République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITÉ MOHAMED KHIDER, BISKRA

FACULTÉ DES SCIENCES EXACTES ET SCIENCES DE LA NATURE ET LA VIE

DÉPARTEMENT DE MATHÉMATIQUES

Domaine Mathématiques et Informatique

Filière : Mathématiques

Spécialité : STATISTIQUE

Mémoire de fin d'étude en Master

Intitulé :

Simulation Statistique

Présenté par **BAISSA Kenza**

Devant le Jury :

Encadreur : **Djabrane YAHIA**

Examinateur 1 : **Benatia FATAH**

Examinateur 2 : **Touba SONIA**

Juin 2012

Remerciements

Tout d'abord, je remercie DIEU, de m'avoir aidé et donné la volonté pour arriver à ce stade et réaliser ce modeste travail.

Mes vifs remerciements sont adressé à mon encadreur Dr Yahia Djabrane pour ces conseils, ses orientations, et la confiance qu'il m'accorde.

Je tien à remercie les membres du Jury qui m'ont fait l'honneur de participer à ma soutenance.

Je remercie tous les enseignants qui ont contribués à ma formation, ainsi que tous les employés du département de Mathématiques.

Je remercie tout particulièrement mes parents pour leurs encouragements et soutien sur tous les aspects et aussi toute ma famille. Je n'oublier pas l'ensemble de mes amis proches et aussi mes collègues du deuxième Mastère.

Je remercie tous ceux qui ont contribué de près ou de loin dans ce mémoire.

Je vous dis : Merci

Kenza Baissa

Table des matières

Table des figures

Résumé

Dans ce mémoire, j'ai utilisé des techniques de simulation par le logiciel R, dont le but principal est de simuler des variables aléatoires continues et discrètes.

Mes objectifs sont : apprendre à manipuler les données, apprendre à faire des graphiques, apprendre à utiliser la documentation et le système d'aide et aussi apprendre les bases du langage.

Introduction

Le terme simulation est dérivé du mot latin " *simulaire* " qui veut dire : copier, faire paraître comme réelle une chose qui ne l'est pas. Simuler le fonctionnement d'un système c'est à dire imiter son fonctionnement au cours de temps en manipulant un modèle. La simulation est donc une méthodologie essentiellement pratique qui permet de modéliser aussi bien des systèmes conceptuels (qu'on veut savoir) que des systèmes existants déjà.

La simulation n'est en effet ni des mathématiques, ni de la physique, ni de l'informatique, ni l'économie, ni de la philosophie, mais c'est un peu de tout cela à la fois. On la définit parfois comme l'ensemble des activités liées au traitement ou la visualisation des données, les calculs numériques ou symboliques et les représentations graphiques.

La programmation des modèles de simulation a été facile par les progrès de l'informatique. Le langage est en quelque sorte l'interface entre la machine et son utilisateur, cependant les langages universels sont également utilisés (FORTRAN, PASCAL, C,...), car dotés de générateur de nombre au hasard en peut utiliser également des logiciels de mathématiques (Mathématica, Matlab, Scilab...) ou statistique (Statistica, SPSS, R...).

Dans beaucoup de situations, que ce soit de la vie courante ou dans la recherche scientifique, le chercheur est confronté à des problèmes dont il recherche des solutions sur la base de certaines hypothèses et contraintes de départ. Pour résoudre ce type de problème, il existe des méthodes analytiques applicables à des situations où le modèle permet de traiter les différentes variables par des équations mathématiquement maniables, et des

méthodes numériques où la complexité du modèle impose un morcellement du problème, notamment par l'identification des différentes variables qui entrent en jeu et l'étude de leurs interactions. Cette dernière approche s'accompagne souvent d'une importante masse de calculs. Les techniques de simulation sont des techniques numériques : simuler un phénomène signifie essentiellement reconstituer son évolution.

L'objectif de ce mémoire est l'étude de simulation statistique. Il est composé de trois chapitres : dans le première chapitre, nous donnons une présentation générale sur le logiciel ou langage de programmation statistique R : l'installation du logiciel R et les packages associe aussi l'aide intégrée au logiciel R comme la commande help ( ) et quelques commandes complémentaires. Créer et manipuler des données par exemple les objets, listes, vecteurs, matrices, tableau de donnes et les fonctions, de plus à les éléments de programmation de R.

La simulation utilise des nombres aléatoires, et les méthodes pour en générer seront abordées au deuxième chapitre. Les méthodes de simulation statistique comprennent des méthodes et des techniques de génération de variables aléatoires de lois discrètes ou continues. Cependant, nous utiliserons déjà des nombres aléatoires dans ce chapitre.

Il existe plusieurs méthodes de simulation, notamment : Méthode d'inversion de la fonction de répartition, la méthode de Box et Muller, la méthode de rejet, simulation par méthode de Monte-Carlo et la méthode de comparaison. On va se donner un problème simple à résoudre, et on va appliquer la démarche dont je viens de parler, au deuxième chapitre.

Finalement, dans le troisième chapitre, je vais étudié quelques estimateurs non paramétrique tels que la fonction de répartition empirique, la fonction des quantiles empiriques et la densité non paramétrique. Une étude de simulation portant sur ces statistiques est donnée.

Chapitre 1

Logiciel R

1.1 Présentation de R

1.1.1 Généralités

R est un système, communément appelé langage et logiciel, qui permet de réaliser des analyses statistiques. Plus particulièrement, il comporte des moyens qui rendent possible la manipulation des données, les calculs et les représentations graphiques. *R* a aussi la possibilité d'exécuter des programmes stockés dans des fichiers textes et comporte un grand nombre de procédures statistiques appelées paquets. Ces derniers permettent de traiter assez rapidement des sujets aussi variés que les modèles linéaires (simples et généralisés), la régression (linéaire et non linéaire), les séries chronologiques, les tests paramétriques et non paramétriques classiques, les différentes méthodes d'analyse des données,... Plusieurs paquets, tels *ade4, FactoMineR, MASS, multivariate, scatterplot3d et rgl* entre autres sont destinés à l'analyse des données statistiques multidimensionnelles.

Il a été initialement créé, en 1996, par *Robert Gentleman* et *Ross Ihaka* du département de statistique de l'Université d'Auckland en Nouvelle Zélande. Depuis 1997, il s'est formé une équipe "*R Core Team*" qui développe *R*. Il est conçu pour pouvoir être utilisé avec les systèmes d'exploitation *Unix*, *Linux*, *Windows* et *MacOS*.

R est devenu un logiciel libre et gratuit en 1995. R est à la fois un langage de programmation et un progiciel de fonctions statistiques. La version de base de R contient déjà un grand nombre de fonctions statistiques et graphiques permettant, par exemple, de calculer une moyenne ou une variance ou de tracer un histogramme. De nombreux chercheurs ont développé au cours des années des fonctions plus avancées qui sont disponibles à tous les utilisateurs de R. Ces fonctions sont regroupées en bibliothèques qui sont disponibles pour téléchargement sur le site du projet R : *http ://www.r-project.org/*.

FIG. 1.1 – La plateforme windows, macos, linux

1.1.2 CRAN

Le "*Comprehensive R Archive Network*" (CRAN[1]) est un ensemble de sites qui fournit ce qui est nécessaire à la distribution de R, ses extensions sa documentation, ses fichier sources et ses fichiers binaires. Le site maître du CRAN est situé en Autriche à vienne , nous pouvons y accéder par L'URL : *http ://cran.r-project.org/*

[1]Réseau d'archives de R globales (en français).

1.1.3 Installation du logiciel R

Le logiciel est télécharger sur le site web officiel de R (*http ://www.r-project.org/*), il faut ensuite se diriger dans download CRAN. Choisissez un site miroir proche de chez vous (par exemple : en France), les téléchargements seront probablement plus rapides, vous trouvez ensuite un encadrement légendé download and install R.

Sous Windows XP

Pour Windows, cliquez sur Windows puis base. Cliquez ensuite sur le fichier "exist" (par exemple : **R-2.14.1-win.exe**). Le programme d'installation est alors téléchargé sur votre ordinateur. Il suffit de cliquer dessus et de suivre les instructions. Un dossier portant le nom de la version de R télécharger (R-2.14.1-win.exe dans ce cas) est crée.

Il est situé, à partir du disque dur C, dans la série de dossiers suivante : program files /R. Dans ce dossier se trouve le dossier library qui comprend les packages base de R. Un autre élément utile doit être localisé : le fichier .R data. Celui-ci n'est pas apparent au début. Il contiendra tous les objets que vous créerez et sauvegardez dans R. Sur mon ordinateur, il apparaît, à partir du disque dur C.

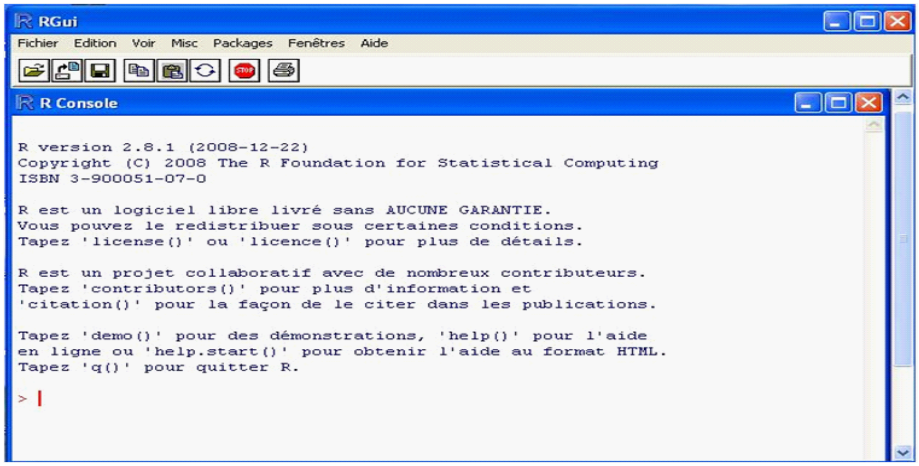


FIG. 1.2 – Démarrage de R pour Windows

1.1.4 Instalation des packages

Définition 1.1 *Un package R se présente sous la forme d'un fichier compressé qui regroupe des fonctions, les documente et peut contenir du package facilite l'utilisation et le partage de fonction de R. Autrement dire, un package est un compilation d'outils certains sont déjà présents dans l'installation de base de R. En effet, lors de l'installation de R, et un dossier Library s'est crée par défaut, sous Windows fichiers binaires pré-compilés. Il comprend les packages de base de R. Mais d'autres packages qui vous seront utiles pour réaliser vos analyses statistiques seront à télécharger puis à installer.*

Sur le CRAN, le réseau officiel de distribution de code et de documentation R. Plus de 3000 bibliothèques R sont en ce moment (il y'a des mois) disponibles sur le site CRAN. D'autres bibliothèques sont disponibles sur la page Web de chercheurs ou en annexe de publications scientifiques.

L'installation du module («package») **package ade4** doit se faire une seule fois, lors de la première utilisation. Pour les utilisations subséquentes, il suffira de charger le package après l'ouverture du logiciel R. On suppose qu'on dispose d'une connexion internet pour effectuer ces opérations. On peut ensuite travailler hors-connexion.

1) Ouvrir le logiciel R en cliquant sur l'icône créé sur votre bureau lors du téléchargement. Dans la fenêtre qui apparaît, aller dans menu "packages / Choisir le site miroir "CRAN" (pas trop loin si possible), par exemple : france lyon et cliquer sur OK (voir figure 1.3).

2) Menu " packages / Installer des packages" , Dans la liste des packages, choisir ade4 et cliquer sur OK.

3) Menu « packages/ Mettre à jour les packages », pour obtenir la derniere version des packages déjà installés.

FIG. 1.3 – Choisir le site miroir CRAN dans la fenêtre

Charger un package déjà installé

Aller dans le menu "packages -Charger le package" puis sélectionner le package ou taper la commande :

```
> library( nom package)
> library(nom-de-la-bibliotheque) # par exemple : library(MASS)
```

ou encore :

```
> require(nom-de-la-bibliotheque) # par exemple : require(MASS)
> library()# permet aussi de lister les packages installes sur notre machine.
```

Obtenir la liste des packages chargés (voir figure 1.4).

1.1.5 Exemple d'utilisation du package : rpart

– Pour utiliser le package rpart (pour construire des arbres de décision), il faut charger le package rpart à chaque ouverture du logiciel R. Utiliser les menus, choisir rpart dans la liste proposée et cliquer sur ok.

FIG. 1.4 – Liste des packages chargés

Exemple 1.1 *ligne de commande :*

> *library(rpart)*

> *search( )*

On peut « décharger » un package (pour éviter les conflits entre fonctions de packages différents par exemple).

Exemple 1.2 > *detach(package : rpart)*

> *search()*

1.2 Aide intégrée au logiciel R

1.2.1 La Commande help ( )

R contient une aide en ligne (help : en anglais) trés complète et trés bien structurée sur toutes les fonctions que vous pouvez utiliser ainsi que sur les différents symboles du langage. Cette aide accessible au moyen de plusieurs commandes dont la principale est help ( ). Elle s'utilise en mode ligne de commande. Taper par exemple :

```
help(help)
```

```
la commande help( ) possede un alliais qui est le point d'interrogation : ?
```

?sum

?sd

?"+"

Attention : il peut arriver que cet alias ne fonctionne pas et il est alors nécessaire d'utiliser la fonction help( ) avec des guillemets.

```
?function # Ne fonctionne pas
help (function ) # Renvoie une erreur
help("function") # Appel correct
```

Notes et commentaires :

Il est important d'ajouter des notes et des commentaires au fichier texte contenant les commandes sauvegardées, afin de se rappeler plus tard de l'objectif des calculs ou encore la raison de telle ou telle façon de faire. Une ligne de commentaires commence par le caractère « # ». Lorsqu'on les copie dans la console R, les lignes de commande ou les sections de lignes commençant par « # » ne sont pas exécutées par R.

1.2.2 Quelques commandes complémentaires

En plus de la commande principale help ( ), il existe quelques autres fonctions complémentaires qui peuvent se révéler utiles lorsque l'on cherche de l'aide sur une commande donnée. Nous les présentons ci-dessous :

– **help.start ( )** : cette fonction ouvre un navigateur avec plusieurs liens vers des manuels au format HTML, de l'aide sur toutes les fonctions présentes dans tous les packages de R (aussi au format HTML), une FAQ (Questions fréquemment posées) qui contient les réponses aux questions les plus fréquentemment posées sur R. Vous y trouvez égale-

ment un moteur de recherche dans l'aide mots clefs. On y trouve aussi quelques autres documents plus techniques.

- **help .search ( )** ou **? ? ( )** : cette fonction est à utiliser lorsque l'on ne connaît pas le nom d'une commande. Elle renvoie une liste des différentes fonctions (et des packages dans les quels elles se trouvent) ayant un rapport avec votre requête. Essayer par exemple : **help.search("mean")**.

- **apropos( )** : cette instruction renvoie une liste de toutes les fonctions dont le nom contient le paramètre d'appel. Ainsi **apropos("mean")** renvoie toutes les fonctions dont le nom contient le mot mean.

- Library (**help : package**) : cette commande permet de lister toutes les fonctions présentes dans la librairie package. Elle donne les mêmes résultats que la commande help("package"). Nous vous conseillons d'essayer les instructions suivantes afin d'obtenir la liste des fonctions principales de R :

```
library(help = base)
library (help = utils)
library(help = datasets )
library(help = stats )
library(help = graphics )
library(help = grDevices)
```

La fonction Library (help = package) renvoie la liste de toutes les fonctions de librairies packages. Inversement, l'instruction **find("fonction")** permet de savoir à quelle librairie appartient la fonction : function( ). Par exemple, la fonction **t.test** est dans le package (stats) :

```
> find("t.test")
[1] "package : stats"
```

Voici trois autres fonctions qui pourront également se révéler utiles :

– **data( )** : cette commande renvoie la liste de tous les jeux de données présents dans R.

– **example( )** : cette instruction donne des exemples sur l'utilisation d'une fonction. Ainsi, example(mean) exécute les instructions présentes dans la section exemples du fichier d'aide obtenu quand on tape **help(mean)**.

```
example("mean")
mean> x <- c(0:10, 50)# 50 valeurs entre 0 et 10
mean> xm <- mean(x)
mean> c(xm, mean(x, trim = 0.10))
[1] 8.75   5.50
```

– **demo( )** : cette instruction est similaire à la précédente, mais n'est accessible que pour un nombre limité de fonctions. Lorsque 'elle est disponible, elle présente un éventail des possibilités offertes par une fonction. Essayez par exemple : demo(graphics)

1.3 Créer et manipuler des données

Voici quelques définitions importantes dans la manipulation des données :

Définition 1.2 (objet) *un objet est un espace dans lequel vous pouvez stocker tout ce qui vous intéresse.*

Définition 1.3 (vecteur) *un vecteur est un objet d'un même mode pour toutes les valeur qui le constituent.*

Définition 1.4 (matrice) *une matrice est un objet d'un même mode pour toutes les valeur qui la constituent.*

Définition 1.5 (liste) *une liste est un objet permettant de stocker des objet qui peuvent être hétérogènes, c'est-à-dire qui n'ont pas tous le même mode ou la même longueur.*

Définition 1.6 (tableau) *un tableau de données, ou data.frame en anglais, est une liste particulière dont les composantes sont même longueur et dont les modes peuvent être différents(est une collection de vecteurs de même longueur). Un data.frame est un tableau à double entrée : les lignes sont les individus sur les quels les mesures sont faites et les colonnes sont les variables.*

1.3.1 Les objets : création et types

Création

Simplement par affectation avec les opérateurs <-, -> en lui donnant un nom :

```
> b <- sqrt(2) (cree l'objet b) # (racine carree)
> x <-b (x recoit la valeur b)
> x=b (x recoit la valeur b)
> b-> x (x recoit la valeur b)
```

Si un objet n'existe pas l'affectation le crée. Sinon l'affectation écrase la valeur précédente.

Affichage de la valeur d'un objet

```
> a (affiche le contenu de a)
> q (affiche le contenu de la fonction q qui permet de quitter)
> print(a) (affiche le contenu de a)
```

Les différents types d'un objet

- Objets simples : ce sont des objets qui ne comportent que des éléments de même mode.
- Objet composés (ou hétérogènes) : ce sont des objets qui comportent des éléments qui peuvent être de modes différents.

Les principaux modes sont :

- null (objet vide) : NULL

- logical (booleen) : TRUE, FALSE ou T, F

- numeric (valeur numurique) : 1,2.3222, pi, 1e-10,

- character (chaine de caractere) : 'bonjour', "K", "Bebe"

- complex (nombre complexe) : 2+0i, 2i

- complex (nombre complexe) : 2+0i, 2i

1.3.2 Vecteur

On déclare nos données sous forme matricielle (vecteur ligne ou matrice) dans plusieurs façons :

- la fonction c( ) pour un vecteur (de type ligne) :

```
> c(1,2,3,4)
[1] 1 2 3 4
```

Pour un vecteur colonne, il faut utiliser la fonction as.matrix() :

```
> as.matrix(c(1,2,3,4))
     [,1]
[1,]   1
[2,]   2
[3,]   3
[4,]   4
```

l'opérateur : (prioritaire sur les opérations au sein d'une expression)

```
> x=1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
```

La fonction seq de différents arguments **seq(form,to)** : form le début de la séquence, to pour la fin :

```
> seq(from=1, to=10)
[1] 1 2 3 4 5 6 7 8 9 10
> seq(from=1.5, to=4.7)
[1] 1.5 2.5 3.5 4.5
```

- **seq(from, to, by=)** : la même chose et by pour le pas ;

```
> seq(from=1, to=10, by=2)
[1] 1 3 5 7 9
```

seq(from, to, length.out=) : il créé une séquence de from jusqu'au to de la longueur length :

```
> seq(from=1, to=10, length=3)
[1] 1.0   5.5   10.0
```

1.3.3 Matrices

La fonction **matrix()** : remplissage par colonne par défaut :

```
matrix(data = NA, nrow = l, ncol =l , byrow = FALSE, dimnames = NULL)
```

data : les données pour remplir la matrice ;

nrow : le nombre de lignes et ncol le nombre de colonnes ;

dimnames : une liste de dimension 2 avec les noms des lignes et/ou colonnes

```
> matrix(data=1:6,nr=2,nc=3,
dimnames=list(c("row1", "row2"), c("C.1", "C.2", "C.3")))
     C.1   C.2  C.3
row1  1    3    5
row2  2    4    6
```

1.3.4 Les listes

la liste est un objet hétérogène. C'est donc un ensemble ordonné d'objets qui ne sont pas forcément tous du même mode ni de même longueur. Les objets sont appelés composants de la liste. Remarquons que

- les composants peuvent être de mode $\neq$,

- les composants peuvent avoir un nom,

- la liste n'est pas une classe.

Création

```
> tablev<- c("Paris", "Lyon")
> list <- list (1:3,ville="tablev") les composants peuvent avoir 1 nom
```

Forme une liste à 5 éléments, qui sont des vecteurs numériques de longueur 1, égaux respectivement à 1,2,3,4,5 :

```
> as.list(1:5), conversion explicite
```

Remarque 1.1 *on peut poser la question à R si l'objet est une liste*
> is.list(x) et la réponce est TRUE ou FALSE.

Extraction

```
> maliste [[2]]
#1eme composante de la liste ville
> maliste$ville
#2eme composante de la liste (Si la composante possede 1 nom)
> maliste [[1]] [1]
#1er element de la composante 1
> maliste$ville[1]
#1er element de la composante "ville"
```

1.3.5 Créer un tableau de données sous R

Pour créer un tableau de données sous R, il faut utiliser la fonction data.frame. Cette fonction permet de concaténer des vecteurs de mêmes longueur et de modes différents.

Exemple 1.3 *Saisissez deux vecteurs notés mat et phy :*

> mat<-c(19.6,17.6,18.2,16.0)

> phy<-c(19.1,17.8,18.7,16.1)

puis construisez le tableau de données, notées, à l'aide de la fonction data.frame en tapant la ligne de commande suivant :

> res<-data.frame(mat,phy)

> res

	mat	*phy*
1	*19.6*	*19.1*
2	*17.6*	*17.8*
3	*18.2*	*18.7*
4	*16.0*	*16.1*

Il est possible de donner des noms aux lignes du tableau de données avec l'option row.names qui doit fournir un vecteur de mode caractère et de longueur égale au nombre de lignes du tableau de données.

Exemple 1.4 *Le tableau de données ci-dessus correspond aux moyennes trimestrielles de mathématiques et de sciences physiques qu'ont obtenu Amin, Ahmed, Aziz et Assia. Donc, vous tapez la ligne de commande suivante pour voir apparaître les noms en face des quatre lignes :*

> res2<-data.frame(mat,phy,row.names = c(" Amin","Ahmed", "Aziz "," Assia"))

> res2

	mat	*phy*
Amin	*19.6*	*19.1*
Ahmed	*17.6*	*17.8*
Aziz	*18.2*	*18.7*
Assia	*16.0*	*16.1*

1.3.6 Calcul arithmétique et fonctions simples

Les opérations suivantes sont effectuées composante par composante :

- la somme des deux vecteurs avec + ;

- le produit avec * ;

- le rapport avec / ;

- puissance avec ^ ;

Voici quelques fonction R simple :

- sum(x) : sommes des composantes de x ;

- prod(x) : produit des composantes de x ;

- max(x) : maximum des composantes de x ;

- which.max(x) : retourne l'indice du maximum des composantes de x ;

- range(x) : idem que c(min(x),max(x)) ;

- length(x) : nombre d'éléments dans x ;

- mean(x) : la moyenne des éléments dans x ;

- median(x) : la médiane des éléments dans x ;

- var(x) : la variance (corrigée) des éléments dans x ;

- cor(x) : matrice de corrélation si x est un data-frame et 1 sinon ;

- cov(x,y) : la covariance entre x et y ;

Le résultat est une valeur, sauf pour range(), var(), cor() et cov().

- round(x,n) : arrondi les éléments de x à n chiffres après la virgule ;

- rev(x) : inverse l'ordre de x ;

- sort(x) : trie les éléments de x dans ordre croissante ;

- rank(x) : rangs des éléments de x ;

- cumsum(x) : un vecteur avec les sommes cumulées de composantes de x ;

- cumprod(x) : idem pour le produit ;

- table(x) : retourne un tableau avec les effectifs de différentes valeurs de x ;

- which(x==a) : retourne un vecteur des indices de x pour les quels l'opération de comparaison est vraie ;

```
x=c(2,6,10,60,34)
> which(x<35)
[1] 1 2 3 5 #les indices pour les quels on a vrai la comparaison
> x[which(x<35)]
[1] 2 6 10 34
```

Graphiques

Possibilité de voir des exemples de graphiques avec demo(graphics) ou demo(persp). Lorsqu'une fonction graphique est tapée sur la console, une fenêtre graphique va s'ouvrir avec le graphe demandé.

Partitionner une fenêtre grapfique

`par(mfcol=c(nr,nc)) :`

on partitionne la fenêtre en une matrice de nr lignes et nc colonnes et le remplissage est réalisé par colonne.

- mfrow : idem mais les graphiques sont dessinés en ligne ;

- layout() : pour des partitions plus complexes

```
> layout(matrix(c(1,2,3,4),2,2)) # pour inserer dans un graphique
```

```
> layout(matrix(c(1,1,2,1),2,2),c(3,1),c(1,3))
> layout.show(2) # et visualiser la partition cree
```

- plot (x) : graphe des valeurs de x (sur l'ordonnée) en fonction des valeurs de x ;
- plot (x,y) : graphe des valeurs de y (l'ordonnée) en fonction des valeurs de x ;
- pie(x) : "camembert" des valeurs de x ;
- boxplot(x) : boxplot de x ;
- hist(x) : histogramme de x (pour x quantitative) ;
- borplot(x) : diagramme en colonnes (pour x qualitative) Pour chaque fonction, on a plusieurs options mais certaines sont communes :

– **type :** "p" : points, "l" : lignes, "b" les deux, "h" : lignes verticales, "s" : escaliers ; for stair steps ;
– **xlab, ylab :** noms des axes, variables caractères entre " " ;
– **main :** variable de type caractère ; sub : sous-titre ;
– **points(x,y) :** ajoute des points ;
– **lines(x,y) :** idem mais avec des lignes ;
– **segments(x0,y0,x1,y1) :** trace une ligne entre les points (x0, y0) et (x1, y1) ;
– **abline(a,b) :** trace une ligne de pente b et ordonnée à l'origine a ;
– **legend(x,y,legend) :** ajoute une légende au point de coordonnées (x, y) avec les symboles donnés par legend.

1.3.7 Les fonctions

Structure générale pour créer des fonctions

La syntaxe générale de définition d'une fonction est la suivante :

```
nom-fonction<-function(arg1[=expr1],arg2[=expr2]...)
{
```

```
blocs d'instructions
}
```

Les accolades permettent de séparer les instructions par rapport à la signature de la fonction, les crochets, eux permettent de spécifies valeurs par défaut des arguments de façon facultative.

Exemple 1.5 *voici un exemple de simulation de la fonction* $f(x) = x^2$:

```
> carre=function(x) {x*x}
> carre(2)
[1] 4
```

Exemple 1.6 *exemple définissant la moyenne d'un vecteur :*

```
moyenne.vec <- function(x){
+ s<-sum(x); # Somme des'elements de x
+ n<-length(x); # Nombre d'elements de x
+ res<-round(s/n,2); # resultat arrondi
+ return(res)
+ }
```

Par l'intermédiaire d'un éditeur de texte grâce à la fonction fix() :

```
> fix(moyenne.vect)
```

Cette commande lance un éditeur de texte qui travaille avec le système R et qui permet de définir des fonctions. Le code de la fonction est écrit à partir de l'éditeur. Cette méthode est plus confortable et pratique que la première.
La fonction return() permet de spécifier le résultat de la fonction, lorsque l'instruction correspondante à return n'est pas utilisée, R retourne le résultat de la dernière expression évaluée dans la fonction.

Les lois de probabilité

R fournit un jeu trés complet de tables statistiques. Les fonctions fournies nous donnent la densité de probabilité cumulée (ou fonction de répartition), $P(X \leq x)$, la densité de probabilité, le quantile (ou fonction de répartition inverse) et des série de nombres pseudo aléatoires générées suivant la loi de probabilité considérée.

Loi de probabilité	**Noms R**	**Arguments supplémentaires**
Beta	beta	shape1, shape2, ncp
Binomiale	binom	size, prob
Cauchy	cauchy	location, scale
Khi2	chisq	df, ncp
Exponentielle	exp	rate
Fisher	f	df1, df2, ncp
Gamma	gamma	shape, scale
Géométrique	geom	prob
Hypergéométrique	hyper	m, n, k
Log-normale	lnorm	meanlog, sdlog
Logistique	logis	location, scale
Négative binomiale	nbinom	size, prob
Normal	norm	mean, sd
Poisson	pois	lambda
Student	t	df, ncp
Uniforme	unif	min, max
Weibull	weibull	shape, scale

Tab.1- Lois de probabilités dans R

Pour les lois Beta, Khi2, Fisher et Student ncp désigne le paramètre de décentrage.

Quatre préfixes sont possibles pour obtenir les sorties désirées :

1. **d** : correspond à la densité de probabilité.
2. **p** : correspond à la fonction de répartition (valeur inverse du quantile).
3. **q** : correspond au quantile.
4. **r** : correspond à la génération aléatoire de nombre.

Le premier argument est 'x' (une valeur) pour dnom-loi, 'q' (un quantile) pour pnom-loi, et 'n' (un nombre) pour rnom-loi.

Exemples avec la loi normale (arguments par défauts : mean=0, sd=1) :

```
> dnorm(0.37)
[1] 0.3725483
```

donne l'ordonnée de la courbe de densité à la valeur 0.37.

```
> pnorm(0)
[1] 0.5
```

donne la probabilité de la fonction de répartition au quantile 0.

```
> round(qnorm(0.975),2)
[1] 1.96
```

donne le quantile pour la probabilité de 0.975 de la fonction de répartition.

```
> rnorm(3) # engendre 3 valeurs de la loi normale
[1] -1.5651925   1.2417975   0.9014982
```

1.3.8 Eléments de programmation

Les instructions de sélection

L'expression if :

```
if ( expression logique) < expression 1>
```

ou

```
if (expression logique)<expression1> else <expression2>
```

Si l'expression logique prend la valeur vrai (TRUE), l'expression 1 est évaluée, sinon c'est l'expression 2.

L'expression if else :

```
ifelse (expression logique, vecteur1, vecteur2)
```

Si l'expression logique prend la valeur vrai, l'expression retourne le vecteur 1, sinon elle retourne le vecteur 2. L'expression switch() aussi permet de réaliser des sélections.

Les instructions de répétition

L'expression while :

```
while (expression logique, expression)
```

Tant que l'expression logique est vraie, l'expression est exécutée. La valeur finale est celle obtenue lors de la dernière évaluation de l'expression.

L'expression for :

```
for (ind in vecteur) expression
```

ind est la variable boucle, vecteur est une expression vectorielle (souvent une séquence comme 1 :30) et expression est souvent un groupe d'expressions qui composent avec la variable ind. expression est évaluée de manière répétitive tant que ind parcourt les valeurs de vecteur.

L'instruction repeat aussi permet de réaliser des répétitions. L'instruction break, pour sortir d'une boucle, peut être utilisée dans toutes ces boucles et elle est la seule façon de terminer la boucle repeat.

L'instruction next peut être utilisée pour sauter une étape et passer à la suivante. Par ailleurs, R autorise la récursivité et il permet à l'utilisateur de créer ses propres opérateurs.

1.4 R est merveilleux !

Comme conclusion de ce chapitre, on peut dire :

1. Il y a (presque) toujours un package adapté à nos problèmes.

2. Nous pouvons programmer dans R pour compléter les fonctions.

3. Nous pouvons programmer de nouveaux packages distribuables.

Chapitre 2

Méthodes de Simulation

2.1 Méthode d'inversion de la fonction de répartition

Les transformations de variables nous permettront de simuler des échantillons fictifs d'une variable aléatoire à partir d'un ensemble de nombres aléatoires uniformément distribués sur [0, 1]. Dans certains cas, par exemple quand on peut expliciter complètement la fonction de répartition, pour atteindre ce but il suffira d'appliquer simplement une formule de transformation.

La méthode de la transformation inverse peut s'utiliser pour générer n'importe quelle variable aléatoire à condition que l'on connaisse analytiquement la fonction de répartition inverse $F^{-1}(x)$. Elle est basée sur le théorème suivant :

Théorème 2.1 *(Théorème de la transformation inverse) Soit X une variable aléatoire et $F(x)$ sa fonction de répartition. Si $F(x)$ est continue et strictement croissante, alors $Y = F(X)$ est une variable aléatoire dont la distribution est uniforme sur $[0, 1]$.*
Dit autrement, si U est une variable aléatoire uniforme sur $[0, 1]$, la variable $F^{-1}(U)$ est une variable aléatoire dont F est sa fonction de répartition.

Preuve. On a

$$\begin{aligned} y &= F(x) = \Pr(X \leq x) \\ &= \Pr(F(X) \leq F(x)) \\ &= \Pr(Y \leq y) = F_Y(y) \end{aligned}$$

ce qui prouve le théorème. ■

Une application directe de ce théorème permet, à partir d'une loi uniforme, de générer n'importe quelle autre loi. Ainsi, pour générer un échantillon fictif issu d'une variable aléatoire X, il faut connaître $F(x)$ et avoir une suite de nombres $y_1, y_2, ..., y_n$ issus d'une variable U uniforme sur $[0, 1]$. L'égalité $x = F^{-1}(y)$ permet d'obtenir l'échantillon

$$x_1 = F^{-1}(y_1), x_2 = F^{-1}(y_2), ..., x_n = F^{-1}(y_n)$$

issu d'une population distribuée selon F. La seule difficulté est celle de calculer F^{-1} en connaissant F. Dans certains cas cela n'est pas possible de manière explicite, et dans ces cas d'autres techniques, que l'on verra plus loin, permettent de simuler un échantillon de X.

Si X suit une loi exponentielle négative de moyenne égale à 1, alors sa fonction de répartition est : $F(x) = 1 - e^{-x}$. Pour appliquer le théorème de la transformation inverse, nous devons cherche la fonction inverse de $u = 1 - e^{-x}$. Cela donne $x = -log(1 - u)$.

Remarquons que si U est une variable aléatoire uniforme, $(1 - U)$ est une aussi variable aléatoire uniforme. Donc, si U est une variable aléatoire uniforme, alors $X = -logU$ est une variable aléatoire de loi exponentielle négative (de moyenne 1).

2.1.1 Algorithme de la transformation inverse

1. Générer U de distribution $U[0, 1]$.

2. Calculer $X = F^{-1}(U)$.

Par le théorème, la variable X aura bien la distribution attendu.

Exemple 2.1 (Génération de variables exponentielles) *pour générer un échantillon de loi exponentielle et en utilise la loi uniforme seulement (U une v.a. uniforme sur $[0,1]$). Soit X une v.a. d'une distribution exponentielle : On a $F(x) = 1 - e^{-x}$. Résoudre en x l'équation $u = 1 - e^{-x}$, conduit à $x = F^{-1}(u) = -\log(1-u)$. Donc, si $U \sim U_{[0,1]}$, alors $X = -\log(1-U)$ suit la loi exponentielle de paramètre égale 1.*

Pour générer un échantillon de taille N, on utilise le syntaxe "Random " et la loi utilisée, comme :

```
le nom de la variable aleatoire<- r le syntaxe de la loi (N, ses parametres)
```

Par exemple : on génère un échantillon de taille $N = 150$ de variable aléatoire uniforme sur $(1,3)$:

```
V <-runif (150,1,3)
```

Code R :

```
N=100 # La taille de l'echantillon
u=runif(N) # nb aleatoire suivant Unif(0,1)
x=-log(1-u) # transformation des uniformes
x
```

Remarque 2.1 *La fonction runif() permet de générer des séries de nombres aléatoires ; tirés d'une loi uniforme ; on a donc un générateur de nombres pseudo aléatoires «classique ».*

Définition 2.1 *on définit un nombre aléatoire comme la réalisation d'une variable aléatoire distribuée uniformément dans l'intervalle [0, 1]. Une suite de nombres aléatoires sera dans la suite de ce mémoire un échantillon issu d'une population distribuée selon la loi U(0, 1) dont les éléments sont indépendants les uns des autres.*

Définition 2.2 *une suite de nombres pseudo-aléatoires est une suite de nombres possédant les mêmes propriétés qu'une suite de nombres aléatoires, mais générée à travers une procédure déterministe. Les nombres pseudo-aléatoires sont donc générés par des fonctions mathématiques ou par des algorithmes.*

On pourrait ainsi créer un vecteur de nombres au hasard prenant comme valeur 0 ou 1, en écrivant par exemple :

```
> x<-runif(100)
> for (i in 1 : length(x)){
if (x[i]<= 0.5) x[i]=0
else x[i]=1
}
x
[1] 0 0 1 0 1 0 0 1 1 0 1 0 1 0 1 0 1 1 1 0 1 0 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1
[38] 1 0 1 0 1 1 1 0 1 0 1 1 0 1 0 0 1 0 0 1 1 0 0 1 1 1 0 1 0 1 1 0 0 1 0 0 0
[75] 0 1 0 0 0 1 1 1 0 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 1
```

2.2 La méthode de Box et Muller

2.2.1 Introduction

La méthode de Box et Muller (1958) est une autre méthode pour obtenir des réalisations de variables dont la distribution est normale. Cette méthode est fondée sur la transformation des coordonnées polaires en coordonnées cartésiennes.

Théorème 2.2 (Théorème de Box et Muller) *Soient U_1 et U_2 deux variables aléatoires uniformes et indépendantes sur l'intervalle $[0,1]$. Les variables*

$$\begin{aligned} Z_1 &= \sqrt{-2\log(U_1)}\cos(2\pi U_2) \\ Z_2 &= \sqrt{-2\log(U_1)}\sin(2\pi U_2) \end{aligned}$$

sont alors deux variables aléatoires normales centrées réduites indépendantes.

Preuve. La démonstration est une application instructive de ce que nous avons vu à propos des transformations de variables continues. Soient Z_1 et Z_2 deux variables aléatoires indépendantes avec $Z_i \sim N(0,1), i = 1, 2.$:

$$f_{Z_1}(z_1) = \frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}z_1^2), \quad f_{Z_2}(z_2) = \frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}z_2^2).$$

Leur densité conjointe est donnée par :

$$f(z_1, z_2) = \frac{1}{2\pi}\exp(-\frac{1}{2}(z_1^2 + z_2^2)).$$

Dans le plan $\mathbb{R}^2$, le point (z_1, z_2) est situé à une distance $r = \sqrt{z_1^2 + z_2^2}$ de l'origine, et la droite passant par $(0,0)$ et (z_1, z_2) forme une angle θ avec l'axe horizontal, tel que $\tan\theta = \frac{z_2}{z_1}$, le passage des coordonnées cartésiennes (z_1, z_2) aux coordonnées polaires s'effectue par la transformation :

$$z_1 = r\cos\theta, \qquad z_2 = r\sin\theta$$

dont le jacobien J est égal à :

$$J = \begin{vmatrix} \cos\theta & -r\sin\theta \\ \sin\theta & r\cos\theta \end{vmatrix} = r.$$

Ainsi :

$$\begin{aligned} f(r,\theta) &= |r|\, f(z_1,z_2) \\ &= r.\frac{1}{2\pi}\exp(-\frac{1}{2}(z_1^2+z_2^2)) = \frac{1}{2\pi}.r\exp(-\frac{r^2}{2}). \end{aligned}$$

Cela montre que les variables aléatoires $R=\sqrt{Z_1^2+Z_2^2}$ et Θ (la variable aléatoire définie à partir de Z_1 et Z_2 comme la fonction qui associe à Z_1 et Z_2 la seule valeur dans $[0,2\pi[$ dont $\cos\Theta$ a la même signe que Z_1; $\sin\Theta$ a la même signe que Z_2 et $Z_1\sin\Theta = Z_2\cos\Theta$) sont des variables indépendantes dont les densités respectives sont :

$$\begin{aligned} f_R(r) &= r\exp(-\frac{r^2}{2}) \text{ pour } -\infty < r < +\infty, \\ f_\Theta(\theta) &= \frac{1}{2\pi} \text{ pour } 0 \le \theta < 2\pi. \end{aligned}$$

En posant $U=\exp(-R^2/2)$, on a, pour $u>0$:

$$\begin{aligned} \Pr(U &\le u) = \Pr(\exp(-R^2/2) \le u) \\ &= \Pr(-R^2/2 \le \log u) \\ &= \Pr(R \ge \sqrt{-2\log u}) \\ &= \int_{\sqrt{-2\log u}}^{\infty} r\exp(-\frac{r^2}{2})dr \\ &= -\exp(-\frac{r^2}{2})\Big|_{\sqrt{-2\log u}}^{\infty} = \exp(-(\sqrt{-2\log u})^2/2) \\ &= \exp(\log u) = u. \end{aligned}$$

On a ainsi transformé (Z_1,Z_2), deux variables aléatoires normales indépendantes, en une paire de variables uniformes indépendantes. ■

Les transformations inverses permettent donc de simuler deux échantillons indépendants d'une loi normale à partir de deux échantillons indépendants issus d'une loi uniforme.

Remarque 2.2 *Cette méthode a l'avantage de ne pas nécessiter l'inverse de la fonction de répartition de la loi normale centrée réduite* Φ *ni la fonction de répartition elle même.*

Remarque 2.3 *Dans le cas de la simulation d'une loi normale, la méthode de l'inversion est d'application difficile, car l'inverse de* Φ *ne s'explicite pas par des fonctions élémentaires, et la fonction de répartition de la loi normale centrée réduite* Φ *s'exprime seulement à travers des fonctions élémentaires.*

2.2.2 Algorithme de la méthode Box-Muller

– Soit V, U deux v.a. indépendantes uniformes sur $[0, 1]$

– $\Theta = 2\pi U$ (suit une loi uniforme sur $[0, 2\pi]$).

– $R = \sqrt{-2 \log V}$.

– d'où :

$$\begin{aligned} X &= \sqrt{-2\log(V)}\cos(2\pi U), \\ Y &= \sqrt{-2\log(V)}\sin(2\pi U), \end{aligned}$$

est un couple de gaussiennes centrées réduites indépendantes.

Exemple 2.2 *Le code R suivant permet de générer* 100 *observations d'une variable aléatoire gaussienne centrée réduite en utilisant la méthode de Box-Muller :*

```
N = 100
X <- numeric(N)
for(i in 1: N){U<-runif(1)
V<-runif(1)
X[i] = sqrt(-2*log(V))*cos(2*pi*U)}
X
qqnorm(X,xlab="Q.Norm",ylab="Q.Theo",main="QQ-plot")
```

```
qqline(X)
```

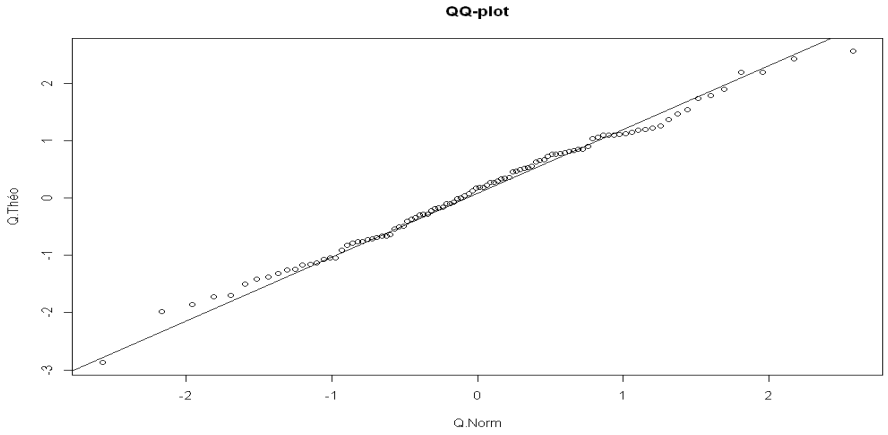


FIG. 2.1 – QQ-plot (méthode de Box-Muller)

2.3 Simulation par la méthode de rejet

La méthode de rejet peut s'utiliser pour la génération de n'importe quel type de variable aléatoire. Elle consiste à générer des données suivant une distribution de fonction de densité proche de celle désirée et d'ensuite éliminer une certaine proportion de ces données de manière à se ramener à des données qui suivent la distribution attendue.

La méthode de rejet est utilisée pour engendrer indirectement une variable aléatoire X, de densité de probabilité f lorsqu'on ne sait pas simuler directement la loi de densité de probabilité f.

Soit (Y, U) un couple de variables aléatoires indépendantes tirées selon une loi uniforme, i.e. (Y, U) est un point tiré uniformément dans le carré unité. On peut alors montrer que la distribution de X est la loi conditionnelle de X sachant l'événement : $M = \{U \leq f_X(Y)\}$. Autrement dit, $f_X(x) = f_Y(x|M)$.

Pour simuler une suite de variables aléatoires réelles $(X_n)_{n\geq 1}$ de distribution identique à celle de X, il suffit donc, dans une suite de tirages de couples (Y_i, U_i) uniformes indépendants, de sélectionner les Y_i correspondant aux tirages vérifiant (Y_i, U_i) et de rejeter les autres.

2.3.1 Algorithme

On voudrait simuler une variable aléatoire réelle X de densité de probabilité f. On suppose qu'il existe une autre densité de probabilité g telle que le ratio $\frac{f}{g}$ soit borné, disons par c (i.e. $f \leq cg$), et qu'on sache simuler Y de densité g.

La version basique de la méthode de rejet prend la forme suivante :

1. Tirer Y de densité g,
2. Tirer U selon la loi uniforme $U(0;1)$, indépendamment de Y,
3. Tant que $U > \dfrac{f(Y)}{cg(Y)}$, reprendre en **1**,
4. Accepter Y comme un tirage aléatoire de densité de probabilité f.

Remarque 2.4 *On remarque que l'algorithme comporte une boucle dont la condition porte sur des variables aléatoires. Le nombre d'itérations, notons-le N est donc lui-même aléatoire. On peut montrer que N suit la loi géométrique de paramètre $\frac{1}{c}$.*

Par suite, l'espérance de N (c-à-d le nombre moyen d'itérations à effectuer avant d'obtenir une réalisation de la densité f) vaut c. On a donc tout intérêt à choisir c le plus petit possible. En pratique, une fois la fonction g choisie, le meilleur choix de c est donc la plus petite constante qui majore le ratio $\frac{f}{g}$, c'est-à-dire :

$$c = \sup \frac{f(x)}{g(x)}.$$

Notons que, soit c est supérieur strict à 1, soit $f \neq g$, la deuxième alternative étant assez théorique : en effet, comme $cg - f \geq 0$. On a donc intérêt à choisir c le plus proche de 1 possible, pour que le nombre d'itérations moyen soit proche de 1 lui aussi. De plus, le choix de g est primordial :

– le tirage de la loi g doit être facile ;

– l'évaluation de $f(x)/g(x)$ doit être aisée ;

– la constante c doit être la plus petite possible ;

– la fonction cg doit majorer la densité f.

2.3.2 Simulation d'une variable normale

pour simuler une variable aléatoire normale centrée réduite X par la méthode du rejet, on choisira $g(y) = e^{-y}$ pour $y > 0$ et $c = \sqrt{e/2\pi} \simeq 0,6577$.

A noter qu'ici $c < 1$. Puisque f, la fonction de densité de X, est symétrique, on simulera seulement des valeurs positives d'une distribution normale, pour ensuite les étaler de manière aléatoire de part et d'autre de l'axe de symétrie. Cette variante de la méthode entraîne seulement $c > 1/2$.

Algorithme :

1)- Générer U selon la loi uniforme $U(0;1)$,

2)- Générer Y de densité g,

3)- Tester si $U \leq f(Y)/cg(Y)$ c'est-à-dire si :

$$U \leq \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} / \sqrt{\frac{e}{2\pi}} \times e^{-y},$$

4)- Si cette condition n'est pas vérifiée retourner à l'étape 1, sinon continuer,

5)- Si $U < 0.5$ mettre un signe positif à Y, sinon le garder négatif.

Une simulation complète est détaillée dans l'exemple 2.3.

Exemple 2.3 *Soit $f(x)$ la densité d'une distribution normale centrée réduite, et $g(x) = e^{-x}$ pour $x > 0$. Une application de la méthode du rejet est utilisée pour simuler des réalisation de la loi normale en accord avec la procédure décrite plus haut.*

Choix de la constante c

La constante c à utiliser doit être telle que l'exponentielle soit toujours plus grande que la courbe normale mais la plus proche possible (jusqu'a la toucher) pour maximiser l'efficacité des tirages.

Pour que la courbe $cg(x)$ soit toujours plus grande que $f(x)$ il faut, dans le cas présent que :

$$c \times e^{-x} \geq \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \forall x \geq 0$$

soit

$$\begin{aligned} c &\geq \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2-2x}{2}} = \frac{\sqrt{e}}{\sqrt{2\pi}} e^{-\frac{x^2-2x+1}{2}} \\ &= \frac{\sqrt{e}}{\sqrt{2\pi}} e^{-\frac{(x-1)^2}{2}} \forall x \geq 0 \end{aligned}$$

Le terme de droite de cette expression prend sa valeur maximum en $x = 1$ et vaut alors

$$\sqrt{\frac{e}{2\pi}}$$

la valeur minimum à donner à c.

Code R pour la méthode de rejet

```
# Simulation de loi normale centrée réduite

N=100

repeat{
```

```
u<-runif(N)
Y<-rexp(N)
I=function(y){(1/sqrt(2*pi))*exp(-y^2/2)}
J=function(y){exp(-y)}
c<-sqrt(exp(1)/2*pi)
if(u<=I(Y)/c*J(Y))
{break}
}
X=numeric(N)
for(i in 1:N){
X[i]=ifelse(u[i]<0.5,Y[i],-Y[i])}
qqnorm(X,xlab="Q.Norm",ylab="Q.Theo",main="QQ-plot")
```

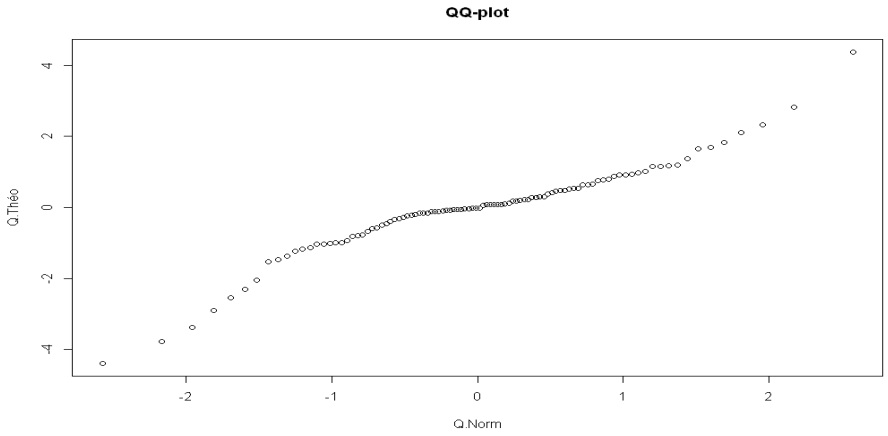


FIG. 2.2 – QQ-plot (méthode de rejet)

2.4 Simulation par méthode de Monte-Carlo

La méthode de Monte Carlo peut être définie comme toute technique numérique de résolution de problèmes, on utilise des nombres aléatoires. On attribue la méthode de Monte Carlo, développée vers 1949, aux mathématiciens américains John von Neumann et Stanislav Ulam. Ce n'est toute fois qu'avec l'avènement des ordinateurs que l'on a pu réellement utiliser cette méthode. Quant au nom de Monte Carlo, on le doit bien sûr à la capitale de la principauté de Monaco. En effet, la roulette est l'un des mécanismes les plus simples pour générer des nombres aléatoires.

L'expression *"Simulation de Monte-Carlo"* recouvre une série de techniques destinées à résoudre des problèmes complexes mais le plus souvent déterministes par l'introduction d'échantillonnages aléatoires. On a recours à une simulation de Monte-Carlo lorsque le problème :

- Est trop complexe pour qu'une résolution par voie purement mathématique soit envisageable.
- Est trop volumineux (en particulier, contient un trop grand nombre de variables) pour que les techniques d'approximation numérique puissent conduire à un résultat précis dans un temps acceptable.

Ce genre de situation est très commun dans tous les domaines ayant recours aux mathématiques appliquées : physique, chimie, biologie, économie, finance, sociologie etc...

A titre d'exemple, nous décrivons ici une des applications les plus simples de simulation de Monte-Carlo : l'intégration d'une fonction dans une région bornée. Comment calculer l'intégrale entre a et b d'une fonction f ? (i.e. $I = \int_a^b f(x)\,dx$).

2.4.1 Intégration unidimensionnelle

Les méthodes de Monte Carlo reposent sur une approximation probabiliste et non déterministe. En ce sens, on ne résout pas l'objet mathématique mais on cherche à l'approcher moyennant la loi forte des grands nombres. Cet objet peut être une intégrale comme c'est le cas dans le présent paragraphe. Ici, nous traiterons les intégrales se présentant sous la forme :

$$I = \int_a^b f(x)\, dx,$$

où : f une fonction intégrable sur $[a, b]$.

La méthode de Monte Carlo pour l'intégration consiste à trouver une variable aléatoire Z telle que : $I = E(Z)$. Ce qui permet d'estimer I en utilisant la loi forte des grands nombres. L'erreur commise est contrôlée par le théorème central limite dès que Z est de carré intégrable.

Si aucune primitive de $f(x)$ n'est connue, l'intégrale ne peut pas être calculée analytiquement. Mais si $f(x)$ peut être facilement calculée en tout point de l'intervalle $[a, b]$, on peut obtenir une bonne approximation de la valeur de cette intégrale par des méthodes numériques.

Il existe de nombreuses méthodes d'intégration numérique. La plus simple consiste à diviser l'intervalle $[a, b]$ par N rectangles adjacents. La hauteur de chaque rectangle est égale à la valeur de $f(x)$ pour x pris au milieu de la base du rectangle. La somme des aires de ces rectangles est une approximation de l'aire sous la courbe représentant $f(x)$, c'est à dire l'intégrale I recherchée :

$$I \simeq \sum_{i=1}^{N} h f(x_i) = h \sum_{i=1}^{N} f(x_i) = \frac{b-a}{N} \sum_{i=1}^{N} f(x_i).$$

Si $f(x)$ a un comportement suffisamment régulier, et si les rectangles sont suffisamment étroits, alors l'aire ainsi calculée sera une bonne approximation de la valeur de l'intégrale.

Proposition 2.1 *On écrit :*

$$I = \int_a^b \frac{g}{f_X}(x) f_X(x)\, dx = E\left(\frac{g(X)}{f_X(X)}\right),$$

où X est une variable aléatoire de densité f_X et on note que $\dfrac{g(X)}{f_X(X)} = Z$ est bien définie car $f_X(X) \neq 0$ presque sûrement. Par ailleurs Z est dans $L^1(a,b)$, donc, d'après la loi forte des grands nombres :

$$\widehat{I}_N = \frac{1}{N}\sum_{i=1}^{N} \frac{g(X_i)}{f_X(X_i)} \overset{P.S.}{\to} I, \quad \textit{quand } N \to \infty.$$

$\widehat{I}_N$ est un estimateur sans biais de I si les X_i sont indépendantes de densité f_X.

Proposition 2.2 *On a*

$$\sqrt{N}\frac{\left(\widehat{I}_N - I\right)}{\sigma} \overset{D}{\to} \mathcal{N}(0,1), \quad \textit{quand } N \to \infty,$$

où $\mathcal{N}(0,1)$ est la loi normale centrée réduite et $\overset{D}{\to}$ désigne la converge en loi. $\sigma^2 < \infty$ est la variance de Z.

Remarque 2.5 *Si F est la fonction de répartition de la loi normale centrée réduite, z_α représente le quantile d'ordre α, i.e tel que $F(z_\alpha) = 1 - \alpha$. De la proposition précédente on peut établir un intervalle de confiance de I :*

$$\lim_{N\to\infty} P\left(\widehat{I}_N - z_\alpha \frac{\widehat{\sigma}_N}{\sqrt{N}} \leq I \leq \widehat{I}_N + z_\alpha \frac{\widehat{\sigma}_N}{\sqrt{N}}\right) = 1 - 2\alpha,$$

$\widehat{\sigma}_N^2$ un estimateur sans biais de la variance de Z :

$$\widehat{\sigma}_N^2 = \frac{1}{N-1}\sum_{i=1}^{N}\left(\frac{g(X_i)}{f_X(X_i)} - \widehat{I}_N\right)^2.$$

2.4.2 Intégrale multiple

Que se passe-t-il si nous utilisons le même approche dans le cas d'une intégrale multiple ?
Nous rencontrons deux difficultés :

1. D'abord, la région d'intégration n'est plus définie par une paire de nombres comme précédemment, mais pas une hyper-surface fermée dont la forme peut être très compliquée même pour des problèmes simples, et impossible à décrire analytiquement.
2. La deuxième difficulté est encore plus grave. Revenons un instant à l'intégrale simple, et supposons que nous ayons décidé d'utiliser 500 rectangles. Puis envisageons une intégrale multiple à 500 variables, et décidons de conserver sur chaque axe la même résolution que dans le cas de l'intégrale simple. Nous devrons alors définir 500×500 hyper-rectangles, un nombre au-delà des capacités des ordinateurs les plus rapides.

Cette difficulté est absolument universelle, et se retrouve dans toute technique locale, qu'elle soit déterministe ou probabiliste. Notez que le nombre de rectangles est choisi essentiellement sur la base considérations portant sur la rapidité avec laquelle la fonction varie dans la région d'intégration. Il n'est donc pas possible de réduire arbitrairement le nombre de rectangles jusqu'à une valeur compatible avec des temps de calcul raisonnables, sous peine de perdre tellement d'information sur la fonction que l'approximation obtenue devienne grossièrement fausse.

2.4.3 Intégration par simulation de Monte-Carlo

C'est alors qu'intervient une idée à la fois simple et efficace. Nous avons remarqué qu'une intégrale n'est, à peu de chose près, que la somme des valeurs de la fonctions prises sur des points régulièrement répartis dans la région d'intégration. Dans cette phrase, remplaçons "**régulièrement répartis**" par "**répartis selon une distribution de probabilité uniforme**", et nous avons notre première simulation de Monte-Carlo.

Donc, dans sa version la plus simple, le calcul d'une intégrale sous la forme :

$$I = \iint\int\cdots\int_{région} f(x_1, ..., x_n)\, dx_1 ... dx_n,$$

par simulation de Monte-Carlo procède ainsi :

1. Se servir d'une distribution uniforme à n dimensions pour tirer N points dans la région d'intégration (dont nous notons le volume V).
2. Additionner tous les $f(p_i)$, multiplier le résultat par V/N.
3. Le résultat est une estimation de la valeur de l'intégrale I :

$$I = \frac{V}{N}\sum_{i=1}^{N} f(p_i).$$

Remarque 2.6 *L'intérêt principal de la l'intégration par simulation de Monte-Carlo réside dans son comportement en grande dimension. Cette question est difficile mais la réponse est simple. Considérons une méthode quelconque d'intégration numérique déterministe reposant sur le calcul de valeurs de la fonction sur les nœuds d'une grille dans la région d'intégration. Si nous comparons les performances de cette méthode et de celle de la méthode de Monte-Carlo pour des dimensions de plus en plus grandes, il se trouvera une dimension d au-delà de laquelle la méthode de Monte-Carlo sera plus efficace que la méthode déterministe pour un nombre N donné de tirages. Ceci veut-dire que les valeurs de l'intégrale trouvées par la méthode de Monte-Carlo seront, le plus souvent, plus proches de la valeur vraie de l'intégrale que celle trouvée par l'approximation déterministe.*

2.4.4 Exemple de calcul d'intégral par la méthode de Monte Carlo

Pour calculer l'intégral :

$$I = \int_2^{12} \exp(-(x^2 + x)/3) \sin(x) dx,$$

on note

$$\Psi(x) = \exp(-(x^2 + x)/3) \sin(x).$$

Pour l'écriture de la fonction, on utilise le syntax "function" du langage R comme suit :

```
le nom de la fonction<- function(liste des arguments) {le corps de la fonction}
```

1) Le code de R ci-dessous :

L'écriture de la fonction : $\Psi(x)$.

```
psi=function(x){exp(-(x^2+x)/3)*sin(x)}
# Est ce qu'elle fonctionne?
psi(0); psi(-3); psi(pi/2)
```

2) Exécution dans le console R :

```
> psi=function(x){exp(-(x^2+x)/3)*sin(x)}
> # Est ce qu'elle fonctionne?
> psi(0); psi(-3);psi(pi/2)
[1] 0
[1] -0.01909852
[1] 0.2602622
```

– Il faut charger le package qui contient le syntaxe utilisé et ça par deux méthodes. La 1^{ere} : on clique sur "packages" ensuite "charger package" puis sélectionne le nom de

package puis clique sur "ok". La 2^{eme} : est plus simple, on écris dans le console R : Library (le nom de package).

– Utilisation de syntaxe "integrate" : Il faut charger le package "stats" pour utiliser le syntaxe "integrate". On utilise le syntaxe "integrate" du langage R sous forme :

```
le nom d'integrale <- integrate(le nom de la fonction,
la borne inferieure d'integrale, la borne superieure d'integrale)
```

Le programme R

```
### Simulation par methode de Monte-Carlo
## Integration par simulation de Monte-Carlo
psi=function(x){exp(-(x^2+x)/3)*sin(x)}
library(stats) # charger le package "stats"
I=integrate(psi,2,12)$v
pis2=function(x){(exp(-(x^2+x)/3)*sin(x))^2}
J=integrate(pis2,2,12)$v
var=10*J-I^2
## Loi Forte des Gr Nbr
N=1000
In=numeric(N)
Zn=numeric(N)
for(k in 1:N){
v=runif(k,2,12)
In[k]=10*mean(psi(v))
Zn[k]=(In[k]-I)/sqrt(var/k)
}
N=1:1000
plot(N,In,type='b',xlab="N",ylab=expression(I_N),main="L.F.G.Nbrs")
```

```
abline(h=I,lwd=3,col=2)
## T C L
for(j in 1:N){}
x11()
qqnorm(Zn,xlab="Q.Norm",ylab="Q.Theo",main="QQ-plot")
qqline(Zn)
x11()
hist(Zn,50,prob=TRUE,xlab="x",ylab="f(x)",main="histogramme ")
x=rnorm(length(Zn))
lines(density(x),col=4,lwd=2)
lines(density(Zn),col=2,lwd=2,lty=2)
legend(1,.4,c("densite Theorique","densite Empirique"),
col=c(4,2), lty=c(1,2), lwd=c(2,2), bty = "n",cex=1.3)
```

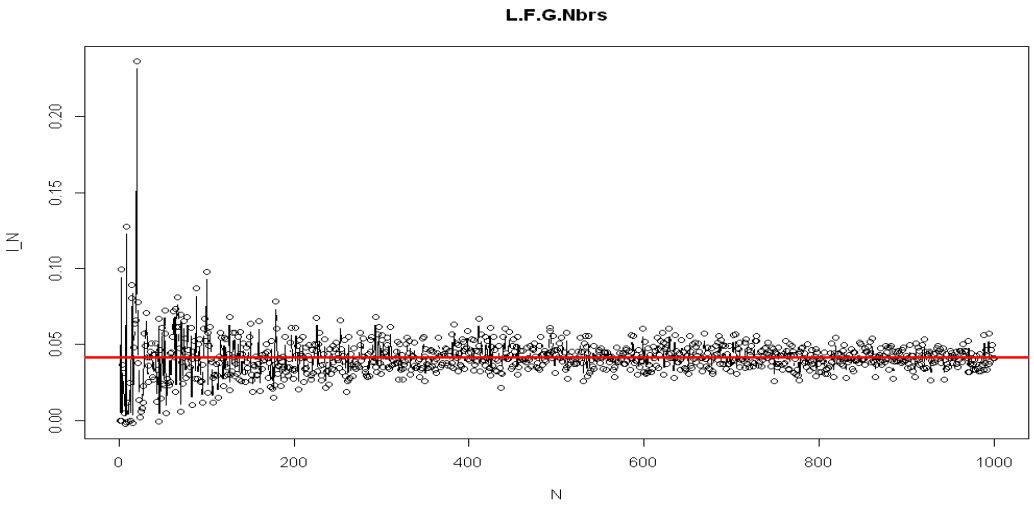


FIG. 2.3 – Convergence de l'intégrale simulé par la méthode de Monte-Carlo.

Remarque 2.7 *Le syntaxe " integrate" qui donne la valeur de I avec l'erreur pour ça on ajoute $v (obtenue sa valeur sans erreur).*

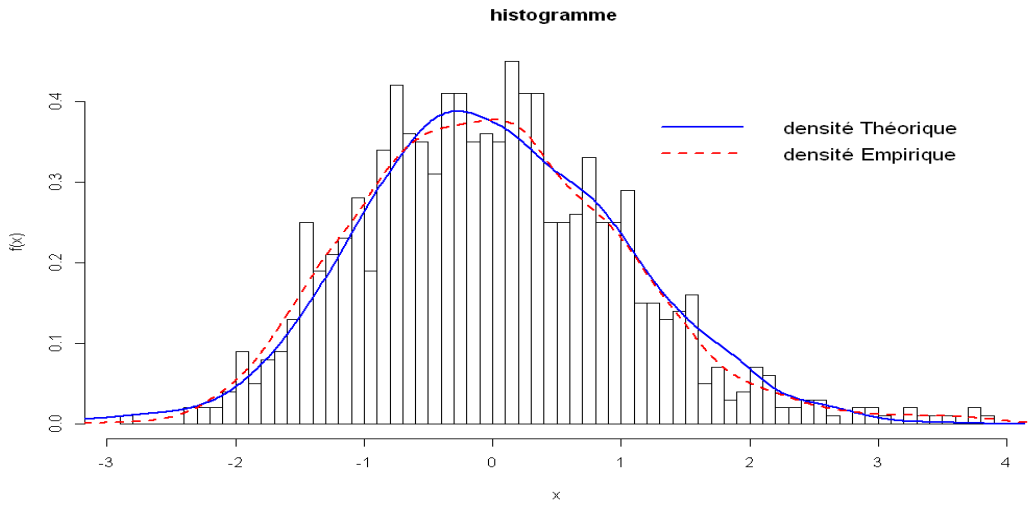


FIG. 2.4 – Histogramme, densité théorique et densité empirique (méthode de Monte Carlo)

2.5 Simulation par la méthode de comparaison

La méthode décrite dans ce paragraphe et que l'on appelle méthode de comparaison permet de simuler des variables aléatoires discrètes à valeurs dans N ou un sous-ensemble de N. Cette méthode est fondée sur la définition des variables aléatoires uniformes.

Considérons une variable U uniforme sur $[0,1]$; on a

$$\Pr(a < U \leq b) = b - a, \quad pour \quad 0 \leq a < b \leq 1.$$

Pour simuler une variable aléatoire discrète X satisfaisant $p_i = \Pr(X = i)$, on utilise le fait que

$$\Pr(X = 0) = p_0 = \Pr(0 < U \leq p_0)$$

et

$$\Pr(X = i) = p_i = \Pr\left(\sum_{j=0}^{i-1} p_j < U \leq \sum_{j=0}^{i} p_j\right).$$

La méthode de comparaison est définie de la manière suivante à l'aide d'un échantillon

issu d'une variable aléatoire uniforme $U(0,1)$:

$$\begin{aligned} X &= 0 \text{ si } 0 < U \leq p_0 \\ X &= i \text{ si } \sum_{j=0}^{i-1} p_j < U \leq \sum_{j=0}^{i} p_j. \end{aligned}$$

On compare donc la valeur de U avec la loi cumulative de X. Remarquons que i peut varier dans un ensemble fini ($i = 1, 2, ..., n$) ou dans un ensemble infini ($i \in \mathbb{N}$).

Exemple 2.4 (Loi de bernoulli) *Considérons une variable aléatoire de bernoulli de paramètre p. Puisque* $\Pr(X = 0) = 1 - p$ *et* $\Pr(X = 1) = p$, *la méthode de comparaison donne dans ce cas :*

$$\begin{aligned} X &= 0 \ si \ 0 \leq U \leq 1 - p \\ X &= 1 \ si \ 1 - p < U \leq 1. \end{aligned}$$

Le jet d'une pièce de monnaie, lancée 10 fois, est simulé dans le programme ci-dessous en posant $X = 0$ *(pile) et* $X = 1$ *(face) avec* $p = \frac{1}{2}$.

L'exécution en R

```
# Simulation par la méthode de comparaison la loi de Bernoulli
# Simulation du lancer d'une pièce de monnaie.
> N=10
> U<-runif(N)
> X<-numeric(N)
> p<-1/2 # le parametre de Bernoulli
> for(i in 1:N){
+ U<-runif(1)
```

```
+ if(0<=U & U<=1-p)
+ X[i]=0
+ else
+ X[i]=1}
> X
[1] 1 0 1 0 1 0 1 1 1 0
```

Chapitre 3

Statistique non paramétrique et simulation

Dans le troisième chapitre, je vais étudié quelques estimateurs non paramétrique tels que la fonction de répartition empirique, la fonction des quantiles empiriques et la densité non paramétrique. Une étude de simulation portant sur ces statistiques est donnée.

3.1 Fonction de répartition empirique

Soit X une v.a. de loi F, avec $F(x) = \mathbb{P}\{X \leq x\}$ la fonction de répartition de X. Soit $(X_1, X_2, ..., X_n)$ un échantillon de X et soient $X_{(1)} \leq X_{(2)} \leq ..... \leq X_{(n)}$ l'échantillon ordonné. Supposons que F soit complètement inconnue (non paramétrique), un estimateur pour F est la fonction de répartition empirique, notée F_n qui est définie par :

$$\begin{aligned} F_n(x) &= \tfrac{1}{n}\#\ \{i =: X_i \leq x\} \\ &= \tfrac{1}{n}\sum_{i=1}^{n} 1_{\{X_i \leq x\}} \\ &= \begin{cases} 0 & si \quad x < X_{(1)} \\ \frac{k}{n} & si \quad X_{(k)} \leq x \leq X_{(k+1)} \\ 1 & si \quad x \geq X_{(n)} \end{cases} \end{aligned}$$

1)- $F_n(x)$ est un estimateur sans biais de $F(x)$:

$$\begin{aligned}\mathbb{E}[F_n(x)] &= \tfrac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[1_{\{X_i\leq x\}}\right]\\ &= \mathbb{P}\{X\leq x\}\\ &= F(x)\end{aligned}$$

2)- On a pour tout x, la variance de l'estimateur $F_n(x)$ est donnée par :

$$var[F_n(x)] = F(x)(1-F(x))$$

3)- Le théorème central-limite donne :

$$\begin{aligned}&\frac{nF_n(x)-nF(x)}{\sqrt{nF(x)(1-F(x))}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1)\\ \Rightarrow\ &\sqrt{n}(F_n(x)-F(x)) \xrightarrow{\mathcal{L}} \mathcal{N}(0,F(x)(1-F(x)))\end{aligned}$$

4)- la loi des grands nombres donne :

$$\forall x\in\mathbb{R} : F_n(x) \xrightarrow{\mathbb{P}} F(x), \quad si\ n\longrightarrow\infty$$

5)- Théorème de Glivenko-Cantelli fournit la convergence presque sûre uniforme de la suite des fonctions de répartition empiriques :

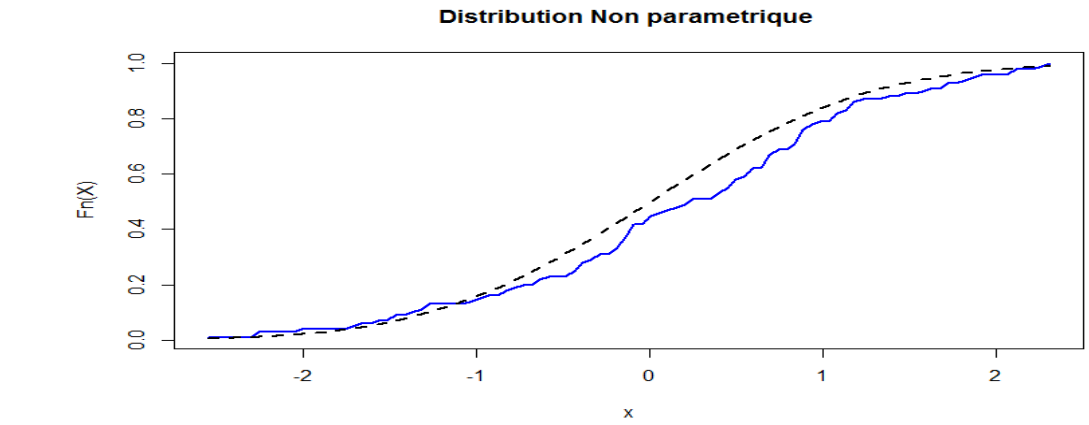
$$\sup_{x\in\mathrm{IR}}|F_n(x)-F(x)| \underset{n\to\infty}{\longrightarrow} 0 \quad p.s$$

Exemple 3.1 *Le programme R suivant permet de simuler la distribution empirique pour un échantillon de taille* 100, *issu d'une v.a. de loi* $N(0,1)$, *dont la distribution théorique est :*

$$F(x) = (2\pi)^{-1/2}\int_{-\infty}^{x}\exp\left(-t^2/2\right)dt, \quad x\in\mathbb{R}.$$

```
N=100 # taille de l'echantillon
X=rnorm(N);Y=numeric(N);V<-numeric(N)
FN=function(x){for(i in 1:N)
{if(X[i]<=x);Y[i]=1;else;Y[i]=0}
mean(Y)}
a=min(X);b=max(X);x=seq(a,b,length=N)
for(i in 1:N){
V[i]<-FN(x[i])}
plot(x,V,xlab="x",ylab="Fn(X)",
main="Distribution Non parametrique ",type='l',col=4, lwd= 2)
lines(x,pnorm(x), lty=2,lwd=2)
```

emp



$_1$1.*pdf*

FIG. 3.1 – Simulation d'une distribution empirique

Dans ce graphe la ligne discontinue c'est la distribution théorique $F(x)$ et la ligne continue c'est la distribution empirique $F_n(x)$.

3.2 Les quantiles empiriques

Le quantile d'ordre p de la population noté $Q(p)$ est l'inverse généralisée de la distribution F. Il est définie pour tout $p \in (0,1)$, par

$$Q(p) = F^{-1}(p) = \inf\{x \in \mathbb{R};\ F(x) \geq p\}$$

qui peut être estimer par

$$Q_n(p) = \inf\{x : F_n(x) \geq p\},$$

le $p^{ème}$ quantile de la fonction de répartition empirique.

Si U est une v.a. de loi uniforme sur $[0,1]$ alors $F^{-1}(U)$ est une v.a. de fonction de répartition F. Réciproquement, si F est continue, $F(X)$ suit la loi uniforme sur $[0,1]$. Ceci est faux si F n'est pas continue.

Définition 3.1 *Un quantile d'ordre p est le réel x_p tel que*

$$\mathbb{P}(X \leq x_p) \geq p \quad et \quad \mathbb{P}(X \geq x_p) \geq 1-p.$$

$x_p := F^{-1}(p)$.

On suppose dans toute la suite que F est continue. Soit $X_{(1)}$ la plus petite valeur de l'´echantillon. On note

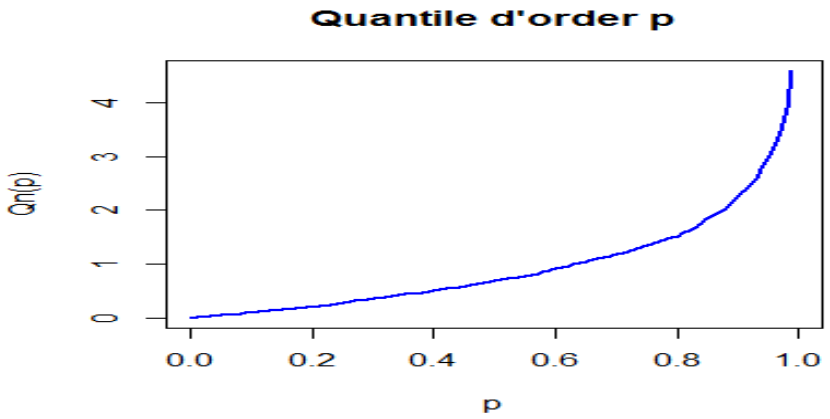
$$Q_{p,n} = X_{([np]+1,n)}$$

le quantile empirique d'ordre p, où $[np]$ désigne la partie entière de np.

Exemple 3.2 *# Dans le programme qui suit, nous donnons la simulation du quantile empirique d'ordre p d'une loi exponentielle de paramètre 1 :*

```
N=1000
X=rexp(N)
p=0.5
Y=sort(X)
QE=function(p){m=floor(p*N)
s=m+1
Y[s]}
plot(QE,xlab="p", ylab="Qn(p)",
main="Quantile d'order p",type='l',col=4, lwd= 2)
```

emp



12.*pdf*

FIG. 3.2 – Simulation du p-ème quantile d'une loi exponentielle

Pour tout $p \in]0,1[$, si F possède un unique quantile d'ordre p, qui est alors égal à x_p (c'est-à-dire que F^{-1} est continue en p), alors

$$Q_n(p) \underset{n\to\infty}{\overset{P.S.}{\to}} x_p.$$

Autrement, on définie $Q_n(p)$ par la façon suivante :

Définition 3.2 (Fonction de répartition empirique inverse) *Soit* $(X_1, ..., X_n)$ *un échan-*

tillon aléatoire issu de X. La fonction de répartition empirique est :

$$F_n(x) = \begin{cases} 0 \quad si\ x < X_{1,n}, \\ \frac{i-1}{n} \quad si\ X_{i-1,n} \leq x < X_{i,n}, \\ 1\ si \quad x \geq X_{n,n}. \end{cases}$$

La fonction quantile empirique est l'application $F_n^{-1} :]0,1[\to \mathbb{R}$ définie par :

$$F_n^{-1}(s) = \inf\{t, F_n(t) \geq s\}.$$

On obtient :

$$F_n^{-1}(s) = X_{i,n}\ si\ \frac{i-1}{n} < s \leq \frac{i}{n}.$$

Exemple 3.3 *Le programme suivant permet d'obtenir se qu'on appel, le QQplot (quantile vs quantile) qui est utiliser souvent dans les comparaisons et les tests entre deux lois :*

```
N=1000;QE=numeric(N);V=numeric(N)
X=rexp(N);Y=sort(X)
QE=function(s){for(i in 1:N)
{if(s<=i/N & s>(i-1)/N)
V[i]=Y[i]
else
V[i]=0}
sum(V)}
h=seq(0,0.9,length=N)
for(i in 1:N){QE[i]=QE(h[i])}
QT=-log(1-h)
plot(QT,QE, xlab="Q-theo", ylab="Q-emp",
col=4,main="QQ-plot")
```

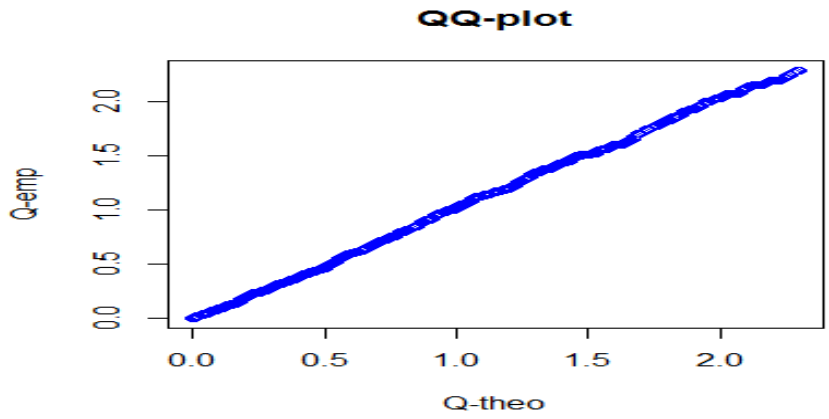


FIG. 3.3 – Quantile théorique vs quantile empirique d'une loi exponentielle

3.3 Densité non paramétrique

Comment estimer non-pramétriquement la densité de probabilité f, en se basant sur les observations $x_1, x_2, ..., x_n$? Il existe plusieurs méthodes d'estimation non paramétrique d'une densité. La méthode la plus simple est celle du noyau qui est basée sur deux conceptes : le noyau K (une densité de probablité) et un paramètre de lissage h ($h = h_n \to 0$ quand $n \to \infty$).

Rappelons que la densité de probabilité f est égale à la dérivée de la fonction de répartition F(si cette dérivée existe). On peut donc écrire

$$f(x) = \lim_{h \to 0} \left[\frac{F(x+h) - F(x-h)}{2h} \right]$$

En remplaçant alors F par F_n, on obtient

$$f_n(x) = \frac{F_n(x+h) - F_n(x-h)}{2h}$$

L'estimateur $f_n(\cdot)$ de $f(\cdot)$ est appelée estimateur de Rosenblatt (1956). Il peut aussi

s'écrire sous la forme

$$\begin{aligned} f_n(x) &= \frac{1}{2h}\frac{\#\{i: x-h<X_i\leq x+h\}}{n} \\ &= \frac{1}{2nh}\sum_{i=1}^{n}\mathbf{1}_{\{x-h<X_i\leq x+h\}} \\ &= \frac{1}{2nh}\sum_{i=1}^{n}\mathbf{1}_{\left\{-1<\frac{X_i-x}{h}\leq 1\right\}} \end{aligned}$$

Notons que cet estimateur peut encore s'écrire comme

$$f_n(x) = \frac{1}{nh}\sum_{i=1}^{n} K_0\left(\frac{X_i - x}{h}\right)$$

où

$$K_0(u) = \mathbf{1}_{]-1,1]}(u)/2 = \begin{cases} 1/2 & \text{si } u\in]-1,1], \\ 0 & \text{si non.} \end{cases}$$

est le noyau de Rosenblatt (1956).

L'estimateur de Rosenblatt peut être généralisé en remplaçant la fonction de poids $K_0(.)$ (la densité de probabilité uniforme) par une fonction de poids plus générale K (par exemple une densité de probabilité quelconque).

Définition 3.3 (*L'estimateur à noyau*) *Étant donné un n-échantillon $X_1, X_2, ..., X_n$ de variables aléatoires indépendantes et de même densité f inconnue. Considérons l'estimateur à noyau de Parzen-Rosenblatt de la densité f donné par :*

$$f_n(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{X_i - x}{h}\right) \tag{1}$$

où h est le paramètre de lissage et K une fonction densité.

Pour estimer f, il faut choisir le noyau K et le paramètre h. Si le choix du noyau n'est pas un problème, il n'est pas de même pour le choix de la largeur de la fenêtre h qui dépend essentiellement de la taille n de l'échantillon. En effet :

– Le biais décroît si h diminue mais la variance augmente.

– La variance diminue si h augmente mais le biais augmente.

– Pourque la variance tend vers zéro, il faut que $nh \to \infty$.

– Plus la courbure de la densité est haute en x, plus le biais est grand.

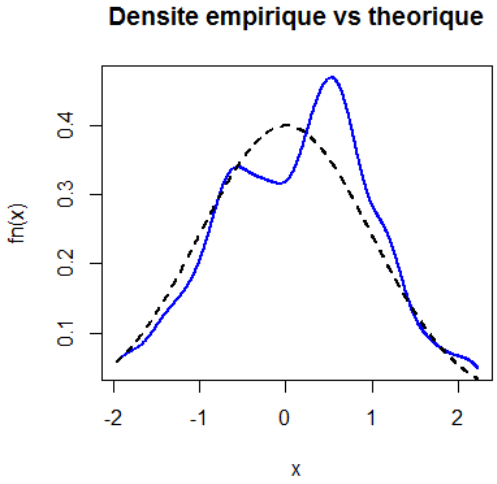
– La variance est plus grande pour des valeurs plus grande de la densité.

Exemple 3.4 *Dans ce dernier exemple, nous donnons la simulation de l'estimateur à noyau d'une densité de loi normale* $N(0,1)$:

```
n=100
X=rnorm(n) # echantillon
# Noyau Gaussien
K=function(t){(1/sqrt(2*pi))*exp(-0.5*t^2)}
h=0.2 # parametre de lissage h
s=1000
a=min(X)
b=max(X)
x=seq(a,b,length=s) # Intervalle [a,b]
w=numeric(n)
fn=numeric(s)
for(j in 1 :s){
for(i in 1 :n){D<-(x[j]-X[i])
w[i]=K(D/h) }
fn[j]=sum(w)/(n*h)}
plot(x,fn,xlab="x", ylab="fn(x)",
main="Densite empirique vs theorique",type='l',col=4, lwd= 2)
lines(x,dnorm(x),lwd= 2,lty=2)
```

Dans ce dernier graphe, la ligne discontinue c'est la densité théorique $f(x)$ d'une loi

emp



$_1$4.*pdf*

FIG. 3.4 – Densités théorique et empirique.

normale centrée réduite et la ligne continue c'est la densité à noyau $f_n(x)$. Ici, le noyau K est une densité normale et la le paramètre de lissage est $h = 0.2$ ($h \simeq n^{-1/5}$).

Conclusion

La simulation a un grand impact dans le monde d'aujourd'hui. Le développement technologique dans les domaines les plus variés demande souvent des simulations à grande échelle qui se révèlent essentielles pour la conception de projets ou la mise en place de stratégies d'action. C'est pourquoi elle est enseignée dans les plus prestigieuses universités et écoles polytechniques de la planète.

Les méthodes de simulation, conçues pour être utilisées en statistique et en recherche opérationnelle, ont connu et connaissent encore un développement rapide dû l'extraordinaire évolution des ordinateurs. Des applications se rencontrent tant dans l'industrie qu'en économie, ou encore en sciences sociales, en physique des particules, en astronomie et dans de nombreux autres domaines.

Je termine ce mémoire et cette conclusion, on répandant à la question suivante :

Pourquoi faire des simulations ?

- *Pour « vérifier » à l'aide de l'ordinateur un résultat mathématique déjà connu.*
- *Pour « démontrer » à l'aide de l'ordinateur un résultat que l'on ne parvient pas à démontrer mathématiquement.*
- *Cela peut nous guider dans la démonstration d'un résultat mathématique difficile.*
- *C'est souvent exigé lorsque l'on essaye de publier un papier dans une revue.*

Bibliographie

[1] Bertrand F., Bertrand M.M. (2010). *Initiation à la Statistique avec R*, Dunod, Paris.

[2] Caflisch R. E. (1998). *Monte Carlo and quasi-Monte Carlo methods*, *Acta Numerica* vol. 7, Cambridge University Press, pp. 1-49.

[3] Devroye L. (1986). *Non-Uniform Random Variate Generation*. New York : Springer-Verlage.

[4] Dodge Y., Melfi G. (2008). *Premiers pas en simulation*. Springer-Verlag France.

[5] Elie L., Lapeyre L. (2001). *Introduction aux Méthodes de Monte-Carlo.*
http ://cermics.enpc.fr/~bl/PS/SIMULATION-X/poly-monte-carlo-x.pdf

[6] Goga C, Labruère C. (2009). Introduction au logiciel R, Ecole Doctorale Dijon.
http ://math.u-bourgogne.fr/IMB/goga/formation_doc_R_2009.pdf

[7] Huillet J. (2002). Initiation à l'envirennement R.
http ://cict.fr/~stpierre/doc-R.pdf

[8] Ihaka R., Gentleman R. (1996). R : A Language for Data Analysis and Graphics. *Journal of Computational and Graphical Statistics* **5** : $299 - 314$.

[9] Lafaye de Micheaux P., Drouilhet R., Liquet B. (2011). *Le logiciel R : Maîtriser le langage, Effectuer des analyses statistiques*. Springer-Verlag, France.

[10] Lalanne C. Palliery C. (2011). Introduction aux Statistiques et a l'utilisation du logiciel R.
http ://www.pallier.org/ressources/tp.stats/tp1/tp1.pdf

[11] Legendre P. (2012). Introduction au langage statistique R.
ht*tp ://biol09.biol.umontreal.ca/BIO6077/Introduction_R.pdf*

[12] Michel J. (2006). Initiation au logiciel R.
http ://www.ceremade.dauphine.fr/~xian/Noise/R.pdf

[13] Ounissi H. (2010). Tests Non Paramétriques sous logiciel R . Mémoire d'Ingéniorat en Statistique. Université de Biskra.

[14] Parzen E. (1962). On estimation of a probability density function and mode, Ann. Math. Stat. 33, pp. 1065-1076.

[15] Robert, C.P., Casella G. (2004) *Monte Carlo Statistical Methods* ($2^{ème}$ edition). New York : Springer-Verlag.

[16] Silverman B.W. (1986). Density Estimation. London : Chapman and Hall.

[17] Wasserman, L. (2005). All of Statistics : A Concise Course in Statistical Inference, Springer Texts in Statistics.

Annexe : Notations et Abréviations

$v.a.$: Variable aléatoire.

$\mathcal{N}(0,1)$: Loi normale centrée et réduite.

$U(0,1)$: Loi uniforme sur l'interval $[0,1]$.

$\mathbb{E}(.)$: Espérence mathématique.

$var[.]$: Variance.

$[np]$: La partie entière de np.

$F(x)$: Fonction de répartition.

$\mathbb{P}(X=i)$: Loi d'une variable aléatoire discrète X.

$F^{-1}(x)$: Fonction de répartition inverse.

f_X : Densité de probabilité.

$f_Y(x|M)$: Loi conditionnelle de X sachant l'événement M.

$f(z_1,z_2)$: Densité de probabilité conjointe.

Φ : Fonction de répartition de la loi normale centrée et réduite.

$F_n(.)$: Fonction de répartition empirique.

$F_n^{-1}(.)$: Fonction quantile empirique.

$Q(p)$: Quantile d'order p.

$f_n(x)$: Densité non paramétrique.

$\mathbf{1}_A$: Fonction indicatrice.

$\mathbb{N}$: Ensemble des entiers naturels.

$\mathbb{R}$: Ensemble des nombres réels.

$L^1(A)$: Espace des fonctions intégrables sur A.

$\overset{P.S.}{\rightarrow}$: Convergence presque sûre.

$\overset{D}{\rightarrow}$: Converge en distribution.

$\xrightarrow{\mathcal{L}}$: Convergence en loi.

$\xrightarrow{\mathbb{P}}$: Convergence en probabilité.

$1-\alpha$: Niveau de confiance.

α : Risque ou seuil.

z_α : Quantile d'ordre α.

h : Paramètre de lissage.

K : Noyau ou une fonction densité.