# Simulation avec R

Hassnaoui M.
UPO Lyon

24 juin 2017

**R-Project** est un logiciel **libre** et **gratuit** et à code source ouvert ( open source), pour le télécharger, il suffit de saisir cette adresse : http ://cran.univ-lyon1.fr/
**R-Project** est un logiciel de statistique crée par Ross Ihaka & Robert Gentleman.Il est à la fois un langage informatique et un environnement de travail. **R-Project** est un logiciel multi-plat-formes,il fonctionne sous UNIX ( et linux), Windows et Mac OS. **R-Project** est un logiciel dans lequel de nombreuses techniques statistiques modernes et classiques ont été implémentées ( statistque descriptive, test d'hypothèse, analyse de la variance, méthode de regression , analyse multivariée, etc...)
Une des grandes forces de **R-Project** réside dans ses capacités à combiner un lagage de programmation avec la possibilité de réaliser des graphiques de qualité. Les graphiques usuels s'obtiennent aisément a moyen de fonctions prédéfinies. Ces dernières possèdent un très grand nombre de paramètres permettant d'ajouter des titres, des légendes ...

Des 'packages' sont installés par défaut pour les fonctions élémentaires, par contre certaines commandes exigent le chargement de packages supplémentaires.

Le manuel de référence de **R-Project** est disponible directement sous **R-Project** : il suffit de taper le nom de la commande sur laquelle on veut des éclaircissements précédé d'un point d'interrogation.

Abstract : Dans ce document nous nous intéressons aux fonctions élémentaires de **R-Project** qui nous seront utiles pour résoudre quelques problèmes de probabilités à l'aide des simulations.

Dans un premier temps, nous passons en revue les fonctions de bases de **R-Project** pour la structure et la représentation des données, ensuite nous donnons, à travers des exemples, les bases de programmation avec **R-Project**, ainsi que les bases des statistiques descriptives et enfin nous terminons cet exposé par les lois de probabilité et les simulations classiques.

## 1 Fonctions de base du logiciel R-Project

**R-Project** est un logiciel gratuit, pour télécharger **R-Project**, saisir l'adresse suivante :

$$cran.univ - lyon1.fr$$

Pour les instructions élémentaires des 'packages' sont installés par défaut, par contre certaines commandes exigent le chargement de packages supplémentaires
Après le lancement de **R-Project** une console s'ouvre et le symbole > attend la saisie d'une commande

**Remarque**

**R-Project** est un langage fonctionnel, toutes les commandes sont considérées comme des fonctions. Chaque fonction exige la saisie de ses propres paramètres ( souvent les paramètres sont définis par défaut), par exemple pour la fonction log, il faut deux paramètres , le nombre qu'on veut calculer son logarithme et la base, par défaut la base est e.
Toutes les variables sont considérées comme des vecteurs. *La rédaction et la lecture des lignes de commande* ***R-Project*** *sont grandement facilitées par l'utilisation d'un éditeur spécifique*

Tout script qui contient plus de trois lignes doit être rédigé dans une nouvelle fenêtre. Dans le menu *fichier*, il suffit d'ouvrir *nouveau script / nouveau document*. Une fois la saisie est terminée , l'exécution du script se fait à travers le menu *édition*, sélectionner les instructions à exécuter puis choisir *exécuter la selection*

## 2 Utiliser R comme une calculatrice

1. Une simple opération de type :$a * b - c$ puis arrondir à 2 chiffres

R code
```
a<-sqrt(17) # racine carrée
b<-exp(1.5) # exponentielle
c<-log(13) # logarithme népérien par défaut
c<-log(13,exp(1))# ici on précise la base
d<-log(2,10) # logarithme décimale
D<-log(3,2)# logarithme de base 2
M<-a*b-c
M
```

sortie
```
[1] 15.91353
```

R code
```
round(M,2)
```

sortie
```
[1] 15.91
```

2. Créer une suite arithmétique : premier terme 0, la raison 1, dernier terme 6

R code
```
R<-c(0:6)
R
```

sortie
```
[1] 0 1 2 3 4 5 6
```

3. Créer une suite arithmétique : premier terme 0, la raison 0.2, le dernier terme est 3

R code
```
S<-seq(0,3,0.2)
S
```

sortie
```
 [1] 0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0
```

4. Créer une suite dont on connaît le premier terme , le dernier terme et la longueur (le nombre d'éléments de la suite)

R code
```
suitenum<-seq(1,2,length=10)
suitenum
```

sortie
```
[1] 1.000000 1.111111 1.222222 1.333333 1.444444 1.555556 1.666667 1.777778
[9] 1.888889 2.000000
```

5. Table de multiplication de 6

R code
```
chiffre<-c(1:9)
M6<-chiffre*6
M6
```

sortie
```
[1]  6 12 18 24 30 36 42 48 54
```

6. Trier des nombres donnés

R code
```
X<-c(log(2),exp(-1), sqrt(2),2*cos(pi/5), tan(pi/7))
X<-round(X,3)
sort(X) # ordre croissant
```

sortie
```
[1] 0.368 0.482 0.693 1.414 1.618
```

R code
```
sort(X,decreasing = TRUE)# ordre décroissant
```

sortie
```
[1] 1.618 1.414 0.693 0.482 0.368
```

R code
```

```

7. Répéter une séquence

R code
```
Repeter<-rep('bla',4)
Repeter
```

sortie

```
[1] "bla" "bla" "bla" "bla"
```

8. créer une fonction numérique

R code

```
f<-function(x)
{
 return((x+1)*exp(-x))
}
f(0)
```

sortie

```
[1] 1
```

9. Calculer une intégrale : $\int_0^1 \mathrm{e}^{-x^2}\mathrm{dx}$

R code

```
g<-function(x)
{
 return(exp(-x^2))
}
I<-integrate(g,0,1)$value
round(I,2)
```

sortie

```
[1] 0.75
```

R code

```
P<-integrate(dnorm,-1.96,1.96)$value# densité de la loi normale standard
round(P,4)
```

sortie

```
[1] 0.95
```

10. Concaténer

R code

```
CT<-'Bac'
CT<-paste(CT,'  série scientifique')
CT
```

sortie

```
[1] "Bac   série scientifique"
```

11. Ajouter des éléments à un tableau ou un vecteur existant

R code
```
tab<-c(seq(1,2,0.5),5:7)
tab
```

sortie
```
[1] 1.0 1.5 2.0 5.0 6.0 7.0
```

R code
```
tab<-append(tab,c(10,100,1000,10000))
tab
```

sortie
```
 [1]     1.0     1.5     2.0     5.0     6.0     7.0    10.0   100.0  1000.0
[10] 10000.0
```

12. Tableau d'effectif

Il existe une fonction très utile dans les simulations : la fonction *table()*, cette fonction permet de créer un tableau d'effectifs :

supposons qu'un vecteur X contient les valeurs suivantes :

2 1 5 3 3 4 3 5 3 5 3 1 4 6 4 1 3 6 1 1

Avec table(X) on obtient :

1 2 3 4 5 6

5 1 6 3 3 2

13. Comparer deux vecteurs de même longueur :

Pour chaque indice $i$, on compare $V_1[i]$ à $V_2[i]$, le résultat obtenu est un vecteur logique de même longueur que $V_1$ et $V_2$

R code
```
V_1<-c(2, 1 ,5 ,3 ,3, 4, 3, 5, 3, 5, 3 ,1, 4, 6 ,4 ,1, 3, 6, 1, 1)
V_2<-c(1, 3 ,2 ,4 ,2, 1, 3, 3 ,1, 4, 4, 5, 2, 2, 4, 6, 1, 4, 5, 4)
result<-c(V_1==V_2)
```

On peut éventuellement trier ( avec la fonction sort()) les deux vecteurs avant de les comparer ce qui donne :

R code
```
V_1<-c(2, 1 ,5 ,3 ,3, 4, 3, 5, 3, 5, 3 ,1, 4, 6 ,4 ,1, 3, 6, 1, 1)
V_2<-c(1, 3 ,2 ,4 ,2, 1, 3, 3 ,1, 4, 4, 5, 2, 2, 4, 6, 1, 4, 5, 4
)
result<-c(sort(V_1)==sort(V_2))
```

# 3 Les structures de données

1. Les vecteurs et les tables : Il n'y a pas de nombre isolé sous R, il n'y a que des vecteurs (un nombre isolé est un vecteur de taille 1). Un vecteur est une liste d'éléments simples (numériques, booléens, chaînes de caractères) tous du même type.

R code

```
# définition d'un vecteur
v<-c(2,13,41)
v <- c(c1=2,c2=13,c3=41)# on nomme le élèments d'un vecteur
length(v) # la longueur du vecteur
```

sortie

```
[1] 3
```

R code

```
print(names(v))# afficher les noms des éléments du vecteur
```

sortie

```
[1] "c1" "c2" "c3"
```

R code

```
v[1]# accède à l'élément 1 du vecteur par son indice
```

sortie

```
c1
 2
```

R code

```
v["c1"] # par son nom
```

sortie

```
c1
 2
```

R code

```
1/v # donne un vecteur dont chaque coordonnée est l'inverse
```

sortie

```
        c1         c2         c3
0.50000000 0.07692308 0.02439024
```

R code

```
diff(v) : #vecteur des différences entre un élément et l'élément précédant (de taille length(v) - 1)
v[1:5]# affiche les 5 premiers éléments, si la longueur de v est inférieur à 5, des NA sont affichés
```

sortie

```
[1] 11 10  9  8  7  6  5  4  3  2
```

R code

```
v[v>10]<-0# remplace tous les éléments supérieurs à 10 par 0
# application d'une fonction à chaque élément
sapply(v,FUN=function(x){x^2+1})
```

sortie

```
c1 c2 c3
 5  1  1
```

2. Les matrices : Une matrice est un tableau à deux dimension. Les matrices sont données par colonne

R code

```
matc<-matrix(c(1,2,3,4),nrow=2,ncol=2)# place les valeurs colonne par colonne
matl<-matrix(c(1,2,3,4),nrow=2,ncol=2, byrow=TRUE)# place les valeurs ligne par ligne
x<-as.vector(matc)# transforme la matrice en un vecteur ligne, donne 1 2 3 4
x<-as.vector(matl)# transforme la matrice en un vecteur ligne, donne 1 3 2 4
# on peut nommer les lignes et les colonnes
MATRI <- matrix(c(0,2,3,0,1,4), nrow = 3, dimnames = list(c("L1", "L2", "L3"), c("C1", "C2")))
nrow(MATRI) # pour avoir le nombre de lignes
ncol(MATRI) # pour avoir le nombre de colonnes
diag(MATRI) # renvoie un vecteur correspondant à la diagonale de la matrice MATRI
diag(v) # renvoie une matrice diagonale de, dim =length(v)xlength(v),
#dont les éléments sont ceux du vecteur v.
diag(k)# (k étant un scalaire) : renvoie une matrice identité carrée de dimension k
diag(k,n) # renvoie une matrice diagonale carrée de dimension n et dont les éléments diagonaux sont égaux à k
M<-matc %*% matl # multiplication matricielle
P<-matc*matl # c'est la multiplication terme à terme
solve(matc) : #calcule l'inverse d'une matrice carrée
matcvp<-eigen(matc) # calcule les valeurs propres et les vecteurs propres et renvoie une liste..
```

3. les *data.frame* : dans les sciences expérimentales , les observations notées ou mesurées ou notées se représentent sous forme d'un tableau :
   — Chaque ligne représente un individu
   — Chaque colonne représente une variable les variables peuvent être de différents types (numérique, chaîne de caractères, ... )

   L'hétérogénéité des données ne permet pas de reprsenter ces observations sous forme de matrice. D'où la nécessité d'utiliser les *data.frame*.

R code

```
matri<-sample(100:200,10)
nom<-LETTERS[1:10]
notemath<-round(rnorm(10,10,3))
notephys<-round(rnorm(10,8,3))
moyenne<-(3*notemath+2*notephys)/5
resultat<-moyenne>=10
affichage<-ifelse (resultat==TRUE, resultat<-'Admis',resultat<-'non admis')
examen<-data.frame(matri,nom,notemath,notephys,moyenne,affichage)
```

l'affichage des différents paramètres donne ceci

sortie

```
[1] 181 166 137 113 131 171 196 117 152 169
```

sortie

```
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"
```

sortie

```
[1] 13 10 13 13 11 11  8 11  8 16
```

sortie

```
[1] 11  5  3  9  9 16 13  6 11  7
```

sortie

```
[1] 12.2  8.0  9.0 11.4 10.2 13.0 10.0  9.0  9.2 12.4
```

sortie

```
[1] "Admis"     "non admis" "non admis" "Admis"     "Admis"     "Admis"
[7] "Admis"     "non admis" "non admis" "Admis"
```

sortie

| | matri | nom | notemath | notephys | moyenne | affichage |
|---|---|---|---|---|---|---|
| 1 | 181 | A | 13 | 11 | 12.2 | Admis |
| 2 | 166 | B | 10 | 5 | 8.0 | non admis |
| 3 | 137 | C | 13 | 3 | 9.0 | non admis |
| 4 | 113 | D | 13 | 9 | 11.4 | Admis |
| 5 | 131 | E | 11 | 9 | 10.2 | Admis |
| 6 | 171 | F | 11 | 16 | 13.0 | Admis |
| 7 | 196 | G | 8 | 13 | 10.0 | Admis |
| 8 | 117 | H | 11 | 6 | 9.0 | non admis |
| 9 | 152 | I | 8 | 11 | 9.2 | non admis |
| 10 | 169 | J | 16 | 7 | 12.4 | Admis |

4. les listes : Les listes sont le moyen de construire des structures de données arbitrairement complexes. Une liste est un tableau ordonné d'éléments qui peuvent être hétérogènes. un élément d'une liste peut être un vecteur, une liste, ...

R code

```
L=list(marque= 'R',puissance =c(3:15),energie=c('G','SP','GPL','ELEC'),prix =runif(10,10000,60000))
Maliste<-list(matricule=matri,noms=nom,matrice= matrix(c(0,2,3,0,1,4), nrow = 3, dimnames = list(c("L1", "L2" "L3")
```

# 4 Programmation

La structure de programmation est identique à celle de tous les langages

Toute ligne précédée du symbole # est considérée comme un commantaire

R n'est pas sensible à l'indentation, par contre il est sensible à la casse R<-2 et r<-3 sont deux variables différentes.

1. Si . . .alors . . .sinon . Exemple : maximum de deux nombres

R code
```
a<-12
b<-15
if(a<b)(M<-b) else (M<-a)
print(M)
# on peut créer une fonction et utiliser ifelse
Max<-function(a,b)
{
 ifelse(a<b,M<-b,M<-a)
return(M)
}
# pour appeler la fonction
Max(a=12,b=15)
```

2. La boucle *pour*

R code
```
# somme des 10 premiers entiers naturels
S<-0
for(i in 1:10){S<-S+i}
print(S)
```

3. la boucle *tantque* : le pgcd de deux entiers naturels

R code
```
pgcd<-function(a,b)
{
  c<-a
  d<-b
  # si a est réel il sera transformé en entier par as.integer#
        a<-as.integer(a)
        b<-as.integer(b)
        while(a!=b)  # les deux symboles "!=" signifie non égale
        {
                if (a>b) {a<-a-b}else{b<-b-a}

        }
        res<-paste("le PGCD de ",c,"et",d,"est",a)
        print(res)
}
```

4. La récursivité : suite de Fibonacci

R code
```
fibo<-function(n){
if (n==0)(result<-1)
else{if (n==1)(result<-1) else {result<-fibo(n-1)+fibo(n-2)}}
return(result)
}
# afficher les dix premiers termes de la suite
suite<-NULL
for(i in 1:10)(suite[i]<-(fibo(i)))
suite
```

sortie

```
[1]  1  2  3  5  8 13 21 34 55 89
```

5. Calcul et représentation des premiers éléments de la suite de Syracus

R code

```
n<-14
i<-0
x<-NULL
s<-n
while(s>1)
{
  ifelse(s%%2==0, s<-s/2,s<-3*s+1)
  i<-i+1
  x[i]<-s
}
temps_vol<-length(x)
altitude<-max(x)
abscisse<-which(x==altitude)
temps_vol
```

sortie

```
[1] 17
```

R code

```
altitude
```

sortie

```
[1] 52
```

R code

```
abscisse
```

sortie

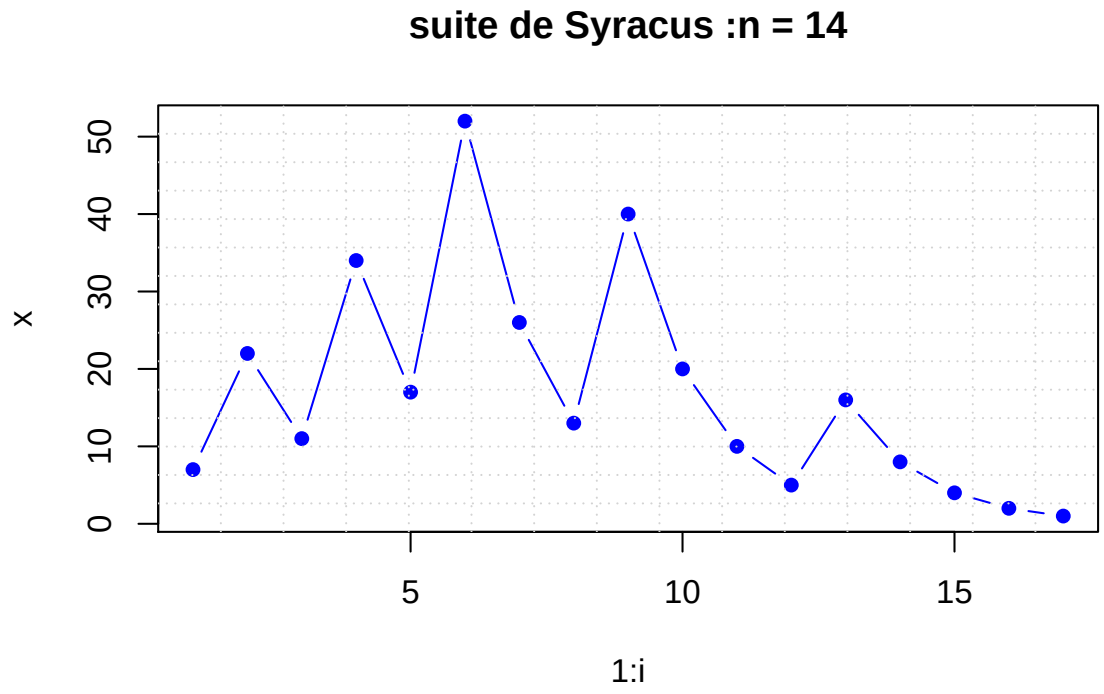
```
[1] 6
```

6. représentation graphique de la suite de Syracus

R code

```
n<-14
i<-0
x<-NULL
s<-n
while(s>1)
  {
  ifelse(s%%2==0, s<-s/2,s<-3*s+1)
  i<-i+1
  x[i]<-s
```

```
  }
temps_vol<-length(x)
altitude<-max(x)
abscisse<-which(x==altitude)
plot(1:i,x,pch=16,type='b',col='blue',main=paste('suite de Syracus :n =',n))
grid(15)# afficher une grille
```




## 5 Statistique à une variable

1. la saisie des données

   Saisir une série statistique à une variable, qui se présente sous forme d'un tableau d'effectifs ou de fréquences :

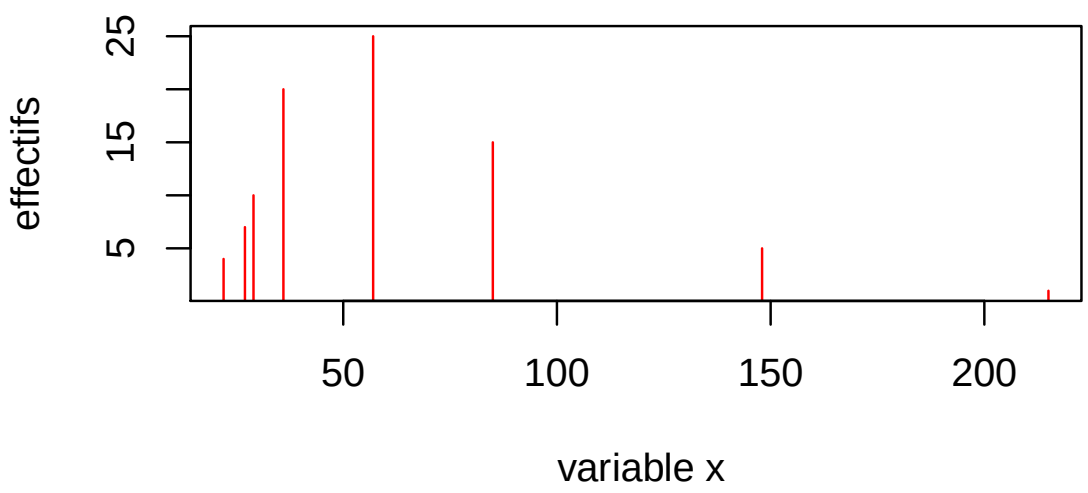
R code
```
x<-c(215,148,85,57,36,29,27,22) # les mesures
f <-c(1,5,15,25,20,10,7,4) # les effectifs
```

2. Diagramme en bâtons

R code
```
plot(x,f,type='h',col='light blue')
```

On peut ajouter un titre ( main), une étiquette sur l'axe des abscisses (xlab), une etiquette sur l'axe des ordonnées, colorier le graphique (col), ...

R code
```
plot(x,f,type='h', main = "diagramme en bâtons", xlab=" variable x", ylab ="effectifs",col="red")
```




**3.** Calcul des paramètres statistique :moyenne, médiane, ...

**Remarque** La fonction *sum(x)* calcule la somme des valeurs prises par le vecteur $x$

On rappalle que $\bar{x} = \dfrac{\sum_{i=1}^{n} n_i \times x_i}{\sum_{i=1}^{n} n_i}$, avec **R** il suffit d'écrire :

R code
```
m<-sum(f*x)/sum(f)
```

De même la variance est :

R code
```
V<-sum(f*(x*x))/sum(f)-m*m
```

**4.** La fonction *summary(X)* affiche les différents paramètres de positions de la série statistique $X$.

R code
```
X<-c(90 ,120,150,180,210,240, 270,300 )
summary(X)
```

sortie
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   90.0   142.5   195.0   195.0   247.5   300.0
```

# 6 Statistique à deux variables

On se limite au cas où il existe deux variables représentées par deux vecteurs sans effectifs. Généralement il existe une variable explicative $x$, $t$, ... et une variable à expliquer $y$, $z$, ... , il faut bien distinguer les deux variables. Étudions l'exemple suivant, en commençant par le nuage de points

1. Nuage de de points

R code
```
x<-c(90 ,120,150,180,210,240, 270,300 )
z<-c(5.77, 5.55, 5.31, 5.10 ,4.89 ,4.65, 4.42, 4.23)
plot(x,z,main="nuage de pts de z en fonction de x")
```

2. Coefficient de corrélation

R code
```
x<-c(90 ,120,150,180,210,240, 270,300 )
z<-c(5.77, 5.55, 5.31, 5.10 ,4.89 ,4.65, 4.42, 4.23)
r<-cor(x,z)
```

3. Droite de regression

C'est le modèle linéaire qui est utilisé par R-Project (*Linear Model ou lm*), basée sur la méthode des moindres carrés.

R code
```
resultat<-lm(z~x)
resultat
```

Nous avons crée une variable *resultat* qui contient tous les résultats réalisés par notre modèle.
On s'aperçois que l'exécution de cette fonction donne les deux coefficient de la droite de regression : une valeur est affichée sous **Intercept** elle correspond à l'ordonnée à l'origine et une valeur sous la lettre $x$ qui correspond au coefficient directeur de la droite de regression

**Remarque** Les équations de droite sont généralement notées : $y = a + bx$, d'où l'origine de la fonction *abline*. Pour tracer, avec R, la droite d'équation $y = 2x + 1 = 1 + 2x$, il suffit de saisir **abline(a=1,b=2)** ou tout simplment **abline(1,2)**.

4. La pente et l'ordonnée à l'origine

On peut isoler les paramètres de la droite de regression de la manière suivante :

R code
```
resultat<-lm(z~x)
ord<- resultat$coefficients[1]
pente<-resultat$coefficients[2]
```

5. Tracé de la droite de regression

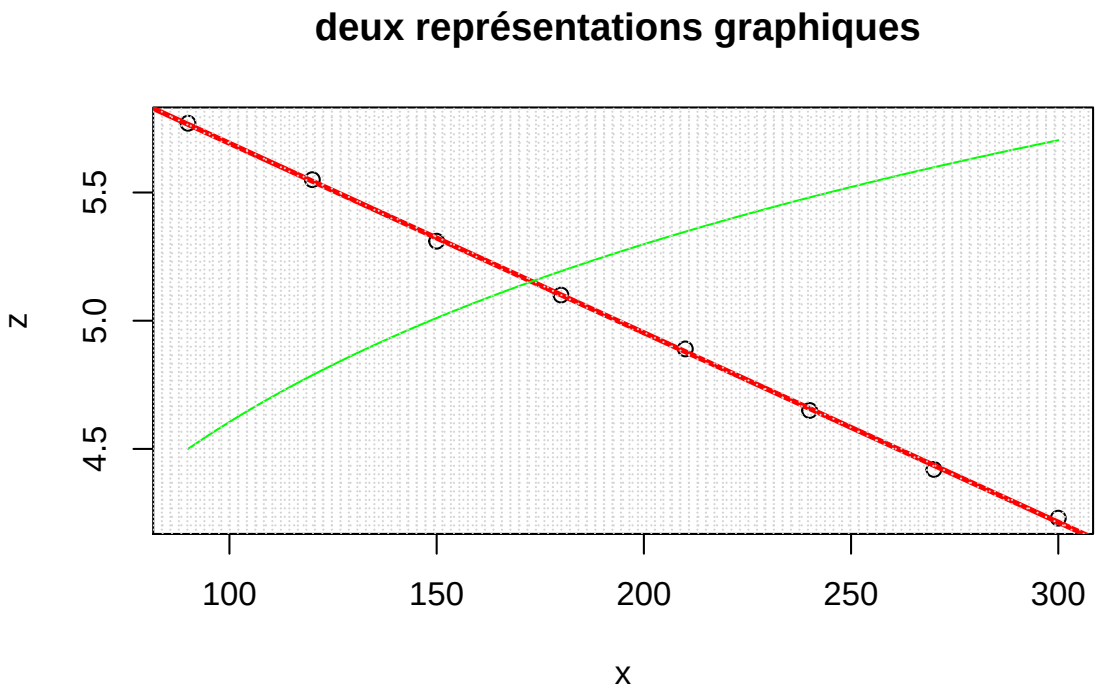
Pour tracer la droite de regression, il faut reprendre le programme de départ et le compléter par abline comme ceci :

R code
```
x<-c(90 ,120,150,180,210,240, 270,300 )
z<-c(5.77, 5.55, 5.31, 5.10 ,4.89 ,4.65, 4.42, 4.23)
plot(x,z,main="nuage de pts de z en fonction de x")
abline(resultat)
grid(20)
```

**6.** Ajout d'une courbe dans un graphique existant ( avec add=TRUE ou add =T)

R code

```
x<-c(90 ,120,150,180,210,240, 270,300 )
z<-c(5.77, 5.55, 5.31, 5.10 ,4.89 ,4.65, 4.42, 4.23)
plot(x,z,main="deux représentations graphiques ")
abline(resultat,col='red',lwd=2.5)
curve(log(x), 90, 300, add=T, col='green') # ajouter la courbe de x--->xexp(-2x) sur [90,300]
grid(100)
```

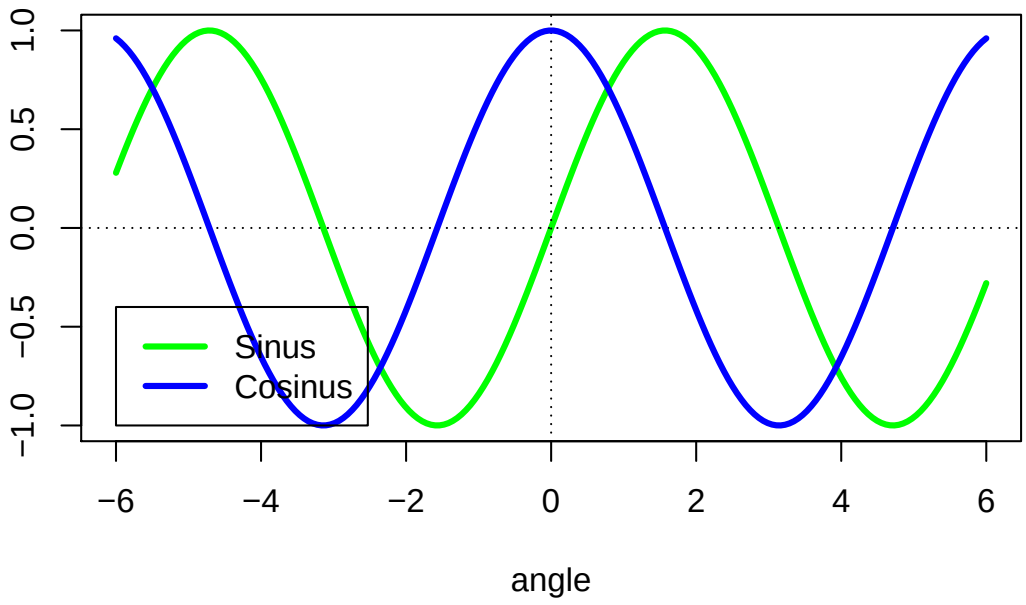



**Remarque** Les réels 90 et 300 sont les bornes de l'intervalle dans lequel la foncion doit être étudiée.

**7.** Ajout d'une legende à un dessin avec la fonction *legend*

R code

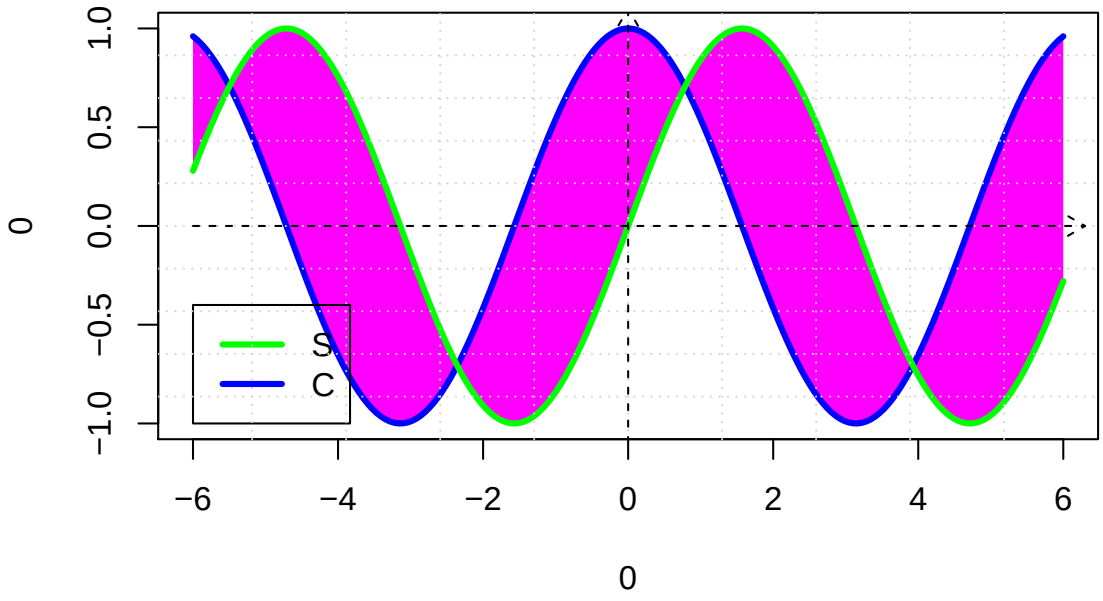
```
x <- seq(-6,6,length=200)
y <- sin(x)
z <- cos(x)
plot(y~x, type='l', lwd=3,
     ylab='', xlab='angle', main="Fonctions trigonométriques",col='green')
abline(h=0,lty=3)
abline(v=0,lty=3)
lines(z~x, type='l', lwd=3, col='blue')
legend(-6,-1, yjust=0,c("Sinus", "Cosinus"),lwd=3, lty=1, col=c('green', 'blue'))
```




**8.** On peut colorier des parties du dessin à l'aide de la commande polygon.

R code

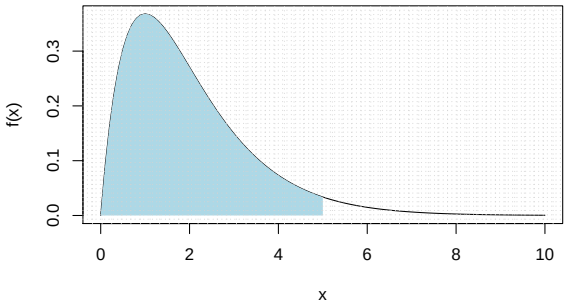
```
ColorDessin <- function (f, g, xmin, xmax, col, N=200) {
  x <- seq(xmin,xmax,length=N)
  fx <- f(x)
  gx <- g(x)
  plot(0,0, type='n',
       xlim=c(xmin,xmax),
       ylim=c( min(fx,gx), max(fx,gx) ) )
  polygon( c(x,rev(x)), c(fx,rev(gx)), col=rgb(1,0,1), border=0 )
  lines(x,fx,lwd=3,col='blue')
  lines(x,gx,lwd=3,col='green')
}
ColorDessin( function(x) cos(x), function(x) sin(x), -6, 6)
legend(-6,-1, yjust=0,c("S", "C"),lwd=3, lty=1, col=c('green', 'blue'))
grid(10)
arrows(-6,0,6.3,0,length = 0.1,lty='dashed')
arrows(0,-6,0,1.1,length = 0.1,lty='dashed')
```

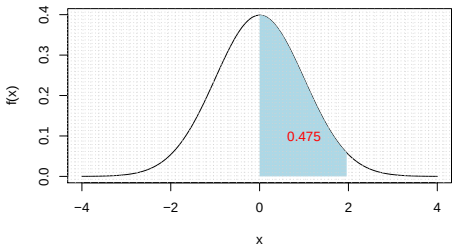



**9.** Hachurer un domaine plan en vue de calculer son aire

R code

```
domaine<- function (f, xmin, xmax,a,b, col, N=200) {
 N<-200
 curve(f(x),xmin,xmax)
 intervalle <-seq(a,b,l=N)
  fx <- f(intervalle)
  gx <- rep(0,N)
  polygon( c(intervalle,rev(intervalle)), c(fx,rev(gx)), col='light blue', border=0 )
  grid(50)
  }
domaine(function(x) x*exp(-x), 0, 10,a=0,b=5)
```




# 7 Lois de probabilité

Les lois de probabilités peuvent étres utilisées de trois manières différentes :

— Faire appel à la densité de probabilité, dans le cas continue cette fonction n'a pas d'utilité pour le calcul des probabilités, dans le cas discret on calculera $P(X = k)$
— Faire appel à la fonction de répartition pour le calcul des probabilités cumulées : $P(X \leq k)$ ou autre.
— Générer des nombres aléatoires d'une distribution connue
Comme dans presque tous les logiciels, **R-Project** utilise les premières initiales pour faire appel aux différentes lois.

Pour appeler la densité de probabilité, on ajoute le préfixe d, pour la fonction de répartition :p, pour la fonction inverse : q ( comme quintile) et pour générer les nombres aléatoire, on ajoute le préfixe r

1. Binomiale paramètres n et p
   — Probabilité ponctuelle : $P(X = k) \longrightarrow$ dbinom(k,n,p)
   — Probabilité cumulée avec le signe $\leq$ : $P(X \leq k) \longrightarrow$ pbinom(k,n,p)
   — Probabilité cumulée avec le signe $\geq$ : $P(X > k) \longrightarrow$ dbinom(k,n,p,FALSE)
   — calcul des fractiles ou quintile : $P(X \leq x) = prob$, l'inconnue est x :qbinom(prob,n,p)
   — Générer un échantillon de taille N, d'une distribution binomiale rbinom(N,n,p)
2. Normale paramètres m et $\sigma$
   — Probabilité cumulée avec le signe $\leq$ : $P(X \leq k) \longrightarrow$ pnorm(k,m,$\sigma$)
   — Probabilité cumulée avec le signe $\geq$ : $P(X \leq k) \longrightarrow$ pnorm(k,m,$\sigma$,FALSE)
   — calcul des fractiles ou quintile : $P(X \leq x) = p$, l'inconnue est x qnorm(p,m, $\sigma$)
   — Générer un échantillon de taille N, d'une distribution normale rnorm(N,m,$\sigma$). par défaut m=0 et $\sigma = 1$
3. Uniforme continue sur [a , b]
   — Probabilité cumulée avec le signe $\leq$ : $P(X \leq k) \longrightarrow$ punif(k,a,b)
   — Probabilité cumulée avec le signe $\geq$ : $P(X \leq k) \longrightarrow$ punif(k,a,b,FALSE)
   — calcul des fractiles ou quintile : $P(X \leq x) = p$, l'inconnue est x qunif(p,m, $\sigma$)
   — Générer un échantillon de taille N, d'une distribution normale runi(N,a,b). par défaut runif(N) avec $a = 0$ et $b = 1$
4. les initiales de la loi de Poisson et de la loi exponentielles sont respectivement pois et exp
5. Uniforme discrète Il n y a pas de loi uniforme discrète, par contre il existe une fonction très utile pour générer des nombres entiers ou effectuer des tirages de n'importe quel type d'objets, c'est la fonction *sample*
   — Lancer de dé sample (1 :6,size=1) pour un seul lancer de dé, ou simplement sample(1 :6,1). Ceci correspond à un tirage d'un objet parmi 1,2,3, ... 6.
   — La fonction sample, qui a pour but de prélever un échantillon, a plusieurs paramètres :
     → le contenu de la population mère comme un dé 1 :6 ou une pièce de monnaie :'P','F'....
     → la taille de l'échantillon
     → replace variable logique qui indique avec ou sans remise
     → prob, c'est un vecteur qui contient les proportions des individus dans la population mère

     ▷ **Exemple 1** □ : *sample(urne<-c('R','B','V'),10,replace=TRUE,prob=c(0.1,0.35,0.55))*, a pour but d'effectuer 10 tirages avec remise dans le vecteur urne composé de 10% de boules rouges, 35% de boules blanches et 55% de boules vertes

▷ **Exemple 2** □

Si $X$ a une distribution binomiale de paramètres $n = 30$ et $p = 0.45$

1. Pour calculer P(X=15) il suffi de saisir dbinom(15,30,0.45), on trouve 0.1242479
2. Par contre, si on veut calculer : $P(10 < X < 19)$, il faut écrire :
   $sum(dbinom(11 : 18, 30, 0.45))$, on trouve $P(10 < X < 19) \simeq 0.83$ ou encore $pbinom(18, 30, 0.45) - pbinom(10, 30, 0.45)$
3. Par défaut pbinom(k,n,p)=$\sum_{j=0}^{k} \binom{n}{j} p^j \times (1-p)^{n-j} = P(X \leq k$, sinon pour calculer $P(X > k)$ on écrit tout simplement : pbinom(k,n,p,lower.tail =FALSE), $k$ est un entier naturel inférieur à $n$
   par exemple $P(X > 10)$ s'obtient de deux manières :
   — 1-pbinom(10,30,0.45)$\simeq$0.86

— pbinom(10,30,45,FALSE)≃0.86

4. si on veut chercher $a$ telle que $P(X \leq a) = prob$
   il suffit de saisir qbinom(prob,n, p)
   Aisni si $X \sim> \mathcal{B}(30\ ,\ 0.45)$, et si on veut déterminer $a$ telle que $P(X \leq a) = 0.025$, on écrit : qbinom(0.025,30,0.45) ce qui donne 8.
   Que vaut qbinom(0.975,30,0.45) ?
5. Si on veut générer 100 nombres aléatoires d'une distribution binomiale de paramètres n=30 et p=0.45
   On écrit :rbinom(100,30,0.45)

▷ **Exercice 1** □

1. Générer 100 entiers d'une distribution binomiale de paramètres n=5 et p=0.35, puis représenter le résultat dans un tableau d'effectif avec la fonction table
2. Générer 50 nombres aléatoires d'une distribution uniforme sur l'intervalle $\left[5\ ,\ 15\right]$
3. $X$ est une variable aléatoire qui suit la loi normale de paramètres $\mu = 175$ et $\sigma = 3$. Déterminer $x$ telle que $P(X > x) = 0.75$
4. **a.** Générer une série de 100 nombres aléatoires d'une distribution normale de moyenne 5 et d'écart type 0.02. Arrondir les résultats à 0,01 près
   **b.** Calculer les paramètres statistiques de la série obtenue.
   **c.** Construire l'histogramme de cette série de données.

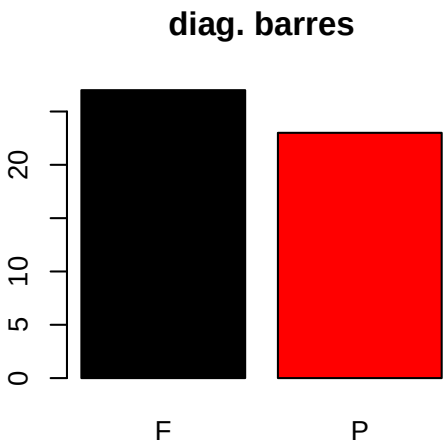
# 8 Les graphiques

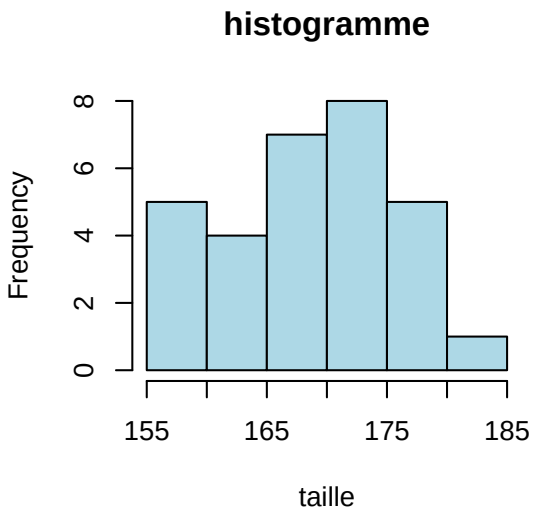
1. diagramme en barre :*barplot*
2. nuage de points ou diagramme en bâtons : *plot* ou *points*
3. histogramme *hist()*
4. diagramme circulaire : *pie ()*
5. courbe d'une fonction : *curve(expression)*

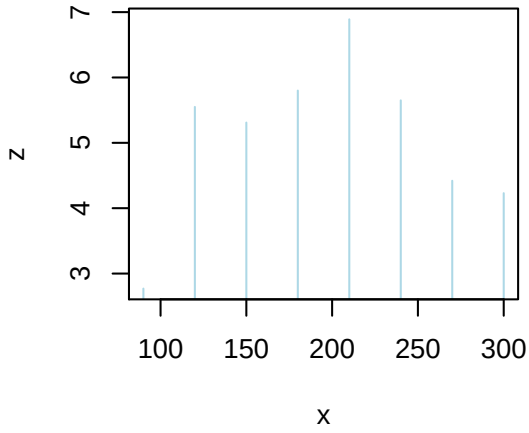
Il est possible de placer plusieurs graphiques dans une seule fenêtre, grâce à l'instruction mfrow ( nbr-ligne,nbr-colonne), si on veut que les graphiques remplissent la fenêtre ligne par ligne ou mfcol(nbr-ligne,nbr-colonne) si le remplissage se fait par colonne

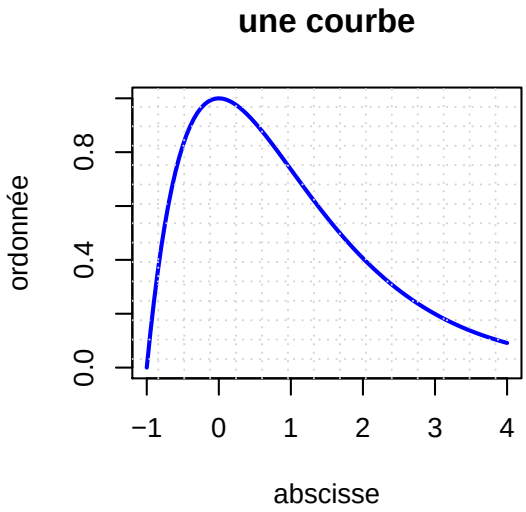
R code

```
par(mfrow=c(2,2)) # quatre graphique dans une seule fenêtre
piece<-c('P','F')
p<-0.35 # probabilité d'avoir pile par exemple
lancerpiece<-sample(piece,50, replace=T, prob=c(p,1-p))
# diagramme en barre
barplot(table(lancerpiece),main="diag. barres",col=c(1,2))
#histogramme
taille<-c(165.97, 179.80, 171.49, 171.60, 167.59, 170.94, 155.69, 164.17 ,183.68,
159.43 ,174.37, 176.27 ,163.29, 173.18, 158.94 ,164.77, 161.80, 158.60,
171.29, 156.62, 171.73, 166.98 ,176.95, 165.89, 169.60, 169.81 ,168.98,
173.60, 177.17, 180.46)
taille<-round(taille,0)# arrondir à l'unité
hist(taille, col="lightblue", main ='histogramme')
# diagramme en batôns
x<-c(90 ,120,150,180,210,240, 270,300 )
z<-c(2.77, 5.55, 5.31, 5.80 ,6.89 ,5.65, 4.42, 4.23)
plot(x,z,type='h',col="lightblue")
# courbe d'une fonction
curve((x+1)*exp(-x),-1,4,lwd='2',col='blue',xlab='abscisse'
      ,ylab='ordonnée',main='une courbe')
grid(15)
```




# 9 les simualtions classiques : dé, urne, pièce de monnaie

1. **Lancer de dé**
   L'équivalent de Alea() sur R-project est runif(), la fonction génératrice de nombres aléatoires dans [0 , 1[ :

R code
```
tirage1<-runif(n=10, min=0,max=1) # tirage de 10 réels compris entre 0 et 1
tirage2<-runif(n=10, min=2,max=5) # tirage de 10 réels compris entre 2 et 5
```

2. Tirage de nombres entiers compris entre une valeur-min et une valeur-max. Voici par exemple une simualtion de 100 lancers d'un dé cubique ( min=1 et max =6)

R code
```
Dé<-c(1:6)# ou Dé<-c(1,2,3,4,5,6)
result<-sample(Dé,size=100, replace=T)
# ou tout simplement
 result<-sample(1:6,100, replace=T)
```

Avec la fonction table on obtient :

R code
```
Dé<-c(1:6)
result<-sample(Dé,size=100, replace=T)
table(result)
```

sortie
```
result
 1  2  3  4  5  6
20 14 15 13 17 21
```

**3.** Lancer de deux dés :somme des numéros des faces obtenues

R code
```
face<-c(1:6)
S<-NULL
for(i in 1:1000){
Dé1<-sample(face,size=1, replace=T)
Dé2<-sample(face,size=1, replace=T)
S[i]<-Dé1+Dé2
                }
table(S)
```

sortie
```
S
  2   3   4   5   6   7   8   9  10  11  12
 30  63  93  95 139 158 133 121  77  57  34
```

**4.** Tirage dans une urne bicolore dont 65% de boules sont rouges (R) et les autres sont vertes (V). Si on veut effectuer 25 tirages avec remise, on procède ainsi :

R code
```
urne<-c('R','V')
Tirage<-sample(urne,25, replace=T, prob=c(0.65,0.35))
```

**5.** Si on souhaite voir la proportion de chaque couleur de notre échantillon

R code
```
urne<-c('R','V')
Tirage<-sample(urne,25, replace=T, prob=c(0.65,0.35))
table(Tirage)
```

sortie
```
Tirage
 R  V
16  9
```

**Remarque** On sait que le tirage avec remise, dans une urne bicolore, d'un échantillon de taille est un schéma de Bernouli. La variable aléatoire $X$ qui comptabilise le nombre de succès ( rouge ou verte) suit la loi binomiale de paramètres n et p, où p est la probabilité de succès.

On suppose que les boules sont codées de la manière suivante : $rouge = 1$ et $verte = 0$ et donc un succès c'est obtenir une boule rouge

Le programme suivant simule ce schéma de Bernoulli

R code

```
urne<-c(1,0)
taille<-5
psuccès<-0.65
resultat<-NULL
nbsimul<-1000# nombre de simulations
for (i in 1:nbsimul )
  {
    Tirage<-sample(urne,taille, replace=T, prob=c(psuccès,1-psuccès))
    resultat[i]<-sum(Tirage)
  }
table(resultat)/nbsimul # renvoie les probabilités attendues
```

sortie

```
resultat
    0     1     2     3     4     5
0.007 0.052 0.194 0.342 0.290 0.115
```

R code

```
mean(resultat)# espérance mathématique attendue de la v.a. X
```

sortie

```
[1] 3.201
```

**6.** Pièce de monnaie On distingue deux cas :

**a.** pièce équilibrée

R code

```
piece<-c('P','F')
lancerpiece<-sample(piece,50, replace=T)
table(lancerpiece)
```

**b.** pièce truquée

R code

```
piece<-c('P','F')
p<-0.35 # probabilité d'avoir pile par exemple
lancerpiece<-sample(piece,50, replace=T, prob=c(p,1-p))
table(lancerpiece)
```

# 10 Théorème de la limite centrée

▷ **Proposition 1** : Soient $X_1$, $X_2$, …, $X_n$, n variables aléatoires indépendantes et indentiquement distribuées de moyenne $m$ et d'écart type $\sigma$.

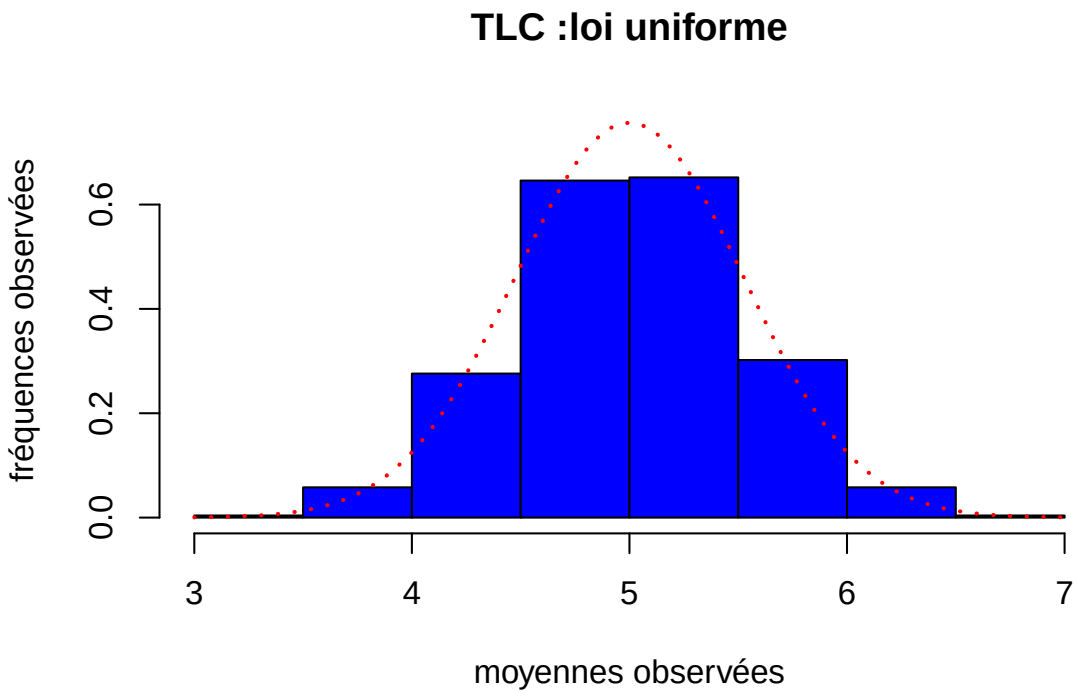
Si $n \geq 30$, la variable aléatoire, moyenne arithmétique, $\bar{X}$, des $X_i$, $\bar{X} = \dfrac{X_1 + X_2 + \cdots + X_n}{n}$ suit une loi normale de paramètres $m$ et d'écart type $\dfrac{\sigma}{\sqrt{n}}$

**Remarque** Si les $X_i$ sont distribuées normalement la condition $n > 30$ n'est pas nécessaire

R code

```
a<-0
b<-10
m=(a+b)/2
sigma<-(b-a)/sqrt(12)
Nbsimul<-1000
taille<-30
Result<-NULL
for(i in 1:Nbsimul)

{
        tirage<-runif(taille,0,10)
        Result[i]<-mean(tirage)
}
hist(Result,freq=FALSE,col='blue',
     ylim=c(0,dnorm(m,m,sigma/sqrt(taille))),
     main = "TLC :loi uniforme",xlab="moyennes observées",ylab="fréquences observées")
curve(dnorm(x,m,sigma/sqrt(taille)),add=T,col='red',lty=3,lwd=2)
```




▹ **Exercice 2** □ Simuler le théorème de la limite centrée avec une suite de variable aléatoire binomiale ou autre de votre choix.

▹ **Exercice 3** □

**les canards et les chasseurs :énoncé**

*dix chasseurs, tous tireurs d'élite, sont à l'affût aux canards devant un rocher. Dix canards se posent sur le rocher. Les chasseurs ne peuvent tirer qu'une seule fois et ne peuvent repérer qui tire sur tel canard. Ils tirent tous en même temps, chacun choisissant sa victime au hasard*

*Combien de canards surviveront en moyenne si l'on répète souvent cette expérience ?*

**Éléments de correction**

1. Calcul de la valeur moyenne
   Considérons un canard quelconque, le numéro 3 par exemple. Il survivra si chacun des dix chasseurs choisit un canard autre que lui. La probabilité de cet évènement est $p = \left(\frac{9}{10}\right)^{10}$. Chacun des dix canards sera sauf avec la même probabilité $p$. D'après l'interprétation en fréquence relative de la probabilité, il y aura en moyenne $10 \times 0,9^{10} \simeq 3,5$
2. Simulation avec un programme R.

Principe :

On suppose que les dix canards portent les numéros 0, 1, . . ., 9.Pour cela on crée un tableau appelé $num-canard$. On tire un échantillon aléatoire et **avec remise** de taille 10. Le résultat est rangé dans un tableau appelé $canard-vise$. A chaque simulation on compare le tableau obtenu avec le tableau initiale $num_canard$, ce qui nous donne le nombre de canards survivants.

R code

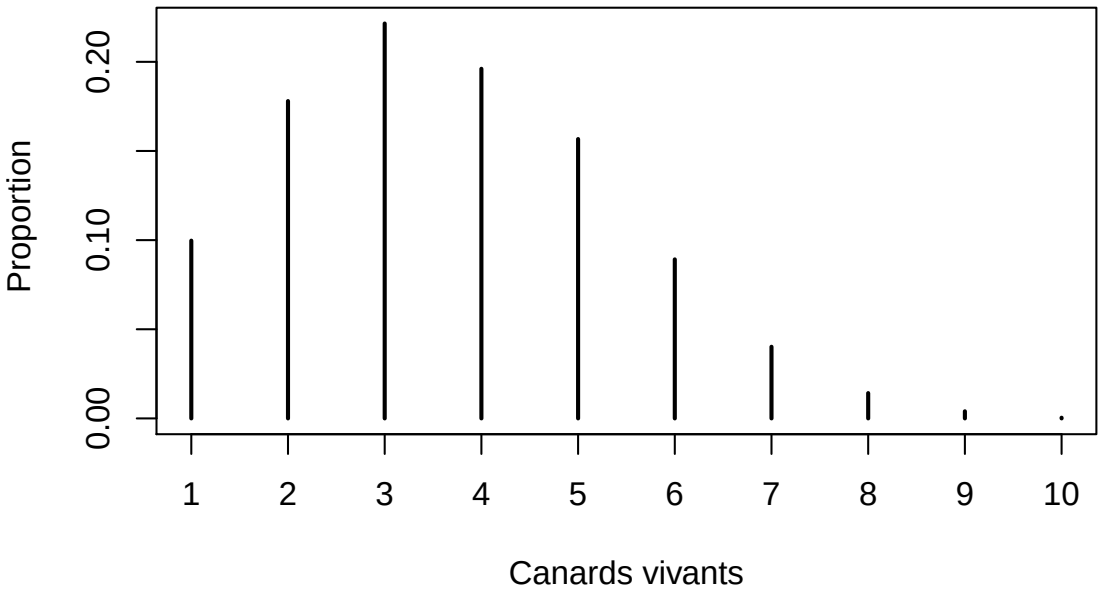
```
nb_canard_vivant<-NULL
num_canard<-c(0:9) # numéros des canards
nbre_simul<-10000 # nombre de simulions
for (i in 1:nbre_simul)
{
canard_vise<-sample(num_canard,10,replace= TRUE) # tirage de 10 entiers compris entre 0 et 9
nb_canard_vivant[i]<-sum(num_canard==sort(canard_vise))
}
table(nb_canard_vivant) # tableau d'effectifs
plot(table(nb_canard_vivant)/nbre_simul, xlab="canards vivants", ylab="la proportion")
mean(nb_canard_vivant)# la moyenne observée
```

sortie

```
nb_canard_vivant
   1    2    3    4    5    6    7    8    9   10
 997 1780 2215 1961 1567  892  402  142   40    4
```

sortie

```
[1] "la moyenne observée est: 3.6583"
```


Proportion
0.00
0.10
0.20
1
2
3
4
5
6
7
8
9
10
Canards vivants

# 11 La Méthode de Monté Carlo :MMC

La méthode de Monte-Carlo est une méthode probabiliste permettant le calcul approché d'intégrales (simples ou multiples) de fonctions quelle que soit leur régularité. C'est cette propriété qui explique son intérêt par rapport aux méthodes déterministes classiques.

Pour simplifier cette présentation, on suppose que l'on cherche à calculer

$$p = \int_0^1 f(t)dt$$

pour une fonction continue sur [0,1] à valeurs dans [0,1]. Il existe deux méthodes

**1.** Méthode dite du 'rejet'

Rappelons que p est l'aire du domaine

$$\mathcal{D} = \{(x, y) \in [0, 1]^2 | y \leq f(x)\}$$

.

Une première méthode possible est de tirer aléatoirement un grand nombre de points du carré $[0, 1]^2$ et de faire le quotient entre le nombre de points situés dans le domaine $\mathcal{D}$ et le nombre total de points.

Calcul de $\int_0^1 \mathrm{e}^{-x^2} dx$ par la méthode du rejet

R code

```
nbsimul<-1000; compteur<-0
for (i in 1:nbsimul)
{
x<-runif(1); y<-runif(1)
if (y<exp(-x^2))(compteur<-sum(compteur,1))
}
aire<-compteur/nbsimul
aire
```

sortie

```
[1] 0.754
```

**2.** Méthode d'espérance

On admet le résultat suivant :
Si $X$ est une variable aléatoire suivant une loi uniforme sur [0,1] et si $f$ est une fonction continue sur [0,1] alors la variable aléatoire $Y = f(X)$ possède une espérance égale à $p = \int_0^1 f(t)dt$

Calcul de $\int_0^1 \mathrm{e}^{-x^2} dx$ par la méthode d'espérance

R code

```
nbsimul<-1000; y<-NULL
for (i in 1:nbsimul){x<-runif(1);y[i]<-exp(-x^2)}
aire<-sum(y)/nbsimul
aire
```

sortie

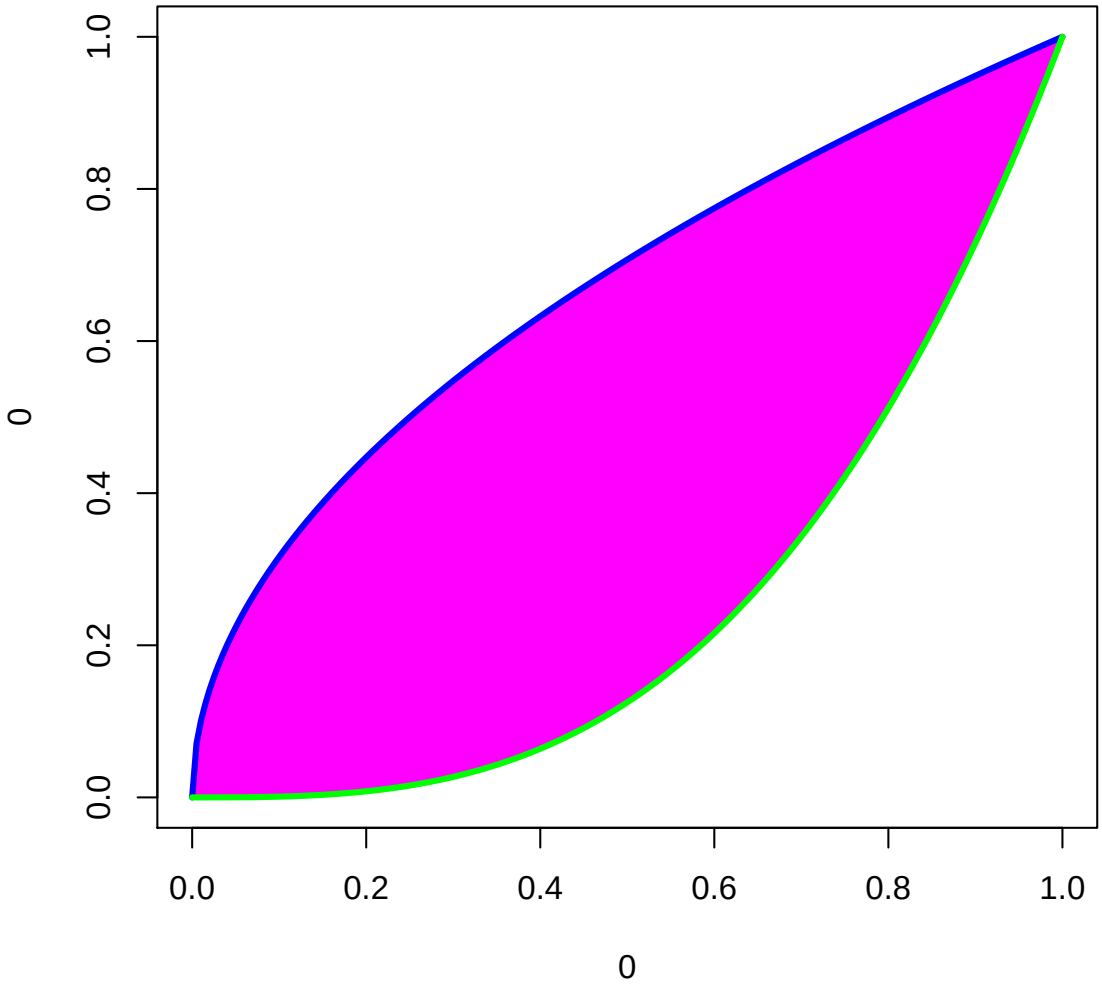
```
[1] 0.7567415
```

**Exercice**

Déterminer, par la méthode du Monté Carlo, l'aire de la figure ci-dessous sous forme d'une feuille, c'est à dire, l'aire du domaine

$$\mathcal{D} = \{(x, y) \in [0, 1]^2 | x^3 \leq y \leq \sqrt{x}\}$$

On utilisera les deux fonctions $x \mapsto x^3$ et $x \mapsto \sqrt{x}$



**1.** Programme

**a.** Méthode dite du 'rejet'

R code
```
nbsimul<-1000; compteur<-0
for (i in 1:nbsimul)
{
x<-runif(1); y<-runif(1)
if (y<sqrt(x)&&y>x^3)(compteur<-sum(compteur,1))
}
aire<-compteur/nbsimul
aire
```

sortie
```
[1] 0.413
```

R code
```
b_inf<-aire-1.96*sqrt(aire*(1-aire))/sqrt(nbsimul)
b_sup<-aire+1.96*sqrt(aire*(1-aire))/sqrt(nbsimul)
print ('intervalle de confiance à 95%')
```

sortie
```
[1] "intervalle de confiance à 95%"
```

R code
```
cat(paste('[',round(b_inf,3),',',round(b_sup,3),']'))
```

sortie
```
[ 0.382 , 0.444 ]
```

**b.** Méthode d'espérance

R code
```
nbsimul<-1000; y<-NULL
for (i in 1:nbsimul){x<-runif(1);y[i]<-sqrt(x)-x^3}
aire<-sum(y)/nbsimul
aire
```

sortie
```
[1] 0.4210367
```

R code
```
b_inf<-aire-1.96*sqrt(aire*(1-aire))/sqrt(nbsimul)
b_sup<-aire+1.96*sqrt(aire*(1-aire))/sqrt(nbsimul)
print ('intervalle de confiance à 95%')
```

sortie
```
[1] "intervalle de confiance à 95%"
```

R code
```
cat(paste('[',round(b_inf,3),',',round(b_sup,3),']'))
```

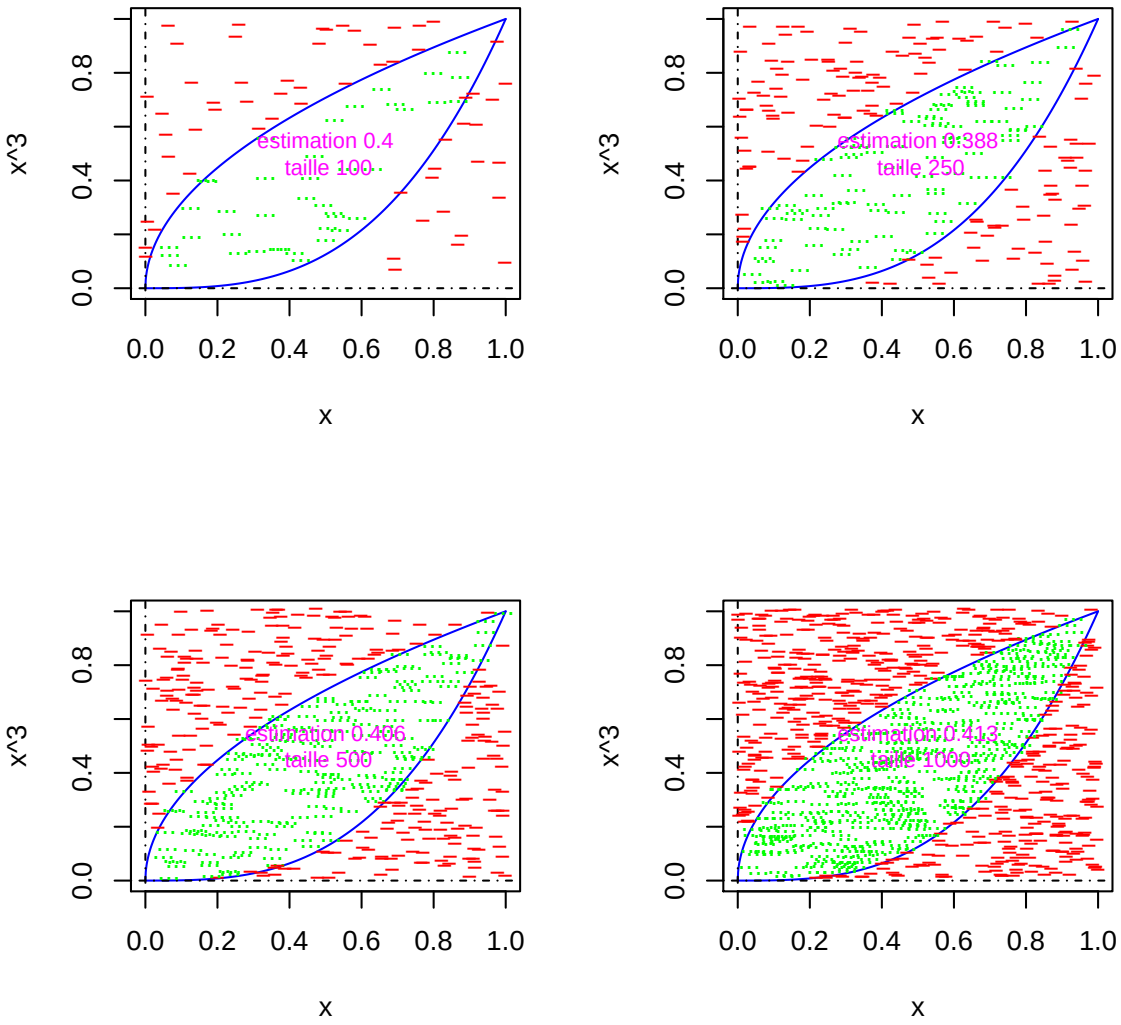
sortie
```
[ 0.39 , 0.452 ]
```

On peut proposer une solution plus élaborée qui permet de visualiser les résultats en fonction du nombre de points simulés, comme ceci :

R code

```
par(mfrow=c(2,2))# quatre figure dans une fenêtre
#------------------------------- Fonction Monté Carlo-------------------------------#
MonteCarlo<-function(nbsimul=100){
curve(x^3, 0,1,n=1000,col='blue')#  bordure basse de la feuille
curve(sqrt(x),0,1,1000,col='blue',add=T)# bordure haute
abline(h=0,v=0,lty=4)# les axes du repère
compteur<-NULL
 for (i in 1:nbsimul){
   x<-runif(1)# tirage d'un nombre aléatoire entre 0 et 1
        y<-runif(1)
        if (y<=sqrt(x) && y>=x^3){
                compteur<-sum(compteur,1)
                 points(x,y,col='green',pch='•')# place dans le repère le point (x,y)
        }
        else (points(x,y,col='red',pch='-'))
         }
aire<-compteur/nbsimul# estimation fréquentiste de l'aire
text(x=0.5,y=0.5,paste(" estimation" ,round(aire,4),"\n taille",nbsimul),col=rgb(1,0,1),font=15,cex=0.8)
  }
#---------------------------------- fin de la fonction----------------------------------------#

#---------programme principal -------------#
 for( n in c(100,250,500,1000))(MonteCarlo(n))
```




# 12 SIMULATIONS, ALGORITHMES EN PROBABILITÉS ET STATISTIQUE(S) AU LYCÉE ET AVEC R

Un excellent document sur le site de l'APMEP tout est dans le titre

http ://www.apmep.fr/IMG/pdf/P1_34_Simulations_Algorithmes_R-2.pdf

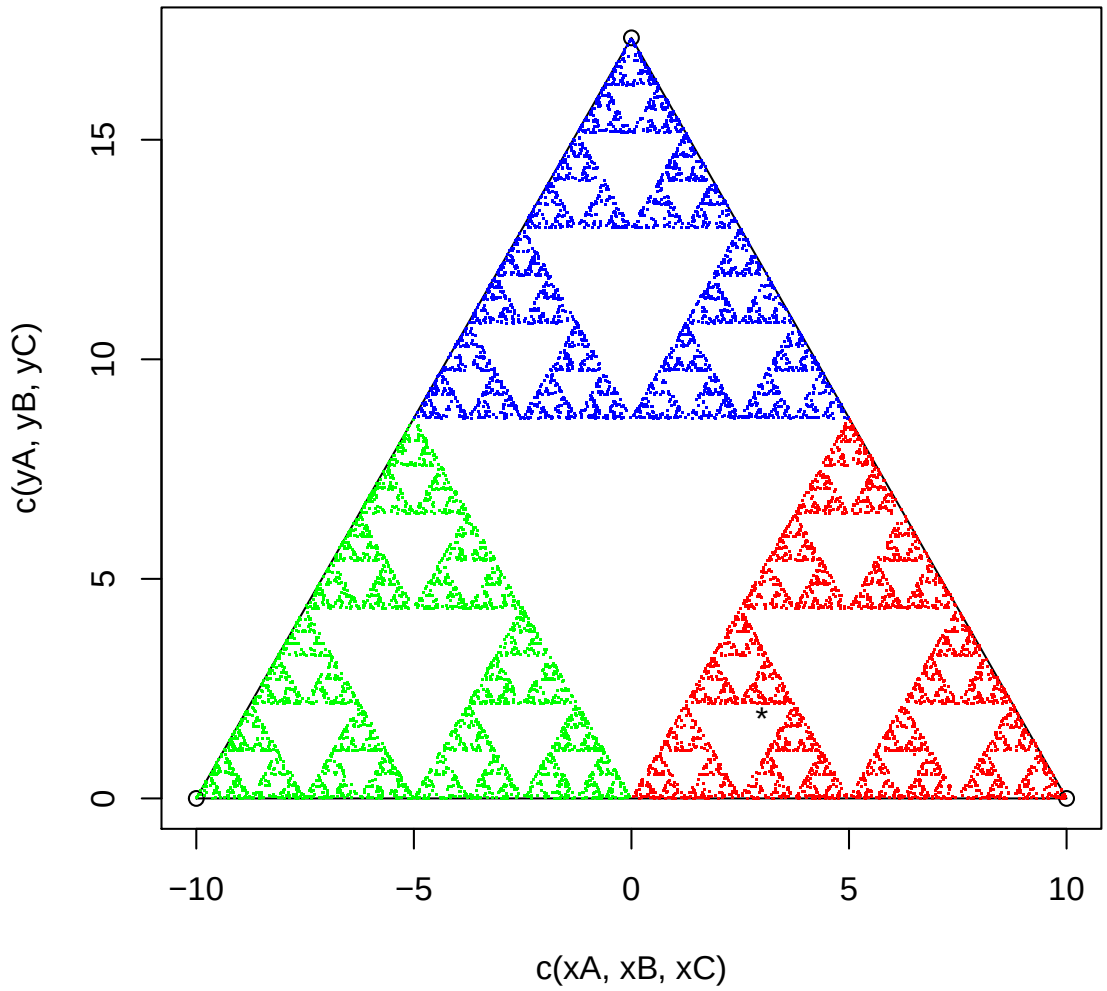
J'ai copié un exemple de ce document sur le triangle de SIERPINSKY

Tracer le triangle $A(-10;0)$, $B(10;0)$, $C(0;10\sqrt{3})$. Placer un point $M0(x;y)$ quelconque dans ce triangle. Pour placer les points $Mi$ suivants : Tirer un nombre entier a au hasard entre 1 et 3. Si $a = 1$ le point $Mi$ sera le milieu du segment $AM_{i-1}$, si $a = 2$ le point $Mi$ sera le milieu du segment $BM_{i-1}$, si $a = 3$ le point $Mi$ sera le milieu du segment $CM_{i-1}$, et ainsi de suite à l'infini ...

R code

```
triangleG2 <- function(nbsim = 10000, xM = 3, yM = 2){
  xA <- -10 ; yA <- 0 ; xB <- 10 ; yB <- 0 ; xC <- 0 ; yC <- 10 * sqrt(3)
```

```
  plot(c(xA, xB, xC), c(yA, yB, yC))
  lines(c(xA, xB, xC, xA), c(yA, yB, yC, yA))
  points(xM, yM, pch = "*") ; x <- xM ; y <- yM
  for(i in 1:nbsim){
    a <- sample(1:3, 1)
      if(a == 1){
         x <- (xA + x) / 2 ; y <- (yA + y) / 2
       points(x, y, pch = ".", col = "green")
              } else {
       if(a == 2){
           x <- (xB + x) / 2 ; y <- (yB + y) / 2
          points(x, y, pch = ".", col = "red")
                 } else {
           x <- (xC + x) / 2 ; y <- (yC + y) / 2
           points(x, y, pch = ".", col = "blue")
                     }
                     }
                }
 }
triangleG2()
```



# 13 Quelques Références, en français

— la meilleur référence est la documentation de R, très riche en exemples, obtenue avec la fonction help()
— Un site de LyonI pour une pensée à mon meilleur professeur de biométrieDaniel Chessel
*http : //pbil.univ − lyon1.fr/R/enseignement_div.php?contents = html/tdr1*
— des livres en français

1. Traitement statistique et programmation avec R Hunault G. Dunod 2017
2. Initiation à la statistique avec R, Bertrand F, Maumy-Bertrand M. Dunod 2010
3. Statistique avec R, Cornillon, presse universitaire de Rennes
4. Le logiciel R, Lafaye de Micheaux P., Drouilhet R, Liquet B, Springer 2011
5. Le livre de R, Desgraupes B., Vuibert 2013
6. R l'essentiel, Adler J. Pearson 2011