Trucs et Astuces sous Scilab
pour l'option Probabilités et Statistiques

Jean-Louis Marchand
Université Rennes 2

24 septembre 2014

Table des matières

1 Générer des variables aléatoires **2**
1.1 La fonction grand 2
1.2 La simulation de lois plus générales 2
1.3 les processus classiques 3

2 Illustrations **5**
2.1 Faire un graphique 5
2.2 Ce qu'il faut y mettre 10

3 Tests statistiques **11**
3.1 Un peu de théorie 11
3.2 Ce que l'on peut faire faire à Scilab lors d'un test 13

4 Identifier une erreur **15**

5 Pour en savoir plus **16**

1 Générer des variables aléatoires

1.1 La fonction grand

C'est la fonction indispensable déjà intégrée à Scilab. De manière formelle, voilà en résumé l'utilisation de la fonction :

```
X=grand(m,n,'[nom de la loi]',[paramètres de la loi]);
```

Cette fonction génère un matrice X à m lignes et n colonnes dont les coefficients sont IID de loi fixée par les derniers arguments. Par exemple, si vous voulez simuler 23 réalisations IID d'une uniforme sur $[0,1]$, il suffit de taper

```
X=grand(23,1,'def');
```

Voici une liste non-exhaustive des lois implémentées dans Scilab, tapez

```
help grand;
```

pour avoir une description complète de la fonction.,,

Distribution	Appel de la fonction	Mises en garde
Binomiale (N,p)	`grand(m,n,'bin',N,p)`	-
Exponentielle (λ)	`grand(m,n,'exp',1/lambda)`	moyenne $\frac{1}{\lambda}$
Gaussienne (μ,σ^2)	`grand(m,n,'nor',mu,sigma)`	écart-type σ
Géométrique (p)	`grand(m,n,'geom',p)`	-
Poisson (λ)	`grand(m,n,'poi',lambda)`	-
Uniforme sur $[0,1)$	`grand(m,n,'def');`	-
Uniforme sur $[a,b)$	`grand(m,n,'unf',a,b);`	-
Uniforme sur $\{k,\ldots,\ell\}$	`grand(m,n,'uin',k,l);`	-

1.2 La simulation de lois plus générales

Une loi finie : La première façon, et peut-être la plus intuitive, consiste à utiliser la loi uniforme sur $[0,1]$ pour réaliser le tirage. Appelons $P=(P_1,\ldots,P_n)$ la probabilité qui nous intéresse. On découpe alors le segment en intervalles de longueurs P_i. On lance ensuite une variable uniforme sur $[0,1]$ et on regarde dans quel intervalle elle tombe

```
P=[1/6; 1/4; 1/6; 1/6; 1/6; 1/12];
I=cumsum(P);//on définit les bornes sup de chaque sous-
   intervalle
U=grand(1,1,'def');
```

```
X=min(find(U<I));//find donne les indices de I qui vérifient l'
    inégalité
```

Pour simuler une loi sur un ensemble fini à k éléments, le plus simple revient à utiliser la loi multinomiale. En effet celle ci s'appelle directement dans Scilab par

```
X=grand(1,'mul',1,P);
```

où P est un vecteur **colonne** de taille $\mathbf{k-1}$, donnant les probabilités des $k-1$ premières classes (Scilab complète tout seul la dernière valeur). Si l'on veut simuler N tirages successifs de cette loi, il suffit d'appeler

```
X=grand(N,'mul',1,P);
```

Par exemple, je veux simuler deux lancers indépendants d'un dé à six faces de distribution $(\frac{1}{6}\ \frac{1}{4}\ \frac{1}{6}\ \frac{1}{6}\ \frac{1}{6}\ \frac{1}{12})$

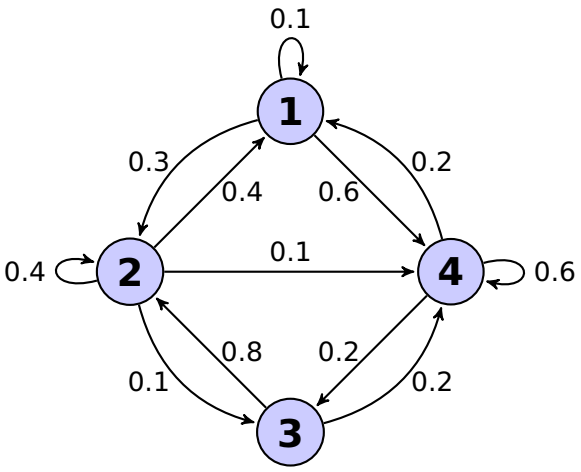
```
P=[1/6; 1/4; 1/6; 1/6; 1/6];//le vecteur est de dimension 5!
grand(2,'mul',1,P);//deux vecteurs colonnes élémentaires
//le 1 donnant la face sortie à chaque tour
```

1.3 les processus classiques

Chaîne de Markov homogène : pour simuler les N premières réalisations d'une chaîne de Markov dans un espace fini de matrice de transition P, la fonction `grand` donne encore la solution

```
grand(N,'markov',P,x0);
```

Par exemple pour simuler les 10 premiers pas de la marche donnée par le graphe suivant, initialisée en 1 :
(Source du graphe : http://www.texample.net/tikz/examples/graph/)



On introduit la matrice de transition, ainsi que la position initiale

```
P=[0.1 0.3 0 0.6;
0.4 0.4 0.1 0.1;
0 0.8 0 0.2;
```

```
0.2 0 0.2 0.6];
x0=1;
X=grand(10,'markov',P,x0);
```

Vous obtenez alors les 10 premiers instants de la chaîne **sans la position initiale !**

Attention, lorsque l'ensemble est de grand cardinal, cela implique que vous ayez le temps de définir une matrice coefficient par coefficient sans vous tromper !

À ce propos des fonctions existent pour générer rapidement des matrices. En effet, les fonctions `diag` et `toeplitz` permettent de créer rapidement des matrices surtout quand celles-ci sont de grande taille.

Tout d'abord pour réaliser une matrice diagonale, il suffit de définir le vecteur des coordonnées diagonales, par exemple

```
D=[1:10];
D=diag(D);
```

La seconde fonction permet de créer des matrices tridiagonales ce qui est souvent utile. Par exemple, si l'on s'intéresse à la ruine du joueur : le jeu permet de remporter au maximum la somme b k€ mise en jeu par la *Banque*, le joueur arrive, lui, dans le jeu avec sa fortune f k€ ($f < b$). Le jeu consiste en une succession de parties,chacune de type expérience de Bernoulli. Le joueur remporte 1000€ en cas de victoire et perd 1000€ dans le cas contraire. Le jeu s'arrête si le joueur est ruiné ou s'il empoche la somme maximum bk€.

Pour réaliser des trajectoires des premiers pas de cette marche, on introduit le paramètre p qui donne la probabilité de gain sur une partie.

```
function [X]=generateur(f,b,p,nb_pas);//0<f<b entiers
        z=zeros(1,b+1);
        u=z;u(2)=1-p;
        v=z;v(2)=p;
        P=toeplitz(u,v);
        P(1,1)=1;P(b+1,b+1)=1;//états absorbants
        P(1,2)=0;P(b+1,b)=0;//on retire les valeurs inutiles
        disp(P);
        X=grand(nb_pas,'markov',P,f+1);
        if X($)==b+1 then
                disp('le joueur remporte le jackpot');
        elseif X($)==1
                disp('le joueur est ruiné');
        else disp('il reste au joueur la somme de '+string(X
           ($)-1));
        end
endfunction
```

Chaîne plus générale : dans des cas non-homogènes et/ou avec espace d'état très grand voire infini, il n'y a généralement pas d'autre solution que de créer une boucle. On adopte alors la logique de dynamique du processus en générant la nouvelle position suivant la précédente. On se rappellera qu'une chaîne réelle peut toujours s'écrire de la formelle

$$X_{n+1} = f_n(X_n, U_{n+1}),$$

où f_n est déterministe et $\{U_n\}_{n\geq 1}$ est une suite IID de variables uniformes sur $[0,1]$. Cette relation s'avérera très utile dans certains cas.

Par exemple, une marche où on lance une gaussienne réduite et centrée en X_{n-1} pour déterminer X_n :

```
function [X]=generateur(N,x0);//N=nb de pas, x0=pos.
   initiale
        X=[x0];//initialisation de la marche
        last_position=x0;//derniere visite de la chaîne
        for k=1:N
                last_position=grand(1,1,'nor',last_position,
                   1);
                X=[X last_position];
        end;
endfunction;
```

Processus de Poisson : premier exemple de processus à temps continu. Dans ce cas précis on se concentre uniquement sur les temps de saut.

```
//génération d'un processus de Poisson(lambda) sur [0,t]
function [T]=generateur(t,lambda);
        while T($)<t
                E=grand(1,1,'exp',1/lambda)
                T=[T T($)+E];
        end;
endfunction;
```

2 Illustrations

2.1 Faire un graphique

Il y a trois types de graphiques de base à connaître : la fonction escalier, la courbe lissée et l'histogramme.

La fonction constante par morceaux : la commande `plot2d2` utilise en argument un vecteur donnant les abscisses, un second de même taille donnant les ordonnées, il y a des arguments optionnels, pour changer la couleur par exemple. Par exemple, pour représenter la fonction de répartition sur le segment $[0,7]$ du dé pipé de distribution $(\frac{1}{6}\ \frac{1}{4}\ \frac{1}{6}\ \frac{1}{6}\ \frac{1}{6}\ \frac{1}{12})$ introduit précédemment

```
x=[0:7];
P=[1/6; 1/4; 1/6; 1/6; 1/6; 1/12];
P=cumsum(P);//vecteur des sommes partielles
P=[0; P; P($)];
clf;//efface les tracés précédents
plot2d2(x,P,5);
```

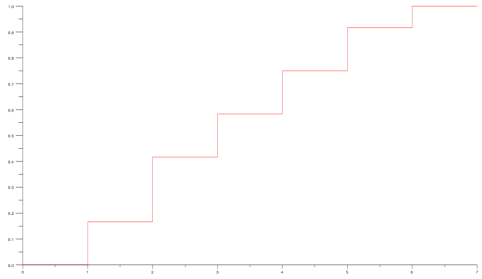


Figure 1 – Fonction de répartition sur le segment [0, 7] du dé pipé.

Le chiffre 5 donne la couleur rouge à la courbe. Voici une liste du code des couleurs

1	2	3	4	5	6	7	8	9
Noir	Bleu	Vert	Cyan	Rouge	Violet	Jaune	Blanc	Bleu Marine

Essayez d'éviter les couleurs trop claires comme le jaune et le vert qui peuvent ne pas apparaître sur l'écran, surtout via un vidéoprojecteur.

La fonction lissée : les arguments de `plot2d` sont de même nature, cependant le graphique est obtenu par interpolation linéaire des points.

Par exemple, pour représenter la densité d'une variable gaussienne, de moyenne 3 et de variance 4 sur le segment [−7, 7]

```
x=[-7:.01:7];
y=x-3;
y=exp(-y^{2}/4);
y=1/(2*sqrt(2*%pi))*y;
clf;
plot2d(x,y,2);
```

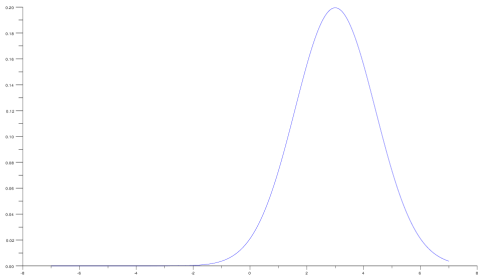


Figure 2 – Densité d'une variable gaussienne de moyenne 3 et de variance 4.

L'histogramme : très apprécié pour illustrer par exemple une convergence en loi en utilisant les densités, celui-ci requiert une collection de valeurs et le nombre total de classes.

Par exemple, on simule un grand nombre (10000) de variables IID de loi exponentielle de paramètre 2, on devrait retrouver à l'aide de l'histogramme une approximation aléatoire de la densité correspondante (avec 20 classes ici)

```
X=grand(10000,1,'exp',.5);
clf;
histplot(20,X,9);
x=[0:.01:max(X)];
plot2d(x,2*exp(-2*x),5);
```

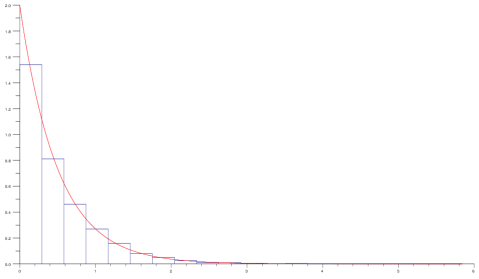


Figure 3 – En bleu marine, histogramme de 10000 copies indépendantes d'une variable exponentielle de paramètre 2 réparties en 20 classes. En rouge, la densité de la variable.

Les tracés 2D de manière plus générale Si l'on veut tracer autre chose que le graphe d'une fonction, on utilise `plot2d`.

Par exemple, pour représenter une marche aléatoire sur $\mathbb{Z}^2$

```
function [X]=trajectoire(nb_pas,P);//P proba sur 1,2,3,4
        X=[0; 0 ];
        nord=[0; 1];
        sud=[0; -1];
        est=[1; 0];
        ouest=[-1; 0];
        M=[nord sud est ouest];
        P(4)=[];//se souvenir que l'on ne doit avoir que 3
           coordonnées
        for k=1:nb_pas
                sens=grand(1,'mul',1,P);
                X(:,k+1)=X(:,k)+M*sens;
        end
        plot2d(X(1,:),X(2,:),2);
endfunction
```

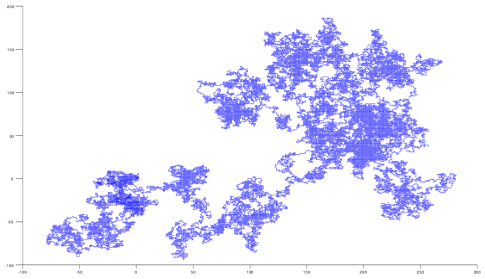


Figure 4 – Représentation des 10000 premiers pas de la marche aléatoire sur $\mathbb{Z}^2$ initialisée en $(0, 0)$.

La décoration : en plus de tracer les bons graphiques, il est toujours de bon goût d'y ajouter deux trois informations qui les rendent lisibles. Il est ainsi possible d'ajouter un titre général, un également pour chaque axe. De plus, lorsqu'il y a plusieurs courbes, une légende permet d'afficher la correspondance entre les couleurs et les courbes représentées Reprenons l'exemple précédent où deux courbes apparaissent, nous pouvons définir un titre de taille de police 4, sinon c'est un peu petit. De même, on obtient les axes et la légende des différentes courbes

```
X=grand(10000,1,'exp',.5);
clf;
histplot(20,X,9);
x=[0:.01:max(X)];
plot2d(x,2*exp(-2*x),5);
title('Illustration de la convergence vers une densité','
   fontsize',4);
xlabel('Axe des x','fontsize',4);
ylabel('Axe des y','fontsize',4);
legends(['Histogramme';'Densité'],[9;5],opt='ur', font_size=
   4);
```

L'argument `opt='ur'` indique l'emplacement de la légende, en haut à droite (*upper-right*).

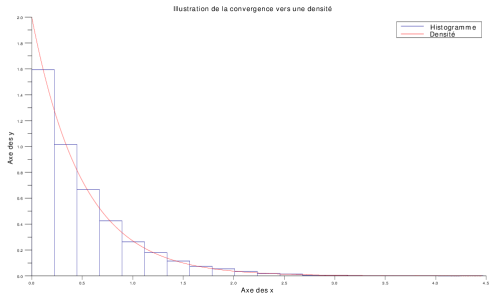


Figure 5 – C'est déjà mieux, non ?

La multiplication des graphes dans une même fenêtre : Si l'on veut disposer deux ou plusieurs graphiques en mosaïque, il y a plus pratique que manipuler plusieurs fenêtres. La fonction utilisée pour cela est `subplot(m,n,N)` qui définit une matrice de taille $m \times n$ d'autant de graphiques, et $1 \leq N \leq mn$ est le numéro lexicographique de la fenêtre dans laquelle on veut tracer.

```
X=grand(10000,1,'exp',.5);
clf;
subplot(1,2,1)//premier graphe
histplot(20,X,9);
x=[0:.01:max(X)];
plot2d(x,2*exp(-2*x),5);
title('Illustration de la convergence vers une densité','
   fontsize',2);
xlabel('Axe des x','fontsize',2);
ylabel('Axe des y','fontsize',2);
legends(['Histogramme';'Densité'],[9;5],opt='ur', font_size=
   2);
subplot(1,2,2)//deuxième graphe
x=[-7:.01:7];
y=x-3;
y=exp(-y^{2}/4);
y=1/(2*sqrt(2*%pi))*y;
plot2d(x,y,2);
title('Densité de la gaussienne de moyenne 3 et de variance
   4','fontsize',2);
xlabel('Axe des x','fontsize',2);
ylabel('Axe des y','fontsize',2);
```

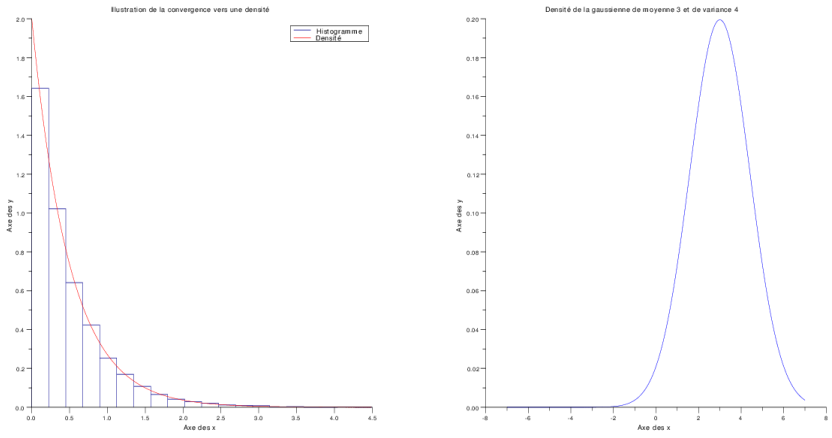


Figure 6 – Disposition de deux graphes en lignes dans une même fenêtre graphique.

2.2 Ce qu'il faut y mettre

Nous allons essayer ici de présenter ce qui est souvent attendu par un jury dans les illustrations graphiques en général.

La plupart du temps, ces graphiques cherchent à mettre en évidence un résultat de convergence. Les outils diffèrent quelque peu, suivant le type de convergence que l'on cherche à démontrer.

La convergence en loi : Pour la convergence en loi, on pourra utiliser des densités ou des fonctions de répartition suivant les cas.

Nous avons déjà vu le cas de la densité, nous allons maintenant voir comment faire pour mettre en évidence la convergence des fonctions de répartition empiriques vers la théorique. Pour cela, nous pouvons par exemple partir de la simulation d'échantillons de taille respectives $n = 100, 1000, 10000$ suivant une loi géométriques de paramètre $0,2$. Pour trier les valeurs et compter le nombre d'occurrences correspondant on utilise la fonction `tabul`

```
function illustration();
        x=grand(1,100,'geom',0.2);
        y=grand(1,1000,'geom',0.2);
        z=grand(1,10000,'geom',0.2);
        x=tabul(x,'i');//i pour increasing
        y=tabul(y,'i');//le tri par défaut est décroissant!
        z=tabul(z,'i');
        x(:,2)=x(:,2)/100;//normalisation des occurences
        y(:,2)=y(:,2)/1000;
        z(:,2)=z(:,2)/10000;
        m=max(x($,1),y($,1),z($,1));
        clf;
        //on réprésente alors les fonctions de répartition
        //empiriques, avec la fonction cumsum
        xx=[0; x(:,1) ;m];xxx=[0; x(:,2) ; x($,2)];
        yy=[0; y(:,1) ;m];yyy=[0; y(:,2) ; y($,2)];
        zz=[0; z(:,1) ;m];zzz=[0; z(:,2) ; z($,2)];
        plot2d2(xx,cumsum(xxx),3);
        plot2d2(yy,cumsum(yyy),4);
        plot2d2(zz,cumsum(zzz),5);
        title('Illustration de la convergence des fonctions
           de répartition empiriques','fontsize',4);
        xlabel('Axe des x','fontsize',4);
        ylabel('Axe des y','fontsize',4);
        legends(['n=100';'n=1000';'n=10000'],[3;4;5],opt='ur
           ', font_size=4);
endfunction;
```

L'intervalle de confiance : le grand classique de la convergence en loi : le théorème central limite. On cherche à mettre en évidence le fait que les erreurs à la moyenne sont quasiment distribués selon une loi normale lorsque la taille de

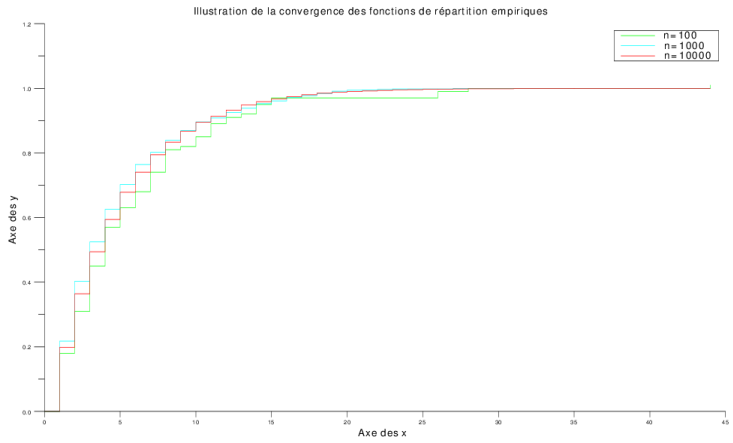


Figure 7 – Notez que l'on a du mal à voir les couleurs très claires !

l'échantillon est grande. Ce résultat donne la vitesse de la convergence et l'intervalle de confiance est un bon moyen de la faire apparaître.

Par exemple, pour illustrer le théorème central limite, on part d'un échantillon de taille croissante d'une loi exponentielle de paramètre $0,2$

```
function tcl(nb_iterations);
        X=grand(1,nb_iterations,'exp',5);
        n=[1:nb_iterations];
        X=cumsum(X)./n;
        ecart=5*1.96./sqrt(n);
        clf;
        plot2d2(n',[X'-ecart' X' X'+ecart'],[5; 2; 5]);
        plot2d(n,ones(n)*5);
        title('Illustration du TCL','fontsize',4);
        xlabel('Taille de l''échantillon','fontsize',4);
        ylabel('Axe des y','fontsize',4);
        legends(['borne haute de l''IC';'estimation de la
           moyenne';'borne basse de l''IC';'moyenne thé
           orique'],[5;2;5;1],opt='ur', font_size=4);
endfunction
```

3 Tests statistiques

3.1 Un peu de théorie

Injustement boudé par des générations de probabilistes à l'agrégation, le test statistique peut rapporter gros. Son but est de développer une méthode pour relier une série de données réelles à un modèle *a priori* du phénomène étudié. La méthode repose sur l'opposition de deux hypothèses :

— l'hypothèse dite *nulle*, notée (H_0) qui est celle que l'on cherche à tester ;

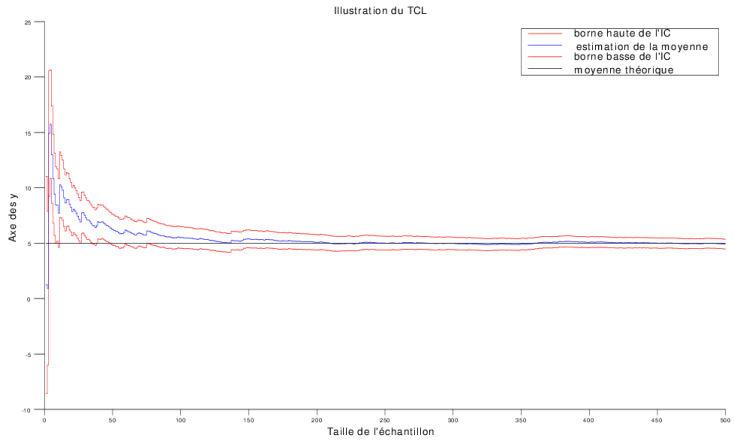


Figure 8 – Pensez à varier les couleurs.

— l'hypothèse dite *alternative*, notée (H_1) qui porte bien son nom.

Le principe : ne pas rejeter l'hypothèse de départ (H_0) si on n'a pas assez d'éléments pour le faire. On aime bien le parallèle avec le système judiciaire français où le suspect bénéficie de la présomption d'innocence (il se fait quand même appeler suspect ou accusé ...). L'accusation cherche alors à réunir un maximum d'éléments pour être en mesure de faire pencher la balance de l'autre côté.

Comme aucun système n'est parfait, les erreurs existent et donc on définit les erreurs de *première* et *seconde* espèce suivant les cas :

Verdict \ Vérité	$(H_0) = innocent$	$(H_1) = coupable$
$(H_0) = innocent$	juste	erreur de 2nde espèce
$(H_1) = coupable$	erreur de 1ère espèce	juste

Attention ! Les rôles de (H_0) et (H_1) ne sont pas symétriques.

L'approche généralement adoptée consiste à chercher à minimiser en priorité l'erreur de première espèce. Bien sûr, on essaie ensuite de contrôler celle de seconde espèce.

Voilà le schéma général lorsque l'on s'intéresse au choix du paramètre d'une loi donnée :

1. on fixe un modèle statistique, un triplet $(\mathcal{X}, \mathcal{A}, \{\mathbb{P}_\theta\}_{\theta\in\Theta})$ pour l'échantillon $X = (X_1, \dots, X_n) \in \mathcal{X}$ qui modélise le jeu $x_{obs} \in \mathcal{X}$ de valeurs observées ;
2. choix de $(H_0) = \theta \in \Theta_0 \subset \Theta$ et $(H_1) = \theta \in \Theta_1 \subset \Theta$;
3. choix du *niveau* α, qui limite l'erreur de première espèce, *i.e.* rejeter (H_0) à tort :
$$\sup_{\theta\in\Theta_0} \mathbb{P}_\theta\Big(\text{rejeter}\,(H_0)\Big) = \alpha;$$
4. partie délicate du travail : trouver une variable aléatoire réelle $T(X)$ dont on connaît la loi exacte (très très dur en général) ou asymptotique (beaucoup plus raisonnable) ;

5. en fonction de la signification de T par rapport aux hypothèses, détermination de la région de rejet R afin de vérifier :

$$\sup_{\theta\in\Theta_0} \mathbb{P}_\theta\Big(T(X)\in R\Big) = \alpha,$$

soit en utilisant la loi exacte de $T(X)$ soit une approximation asymptotique ;

6. maintenant que R est connue, évaluer $T(x_{obs})$ et décider si l'on rejette ou non (H_0) au niveau α ;
7. évaluer la *puissance* β, qui donne l'erreur de seconde espèce, *i.e.* rejeter (H_1) à tort :

$$\sup_{\theta\in\Theta_1} \mathbb{P}_\theta\Big(\text{rejeter}\,(H_1)\Big) = \sup_{\theta\in\Theta_1} \mathbb{P}_\theta\Big(T(X)\notin R\Big) = \beta.$$

3.2 Ce que l'on peut faire faire à Scilab lors d'un test

A la recherche des quantiles : hormis certains cas où il est facile de les calculer, Scilab a déjà en mémoire tout ce dont vous pouvez avoir besoin.

Concentrons-nous sur une de ces fonctions qui permettent d'évaluer les quantiles : `cdfnor` pour la distribution gaussienne. En réalité, cette fonction en comprend 3 différentes suivant l'information qui vous intéresse :

La fonction de répartition (fdr) : vous choisissez en premier lieu la façon dont vous voulez vous servir de `cdfnor`, c'est le premier argument `'PQ'`. Ensuite vous fixez la moyenne et l'écart type, puis l'abscisse à laquelle vous voulez évaluer la fdr

```
P=cdfnor('PQ',X,mu,sigma);
```

Par exemple, l'évaluation en $X = 5$ pour une moyenne de $\mu = -1,5$ et un écart-type $\sigma = 3$

```
P=cdfnor('PQ',5,-1.5,3);
```

Quantile : pour signifier que vous voulez le quantile, le premier argument est alors `'X'`. Vous appelez la fonction alors par

```
X=cdfnor('X',mu,sigma,P,1-P);
```

où P est la probabilité qui vous intéresse. Par exemple, pour trouver le quantile de la gaussienne centrée réduite correspondant à $P = 97,5\%$

```
X=cdfnor('X',0,1,.975,.025);
```

Estimation des paramètres : on ne le développe pas ici, je vous vous invite à consulter l'aide en ligne. Pour chaque loi, vous pouvez estimer un des paramètres en fixant les autres, ainsi qu'une probabilité et son quantile associé.

Voici la liste des distributions que l'on retrouve fréquemment lors des tests, ,,

Distribution	Fonction de répartition	Quantile
Binomiale(n, p)	`P=cdfbin('PQ',S,n,p,1-p)`	`S=cdfbin('S',n,p,1-p,P,1-P)`
Chi2(n)	`P=cdfchi('PQ',X,n)`	`X=cdfchi('X',n,P,1-Q);`
Fisher(n, m)	`P=cdff('PQ',F,n,m)`	`F=cdff('F',n,m,P,Q);`
Gaussienne(μ, σ^2)	`P=cdfnor('PQ',X,mu,sigma)`	`X=cdfnor('X',mu,sigma,P,1-P)`
Poisson(λ)	`P=cdfpoi('PQ',S,lambda)`	`S=cdfpoi('S',lambda,P,1-Q)`
Student(n)	`P=cdft('PQ',T,n)`	`T=cdft('T',n,P,1-P)`

L'apprentissage par l'exemple : Nous allons développer un exemple pour montrer qu'en pratique tout cela va très vite. Voici un exercice élémentaire de test (piqué à une planche d'exos de l'ENSAI) :

> Un négociant de vin s'intéresse à la contenance des bouteilles d'un producteur soupçonné par certains clients de frauder. Il souhaite s'assurer que cette contenance respecte bien en moyenne la limite égale de 75 cL. À cet effet, il mesure le contenu de 10 bouteilles prises au hasard et obtient les valeurs suivantes (en cL) :
>
> 75.2, 72.6, 74.5, 75, 75.5, 73.7, 74.1, 75.8, 75.8, 75.
>
> On suppose que la contenance des bouteilles (en cL) suit une loi normale d'espérance θ inconnue, d'écart-type connu égal à 1.
>
> 1. Écrire le modèle statistique considéré.
> 2. Le négociant décide de tester l'hypothèse nulle $(H_0) : \theta = 75$ contre l'alternative $(H_1) : \theta < 75$. Quel point de vue le négociant adopte-t-il en choisissant ces hypothèses ?
> 3. Construire, à l'aide d'une règle de décision intuitive basée sur la moyenne empirique, un test de niveau $\alpha = 1\%$. Quelle est la conclusion du test pour les valeurs données ?

Nous répondons alors aux questions :

1. le modèle statistique $(\mathcal{X}, \mathcal{A}, \{\mathbb{P}_\theta\}_{\theta\in[0,75]})$

$$\mathcal{X} = \mathbb{R}^{10},$$
$$\mathcal{A} = \mathcal{B}\left(\mathbb{R}^{10}\right),$$
$$\mathbb{P}_\theta = \mathcal{N}(\theta, 1)^{\otimes 10};$$

2. le négociant fait confiance au producteur et désire avoir assez d'éléments pour rejeter (H_0) ;
3. soit $T(x) = \frac{1}{10}\sum_{k=1}^{10} x_k$ l'estimateur empirique de l'espérance θ. On rejette alors (H_0) pour de petites valeurs de $T(x)$ qui tendraient à promouvoir (H_1). On définit alors la fonction de répartition φ par :

$$\begin{aligned}\varphi : \mathbb{R}^{10} &\to \{0,1\}\\ x &\mapsto \mathbf{1}_{T(x)\leq s},\end{aligned}$$

où le *seuil* s reste à déteminer. Pour cela, on utilise le niveau, car alors l'erreur de 1ère espèce est donnée par

$$\mathbb{P}_{\theta=75}\Big(\varphi(X)=1\Big) = \mathbb{P}_{\theta=75}\Big(T(X)\leq s\Big) = \alpha = 1\%.$$

Sous (H_0) : $\theta = 75$, la variable $T(X)$ suit une gaussienne de moyenne 75 et de variance $\frac{1}{10}$. Pour déterminer le seuil, on utilise Scilab

```
s=cdfnor('X',75,sqrt(.1),.01,.99);
```

On trouve alors $s = 74,264$. La fonction de test est donc $\varphi(x) = \mathbf{1}_{T(x)\leq 74,264}$. On calcule alors $T(x_{obs})$

```
x_obs=[75.2 72.6 74.5 75 75.5 73.7 74.1 75.8 75
    .8 75];
T_obs=mean(x_obs);
```

Il n'y a plus qu'à comparer la valeur de $T(x_{obs}) = 74.42$ à $s = 74.246$ pour déduire qu'on ne rejette pas (H_0) sur la base des données.

On peut alors créer une fonction en généralisant à une taille n plus générale.

```
function test(x_obs);
        n=length(x_obs);
        T_obs=mean(x_obs);
        s=cdfnor('X',75,sqrt(1/n),.01,.99);
        decision=1*(T_obs<= s);
        disp('H'+string(decision));
endfunction;
```

4 Identifier une erreur

Lorsque le programme plante, l'éditeur incorporé de Scilab a tendance à afficher la bonne ligne où se situe l'erreur.

Cependant, il arrive souvent que cette erreur soit un symptôme plus que l'origine du mal. Pour bien isoler la ligne que Scilab n'arrive pas à exécuter, on peut recourir à l'utilisation abondante de la fonction `disp`. Cette fonction permet d'afficher sur la console ce que l'on veut : la valeur d'une variable, une chaîne de caractères, etc.

On s'en sert alors pour fixer des jalons le long du code, ainsi lorsque le programme plante, il suffit de regarder quel dernier appel de `disp` a fonctionné.

Par exemple, sur une boucle, l'erreur peut survenir sur les dernières itérations, cas très classique où les bornes sont mal définies

```
function generateur();
    X=[0];
        for k=1:10
                disp(k);
                X(k)=1/(10-k);
        end
endfunction
```

Ce programme une fois lancé s'arrêtera à la dixième itération de la boucle parce qu'on tente une division par 0. Le dernier élément affiché sur la console est bien $k = 10$.

5 Pour en savoir plus

Les éléments donnés ici donnent une idée de la structure générale des commandes Scilab. Il faut savoir aussi aller chercher rapidement l'information dans l'aide du logiciel car vous ne disposerez d'aucun autre moyen le jour J. Vous y avez accès dans la barre de tâche ou alors tapez directement

```
help;//pour ouvrir l'aide
help [nom de la commande];
//pour arriver directement sur la description de celle-ci
```

En attendant, il vaut mieux s'entraîner tant qu'un moteur de recherche est à portée de main pour répondre rapidement à vos questions. Voici quelques pages internet en français, la première est la traduction officielle de l'aide de Scilab, les deux autres sont des introductions aux différentes fonctionnalités de Scilab avec beaucoup plus de pédagogie
http://help.scilab.org/docs/5.3.3/fr_FR/index.html
http://www.iecn.u-nancy.fr/~pincon/scilab/scilab.html
http://ljk.imag.fr/membres/Bernard.Ycart/polys/demarre_scilab/demarre_scilab.html

Ensuite, une simple recherche sur internet vous donne accès à de nombreux polycopiés avec des angles et des publics visés différents. Il devient alors facile de trouver un document qui vous conviendra.

Index

grand, 2

Binomiale, 2, 13

cdfbin, 13
cdfchi, 13
cdff, 13
cdfnor, 13
cdfpoi, 13
cdft, 13
Chi2, 13
clf, 5
cumsum, 2, 5

diag, 4
disp, 4, 15

else, 4
elseif, 4
Exponentielle, 2

find, 2
Fisher, 13
Fonction de répartition empirique, 11

Gaussienne, 2, 13
Géométrique, 2

help, 2, 16
histplot, 6

if, 4

legends, 8
length, 15

Marche aléatoire 2d, 8
Markov, 3
max, 8
mean, 15
min, 2
Multinomiale, 2

plot2d, 6
plot2d2, 5
Poisson, 2, 13
Processus de Poisson, 5

sqrt, 11
string, 4, 15
Student, 13
subplot, 9

tabul, 10
title, 8
toeplitz, 4

Uniforme, 2

xlabel, 8

ylabel, 8