TP 1
Initiation au logiciel "R"

L'objectif de ce TP est de découvrir le logiciel de statistiques et d'analyse de données "R". "R" s'avère être un outil simple d'utilisation et performant pour le calcul de probabilités et de statistiques sur des ensembles de données receuillies lors d'expériences.

1 Premiers pas avec "R"

Tapez les commandes suivantes dans la fenêtre "R".

```
a = 5
a
a + 3
b = 11
a + b
c = a + b
c
```

Vecteurs et Matrices

Les vecteurs et les matrices sont des ensembles de données de même type. Dans "R", les types de valeurs possibles sont les types numérique et chaîne de caractères. Précisons que dans "R", les vecteurs et les matrices sont indicées à partir de 1 contrairement au C/C++ où ils sont indicés à partir de 0.

Les vecteurs sont des tableaux unidimensionnels. Ils peuvent être créés à l'aide des fonctions `c()` et `rep()`. La fonction `c()` génére un vecteur dont les composantes lui sont passées en argument. La fonction `rep()`, quant à elle, sert à générer un vecteur dans lequel certaines valeurs sont répétées. Cette dernière fonction peut être très utile quand il s'agit d'initialiser un vecteur à une valeur particulière (zéro par exemple). Tapez les commandes suivantes et observez le résultat.

```
a = c(1,4,2,5,2,7,10,23,1,6)
a
length(a)
b = c(1 :10)
b
c = a + b
a
b
c
d = 1/c
```

```
d
e = 2*a
couleur = c("noir","rouge","vert","bleu")
couleur
length(couleur)
couleur[1]
couleur[3]
```

Remarquez que la commande `b = c(1 :10)` sert à initialiser le vecteur `b` par des entiers allant de 1 à 10. Tapez les commandes suivantes et observez le résultat.

```
f = rep(0,times=10)    //génère un vecteur de longueur 10 et initialiser à 0
x = c(1,3,5)
g = rep(x,times=2)
g
h = rep(x,each=3)
h
```

Une matrice est la réunion de plusieurs vecteurs de même dimension. La création d'une matrice peut être faite grâce à la fonction `matrix(val,nbligne,nbcol)`. `val` est la valeur d'initialisation de la matrice. Ce peut être une valeur unique (par exemple 0, 15, etc.) ou un ensemble de valeurs contenues dans un vecteur ou une autre matrice. L'accès aux éléments de la matrice se fait avec la commande `mat[i,j]` où `i` est l'indice de ligne et `j` l'indice de colonne. L'accès à une ligne entière se fait grâce à la commande `mat[i,]` où `i` est l'indice de la ligne concernée. L'accès à une colonne se fait avec la commande `mat[,j]`, `j` étant l'indice de la colonne.

La fonction `dim(mat)` fournit un vecteur contenant le nombre de lignes et de colonnes de la matrice `mat`.

Dans la fenêtre, tapez les commandes suivantes et observez le résultat.

```
mat = matrix(0,10,5)
mat
dim(mat)
mat[1,3]
mat[1,]
mat[,2]
mat[1,] = 11 //Affecte 11 à toutes les cases de la ligne 1
mat
val = c(-4 :5)
mat[,4] = val
mat
```

"R" donne la possibilité d'extraire d'un vecteur des ensembles de données suivant un critère donné. Par exemple pour un vecteur `x = (-2,0,1,-1,3,-4)`, on peut extraire de `x` l'ensemble des valeurs positives. Tapez les commandes suivantes et observez le résultat :

```
x = c(-2,0,1,-1,3,-4,3,-5)
```

```
x[x >= 0]
x[x == 3]
```

L'instruction `x[expression logique]` renvoie toutes les valeurs du vecteur x pour lesquels `expression logique` est vrai (par exemple $x[i] \geq 0$).

Gestion des variables

Les commandes `ls()` et `rm()` permettent respectivement d'afficher et de supprimer les variables créées au cours de l'utilisation du logiciel. Tapez dans la fenêtre les commandes suivantes et observez le résultat.

```
ls()
rm(a,b,c)
ls()
```

Définition de fonctions

"R" offre la possibilité de définir ses propres fonctions grâce la commande `function`. Tapez dans la fenêtre les commandes suivantes.

```
f←function(x){return(x*x)}
f(2)
a = c(0 :10)
b = f(a)
b
```

Contrairement aux langages de programmations comme C/C++, lors de la déclaration d'une fonction aucun type n'est précisé pour les paramêtres d'entrée. Le type par défaut des paramêtres et des variables temporaires utilisées dans une fonction est le type numérique (entier ou réel) mais on peut aussi utiliser des variables de type chaîne de caractères. L'exemple suivant illustre l'utilisation de variables temporaires. Tapez les commandes et observez le résultat.

```
h←function(x)
{
y1 = x*x ;
y2 = x+1 ;
z = y1/y2 ;
return(z) ;
}
f(2)
a = c(0 :10)
b = f(a)
b
```

Vous pouvez modifier le corps d'une fonction déjà écrite en utilisant la commande `edit()` : `g←edit(f)`. Un éditeur de texte s'ouvre alors en affichant les intructions de la fonction déjà existante. Une fois les modifications effectuées, enregistrez le fichier texte.

Une fênêtre s'ouvre dans laquelle vous pouvez effectuez vos modifications. Celles-ci sont enregistrées dans une nouvelle fonction g. Vous pouvez écraser la fonction f en tapant `f←edit(f)`.

Tapez dans la fenêtre de commande `f ←edit(f)` et modifier la fonction f de sorte que $f(x) = 2x + 1$.

Les tests et les boucles

"R" permet d'effectuer des tests sur des expressions logiques. Les syntaxes possibles sont les suivantes :

1. `if(condition){expression1}else{expression2}`
2. `if(condition){expression1}`

Il est aussi possible de réaliser des boucles `for`, `while` et `repeat`. La syntaxe pour chacune d'elle est la suivante.

- `for(nom_var in expression1){expression2}`
- `while(condition){expression}`
- `repeat{expression}`

Exemple :

```
//x est un vercteur de taille 100
for(i in 1 :100)
{
x[i] = i
}

i = 1 ;
while(i <= 100)
{
x[i] = 100 ;
i = i + 1 ;
}
```

Les opérateurs logiques *ET* et *OU* sont respectivement `&&` et `||`.

Ecrivez une fonction `Comb(n,p)` qui prend en paramêtre deux entiers n et p et calcule le nombre de combinaisons de p éléments parmi n. Remarquez que si $n < p$ alors $C_n^p = 0$. La factorielle d'un nombre est calculé grâce à la fonction `factotial()`.

Ecrivez une fonction `Pascal(n)` qui calcule et renvoie sous la forme d'un tableau à deux dimensions le triangle de Pascal pour un entier n donné (il n'est pas nécessaire d'utiliser la formule $C_n^p = C_{n-1}^p + C_{n-1}^{p+1}$).

Obtenir de l'aide sur une fonction dans "R"

La fonction `help(topic)` permet d'afficher l'aide en ligne sur la fonction désignée par "topic".
Exemple : `help(rep)` fournit l'aide sur l'utilisation de la fonction `rep()`.

Quitter "R" et sauvegarder la session courant

Pour terminer votre session "R", tapez la commande `q()`. A ce moment, le logiciel vous laisse la possibilité de sauvegarder les données relatives à votre session. Il enregistre en particulier toutes les variables et fonctions que vous avez créé au cours de la session. Ces données seront restorées lors de la session suivante.

Après avoir tapez `q()`, répondez par `y` (pour "yes") si vous souhaitez quitter et sauvegarder la session courante, `n` (pour "no") si souhaitez quitter sans sauvegarder la session et `c` (pour "continue") si vous ne souhaitez pas quitter la session.

Exercice

Ecrivez une fonction `discretise(a,b,n)` permettant de discrétiser un intervalle $[a, b]$ en $n-1$ sous-intervalles de même longueur. Le résultat de cette discrétisation sera renvoyé sous la forme d'un vecteur contenant les différents pas (ou "breaks"). Rapellons que le premier pas est a et le dernier est b. Le vecteur est donc de longueur $n+1$. Par exemple `discretise(0,1,10)` renverra le vecteur $x = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]$.

2 Passons aux Probabilités

"R" permet de manipuler de nombreuses lois de probabilités. Chacune de ces lois est associée à différentes fonctions qui calcule la probabilité d'un évènement, le quantile associé à une probabilité, etc. Si par exemple X est une variable aléatoire suivant une distribution uniforme sur l'intervalle $[a, b]$, alors la fonction `punif(x,min=a,max=b)` calcule la probabilité $P(X \leq x)$. La fonction `qunif(p,min=a,max=b)` calcule le quantile associé à la probabilité p c'est-à-dire le plus petit réel $x \in [a, b]$ tel que $P(X \leq x) > p$. La fonction `runif(n,min=a,max=b)` quant à elle génère une série de n nombres compris entre a et b distribués suivant une loi uniforme.

Chaque loi de probabilité intégré au logiciel "R" est associé à trois fonctions du type `pxxx()`, `qxxx()` et `rxxx()` qui calcule, respectivemenent, la probabilité, le quantile et une série de nombre aléatoires suivant ladite loi.

Les Histogrammes

Dans la fenêtre, tapez les commandes suivantes.

u = runif(100)

hist(u)

u = rnorm(100)

hist(u)

u = rnorm(1000)

hist(u)

u = rnorm(10000)

hist(u)

Vous aurez remarquer que plus l'échantillon est grand plus la forme de l'histogramme (la fréquence d'apparition des nombres) se rapproche de cellle d'une distribution gaussienne (loi normale).

Générez une série de 1000 nombres aléatoires suivants les distributions ci-dessous et tracez l'histogramme correspondant (consultez l'aide en ligne pour les paramêtres des différentes fonctions) :

- poisson de paramêtre λ `= 5` (fonction `rpois()`)
- binomiale de paramêtre `n = 20` et `p = 0.1` (fonction `rbinom()`)
- Bernouilli de paramêtre `p = 0.1` (loi binomiale sur 1 tirage : `rbinom(n,1,p)`)
- exponentielle de paramêtre λ `= 5` (fonction `rexp()`).

La fonction `rnorm()` par défaut génère des nombres aléatoires distribués suivant une loi normale centrée réduite ($\mu = 0$ et $\sigma = 1$). Générez 1000 nombres aléatoires distribués suivant une loi normale de moyenne 100 et d'écart-type 5 (consulter l'aide en ligne pour connaître les paramêtres nécessaires) et tracer l'histogramme correpsondant. Faites varier l'écart-type et observez l'étalement de la courbe.

Utilisez la fonction `runif()` pour générer des nombres suivants une distribution uniforme compris entre 1 et 100 (consultez l'aide en ligne). Tracez l'histogramme correspondant.

Utilisez les fonctions `mean()`, `var()` et `sd()` pour calculer respectivement la moyenne, la variance et l'écart-type de l'échantillon.

La fonction `hist()` utilise par défaut un algorithme particulier pour diviser l'intervalle dans lequel se trouve les valeurs de l'échantillon en plusieurs sous-intervalles ou classes. Lorsque le nombre de classes ne correspond pas à ce que l'on cherche, on peut fournir à la fonction `hist()` un ensemble de "breaks" qui détermineront les différentes classes à utiliser lors du comptage des effectifs. Cela nous sera très utile par la suite lors de tests d'hypothèse comme le test du $\chi 2$. Pour fixer le nombre de classes, on commence par discrétiser l'intervalle des valeurs en k classes (utilisez la fonction `discretise()`) et on fournit le vecteur obtenu en argument à la fonction `hist()`.

Exemple :

Commencez par générer un vecteur `u` contenant 1000 nombres entiers compris entre 0 et 100 distribués suivant une loi uniforme. Discrétisez ensuite l'intervalle $[0, 100]$ en $k = 5$ classes et stockez le résultat dans le vecteur `inter`. Appelez ensuite la fonction `hist()` en lui fournissant en argument l'échantillon `u` et le vecteur de "breaks" `inter` : `hist(u,breaks=inter)`. Observez le résultat. Refaites plusieurs fois la manipulation en faisant varier le nombre de classes.

Après son exécution, la fonction `hist()` renvoie un objet contenant plusieurs attributs. Chacun de ces attributs donnent les caractéristiques de l'histogramme. Ces attributs sont obtenus grâce à l'operateur `$`. L'un d'eux en particulier est le vecteur `counts` qui donne les effectifs obtenus dans chaque classe. Dans la fenêtre de commandes tapez

```
inter = discretise(0,100,20)
res = hist(u,breaks=inter,plot=FALSE)
res
res$counts
res$breaks
```

Vous remarquerez qu'à la suite de la dernière commande, le vecteur afficher correspond au vecteur `inter` passer en argument. Dans toute la suite, nous utiliserons le vecteur `counts` renvoyer par la fonction `hist()` pour déterminer des effectifs relatifs à des classes que nous aurons déterminé au préalable.

La fonction `plot()`

La fonction `plot()` sert à afficher des points dans une fenêtre graphique. Elle est aussi utilisée pour tracer des courbes de fonctions.

Tapez les commandes suivantes et observez le résultat.

```
f←function(x){x*x}
x = c(-10 :10)
y = f(x)
plot(x,y)
plot(x,y,type="l",col=2,pch="0")
```

Essayez différentes valeurs pour les paramêtres `col` et `pch` et observez le résultat.

```
x = discretise(-10,10,20)
y = f(x)
plot(x,y,type="l",col=2,pch="0")
```

3 Etude d'un générateur de nombres aléatoires

Nous allons maintenant étudier, en nous servant de "R", le générateur de nombres aléatoires `rand()` du C/C++ et déterminer quelle est la meilleure façon de générer des nombres aléatoires suivant une loi uniforme et compris entre 0 et 30. Nous utiliserons pour le cela des fichiers de données téléchargeables à l'adresse
`http://epoc.isima.fr/~diarrassouba/enseignements/proba_stat/`.

Avant de commencer l'étude, rappellons que la fonction `rand()` génère des nombres entiers suivant une distribution uniforme compris entre 0 et `RAND_MAX` $= 2^{31}$. Pour générer des nombres compris entre 0 et 30 nous pouvons utiliser l'une des deux méthodes suivantes :

1. `x = (int)(1.0*rand()/RAND_MAX)*30 //génère des nombres entre 0 et 30`
2. `x = rand()%31 // génère des nombres entre 0 et 30`

Le but de cet exercice est de voir si les nombres générés par ces deux méthodes suivent effectivement une distribution uniforme.

Q1. Téléchargez le fichier `rand.txt` qui contient un échantillon de 20000 nombres compris entre 0 et 1 générés par la méthode `x = 1.0*rand()/RAND_MAX`. Lisez les données contenues dans le fichier en utilisant la fonction `scan()` : `u = scan("rand.txt")`. Calculer la moyenne, la variance et l'écart-type empirique de l'échantillon `u` (utilisez les fonction `mean()`, `var()` et `sd()`) et vérifiez que ces valeurs s'approchent des valeurs théoriques de $\frac{1}{2}$, $\frac{1}{12}$ et $\sqrt{\frac{1}{12}}$, respectivement, obtenues pour une loi uniforme sur $[0, 1]$:

```
mean(u)
var(u)
sd(u)
```

Q2. Téléchargez les fichiers `rand30.txt` et `rand30-2.txt` qui contiennent chacun un échantillon de 20000 nombres entiers compris entre 0 et 30 générés en utilisant les méthodes 1) et 2), respectivement. Lisez, avec la fonction `scan()`, les données contenues dans ces fichiers et stockez-les dans deux vecteurs `u1` et `u2`.

Q3. Tracez les histogrammes associés aux échantillons `u1` et `u2` en divisant l'intervalle $[0, 30]$ en 20 classes. Précisez `right = FALSE` pour la fonction `hist()`. Cela permet de ne considérer

dans un intervalle $[a, b]$ que les valeurs x telles que $a \leq x < b$.

Q4. Calculer la moyenne, la variance et l'écart-type de chacun des échantillons `u1` et `u2` et comparez les résultats obtenus aux valeurs théoriques attendues : 15, 75 et $\sqrt{75}$, respectivement.

Q5. Vérification avec le test du χ^2. En utilisant la fonction `hist()`, divisez l'échantillon `u1` en 20 classes de longueur 1.5. Grâce aux résultats fournies par la fonction `hist()`, déterminez les effectifs de chaque classe et stockez le résultat dans un vecteur `obs`. Créez un vecteur `theo` de taille 20 qui contient les probabilités théoriques attendues pour une loi uniforme sur $[0, 30]$ dans chaque classe. Déterminez, grâce à la fonction `chisq.test()` la valeur du χ^2 associée à `u1`. Peut-on accepter l'hypothèse "`u1` suit une distribution uniforme sur $[0, 30]$" en considérant un risque d'erreur $\alpha = 1\%$?

Q6. Réalisez le test du χ^2 pour l'hypothèse "`u2` suit une distribution uniforme sur $[0, 30]$" avec un risque d'erreur $\alpha = 1\%$. Peut-on accepter cette hypothèse ?

Q7. Suivant les résultats du test du χ^2 pour les échantillons `u1` et `u2` que peut-on déduire sur la manière de générer efficacement en C/C++ des nombres aléatoires compris entre 0 et 30 suivant une distribution uniforme ?

4 Théorème central limite

Dans cette partie, nous allons observer de façon expérimentale le resultat du théorème central limite. Celui-ci stipule si X_i, $i = 1, ..., N$, sont des variables aléatoires indépendantes et identiquement distribuées de moyenne μ et d'écart-type σ, alors la distribution de la variable aléatoire $X = \sum_{i=1}^{N} X_i$ tend vers une loi normale de moyenne $N\mu$ et d'écart-type $\sigma\sqrt{N}$ lorsque N augmente. Dans la pratique, lorsque $N \geq 30$, on considère que la somme de N variables aléatoires tend vers une loi normale.

Pour le vérifier expérimentalement, nous allons générer une série de `N` variables distribuées suivant une loi exponentielle de paramêtre $\lambda = 0.5$. Pour chaque variable, nous génèrerons `n` éléments. Ecrivez la fonction suivante

```
Genere(n,N)
{
x = matrix(rep(0,times=n*N),nrow=N,ncol=n,byrow=TRUE) ;
y = rep(0,times=n) ;

for(i in 1 :N)
x[i,] = rexp(n,rate=0.5) ;

for(i in 1 :n)
y[i] = sum(x[,i]) ;

return(y) ;
}
```

Cette fonction génére `N` vecteurs $X_i = (X_i^j, j = 1, ..., n)$ de `n` éléments distribués suivant une

loi exponentielle de paramêtre $\lambda = 0.5$. Le vecteur `y` qui est renvoyé est de taille `n` et contient la somme des valeurs des échantillons X_i sur une colonne $j = 1, ..., n : y[j] = \sum_{i=1}^{N} X_i^j$.

Générez, avec cette fonction, des échantillons de `n = 10000` éléments pour `N` variables aléatoires pour les valeurs suivantes de `N`. Pour chaque essaie, tracez l'histogramme associé au vecteur `x` et observez l'évolution de l'histogramme. `N = 1, N = 5, N = 10, N = 20, N = 30`.

Calculez pour chaque essaie, la moyenne, la variance et l'écart-type de l'échantillon `x` et comparez-les aux valeurs $\frac{1}{\lambda}N$, $\frac{1}{\lambda^2}N$ et $\frac{1}{\lambda}\sqrt{N}$.