Démarrer en Matlab

B. Ycart

UFR Mathématiques et Informatique
Université René Descartes, Paris
ycart@math-info.univ-paris5.fr

Matlab (contraction de *Matrix Laboratory*) est basé sur le principe que tout calcul, programmation ou tracé graphique peut se faire à partir de matrices rectangulaires de nombres. A partir de cette idée simple, un très grand nombre de fonctionnalités ont été développées, et Matlab est devenu un environnement de calcul extrêmement répandu, dans tous les domaines d'applications. Le but de ce qui suit est d'aider le débutant en introduisant quelques unes des commandes les plus courantes. Il est conseillé de lire ce document après avoir lancé Matlab, en exécutant les commandes proposées une par une pour en observer l'effet. Les exemples ont été préparés à partir de la version étudiant 4 pour Windows. Compte tenu de leur caractère élémentaire, ils devraient fonctionner sur d'autres versions, moyennant quelques aménagements mineurs.

1 Vecteurs et matrices

En Matlab, tout est matrice : les scalaires sont des matrices 1×1, les vecteurs lignes des matrices $1 \times n$, les vecteurs colonnes, des matrices $n \times 1$. On peut rentrer un vecteur ligne en saisissant ses valeurs, séparées par des virgules ou des blancs. Pour une matrice, les lignes sont séparées par des points-virgules. La transposée est notée par une apostrophe. Elle permet en particulier de transformer un vecteur ligne en un vecteur colonne.

Les commandes peuvent être saisies ligne par ligne. Ajouter un point-virgule en fin de ligne supprime la sortie (pour éviter les longs défilements à l'écran). La touche "flèche haute" utilisée de façon répétée permet de rappeler les lignes de commande précédentes. L'exécution d'une ligne se fait simplement par la touche "Entrée". Une même séquence d'instructions peut continuer sur plusieurs lignes, si chacune se termine

par trois points. Les réponses sont affectées par défaut à la variable temporaire `ans` (answer).

```
x=[1,2,3]
x'
[x,x,x]
[x x x]
[x;x;x]
A=[1,2,3;4,5,6;7,8,9]
A'
A=[1,2,3;...
   4,5,6;...
   7,8,9]
```

Les commandes d'itérations permettent de construire des vecteurs de nombres séparés par des pas quelconques (positifs ou négatifs). On peut étendre un vecteur en affectant de nouvelles coordonnées, au-delà de sa taille. Les coordonnées intermédiaires sont nulles par défaut.

```
x=2:6
y=10:-1:5
z=1:0.3:pi
x(2)=y(3)
x(8)=sqrt(2)
z(3:5)=[3,2,1]
z(3:5)=[]
```

Vecteurs	
`n:m`	nombres de `n` à `m` par pas de 1
`n:p:m`	nombres de `n` à `m` par pas de `p`
`length(x)`	longueur de `x`
`x(i)`	`i`-ème coordonnée de `x`
`x(i1:i2)`	coordonnées `i1` à `i2` de `x`
`x(i1:i2)=[]`	supprimer les coordonnées `i1` à `i2` de `x`
`[x,y]`	concaténer les vecteurs `x` et `y`
`x*y'`	produit scalaire des vecteurs (lignes) `x` et `y`
`x'*y`	produit scalaire des vecteurs (colonnes) `x` et `y`

Matrices	
`size(A)`	nombres de lignes et de colonnes de `A`
`A(i,j)`	coefficient d'ordre `i,j` de `A`
`A(i1:i2,:)`	lignes `i1` à `i2` de `A`
`A(i1:i2,:)=[]`	supprimer les lignes `i1` à `i2` de `A`
`A(:,j1:j2)`	colonnes `j1` à `j2` de `A`
`A(:,j1:j2)=[]`	supprimer les colonnes `j1` à `j2` de `A`
`A(:)`	concaténer les vecteurs colonnes de `A`
`diag(A)`	coefficients diagonaux de `A`

Opérations matricielles	
`A'`	transposée de `A`
`rank(A)`	rang de `A`
`inv(A)`	inverse de `A`
`expm(A)`	exponentielle de `A`
`det(A)`	déterminant de `A`
`trace(A)`	trace de `A`
`poly(A)`	polynôme caractéristique de `A`
`eig(A)`	valeurs propres de `A`
`[U,D]=eig(A)`	vecteurs propres et valeurs propres de `A`
`+ -`	addition, soustraction
`* ^`	multiplication, puissance (matricielles)
`.* .^`	multiplication, puissance terme à terme
`A\b`	solution de `A*x=b`
`b/A`	solution de `x*A=b`
`./`	division terme à terme

Matrices particulières	
`zeros(m,n)`	matrice nulle de taille `m,n`
`ones(m,n)`	matrice de taille `m,n` dont les coefficients valent 1
`eye(n)`	matrice identité de taille `n`
`diag(x)`	matrice diagonale dont la diagonale est le vecteur `x`
`magic(n)`	carré magique de taille `n`
`rand(m,n)`	matrice de taille `m,n` à coefficients i.i.d, uniformes sur $[0,1]$
`randn(m,n)`	matrice de taille `m,n` à coefficients i.i.d, de loi normale $\mathcal{N}(0,1)$

```
b=1:5
b*b'
b'*b
b.*b
A=ones(5)+eye(5)
[A,b]
[A,zeros(5,2);zeros(2,5),eye(2)]
A*b'
A\b'
inv(A).b'
b/A
b*inv(A)
A.^4
A^4
exp(A)
expm(A)
diag(A)
det(A)
eig(A)
```

```
[U,D]=eig(A)
U*D*inv(U)

A=[1,2,3;4,5,6;5,7,9]
rank(A)
inv(A)
A*inv(A)
b=[1,2,3]
x=A\b'
A*x
x=b/A
x*A
```

Les entrées d'une matrice peuvent être des nombres complexes. Le nombre $\sqrt{-1}$ est affecté par défaut aux deux variables i et j. Leur contenu peut être changé. On pourra alors le restaurer ou affecter $\sqrt{-1}$ à une nouvelle variable.

```
A=[1,2;3,4]+i*[0,1;2,3]
A=[1,2+i;3+2i,4+3i]
A^2
abs(A)
real(A)
imag(A)
conj(A)
angle(A)
A'
A.'
i=1
A=[1,2;3,4]+i*[0,1;2,3]
ii=sqrt(-1)
A=[1,2;3,4]+ii*[0,1;2,3]
```

2 Fonctions

De très nombreuses fonctions sont disponibles. On peut en obtenir la description par `help`. Pour des exemples de programmation, on peut lister le code d'une fonction par `type`. On peut retrouver une fonction avec `lookfor`. Essayez aussi `demo`.

```
help help
help erf
help hist
type hist
lookfor eigenvalue
```

Les fonctions numériques s'appliquent en général à chaque terme d'un vecteur ou d'une matrice. Pour éviter la confusion avec leurs équivalents matriciels, la multiplication,

la puissance, la division doivent être précédées par un point pour s'appliquer terme à terme.

```
x=1:3
y=x+1
z=sqrt(y)
A=[x;y;z]
A.^2
A^2
```

Fonctions élémentaires		
sqrt	exp	log
sin	cos	tan
asin	acos	atan
round	floor	ceil
abs	angle	conj

Certaines fonctions s'appliquent à l'ensemble d'un vecteur, ou à une matrice, colonne par colonne.

```
x=rand(1,5)
mean(x)
std(x)
median(x)
sort(x)

A=rand(3)
sort(A)
sort(A')
max(A)
max(A')
max(max(A))
```

Fonctions vectorielles	
max	Maximum
min	Minimum
sort	Tri par ordre croissant
sum	Somme
prod	Produit
cumsum	Sommes cumulées
cumprod	Produits cumulés
mean	Moyenne
median	Médiane
std	Ecart-type

3 Graphiques

Le principe général des représentations graphiques est de se ramener à des calculs sur des matrices ou des vecteurs. Ainsi la représentation d'une fonction de $\mathbb{R}$ dans $\mathbb{R}$ commencera par la création d'un vecteur d'abscisses, en général régulièrement espacées, auxquelles on applique la fonction pour créer le vecteur des ordonnées. Pour la représentation d'une surface, il faudra créer la matrice des points d'une grille rectangulaire dans $\mathbb{R}^2$. La commande `hold` permet de superposer des tracés successifs sur une même figure.

Dimension 2	
`plot`	représenter des listes de points
`fplot`	représenter des fonctions
`polar`	représenter en coordonnées polaires
`quiver`	champ de vecteurs
`bar`	diagramme en bâtons
`hist`	histogramme
`-,--,:,-.,.,+,*,o,x`	types de courbes
`y,m,c,r,g,b,w,k`	couleurs

```
x=-pi:0.1:3*pi; y=x.*sin(x);
plot(x,y)
plot(x,y) axis([-pi,3*pi,-6,9])...
     xlabel('x') ylabel('y') title('x sin(x)')
plot(x,y,x,2*y)
plot(x,[y;2*y])
plot(x,y,'r--',x,2*y,'g+')

fplot('[x*sin(x),2*x*sin(x)]',[-pi,3*pi])
fplot('x*sin(x)',[-pi,3*pi],'b-') hold on
fplot('2*x*sin(x)',[-pi,3*pi],'yo') hold off

t=0:0.1:2*pi;
plot(sin(t),sin(2*t))
plot(sin(t),sin(2*t),'c-')
polar(t,t)
```

Dimension 3	
`plot3`	lignes et points en dimension 3
`contour`	courbes de niveaux en projection
`contour3`	courbes de niveaux en dimension 3
`meshgrid`	grille de représentation
`mesh`	représenter des points sur une grille
`meshc`	grille et courbes de niveaux
`surf`	représenter une surface
`surfc`	surface et courbes de niveaux

```
[x,y] = meshgrid(-pi:0.2:pi,-pi:0.2:pi);
z = sin(x.*y);
contour(z)
contour3(z)
mesh(z)
surf(z)
surf(z), colormap(hot)
surf(z), shading interp
surfc(z)
meshc(z)

t=0:0.01:2*pi;
x=sin(t); y=sin(2*t); z=sin(3*t);
plot3(x,y,z)
u=0:0.1:1;
x=u'*x; y=u'*y; z=u'*z;
surf(x,y,z)
```

4 Calcul numérique

Matlab n'est pas à la base un langage de calcul formel, comme Mathematica, Maple et Mupad. Il dispose par contre d'un grand nombre de fonctions permettant de résoudre numériquement les problèmes usuels. Les fonctions les plus courantes sont les suivantes.

Différentiation	
`diff`	Dérivées
`polyder`	Dérivée d'un polynôme
`gradient`	Gradient
`del2`	Laplacien en dimension 2

```
[x,y] = meshgrid(-2:.2:2, -2:.2:2);
z = x .* exp(-x.^2 - y.^2);
[px,py] = gradient(z,.2,.2);
contour(z),hold on, quiver(px,py), hold off
```

Intégration	
`trapz`	Méthode des trapèzes
`quad`	Méthode de Simpson
`quad8`	Méthode de Newton-Cotes

```
x = [-1:0.01:0];
y=exp(x);
trapz(x,y)
quad('exp',-1,0)
quad('exp',-1000,0)
quad8('exp',-1000,0)
```

Résolutions et optimisation	
`fzero`	Equations à une variable
`roots`	Racines d'un polynôme
`fmin`	Minimum sur une variable
`fmins`	Minimum sur plusieurs variables

```
fzero('cos',0)
fzero('cos',2)
fmin('cos',0,4)
fmin('cos',4,10)
```

Matlab permet également d'accéder à une version simplifiée de Maple, et donc au calcul symbolique. Certaines fonctions, comme `solve` et `diff`, peuvent retourner selon le cas des solutions symboliques ou numériques.

```
help solve
solve('a*x^2+b*x+c')
x=solve('x^5+2*x+1')
numeric(x)
[x,y] = solve('x^2+2*x*y+y^2=4','x^3+4*y^3=1')
numeric([x,y])
solve('cos(x) = x')

help diff
der=diff('sin(x^2)')
fplot(der,[0:1])
xh=[0:0.01:1];
yh=diff(sin(xh.^2))/0.01;
xh(length(xh))=[]
plot(xh,yh,'w.')
diff('sin(a*x^2)')
diff('sin(a*x^2)','a')
diff('sin(a*x^2)','a',2)
```

5 Fichiers

Matlab travaille à partir d'un répertoire de base, qui est donné par les commandes `pwd` ou `cd`. C'est là qu'il va chercher par défaut les fichiers à charger ou à exécuter. La liste des répertoires accessibles est fournie (et peut être modifiée) par `path`. Il faut bien sûr s'assurer que les fichiers que l'on souhaite charger se trouvent dans un répertoire accessible. Pour cela, on peut soit changer le répertoire courant par `cd`, soit rajouter le répertoire souhaité par `path`. Une solution plus radicale consiste à rajouter les chemins d'accès souvent utilisés dans le fichier d'initialisation de Matlab (`startup.m` ou bien `matlabrc.m`. L'exemple ci-dessous concerne Windows.

```
pwd
cd
dir
what
cd ..
cd
cd toolbox\stat
dir
path
help path
path(path,'c:\matlab\toolbox\travail') % Windows
path(path,'~/matlab/toolbox/travail')  % Linux
```

Il est conseillé de maintenir ouvertes deux fenêtres, la fenêtre matlab, et une fenêtre d'édition (Emacs sous Linux, WordPad sous Windows par exemple). On peut alors entrer des comandes dans la fenêtre d'édition et les copier-coller dans la fenêtre Matlab. Matlab distingue trois sortes de fichiers.

1. *Les fichiers de sauvegarde.* Ils ont une extension `.mat`, et ne sont lisibles que par Matlab. Ils sont crées par la commande `save` et rappelés par `load`. Ceci permet de reprendre un calcul en conservant les valeurs déjà affectées. On peut aussi sauver des variables dans un fichier texte avec l'option `-ascii`.

   ```
   A=rand(10); b=rand(10,1);
   save res1 A            % sauve A dans le fichier res1.mat
   save res2 A b          % sauve A et b dans res2.mat
   load res1              % charge les variables sauvees dans res1
   save res1.dat A -ascii % sauve la matrice A au format ascii
   load res1.dat          % affecte a res1 le contenu de res1.dat
   ```

2. *Les fichiers d'instructions.* Ce sont des fichiers texte avec une extension `.m`. Il sont édités par l'éditeur courant. Ils contiennent des suites d'instructions matlab, qui sont exécutées les unes après les autres. Par exemple, sauvez dans le répertoire courant les trois lignes suivantes sous le nom `losange.m`.

   ```
   x=[0 -1 0 1 ; -1 0 1 0]
   y=[-1 0 1 0 ; 0 1 0 -1]
   plot(x,y)
   ```

   La commande `losange` affichera `x`, puis `y`, puis tracera un losange.

3. *Les fichiers de fonction.* Comme les fichiers d'instructions, ce sont des fichiers texte avec l'extension `.m`. Leur syntaxe est particulière. Ils contiennent la définition d'une fonction, et portent le nom de cette fonction. Par exemple le fichier `cloche.m` pourra contenir

```
function d = cloche(x)
% CLOCHE densite de la loi normale N(0,1)
%        cloche(x) retourne (1/sqrt(2*pi)) exp(-x^2/2)
%
d = (1/sqrt(2*pi))*exp(-x.^2/2);
```

Si ce fichier est placé dans un répertoire accessible, la fonction `cloche` devient une fonction de Matlab, comme toutes les autres. Remarquez le texte placé en commentaire après la première ligne, qui est le contenu de l'aide pour la nouvelle fonction.

```
help cloche
x=-3:0.1:3; y=cloche(x);
plot(x,y,'wo')
fplot('cloche',[-3,3])
quad(cloche,-5,1.96)
```

6 Programmation

Matlab propose toutes les commandes des langages de programmation classiques. La philosophie est celle d'un langage fonctionnel. Au lieu de créer un logiciel avec programme principal et procédures, on étend le langage par les fonctions dont on a besoin. Toutes les fonctions, même celles livrés avec le langage, se présentent sous forme de fichiers texte, et on peut donc les afficher à l'écran (commande `type`) pour avoir des modèles de programmation. Le signe `%` met en commentaire le reste de la ligne.

Certaines erreurs difficiles à trouver proviennent de confusions entre noms de variables ou de fonctions. Matlab garde en mémoire tous les noms introduits tant qu'ils n'ont pas été libérés par `clear`. Il est donc prudent de donner des noms assez explicites aux variables. Les variables introduites dans la session sont globales. Par défaut, toutes les variables introduites à l'intérieur d'une fonction sont locales. Pour passer une même variable d'une fonction vers une autre, il faut utiliser la commande `global`.

Quand on a déjà appelé une fonction ou une procedure, et que l'on est amené à la modifier, par exemple pour corriger une erreur, il est fréquent que la modification ne soit pas prise en compte lors du prochain appel. On peut alors libérer le nom de la fonction par `clear` ou plus radicalement, quitter la session pour en ouvrir une autre.

```
x=[1,2]; A=[1,2,3,4];
who
whos
clear x
who
clear
who
```

Pour comparer l'efficacité des algorithmes, on peut utiliser `tic` et `toc` qui permettent de compter le temps CPU écoulé, mais ceci ne donne qu'une indication approximative. Matlab permet aussi de compter les opérations en virgule flottante par `flops`.

```
A=rand(50);
b=rand(50,1);
flops(0), x=A\b; flops
flops(0), x=inv(A)*b; flops
tic, x=inv(A)*b; toc
```

Commandes principales				
Pour	`for`	`x=vecteur`	`instruction;`	`end`
Tant que	`while`	`relation`	`instruction;`	`end`
Si	`if`	`relation`	`instruction;`	`end`

La boucle `for` peut aussi s'appliquer à une matrice `A`. Dans

`for x=A instruction; end`,

l'instruction sera exécutée pour `x` prenant successivement comme valeurs les colonnes de `A`.

Opérateurs logiques			
`==`	$=$	`~=`	$\neq$
`<`	$<$	`>`	$>$
`<=`	$\leq$	`>=`	$\geq$
`&`	et	`|`	ou
`~`	non	`xor`	ou exclusif

Les booléens sont `0` et `1`. Les relations logiques sont matricielles, les comparaisons étant faites élément par élément. Si `A` et `B` sont deux matrices de même taille, `A<B` est la matrice des booléens `A(i,j)<B(i,j)`. Utilisée dans un test, cette matrice se comportera comme le booléen vrai si pour tout `(i,j)`, `A(i,j)<B(i,j)`. On peut modifier ceci en utilisant les opérateurs `any` et `all` qui s'appliquent colonne par colonne. Par exemple `all(any(A<B))` sera vrai si dans chaque colonne de `A`, au moins un des éléments est inférieur à l'élément correspondant de `B`.

```
x=rand(2,10)
p=x<0.5*ones(2,10)
p1=p(1,:); p2=p(2,:);
p1 | p2
any(x<0.5*ones(2,10))
p1 & p2
all(x<0.5*ones(2,10))
```

Voici deux fonctions, qui seront utilisées pour les exemples de simulation dans la section suivante. La première fonction doit être placée dans le fichier `ech_dist.m`.

```
function e = ech_dist(x,d,m,n)
%ECH_DIST echantillon aleatoire de loi donnee
%      ECH_DIST(x,d,m,n) retourne une matrice de taille mXn dont les
%      coefficients sont des realisations independantes de la loi
%      sur x specifiee par le vecteur d, normalise a 1.

if nargin==3, n=m; end           % pour engendrer une matrice carree

taille=min(length(x),length(d)); % ajuster les longueurs
x=x(1:taille);
d=d(1:taille);

loi = d/sum(d);                  % normaliser par la somme
loi=cumsum(loi);                 % calculer la fonction de repartition

for i=1:m
     for j=1:n
          k=1;
          r=rand(1,1);           % appel de random
          while r>loi(k)         % simulation par inversion
               k=k+1;
          end
          e(i,j)=x(k);
     end
end
```

La seconde fonction doit être placée dans le fichier `freq.m`.

```
function [v,f] = freq(ech)
%FREQ  Frequences empiriques d'un echantillon.
%      FREQ(ech) Calcule les frequences d'apparition des valeurs
%      differentes de ech. Retourne un vecteur des valeurs
%      differentes de ech et un vecteur des frequences
%      correspondantes.

taille=length(ech);              % taille de l'echantillon
v=[ech(1)];                      % valeurs differentes
for k = 2:taille                 % parcourir l'echantillon
     if ech(k)~=v                % la valeur v(k) est nouvelle
          v = [v,ech(k)];        % la rajouter
     end;
end;
v = sort(v);                     % trier les valeurs trouvees
nbval=length(v);                 % nombre de valeurs differentes
```

```
effectifs = [];                  % effectifs des valeurs
for k = 1:nbval                  % parcourir les valeurs
     e = length(find(ech==v(k)));% calculer l'effectif de la valeur k
     effectifs = [effectifs,e];  % le rajouter
end

f=effectifs/sum(effectifs);      % calculer les frequences
```

7 Probabilités et statistiques

7.1 Version standard

Dans la version standard, sont disponibles essentiellement

- diagrammes en bâtons et histogrammes (`bar` et `hist`)
- moyenne, médiane, écart-type et matrice de covariance (`mean`, `median`, `std` et `cov`),
- les générateurs pseudo-aléatoires uniformes et gaussiens (`rand` et `randn`),
- les fonctions `erf` et `erfinv` à partir desquelles on obtient facilement la fonction de répartition Φ et la fonction quantile Φ^{-1} de la loi normale $\mathcal{N}(0,1)$.

$$\begin{aligned}
\texttt{erf}(x) &= \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt \\
&= \frac{2}{\sqrt{2\pi}} \int_0^{\sqrt{2}x} e^{-s^2/2}\, ds \\
&= 2\Phi(\sqrt{2}x) - 1 \; .
\end{aligned}$$

Soit :

$$\begin{aligned}
\Phi(x) &= \frac{1}{2}(\texttt{erf}(x/\sqrt{2}) + 1) \; , \\
\Phi^{-1}(u) &= \sqrt{2}\,\texttt{erfinv}(2u-1) \; .
\end{aligned}$$

```
x=rand(1000,3);
mean(x)
median(x)
std(x)
covar = cov(x)
ectinv = diag(1./std(x))
correl = ectinv*covar*ectinv

x=randn(1000,3);
```

```
cov(x)
A=[1,1,-1;-1,-1,1;1,-1,1]
y=x*A;
cov(y)
A*A'

x=[-2.5758,-2.3263,-1.96,0,1.96,2.3263,2.5758]
u=(erf(x/sqrt(2))+1)/2
u=[0.005,0.01,0.025,0.5,0.975,0.99,0.995])
x=(erfinv(2*u-1)*sqrt(2)
```

Voici quelques exemples de simulations, dont certaines utilisent les fonctions `freq` et `ech_dist` données dans la section précédente.

```
pileface = 2*(rand(1,1000)<0.5)-1;
freq(pileface)
gains = cumsum(pileface)
plot(gains)

bin=sum(rand(4,1000)<0.5);
[v,f] = freq(x)
bar(v,f)

x = ech_dist(['a','b','c'],[0.5,0.3,0.2],1,100)
x = ech_dist(['a','b','c'],[0.5,0.3,0.2],1,1000);
[v,f] = freq(x)
bar(f)
x = ech_dist([1:10],ones(1,10),1,1000);
[v,f] = freq(x)
bar(v,f)

unif=rand(1,4000);
sum((unif>0.3)|(unif<0.7))/4000
moyennes=cumsum(unif)./[1:4000];
plot(moyennes)
hist(unif)

expo=-log(unif);
mean(expo)
median(expo)
std(expo)
hist(expo)
hist(expo,[0.2:0.5:4.2])
geom = ceil(expo);
[v,f] = freq(geom)
```

```
bar(v,f)

gaus=randn(1,4000);
sum((gaus>-0.5)|(gaus<1.2))/4000
(erf(1.2/sqrt(2))-erf(-0.5/sqrt(2)))/2
mean(gaus)
std(gaus)
hist(gaus)
hist(gaus,[-3:6/20:3])

plot(rand(1,2000),rand(1,2000),'r.')

tria1=max(rand(2,4000));
hist(tria1)
tria2=min(rand(2,4000));
hist(tria2)
plot(tria1,tria2,'r.')

x=randn(1,4000);
y=randn(1,4000);
plot(x,y,'m.')
r=0.8
z=sqrt(1-r^2)*x+r*y;
plot(x,z,'r.')
r=-0.8
z=sqrt(1-r^2)*2*x+r*y;
plot(x,z,'m.')
```

7.2 Stixbox

Stixbox est une toolbox, soit un ensemble de fonctions, qui couvre l'essentiel des besoins élémentaires en probabilités et statistiques. Développée par Anders Holtsberg, elle est téléchargeable gratuitement depuis plusieurs sites. On y trouve en particulier les densités, fonctions de répartitions, fonctions quantiles et générateurs aléatoires, relatifs aux lois de probabilités les plus classiques.

Traitement des lois de probabilités				
Famille de lois	densité ou probabilités	fonction de répartition	fonction quantile	générateur
Béta	`dbeta`	`pbeta`	`qbeta`	`rbeta`
Binomiale	`dbinom`	`pbinom`	`qbinom`	`rbinom`
Chi-deux	`dchisq`	`pchisq`	`qchisq`	`rchisq`
Fisher	`df`	`pf`	`qf`	`rf`
Gamma	`dgamma`	`pgamma`	`qgamma`	`rgamma`
Géométrique				`rgeom`
Hypergéométrique	`dhypg`	`phypg`	`qhypg`	`rhypg`
Kolmogorov-Smirnov		`pks`		
Normale	`dnorm`	`pnorm`	`qnorm`	`rnorm`
Poisson				`rpoiss`
Student	`dt`	`pt`	`qt`	`rt`
Weibull				`rexpweib`

On y trouve aussi des intervalles de confiance et des tests de base, ainsi que quelques outils graphiques, comme la régression linéaire ou polynomiale (`linreg`) et des ajustements graphiques de distributions (`qqplot`, `qqnorm`). Des ensembles de données de démonstration sont proposés.

```
help stixbox
stixdemo

x=sum(rand(12,500))-6;
help qqnorm
qqnorm(x,'.')
y=randn(1,500);
help qqplot
qqplot(x,y,'.')
x=sort(x);
y=sort(y);
help linreg
linreg(y,x')
```