Programmes Scilab pour l'option Probabilités

Auteur: Emeline LUIRARD
adapté du fichier de Laura GAY

Table des matières

Mettre les références des bouquins quand il y a.

1 Pour faire joli sur les graphes

Pour avoir plusieurs fenêtres de graphes qui s'ouvrent

Code Scilab

```
figure(1)
plot(...)
figure(2)
plot(...)
```

Pour la couleur et les titres des graphes

Code Scilab

```
xlabel('Mon titre en x','FontSize',8) // titre des abscisses le 8 est la couleur
ylabel('Mon titre en y','FontSize',8) // titre des ordonnées
```

ou encore

Code Scilab

```
plot2d([3:n],[B(3:n) D(3:n)],style=[5 2])
xtitle("Illustration␣de␣la␣consistance␣de␣l'estimateur␣de␣b","n","b_n") //titre du
 graphe
legends(["Valeur␣de␣b_n␣en␣fonction␣de␣n";"Droite␣d'équation␣y=b"],[5 2]) //
attention le deuxième argument est le même que dans le plot
```

legends sans opt demande où est-ce qu'il doit se mettre. On peut rajouter opt='ur' si on le veut en haut à droite par exemple.

Pour écrire en lateX

Code Scilab

```
title("$\text{Tracé␣de␣}␣\mathbb{P}(T>n)$")
```

Pour demander au jury quel paramètre il veut (un peu inutile)

Code Scilab

```
lambda=input('Entrer la valeur du paramètre lambda : ');
```

Pour afficher un paramètre par exemple en légende du graphe

A REVOIR

Code Scilab

```
€€€ string
```

Pour définir exactement la fenêtre du graphe

Code Scilab

```
plot2d(...,rect=[xmin,ymin,xmax,ymax])
```

Pour mettre plusieurs graphiques côte à côte

Code Scilab

```
début du programme
subplot(1,2,1)//premier graphe
blabla sur le premier plot
subplot(1,2,2)//deuxième graphe
blabla sur le second plot
```

Pour faire une fonction constante par morceaux (par exemple une fonction de répartition)

Code Scilab

```
plot2d2(abscisses,ordonnées,arguments)
```

2 Premiers trucs faciles

Pour ouvrir un fichier de données (normalement le jour J, il y aura les explications)

Code Scilab

```
[N,Texte,Feuilles,DebutFeuille]=xls_open('blabla.xls')
[valeurs, txt]=xls_read(N, DebutFeuille(i)) // On extrait la i ème feuille du
fichier N
```

2.1 Astuces dans les matrices et les vecteurs

Pour faire une matrice par blocs

Code Scilab

```
x=[1:2]; P=[x 2*x;3*x 4*x] // Ca donne la matrice P=[1,2,2,4;3,6,4,8]
```

Création d'une matrice ayant n lignes du vecteur x

Code Scilab

```
ones(n,1)*x
```

Pour calculer un produit scalaire

Code Scilab

```
x'*y
```

Pour faire un vecteur de taille 100 qui va de 1 à 8

Code Scilab

```
linspace(1,8,100)
```

Pour faire une matrice tridiagonale

Code Scilab

```
u=zeros(1,5); u(2)=8; //je crée ma sous-diagonale
v=zeros(1,5); v(2)=7; //je crée ma sur-diagonale
Q=3*eye(5,5)+ toeplitz(u,v)//avec le eye je fais ma diagonale
```

Tabul : étant donné un vecteur x je renvoie un vecteur à 2 colonnes avec toutes les occurrences de x et le nombre de fois où elles apparaissent

Code Scilab

```
X = [2 8 0 3 7 6 8 7 9 1 6 7 7 2 5 2 2 2 9 7]
m1 = tabul(X) //ordre décroissant je vais avoir un tableau avec à gauche 9 8 etc..
 et à droite le nb d'occurences 2 2 etc..
m2 = tabul(X, "i") //idem en ordre croissant
```

Pour calculer une norme euclidienne d'un vecteur

Code Scilab

```
alea3=2*rand(3,1000)-1;
norme3=diag(alea3'*alea3); //cela calcule x^2+y^2+z^2 pour chaque colonne du
vecteur
```

ou

Code Scilab

```
norm(v) // où v est un vecteur. norm a d'autres options selon quelle norme on veut
 calculer.
```

Extraire les lignes i_1 à i_2 d'une matrice

Code Scilab

```
A(i1:i2,:)
```

2.2 Divers

Booléens

Code Scilab

```
| // ou
& // et
bool2s(v) // transforme un vecteur de T et de F en vecteur de 0 et de 1.
```

Pour faire une moyenne d'un vecteur

Code Scilab

```
mean(X)
```

Pour ranger un vecteur par ordre décroissant

Code Scilab

```
gsort(X)
```

Pour ranger un vecteur par ordre croissant

Code Scilab

```
gsort(X, 'g', 'i')  // 'i' pour increasing. Le 'g' c'est pour trier toute la
matrice.
```

Pour inverser le sens de lecture d'un vecteur

Code Scilab

```
v(length(v):-1:1)
```

Pour extraire certains éléments d'un vecteur

Code Scilab

```
find(A>0.4) // extrait les éléments de A qui sont >0.4
vectorfind(M,v, 'r') // Cherche dans les lignes ('r') de M le vecteur v. Ca
renvoie les positions.
```

Pour faire π

Code Scilab

```
%pi
```

Solution de $Ax = b$

Code Scilab

```
x=A\b // ou
x=solve(A,b)
```

Liste des coefficients binomiaux pour éviter d'avoir à les coder

Code Scilab

```
2^n*binomial(1/2,n) // Liste des k parmi n
```

Pour simuler une probabilité conditionnelle $\mathbb{P}(.|\mathbb{A})$: on fait une suite de tirages successifs sous la proba $\mathbb{P}$, et on garde seulement les tirages pour lesquels A est réalisé.

Aux valeurs dans grand

Code Scilab

```
grand(m,n,'exp',1/lambda) // moyenne 1/lambda
grand(m,n,'exp',sigma) // écart type sigma
```

Ne pas hésiter à mettre des disp en fin de programme. Par exemple dans la ruine du joueur :

Code Scilab

```
disp('il reste au joueur la somme de '+string(X($))) // on rappelle que $ prend le
 dernier terme
```

3 Pour simuler des variables aléatoires

3.1 Une Cauchy (TP 1)

On utilise le résultat suivant : si F, la fonction de répartition de X, est inversible et U suit la loi uniforme sur $[0;1]$ alors $F^{-1}(U)$ [1]a même loi que X. Lorsque F n'est pas inversible, on utilise l'inverse généralisé $F^{-1}(u) = inf\{x \in \mathbb{R}, F(x) \geq u\}$
Par exemple, pour une Cauchy, on connait sa densité : $f(x) = \frac{a}{\pi(a^2+x^2)}$ et on peut donc aisément calculer sa fonction de répartition F qui est à constantes près une Arctan, qui s'inverse facilement. Si on veut montrer la convergence ps d'une Cauchy :

Code Scilab

```
C=tan(%pi*rand(1000,10)+%pi/2);
```

3.2 Une finie (Guide de survie)

Si je veux simuler deux lancers indépendants d'un dé à 6 faces de distribution $\left(\frac{1}{6} \quad \frac{1}{4} \quad \frac{1}{6} \quad \frac{1}{6} \quad \frac{1}{6} \quad \frac{1}{12}\right)$

Code Scilab

```
P=[1/6;1/4;1/6;1/6;1/6] // vecteur de taille 5 ! (scilab complète tout seul la
dernière valeur) Attention ça doit être un vecteur colonne !
grand(2,'mul',1,P) // on utilise la loi multinomiale, le 2 est le nombre d'expé
riences
```

Cela renvoie 2 colonnes (les 2 expériences) et un 1 est présent à la face qui est sortie. On a quelque chose du type $\begin{matrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{matrix}$.

1. Attention, parfois on ne la connait pas !

3.3 Une Bernoulli

Code Scilab

```
B=(rand(1,n)<p)
//ou
B=grand(1,n,'bin',1,p) // Une binomiale avec une seule expérience ;)
```

4 Outils pour les v.a.

4.1 Lois discrètes

Probabilités de la loi binomiale $\mathcal{B}(20, 0.6)$

Code Scilab

```
x=binomial(0.6,20) // le vecteur des probabilités
plot2d3(0:20,x) // diagramme en bâtons
bar(0:20,x) //diagramme en barres
```

On pourrait utiliser "bar" pour faire des histogrammes de lois discrètes mais il faut faire attention à ce que les 2 entrées soient de même longueur (moins pratique que "histplot").

4.2 Fonctions de répartition de lois connues

On utilise cdf. Par exemple, pour la fonction de répartition d'une loi $\mathcal{N}(0, 1)$:

Code Scilab

```
x=linspace(-3,3)
y=cdfnor('PQ', x, zeros(x), ones(x));
plot2d(x,y)
```

4.3 Quantiles de lois connues

On utilise aussi cdf mais en changeant les arguments.

Code Scilab

```
alpha=0.05
q=cdfnor('X',0,1, 1-alpha/2, alpha/2) // pour trouver q tel que P(|N|>q)=alpha où
N est une gaussienne centrée réduite.
q=cdfchi('X',7,alpha,1-alpha) // pour trouver le quantile d'une chi_2(7) pour
alpha.
```

5 Les convergences (TP)

5.1 Convergence en loi : convergence des fonctions empiriques (Guide de survie)

Code Scilab

```
    function illustration();
                x=grand(1,1000,'nor',0,1); //mon vecteur empirique
                x=tabul(x,'i');//i pour increasing // cf premiers trucs faciles
                x(:,2)=x(:,2)/1000;//normalisation des occurrences //la colonne de
 droite va être les probas de chaque occurrence
                clf;
                //on représente alors les fonctions de répartition empiriques,
avec la fonction cumsum
        xx=x(:,1);xxx=x(:,2) //je sépare chaque colonne
        [P,Q]=cdfnor("PQ",xx,zeros(length(xx),1),ones(length(xx),1)); //attention
la moyenne et la variance doivent être de même taille que X
        F=P //j'extrais ma fonction de répartition aux valeurs des occurrences
                plot2d(xx,cumsum(xxx),3); //je plot l'empirique
                plot2d(xx,P); //je plote la vraie
                //j'ai utilisé plot2d car je suis à densité mais si j'avais été
discrète j'aurais mis plot2d2 !
                title('Illustration de la convergence des fonctions de répartition
 empiriques','fontsize',4);
                xlabel('Axe des x','fontsize',4);
                ylabel('Axe des y','fontsize',4);
        endfunction;
```

5.2 Loi des grands nombres

C'est très simple, il s'agit juste de tracer le vecteur des sommes cumulées que l'on divise à chaque fois par le n i.e. on divise par un vecteur allant de 1 en 1 de même taille

Code Scilab

```
plot2d([1:1000],cumsum(grand(1,1000,'exp',2))./[1:1000],4)
```

Attention, ici on n'a fait qu'une seule expérience. Pour voir une convergence presque sûre il faut en faire plusieurs, par exemple 10. Pour cela, on utilise les matrices. Mais on peut aussi relancer la simulation plusieurs fois devant le jury.

Code Scilab

```
plot2d([1:1000]',cumsum(grand(1000,10,"nor",0,1),'r')./([1: 1000]'*ones(1,10)))
// cumsum(,'r') somme cumulée des colonnes
// cumsum(,'c') somme cumulée des lignes
```

5.3 Théorème central limite

5.3.1 Méthode facile

On fait un histogramme de nos expériences et on superpose à la densité de la Gaussienne.

Code Scilab

```
function TLC(k,n) //k est le nombre de réalisations et n ce qui modélise notre
infini
lambda=2 //donc moyenne = 0.5 variance = 2
clf();
Z=variance*sqrt(n)*(mean(grand(k,n,'exp',0.5),'c')-moyenne); //le terme gauche de
la convergence; ici mean va faire un vecteur colonne avec les moyennes de chaque
ligne donc de chaque réalisation
histplot(100,Z);
z1=min(Z); z2=max(Z);
abcisse=linspace(z1,z2,100)';
plot2d(abcisse,exp(-(abcisse.^2)/2)*(1/sqrt(2*%pi)),5); //tracé de la densité de
la loi N(0,1)
endfunction
```

histplot(nb de classes, vecteur de données), en général nb de classes $= \sqrt{\text{taille des données}}$ ou histplot(vecteur définissant les classes, vecteurs de données)

6 Méthode de rejet (TP 2)

A REMPLIR

7 Méthode de Monte-Carlo (TP 3)

7.1 Pour les intégrales

On utilise le résultat suivant qui découle de la loi des grands nombres : Si (X_n) suite de va iid de $\mathcal{U}([0,1]^m)$ et $f : [0,1]^m \to \mathbb{R}$ intégrable alors presque sûrement

$$\frac{1}{n}(f(X_1) + \cdots + f(X_n)) \underset{n\to+\infty}{\longrightarrow} \int_{[0,1]^m} f$$

La vitesse de convergence est donnée par le Théorème Central Limite : $\frac{\sqrt{n}}{\sigma}$ pour un coût de $O(nm)$. Il y a meilleure convergence lorsque la variance est petite.

Par exemple pour $\int_0^1 4\sqrt{1-x^2}\mathrm{d}x$:

Code Scilab

```
alea=rand(1,1000); //on fait nos Xn
clf()
plot2d([1:1000], cumsum(4*sqrt(1-alea.^2))./[1:1000],2)
plot2d([1 1000], [%pi %pi], 5)
```

Au pavé sur lequel on intègre. Par exemple pour des $\mathcal{U}([0,4]^m)$, on a presque sûrement

$$\frac{1}{n}(f(X_1) + \cdots + f(X_n)) \underset{n\to+\infty}{\longrightarrow} \frac{1}{4^m}\int_{[0,4]^m} f$$

7.2 Algorithme de Métropolis Hastings : théorie (Stage + CMMA) H.P.

Il s'agit d'appliquer ici la méthode de Monte Carlo via les chaînes de Markov. Nous détaillons ici cette méthode. Les algorithmes sont disponibles en annexe.
Cette méthode permet d'estimer l'espérance d'une variable aléatoire lorsque la loi est inconnue ou difficile à simuler ce qui est le cas ici.

Rappel :

[Théorème ergodique, Birkhoff] Soit μ une mesure.
Si f est une fonction positive telle que $\mathbb{E}_\mu[f] < \infty$
Si $(X_n)_{n\geq 0}$ est une chaîne de Markov irréductible et récurrente positive ;
Si elle admet une mesure μ pour mesure invariante ;
Alors $\frac{1}{n}\sum_{k=0}^{n-1} f(X_k) \underset{n\to+\infty}{\longrightarrow} \mathbb{E}_\mu[f]$

Notre but va donc être de construire une telle chaîne $(X_n)_{n\geq 0}$ avec pour mesure $\mu_{N,\beta,h}$.

On choisit Q_1 la matrice de transition telle que si ω est une configuration, $Q_1(\omega,\cdot)$ est la loi uniforme sur les configurations qui différent de ω d'au plus un spin.
Comme cette loi est uniforme, elle est symétrique. Alors on a, pour ω et ω' deux configurations :

$$Q_1(\omega,\omega') = Q_1(\omega',\omega)$$

On définit maintenant une nouvelle matrice de transition Q construite ainsi :

$$\forall \omega,\omega' \in \Omega_N, \quad \begin{array}{l} \text{si } \omega' \neq \omega, \quad Q(\omega,\omega') = Q_1(\omega,\omega')\left(1 \wedge \dfrac{\mu_{N,\beta,h}(\omega')}{\mu_{N,\beta,h}(\omega)}\right) \\ \text{sinon}, \quad Q(\omega,\omega) = 1 - \displaystyle\sum_{\omega'\neq\omega} Q(\omega,\omega') \end{array}$$

Autrement dit :

$$\forall \omega,\omega' \in \Omega_N, \quad \begin{array}{c} Q(\omega,\omega') = \begin{cases} Q_1(\omega,\omega') & \text{si} \quad \mu_{N,\beta,h}(\omega') > \mu_{N,\beta,h}(\omega) \\ Q_1(\omega,\omega')\dfrac{\mu_{N,\beta,h}(\omega')}{\mu_{N,\beta,h}(\omega)} & \text{sinon} \end{cases} \\ Q(\omega,\omega) = 1 - \displaystyle\sum_{\omega'\neq\omega} Q(\omega,\omega') \end{array}$$

La mesure de probabilité $\mu_{N,\beta,h}$ est réversible pour Q (donc invariante)

La chaîne $(X_n)_{n\geq 0}$ définie par la matrice Q est irréductible, récurrente positive.

Il s'agit maintenant de construire la matrice Q définie comme précédemment afin de lui associer une chaîne de Markov. Le théorème ergodique s'appliquera donc (on a fait en sorte que) et les simulations nous permettront d'approcher certaines espérances.

La construction de $(X_n)_{n\geq 0}$ s'effectue par récurrence. On détaille le cas $n = 0$ ici :

$$X_0 \xrightarrow[①]{\text{proposition}} Y_0 \xrightarrow[②]{\text{acceptation / rejet}} X_1 = (X_0 \quad \text{ou} \quad Y_0)$$

Etape ⓪ : Initialiser la matrice initiale de spins (configuration) notée X_0 (de préférence aléatoire)

Etape ① : Choisir la configuration Y_0 avec la loi $Q_1(X_0,\cdot)$. Pour cela, il suffit de tirer de manière uniforme une case de la matrice et d'inverser son spin. (en effet, $Q_1(\omega,\cdot)$ est la loi uniforme sur les configurations qui différent de ω d'au plus un spin.)

Etape ② : Accepter ou non la proposition avec la loi Q. En effet, la probabilité d'acceptation est

$$P(X_0, Y_0) = \begin{cases} 1 & \text{si} \quad \mu_{N,\beta,h}(Y_0) > \mu_{N,\beta,h}(X_0) \\ \dfrac{\mu_{N,\beta,h}(Y_0)}{\mu_{N,\beta,h}(X_0)} & \text{sinon} \end{cases}$$

On tire donc un nombre aléatoire dans $[0,1]$. S'il est inférieur à $P(X_0, Y_0)$, on renvoit $X_1 = Y_0$. Sinon, on renvoit $X_1 = X_0$.

On généralise ce procédé à un terme n quelconque de la chaîne, selon le schéma ci-dessous :

$$X_n \xrightarrow{\text{proposition}} Y_n \xrightarrow{\text{acceptation / rejet}} X_{n+1} = (X_n \quad \text{ou} \quad Y_n)$$

N.B. : Le rapport de la probabilité d'acceptation n'est pas compliqué à calculer :

$$\frac{\mu_{N,\beta,h}(Y_n)}{\mu_{N,\beta,h}(X_n)} = \exp\left(\mathcal{H}_{N,\beta,h}(Y_n) - \mathcal{H}_{N,\beta,h}(X_n)\right)$$

8 Chaînes de Markov (TP 4)

8.1 Le truc classique (Guide de survie)

On utilise la fonction grand pour simuler les N premières réalisations d'une chaîne de Markov de position initiale x_0 dans un espace fini de matrice de transition P.

Code Scilab

```
grand(N,'markov',P,x0) // Si on met un vecteur colonne pour x0, ça renverra les
différentes trajectoires sous la forme d'une matrice.
```

Attention, cela ne ressort pas la position initiale !
Attention à l'espace d'états, Scilab prend pour espace d'états $\{1, 2, \cdots, d\}$.
Quand l'espace d'états est trop grand, il vaut mieux utiliser la relation de récurrence pour une chaîne de Markov : $X_{n+1} = f(X_n, U_{n+1})$ où f est une fonction mesurable et (U_n) une suite de v.a. indépendantes et indépendante de X_0.

Lorsqu'on veut la matrice de transition d'une chaîne de Markov ayant p classes récurrentes de cardinaux n_i et n_t états transients communiquant tous entre eux.

Code Scilab

```
M=genmarkov([n1,..., np], nt)
```

8.2 Propriétés des états de la chaîne

A partir de la matrice de transition d'une chaîne de Markov, on peut récupérer des informations sur les états :

Code Scilab

```
[perm,rec,tr,indsRec,indsT]=classmarkov(P)
// tr: nb d'états transients.
// rec: vecteur de la taille des classes récurrentes.
// indsRec : vecteur des indices des états récurrents.
// indsT : vecteur des indices des états transients.
```

Cependant, sur certaines versions de Scilab, la fonction ne marche pas.
Pour mettre en évidence un état récurrent ou transient on peut simuler une longue trajectoire de la chaîne et représenter la proportion de temps passé en chaque état, avec la fonction tabul par exemple !

8.3 Mesure invariante

A partir de la matrice de transition P de la chaîne, on peut récupérer la proba invariante (si elle existe !)

Code Scilab

```
[x0, KerA]=linsolve(P'-eye(P),0) //x0+KerA est l'ensemble des solutions de (P'-id)
x=0.
y=KerA // Quand il y a irréductibilité, la dimension de Ker(P'-id) est 1, donc
KerA ne contient qu'un vecteur, qui est la mesure invariante.
z=y./sum(y) // pour récupérer la proba invariante.
```

On peut aussi approximer la mesure invariante, lorsqu'il y a convergence en loi de la loi de X_n vers la mesure invariante : on a

$$\mathbb{P}(X_n = x) \underset{n\to+\infty}{\longrightarrow} \pi(x)$$

On peut faire un histogramme empirique de la loi de X_n et de π. Et on ne dit pas "on voit que c'est proche" ! ! On fait plutôt un test du χ_2 puisqu'on a une loi discrête sur un nombre fini d'états.

8.4 Marche aléatoire sur $\mathbb{Z}^2$ (Guide de survie)

L'idée est la même que pour le dé. Partant d'un point, on a 4 directions possibles. On a en entrée un vecteur de proba (attention de taille 3 car la dernière est complétée automatiquement)

Code Scilab

```
    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];
        for k=1:nb_pas
            sens=grand(1,'mul',1,P); //je tire une direction avec la multinomiale
            X(:,k+1)=X(:,k)+M*sens;      //je rajoute ma direction à X
        end
        plot2d(X(1,:),X(2,:),2); //je plote la deuxième coordonnée de X en fct de
la première
    endfunction
```

On adapte très facilement ce programme sur $\mathbb{Z}$. Je l'écris quand même ici (on va ploter la position de X en fonction du temps par contre).

Code Scilab

```
function [X]=trajectoireZ(nb_pas,P);//P proba sur 1,2
    X=0;
    gauche=-1;
    droite=+1;
    M=[gauche droite];
    P(2)=[];
    for k=1:nb_pas
        sens=grand(1,'mul',1,P);
        X(k+1)=X(k)+M*sens;
    end
    plot2d([1:(nb_pas+1)],X,2);
endfunction
```

9 Processus de Poisson (TP 8)

9.1 Tracé d'un processus de Poisson jusqu'à un nombre de sauts

On fait les temps intersauts, les temps de sauts et on plot

Code Scilab

```
function poisson(nbsaut, lambda)
    E=grand(1,nbsaut,'exp',1/lambda) // temps intersauts
    T=cumsum(E) //temps de sauts
    plot2d2(T,[1:nbsaut]) // je plote de 1 à 20 en fonction de mes temps de sauts
endfunction
```

9.2 Tracé d'un processus de Poisson jusqu'à un temps t

Même principe mais on bidouille un peu pour que ça coupe au bon moment

Code Scilab

```
function poisson2(t)
    clf()
    T=0
    while T($)<t
        T=[T T($)+grand(1,1,'exp',1)]
    end
    T($)=20
    N=[0:length(T)-2, length(T)-2]
    plot2d2(T,N)
endfunction
```

ou encore en se servant de la loi même du processus : Sachant que $N_t = k$, la loi de $(T_1, \cdots, T_k)$ est celle d'un k échantillon ordonné de v.a. iid de $\mathcal{U}([0,t])$.

Code Scilab

```
function generateur(t,lambda)
    k=grand(1,1,'poi',lambda*t)
    T=-gsort(-t*rand(1,k)) // ou gsort(t*rand(1,k), 'g', 'i')
    plot2d2(T,[1:length(T)])
endfunction
```

9.3 Temps de saut

Si on veut simuler les temps de sauts d'un processus de Poisson $\mathcal{P}(\lambda)$ sur $[0,t]$

Code Scilab

```
function [T]=generateur(t,lambda)
    T=0
    while T($)<t
        E=grand(1,1,'exp',1/lambda) // on sait que le temps entre chaque saut suit
 une loi exponentielle
        T=[T T($)+E] // T va être le vecteur donnant le temps de chaque saut,
ainsi la taille de T est le nombre de sauts
    end
endfunction
```

10 Quantiles empiriques (TP 7)

A REMPLIR

11 Tests

11.1 Test de Kolmogorov-Smirnov (TP 5)

Attention ! Ce test ne marche que pour tester des lois à densité !

Principe : à chaque point x pour lequel on aura eu un saut on va comparer le valeur de la fonction empirique avec la théorique. Mais on connait les valeurs en les sauts : elles augmentent de $1/n$ à chaque fois (cf vert). Attention, en chaque point, on doit prendre le max entre les valeurs au dessus et en dessous pour prendre l'écart maximal (flêche bleu).

Premier programme : vérifier si la statistique suit une loi KS. Regardons si des observations suivent bien une loi exponentielle de paramètre λ. On a bien convergence vers une loi finie qui est KS dans le cas $\mu = \lambda$ et environ divergence sinon (ie des valeurs anormalement grandes).

Code Scilab

```
function kolmo(n,N,mu,lambda)
    D=zeros(1,N)
    for i=1:N
        X=grand(1,n,'exp',1/mu) // on crée n observations
        X=gsort(X,'g','i') // on les met dans l'ordre
        for j=1:n
            Y(j)=1-exp(-lambda*X(j)) // on crée la fonction de répartition thé
orique F(X) si elles sont bien de paramètre lambda
        end
        D(i)=sqrt(n)*max(max(abs([1/n:1/n:1]-Y')),max(abs([0:1/n:(1-1/n)]-Y'))) //
statistique du test de KS
    end
    clf()
    histplot(50,D)
endfunction
```

Deuxième programme : rejeter ou non l'hypothèse d'adéquation. X est le vecteur des observations (il suffit d'en avoir plus de 35 pour ce quantile à 95 %) et F est la fonction de répartition de la loi théorique contre laquelle on veut tester les observations.

Code Scilab

```
function testKS(X,F)
    X=gsort(X,'g','i')
    n=length(X)
    for j=1:n
        Y(j)=F(X(j))
    end
    D=sqrt(n)*max(max(abs([1/n:1/n:1]-Y')),max(abs([0:1/n:(1-1/n)]-Y')))
    if (D<=1.36) then disp('Ho n est pas rejetée')
        else disp('Ho est rejetée')
    end
endfunction
```

11.2 Test du χ_2 (TP 5)

En pratique on considère que l'approximation en loi par χ_2 est valide sous H_0 si $n * min(p_j) \geq$ sinon il vaut mieux regrouper les valeurs.

11.2.1 Adéquation

On a juste à calculer la grosse somme et à comparer avec le quantile.

Code Scilab

```
function [D,pval]=testchi2adeq(Q,P,n,erreur) //Q proba empirique. P proba thé
orique. n taille de l'échantillon.
    Y=(n.*P-n.*Q).^2
    Y=Y./(n.*Q)
    D=sum(Y)
    [p,pval]=cdfchi('PQ',D,length(Q)-1)
    if (D<cdfchi('X',length(Q)-1,1-erreur,erreur)) then disp('Ho n est pas rejetée
')
        else disp('Ho est rejetée')
    end
endfunction
```

On fait de la même manière pour le test d'indépendance.

12 Régression linéaire

A REMPLIR

13 Remarques

- Attention à ne pas écrire une égalité entre deux variables aléatoires à la place d'une égalité en loi.
- Avoir plus de réflexes sur les stats et les tests !
- Avoir du recul sur les simulations qu'on obtient. Vérifier l'intuition sur les graphes. Notamment, faire attention à ce que la somme de l'histogramme fasse bien 1.
- Ne pas dire "on voit que l'histogramme s'approche d'une loi uniforme" mais dire qu'on peut faire un test du χ^2 par exemple pour évaluer l'écart avec ce qu'on veut obtenir.
- Comment savoir la précision d'un estimateur, d'une approximation ? avec un intervalle de confiance ou avec un test.
- Faut-il expliquer le code avant de le lancer ? On peut l'expliquer mais on n'est pas obligé du tout !
- Peut-on faire référence au texte quand on ne veut pas recopier une formule par ex ? Oui on a le droit, ça ne sert à rien de recopier tout un tas de grosses formules au tableau, surtout si on se trompe en les recopiant.
- Une moyenne du temps à passer sur la simulation pendant la préparation : 1h30. Cela permet d'avoir des simulations consistantes tout en ne passant pas trop de temps dessus et en ayant le temps de faire des maths.