ASI (L2) : TP4 Programmation sous *Rstat*

Attention *RSTAT* 2.0 est disponible via Rgui

(on le vérfie avec `factorial(5)` ; montrer `Edit/GUI Preferences`.)

Objectifs du TP :

Savoir programmer des fonctions et des boucles sous *Rstat*.

1. Fonctions rapides

Rappeler comment on écrit une fonction sous *Rstat*. Faire des OEP ("one-expression programs") c'est à dire des programmes en une seule expression comme pour

- $carre(x)$ renvoie le carré de l'argument,
- $sdpe(n)$ renvoie la somme des n premiers entiers,
- $duree(j, m, a)$ secondes écoulées entre an 0 et jour j mois m an a,
- $decrit(v)$ renvoie longueur, moyenne et écart-type de v,
- $binom1(n, p)$ affiche les valeurs de $\mathcal{B}(n, p)$
- $poiss1(m, k)$ affiche les k premières valeurs de $\mathcal{P}(m)$

Tester ce que fait *carre* pour un nombre, un vecteur, une matrice. Afficher avec $binom1$ et $poiss1$ juste le vecteur des valeurs de la loi.

Utiliser $n = 5$, $p = 0.3$ et $\lambda = 1.5$ pour $k = 10$ termes.

Ces fonctions ne font aucun test, aucune boucle. Elles se contentent d'appeler des fonctions de *Rstat*.

1.1 Fonctions soignées

Pour ce qui suit, on fera utiliser *Notepad* pour écrire les fonctions et les charger par `source()` ou via le menu "File/Source". On utilisera `cat` pour afficher avec `\n` pour sauter une ligne.

Reprendre *binom*1 et *poiss*1 pour en faire *binom*2 et *poiss*2 qui donnent un titre, qui affichent un beau tableau des valeurs x_i, des valeurs p_i, du cumul c_i et qui pour un troisième paramètre (facultatif) nommé `et` comme "Effectif Total" donnent en entier les effectifs associés à chaque x_i. On se servira de la page *Web*

`http://www.info.univ-angers.fr/pub/gh/vitrine/Democgi/loisStatp.htm`

comme modèle. Ces fonctions ne font que définir une matrice et la remplir via les bonnes fonctions (aucun test, aucune boucle). Prendre `et` = 150.

Rappeler ce qu'est un test en `si` en algorithmique et donner la traduction en *Rstat*. Reprendre *binom*2 pour en faire *binom*3 qui refuse de calculer si n n'est pas un entier, si p n'est pas un réel compris entre 0 et 1 sachant que *Rstat* dispose des fonctions *as.integer*, *is.numeric* que `non` se note `!` et que `et` se note `&`.

Rappeler ce qu'est une boucle `pour` en algorithmique et donner la traduction en *Rstat*. Ecrire un programme *imax* qui donne le maximum et son indice pour un vecteur passé en paramètre. Insister sur les variables locales, donner un algorithme rapide. Tester avec `c(1,2,8,3,8,5)`.

Rappeler ce qu'est une boucle `tant que` en algorithmique et donner la traduction en *Rstat*. Ecrire un programme *asprite* qui donne afiche les valeurs de n et α_n tant que $|\alpha_n - 1/3| > 10^{-5}$ où α_n correspond à la position du sprite.

On redonne

$$\alpha_n = \frac{1}{3} + \frac{2}{3}\left(\frac{-1}{2}\right)^n$$

2. Esquisse de solution en *Rstat*

```
# la fonction carré

  carre <- function(x) { x*x }

# affichage

  carre

# utilisation

  carre(3)

  carre( sqrt(123456789) )

  carre(0:10)

m <- matrix(nrow=3,ncol=2,1:6)
m

  carre(m)

# la somme des n premiers entiers

  sdpe <- function(n) { sum(0:n) }

# affichage

  sdpe

# utilisation

  sdpe(10)

# une autre solution  si on connait la formule
# ce qui n'est sans doute pas le cas pour la somme
# des n premiers carrés, des n premiers cubes etc;

  sdpe2 <- function(n) { n*(n+1)/2.0 }
```

```
# secondes écoulées

  jpm   <- c( 0 , 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)

  duree <- function(j,m,a) {
    ( (365*a) + sum(jpm[1:(m-1)] + j-1 ) ) * 24 * 60 * 60 }

  duree(1,1,2005)

  24 * 60 * 60

  duree(2,1,2005)

# description statistique

  decrit <- function(x) { c( length(x), mean(x), sd(x) ) }

  decrit(1:10) # de quelle loi s'agit-il ? UD(10) bien sûr !!

# remarque : sd n'est pas "pile poile" notre formule d'écart-type
# la raison en sera donnée plus tard

# loi binomiale version simple

  binom1 <- function(n,p) { dbinom(0:n,n,p) }
  binom1(5,0.3)

# plus lisible

  binom1 <- function(n,p) { dbinom(0:n,size=n,prob=p) }
  binom1(5,0.3)

# loi de poisson version simple

  poiss1 <- function(m,k) { dpois(0:(k-1),lambda=m) }
  # attention 0:k-1 ne fait pas 0:(k-1) mais (0:k) - 1
  poiss1(1.5,10)
```

```
# une binomiale plus jolie

  binom2 <- function(n,p,et=100) {

    cat("\n Voici les valeurs de la loi binomiale
      B(",n,",",p,") et = ",et," \n\n" )

   mdr     <- matrix( nrow=n+1, ncol=4) # matrice des résultats
   mdr[,1] <- 0:n
   mdr[,2] <- binom1(n,p)
   mdr[,3] <- pbinom(0:n,n,p)
   mdr[,4] <- round(mdr[,2]*et)

   colnames(mdr) <- c("xi", "pi" , "ci" ,"thi" )
   rownames(mdr)      <- rep("",n+1)
   mdr

  } ; # fin de fonction binom2

  binom2(5,0.3,150)

# utilisation de la valeur par défaut

  binom2(5,0.3)

# un beau poisson
#   sous éditeur, on fait du copier coller
#   puis du remplacer binom par pois


poiss2 <- function(m,k,et=100) {

 cat("\n Voici les valeurs de la loi de poisson P(",m,")\n" )
 cat(" limitées aux ",k," premiers termes ; effectif total = ",et," \n\n" )

 mdr     <- matrix( nrow=k, ncol=4) # matrice des résultats
 mdr[,1] <- 0:(k-1)
 mdr[,2] <- poiss1(m,k)
 mdr[,3] <- ppois(0:(k-1),m)
 mdr[,4] <- round(mdr[,2]*et)
```

```
 colnames(mdr) <- c("xi", "pi" , "ci" ,"thi" )
 rownames(mdr)      <- rep("",k)
 mdr

} ; # fin de fonction poiss2

# utilisation de la valeur par défaut

poiss2(1.5,10)

# test avec si

binom3 <- function(n,p,et=100) {

 # testons si n est entier

 if (!(n==as.integer(n))) {
    cat(" désolé n devrait être entier, et c'est ",n,"\n")
    return("") ;
 } ;

 # testons si p est numérique enter 0 et 1

 if (!is.numeric(p)) {
    cat(" désolé p devrait être un nombre, et c'est ",p,"\n")
    return("") ;
 } ;

 # testons si p est entre 0 et 1

 if (!( (p>=0) & (p<=1) )) {
    cat(" désolé p devrait être entre 0 et 1 et c'est",p,"\n")
    return("") ;
 } ;

 # arrivé ici on peut appeler binom2

 binom2(n,p,et)
} ; # fin de fonction binom3
```

```
# quelques essais

binom3(5.1,0.3,150)
binom3(5,"oui",150)
binom3(5,1.3,150)
binom3(5,0.3,150)

# boucle pour

imax <- function(v) {

  # initialisations

  lemax <- v[1]
  sapos <- 1

  # boucle de parcours

  for (indi in 2:length(v)) {
   valc <- v[indi] # valeur courante
   if (valc>lemax) { # mise à jour
      lemax <- v[indi]
      sapos <- indi
   } ; # fin si
  } ; # fin pour indi

  # affichage

  cat(" pour le vecteur \n",v,"\n")
  cat(" le maximum est ",lemax,"vu en position ",sapos,"\n")

} ; # fin de fonction imax

imax( c(1,2,8,3,8,5) )

# remarque 1 : c'est la première position du max
# si on avait mis if (valc>=lemax)
# c'aurait été la dernière
```

```
# remarque 2 : pas besoin de boucle pour car

jmax <- function(v) {

  lemax  <- max( v )
  sespos <- (1:length(v))[ v == lemax ]
  cat(" pour le vecteur \n",v,"\n le maximum est ",lemax)
  cat(" on le voit ",length(sespos)," fois en position ",sespos,"\n")

} ; # fin de fonction jmax

jmax( c(1,2,8,3,8,5) )

# définition de alpha n

alphan <- function(n) { (1/3) + (2/3)*(-1/2)^n }

# quelques essais

alphan(0)
alphan(1)
alphan(0:20)

# faisons un peu plus général avec eps au lieu de 10^-5

asprite <- function(eps) {

  cat(" affichage des alphan jussqu'à une différence de ",eps," avec 1/3\n")
  cat(" n alphan différence avec 1/3\n")
  n <- 0
  while( abs(1/3-alphan(n))>eps ) {
    cat( n, " ",alphan(n) , " ",alphan(n)-1/3,"\n")
    n <- n + 1
  } ; # fin de tant que
  cat("\ la preuve : \n")
  cat( n, " ",alphan(n) , " ",alphan(n)-1/3,"\n")

  cat("\n et voilà !\n")

} ; # fin de fonction asprite
```

```
# utilisation

asprite( 10^(-5) )

# un meilleur affichage

asprite <- function(eps) {

  cat("\n affichage des alphan jussqu'à une différence de ",eps," avec 1/3\n")
  cat("\n n alphan différence avec 1/3\n")
  n <- 0
  while( abs(1/3-alphan(n))>eps ) {
    f_n <- sprintf("%3.0f",n)
    f_a <- sprintf("%12.9f",alphan(n))
    f_d <- sprintf("%12.9f",alphan(n)-1/3)
    cat(f_n, f_a, f_d,"\n")
    n <- n + 1
  } ; # fin de tant que

  cat("\n et voilà !\n")

} ; # fin de fonction asprite

# utilisation

asprite( 10^(-5) )
```