avec Maple. Un peu rapidement, on prend la partie entière supérieure de $n \ln b / \ln 10$ pour que l'expression flottante de `x` en base 10 comprenne suffisamment de décimales pour avoir `n` chiffres de la partie fractionnaire de `x` en base b.

Pour le reste de la procédure, tout est à peu près intuitif.

c) Il pourrait être intéressant de représenter sur un même graphique les suites de fréquences des différents chiffres d'un nombre donné comme sur ce qui suit :



On y voit représentées par des lignes brisées les fréquences d'apparition du développement en base 10 des mille premières décimales de π. Même s'il y a peut-être convergence vers 1/10 pour chaque suite de fréquences on n'espère jamais que ce soit très rapide.

TRAVAUX PRATIQUES DE PROBABILITÉS AVEC SCILAB

Une bonne partie du texte de ces travaux pratiques provient de

http : //cermics.enpc.fr/scilab_new/site/Tp/Probabilites/tp_proba_Q/

qui est une des pages pédagogiques de l'École Nationale des Ponts et Chaussées de Paris. En adaptant le texte, des incohérences auront pu apparaître. Soyez vigilants.

La séance de TP se fait sous environnement Linux. Pour commencer la séance, ouvrir un shell, créer un répertoire `tp_scilab` par exemple (`mkdir tp_scilab`) puis se placer dans ce répertoire (`cd tp_scilab`).

Lancer ensuite Scilab depuis ce répertoire `tp_scilab` (`scilab`). L'ensemble des questions pourront être traitée dans un fichier ou plusieurs (`1m02-tp.sce` par exemple) qu'on pourra éditer à l'aide de `emacs` ou de l'éditeur de Scilab. Pour l'éxécuter, il suffira de taper sur la ligne de commande Scilab :

```
-->exec 1m02-tp.sce;
```

Pour un rappel sur les opérations élémentaires de Scilab, on renvoie à ce qui aura été fait en travaux pratiques de math1modan (méthodologie).

On rappelle que pour avoir de l'aide sur une commande Scilab, il suffit de taper sur la ligne de commande Scilab :

```
-->help nom_de_la_commande;
```

1. Test du générateur aléatoire

Nous allons utiliser le générateur aléatoire de Scilab `rand`. On rappelle que la commande rand(n,m) renvoie une matrice aléatoire de taille $n \times m$, dont les composantes sont des réalisations indépendantes de variables aléatoires de loi uniforme sur $(0, 1)$.

Pour le tracé d'histogrammes, nous allons utiliser la commande `histplot`. On rappelle que la commande `histplot(n,x)` trace un histogramme des valeurs contenues dans le vecteur x, avec n barres de même largeur. On peut également utiliser la commande histplot(r,x), avec r un tableau donnant les valeurs pour échantillonner x.

Exercice 1. — Générer un vecteur de taille $N = 1000$ (i.e. une matrice de dimension $(1, N)$) dont les composantes sont des réalisations indépendantes de variables aléatoires de loi uniforme sur $(0, 1)$ avec la fonction rand. Tracer l'histogramme correspondant avec la fonction histplot. Augmenter N ($N = 10\,000$, $100\,000 \ldots$). Que constatez-vous ? Pouvez-vous l'expliquer ?

```
// Remplacer les ... par la commande appropriée
N=1000; // Nombre de réalisations de la variable aléatoire générée
// Génère un tableau x de taille (1,N) de réalisations
// de variables aléatoires indépendantes uniformes sur (0,1)
...
xbasc(); // Efface la fenêtre graphique
... // Trace un histogramme de x avec 100 barres.
```

2. **Simulation de variables aléatoires de loi exponentielle**

Exercice 2. — Choisir un réel $\lambda > 0$ et un entier N assez grand $(100, 1\,000, 10\,000, \ldots)$. Écrire une ligne de code Scilab qui retourne N réalisations indépendantes de loi exponentielle de paramètre λ. Tracer l'histogramme de ces N réalisations et lui superposer la densité de la loi exponentielle. Vérifier graphiquement la proximité à la loi originale. (On rappelle qu'en Scilab, si x est un vecteur, alors $\log(x)$ est le vecteur de composantes le logarithme des composantes de x.)

```
// Remplacer les ... par la commande appropriée
lambda=0.5; // Paramètre de la loi exponentielle
N=10000; // Nombre de réalisations de la variable aléatoire générée
// Génère un tableau z de taille (1,N) de réalisations d'une variable
// aléatoire exponentielle de paramètre lambda
...
xbasc(); // Tableau de discrétisation des abscisses
// L'intervalle (0,12) est divisé en 24 intervalles de même longueur
x=linspace(0,12,25);
...
// Trace l'histogramme de z échantillonné suivant x
// Calcule y, l'image de x par la fonction densité
// de la loi exponentielle de paramètre lambda
...
// Trace y en fonction de x
plot2d(x,y);
```

3. **Simulation de variables aléatoires de loi binomiale**

Exercice 3. — Expliquer pourquoi le vecteur y du programme suivant renvoie bien un vecteur de taille N dont les composantes sont des réalisations d'une variable aléatoire de loi binomiale de paramètres (n, p). Faire varier p et commenter les résultats.

Cette méthode pour générer des réalisations d'une variable aléatoire de loi binomiale est préférable à l'utilisation de boucles for qui sont très lentes en Scilab.

On rappelle que la fonction sum(w,'r') appliquée à une matrice w de taille $n \times m$ renvoie une matrice de taille $1 \times m$ dont chaque composante contient la somme des composantes de w par colonnes.

```
n=10;p=0.4; // Paramètres de la loi binomiale
N=10000; // Nombre de réalisations de la variable aléatoire générée
// Génère un tableau y de taille (1,N) de réalisations
// d'une variable aléatoire binomiale de paramètres (n,p)
y=sum(rand(n,N)<p,'r');
xbasc();
// Trace le diagramme en bâton de y
[ind, occ] = dsearch(y, 0:n, "d");
xbasc();
plot2d3(0:n,occ/N,rect=[-1,0,n+1,max(occ/N)*1.1],nax=[0,n+3,1,9]);
// Trace en bleu les probabilités exactes
plot2d3((0:n)+0.2,binomial(p,n),style=2);
```

4. **Loi forte des grands nombres**

EXERCICE 4. — Tirer N réalisations indépendantes $X_1,\ldots, X_N$ d'une loi exponentielle et tracer le graphique donnant $k \mapsto \dfrac{X_1+\cdots+X_k}{k}$. Commenter.

On utilisera la division terme à terme « `./` » et la fonction `cumsum` qui évalue des sommes cumulées, plutôt qu'une boucle. On rappelle par ailleurs que l'on peut générer le vecteur $(1,2,\ldots,N)$ sous Scilab par (1:N).

```
// Remplacer les ... par la commande appropriée
lambda=5; // Paramètre de la loi exponentielle
N=10000; // Nombre de réalisations de la variable aléatoire générée
// Génère un tableau x de taille (1,N) de réalisations d'une variable
// aléatoire exponentielle de paramètre lambda
...
xbasc();
// Crée un tableau y de taille (1,N) tel que
// y(i)=(x(1)+...+x(i))/i
...
// Trace y
plot(y);
```

EXERCICE 5. — Même question avec des X_i i.i.d. suivant une loi de Cauchy de paramètre a. Exécuter plusieurs fois le programme et commenter.

On rappelle que sous Scilab, la valeur de π est stockée dans `%pi`.

```
// Remplacer les ... par la commande appropriée
a=5;
// Paramètre de la loi de Cauchy
N=1000;
// Nombre de réalisations de la variable aléatoire générée
// Génère un tableau x de taille (1,N) de réalisations d'une variable
// aléatoire de Cauchy de paramètre a
...
xbasc();
// Crée un tableau y de taille (1,N) tel que y(i)=(x(1)+...+x(i))/i
...
// Trace y
plot(y);
```

5. **Théorème de la limite centrale**

EXERCICE 6. — Choisir un entier n assez grand. Tirer n réalisations indépendantes X_1, …, X_n d'une loi uniforme sur $[0,1]$ et calculer $\dfrac{X_1+\cdots+X_n-n/2}{\sqrt{n/12}}$. Choisir un entier N assez grand. Recommencer N fois et tracer un histogramme de la loi de $\dfrac{X_1+\cdots+X_n-n/2}{\sqrt{n/12}}$. Superposer la densité d'une loi normale centrée réduite. Jouer sur les paramètres n et N. Commenter.

On pourra utiliser la fonction `sum(.,'r'))`.

```
// Remplacer les ... par la commande appropriée
// stacksize(10000000);
// A décommenter si problème de mémoire ('stack size exceeded')
n=50;
```

```
// indice pour la convergence du TCL
N=1000; // nombre de réalisations pour tracer l'histogramme
// Crée un tableau w de taille (n,N) contenant des réalisations
// indépendantes d'une variable aléatoire uniforme sur (0,1)
...
// Calcule le vecteur z de taille (1,N) tel que
// z(i)=((w(1,i)+...+w(n,i))-n/2)/sqrt(n/12)
...
xbasc();
// Tableau de discrétisation des abscisses
// L'intervalle (-3,3) est divisé en 24 intervalles de même longueur
x=linspace(-3,3,25);
...
// Trace l'histogramme de z échantillonné suivant x
// Raffine la grille en abscisse pour tracer la densité
x=linspace(-3,3,250);
// Calcule y, l'image de x par la fonction densité
// de la loi normale centrée réduite
...
// Trace y en fonction de x
plot2d(x,y);
```

6. **Simulation de variables aléatoires de loi Beta(2,2) par la méthode du rejet**

La variable aléatoire X suit une loi Beta de paramètres $(2,2)$ si X admet la densité

$$p(x) = 6x(1-x)\mathbf{1}_{[0,1]}(x).$$

On propose de simuler cette variable aléatoire par la méthode du rejet en comparant cette loi à la loi uniforme sur $(0,1)$. On a évidemment :

$$6x(1-x)\mathbf{1}_{[0,1]}(x) \leqslant (3/2)\mathbf{1}_{[0,1]}(x).$$

On sait donc que si on se donne une suite i.i.d. (Y_i, U_i) avec Y_1 et U_1 indépendants de loi uniforme sur $(0,1)$, alors X a même loi que Y_N où $N = \inf\{i : (3/2)U_i \leqslant p(Y_i)\}$.

Exercice 7. — Programmer une fonction qui rend un vecteur de N réalisations indépendantes suivant la loi Beta de paramètres $(2,2)$ (on pourra utiliser une boucle while : taper `help while` pour avoir une description de la commande). Vérifier que l'on obtient bien la bonne densité en traçant un histogramme.

```
// Remplacer les ... par la commande appropriée function
[x]=rejetbeta(N)
x=zeros(1,N);
for i=1:N,
// Mettre une réalisation d'une variable aléatoire
// suivant une loi beta de paramètre (2,2) dans x(i).
...
end;
endfunction
N=10000; // Nombre de réalisations de la variable aléatoire générée
z=rejetbeta(N);
// Tableau de discrétisation des abscisses
// L'intervalle (0,1) est divisé en 24 intervalles de même longueur
x=linspace(0,1,25);
xbasc();
```

```
histplot(x,z); // Trace l'histogramme de z échantillonné suivant x
// Calcule y, l'image de x par la fonction densité
// de la loi beta de paramètre (2,2)
y=6*x.*(1-x).*(x<1 & x>0);
// Trace y en fonction de x
plot2d(x,y);
```

7. **Simulation de variables aléatoires de loi normale**

Exercice 8. — Choisir un entier N assez grand (100, 1 000, 10 000...). Utiliser la méthode polaire pour créer un vecteur de taille N dont les composantes sont des réalisations indépendantes d'une loi normale centrée réduite. Tracer l'histogramme de ces N réalisations et lui superposer la densité de la loi normale. Vérifier graphiquement la proximité à la loi originale. On pourra utiliser la multiplication terme à terme « `.*` ».

```
// Remplacer les ... par la commande appropriée N=1000;
// Nombre de réalisations de la variable aléatoire générée
// Génère un tableau z de taille (1,N) de réalisations d'une variable
// aléatoire normale centrée réduite
...
xbasc();
// Tableau de discrétisation des abscisses
// L'intervalle (-3,3) est divisé en 24 intervalles de même longueur
x=linspace(-3,3,25);
...
// Trace l'histogramme de z échantillonné suivant x
// Raffine la grille en abscisse pour tracer la densité
x=linspace(-3,3,250);
// Calcule y, l'image de x par la fonction densité
// de la loi normale centrée réduite
...
// Trace y en fonction de x
plot2d(x,y);
```

8. **L'aiguille de Buffon**

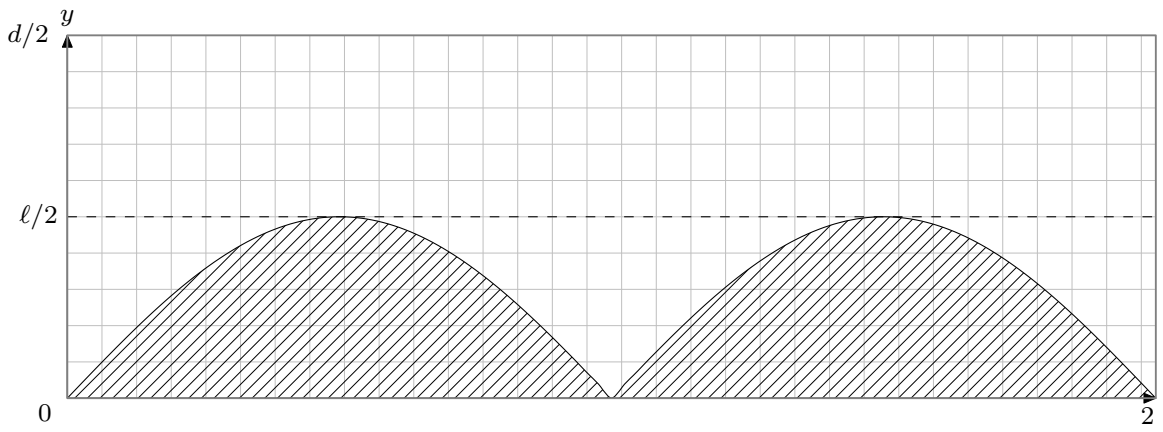
L'« aiguille de Buffon » est un classique incontournable du Calcul de Probabilités. On dispose d'un aiguille, ou d'un bâton, de longueur ℓ (dont diamètre est négligé). On la lance « au hasard » sur un parquet en bois formant un réseau régulier de droites parallèles d'égale distance d avec $\ell < d$. La question est de savoir quelle est la probabilité que l'aiguille intersecte l'une de ces droites.

La difficulté de ce problème tient à la modélisation. L'aiguille doit se répartir uniformément sur le plan — implicitement infini — de lancer. Il est donc nécessaire de faire quelques réductions pour munir le modèle (qui doit décrire comment tombe l'aiguille) d'une mesure de probabilité.

Si M désigne le centre de l'aiguille, sa coordonnée parallèle aux lattes peut être ignorée. Sa coordonnée orthogonale aux lattes est supposée de « loi » uniforme sur $\mathbb{R}$; ainsi, modulo d, cette coordonnée est de loi uniforme sur $[0, d]$. En ne s'intéressant qu'à la droite la plus proche de ce centre — la seule qui pourrait être intersectée puisque $\ell < d$, la variable que nous retenons, et que nous notons Y, est la distance du centre à cette droite ; elle est de loi uniforme sur $[0, d/2]$. L'aiguille pouvant avoir un bout pointu et un autre qui ne l'est

pas, désignons par Θ à valeurs dans $[0, 2\pi[$ l'angle formé par le bout pointu et une des deux directions des lattes, la variable Θ est supposée de loi uniforme.

On constate qu'il ne peut y avoir intersection que si $Y \leqslant |\sin\Theta| \times \ell/2$.



En représentant les coordonnées (θ, y) dans $[0, 2\pi] \times [0, d/2]$ muni de la mesure de probabilité uniforme (loi du couple (Θ, Y)), tout revient à calculer l'aire normalisée sous la courbe $\theta \mapsto |\sin\theta| \times \ell/2$ dont la valeur est

$$\int_0^{2\pi} |\sin\theta| \times \ell/2 \,\mathrm{d}\theta \times \frac{1}{2\pi \times d/2} = \int_0^{\pi} \sin\theta \times \ell \,\mathrm{d}\theta \times \frac{1}{\pi \times d} = \frac{2\ell}{\pi \times d}\,.$$

Exercice 9. — (i) Préciser comment avec une suite de vecteurs aléatoires $(\Theta_n, Y_n)_{n\geqslant 1}$ indépendants de lois uniforme sur $[0, 2\pi] \times [0, d/2]$ il est possible d'approcher la probabilité précédente.

(ii) Prendre $\ell = d/2 = 1$. À l'aide d'appels successifs de la fonction `rand()`, simuler avec une procédure un nombre variable d'issues du couple (Θ, Y) (la quantité Scilab `%pi` intervient). Se servir de ces différentes issues pour former un tableau où figurera les approximations successives de la probabilité cherchée. Représenter cette suite graphiquement.

(iii) Le Théorème Central Limite donne-t-il une information sur l'erreur d'estimation ?