QUELQUES EXEMPLES COMMENTÉS POUR DÉMARRER AVEC R
PROBABILITÉS, SIMULATIONS ET EXPLORATION DES SÉRIES SIMULÉES

I - INSTALLATION – MISE EN ROUTE

1° Installation

● **R** est un logiciel libre et gratuit téléchargeable à **"http://cran.univ-lyon1.fr/"**, (site miroir) le site parent étant **"http://www.r-project.org/"**. Il est multiplateforme, c'est à dire qu'il existe des versions qui tournent sous **linux, mac et windows**.

● Il existe quelques ouvrages et un grand nombre de sites en français, d'IUT, d'universités, d'organismes de recherche et d'écoles d'ingénieurs, traitant de l'utilisation de **R**. J'en indiquerai certains en bibliographie.

● L'installation se fait très rapidement et simplement à partir du fichier exécutable téléchargé. Ce "package" de base est complet et permet d'effectuer tous les traitements statistiques courants (description, analyse exploratoire des données, probabilités, simulation, tests statistiques).

● L'utilisation de **R** peut se faire en ligne de commande, l'installation de base y suffit. On peut aussi utiliser certaines fonctionnalités de **R** sous forme classique de menus cliquables en français, il faut alors installer le package **"Rcmdr"**. Les commandes **R** correspondant à chaque menu sont affichées, ce qui facilite une première prise en main. La rédaction et la lecture des lignes de commande **R** sont grandement facilitées par l'utilisation d'un éditeur spécifique, **"Tinn-R"** (téléchargeable à **"http://sourceforge.net/projects/tinn-r/"**) qui identifie toutes les commandes **R** et leurs paramètres et les colorie de façon différenciée pour en faciliter l'identification et l'utilisation.

● Dans les fichiers mis à disposition, figurent deux "Reference card" (**"Rrefcard2.pdf" et "ShortRefCard.pdf"**) qui contiennent les principales commandes **R** classées par thème.

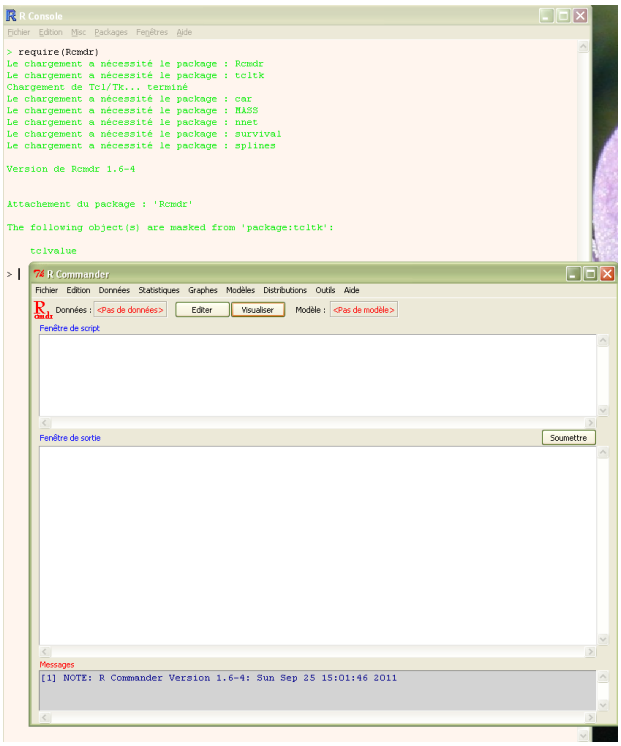
● La communauté des utilisateurs de **R** développent, pour les besoins des structures dans lesquelles ils travaillent ou des recherches engagées, des "packages" "agréés" par une "R-core-team", qu'ils mettent à disposition sur les sites et qui peuvent s'installer automatiquement. Il en existe plusieurs centaines. J'en mentionne deux dans certains fichiers annexés au document d'accompagnement, qui sont **"lattice"** (graphismes avancés) et **"Hmisc"** (présentation avancée de résumés numériques). De même il existe un package (et un ouvrage en français) dédiés à l'analyse des données à la française, **"FactoMineR"**, développé par trois enseignants chercheurs de l'AgroCampus de Rennes.

2° Utilisation de l'interface avec menus à cliquer en français : "Rcmdr"

● On peut utiliser **R** en mode **menus à cliquer** en français. Le code de chaque commande sollicitée par les menus apparaît dans la "Fenêtre de script" est exécutée dans la "Fenêtre de sortie", qui affiche aussi les résultats. Cet affichage des commandes correspondant aux menus cliqués permet l'apprentissage progressif des codes des commandes **R**. C'est aussi un bon outil pour initier les élèves.

Il faut pour cela installer le package **Rcmdr :** Après avoir lancé **R** (RGui) et être connecté à internet, c'est automatique via le menu "Package – Installer le package"). Pour l'exécuter il faut cliquer le menu Package – Charger le package" ou taper **require(Rcmdr)** dans la console **R**.

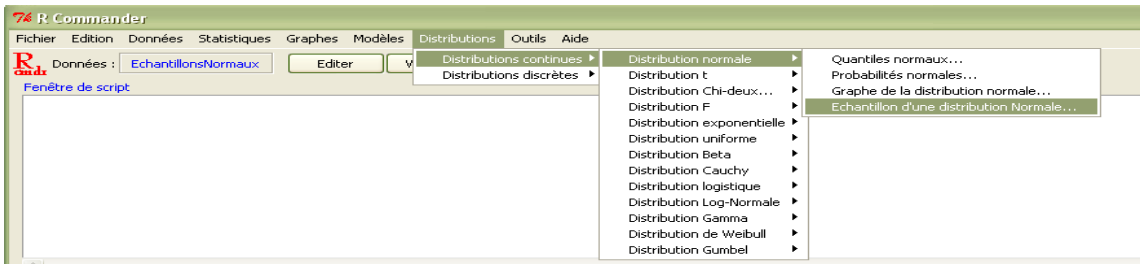
Voici un exemple simple de simulation d'une série de nombres tirés d'une loi normale centrée réduite, que l'on décrit ensuite par un tiges et feuille et un histogramme.

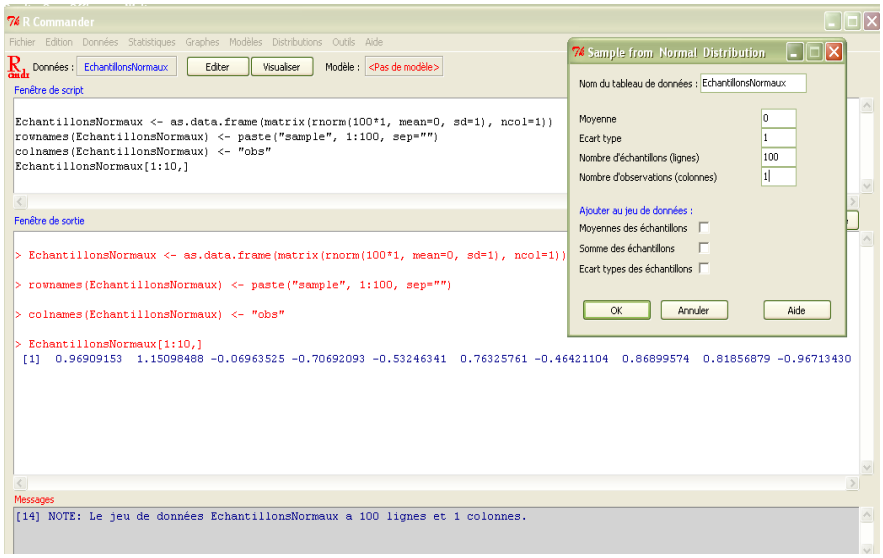


Chargement et exécution de Rcmdr

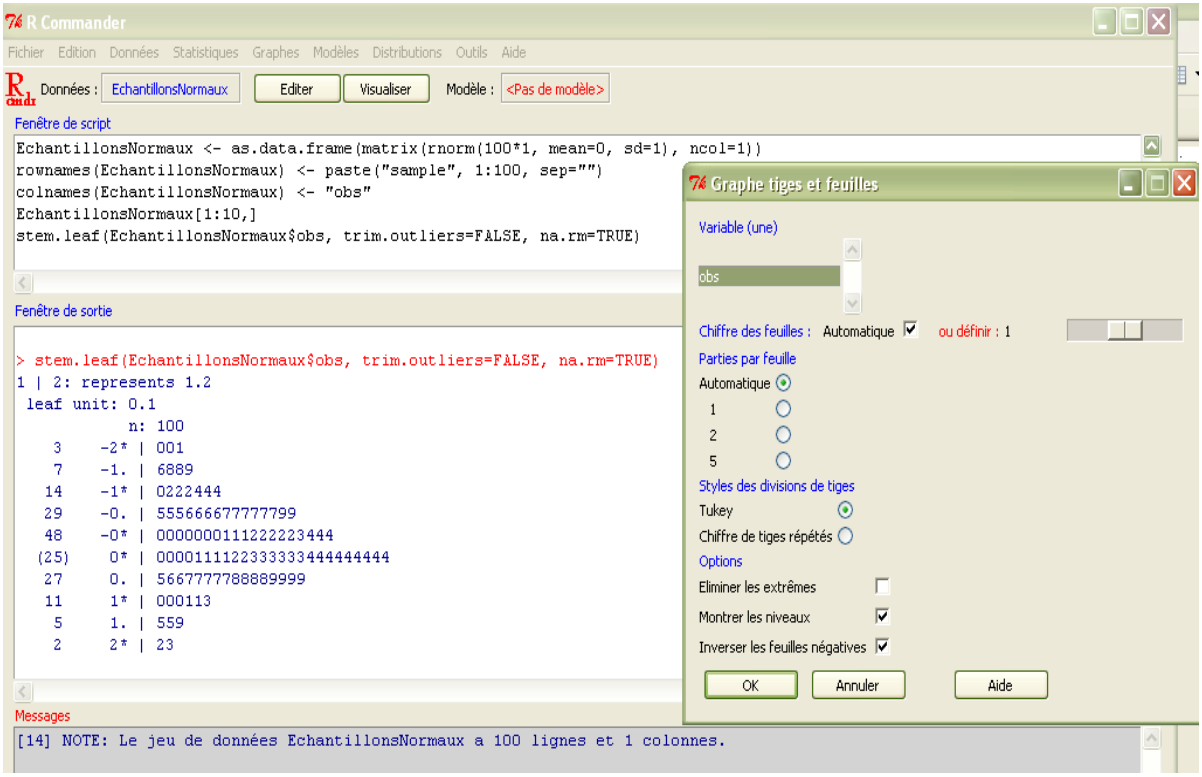
L'interface des menus à cliquer en français est lancée.

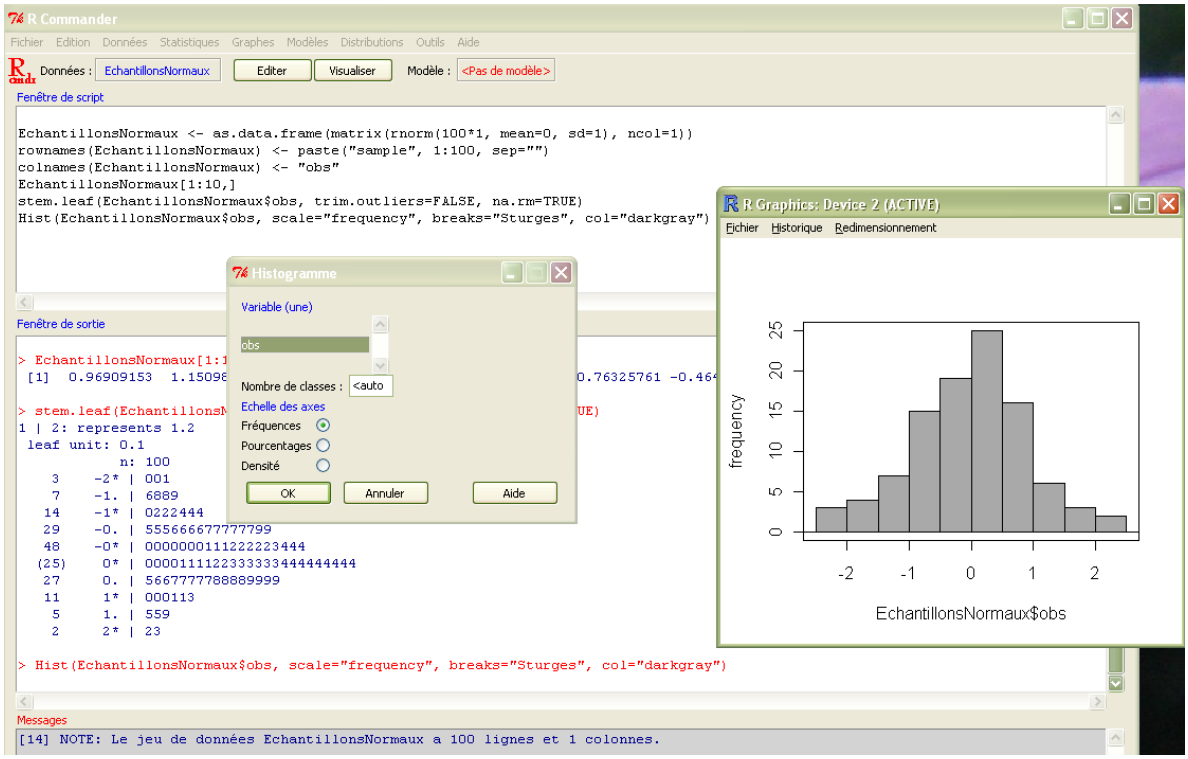
Utilisation directe : simulation de 100 nombres distribués selon la loi normale centré réduite et on fait afficher les 10 premières valeurs de la série simulée "EchantillonsNormaux".





Description de la série simulée, diagrammes en tiges et feuilles et histogramme :





3° Création et exécution des lignes de commandes

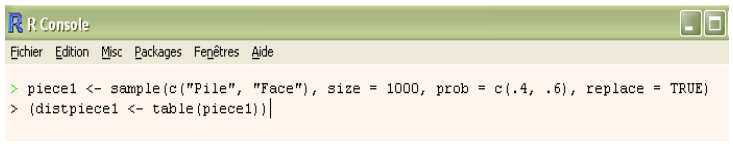
► Pour enchaîner les traitements ou programmer des fonctions **R**, il faut passer par les lignes de commande. On peut le faire directement dans la "console **R**", mais il est plus facile d'utiliser l'éditeur **Tinn-R** car il permet de saisir, d'enregistrer et d'utiliser les lignes de code que l'on crée pour un traitement ou une fonction.

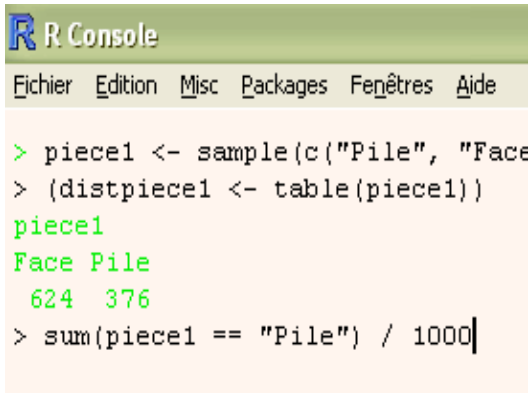
► On peut saisir et exécuter directement les lignes de commandes dans la console **R (R Console)**.

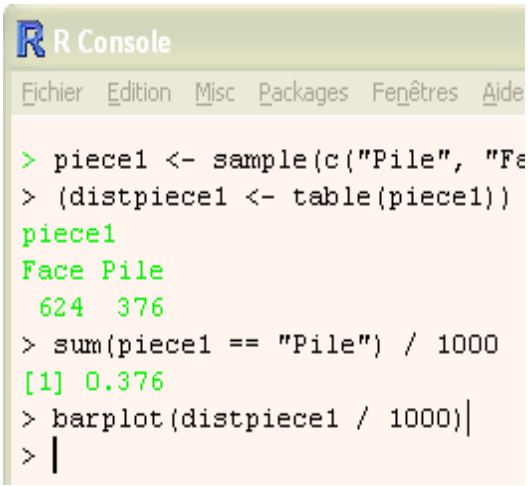
Pour accéder à la console **R**, il faut cliquer sur l'icône **R** créée lors de l'installation. Un fenêtre **RGui** (**G**raphic **u**ser **i**nterface) s'ouvre, qui contient, par défaut la console **R**, dans laquelle s'écrivent et s'exécutent les commandes et les fonctions **R**. La console **R** s'utilisera de préférence lorsque chaque ligne de commande est exécutée au fur et à mesure du traitement prévu. On exécute une ligne de commande en appuyant sur la touche entrée (valider). Une ligne peut comporter plusieurs commande séparées par des **";"**. Une commande peut s'écrire sur plusieurs lignes, des + apparaissent alors en début de ligne.

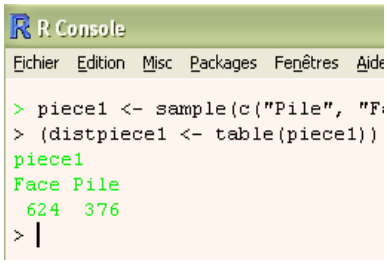
UTILISATION DIRECTE DE LA CONSOLE R

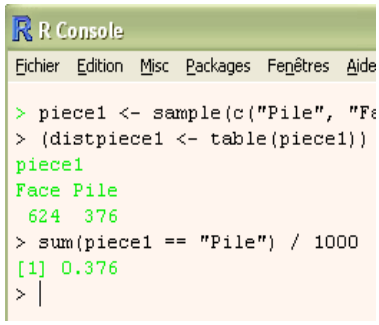
SAISIE DES LIGNES DE COMMANDE(S) ----->	**AFFICHAGE DU RÉSULTAT**

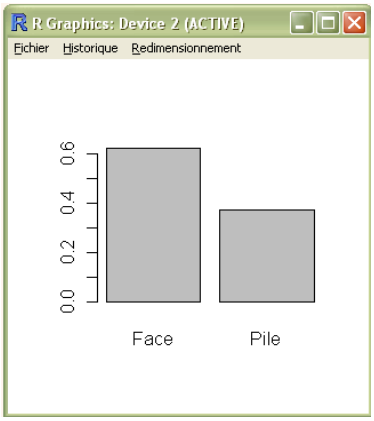










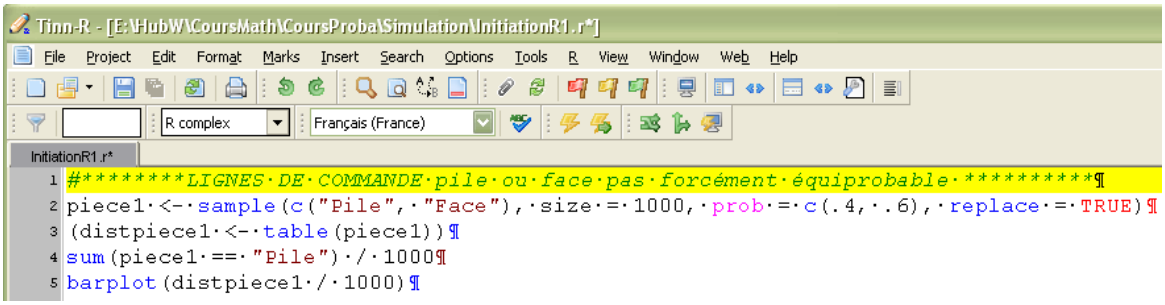


► Lignes de commandes saisies dans **Tinn-R** et exécutées dans la console **R**.

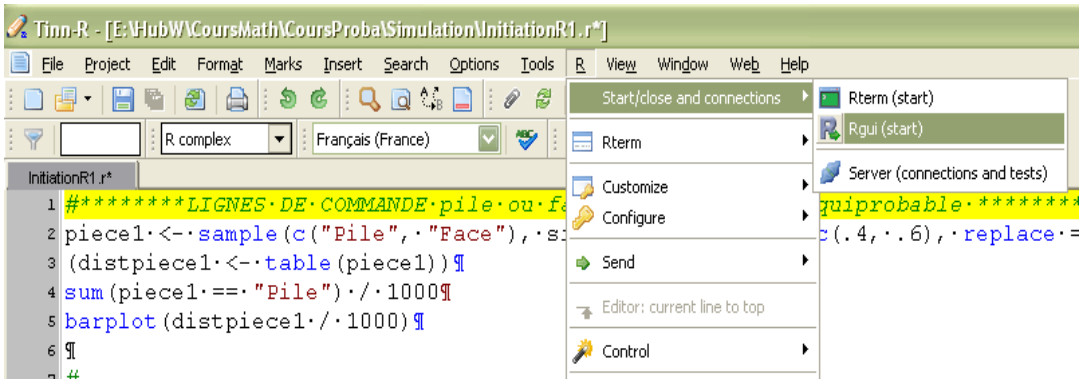
Lorsque l'on veut faire des traitement par lots (exécuter plusieurs lignes de commandes groupées) ou programmer des fonctions, l'utilisation de l'éditeur **Tinn-R** facilitera grandement l'écriture, la vérification et l'exécution des procédures ainsi créées.

La procédure classique consiste à suivre les étapes suivantes :

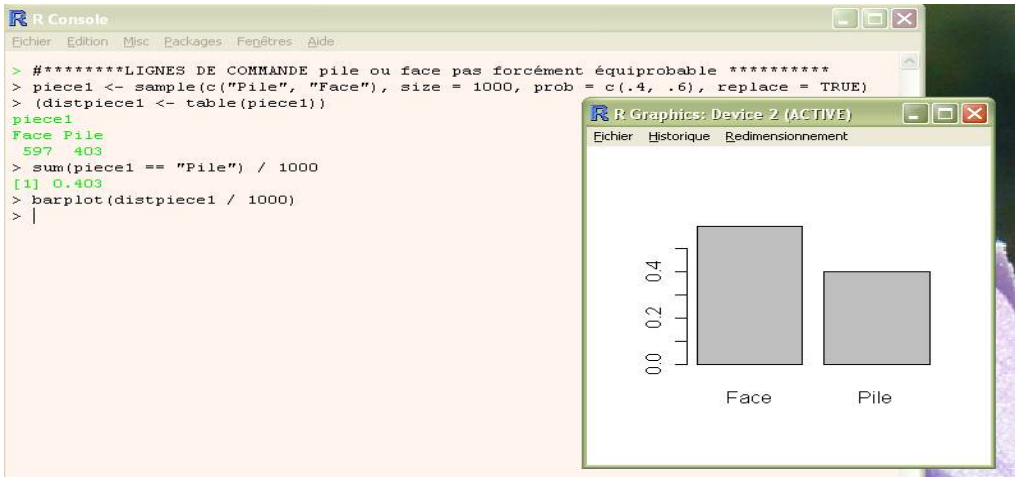
a Écriture du code dans **Tinn-R**.



b Lancer la console **R** (Rgui pour Graphic User Interface) à partir des menus **Tinn-R**.



c Copier coller les lignes de commande de **Tinn-R** dans la console **R**. Les lignes de commandes sont exécutées automatiquement et les résultats affichés, dans la console pour les résultats numériques, dans une ou plusieurs fenêtres graphiques, pour les graphiques.



3° Utilisation de fichiers contenant les lignes de commande ou les fonctions à exécuter

Pour utiliser les exemples fournis dans des fichiers, plusieurs cas peuvent se présenter.

► S'il s'agit de **lignes de commandes** fournies **dans le texte d'un fichier texte**, il suffit alors de sélectionner les lignes concernées et de les copier-coller dans la console **R** où elles seront automatiquement exécutées (sauf peut-être la dernière qu'il faudra valider) et les résultats seront affichés. Si on veut les modifier pour les adapter il faut soit les modifier dans le traitement de texte ou bien passer par **Tinn-R**.
Par contre si c'est un fichier du type pdf il risque d'y avoir des problèmes causés par le fait que les fins de ligne sont transformés en fin de paragraphe. Il vaut mieux donc éviter, au moins dans la période d'apprentissage.
Il se peut arriver, rarement, que la console **R** interprète mal certains caractères du traitement de texte, ayant un aspect visuel "normal". L'erreur est alors difficilement décelable, sauf en passant par **Tinn-R**.

► S'il s'agit de **lignes de commandes constituant une fonction**, fournies **dans le texte d'un fichier texte**, on procède comme précédemment en prenant bien soin de s'assurer que la dernière ligne a bien été validée (dans le cas contraire un + apparaît en début de ligne). La fonction vient d'être introduite en mémoire. Pour l'utiliser il suffit de saisir son **nom**, suivit sans espace de (). Ce **nom** figure obligatoirement en début du code. Dans ce cas, ce sont les valeurs des paramètre par défaut, indiquées dans la première ligne du code de la fonction, qui seront pris en compte. Pour utiliser d'autres valeurs, il suffit de les indiquer à l'intérieur des (). Exemple : **pileface()** réalise 2000 simulations du jet de deux pièces équilibrées (cf. le **II**). Pour en réaliser 6000, je saisis **pileface(6000)** ou **pileface(nbsim = 6000)**.

► Une bonne solution consiste à disposer des fichiers texte au format **Tinn-R** (extension **.r**) contenant les lignes de codes voulues. Un fichier **Tinn-R** est un simple fichier texte basique. Il peut contenir les lignes de code de une ou plusieurs procédures, les lignes de code de une ou plusieurs fonctions ou un mélange de lignes de procédures et de ligne de fonctions, comme par exemple dans le fichier **"InitiationR1.r"** qui contient les lignes de code des procédures et les lignes de codes des fonctions présentées dans les tableaux du **II**.
Il suffit alors de copier-coller les lignes de code que l'on veut exécuter.

► Une autre solution pertinente lorsqu'il s'agit d'utiliser une fonction, consiste à la faire lire et charger directement depuis le fichier source sur le disque dur. Prenons l'exemple de la fonction **IFexact1()**, dont les lignes de code sont dans le fichier **IF_BinomialExact1.r**. Il peut d'abord indiquer à **R** le dossier par défaut dans lequel se trouve le fichier à charger, en utilisant le menu « Fichier–Changer le répertoire courant ». Puis taper dans la console **R**, la commande **source("IF_BinomialExact1.r")**. Pour exécuter la fonction, il suffit ensuite de taper **Ifexact1()** ou, par exemple, **IFexact1(n = 150, p = .2, kobs = 25, proba = .95)**.

II - QUELQUES EXEMPLES SIMPLES COMMENTÉS

Convention typographique : `Les lignes en orange` contiennent les lignes de commande **R**. ***`Les lignes en italique vert`*** sont des parties de réponses de **R** (à ne pas coller dans la console). Les textes en `turquoise` ou **`bleu clair`** contiennent le code des fonctions **R**. **`Les # commentaires sont en noir`**, précédés de #. Les **`mots en rouge sombre`** sont les mots réservés aux commandes et fonctions internes de **R**.

`` #**LIGNES DE COMMANDE pile ou face pas forcément équiprobable ** piece1 <- sample(c("Pile", "Face"), size = 1000, prob = c(.4, .6), replace = TRUE) (distpiece1 <- table(piece1)) barplot(distpiece1 / 1000) sum(piece1 == "Pile") / 1000 ```	``` <- est la commande d'affectation. C() créé un vecteur (au sens informatique). sample tire size(1000) fois avec remise dans l'ensemble {"Pile", "Face"}, avec une probabilité de 0,4 pour "Pile" et de 0,6 pour "Face". Les n(1000) résultats obtenus sont mis dans le vecteur piece1. table(piece1) effectue le tri à plat (tableau des effectifs) de la série obtenue. barplot effectue le diagramme en barre. sum(piece1 == "Pile") compte le nombre de "Pile". ```
`` #**LIGNES DE COMMANDE pile ou face avec 2 pièces différentes ** piece1 <- sample(c("Pile", "Face"), size = 1000, prob = c(.4, .6), replace = T) piece2 <- sample(c("Pile", "Face"), size = 1000, prob = c(.5, .5), replace = T) deuxpieces <- paste(piece1,piece2, sep = "") table(deuxpieces) barplot(table(deuxpieces) / 1000) ```	``` La pièce 1 est déséquilibrée, la pièce 2 non. paste réuni deux à deux chacun des 1000 résultats de piece1 et piece2, par exemple PileFace ... ```
`` #**FONCTION jet simultané de 2 pièces identiques équilibrées** pileface <- function(nbsim = 2000){ resultats <- rep(NA, 3) names(resultats) <- c("deuxpils", "deuxfaces", "autre") for(i in 1:nbsim){ pieceA <- sample(c("Pile", "Face"), 1) pieceB <- sample(c("Pile", "Face"), 1) if(pieceA == "Pile" & pieceB == "Pile") { resultats[1] <- resultats[1] + 1} else { if(pieceA == "Face" & pieceB == "Face") { resultats[2] <- resultats[2] + 1} else { resultats[3] <- resultats[3] + 1 } } } print(resultats) print(resultats / nbsim) barplot(resultats / nbsim) } ```	``` Fonction effectuant nbsim (2000) lancers de deux pièces. Paramètres et valeurs par défaut,début du corps de fonction Initialisation d'un "vecteur" à 3 composantes nommer les 3 composantes du vecteur début de la boucle des nbsim lancers pieceA équilibrée pieceB équilibrée comptage des "Pile Pile" comptage des "Face Face" comptage des autres résultats. Fin des tests et des boucles Affichage des résultats. Fin du corps de fonction. ```
`` #Le problème historique du grand duc de Toscane (Somme de 3 dés) #****LIGNES DE COMMANDE Simulation GrandDuc**** de1 <- sample(c(1:6), 1000, replace = TRUE) (distde1 <- table(de1)) barplot(distde1 / 1000) de2 <- sample(c(1:6), 1000, replace = T) (distde2 <- table(de2)) dev.new() barplot(distde2 / 1000) de3 <- sample(c(1:6), 1000, replace = T) (distde3 <- table(de3)) dev.new() barplot(distde3 / 1000) de <- de1 + de2 + de3 (distde <- table(de)) dev.new() barplot(distde / 1000) nbneuf <- sum(de == 9) nbdix <- sum(de == 10) cat("Fréquence des neuf =", nbneuf / 1000, "\n") cat("Fréquence des dix =", nbdix / 1000, "\n") barplot(distde, xlab = "Somme des numéros des 3 faces", ylab = "Effectifs simulés", main = paste("Diagramme en barre de 1000 simulations\n", "du jet de 3 dés équilibrés")) ```	``` 1000 jets d'un dé équilibré, la série des 1000 résultats est mise dans le vecteur de1 tableau des effectifs de la série obtenus diagramme en barres ouvre une nouvelle fenêtre graphique même chose avec un autre dé équilibré, la série des 1000 résultats est mise dans le vecteur de2 même chose avec un autre dé équilibré, la série des 1000 résultats est mise dans le vecteur de3 somme, composante à composante des 3 vecteurs, les 1000 résultats sont mis dans le vecteur de. Tableau des effectifs de la série de, diagramme en barres comptage du nombre de neuf et du nombre de 10. affichage des résultats. ```
`` #****FONCTION simulation Grand Duc******** simgrandduc <- function(nbsim=1000){ de1 <- sample(c(1:6), nbsim, replace = TRUE) de2 <- sample(c(1:6), nbsim, replace = TRUE) de3 <- sample(c(1:6), nbsim, replace = TRUE) de <- de1 + de2 + de3 distde <- table(de) print(distde) barplot(distde / nbsim) } ```	``` La fonction effectue nbsim lancers de 3 dés. On additionne les résultats obtenus tableau des effectifs des 1000 sommes obtenues et leur diagramme en barres. ```

```
#**LIGNES DE COMMANDE probabilité Grand Duc********
#**Somme des valeurs des faces obtenues en jetant 3 dés**
#***Calculer avec le modèle mathématique "exact"****
#****Construire l'univers correspondant à cette expérience***
serieS3de <- array(data = NA, dim = c(6, 6, 6))
 for(i in 1:6){
   for(j in 1:6){
     for(k in 1:6){
        serieS3de[i, j, k] <- i + j + k
                  }
               }
            }
 serieS3de
 (distS3de <- table(serieS3de))
 nbneuf <- sum(serieS3de == 9)
 nbdix <- sum(serieS3de == 10)
 cat("Probabilité de neuf =",nbneuf / 216,"\n")
 cat("Probabilité de dix =",nbdix / 216,"\n")
 dev.new()
 barplot(distS3de)
 graphics.off()
```

Initialisation d'un tableau de dimension3

boucles imbriquées pour parcourir tous les triplets possibles et générer l'univers des résultats possibles : les 216 valeurs obtenues sont mises dans le vecteur serieS3de.

Tableau des effectifs

Comptage du nombre de 9 et de10
calcul de la probabilité.

```
#**Fonction probabilité Grand Duc********
probgranduc <- function(){
serieS3de <- array(data = NA, dim = c(6, 6, 6))
 for(i in 1:6){
   for(j in 1:6){
     for(k in 1:6){
        serieS3de[i, j, k] <- i + j + k
                  }
               }
            }
 serieS3de
 distS3de <- table(serieS3de)
 nbneuf <- sum(serieS3de == 9)
 nbdix <- sum(serieS3de == 10)
 cat("Probabilité de neuf =",nbneuf / 216,"\n")
 cat("Probabilité de dix =",nbdix / 216,"\n")
 print(distS3de)
 barplot(distS3de / 216)
}
```

Même chose sous la forme d'une fonction.

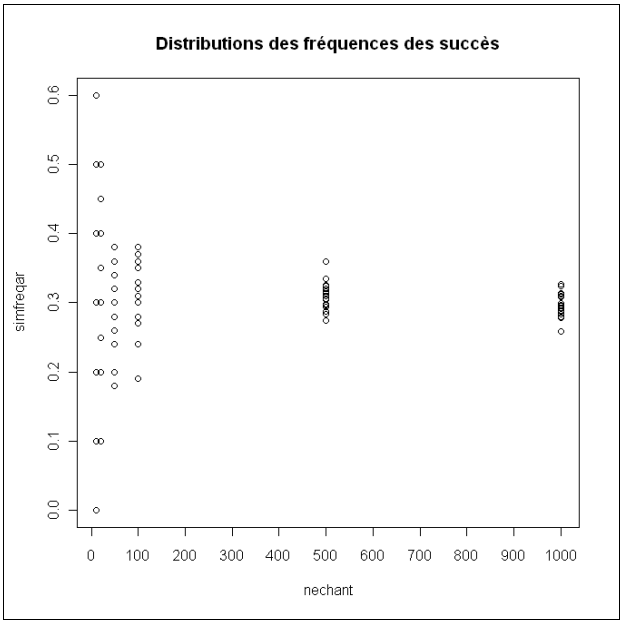
```
######LIGNES DE COMMANDES-- CALCULS DE PROBABILITÉS ##########
#-------------- Loi binomiale ---------------
# Calcul de P(A <= X <= B), X étant une v.a. de distribution
# binomiale
# de paramètres n=100 et p=0,52.
# Les exemples choisis peuvent servir de base à une réflexion
# sur les différentes façons de déterminer un intervalle de
# fluctuation, à partir
# de l'exemple 1 (Monsieur Z du document d'inspection).
# P(42<=X<=62):
sum(dbinom(42:62, 100, .52))
# P(43<=X<=62):
sum(dbinom(43:62, 100, .52))
# P(42<=X<=61):
sum(dbinom(42:61, 100, .52))
# P(X<=41) ; P(X<=42) ; P(X<=43):
pbinom(41:43, 100, .52)
#----- Combinaisons et loi hypergéométrique --------
# Calcul de P(X=3) ; x=3 ; X de loi hypergéométrique de
paramètres
# m = 3, n = 5, k = 4,
(proba <- choose(3, 3) * choose(5, 4-3) / choose(3+5, 4))
# Pour vérification :
(proba <- dhyper(x = 3, m = 3, n = 5, k = 4))
```

42:62 génère la suite des entiers de 42 à 62
dbinom génère un vecteur des probabilités binomiales de P(X=42) à P(X=62). sum en fait la somme

choose calcule les combinaisons

dhyper calcule les probabilités hypergéométriques.

```
# **LIGNES DE COMMANDES ****** SIMULATIONS NUMÉRIQUES *****
# Illustration graphique de la loi des grands nombres :
# Lorsque n augmente, on observe les suites de distributions
# Les fréquences tendent vers une valeur limite : la
probabilité
# Les écarts à cette valeur limite sont de plus en plus faibles
#
nechant <- rep(c(10, 20, 50, 100, 500, 1000),
               c(20, 20, 20, 20, 20, 20))
simfreqar <- c(rbinom(20, 10, .3)/10,
             rbinom(20, 20, .3)/20, rbinom(20, 50, .3)/50,
             rbinom(20, 100, .3)/100, rbinom(20, 500, .3)/500,
             rbinom(20, 1000, .3)/1000)
dev.off()
plot(nechant, simfreqar, xaxp = c(0, 1000, 10),
  main = "Distributions des fréquences des succès")
dev.new()
plot(as.factor(nechant), simfreqar,
  xlab = "Échantillons par tailles",
  ylab = "Fréquence des succès",
  main = "Résumé des distributions")
```



Pour s'entrainer : Réaliser cette simulation sous forme d'une fonction paramétrable...

On génère les tailles d'échantillons comme suit :
20 répétitions du nombre 10, 20 répétitions du nombre 20, …., 20 répétitions du nombre 1000, mise dans nechant qui constituerons les abscisses des points à tracer.
20 nombres au hasard sont tirés dans une distribution binomiale(10, 0,3), 20 nombres au hasard sont tirés dans une distribution binomiale(20, 0,3), …, 20 nombres au hasard sont tirés dans une distribution binomiale(1000, 0,3), Les fréquences sont calculées en même temps.
Nuage de points des fréquences en fonction des tailles d'échantillons.
Résumé des séries de 20 valeurs (binomiales) sous formes de boites à moustaches.