TP 1 - Introduction (probabiliste) à Scilab
Illustrations de convergences presque sûre[1]

1 Prise en main du logiciel

1.1 Introduction

Environnement Scilab

Scilab est un environnement de calcul numérique libre qui peut-être téléchargé à l'URL `http://www.scilab.org/fr`.

L'espace de travail présente plusieurs fenêtres :

La console pour faire des calculs

L'éditeur pour écrire des programmes

Le navigateur de variables

L'historique de commandes

Le menu **Applications** permet d'accéder à ces différentes fenêtres si elles ont été fermées.

Pour des calculs simples on utilise la console : les lignes de commandes sont directement tapées après la flèche `-->`, et exécutées grâce à la touche Entrée. Un point-virgule à la fin d'une ligne de commande permet d'exécuter ladite commande sans faire afficher le résultat. Une virgule entre deux commandes sur la même ligne permet d'exécuter deux commandes à la fois. On peut accéder aux commandes précédentes à partir des flèches du clavier. La tabulation permet, quand on a commencé de taper une commande, de rechercher une complétion automatique.

Scilab possède une aide en ligne. On peut y accéder par le menu **?**. La commande `help` suivie du nom d'une fonction ou commande permet d'accéder directement à la page d'aide concernant cette fonction. On pourra trouver également sur Internet des supports de cours bien plus complets que cette (trop) rapide introduction. Une référence est par exemple `http://ljk.imag.fr/membres/Bernard.Ycart/mel/ds/ds.pdf`.

Affectation de variables

On utilise le symbole `=` pour l'affectation de variables. Les variables peuvent être de différents types : des nombres, des matrices, ou des chaînes de caractères.

Les noms de variables peuvent comporter lettres et chiffres, mais pas d'accents ou de caractères spéciaux. Exemples :

`--> a=3` : la valeur `3` est affectée à la lettre `a`.

`--> a=3 ; b=5 ; a=a+b` : la nouvelle valeur de `a` est `8`.

Enfin, `%pi` représente π, `%e` représente e, et `%i` représente le complexe i.

La commande `clear` permet d'effacer le contenu de l'ensemble des variables en mémoire.

⚠ Scilab est sensible à la casse (minuscules/majuscules) : `A` et `a` ne désignent pas la même variable !

1. Enseignant : G. Chagny, bureau M.2.35. `gaelle.chagny@univ-rouen.fr`.

Comparaison et tests

On peut comparer deux variables grâce aux commandes suivantes : `<` (infériorité stricte), `<=` (infériorité large), `>` (supériorité stricte), `>=` (supériorité large), `==` (égalité), `<>` (non égalité). On peut imbriquer plusieurs tests, avec `&` (et), `|` (ou), `~` (non). Exemple :

```
--→ a=3; b=(a<2 | a>2.5)
```

La réponse à un test de comparaison est une valeur booléenne : `T` (true) ou `F` (false). Si on souhaite définir soit-même une variable booléenne, il faut utiliser `%t` (true) ou `%f` (false). On peut convertir un booléen en une valeur numérique (0 ou 1) en le multipliant par 1 ou à l'aide de la commande `bool2s`.

Remarque : Toutes ces commandes, comme les suivantes indiquées dans la fiche, peuvent généralement être utilisées pour comparer deux matrices (ou deux vecteurs) : la comparaison se fait alors terme à terme.

Affichage d'un résultat - Dialogue avec l'utilisateur

Le résultat d'une ligne de commande est affiché dès que celle-ci ne se termine pas par un point-virgule. La commande `disp` permet de forcer un affichage propre. Elle sera également nécessaire dans les scripts et fonctions (voir Section 1.4). Exemple :

```
--→ a=3;b=%t; disp('valeur de a='+string(a)+' et valeur de b='+string(b)').
```

Via la commande `input`, on peut demander à l'utilisateur de saisir une entrée. Exemple :

```
--→ x=input('Entrer un nombre réel')
```

1.2 Vecteurs et matrices

Scilab est un langage vectoriel, où toutes les fonctions s'appliquent à des vecteurs et les opérations vectorielles/matricielles sont généralement plus efficaces que l'emploi de boucles. Autant que possible, on tentera donc de remplacer les boucles par des opérations matricielles.

Création de matrices

Fabrication manuelle. Pour définir un vecteur ou une matrice, on utilise les crochets [ ]. A l'intérieur des crochets, on utilise le point-virgule pour former des colonnes, et la virgule (ou l'espace) pour former des lignes. Exemples :

```
--→ [1;2;3;4;5;6], [1 2 3 4 5 6], [1,2,3,4,5,6], [1;2;3,4;5;6], [1,2,3;4,5,6]
```

Fabrication automatique. Exemples :

Vecteurs.

```
--→ 1:5, 0:0.1:1, linspace(0,1,21)
```

Matrices.

```
--→ eye(3,3), eye(3,7), ones(3,5), zeros(2,5), diag([1 5 -7])
```

Réordonner les éléments d'une matrice.

Fabrication d'une matrice de dimensions données à partir de coefficients.

```
--→ x=1:8; A=matrix(x,2,4), B=matrix(x,4,2)
```

⚠ Les arguments de la commande `matrix(vect, nb_lignes, nb_colonnes)` doivent être compatibles : `nb_lignes*nb_colonnes` doit être égal à la longueur du vecteur `vect`.

Concaténation des éléments d'une matrice en un seul vecteur colonne.

```
--→ A(:)
```

Concaténation de matrices. Extraction - Remplacement d'éléments.

On peut concaténer vecteurs et matrices, tant que leurs tailles sont compatibles. Exemples :

```
--→ u=[1,2,3,4] ; v=[5,6,7,8]; w=[u v] , A=[u ; v]
--→ B=[9 10 11 12;13 14 15 16] ; C=[A ; B]
```

On peut extraire un (ou plusieurs) éléments, une (ou plusieurs) ligne(s), une (ou plusieurs) colonne(s) d'une matrice, ou en supprimer certaines parties. Exemples :

```
--→ A=[1 2 3;-1 -2 -3], A(1,2), A(2,[1,3]), A(:,2), A(2,:), A(:,3)=[]
```

On peut définir un vecteur ou une matrice coordonnée par coordonnée, sans avoir besoin de prédéfinir sa taille. Le logiciel complète par des zéros les cases non spécifiées.

```
--→ u(1)=1; u(2)=3
--→ v(1)=1; v(2)=3; v(5)=-1;
--→ w(1,1)=5; w(3,5)=4
```

⚠ Scilab indexe toujours les vecteurs en commençant par l'entier 1 (de même pour les numéros de colonnes ou de lignes de matrices). Un essai pour accéder à une "case" d'indice 0 provoque une erreur.

Les commandes `size` et `length` renvoient respectivement les dimensions (nombres de lignes et de colonnes) et le nombre de coefficients d'une matrice (la longueur d'un vecteur).

Opérations élémentaires

On peut effectuer les opérations classiques sur les matrices (ou les vecteurs, ou les scalaires) en prenant garde si besoin à la compatibilité des tailles de matrices :

```
--→ A', A+B, 1-A, A*B, A^n,...
--→ det(A), trace(A), spec(A)...
```

On peut aussi faire certaines de ces opérations terme à terme :

```
--→ A.*B, A./B, A.^n, sqrt(A), ...
```

⚠ Pour calculer l'inverse terme à terme d'une matrice `A`, la commande est `1 ./A` (et pas `1./A`). Pour éviter ce problème, on peut aussi utiliser `A.^(-1)`.

Certaines fonctions peuvent s'appliquer à l'ensemble de la matrice, ligne à ligne, colonne par colonne. Exemples :

```
--→ sum(A), sum(A,'c'), sum(A,'r')
--→ prod(A), cumprod(A), cumsum(A), mean(A), gsort(A), stdev(A), variance(A)...
```

⚠ Pour le calcul de la variance (`variance`) et de l'écart-type (`stdev`) empirique, la normalisation est en $1/(n-1)$ où n est le nombre d'éléments de la matrice.

1.3 Structures itératives et conditionnelles

La structure correspondant au branchement conditionnel est la suivante :

```
--→ if condition (1) then
--→ instructions (1) ...
--→ elseif condition (2) then
--→ instructions (2) ...
--→ else
--→ instructions (3)...
--→ end
```

Même s'il est recommandé d'éviter autant que possible les boucles, voici leurs syntaxes :

```
--→ while condition
--→ instructions...
--→ end
```

```
--→ for i=1:n
--→ instructions...
--→ end
```

1.4 Utilisation de l'éditeur

Présentation de Scinotes

Dès que les calculs à effectuer requièrent plusieurs lignes de commandes, il est recommandé d'utiliser l'éditeur de Scilab, *Scinotes*, pour sauvegarder le code et le modifier facilement. Le jour de l'agrégation, ne pas oublier de sauver TOUTES les commandes que l'on souhaite montrer au jury (l'historique de commandes, accessible par les flèches du clavier, peut dépendre de l'ordinateur sur lequel on travaille).

Scinotes est accessible depuis le menu **Applications.** On peut créer deux types de fichiers texte, des fichiers de commandes ou scripts, et des fichiers de fonctions. Pour utiliser ces fichiers (exécuter les commandes, charger les fonctions), il faut se placer dans le répertoire courant (celui où Scilab va chercher par défaut les fichiers à exécuter), ou à défaut, il faudra saisir le chemin complet pour accéder au fichier que l'on souhaite utiliser. Il est conseillé pour chaque TP, ou pour l'ensemble des TP, de créer un répertoire dans lequel stocker les fichiers Scilab correspondants, puis de s'y placer via la commande **Changer le répertoire courant...** du menu **Fichier**, ou directement via le navigateur de fichiers.

Les fichiers de commandes et de fonctions peuvent (doivent) contenir des commentaires : on commence les lignes correspondantes par `//`. Ceux-ci facilitent la relecture des programmes et sont très utiles aussi le jour de l'agrégation pour montrer ses programmes au jury.

Fichiers de commandes

Ce sont des fichiers texte d'extension `.sce`, contenant des suites d'instructions. Il faut obligatoirement enregistrer un script pour pouvoir exécuter les commandes qu'il contient. On peut ensuite cliquer sur l'icône "exécuter" (ou "enregistrer et exécuter" si le fichier a été modifié entre-temps), ou sélectionner un mode d'exécution dans le menu **Exécuter** de Scinotes, ou encore utiliser la commande `exec('nom_fichier.sce')` dans la console.

Par défaut, l'exécution fonctionne comme s'il y avait des points-virgules partout (et donc n'affiche rien). Il faut utiliser la commande "Exécuter avec écho" du menu **Exécuter** ou utiliser `disp` pour afficher les variables.

⚠ Il est recommandé d'écrire en première ligne, pour tout calcul dans l'éditeur, les commandes `clear` et, si nécessaire, `clf`, qui permettent d'effacer respectivement les données mises en mémoire, et les figures. Cela est utile pour éviter les erreurs et libérer la mémoire.

Fichiers de fonctions

Ce sont également des fichiers texte, dont l'extension standart est `.sci`. Ils contiennent les définition d'une ou plusieurs fonctions. La syntaxe utilisée pour définir une fonction prenant $x_1,\dots,x_n$ comme arguments et $y_1,\dots,y_m$ comme valeurs de sortie, est la suivante :

```
--→ function [y1,...,ym]=nom_de_la_fonction(x1,...,xn)
--→   instructions...
--→ endfunction
```

Les variables $(y_i)_{1\leq i\leq m}$ et $(x_i)_{1\leq i\leq n}$ sont des variables locales (ou variables muettes) et peuvent donc, par exemple, être réutilisées dans d'autres fonctions.

Remarque : Si l'on veut une fonction qui demande une saisie au clavier des arguments, on crée une fonction sans argument et on utilise la commande `input`.

Il faut ensuite charger la (les) fonction(s) avec la commande `exec('nom_fichier.sci')` (dans un script par exemple) ou le menu **Exécuter** avant de pouvoir l' (les) utiliser.

⚠ Il n'y a *a priori* pas de raisons de distinguer les fichiers d'extension `.sce` et les fichiers `.sci` : on pourrait tout à fait définir des fonctions au milieu de la suite des commandes d'un script. Cependant, pour la clarté de la programmation, il est peut-être mieux de séparer les commandes des fonctions.

1.5 Représentations graphiques

Fenêtres graphiques

Par défaut, les graphiques successifs sont superposés sur la même fenêtre. On efface la fenêtre courante par `clf()`. La commande `figure` permet d'ouvrir une nouvelle fenêtre. De manière plus précise, on ouvre la fenêtre numéro `i` par `scf(i)`. On efface son contenu avec `clf(i)`, et on ferme la fenêtre avec `xdel(i)`.

On peut subdiviser une fenêtre graphique en plusieurs sous-fenêtres, à l'aide de la commande `subplot`. Par exemple, la commande `subplot(2,3,1)` permet de séparer la fenêtre graphique courante en 6 sous-graphiques (2 lignes, 3 colonnes), et définit la sous-fenêtre courante comme étant la première.

Commande `plot2d`

On s'intéresse ici à représenter le graphe d'une fonction quelconque (voir TP suivant pour les histogrammes, ou diagrammes en barre). La commande `plot2d` permet de tracer courbes et nuages de points dans le plan. Si on exécute plusieurs `plot2d` à la suite, les courbes correspondantes seront par défaut sur la même figure. Pour effacer le contenu de la figure, il faut utiliser `clf` (voir ci-dessus).

Pour représenter une fonction f de $\mathbb{R}$ dans $\mathbb{R}$, on trace en fait l'interpolation linéaire d'un nuage de points $\{(x_i, f(x_i)), i = 1, \dots, n\}$ où le vecteur x est choisi en fonction de l'intervalle sur lequel on veut tracer la courbe, et de la précision cherchée pour le tracé. La commande `plot2d(x,y)` représente les points de coordonnées $(x_i, y_i)_i$ en les joignant par des traits noirs (par défaut), ou selon un autre style, si le style de base a été changé. Exemples :

```
--→ plot2d(1:4,[0 2 -3 4])
--→ x=0:0.1:1; plot2d(x,x.^2)
```

La syntaxe générale de la fonction `plot2d` est la suivante :

```
--→ plot2d(X, Y, options)
```

`X,Y` Si ces paramètres sont des vecteurs, ils peuvent être ligne ou colonne, et une seule courbe est tracée. Si ce sont des matrices, plusieurs courbes seront tracées (sauf erreur !). Par défaut les points sont reliés par des segments et à chaque courbe correspond une couleur (jusqu'à 32). Deux possibilités :

- Soit `X` est un vecteur colonne à n lignes, et `Y` une matrice à p colonnes et n lignes : p est alors le nombre de courbes tracées, à partir de points ayant les mêmes abscisses spécifiées dans `X`, et dont les ordonnées sont définies dans chacune des colonnes de `Y`. Exemple :

```
--→ x=linspace(-1,1,30); y1=x.^2; y2=x.^3; plot2d(x', [y1',y2'])
```

- Soit `X` et `Y` sont des matrices de même dimension (n, p) : n, le nombre de ligne est le nombre de points de représentation, et p le nombre de courbe. Chaque colonne de `X` est un vecteur d'abscisses auquel correspond un vecteur d'ordonnées (la colonne de même indice dans `Y`). Exemple :

```
--→ x1=linspace(-1,1,30); x2=linspace(0,1,100);
    y1=x.^2; y2=x.^3; plot2d([x1',x2'], [y1',y2'])
```

options De nombreuses options de tracé sont possible, présentées sous la forme `nom_option=valeur` et séparées par des virgules. Voici quelques exemples (se référer à l'aide pour plus de possibilités).

Style de tracé. `style=valeur`. `valeur` est un vecteur ligne d'entiers dont la dimension est le nombre de courbes à tracer. Les coordonnées sont positives ou négatives. Si la coordonnée i est strictement positive, la courbe i sera tracée en ligne pleine de couleur (les couleurs sont numérotées de 1 à 32, voir `getcolor`. Si le style est négatif ou nul, on obtient un nuage de points (les styles de points sont numérotés de -14 à 0).

Rectangle de représentation. `rect=valeur`. `valeur` est un vecteur ligne de 4 coordonnées sous la forme `[xmin,ymin,xmax,ymax]`.

Légendes. `leg=valeur`. `valeur` est une chaîne de caractères (`valeur=''...''`) contenant les légendes des différentes courbes, séparées par le symbole @. On peut aussi utiliser la commande `legend`, après avoir lancé la commande `plot2d`.

Les commandes `title` ou `xtitle` permettent respectivement d'ajouter un titre au graphique, des titres pour le graphique et les axes. Exemple :

```
--→ x=linspace(-1,1,30); X=x'*ones(1,5); y=x.^2; Y=y'*[1:5];
--→ styles=[-2:2]; legendes="x.^2@2x.^2@3 x.^2@4 x.^2@5 x.^2"
--→ clf; plot2d(X,Y,style=styles,leg=legendes); xtitle('Exemples de tracé', 'x','y')
```

La fonction `plot2d2` permet de tracer de la même façon des fonctions en escalier (utile pour représenter les fonctions de répartition empiriques).

1.6 Générer des variables aléatoires

Simulation d'échantillons de loi uniforme sur $[0; 1]$

Scilab dispose d'un générateur de nombres pseudo-aléatoires, c'est-à-dire une séquence déterministe de réels dont le but est de reproduire une séquence de variables aléatoires indépendantes et identiquement distribuées de loi $\mathcal{U}_{[0;1]}$. La fonction est `rand`. Exemples :

```
--→ rand, rand(2,3)
```

Simulation d'échantillons de lois de probabilité classiques

La fonction `grand` permet de simuler des tirages de variables aléatoires de lois usuelles (exponentielle, normale, binomiale, Poisson,...). La syntaxe générale est :

```
--→ grand(n,m,'nom',parametres)
```

Cette commande renvoie un échantillon présenté sous la forme d'une matrice à `n` lignes et `m` colonnes d'une variable de loi précisée dans la chaîne de caractère `'nom'` et dont les paramètres sont spécifiés dans `parametres`. La chaîne de caractères `'nom'` peut par exemple être `'bin'` (loi binomiale), `'poi'` (loi de Poisson), `'nor'` (loi normale), `'exp'` (loi exponentielle), `'bet'`

(loi béta), `'gam'` (loi gamma), `'chi'` (loi du Chi-deux)... On se référera à l'aide pour avoir la liste complète des lois simulables via `grand`, et les détails de leurs paramètres.
⚠ Attention aux paramètres pour les lois exponentielles et normales, ce ne sont pas forcément ceux sous-entendus par les notations $\mathcal{E}(\lambda)$ et $\mathcal{N}(m, \sigma^2)$...

Calcul sur les lois de probabilité classiques

On dispose en Scilab de fonctions dont le nom commence par `cdf` et permettant de retrouver la fonction de répartition F, la fonction quantile $F^{(-1)}$ (inverse généralisé de la fonction de répartition : à une probabilité $p = F(x)$, on associe le quantile x d'ordre p), et la densité des lois usuelles : `cdfbin` (loi binomiale), `cdfpoi` (loi de Poisson), `cdfnor` (loi normale), `cdfgam` (loi gamma), `cdfchi` (loi du Chi-deux), `cdft` (loi de Student)... Chaque loi dépend de paramètres. Une fois ceux-ci fixés, la fonction de répartition de la loi et les quantiles sont déterminés de manière unique.

Toutes les fonctions `cdf` fonctionnent sur le principe suivant : on leur donne en entrée toutes les quantités (paramètres, quantiles, valeurs de la fonction de répartition) sauf une, ainsi que l'option choisie et la fonction renvoie la quantité manquante.

On prend l'exemple de la fonction `cdfnor` associée à la loi normale $\mathcal{N}(m, \sigma^2)$ d'espérance m et de variance σ^2 et dont on note $F_{m,\sigma}$ la répartition (voir l'Exercice 1 pour un autre exemple avec `cdfbin`). Les différentes possibilités sont les suivantes :

- `--→ [P,Q]=cdfnor('PQ',X,m,sigma)` : renvoie $P = F_{m,\sigma}(X)$, $Q = 1 - P$.
  La commande `cdfnor('PQ',X,m,sigma)` renvoie par défaut uniquement P.
- `--→ X=cdfnor('X',m,sigma,P,Q)` : renvoie $X = F^{(-1)}(P)$. Les paramètres en entrée peuvent aussi être des vecteurs de même taille.
- `--→ m=cdfnor('Mean',sigma,P,Q,X)` : renvoie l'espérance m.
- `--→ sigma=cdfnor('Std',P,Q,X,m)` : renvoie l'écart-type σ.

Les paramètres en entrée peuvent dans chacun des cas être des vecteurs de même taille, et dans ce cas on obtient en réponse un vecteur (ou deux) de même taille. Par exemple, dans le premier cas, si `X,mu,sigma` sont des vecteurs, alors P et Q sont des vecteurs de même taille et sont tels que $P_i = F_{m_i,\sigma_i}(X_i)$.

Remarque : La différentielle discrète de la fonction de répartition donne une approximation de la densité. Exemple :

```
--→  x=linspace(-3,3,100); Fx=cdfnor("PQ",x,zeros(x),ones(x));
--→  fx=(Fx(2:100)-Fx(1:99))*100/6; x(1)=[]; plot2d(x,fx);
```

2 Exercices

2.1 Échauffement

Exercice 1 *Écrire une fonction, représenter une courbe.*

1. Écrire une fonction qui prend en entrée un vecteur $vect\_x$, un réel m et un réel strictement positif s et qui retourne les valeurs de la densité de la loi normale $\mathcal{N}(m, s^2)$ aux points d'abscisses $vect\_x$.
2. Utiliser la fonction pour tracer les courbes représentatives des densités des lois normales $\mathcal{N}(0, 1)$, $\mathcal{N}(0, 4)$ et $\mathcal{N}(0, 0.2)$ sur un même graphique.

Exercice 2 *Utiliser la fonction* `grand`.

1. Construire un vecteur de n nombres aléatoires x_i, $i = 1, \ldots, n$ issus d'une loi normale de moyenne 3 et de variance 1.
2. Calculer la moyenne et la variance empiriques de l'échantillon simulé.

2.2 Autour de la loi binomiale

Exercice 3 *Calcul de coefficients binomiaux.* L'objectif est de calculer les coefficients binomiaux $\binom{n}{k}$, $k \in \{0, \ldots, n\}$, $n \geq 1$ à l'aide de plusieurs méthodes.

1. Comment calculer la valeur $k!$ pour un entier k dans Scilab ? Faire des tests pour différentes valeurs de k. En particulier, tester $k = 170$ et $k = 171$. Comment calculer la liste $1!, 2!, \ldots, k!$ en une seule ligne de commande ?
2. Implémenter des fonctions qui renvoient le vecteur ligne des $\binom{n}{k}$, $k \in \{0, \ldots, n\}$ pour un paramètre n, en utilisant la définition (éventuellement après simplification) des coefficients binomiaux. On pourra commencer par écrire une fonction faisant intervenir une boucle (calcul pour les différentes valeurs de k), puis en ré-écrire une seconde sans boucle, en vectorisant les opérations de calcul.
3. Exprimer les coefficients $\binom{n}{k}$ en utilisant la fonction de répartition F_n d'une loi binomiale $\mathcal{B}(n, 1/2)$. En déduire une troisième fonction retournant le vecteur ligne des $\binom{n}{k}$, $k \in \{0, \ldots, n\}$.
4. Rappeler la formule permettant de construire un triangle de Pascal. En déduire encore une autre fonction de calcul des $\binom{n}{k}$, $k \in \{0, \ldots, n\}$.

Exercice 4 *Approximation d'une fonction continue par des polynômes de Bernstein.* Soit f une fonction continue sur $[0; 1]$. Pour $n \geq 1$, le $n-$ième polynôme de Bernstein (fonction polynomiale) associé à f est défini par

$$B_n f(x) = \sum_{k=0}^{n} \binom{n}{k} x^k (1-x)^{n-k} f\left(\frac{k}{n}\right), \ x \in [0; 1].$$

On rappelle que la suite $(B_n f)_n$ converge uniformément vers f sur $[0; 1]$ (voir Exercice 13 du polycopié de cours).

1. Écrire une fonction qui prend en entrée une fonction f, un entier n et un vecteur d'abscisses $vect\_x$ et qui retourne les valeurs $(B_n f(x))_{x \in vect\_x}$.
2. Tester la fonction en traçant sur un même graphique la courbe représentative d'une fonction f et celles des polynômes de Bernstein associés pour différentes valeurs de n. On pourra d'abord tester avec des fonctions déjà définies dans le logiciel (`sin, exp...`)

puis (bonus !) utiliser la commande `deff` de Scilab pour introduire des fonctions variées (tester par exemple avec $f(x) = \sqrt{|x - 1/2|}$, avec des fonctions monotones, convexes...).

2.3 Illustration de convergences presque-sûres

Exercice 5 *Illustration de la loi des grands nombres.* On considère les lois de probabilité suivantes : exponentielle, normale, uniforme sur les entiers. Pour chacune d'entre elles, générer un échantillon $(X_i)_i$ de taille n (avec n assez grand), et représenter sur le même graphe la suite des valeurs de $\sum_{i=1}^{m} X_i/m$, en fonction de $m = 1, \ldots, n$, ainsi qu'une droite horizontale correspondant à l'espérance de la loi correspondante. Que se passe-t-il si l'on choisit une loi de Cauchy ?
Indication : pour simuler une variable de loi de Cauchy, on pourra par exemple utiliser que si X_1 et et X_2 sont deux variables indépendantes de loi $\mathcal{N}(0, 1)$, alors X_1/X_2 suit une loi de Cauchy (le démontrer).

Exercice 6 *Convergence d'une suite de maximums.* Soit $(X_i)_{i\geq 1}$ une suite de variables aléatoires *i.i.d.* de loi $\mathcal{U}_{[0;1]}$, et pour tout $n \geq 1$, $M_n = \max(X_1, \ldots, X_n)$.

1. Justifier la convergence presque sûre de la suite $(M_n)_{n\geq 1}$.
2. Générer un échantillon de variables $(X_i)_i$ et représenter M_n en fonction de n. Recommencer sur plusieurs échantillons (on pourra éventuellement les représenter sur le même graphique). Quelle hypothèse peut-on faire quant à la limite presque-sûre ? Le démontrer.