TP Probabilités-Statistique n°1

L'objectif de ce TP est de vous familiariser avec le logiciel Matlab (qui n'est pas libre...). Matlab n'est pas à la base un langage de calcul formel comme Mathematica ou Maple. Il s'agit d'un interpréteur de commandes écrites en langage Matlab. Ce logiciel a été conçu pour faciliter les opérations sur les vecteurs, matrices et tableaux.
Le logiciel libre "octave" devrait fonctionner de manière identique (mais sans jolie interface graphique), et normalement les fichiers pour l'un devraient être compatibles avec l'autre. Pour les probabilités, on utilisera les fonctions du package gratuit "stixbox" que l'on peut facilement télécharger.
Vous pouvez aussi aller travailler au CREMI quand vous voulez, et même vous connecter à distance sur les machines du CREMI depuis chez vous, en suivant les consignes sur l'intranet du site du CREMI.
Après les TPs, les corrigés seront sur ma page
http://www.math.u-bordeaux1.fr/~chabanol/stat.html

1. Fonctionnement des TP

La commande suivante à taper dans un terminal vous ouvrira une fenêtre, dans laquelle vous aurez à tout instant une copie de mon écran :

```
krdc vnc://nom_du_serveur
```

(Je vous communiquerai le nom_du_serveur à chaque séance)

2. Pour commencer

Cela peut être une bonne idée de commencer par créer un répertoire où seront stockés tous vos fichiers matlab, et de travailler dans ce répertoire.
Lancer matlab : >matlab &
Lorsqu'on lance Matlab une fenêtre divisée en trois parties s'ouvre. La première partie est le *Workspace* où sont stockées les variables. La deuxième est le *Command History*, qui donne un historique de l'ensemble des commandes utilisées. Enfin la dernière partie est le *Command Window* qui est l'endroit où sont tapées les instructions et où s'affichent les résultats. Dans cette sous fenêtre on peut remarquer le *prompt* (») qui indique que Matlab attend des instructions.

Pour obtenir de l'aide :

- **Menu Help** puis **Product help** : ouvre une fenêtre d'aide contenant toutes les commandes
- **help** nom : décrit la fonction nom.m

Pour gérer les fichiers ou les répertoires on peut utiliser la petite fenêtre associée à la commande window.

Matlab conserve l'historique des commandes. Il est donc possible de récupérer des instructions déjà saisies et ensuite de les modifier dans le but de les réutiliser. En double cliquant sur une de ces instructions passées, on la réexécute.
Bon à savoir : on peut aussi toujours faire F9 sur des instructions sélectionnées pour les exécuter.

Enfin, pour sauver certaines commandes de l'historique dans un fichier, on peut sélectionner des entrées dans cette fenêtre (on peut par exemple sélectionner tout l'historique d'une session) faire un clic droit et sauver dans un fichier ".m" à l'aide de "Create m-file".

↑, ↓, →, ← permet de se déplacer dans les lignes de commandes tapées dans la fenêtre de commandes. Cela évite de retaper des instructions !

3. Notions de base

3.1. **Créations et Manipulations de variables sous Matlab.**

`a=4;b=pi` La valeur 4 est affectée à a et la valeur π à b.
Le point virgule a pour effet que matlab n'affiche pas le résultat. Remarque : par défaut, `i` et `j` sont deux variables qui contiennent le nombre imaginaire i. Mais vous pouvez les utiliser quand même et changer leur valeur...

Matlab garde en mémoire les variables créées. On les voit dans le "Workspace" (avec leurs dimensions!). C'est très utile pour trouver des bugs... On peut également les afficher avec :

`who`	Liste les variables Matlab utilisées dans la session.
`clear b`	Destruction de la variable b
`who`	Vérification que la variable b n'existe plus.
`sqrt(a)`	Calcule la racine carrée de la variable a
`A=[1 2 3;4 5 6; 7 8 9]`	Création « à la main » de la matrice A.

Les colonnes sont séparées par des espaces (ou des virgules), les lignes par des point virgule.
`A(2,3)` désigne l'élément de la deuxième ligne et de la troisième colonne de A.

3.2. **Utilisation des deux points (:)**

`X=[1:20]`	X est le vecteur ligne contenant les entiers de 1 à 20.
`Y=[1:2:20]`	Y est le vecteur ligne contenant les entiers $1, 3, \cdots, 19$.

1. A votre avis que font les deux commandes suivantes ? Vérifier.
`2+1:5`
`2+(1:5)`

`b=A(2,:)`	b est le vecteur ligne contenant la deuxième ligne de A.
`c=A(:,3)`	c est le vecteur colonne contenant la troisième colonne de A.
`B=A(1:2,1:2)`	B est la première sous-matrice carrée d'ordre 2 de A .

2. Obtenir la matrice contenant les deux premières colonnes de A.

3.3. **Opérations sur les matrices et les vecteurs (faire help elmat).**

`length(A)`	donne la taille de A
`C=A'`	C est la matrice transposée de A
`A+C,A*C,2*A`	Addition et multiplications matricielles classiques
`n=4;I4=eye(n)`	Création de la matrice identité d'ordre 4
`B=zeros(n)`	B est la matrice nulle d'ordre 4
`D=ones(3,2)`	D est la matrice 3×2 ne contenant que des 1 .
`p=2; A^p`	Calcul de A^2
`det(A) ,trace(A) ,eig(A)`	Déterminant, trace et valeurs propres de A
`sum(X)`	Somme des éléments d'un vecteur
`cumsum(X)`	Vecteur $[X_1, X_1 + X_2, \ldots, X_1 + \ldots X_n]$ (sommes cumulées)

3. Obtenir un vecteur colonne de taille 10 dont tous les éléments valent 5. (utiliser `ones`)
5. Calculer la somme des 50 premiers entiers impairs
La plupart des fonctions matlab peuvent s'appliquer à des matrices ou à des vecteurs, et s'effectuent élément par élément : c'est le cas par exemple de `cos`, `sin`, `sqrt` et `exp`. Attention : si on veut calculer l'exponentielle d'une matrice il faut utiliser `expm`.
Cela marche aussi avec les opérateurs logiques : *essayer `A>2`, qui renvoie une matrice ne contenant que des 0 et des 1.*
6. Construire un vecteur contenant les racines carrées des entiers impairs de 1 à 9.
7. Construire un vecteur contenant les cosinus des entiers de 1 à 100. Utiliser `sum` et un opérateur logique pour savoir combien d'entiers de 1 à 100 ont un cosinus supérieur à 0.5.

3.4. **Utilisation du point (.)** L'opérateur point (.) permet de transformer une commande matricielle (typiquement, une opération "puissance", ou une multiplication) en une commande élément par élément. *Comparer par exemple `A^2` et `A.^2`, `A*C` et `A.*C`*

8. Construire (sans faire de boucle) un vecteur contenant les carrés des entiers de 1 à 10.
9. Construire (sans faire de boucle) un vecteur contenant les inverses des entiers de 1 à 10.
10. Construire (sans faire de boucle) un vecteur contenant les sommes harmoniques $1, 1+\frac{1}{2}, \ldots, 1+\frac{1}{2}+\ldots\frac{1}{10}$. *(penser à `cumsum`)*

3.5. **Premières commandes statistiques.**

`rand`	Générateur aléatoire associé à la loi uniforme $\mathcal{U}([0,1])$.
`randn`	Générateur aléatoire associé à la loi normale $\mathcal{N}(0,1)$
`X=rand(1,1000);`	X est un vecteur ligne contenant $n = 1000$ réalisations i.i.d. $\mathcal{U}([0,1])$
`D=rand(2,3);`	D est une matrice dont chaque élément est aléatoire uniforme sur $[0,1]$
`m=mean(X)`	Calcule la moyenne empirique (i.e. arithmétique) de X
`v=var(X)`	Calcule la variance empirique de X (avec une division par $n-1$)
`sigma=std(X)`	Calcule l'écart-type empirique de X (avec une division par $n-1$)

Un vecteur comportant n réalisations indépendantes d'une même loi est aussi appelé n-échantillon.
11. Générer un 1000-échantillon X de loi uniforme sur $[0,1]$. *Comparer sa moyenne empirique avec l'espérance $E[X]$.*
12. Comparer la moyenne empirique de X^2 avec $E[X^2]$ Mêmes questions avec une loi normale centrée réduite.

13. Générer un 20-échantillon de loi uniforme sur $[0,1]$ *et déterminer le nombre* N *d'éléments de votre échantillon qui sont inférieurs à* 0.3. *N est donc aléatoire. Quelle est sa loi ?*

3.6. **Boucles.** Toutes ces commandes permettent bien souvent de ne pas avoir à faire de boucle. Néanmoins ce n'est pas toujours possible... La syntaxe pour une boucle est par exemple

```
 for k=1:10;
INSTRUCTIONS
end
```

Il est assez pénible de taper des boucles directement dans la fenêtre de commande. La manière la plus pratique dès qu'on commence à avoir des suites d'instructions à taper est de créer un fichier ".m" avec ces instructions (un "script"); on peut ouvrir un tel fichier dans le menu "file", qui ouvrira l'éditeur de matlab.
14. Créer un script boucle.m qui contienne une boucle qui effectue 500 fois la question précédente et stocke à chaque fois le résultat N dans un vecteur, pour obtenir donc à la fin un vecteur de taille 500. Calculer la moyenne arithmétique de ce vecteur.

Pour exécuter le script, il suffit (après avoir sauvegardé le fichier) de taper "boucle" dans la fenêtre de commande. On peut aussi sélectionner les instructions du script et faire F9.

3.7. **Représentations graphiques.** Si `x` et `y` sont deux vecteurs de même taille, la commande `plot(x,y)` fournit le graphe formé à partir des points $[x_i, y_i]$. Ainsi :

`t=[0:1/10:3]; plot(t,exp(t)+3,'r')` graphe de la fonction $t \mapsto \exp(t)+3$ en rouge.

Si on souhaite rajouter une (ou des) courbe(s) sur la même figure, on utilise
`hold on` avant la (ou les) commandes `plot` suivante.

`hold on`	n'efface plus la figure
`clf`	efface le graphe.
`hold off`	annule la commande `hold on`
`figure`	ouvre une nouvelle fenêtre pour une nouvelle figure

15. Obtenir sur le même graphe les courbes représentatives de cos, et $x \mapsto 1 - \frac{x^2}{2}$. *sur* $[-\pi, \pi]$. *Essayez d'avoir des couleurs différentes.*

3.8. **Histogrammes.** Un histogramme peut être normalisé ou non, selon que les hauteurs des colonnes sont calculées de telle sorte qu'il soit d'aire 1, ou que les hauteurs sont les effectifs de chaque classe.
Pour comparer un histogramme à une densité de probabilités, il vaut mieux qu'il soit normalisé...

Si X est un vecteur, `histo(X)` trace un histogramme non normalisé de X. Pour le normaliser, il faut donner 4 paramètres, par exemple `histo(X,10,0,1)`. Le deuxième paramètre est le nombre de classes (ou de colonnes, en anglais "bins", de l'histogramme). En général on le prend de l'ordre de $\sqrt{n}$ où n est le nombre de réalisations. Le troisième paramètre vaut 0 ou 1 (en général 0 pour une loi continue, 1 pour une discrète mais ce n'est pas crucial), et le quatrième vaut 1 pour indiquer que l'histogramme est normalisé.

16. Construire plusieurs échantillons d'une loi normale centrée réduite en faisant varier le nombre de réalisations n entre 100 *à* 100 000 *. Tracer à chaque fois l'histogramme de X, en faisant éventuellement varier le nombre de classes de l'histogramme.*

On voit de plus que la forme de l'histogramme (et le fait qu'on reconnaît ou non la densité sousjacente) peut dépendre pas mal des choix des paramètres de histo; c'est pourquoi on travaillera en fait plus tard avec la "fonction de répartition empirique", qui a de bonnes propriétés.

3.9. **Les fonctions.** Une fonction matlab est un programme qui peut prendre en entrée des arguments et peut ressortir un ou des résultats en sortie. Comme les scripts, a priori une fonction s'écrit dans un fichier (par exemple à l'aide de l'éditeur) .m, **qui devra porter le nom de la fonction**. On peut par exemple mettre les commandes suivantes dans un fichier "toto.m" :

```
function y=toto(t);
% Ligne de commentaire éventuelle
y=t.^2-3;
% On utilise le . avant le carré car on veut que la fonction puisse s'appliquer à un vecteur
end
```

`toto(2)` renvoie alors la valeur 1.

17. Créer une fonction dans un fichier qui prend en entrée un réel x et calcule $\frac{1}{\sqrt{2\pi}}\exp(-\frac{x^2}{2})$.

18. Comparer sur le même graphique la courbe de la densité de la loi normale centrée réduite et un histogramme normalisé d'un échantillon de 1000 réalisations d'une loi normale centrée réduite.