Université Denis Diderot Paris 7

Simulations probabilistes

Préambule

Ce module est un complément du cours de Probabilités de L3. Il est obligatoire pour la filière Mathématiques appliquées, et optionnel pour la filière Maths fondamentales.

Le volume horaire est de 1h cours / sem (Me 14.30) et 1h30 de TP / sem (Je 16.00 Maths Appli., Je 10.00 Maths Fonda.).

De sujets de projet seront distribués à la mi-mars. Les projets seront réalisés en binôme, et l'évaluation du module portera sur le compte-rendu écrit et la soutenance orale du projet.

Les simulations sont effectuées ici avec le logiciel scilab 5.5.1, disponible sur `http://www.scilab.org/fr`. [On poura cependant choisir d'effectuer le projet en utilisant un autre langage.]

1 La fonction `rand`

Les fonctions `rand, grand` de scilab sont des *générateurs de nombres (pseudo)-aléatoires*. On va accepter dans la suite de ce cours que la fonction `rand` permet de simuler des variables indépendantes et uniformes sur l'intervalle $[0, 1]$: autrement dit, cette fonction permet de simuler des choix indépendants et uniformes de nombres de l'intervalle $[0, 1]$ (la fonction `grand` possède un panel plus large d'options, on y reviendra dans la deuxième partie). Pour justifier au moins de façon heuristique cette affirmation, on va commencer dans la partie qui suit par observer les propriétés de cette commande `rand`.

1.1 Premier contact avec la fonction `rand`

Quand on appelle la commande (rand()), scilab renvoie un nombre entre 0 et 1 :

```
-->rand()
 ans =
  0.2113249
```

Attention, par défaut, scilab ne donne "que" 7 décimales, mais en réalité la précision est plus fine :

```
 --> ans - 0.2113249
 ans =
  - 3.454D-08
```

A chaque fois qu'on va rappeler la commande on va obtenir un résultat différent [1] :

```
-->rand()
 ans =
  0.7560439

 -->rand
 ans =
  0.0002211
```

On peut effectuer la commande plusieurs fois d'un seul coup : `rand(1:n)` renvoie ainsi une liste de n nombres entre 0 et 1, `rand(m,n)` renvoie un tableau de m lignes et n colonnes de tels nombres.

```
 -->rand(1:4)
 ans =
  0.1985144 0.5442573 0.2320748 0.2312237

-->rand(5,5)
 ans =
  0.2164633  0.2146008  0.4826472  0.2693125  0.4818509
  0.8833888  0.312642   0.3321719  0.6325745  0.2639556
  0.6525135  0.3616361  0.5935095  0.4051954  0.4148104
  0.3076091  0.2922267  0.5015342  0.9184708  0.2806498
  0.9329616  0.5664249  0.4368588  0.0437334  0.1280058
```

1.2 Propriétés de `rand`

Les nombres générés par la fonction `rand` possèdent les propriétés apparentes d'une suite de variables aléatoires i.i.d, uniformes sur $[0, 1]$:

(i) La suite de nombres se comporte de façon chaotique, de sorte que les éléments successifs de cette suite de nombres semblent imprévisibles. Plus précisément, la connaissance des k premiers termes de la suite ne semble pas nous donner d'information sur le $(k+1)$-ème terme [2].

(ii) une observable moyennée sur un grand nombre de telles réalisations semble se concentrer autour d'une quantité fixée.

1. Il est cependant à noter que si on ferme scilab puis on le redémarre, le premier appel à la commande `rand()` va à nouveau fournir le résultat `0.2113249`, le deuxième va à nouveau fournir le r résultat `0.7560439`, etc... Si on souhaite éviter cela on peut réinitialiser aléatoirement la racine de `rand`, en utilisant par exemple la date (en secondes) `n=getdate("s"); rand("seed",n);`

2. si on s'intéresse d'un peu plus près à la façon dont les nombres pseudo-aléatoires sont générés, cette propriété n'est pas tout à fait vraie. Cependant, pour pouvoir prédire ce $(k+1)$-ème nombre, il faudrait non seulement connaître la façon dont la fonction est programmée, mais également connaître le $k$ème terme de la suite avec une précision maximale (ici 10 décimales). Dans le cas qui nous intéresse, même en connaissant la façon dont la fonction est programmée, la simple connaissance des 7 premières décimales du k-ème terme ne nous donnerait qu'une information très partielle sur le $(k+1)$-ème terme — dans cet exemple, et avec cette information partielle les décimales d'un tel nombre, y compris la première, seraient essentiellement non prévisibles.

(iii) aucune région de l'intervalle $[0,1]$ ne semble privilégiée.

Pour préciser ce qu'on entend par (ii) et (iii) ci-dessus on a besoin d'introduire (de façon informelle pour le moment) plusieurs notions.

1.3 Evénements, fréquence empirique

La réalisation de `rand()` est un exemple concret d'*expérience aléatoire*.
Informellement, la notion d'expérience aléatoire correspond justement à une expérience qui vérifie les propriétés (i) et (ii) ci-dessus.
Informellement, un *événement* correspond à une propriété qu'on peut déclarer vraie ou fausse lors de la réalisation d'une expérience aléatoire.
Plus rigoureusement, on notera Ω l'ensemble des expériences aléatoires et un événement $\mathcal{E}$ va correspondre à une partie de Ω. On mesurera la taille d'un événement à l'aide d'une mesure de probabilité, de sorte que l'ensemble des événements se doit de former une tribu $\mathcal{F}$ sur Ω.
La fonction $\mathbb{1}_{\{\mathcal{E}\}} : \Omega \to \{0,1\}$ est une *fonction de la réalisation de notre expérience aléatoire* : elle permet de décider si lévénement est réalisé ou non lors de l'expérience aléatoire.
Lorsque $\omega \in \mathcal{E}$ on a $\mathbb{1}_{\{\mathcal{E}\}}(\omega) = 1$ et on dit que $\mathcal{E}$ est réalisé. Lorsqu'au contraire $\omega \notin \mathcal{E}, \mathbb{1}_{\{\mathcal{E}\}}(\omega) = 0$ et on dit que $\mathcal{E}$ n'est pass réalisé.
Côté informatique, évaluer $\mathcal{E}$ consiste à effectuer un test, i.e. renvoyer le booléen `T` (True) si l'événement est réalisé, et `F` (False) sinon. Comme on va le voir, du point de vue informatique, `T`, est associé, et même en fait souvent confondu, avec le nombre 1, tandis que `F` est associé/confondu avec 0.
Dans l'exemple qui nous intéresse notre expérience aléatoire est le tirage du nombre `rand()` $\in [0,1]$, on peut *par exemple* tester si ce nombre est inférieur ou égal à 0.5, ou encore s'il appartient à l'intervalle $(0.3, 0.6]$:

$$\mathcal{E}_1(\texttt{rand()}) = \{\texttt{rand()} \leq 0.5\}, \qquad \mathcal{E}_2(\texttt{rand()}) = \{0.3 < \texttt{rand()} \leq 0.6\}.$$

Le langage scilab est assez pratique pour évaluer les réalisations d'un événement, e.g. $\mathcal{E}_1$, lors de la réalisation de plusieurs expériences. En effet la commande `[x<=0.5]` permet d'effectuer d'un seul coup le test ≤ 0.5 pour chacune des entrées de la liste, du vecteur, ou de la matrice x :

```
-->x=rand(1,10)
 x =
  0.2922267  0.5664249  0.4826472  0.3321719  0.5935095 0.5015342
0.4368588  0.2693125  0.6325745  0.4051954
-->[x <= 0.5]
 ans =
  T F T T F F T T F T

-->x=rand(5,5)
 x =
```

```
  0.9184708  0.2806498  0.6856896  0.4094825  0.5896177
  0.0437334  0.1280058  0.1531217  0.8784126  0.6853980
  0.4818509  0.7783129  0.6970851  0.1138360  0.8906225
  0.2639556  0.2119030  0.8415518  0.1998338  0.5042213
  0.4148104  0.1121355  0.4062025  0.5618661  0.3493615
-->[x <= 0.5]
 ans =
  F  T  F  T  F
  T  T  T  F  F
  T  F  F  T  F
  T  T  F  T  F
  T  T  T  F  T
```

La proportion d'expériences pour lesquelles un événement $\mathcal{E}$ a été réalisé, lors de plusieurs réalisations indépendantes d'une même expérience est appelée *fréquence empirique* de $\mathcal{E}$. Plus précisément, si $(x_1, ..., x_n)$ sont n réalisations de `rand()`, la fréquence empirique de $\mathcal{E}$(`rand()`) est définie par

$$f_n(\mathcal{E}) := \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{\mathcal{E}(x_i)=T\}}$$

Là encore, scilab permet d'écrire les choses de façon très succinte : en effet si `M` est une matrice contenant des booléens, `sum(M)` compte le nombre de valeurs `T` apparaissant dans `M`. Pour l'exemple ci-dessus de `x` tableau $5 * 5$ (soient 25 réalisations de l'événement $\mathcal{E}$) on a la fréquence empirique :

```
-->sum(x)/25
 ans =
  0.4640052
```

De façon plus intéressante, on peut vérifier que la fréquence empirique de l'événement $\mathcal{E}_1$ semble se concentrer autour de la valeur 0.5 lorsqu'on réalise un grand nombre d'expériences :

```
-->sum(rand(1:10^3)<=0.5)/10^3
 ans =
  0.52
-->sum(rand(1:10^4)<=0.5)/10^4
 ans =
  0.4976
-->sum(rand(1:10^5)<=0.5)/10^5
 ans =
  0.5026
-->sum(rand(1:10^6)<=0.5)/10^6
 ans =
  0.500427
```

Notons de plus que la valeur de la fréquence empirique semble être d'autant plus proche de 0.5 que le nombre d'expériences est élevé.

`scilab` se met à râler lorsque les tailles de vecteurs atteignent 10^7, on peut cependant aller un peu plus loin grâce à la commande `stacksize('max')` :

```
-->sum(rand(1:10^7)<=0.5)/10^7
 !--error 17
Taille de la pile dépassée
Utilisez la fonction stacksize pour l'augmenter.
Mémoire utilisée pour les variables : 12730
Mémoire intermédiaire requise : 10000001
Mémoire totale disponible : 10000000
-->stacksize('max')
-->sum(rand(1:10^7)<=0.5)/10^7
 ans =
  0.4997761
-->sum(rand(1:5*10^7)<=0.5)/(5*10^7)
 ans =
  0.4999548
```

On peut effectuer plusieurs fois $5 * 10^7$ tirages de `rand()` pour se rendre compte que la fréquence empirique obtenue varie légèrement mais reste systématiquement proche de 0.5 :

```
 -->sum(rand(1:5*10^7)<=0.5)/(5*10^7)
 ans =
  0.4999483
-->sum(rand(1:5*10^7)<=0.5)/(5*10^7)
 ans =
  0.5001604
-->sum(rand(1:5*10^7)<=0.5)/(5*10^7)
 ans =
  0.5001188
-->sum(rand(1:5*10^7)<=0.5)/(5*10^7)
 ans =
  0.4999207
```

On peut de manière similaire évaluer la fréquence empirique de $\mathcal{E}_2$, mais on fera attention que $\mathcal{E}_2$ étant la conjonction des deux conditions $\mathcal{E}_2 = \{\texttt{rand()} > 0.3\} \cap \{\texttt{rand()} \leq 0.6\}$, l'écriture scilab devient légèrement plus subtile :

```
-->x=rand(1:10)
 x =
  0.3849996  0.5222339  0.4530201  0.8954992  0.7741357  0.3551975
0.6371357  0.6708531  0.1094476  0.3772180
-->y=[[x>0.3];[x<=0.6]]
 y =
  T  T  T  T  T  T  T  T  F  T
  T  T  T  F  F  T  F  F  T  T
-->z=and(y,'r')
```

```
 z =
  T  T  T  F  F  T  F  F  F  T
-->sum(z)/10
 ans =
0.5
-->x=rand(1:10^7);
-->y=[[x>0.3];[x<=0.6]];
-->z=and(y,'r');
-->sum(z)/10^7
 ans =
  0.3001655
-->x=rand(1:10^7);
-->y=[[x>0.3];[x<=0.6]];
-->z=and(y,'r');
-->sum(z)/10^7
 ans =
  0.3000544
```

En théorie, la taille d'un événement peut par ailleurs être mesurée à l'aide d'une *mesure de probabilité* $\mathbb{P}$ (i.e. une mesure positive, de masse totale 1). Naturellement, l'ensemble des événements que l'on pourra mesurer va constituer une tribu, que l'on note $\mathcal{F}$ et que l'on suppose systématiquement complétée.
Pour l'instant, on va tout simplement accepter que la mesure de cette fréquence empirique pour un grand nombre de tirages de (rand()) nous permet d'évaluer la probabilité d'événements associés.
Dans le cadre de nos exemples, nos simulations suggèrent donc que

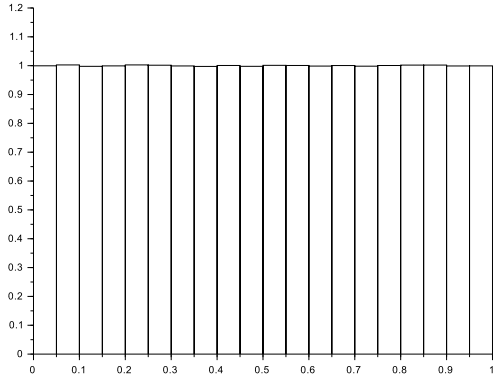
$$\mathbb{P}(\mathcal{E}_1(\texttt{rand()})) = \mathbb{P}(\texttt{rand()} \leq 0.5) \approx 0.5 \qquad \mathbb{P}(\mathcal{E}_2(\texttt{rand()})) = \mathbb{P}(0.3 < \texttt{rand()} \leq 0.6) \approx 0.3.$$

1.4 Histogrammes

Pour bien comprendre les propriétés de `rand()`, on aimerait pouvoir évaluer d'un seul coup d'oeil les fréquences empiriques d'une large collection d'événements.
L'idée est de découper $[0,1]$ en K "petits" intervalles adjacents $I_1, ...I_K$, et de compter, sur n réalisations indépendantes de `rand()`, combien de réalisations tombent dans chacun de ces intervalles. Par défaut, `scilab` choisit de diviser ce nombre de réalisations par n*(la taille de l'intervalle correspondante).
Précisément, pour un vecteur `x=rand(1:n)` et une *subdivision* `a=[`$a_0 \ \cdots a_K$`]` de l'intervalle $[0,1]$: $a_0 = 0 < a_1 < \cdots < a_{n-1} < a_K = 1$, la commande `--> histplot(a,x)` va renvoyer

— une liste de taille K dont la i-ème entrée est

$$H_i := \frac{\#\{k \in \{1,...,n\} : x(k) \in [a_{i-1}, a_i]\}}{n(a_i - a_{i-1})}, i \in \{1,...,K\}$$



— un graphe, où pour tout $i \in \{1, ..., K\}$ on voit un rectangle de base $[a_{i-1}a_i]$ et de hauteur H_i.

On notera que le choix de la renormalisation fait que l'aire du i-ème rectangle correspond exactement à la fréquence empirique (proche de la probabilité lorsque $n >> K$) de l'événement

$$\mathcal{E}_i := \{\texttt{rand()} \in [a_{i-1}, a_i]\}.$$

En particulier, la somme des aires des rectangles vaut toujours 1.

Très souvent, on se contente de prendre une subdivision régulière de l'intervalle $[0, 1]$. Par exemple

```
-->x=rand(1:10^7);
-->a=[0:0.05:1];
-->histplot(a,x)
 ans =
  0.999522  1.002574  0.997698  0.999238  1.00261  1.001586  0.99904
0.99738  1.000664  0.997734  1.001152  1.000674  0.99858  1.000444
0.998244  1.000584  1.001976  1.00207  0.998972  0.999258
```

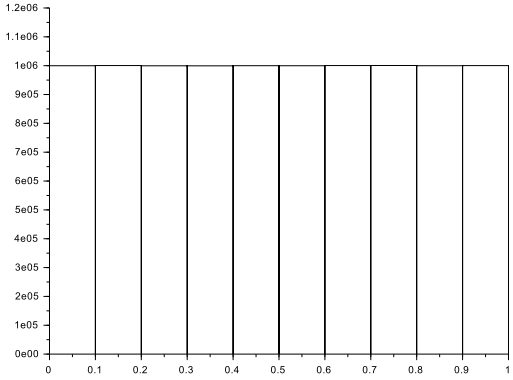
L'histogramme non normalisé est obtenu grâce à l'option `normalization=% F` de la commande `histplot`. Par exemple `-->x=rand(1:10^7);`

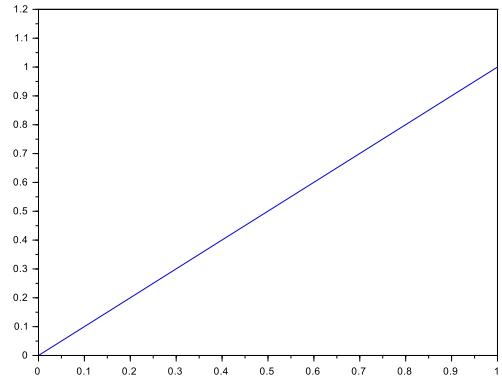
```
-->a=[0:0.1:1];
-->histplot(a,x,normalization=% F)  ans =
  999813.  1000401.  999562.  999450.  999913.  999778. 1000428.  1000589.
999837.  1000229.
```

Par exemple, la valeur 1000401 est exactement le cardinal de
$\{1 \leq i \leq n : \texttt{x(i)} \in [0.1, 0.2)\}$.

Une autre normalisation peut être intéressante, elle consiste à donner les valeurs respectives des fréquences empiriques.

```
-->x=rand(1:10^7);
-->a=[0:0.1:1];
-->histplot(a,x,normalization=%F)/10^7 ans =
```





`0.1000881 0.0999592 0.1000405 0.099835 0.0999332 0.0999883 0.1000644 0.1000009 0.1000435 0.1000469`

qui donne une figure d'allure en tous points similaire à la précédente simulation (la seule différence est l'échelle verticale).

Grâce à la commande `cumsum`, cette dernière normalisation permet de facilement simuler la *fonction de répartition empirique* de n réalisations de `rand()`.

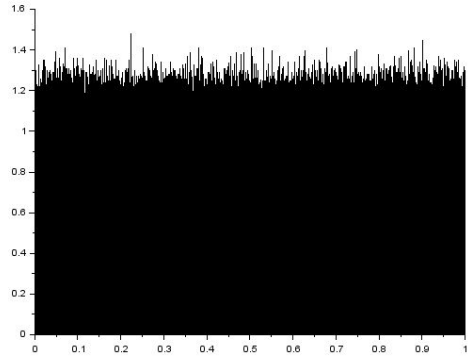
La fonction de répartition empirique d'une variable réelle associe au point $x \in \mathbb{R}$ la fréquence empirique de l'intervalle $(-\infty, x]$. Ici on peut se contenter des valeurs $x \in [0,1]$ puisque `rand()` $\in [0,1]$: `-->x=rand(1:10^7);`

`-->a=[0:0.001:1];`

`-->h=histplot(a,x,normalization=%F)/10^7; -->F=cumsum(h); -->plot(a,F)`

Au vu de ce dernier graphe, il semble raisonnable d'affirmer que la fonction de répartition empirique d'un grand nombre de tirages de `rand` est très proche de la fonction identité sur l'intervalle $[0,1]$.

Le lien entre la fonction de répartition empirique `F` et l'histogramme normalisé est donc le suivant : `F(x)` est la somme des aires des rectangles entre 0 et x. Si elle existe, définissons



la fonction f dont les valeurs seraient la limite, lorsque $n \to \infty$ du bord supérieur des rectangles de l'histogramme. D'après notre raisonnement on aurait donc, pour tout $x \in [0,1]$,

$$F(x) = \int_0^x f(u)du \qquad f(x) = F'(x).$$

Ici nos simulations suggèrent que $f(x) = 1$ pour tout $x \in [0,1]$
Remarque : Pour estimer convablement la fonction de répartition empirique il convient de choisir une subdivision assez fine de l'intervalle $[0,1]$.
On pourrait croire que de façon similaire, plus la subdivision choisie est fine, plus l'histogramme obtenu donne des informations précises mais ce n'est pas tout à fait vrai : il faut respecter la contrainte $K << n$ de sorte que les fréquences empiriques restent proches de la probabilité de tomber dans l'un des intervalles. Un découpage trop fin va engendrer des variations trop importantes de ces fréquences, rendant l'histogramme obtenu moins lisible, voire trompeur, nous empêchant de deviner f :

```
-->x=rand(1:10^7);
-->a=(0:0.00001:1);
-->h=histplot(a,x);
```

fournit la figure ci-dessus,
pour laquelle on pourrait croire que les valeurs des bords supérieurs des rectangles se situent en moyenne autour de 1.25. En fait il n'en est rien, si on regarde par exemple les 20 première valeurs de ces fréquences renormalisées on constate qu'elles oscillent autour de 1. C'est l'echelle de la figure qui fait qu'on voit surtout apparaître les maximas (on peut d'ailleurs utiliser la fonction zoom des figures de `scilab` pour s'en rendre compte).

```
 -->h(1:20)
 ans =
  column 1 to 10
  1.01 1.06 0.91 1.04 0.96 1.01 0.9 0.98 1.11 1.02
  column 11 to 20
  0.91 0.89 0.98 1.05 0.99 1.06 1. 1.23 1.1 1.06
```

1.5 Loi de `rand()`

Nos histogrammes et notre simulation de la fonction de répartition empirique suggèrent que pour tout intervalle $I \subset [0,1]$, de longueur $|I|$, on a

$$\mathbb{P}(\texttt{rand()} \in I) \approx |I|.$$

A ce stade, laissons tomber les approximations pour écrire

$$\mathbb{P}(\texttt{rand()} \in I) := |I|.$$

Nous sommes en mesure de décrire la *loi* de la variable $X = \texttt{rand()}$. Comme X est à valeurs entre 0 et 1, on a plus généralement que pour tout intervalle $I \subset \mathbb{R}$:

$$\mathbb{P}(X \in I) = |I \cap [0,1]|.$$

Puisque $\mathbb{P}$ est une mesure, on voit qu'on peut sans mal étendre à tout borélien de $B \subset \mathbb{R}$:

$$\mathbb{P}(X \in I) = \lambda(|B \cap [0,1]|),$$

où λ est la mesure de Lebesgue.
La mesure-image de $\mathbb{P}$ par l'application $X : \Omega \to \mathbb{R}$ est la mesure

$$\mathbb{P}_X(A) = \mathbb{P}(X \in A)$$

C'est une mesure de probabilité sur les boréliens de $\mathbb{R}$, et ici il s'agit tout simplement la mesure de Lebesgue restreinte à l'intervalle $[0,1]$. C'est précisément cette mesure $\mathbb{P}_X$ qu'on appelle la *loi* de X. Ici on dit que cette loi est *uniforme* sur $[0,1]$, et on vient de détailler le fait que $X \sim \text{Unif}[0,1]$.
La fonction

$$F_X(x) = \mathbb{P}(X \leq x) = \begin{cases} 0 \text{ si } x < 0 \\ x \text{ si } x \in [0,1] \\ 1 \text{ si } x > 1 \end{cases},$$

est la fonction de répartition de la variable X. C'est bien la limite lorsque $n \to \infty$ de la fonction de répartition empirique `F` décrite plus haut.
La fonction $f_X(x) = \mathbb{1}_{[0,1]}(x)$ est telle que

$$F_X(x) = \int_{-\infty}^{x} f_X(u)du, \quad \forall x \in \mathbb{R}.$$

En tout point x où F_X est dérivable, $f_X(x) = F_X'(x)$. On dit que f_X est la *densité*[3] de X. Ici la densité est constante sur $[0,1]$, reflétant le fait que notre variable est uniforme sur cet intervalle.

3. intuitivement, en tout point de continuité de f_X on a $f_X(x)dx = \mathbb{P}(X \in (x - dx/2, x + dx/2))$, c'est pourquoi on parle de densité de probabilité

1.6 Quantités moyennées : un cadre un peu plus général

Outre les fréquences empiriques d'événements, de nombreuses autres quantités moyennées vérifient la propriété (ii). C'est par exemple le cas de la *moyenne empirique* :

```
 -->sum(rand(1:10^7))/10^7
 ans =
  0.4999098
-->sum(rand(1:10^7))/10^7
 ans =
  0.4998954
```

C'est aussi le cas de la *variance empirique* :

```
 -->x=rand(1:10^7);
-->m=sum(x)/10^7;
-->sum((x-m).^2)/10^7
 ans =
  0.0833371
-->x=rand(1:10^7);
-->m=sum(x)/10^7;
-->sum((x-m).^2)/10^7
 ans =
  0.0833164
```

De façon plus générale, pour une fonction $f : [0,1] \to \mathbb{R}$ et des réalisations $(x_1, ..., x_n)$ de la fonction `rand()`, on peut calculer la valeur moyenne de $f($`rand()`$)$ via

$$\frac{1}{n}\sum_{i=1}^{n} f(x_i).$$

On va considérer deux exemples de fonctions de $[0,1]$ à valeurs réelles :

$$f_1 : \begin{cases} [0,1] \to [0,1] \\ x \to x^{10} \end{cases}, \qquad f_2 : \begin{cases} (0,1] \to [0,1] \\ x \to 1/x \end{cases}, \ f_2(0) = 0,$$

que l'on peut implémenter en `scilab` de la façon suivante (on utilise un fichier `SciNotes` pour définir les fonctions proprement) :

```
 function y=f1(x)
y=x^10;
endfunction

function y=f2(x)
if x==0 then y=0;
else y=1/x;
end
endfunction
```

Après avoir executé le fichier, on peut revenir à la console `scilab`, nos fonctions `f1` et `f2` sont désormais bien définies :

```
-->f1(0.5)
 ans =
  0.0009766
-->f2(0.5)
 ans =
  2.
```

On peut simultanément évaluer `f1` ou `f2` sur les coordonnées d'un vecteur à l'aide de la commande `feval` :

```
 -->x=(0:0.1:1);
-->feval(x,f1)
 ans =
  0.  1.000D-10  0.0000001  0.0000059  0.0001049  0.0009766  0.0060466
0.0282475  0.1073742  0.3486784  1.
-->feval(x,f2)
 ans =
  0.  10.  5.  3.3333333  2.5  2.  1.6666667  1.4285714  1.25  1.1111111
1.
```

Il est donc aisé d'obtenir la valeur moyenne de `f1` pour 10^7 réalisations de `rand()` :

```
-->x=rand(1:10^7);
-->y=feval(x,f1);
-->sum(y)/10^7
 ans =
  0.0908378
```

On peut répéter cette procédure plusieurs fois, on trouve sysématiquement un résultat proche de $1/11 \approx 0.0909.$:

```
-->x=rand(1:10^7);
-->y=feval(x,f1);
-->sum(y)/10^7
 ans =
  0.0908723
-->x=rand(1:10^7);
-->y=feval(x,f1);
-->sum(y)/10^7
 ans =
  0.0909280
```

Faisons la même chose avec la fonction `f2` : cette fois le résultat ne semble pas rester proche d'une valeur fixée, et reste essentiellement imprévisible :

```
-->x=rand(1:10^7);
-->y=feval(x,f2);
-->sum(y)/10^7
 ans =
```

```
  17.658862
-->x=rand(1:10^7);
-->y=feval(x,f2);
-->sum(y)/10^7
 ans =
  13.631868
-->x=rand(1:10^7);
-->y=feval(x,f2);
-->sum(y)/10^7
 ans =
  28.074823
```

Remarque : La différence fondamentale entre ces deux exemples est que $\mathbb{E}[\texttt{f1(rand())}] = 1/11$, de sorte que la loi forte des grands nombres assure que $\left(\frac{1}{n}\sum_{i=1}^{n} f_1(x_i)\right)_n$ converge p.s. vers $\mathbb{E}[\texttt{f1(rand())}]$, tandis que $\mathbb{E}[\texttt{f2(rand())}] = +\infty$ de sorte qu'on ne peut pas appliquer la loi forte des grands nombres dans ce deuxième cas. En réalité pour cet exemple précis, si les x_i sont i.i.d suivant la loi uniforme$[0,1]$, on pourrait vérifier (on y reviendra plus tard dans le module Probabilités) que la suite $\left(\frac{1}{n}\sum_{i=1}^{n} \frac{1}{x_i}\right)_n$ ne converge pas vers une quantité déterministe, au lieu de ça elle converge en loi vers une variable limite non dégénérée.

2 Simulation d'une expérience aléatoire à partir de `rand()` : quelques exemples.

2.1 Simulation du jeu de pile ou face, pièce équilibrée

Lors d'un jet d'une pièce équilibrée, on obtient Pile avec probabilité 1/2, et Face avec probabilité 1/2. Dans la suite, pour simplifier un peu, on va "coder" Pile par 1 et Face par 0.
L'idée est donc de fabriquer un événement $\mathcal{E}(\texttt{rand()})$ de probabilité 1/2 pour simuler ce jet de pièce. Il y a en fait tout un tas de façons de le faire, l'une des plus simples est par exemple de choisir

$$\mathcal{E}(\texttt{rand()}) = \{0 \leq \texttt{rand()} < 0.5\}.$$

Avec ce choix, un tirage de `rand()` qui donne un nombre < 0.5 correspond à un jet Pile, un tirage de `rand()` qui donne un nombre ≥ 0.5 correspond à un jet Face.
La commande `rand()` $\leq p$ renvoie T ou F suivant que l'événement $\mathcal{E}$ est réalisé ou non. En fait `scilab` ne fait presque aucune différence entre les booléens T, F et les flottants $1., 0$. On passe donc très facilement de booléens aux flottants [4], par exemple de la façon

4. il aurait été également tout à fait possible, à l'aide de la commande `disp`, de renvoyer la chaîne de caractères "pile" lorsque l'événement est réalisé (par exemple en écrivant `if rand()<0.5 then disp('pile'); else disp('face'); end`), mais cette manipulation ne fait que ralentir les choses et ne présente donc pas vraiment d'intérêt.

suivante :

```
-->x=rand()<0.5
 x =
  F
-->2*x/2
 ans =
  0.
```

On peut donc simuler facilement n réalisations indépendantes de jets de P/F (dans l'exemple ci-dessous $n = 5$) :

```
-->x=[rand(1:5)<0.5]
 x =
  T  F  F  F  T
-->2.*x/2
 ans =
  1.  0.  0.  0.  1.
```

On pourrait également suivre l'évolution de la fortune d'un joueur de P/F (qui dans l'exemple ci-dessous, joue n parties, parie systématiquement sur le pile à chaque partie, gagne 1 euro par pari gagné, et perd 1 euro par pari perdu). On représente alors par un graphe l'évolution des gains de ce joueur, à l'aide du programme suivant :

```
-->x=[rand(1:n)<0.5];
--> y = 2*x -1;
--> z = cumsum(y); --> s = [0 z]; --> plot((0:n),s);
```

Pour $n = 10, 100, 1000, 10^4, 10^5, 10^6$, on obtient (par exemple) les figures suivantes

2.2 Simuler un jeu de roulette

On considère une roulette à 38 cases, dont 18 rouges, 18 noires et 2 vertes. On estime que la bille s'arrête sur l'une des 38 cases de manière équiprobable.
Pour simuler la couleur obtenue lors d'un lancer de la bille, on peut par exemple considérer les trois événements suivants :

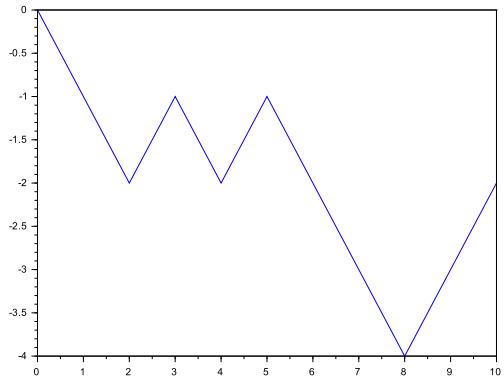
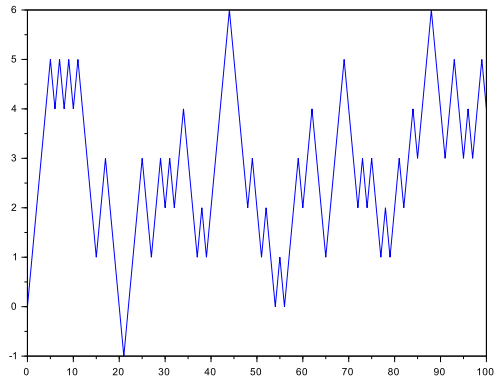
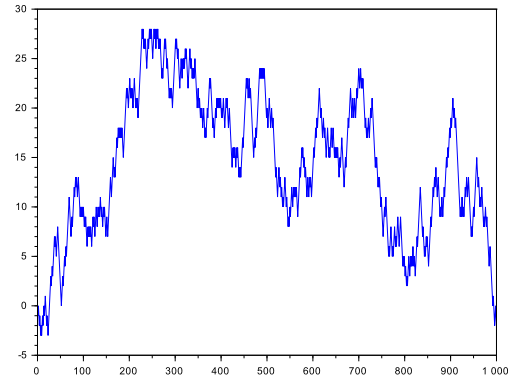
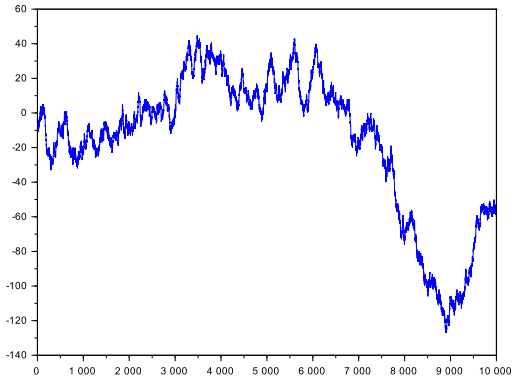
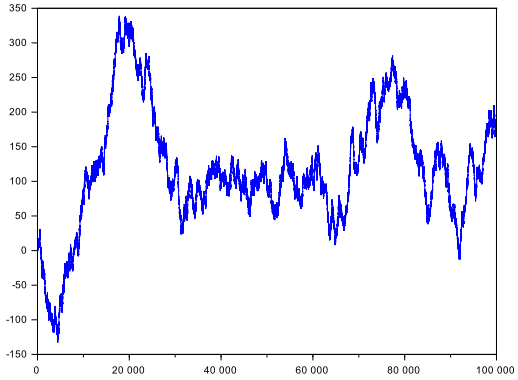
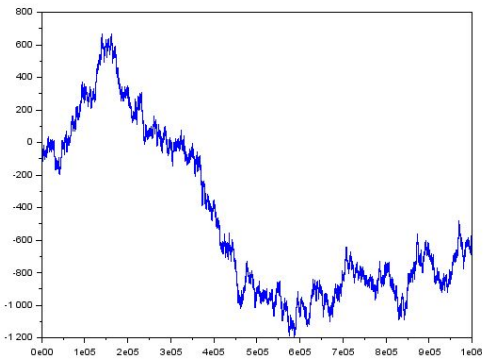
$$\mathcal{E}_1 := \{\texttt{rand()} < 1/19\}, \quad \mathcal{E}_2 := \{1/19 \leq \texttt{rand()} < 10/19\}, \quad \mathcal{E}_3 := \{10/19 \leq \texttt{rand()}\}.$$

Avec ce choix, pour un tirage de `rand()`, réaliser $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3$ correspond, resp., à ce que la bille s'arrête sur une case de couleur verte, rouge, noire, resp.
Considérons alors l'événement

$$\mathcal{E} := \{1 \text{ rouge, } 1 \text{ noire, } 1 \text{ verte lors des } 3 \text{ premiers lancers}\}.$$

Notons que si on vérifie qu'on a obtenu 1 verte et 1 noire lors des 3 premiers lancers, la couleur manquante est forcément rouge.
Grâce au programme suivant, on estime la fréquence empirique de l'événement $\mathcal{E}$, pour n répétitions de 3 lancers.

```
function e=estimeroulette(n)
```

```
 x=rand(n,3);
 y=[x<1/19];
 z=[x>=10/19];
 r=[sum(y,'c')==1];
 s=[sum(z,'c')==1];
 t=[r+s==2]
 e=sum(t)/n;
endfunction
```

Note : le tableau `y` permet de tester lesquels des lancers correspondent à la couleur verte, `z` lesquels correspondent à la couleur noire. On notera que les lancers rouges correspondent à `ones(n,3)-y-z`.
Le vecteur `r` teste pour chaque groupe de 3 lancers si le nombre de verts obtenu est égal à 1, `s` fait de même pour les noirs. Le vecteur `t` teste donc si on obtient exactement 1 vert et 1 noir (et donc l'événement $\mathcal{E}$) pour chaque groupe de 3 lancers.
On obtient par exemple, pour 2 lancements du programme avec $n = 10^7$:

```
-->estimeroulette(10000000)
ans =
0.0708262
-->estimeroulette(10000000)
ans =
0.070949
```

de sorte que

$$\mathbb{P}(\mathcal{E}) \approx 0.071.$$

Exercice : Vérifier par un calcul direct la validité de cette approximation.

2.3 Simuler le conditionnement : méthode du rejet

Conditionner à un événement $\mathcal{E}$ revient à restreindre l'ensemble des $\omega \in \Omega$ que l'on peut choisir à ceux qui réalisent $\mathcal{E}$.
En pratique pour simuler le conditionnement, on va se contenter de rejeter les expériences qui ne réalisent pas cet événement.
La méthode la plus simple consiste à utiliser la fonction `find`, qui permet de dégager les coordonnées d'une liste qui vérifient un test.
Voici un exemple avec $\mathcal{E} = \{$`0.3 < rand() < 0.6`$\}$, et n réalisations de `rand()` :

```
function r=rejet1(n)  x = rand(1:n);
 y=and([x>0.3; x<0.6], 'r');  z=find(y);
 r=x(y);
endfunction
```

Le seul petit désagrément est que la taille du vecteur `r` obtenu ci-dessus n'est pas toujours la même : elle vaut précisément le nombre de fois parmi les n où l'événement $\mathcal{E}$ a été réalisé.
Ainsi : `-->r=rejet1(1000000);`

```
-->length(r)
 ans =
  299276.
-->r=rejet1(1000000);
-->length(r)
 ans =
  300079.
```

Etant donné que dans notre exemple $\mathbb{P}(\mathcal{E}) = 0.3$, il est normal que sur 10^6 essais, environ 300000 réalisent l'événement $\mathcal{E}$, c'est pourquoi la longueur du vecteur `r` obtenu reste proche de 300000.
Ceci étant dit, chaque coordonnée du vecteur `r` obtenu est bien une réalisation de `rand()` conditionnée à se trouver dans l'intervalle $(0.3, 0.6)$. On peut par exemple lister les 10 premières coordonnées de `r` : `-->r(1:10)`

```
 ans =
  0.4798563  0.5181523  0.5836449  0.5009841  0.5315327  0.5715924
0.4487466  0.4312423  0.3099710  0.4855518
```

Si on veut simuler exactement k réalisations de la variable conditionnée, on peut s'y prendre de deux manières : on en simule volontairement un peu trop (typiquement on peut faire $n = 2k/\mathbb{P}(\mathcal{E})$ essais pour être très confiant qu'on va obtenir au moins k réalisations de $\mathcal{E}$ — pour k très grand, on peut se contenter par exemple de $n = 1.1k/\mathbb{P}(\mathcal{E})$). Si jamais la longueur de la liste obtenue est malgré tout trop petite on recommence. Dès qu'on a une liste suffisamment grande on la restreint à ses k premières coordonnées.
Exemple avec $k = 100$: `-->r=rejet1(200/0.3); while length(r)<100 do r=rejet1(200/0.3); end`
`--> s=r(1:100);`
Une troisième méthode est de mettre en place un compteur et d'utiliser une boucle. La deuxième méthode présente le désavantage d'être beaucoup plus lente puisqu'elle fait intervenir une boucle de longueur approximative $k/\mathbb{P}(\mathcal{E})$. Un léger [5] avantage est qu'elle ne nécessite pas la connaissance préalable de $\mathbb{P}(\mathcal{E})$.
Toujours pour le même exemple, pour mettre en place cette troisième méthode on peut par exemple utiliser le programme

```
function r=rejet2(k)
 c=1; z=[];
 while c<=k do
  x= rand();
  if and([x > 0.3 x< 0.6]) then z(c)=x; c=c+1;
  end
 end
endfunction
```

5. l'avantage peut sembler important au premier abord, mais en réalité la longueur de la liste obtenue par la première méthode permet justement d'estimer $\mathbb{P}(\mathcal{E})$...

3 Simuler des v.a.r discrètes

3.1 Simuler les variables usuelles discrètes à partir de `rand`

3.1.1 Ber(p)

On dit que X suit une loi de *Bernoulli* de paramètre $p \in [0,1]$ et on note $X \sim \text{Ber}(p)$ ssi

$$\mathbb{P}(X=0) = 1-p, \quad \mathbb{P}(X=1) = p.$$

La loi de Bernoulli dépasse très largement l'exemple classique de "Pile ou Face" :

Lemme 3.1. *Soit $(\Omega, \mathcal{F}, \mathbb{P})$ un espace probabilisé et $\mathcal{E} \in \mathcal{F}$. Notons $p = \mathbb{P}(\mathcal{E}) \in [0,1]$. Alors $\mathbb{1}_{\mathcal{E}} \sim \text{Ber}(p)$.*

Pour simuler une variable de Bernoulli de paramètre $p \in [0,1]$ à partir de `rand`, il suffit donc de trouver un événement $\mathcal{E}$(`rand()`) de probabilité p. Le plus simple est comme dans le paragraphe précédent, de choisir

$$\mathcal{E}(\texttt{rand()}) = \{0 \leq \texttt{rand()} \leq p\}$$

Pour $n, m \in \mathbb{N}, p \in [0,1]$, la fonction suivante permet de simuler un tableau de taille $n \times m$ de variables i.i.d, $\sim \text{Ber}(p)$:

```
 function b=ber(n,m,p)
 x=rand(n,m);
 b=2.*[x<p]/2;
endfunction
```

Par exemple, pour $n=3, m=5, p=0.4$, on obtient :

```
-->ber(3,5,0.4)
 ans =
  1.  1.  1.  0.  1.
  1.  0.  0.  1.  0.
  0.  0.  1.  1.  0.
```

3.1.2 Unif$\{1,...,k\}$

On dit que $X \sim \text{Unif}\{1,...,k\}$ ssi

$$\mathbb{P}(X=i) = 1/k, \quad \forall i \in \{1,...,k\}.$$

Il est facile de simuler une liste de telles variables à l'aide d'une liste de tirages de `rand()` et de la fonction `ceil` qui arrondit à l'entier supérieur :

```
function u=uniforme(n,k)
x = rand(1:n); u=ceil(k*x);
```

Par exemple, pour $k=10, n=5$, `-->x=rand(1:10); ceil(5*x)`

```
 ans =
```

```
  1.  3.  2.  3.  3.  1.  3.  4.  5.  1.
```

Note : De façon plus générale, si `x` correspond à n réalisations indépendantes d'une loi uniforme sur $\{1, ..., k\}$, alors `a.*x+b` correspond à n réalisations indépendantes de choix uniformes parmi $\{b + a, b + 2a, ..., b + ka\}$.

3.1.3 Bin(n, p)

Si $(X_1, ..., X_n)$ sont des variables i.i.d. $\sim$ Ber(p), alors $S_n := \sum_{i=1}^{n} X_i \sim$ Bin(n, p), et on dit que S_n suit la loi *binômiale* de paramètres n, p.
La simulation d'une variable binômiale en `scilab` à partir de `rand()` est donc également très facile :

```
-->s=sum(ber(1,10,0.6))
 s =
  5.
```

Plus généralement, grâce à la fonction `ber` définie précédemment, la fonction suivante permet de simuler une liste de taille m de variables i.i.d, $\sim$ Bin(n, p) :

```
function s=bin(m,n,p)
 s=sum(ber(n,m,p),'r')
endfunction
-->bin(5,10,0.7)
 ans =
  6.  8.  6.  7.  6.
```

3.1.4 Geom(p)

Si $(X_i, i \geq 1)$ est une suite de variables i.i.d $\sim$ Ber(p), l'indice de premier succès $G = \inf\{i : X_i = 1\} \sim$ Geom(p), et on dit que G suit une loi *géométrique* de paramètre p.
A l'aide de la commande `while`, il est facile de simuler une telle variable à partir de réalisations successives de `rand()` :

```
function g=geo(p)
 g=1;
  while rand()>p do g=g+1;
  end
 g
endfunction
```

Par exemple, pour $p = 0.1$,

```
-->geo(0.1)
 ans =
  2.
-->geo(0.1)
 ans =   27.
-->geo(0.1)
 ans =
```

```
  7.
-->geo(0.1)
 ans =
  4.
```

Pour simuler une liste de taille n de variables i.i.d $\sim \text{Geom}(p)$, on peut effectuer une boucle :

```
function g=geom(n,p)
 g=ones(1:n);
  for i=1:n do
   while rand() >p do
    g(i)=g(i)+1;
   end
  end
 g
endfunction
```

Par exemple pour $n = 20, p = 0.1$, on obtient

```
-->geom(20,0.1)
 ans =   13.  3.  8.  6.  5.  10.  25.  2.  5.  3.  8.  6.  4.  4.  10.
3.  1.  8.  4.  28.
```

Il est par ailleurs immédiat que

$$\forall k \in \mathbb{N} \quad \mathbb{P}(G > k) = \mathbb{P}(X_i = 0, i \in \{1, ..., k\}) = (1-p)^k,$$

et donc

$$\forall k \in \mathbb{N}^* \quad \mathbb{P}(G = k) = \mathbb{P}(G > k-1) - \mathbb{P}(G > k) = p(1-p)^{k-1}.$$

On peut donc, de façon alternative, directement simuler une variable géométrique à partir d'une seule réalisation de `rand()`. L'idée est de considérer la suite d'intervalles (formant une partition de $[0, 1]$)

$$I_1 = [a_0, a_1] = [0, p], \quad I_2 = (a_1, a_2] = (p, p + p(1-p)], \cdots$$
$$I_k = (a_{k-1}, a_k] = (1 - (1-p)^{k-1}, 1 - (1-p)^k], \cdots,$$

et la variable G égale à l'indice de l'intervalle dans lequel se trouve $X =$ `rand()` :

$$G = k \text{ tel que } X \in I_k, \text{ i.e. } G = \min\{k : X \leq a_k\} = \max\{k : X > a_{k-1}\}$$

Comme il y a une infinité d'intervalles, on n'échappera cependant pas à l'implémentation d'une boucle `while` :

```
function g=geo2(p)
 x=rand(); a=p; g=1;
  while x>a do
   g=g+1;
   a=1-(1-p)^g;
  end
```

```
 g
endfunction
```
Lorsqu'on veut alors simuler n variables géométriques, on peut alors effectuer par exemple
```
function g=geom2(n,p)
 x=rand(1:n); m=max(x); a=p; g=ones(1:n); k=1;
  while m>a do
   g=g+[x>a];
   k=k+1;
   a=1-(1-p)^k;
  end
 g
endfunction
```
La fonction `geom2` présente l'avantage non négligeable d'être plus rapide que `geom` définie plus haut.

3.1.5 $\mathcal{P}(\lambda)$

La variable X suit la loi de Poisson de paramètre λ (on note $X \sim \mathcal{P}(\lambda)$) ssi

$$\forall k \in \mathbb{N}, \quad \mathbb{P}(X = k) = \frac{\lambda^k}{k!} \exp(-\lambda).$$

L'idée est la même qu'au paragraphe précédent : on considére la suite d'intervalles (formant une partition de $[0, 1]$)

$$I_0 = [a_0, a_1] = [0, \exp(-\lambda)], \quad I_1 = (a_1, a_2] = (a_1, a_1 + \lambda \exp(-\lambda)], \cdots$$
$$I_k = (a_k, a_{k+1}] = (a_k, a_k + \frac{\lambda^k}{k!} \exp(-\lambda)], \cdots,$$

et la variable Y égale à l'indice de l'intervalle dans lequel se trouve $X =$ `rand()` :

$$Y = k \text{ tel que } X \in I_k \text{ i.e. } Y = \min\{k : X \leq a_{k+1}\} = \max\{k : X > a_k\}.$$

```
function y=poiss(lambda)
 x=rand(); a=%e^(-lambda); y=0;
  while x>a do
   y=y+1;
   a=a+ %e^(-lambda)*(lambda^k)/factorial(k) ;
  end
 y
endfunction
```
Lorsqu'on veut simuler n variables de Poisson, on peut alors effectuer par exemple
```
function y=poisson(n,lambda)
 x=rand(1:n); m=max(x); a=%e^(-lambda); y=zeros(1:n); k=0;
  while m>a do
```

```
   y=y+[x>a];    k=k+1;    a=a+ %e^(-lambda)*(lambda^k)/factorial(k) ;
  end
 y
endfunction
```
Par exemple : `-->poisson(15,4)`
```
 ans =
 3.  7.  7.  3.  3.  5.  3.  4.  3.  4.  4.  2.  2.  4.  5.
```

3.2 Simuler directement ces variables à l'aide de `grand`

La fonction `grand` possède de nombreuses options, et permet de simuler directement les lois usuelles discrètes.
Les commandes `grand(k,l,"bin",1,p)`, `grand(k,l,"uin",m,M)` `grand(k,l,"bin",n,p)`, `grand(k,l,"geom",p)`, `grand(k,l,"poi",lambda)` permettent, respectivement, de simuler des tableaux de taille $k \times l$ de variables i.i.d., de loi $\text{Geom}(p), \text{Unif}\{m, ..., M\}, \text{Bin}(n, p), \text{Geom}(p), \mathcal{P}(\lambda)$ respectivement (on a utilisé que les lois $\text{Ber}(p)$ et $\text{Bin}(1, p)$ sont identiques).
Evidemment, dans la suite, on se contentera d'utiliser ces fonctions pour simuler ces variables usuelles.

3.3 Simuler une v.a.r discrète quelconque

Notre variable est discrète et à valeurs réelles, i.e. pour $(a_k, k \geq 0)$ une suite de rééls, on a

$$\forall k \in \mathbb{N}, \mathbb{P}(X = a_k) = p_k,$$

avec $\sum_{k\geq 0} p_k = 1$ (possiblement les $p_k, k \geq 0$ sont nuls à partir d'un certain rang).
Remarque : La simulation est plus rapide lorsque $(p_k)_{k\geq 0}$ est décroissante. On notera que quitte à modifier $(a_k)_{k\geq 0}$, on pourrait toujours se ramener à cette situation. Cependant on fait rarement cette modification (cf l'exemple de la variable de Poisson plus haut).
L'idée est toujours essentiellement la même que pour simuler une variable de Poisson comme plus haut : on découpe l'intervalle $[0, 1]$ en morceaux de taille $p_0, p_1, p_2, ...$ et on attribue la valeur a_k lorsque `rand()` tombe dans le k-ème morceau de l'intervalle [6]. Pour cela on utilise une boucle `while` :
```
 x=rand(); p=p(0); y=a(0); k=0;
  while x>p do k=k+1; y=y+a(k)-a(k-1) p=p+p(k)-p(k-1);
  end
disp(y);
```

6. Pour pouvoir réaliser un tel programme en pratique, il faut pouvoir écrire $a_k - a_{k-1}, p_k - p_{k-1}$ pour tout $k \geq 1$. On peut toujours le faire quand les p_k sont nuls à partir d'un certain rang. On peut aussi le faire si les suites $(a_k - a_{k-1})_{k\geq 1}, (p_k - p_{k-1})_{k\geq 1}$ s'expriment de façon explicite en fonction de k (cf les exemples des fonctions `geo2`, `poiss` décrits plus haut)

3.4 Histogrammes, Fonctions de répartition, Moments, et Fonctions génératrices des lois usuelles discrètes

3.4.1 Histogrammes

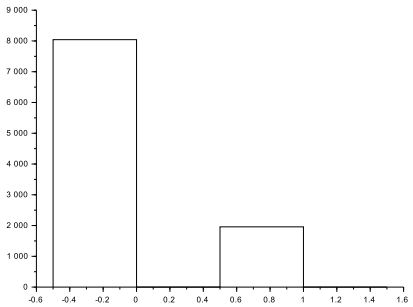
Commençons par une remarque préalable sur les histogrammes de n variables discrètes. De telles variables ne prennent leurs valeurs que dans un ensemble discret et dénombrable (le plus souvent une partie de $\mathbb{N}$).
Pour une liste $a = [a_1, ..., a_k]$ la commande `histplot(a,x, normalization=%F)` renvoie l'histogramme non renormalisé associé aux k intervalles

$$I_1 = [a_1, a_2], I_j = (a_j, a_{j+1}]j \in \{1, ...k\}.$$

Précisément, pour chaque intervalle I_j, `scilab` dessine un rectangle dont la base est I_j est la hauteur est $\#\{i : \texttt{x(i)} \in I_j\}$. On a deux problèmes, mais on peut aisément les contourner :

- pour des variables à valeurs dans un ensemble non compact, on ne sait pas a priori sur quel partie compacte de $\mathbb{R}$ représenter l'histogramme. Pour garantir de n'oublier aucune valeur [7], on peut toujours choisir $a_1 \leq \texttt{min(x)}, a_{k+1} \geq max(x)$.
- des rectangles dont la base est trop large induisent en erreur (voir les exemples plus bas) : en réalité les variables ne prennent leurs valeurs que dans un ensemble discret, et on va donc chercher à choisir $a_{j+1} - a_j$ suffisamment petit pour qu'on ne distingue plus cette épaisseur, de sorte que notre histogramme ressemblera à un diagramme en bâtons.

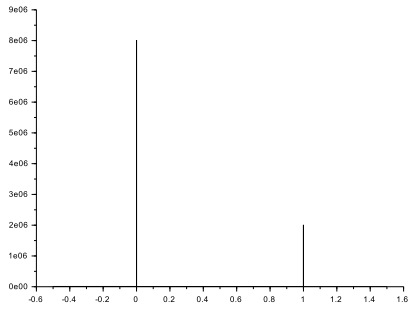
Il est facile d'obtenir un histogramme de n réalisations de variables i.i.d. $\sim \text{Ber}(p)$. La seule préoccupation est esthétique. Comme vient de le mentionner plus haut, si on prend par exemple `a=[-0.5 0 0.5 1 1.5], -->x=grand(1,10^4,"bin",1,0.2);`
`-->a=[-0.5 0 0.5 1 1.5];`
`-->histplot(a,x,normalization=%F);` fournit la première figure ci-dessous, qui n'est pas très parlante : les variables considérées semblent pouvoir prendre des valeurs négatives entre -0.5 et 0, ou des valeurs entre 0.5 et 1.



7. Si le résultat n'est pas assez lisible, on peut aussi choisir de représenter l'histogramme sur des intervalles où une grande proportion de valeurs se concentrent.

En réduisant la taille des intervalles on obtient une figure plus précise.

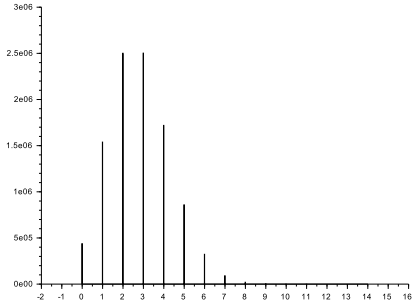
```
 -->x=grand(1,10^7,"bin",1,0.2);
-->a=(-0.01:0.001:1.01);
-->histplot(a,x,normalization=%F);
```



De la même manière on obtient facilement les histogrammes de variables $\mathrm{Bin}(n,p)$, par exemple

```
 -->x=grand(1,10m^7,"bin",14,0.2);
-->a=(-0.01:0.01:14.01);
-->histplot(a,x,normalization=%F);
```

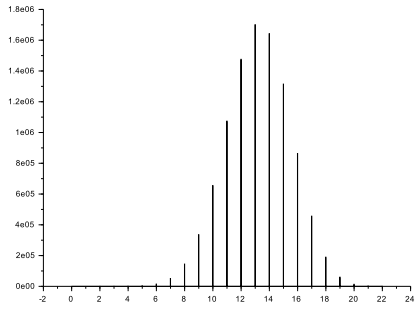
fournit



tandis que `-->x=grand(1,10^7,"bin",22,0.6);`

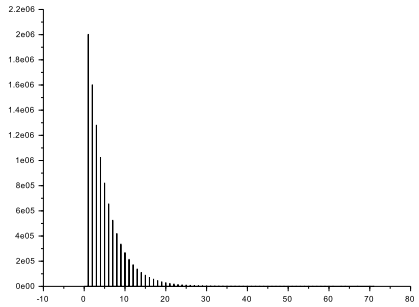
```
-->a=(-0.01:0.01:22.01);
-->histplot(a,x,normalization=%F);
```

fournit la figure

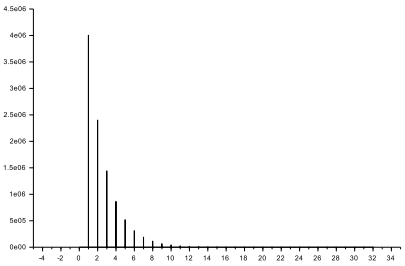


Les histogrammes de variables Geom(p) s'obtiennent de façon similaire, à ceci près qu'il peut être intéressant de représenter l'histogramme sur un intervalle incluant le maximum des valeurs obtenues :

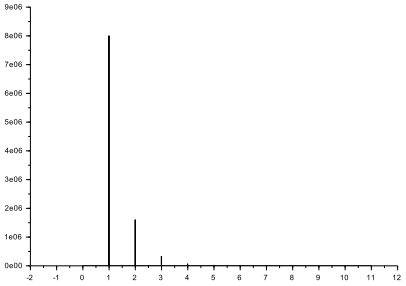
```
-->x=grand(1,10^7,"geom",0.2);
-->b=max(x)
 b =
  71.
-->a=(-0.01:0.01:b+0.01);
-->histplot(a,x,normalization=%F);
```



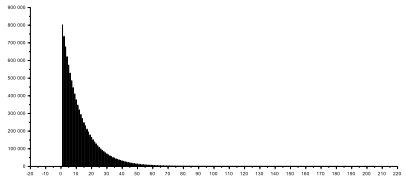

```
-->x=grand(1,10^7,"geom",0.4);
-->b=max(x)
 b =
  32.
-->a=(-0.01:0.01:b+0.01);
-->histplot(a,x,normalization=%F);
```



```
-->x=grand(1,10^7,"geom",0.8);
-->b=max(x)
 b =
  11.
-->a=(-0.01:0.01:b+0.01);
-->histplot(a,x,normalization=%F);
```



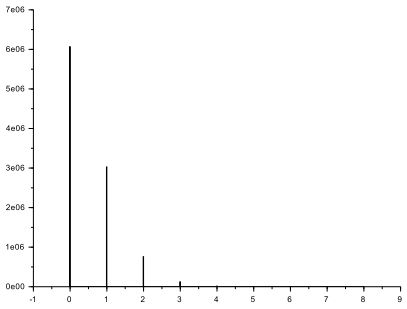
```
-->x=grand(1,10^7,"geom",0.08);
-->b=max(x)
 b =
  205.
-->a=(-0.01:0.01:b+0.01);
-->histplot(a,x,normalization=%F);
```



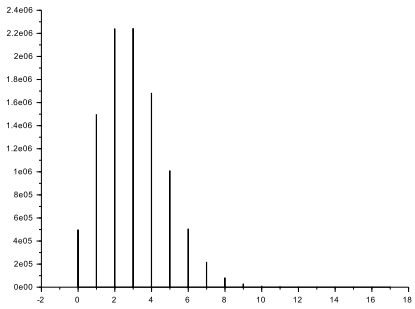
Des considérations similaires permettent de dresser les histogrammes de variables $\sim \mathcal{P}(\lambda)$.

```
-->x=grand(1,10^7,"poi",0.5);
-->b=max(x)
```

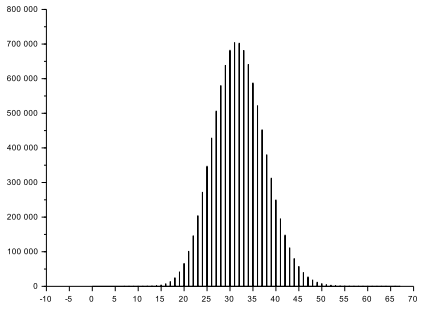
```
 b =
  8.
-->a=(-0.01:0.01:b+0.01);
-->histplot(a,x,normalization=%F);
```




```
-->x=grand(1,10^7,"poi",3);
-->b=max(x)
 b =
  17.
-->a=(-0.01:0.01:b+0.01);
-->histplot(a,x,normalization=%F);
```



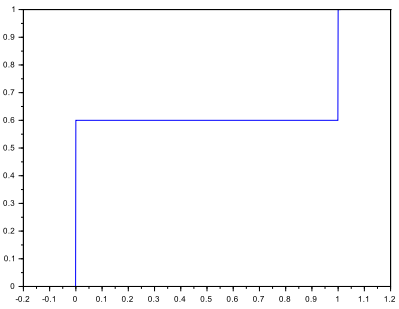

```
-->x=grand(1,10^7,"poi",32);
-->b=max(x)
 b =
  67.
-->a=(-0.01:0.01:b+0.01);
-->histplot(a,x,normalization=%F);
```



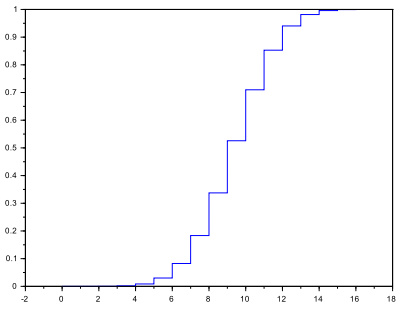
3.4.2 Fonctions de répartition

La commande `h=histplot(a,x,normalization=%F)/n` permet de lister les fréquences empiriques des intervalles $I_1, ..., I_k$. Pour évaluer la fonction de répartition empirique il suffit de réaliser les sommes cumulatives grâce à la commande `cumsum`.
Comme dans le cas de `rand` il est souhaitable de choisir un nombre k d'intervalles suffisamment grand afin d'obtenir une bonne précision pour le graphe de cette fonction de répartition empirique.

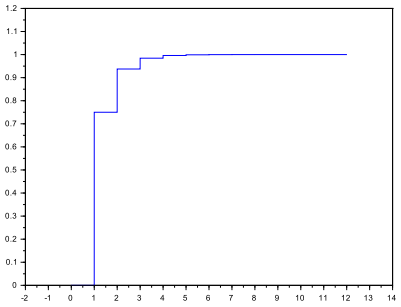
```
 -->x=grand(1,10^7,"bin",1,0.4);
-->a=(-0.01:0.001:1.01);
-->h=histplot(a,x,normalization=%F)/10^7;
-->f=[0 cumsum(h)];
-->clf();
-->plot(a,f);
```



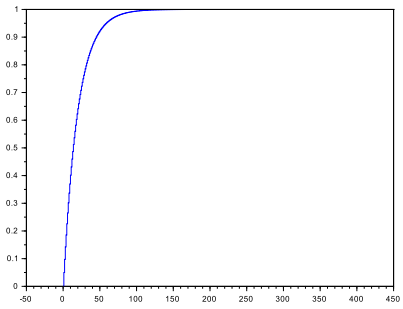

```
 -->x=grand(1,10^7,"bin",17,0.55);
-->a=(-0.01:0.01:17.01);
-->h=histplot(a,x,normalization=%F)/10^7;
-->f=[0 cumsum(h)];
-->clf();
-->plot(a,f);
```



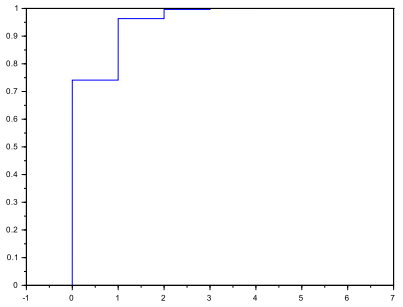
```
 -->x=grand(1,10^7,"geom",0.75);
-->b=max(x);
-->a=(-0.01:0.01:b+0.01);
-->h=histplot(a,x,normalization=%F)/10^7;
-->f=[0 cumsum(h)];
-->clf();
-->plot(a,f);
```



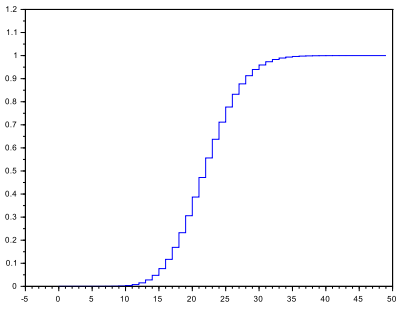
```
 -->x=grand(1,10^7,"geom",0.05);
-->b=max(x);
-->a=(-0.01:0.01:b+0.01);
-->h=histplot(a,x,normalization=%F)/10^7;
-->f=[0 cumsum(h)];
-->clf();
-->plot(a,f);
```



```
 -->x=grand(1,10^7,"poi",0.3);
-->b=max(x);
-->a=(-0.01:0.01:b+0.01);
-->h=histplot(a,x,normalization=%F)/10^7;
-->f=[0 cumsum(h)];
-->clf();
-->plot(a,f);
```



```
 -->x=grand(1,10^7,"poi",22);
-->b=max(x);
-->a=(-0.01:0.01:b+0.01);
-->h=histplot(a,x,normalization=%F)/10^7;
-->f=[0 cumsum(h)];
-->clf();
-->plot(a,f);
```



3.4.3 Quelques exemples d'estimations de moments

Soit X une v.a.r., $f : \mathbb{R} \to \mathbb{R}$ telle que $\mathbb{E}[|f(X)|] < \infty$.
La loi forte des grands nombres nous assure que si $(X_i)_{i\geq 1}$ est une suite de v.a.r i.i.d suivant la loi de X, alors [8]

$$\frac{1}{n}\sum_{i=1}^{n} f(X_i) \underset{n\to\infty}{\longrightarrow} \mathbb{E}[f(X)].$$

Le k-ème moment de X est défini par $\mathbb{E}[X^k]$. Détaillons trois exemples d'estimation, dans les trois cas on répète quatre fois l'estimation pour $n = 5 * 10^7$.

(i) $\mathbb{E}[X^3]$ lorsque $X \sim \text{Geom}(1/3)$.

```
-->x=grand(1,5*10^7,"geom",1/3);
-->y=x.^3;
-->sum(y)/(5*10^7)
 ans =
 110.92499

-->x=grand(1,5*10^7,"geom",1/3);
-->y=x.^3;
-->sum(y)/(5*10^7)
 ans =
 110.94455

 -->x=grand(1,5*10^7,"geom",1/3);
-->y=x.^3;
-->sum(y)/(5*10^7)
 ans =
 110.95707

-->x=grand(1,10^7,"geom",1/3);
```

8. pourvu que $\text{Var}[f(X)] < \infty$, l'inégalité de Chebychev permet d'assurer que $\mathbb{P}(|\frac{1}{n}\sum_{i=1}^{n} f(X_i) - \mathbb{E}[f(X)]| \geq \varepsilon) \leq \frac{\text{Var}(f(X))}{n\varepsilon^2}$ Autrement dit, lorsque $n = \frac{\varepsilon^{-2}}{p\text{Var}(f(X))}$ on obtient donc une précision ε avec probabilité au moins p. On reviendra plus précisément sur cet argument par la suite. Cependant, en pratique, on se contente souvent de réaliser l'estimation quatre ou cinq fois afin d'évaluer heuristiquement sa précision.

```
-->y=x.^3;
-->sum(y)/10^7
 ans =
 111.00334
```

On constate donc que $\mathbb{E}[X^3] \approx 111$ lorsque $X \sim \text{Geom}(1/3)$.
Exercice : Vérifier la validité de cette approximation par un calcul (on pourra utiliser la fonction génératrice des moments).

(ii) $\mathbb{E}[X^4]$ avec $X \sim \text{Bin}(6, 0.5)$

```
-->x=grand(1,5*10^7,"bin",6,0.5);
-->y=x.^4;
-->sum(y)/(5*10^7)
 ans =
 167.97563

-->x=grand(1,5*10^7,"bin",6,0.5);
-->y=x.^4;
-->sum(y)/(5*10^7)
 ans =
 168.00921

-->x=grand(1,5*10^7,"bin",6,0.5);
-->y=x.^4;
-->sum(y)/(5*10^7)
 ans =
 167.96817

-->x=grand(1,5*10^7,"bin",6,0.5);
-->y=x.^4;
-->sum(y)/(5*10^7)
 ans =
 168.02699
```

On constate donc que $\mathbb{E}[X^4] \approx 168$ lorsque $X \sim \text{Bin}(6, 0.5)$.
Exercice : Vérifier la validité de cette approximation par un calcul (on pourra utiliser la fonction génératrice des moments).

(iii) $\mathbb{E}[X^2]$ avec $X \sim \text{Poisson}(1)$. `-->x=grand(1,5*10^7,"poi",1);`

```
-->y=x.^2;
-->sum(y)/(5*10^7)
 ans =
 2.0000672

-->x=grand(1,5*10^7,"poi",1);
-->y=x.^2;
-->sum(y)/(5*10^7)
 ans =
```

```
 2.0000426
-->x=grand(1,5*10^7,"poi",1);
-->y=x.^2;
-->sum(y)/(5*10^7)
 ans =
 1.9988423
-->x=grand(1,5*10^7,"poi",1);
-->y=x.^2;
-->sum(y)/(5*10^7)
 ans =
 1.9995383
```

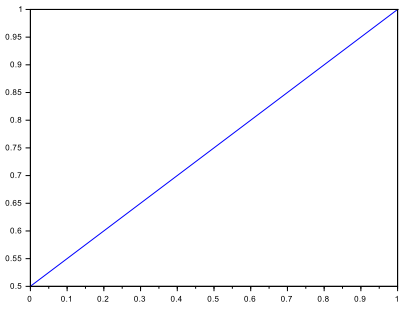
On constate donc que $\mathbb{E}[X^2] \approx 2$ lorsque $X \sim \text{Poisson}(1)$. Cette approximation est aisément vérifiée par un calcul direct.

3.4.4 Fonctions génératrices

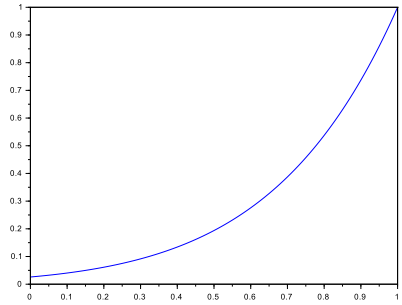
Pour un t donné, lorsqu'on veut estimer $\mathbb{E}[t^X]$, on utilise la même idée que dans le paragraphe précédent. Pour avoir un graphe de fonction génératrice, il faut faire ces estimations pour suffisamment de valeurs de t. On notera que $G_X(\cdot)$ est toujours définie sur $[0, 1]$, mais pas forcément sur $\mathbb{R}_+$ tout entier (cf e.g. le cas d'une variable géométrique).
Dans ce paragraphe on se contente de représenter nos fonctions génératrices sur cet intervalle $[0, 1]$.
Si on suppose que `x` est un vecteur aléatoire de taille $5 * 10^7$ qui simule des réalisations i.i.d suivant la loi de la variable X, on peut par exemple estimer G_X aux points $0, 0.01, 0.02, ..., 0.99, 1$, puis on représente le graphe de G_X : `g=zeros(1:101);`

```
 for i=1:101 do
  t=(i-1)/100;
  y=t.^x;
  g(i)=sum(y)/(5*10^7);
 end
a=(0:0.01:1);
plot(a,g);
```

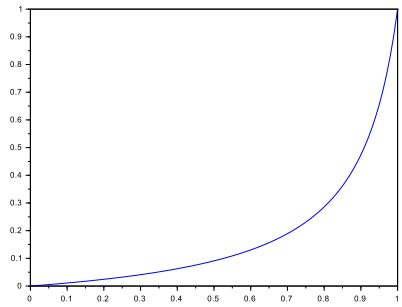
Lorsque `x=grand(1,5*10^7, "bin", 1, 0.5)` on obtient le graphe suivant :



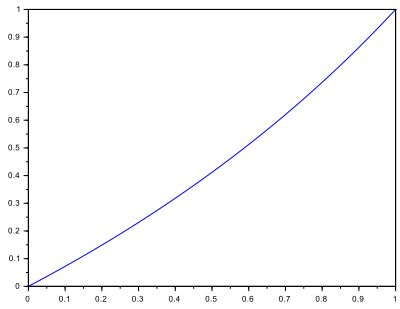
Lorsque `x=grand(1,5*10^7, "bin", 9, 1/3)` on obtient :



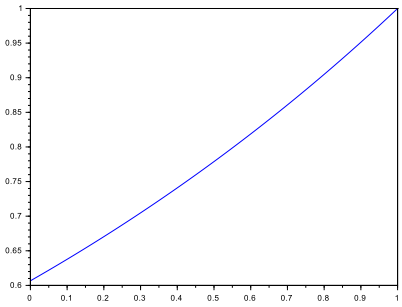
Lorsque `x=grand(1,5*10^7, "geom", 0.1)` on obtient :



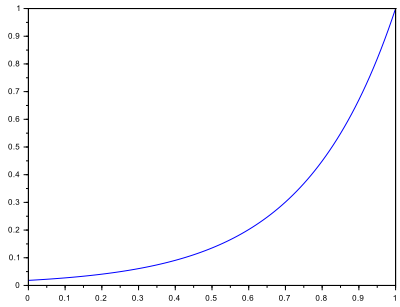
Lorsque `x=grand(1,5*10^7, "geom", 0.7)` on obtient :



Lorsque `x=grand(1,5*10^7, "poi", 0.5)` on obtient :



Lorsque `x=grand(1,5*10^7, "poi", 4)` on obtient :



4 Simuler des v.a.r. quelconques

4.1 Une propriété générale

Soit une v.a.r. X quelconque, et F_X sa fonction de répartition. Pour tout $u \in (0,1)$ on introduit

$$H_X(u) := \inf\{x \in \mathbb{R} : F_X(x) \geq u\},$$

la réciproque généralisée, continue à gauche, de F_X.

Théorème 4.1. *Si $U \sim \text{Unif}[0,1]$, alors $H_X(U)$ a la loi de X.*

Preuve : Comme F_X est continue à droite on a, par définition de H_X,et pour tous $a \in \mathbb{R}, u \in (0,1)$,

$$\{H_X(u) \leq a\} = \{F_X(a) \geq u\},$$

de sorte que pour tout $a \in \mathbb{R}$,

$$\mathbb{P}(H_X(U) \leq a) = \mathbb{P}(F_X(a) \leq U) = F_X(a). \qquad \square$$

A priori, le théorème ci-dessus permet de simuler n'importe quelle v.a.r. Cependant, il faut bien avoir conscience qu'un calcul explicite de H_X n'est souvent pas possible (cf l'exemple simple de la loi normale centrée réduite).

4.2 Le cas des v.a.r. à densité usuelles

4.2.1 Uniformes

On a vu dès le premier paragraphe que $X = \texttt{rand} \sim \text{Unif}[0,1]$. La fonction de répartition de X est

$$F_X(x) = \begin{cases} 0 \text{ si } x \leq 0 \\ x \text{ si } 0 \leq x \leq 1 \\ 1 \text{ si } x \geq 1 \end{cases} = \int_{-\infty}^{x} f_X(y)dy = \int_{-\infty}^{x} \mathbb{1}_{\{[0,1]\}}(y)dy,$$

de sorte que X possède la *densité* $f_X = \mathbb{1}_{\{[0,1]\}}$.
Pour obtenir une variable uniforme sur l'intervalle $[a,b]$, il suffit de translater/dilater $X = \texttt{rand()}$, car $Y = (b-a)X + a \sim \text{Unif}[a,b]$. On peut donc très aisément simuler une telle variable, ou une liste de telles variables

```
--> x= rand(1:n);
y=(b-a)*x+a;
```

Par exemple, on peut ainsi simuler 10 réalisations indépendantes d'une variable Unif$[-2,1]$:

```
 -->x= rand(1:10);
-->y=3*x-2
 y =
  - 1.3037756  - 1.3063288  - 1.3506102  0.6501663  - 0.0424595  -
1.0771728  0.7988849  - 1.3561976  - 1.062074  - 0.9150917
```

4.2.2 Exponentielles

Pour simuler une variable exponentielle $X \sim \exp(\lambda)$, où $\lambda > 0$, on se sert du fait que

$$F_X(x) = 1 - \exp(-\lambda x), \quad \forall x \geq 0,$$

de sorte que

$$H_X(u) = -\frac{1}{\lambda}\ln(1-u), \quad \forall u \in (0,1).$$

Il est donc aisé de simuler des réalisations indépendantes de variables exponentielles de paramètre λ :

```
 function h=poursimulexp(lambda,u)
1 h= -(1/lambda)*log(1-u);
endfunction
```

On peut alors simuler une liste de taille n de variables i.i.d, suivant la loi $\exp(\lambda)$:

```
--> x =rand(1:n)
--> feval(lambda,x,poursimulexp)
```

Par exemple, lorsque $n = 10$, $\lambda = 1$,

```
 -->x=rand(1:10)
 x =
  0.4062025  0.4094825  0.8784126  0.1138360  0.1998338  0.5618661
0.5896177  0.6853980  0.8906225  0.5042213
-->feval(1,x,poursimulexp)
 ans =
  0.5212169  0.5267561  2.1071218  0.1208532  0.2229358  0.8252306
0.8906662  1.1564468  2.2129498  0.7016256
```

Exercice 1 On sait qu'une somme de k variables exponentielles indépendantes de loi $\exp(\lambda)$ suit une loi $\Gamma(k,\lambda)$. Ecrire un programme permettant de simuler n réalisations indépendantes de variables i.i.d suivant la loi $\Gamma(6, 1.3)$.

4.2.3 Normale centrée réduite

Pour simuler une loi normale centrée réduite on peut se servir du résultat suivant.

Théorème 4.2. *(Box-Muller) Soit (U,V) i.i.d suivant la loi uniforme sur $[0,1]$. Alors $(X,Y) = (\sqrt{-2\ln(U)}\cos(2\pi V), \sqrt{-2\ln(U)}\sin(2\pi V))$ sont i.i.d suivant la loi normale centrée réduite.*

Preuve : Soit $\phi : \mathbb{R}^2 \to \mathbb{R}^2$ bornée mesurable,

$$\begin{aligned}
\mathbb{E}[\phi(X,Y)] &= \int_0^1\int_0^1 \phi(\sqrt{-2\ln(u)}\cos(2\pi v), \sqrt{-2\ln(u)}\sin(2\pi v))dudv \quad (1)\\
&= \int_{\mathbb{R}^2} \phi(x,y)\frac{u}{2\pi}dxdy \quad (2)\\
&= \int_{\mathbb{R}^2} \phi(x,y)\frac{\exp(-x^2/2-y^2/2)}{2\pi}dxdy, \quad (3)
\end{aligned}$$

où on a utilisé le changement de variable $(x,y) = \sqrt{-2\ln(u)}\cos(2\pi v), \sqrt{-2\ln(u)}\sin(2\pi v)$ dont l'inverse du jacobien vaut $\frac{u}{2\pi}$. □

Ce résultat permet simuler $2n$ réalisations indépendantes de normales centrées réduites à partir de $2n$ réalisations indépedantes d'uniformes sur $[0, 1]$.

```
-->x=[];
 for i=1:n do
  u=[rand() rand()];   y=poursimulnor(u);
  x(1,i)=y(1); x(2,i)=y(2);
 end
```

Par exemple, pour $n = 5$, on obtient avec la boucle précédente

```
-->x
 x =
  - 0.0846978  0.2204202  - 1.4013149  0.4230943  0.3054806
  2.2690754  - 2.3142386  2.0950159  1.75212  - 1.4910239
```

4.2.4 Normales

Lorsque $X \sim \mathcal{N}(0, 1)$, $\sigma X + \mu \sim \mathcal{N}(\mu, \sigma^2)$. Si on simule `x` : $2n$ réalisations indépendantes de la loi normale centrée réduite, comme dans le paragraphe précédent, alors

```
--> sigma .* x+mu
```

correspond à $2n$ réalisations i.i.d suivant la loi $\mathcal{N}(\mu, \sigma^2)$. Par exemple, pour $\mu = -1, \sigma^2 = 4$,

```
-->2.*x-1
 ans =
  - 1.1693955  - 0.5591597  - 3.8026297  - 0.1538115  - 0.3890388
  3.5381509  - 5.6284771  3.1900317  2.50424  - 3.9820479
```

4.3 Simulations directe à partir de `grand`

Les commandes

```
--> grand(n,m,"unf",a,b);
--> grand(n,m,"exp",1/lambda);
--> grandn,m,"nor",mu,sigma;
```

simulent des tableaux à n lignes et m colonnes de variables i.i.d, resp. suivant les lois $\text{Unif}[a, b], \exp(\lambda), \mathcal{N}(\mu, \sigma^2)$.

Il faut bien faire attention qu'en `scilab`, le paramètre à rentrer pour une loi exponentielle est sa moyenne $1/\lambda$. Il faut également faire attention que le deuxième paramètre à rentrer pour une loi normale est son écart-type σ plutôt que sa variance σ^2.

La commande `grand` permet en fait de simuler directement de nombreuses autres lois : β, χ^2, Γ, etc... On pourra consulter l'aide de `scilab` pour plus de détails.

4.4 Histogrammes, Densités, Fonctions de répartition, Transformées de Laplace

Supposons que X possède une densité f_X, et que x est un point de continuité de f_X. Supposons par ailleurs que les intervalles constituant les bases des rectangles de l'histogramme normalisé de X soient tous de longueur ε. Supposons enfin que l'échantillon dont on se sert pour réaliser l'histogramme est de taille n. Notons I_x l'unique intervalle constituant l'une des bases des rectangles de l'histogramme et contenant le point x. La hauteur du rectangle correspondant est égale à

$$\frac{\#\{\text{nombre de valeurs de l'échantillon qui tombe dans } I_x\}}{\varepsilon n}.$$

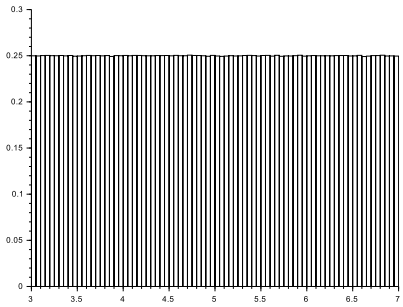
D'après la loi forte des grands nombres, ceci converge, lorsque $n \to \infty$, vers

$$\frac{1}{\varepsilon}\mathbb{P}(X \in I_x).$$

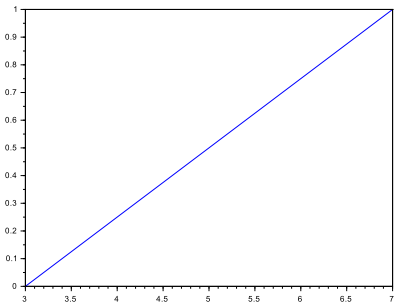
Par ailleurs, si f_X est continue en x, $\mathbb{P}(X \in I_x) \sim \varepsilon f_X(x)$, et donc la hauteur de notre histogramme au point x devrait être proche de $f_X(x)$.
Ce raisonnement permet de comprendre que l'allure des histogrammes pour de très grands échantillons permet, lorsqu'elle existe, et là où elle est suffisamment régulière (au moins continue), de deviner la fonction de densité.

4.4.1 Uniformes

Prenons l'exemple de la loi uniforme sur $[3, 7]$. `--> x=grand(1,5*10^7,"unf",3,7);`
`--> a=(3:0.05:7); --> h=histplot(a,x);`



La figure ci-dessus permet de se convaincre que lorsque $X \sim \text{Unif}[3, 7]$,
$f_X(x) = \frac{1}{4}\mathbb{1}_{\{[3,7]\}}(x)$.
La fonction de répartition s'obtient en intégrant :
`--> s=cumsum(h)*0.05; s= [0 s]; plot(a,s);`

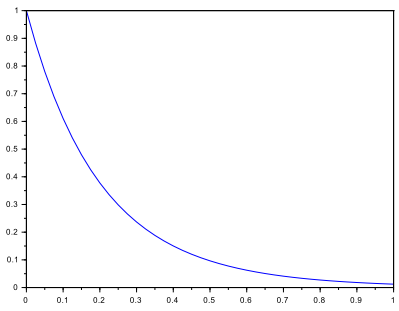


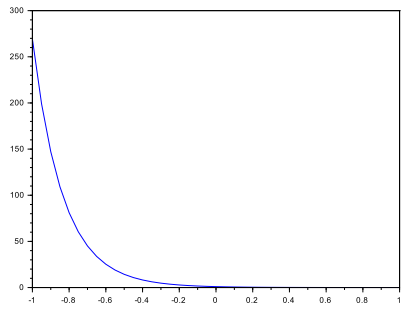
La figure ci-dessus permet donc de vérifier que

$$F_X(x) = \begin{cases} 0 \text{ si } x \leq 3 \\ x/4 - 3/4 \text{ si } 3 \leq x \leq 7 \\ 1 \text{ si } x \geq 7 \end{cases}.$$

Enfin, il est aisé d'évaluer la transformée de Laplace dans cet exemple, et de la représenter (par exemple sur l'intervalle $[0, 1]$, puis sur l'intervalle $[-1, 1]$).

```
--> t=0; s=[]; c=1; while c<=41 do a=-t.*x; b=exp(a); s(c)=sum(b)/(5*10^
7); c=c+1; t=t+0.025; end
-->plot((0:0.025:1),s')
--> t=-1; s=[]; c=1; while c<=41 do a=-t.*x; b=exp(a); s(c)=sum(b)/(5*10^
7); c=c+1; t=t+0.05; end
-->plot((-1:0.05:1),s')
```



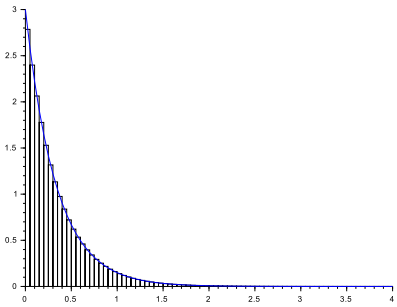


Les figures ci-dessus permet de s'assurer (par exemple en faisant figurer son graphe sur la figure) qu'on a bien affaire à la fonction $t \to \frac{\exp(-3t)-\exp(-7t)}{4t}$.

4.4.2 Exponentielles

Prenons l'exemple de la loi exponentielle de paramètre 3, i.e. de moyenne 1/3. `-->` `x=grand(1,5*10^7,"exp",1/3);`

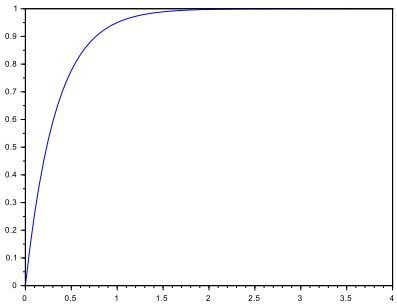
`--> a=(0:0.05:4); --> h=histplot(a,x);`
On va également faire figurer le graphe de $x \to 3\exp(-3x)$ sur la même figure. Dans `scinotes` on définit `function y=expdensity(x)`
`if x <0 then y=0;`
`else y=3*exp(-3*x);`
`end endfunction`
puis on trace le graphe de cette fonction
`-->feval(a,expdensity) --> plot(a,y);` Si on fait figurer sur une même figure ce graphe et l'histogramme plus haut on obtient :



ce qui permet de se convaincre que la densité de notre variable exp(3) est bien

$$f_X(x) := 3\exp(-3x)\mathbb{1}_{\{x\geq 0\}}.$$

La fonction de répartition s'obtient en intégrant :
`--> s=cumsum(h)*0.05; s= [0 s]; plot(a,s);`

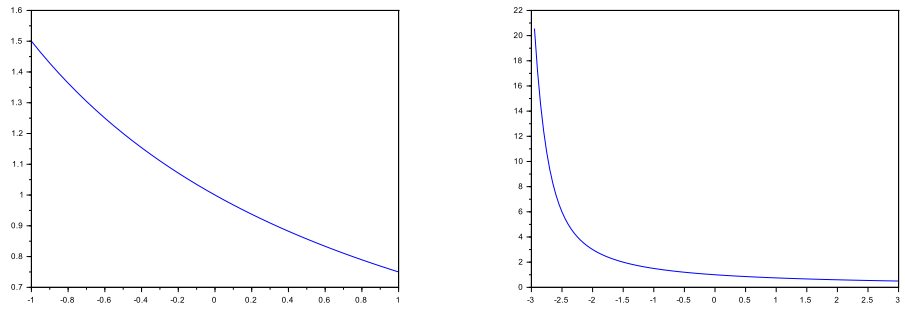


La figure ci-dessus permet de vérifier que

$$F_X(x) = \begin{cases} 0 \text{ si } x \leq 0 \\ 1-\exp(-3x) \text{ si } x > 0 \end{cases}.$$

Enfin, il est aisé d'évaluer la transformée de Laplace dans cet exemple, et de la représenter (par exemple sur l'intervalle $[-1, 1]$, puis sur l'intervalle $[-2.95, 3]$: rappelons qu'elle est définie sur $(-3, +\infty)$).

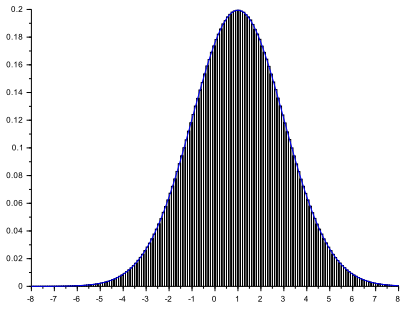
```
 --> t=-1; s=[]; c=1; while c<=41 do a=-t.*x; b=exp(a); s(c)=sum(b)/(5*10^
7); c=c+1; t=t+0.05; end
-->plot((-1:0.05:1),s')
 --> t=-2.95; s=[]; c=1; while c<=120 do a=-t.*x; b=exp(a); s(c)=sum(b)/10^
7; c=c+1; t=t+0.05; end
-->plot((-2.95:0.05:3),s')
```



Cette figure permet de vérifier que cette transformée de Laplace est bien la fonction $t \to \frac{3}{3+t}$ pour $t > -3$.

4.4.3 Normales

Prenons l'exemple de la loi normale de paramètres $\mu = 1, \sigma^2 = 4$. `--> x=grand(1,5*10^7,"nor",1,2);`
`--> a=(-8:0.1:8); --> h=histplot(a,x);`
On va également faire figurer le graphe de $x \to \exp(-(x-1)^2/8)/(2 * \sqrt{2\pi})$ sur la même figure. Dans `scinotes` on définit `function y=nordensity(x)`
`y=exp(-(x-1)^2/8)/(2*sqrt(2*%pi));`
`endfunction`
puis on trace le graphe de cette fonction
`-->feval(a,nordensity) --> plot(a,y);` Si on fait figurer sur une même figure ce graphe et l'histogramme plus haut on obtient :

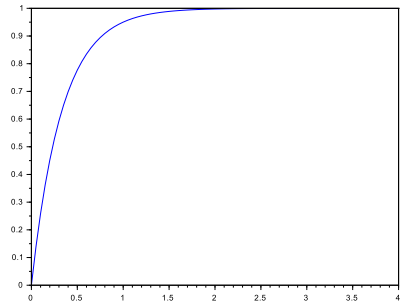


ce qui permet de se convaincre que la densité de notre variable $\mathcal{N}(1,4)$ est bien

$$f_X(x) := \frac{1}{2\sqrt{2\pi}} \exp\left(-\frac{(x-1)^2}{8}\right).$$

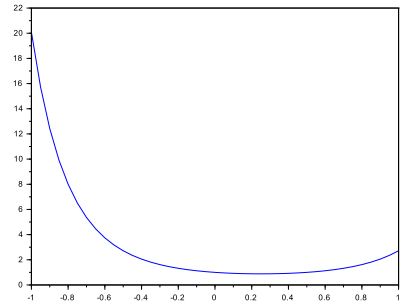
La fonction de répartition s'obtient en intégrant :

```
--> s=cumsum(h)*0.1; s= [0 s]; plot(a,s);
```



Enfin, il est à nouveau aisé d'évaluer la transformée de Laplace dans cet exemple, et de la représenter (par exemple sur l'intervalle $[-1,1]$: rappelons qu'elle est définie sur $\mathbb{R}$).

```
 --> t=-1; s=[]; c=1; while c<=41 do a=-t.*x; b=exp(a); s(c)=sum(b)/(5*10^
7); c=c+1; t=t+0.05; end
-->plot((-1:0.05:1),s')
```



La figure ci-dessus permet de vérifier (par exemple en superposant les graphes) que cette transformée est bien $t \to \exp(2t^2 - t)$.