Université de Montréal

Le progiciel PoweR : un outil de recherche reproductible pour faciliter les calculs de puissance de certains tests d'hypothèses au moyen de simulations de Monte Carlo

par

Viet Anh Tran

Département de mathématiques et de statistique

Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures

en vue de l'obtention du grade de

Maître ès sciences (M.Sc.)
en Statistiques

juin 2013

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé

Le progiciel PoweR : un outil de recherche reproductible pour faciliter les calculs de puissance de certains tests d'hypothèses au moyen de simulations de Monte Carlo

présenté par

Viet Anh Tran

a été évalué par un jury composé des personnes suivantes :

Alejandro Murua

(président-rapporteur)

Pierre Lafaye de Micheaux

(directeur de recherche)

Jean-François Angers

(membre du jury)

Mémoire accepté le:

20 juin 2013

TABLE DES MATIÈRES

Liste des figures xi

Liste des tableaux xiii

Remerciements 1

Introduction 3

Chapitre 1. Recherche reproductible (avec R & LaTeX) 5

1.1. Introduction 5

1.2. Approche classique pour générer des rapports statistiques 7

1.3. Problèmes rencontrés avec l'approche classique 8

1.4. Une approche alternative 8

1.5. Recherche reproductible avec Sweave 9

1.6. **PoweR** - un outil de recherche reproductible 10

Chapitre 2. Tests d'hypothèses 13

2.1. Rappels théoriques 13

2.2. Tests d'adéquation 14

2.3. Méthode de Monte Carlo 15

Chapitre 3. PoweR : Buts et Fonctionnalités 19

3.1. Problématique 19

3.2. Matériels essentiels du progiciel **PoweR** 22

3.2.1. Liste des lois de probabilité 22
3.2.2. Liste des tests d'adéquation présents dans le progiciel 28
3.2.2.1. Tests de normalité 28
3.2.2.2. Tests pour une distribution de Laplace 30
3.2.2.3. Tests d'uniformité 31

3.3. Fonctionnalités du progiciel **PoweR** 32
3.3.1. Génération de données 34
3.3.2. Calcul de la statistique d'un test 35
3.3.3. Calcul des quantiles empiriques 38
3.3.4. Calculs de puissances de tests 41

3.4. Informations utiles concernant l'utilisation du progiciel **PoweR** 46
3.4.1. Liste des fonctions dans le progiciel **PoweR** 46
3.4.2. Procédure pour rajouter une nouvelle loi ou un nouveau test dans le progiciel **PoweR** 47

Chapitre 4. Applications 51

4.1. Reproduire les résultats d'articles publiés 51

4.2. Détecter des erreurs dans les résultats publiés 57
4.2.1. Erreurs dues au manque de puissance des outils informatiques 57
4.2.2. Erreur dans les équations ou dans la programmation 62

4.3. Proposer de nouvelles comparaisons 66

4.4. Robustesse du test t de Student à la non-normalité 72

Conclusion 77

Bibliographie 79

Annexe A. Exemple de recherche reproductible avec Sweave . . . . . A-i

Annexe B. Débogage de fonctions . . . . . B-i

B.1. Erreur dans un code R . . . . . B-i

B.2. Erreur dans un code C/C++ . . . . . B-i

Annexe C. Reproduire certains résultats pour des tests de normalité dans des articles publiés . . . . . C-i

C.1. Likelihood-ratio tests for normality - Zhang and Wu (2005) . . . . . C-i

C.2. A test for normality of observations and regression residuals - Jarque and Bera (1987) . . . . . C-ii

C.3. A robust modification of the Jarque-Bera test of normality - Gel and Gastwirth (2008) . . . . . C-iii

C.4. An analysis of variance test for normality : Complete samples - Shapiro and Wilk (1965) . . . . . C-iv

C.5. A modification of the test of Shapiro and Wilk for normality - Rahman and Govindarajulu (1997) . . . . . C-v

C.6. The probability plot correlation coefficient test for normality - Filliben (1975) . . . . . C-vi

C.7. A goodness-of-fit test for normality based on polynomial regression - Coin (2008) . . . . . C-vii

C.8. A test of normality based on empirical characteristic function - Epps and Pulley (1983) . . . . . C-ix

C.9. A test for departure from normality based on a biweight estimator of scale - Martinez and Iglewicz (1981) . . . . . C-x

C.10. Robust directed tests of normality against heavy-tailed alternatives - Gel et al. (2007) . . . . . C-xii

C.11. A test for Normality against symetric alternatives - Spiegelhalter (1977) . . . . . C-xii

Annexe D. Reproduire certains résultats pour des tests pour une distribution de Laplace dans des articles publiés . . . . . D-i

D.1. Modified goodness-of-fit test for the Laplace distribution - Yen and Moore (1988) . . . . . D-i

D.2. Tests of fit for the Laplace distribution, with applications - Puig and Stephens (2000) . . . . . D-ii

D.3. A Class of Omnibus Tests for the Laplace Distribution Based on the Empirical Characteristic Function - Meintanis (2004) . . . . . D-iv

D.4. Testing goodness-of-fit for Laplace distribution based on maximum entropy - Choi and Kim (2006) . . . . . D-vi

D.5. Tests for the Goodness-of-fit of the Laplace distribution - Chen (2002) . D-x

D.6. Goodness of fit test for the rayleigh and the Laplace distributions - Gulati (2011) . . . . . D-xi

D.7. Test of fit for a Laplace distribution against heavier tailed alternatives - Gel (2010) . . . . . D-xii

Annexe E. Reproduire certains résultats pour des tests d'uniformité dans des articles publiés . . . . . E-i

E.1. Test for serial correlation in Regression Analysis based on the periodogram of least-squares residuals - Durbin (1969) . . . . . E-i

E.2. Some new goodness-of-fit tests using order statistics - Hegazy and Green (1975) . . . . . E-i

E.3. A test for uniformity based on informational energy - Pardo (2003) . . . . E-ii

E.4. Powerful goodness-of-fit tests based on the likelihood ratio - Zhang (2002) . . . . . E-iii

LISTE DES FIGURES

1.1 Diagramme du processus Sweave 10

3.1 Diagramme UML du progiciel PoweR 21

3.2 Interface graphique du progiciel PoweR. 33

3.3 Fenêtre d'aide via le bouton ?. 33

3.4 Génération de données d'une loi $\mathtt{Laplace}(0,1)$. 35

3.5 Calcul de la statistique du test de Shapiro-Wilk pour des données suivant la loi $\mathtt{Normal}(0,1)$. 37

3.6 Calcul des valeurs critiques des tests d'uniformité de Quesenberry-Miller et de Pardo. 40

3.7 Calcul des puissances des tests pour une distribution de Laplace W^2, U^2, A^2, $\sqrt{n}D$ et V contre quelques alternatives. 44

A.1 Document final créé par LaTeX : toto.pdf. . . . . A-vi

LISTE DES TABLEAUX

3.1 Lois de probabilité 24

3.2 Tests de normalité 29

3.3 Tests pour une distribution de Laplace 30

3.4 Tests d'uniformité 31

3.7 Fonctions du progiciel PoweR 46

4.1 Valeurs critiques du test de Cramér-von Mises W^2 52

4.2 Valeurs critiques du test de Shapiro-Wilk W 54

4.3 Puissances des tests (en %) de Meintanis 56

4.4 Valeurs critiques du test de normalité de Filliben 58

4.5 Valeurs critiques du test de normalité de Martinez-Iglewicz 60

4.6 Puissances des tests (en %) $T^*(\alpha)$ 62

4.7 Valeurs critiques du test de Choi-Kim $T^E_{m,n}$ 63

4.8 Puissances des tests (en %) V_3, V_4, A^2, K_1, $T^{(1)}_{n,a}$, et $T^V_{m,n}$ pour une distribution de Laplace 65

4.9 Comparaison de puissances des tests (en %) de Laplace 68

4.11 Puissance (en %) du test t de Student 74

REMERCIEMENTS

Je tiens tout d'abord à remercier les membres du Département de mathématiques et de statistique (DMS) pour m'avoir accueilli durant ces deux années d'études. Je remercie tout particulièrement le Professeur Pierre Lafaye de Micheaux, mon directeur de recherche, pour son aide très précieuse et le temps qu'il a bien voulu me consacrer. J'espère que vous garderez un aussi bon souvenir que moi de cette période.

Je souhaite aussi remercier tous les professeurs et les étudiants ainsi que toutes les personnes travaillant au DMS pour leur aide et leur sympathie. Tous vos avis d'expert m'ont énormément aidé à finir ce mémoire.

Je remercie également Pierre Duchesne, responsable du programme de statistique, et Anne-Marie Dupuis, technicienne en gestion des dossiers étudiants aux cycles supérieurs, pour m'avoir donné la possibilité d'effectuer mes études au sein du Département de mathématiques et de statistique.

INTRODUCTION

Un test d'hypothèses est un procédé d'inférence permettant de décider, à partir de l'étude d'un ou de plusieurs échantillons aléatoires, la validité d'hypothèses relatives à une ou plusieurs populations. Les méthodes de l'inférence statistique nous permettent de déterminer, avec une probabilité donnée, si les différences constatées au niveau des échantillons peuvent être imputables au hasard ou si elles sont suffisamment importantes pour signifier que les échantillons proviennent de populations vraisemblablement différentes. Les tests d'hypothèses font appel à un certain nombre d'hypothèses concernant la nature de la population dont provient l'échantillon étudié (normalité de la variable, égalité des variances, etc).

En fonction de l'hypothèse testée, plusieurs types de tests peuvent être réalisés. Dans le cadre de ce mémoire, nous nous intéressons aux tests destinés à vérifier si un échantillon peut être considéré comme extrait d'une population ayant une certaine distribution (tests d'adéquation).

De nos jours, la supposition de la normalité d'un échantillon ou d'une statistique est courante lorsqu'on effectue certaines analyses statistiques. Cependant, la vérification de l'hypothèse de normalité est souvent négligée. Nous pouvons donc mettre en question la validité de ces analyses. Pour cette raison, nous avons dans un premier temps créé des fonctions qui facilitent la vérification de la non-normalité d'un échantillon. Nous avons intégré ces fonctions dans un progiciel du logiciel R, un outil statistique devenu incontournable.

Nous avons programmé de nombreux tests de normalité existant dans la littérature, puis nous avons généralisé notre progiciel en y incluant des tests pour une distribution de Laplace, des tests d'uniformité, etc. L'utilisateur a le choix entre plusieurs tests d'adéquation quant à la vérification de la loi de ses données, lui permettant ainsi de

choisir une distribution compatible avec ses données. Ce problème ne sera toutefois pas traité dans ce mémoire. D'autre part, certains chercheurs en statistique cherchent à construire de nouveaux tests d'adéquation et notre objectif est de leur fournir un outil qui leur permette de faire une étude de puissance (au moyen de simulations de Monte Carlo) pour de nombreuses distributions alternatives. Notre progiciel contient une liste de lois de probabilité très complète pour la génération de données. Un autre atout important du progiciel développé est de faciliter la reproductibilité de résultats de recherches (numériques) publiés dans ce domaine.

Pour construire ce progiciel, nous avons commencé à écrire des programmes en langage R. Cependant, pour un grand nombre de répétitions de la méthode de simulation de Monte Carlo (nécessaire pour une précision numérique accrue), le temps de calcul devient prohibitif. Nous avons donc décidé de programmer certaines fonctions en C/C++, et de les utiliser depuis R. Cette méthode permet d'améliorer grandement la rapidité d'exécution de nos programmes codés en R. Pour diminuer encore le temps de calcul, un paramètre indiquant le nombre de processeurs souhaité pour le calcul parallèle est disponible. De plus, un interface d'utilisateur (GUI) pour ce progiciel et la possibilité d'avoir des résultats en format LaTeX sont aussi disponibles. Notons que le progiciel que nous avons développé comprend environ 10 000 lignes de code en R et 12 000 lignes de code en C++. Un travail conséquent a également été fourni afin de recenser de façon la plus exhaustive possible les tests d'adéquation de normalité et de Laplace disponibles dans la littérature avant de programmer ces tests en C++.

Le mémoire est organisé comme suit. Nous présentons dans le chapitre 1 une courte introduction à la recherche reproductible. Les notions de base des tests d'hypothèses et la méthode de simulation de Monte Carlo sont abordés dans le chapitre 2. Les buts et les fonctionnalités principales du progiciel **PoweR** se trouvent au chapitre 3. Dans le chapitre 4, nous présentons quelques exemples d'obtention (au moyen de notre package) de tables de puissances issues d'articles publiés dans la littérature.

Chapitre 1

RECHERCHE REPRODUCTIBLE (AVEC R & LaTeX)

1.1. INTRODUCTION

Notre génération de scientifiques computationnels vit dans une époque passionnante : nous disposons d'algorithmes efficaces et de machines de calcul très puissantes. Il a fallu de très nombreuses années aux branches théoriques et expérimentales de la science pour développer des normes de publication. Il est temps de développer également des normes sur la façon dont la recherche computationnelle doit être effectuée et publiée.

L'idée d'une « reproductibilité des résultats par d'autres scientifiques », en référence au calcul computationnel, est plus connue sous le nom de « recherche reproductible », terme inventé dans un context informatique par Jon Claerbout, un professeur de géophysique à l'Université de Stanford. Son idée était que le produit ultime de la recherche est l'article publié, ainsi que l'environnement computationnel complet utilisé pour produire les résultats, incluant le code, les données, etc. Ces derniers sont nécessaires pour la reproduction des résultats de recherche. Peu de temps après sa création, le journal CiSE (*Computing in Science and Engineering*) a publié un document rédigé par Schwab et coll. (2000) sur leur expérience de la création d'un environnement de recherche reproductible. Plus de détails sur le cheminement de cette idée depuis les années 80 sont disponibles dans l'article de Fomel et Claerbout (2009).

Définition 1.1.1. *La reproductibilité est le degré de concordance entre les mesures et observations effectuées sur des échantillons répétés dans des endroits différents par*

des personnes différentes. La reproductibilité fait partie de la précision d'une méthode d'essai.

La reproductibilité réfère aussi à la capacité, pour une personne indépendante, de reproduire une expérience ou une étude. Il s'agit de l'un des grands principes de la méthode scientifique.

De nos jours, la notion de « recherche reproductible » est présente dans plusieurs domaines. Par exemple, Donoho et coll. (2009) décrivent leurs 15 ans d'expérience dans la promotion de la recherche reproductible en analyse de calcul harmonique, en utilisant des outils basés sur le logiciel Matlab tels que les progiciels Wavelab et Sparselab (très connus dans ce domaine). En particulier, les auteurs soulignent le succès de la recherche reproductible par un gain en fiabilité de la recherche informatique, et les efforts nécessaires pour mettre en oeuvre cette discipline dans un groupe de recherche, ainsi que les critiques qui en ont été faites dans les années précédentes. L'article se termine par un appel aux organismes de financement pour soutenir la recherche reproductible.

L'article de LeVeque (2009) décrit une interface Python du progiciel bien connu Clawpack pour résoudre les équations hyperboliques aux dérivées partielles qui apparaissent dans les problèmes de propagation des ondes. L'auteur a été conduit à la recherche reproductible par frustration personnelle avec des publications dans des revues scientifiques et mathématiques qui sont « remplies par de jolies images d'expériences computationnelles que le lecteur n'a aucune chance de reproduire ». Il plaide fortement en faveur de calculs reproductibles et montre un exemple d'utilisation d'une interface simplifiée de Python au code Fortran.

Nous considérons aussi l'article rédigé par Peng et Eckel (2009), dans le domaine de la bio-informatique, un bastion de la recherche reproductible. Leur article décrit le progiciel **cacher**, qui est construit pour le logiciel R. Celui-ci permet une approche modulaire de calculs reproductibles, en travaillant sur les résultats de calculs intermédiaires stockés dans une base de données.

Les exemples ci-dessus nous montrent une grande variété de domaines scientifiques où peut se nicher la notion recherche reproductible. Néanmoins, dans ce rapport, nous abordons seulement la recherche reproductible en statistique, et plus particulièrement

dans le contexte des tests d'hypothèses. En fait, nous avons constaté dans ce contexte qu'il est difficile de trouver un article présentant des simulations dont l'auteur a fournit les codes pour retrouver ses résultats. Nous pouvons citer ici l'article de Zhang (1999) où il a donné le code R pour programmer la statistique Q. Notre premier but est alors de fournir au lecteur un outil qui facilite la reproductibilité de résultats publiés dans le domaine des tests d'adéquation. Les détails techniques se trouvent aux chapitres 3 et 4.

Le lecteur intéressé pourra consulter avec profit les articles Chen et coll. (2011) et Pinto et Vetterli (2009) ainsi que le site web `http://reproducibleresearch.net/`.

1.2. Approche classique pour générer des rapports statistiques

Pour effectuer une analyse statistique, la démarche classiquement utilisée consiste à suivre les étapes ci-dessous :

① Traiter des données (données observées ou simulées) :

(i) saisir des données et les stocker ;

(ii) nettoyer les données, y compris la vérification, la résolution et la correction des erreurs de saisie de données ;

(iii) préparer les données, y compris la transformation / recodage des variables, la création de nouvelles variables, et la création de sous-ensembles nécessaires.

② Effectuer les analyses statistiques proposées et générer les graphiques désirés.

③ Enregistrer les résultats, tableaux et graphiques souhaités.

④ Finalement, écrire un rapport sur les résultats obtenus qui peut inclure du texte, des tableaux et des graphiques.

Les étapes ①, ② et ③ peuvent être effectuées à l'aide de logiciels comme R, SAS, SPSS ou encore Microsoft Excel. Après avoir sauvegardé tous les résultats, tableaux et graphiques nécessaires, nous utilisons un éditeur de texte comme Microsoft

Word ou OpenOffice pour compléter l'étape ④ de la procédure. Il est également important de noter que nous profitons de tableaux et graphiques pré-formatés générés par de nombreux logiciels statistiques. Ensuite, il nous reste à les copier-coller et à taper les résultats à la main dans notre éditeur de texte favori.

1.3. Problèmes rencontrés avec l'approche classique

Commençons par un petit exemple. Nous sommes sur le point de terminer la rédaction de notre rapport statistique. Nous nous rendons compte que le résultat doit être vérifié en exécutant une analyse supplémentaire et donc nous devons d'abord réexécuter l'analyse principale. Nous pouvons alors rencontrer les problèmes suivants :

- les premiers résultats ne correspondent pas à ce que nous avons dans notre rapport, y compris les chiffres, les tableaux et les graphiques sauvegardés.
- Lorsque nous regardons dans notre dossier de projet pour exécuter l'analyse supplémentaire, nous trouvons plusieurs fichiers de données, d'analyse et de résultats. Nous ne nous souvenons pas de celles qui sont pertinentes.
- Parfois, nous devons regénérer le rapport basé sur un sous-ensemble de l'ensemble des données d'origine, au moyen d'une série d'analyses supplémentaires.
- etc.

Pour une analyse statistique, il est toujours difficile d'enregistrer et/ou sauvegarder les étapes effectuées qui nous mènent à générer les résultats. Nous conservons souvent différentes versions des codes d'analyse, des résultats obtenus et des rapports dans des fichiers séparés. Pourtant, après plusieurs modifications d'un ou de plusieurs éléments, il devient plus difficile de déterminer quelles versions correspondent exactement à quelles analyses. À chaque fois que l'analyse change, nous devons regéné ner le rapport à la main et cela prend beaucoup de temps. En outre, il est facile de faire des erreurs en copiant-collant les résultats, tableaux et graphiques.

1.4. Une approche alternative

Nous présentons ici une approche alternative pour rédiger des rapports statistiques :

① utiliser R pour traiter les données et effectuer des analyses statistiques, y compris la récupération des résultats, tableaux et graphiques.

② Rédiger le rapport avec LaTeX - un système de préparation de documents.

Même si nous avons gagné du temps en utilisant R et LaTeX, nous n'avons pas éliminé le besoin de copier-coller les résultats et / ou de les taper à la main. Il est donc préférable d'intégrer l'analyse dans le rapport, c'est-à-dire d'intégrer le code R pour traiter les données et effectuer l'analyse statistique souhaitée dans le corps du document LaTeX qui contient le texte du rapport.

Pour mettre en oeuvre cette approche, il est possible d'utiliser un outil qui s'appelle **Sweave**, (de l'anglais weave, signifiant tissage).

1.5. Recherche reproductible avec Sweave

Sweave est une fonction intégrée à R et créée par Leisch (2002a), présentement professeur à l'Université des ressources naturelles et des sciences de la vie de Vienne (University of Natural Resources and Life Sciences, Vienna).

Sweave est un outil qui permet d'intégrer le code R pour les analyses de données complètes dans les documents LaTeX. Le but est de créer des rapports dynamiques, qui peuvent être mis à jour automatiquement si les données ou les analyses changent. Au lieu d'insérer un graphique ou un tableau préfabriqué dans le rapport, le document contient le code R nécessaire pour l'obtenir. Lorsqu'il est exécuté par R, toutes les sorties des analyses (tableaux, graphiques, etc) sont créées à la volée et insérées dans un document LaTeX final. Cela permet une recherche véritablement reproductible.

La figure 1.1 en page 10 illustre les étapes principales du processus.

Pour obtenir plus d'information sur **Sweave**, vous pouvez consulter Leisch (2002b), Leisch (2003), ainsi que le site web *The Sweave Homepage* (`http://www.statistik.uni-muenchen.de/~leisch/Sweave`).

Remarque 1.5.1.

- *Un exemple détaillé est présenté à l'annexe A pour mieux vous montrer la procédure de la recherche reproductible avec R et* ***Sweave****.*
- *Inspiré par le concept de* ***Sweave****, Lenth et Højsgaard (2007) ont crée* ***SASweave****, un moyen facile et fiable de préparer des documents LaTeX contenant le code SAS, la sortie et les graphiques. Leur site web se trouve à l'adresse suivant :* `http://homepage.cs.uiowa.edu/~rlenth/SASweave`*.*

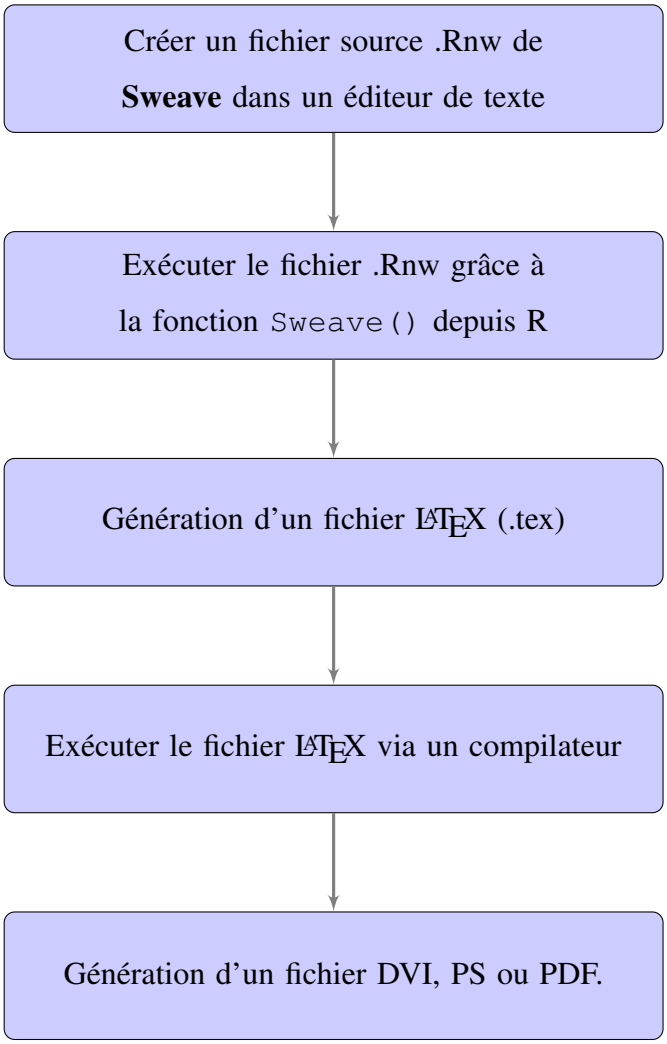

FIGURE 1.1. Diagramme du processus **Sweave**.

1.6. **POWER** - UN OUTIL DE RECHERCHE REPRODUCTIBLE

Comme nous l'avons mentionné dans l'introduction, notre progiciel **PoweR** a été construit dans l'optique de reproduire des résultats numériques de certains articles publiés, afin de vérifier leur exactitude et précision. Pendant notre recherche, nous nous sommes rendus compte que les articles de type comparaison numérique de tests d'adéquation sont très longs à réaliser. Prenons l'exemple de deux articles de ce type, l'un sur la comparaison des tests de normalité de Romão et coll. (2010) et l'autre sur la comparaison des tests d'uniformité de Marhuenda et coll. (2005b). Dans chaque article, les auteurs ont dû programmer tous les tests ainsi que toutes les alternatives qu'ils ont voulu utiliser. Il faut en outre tenir compte du temps d'exécution, de vérification et

de rédaction des résultats obtenus en LaTeX. Il faut compter des mois de travail pour ce type d'article. À l'aide du progiciel **PoweR**, vous pouvez facilement et rapidement retrouver les tableaux des résultats dans les deux articles ci-dessus :

- choisir les tests d'adéquation souhaités ;
- choisir les distributions alternatives ;
- fixer des niveaux, des tailles d'échantillon, le nombre de répétitions de simulation de Monte Carlo utilisées dans l'article, etc ;
- gagner du temps en choissisant le nombre de processeurs souhaités pour le calcul parallèle ;
- lancer le calcul des puissances des tests en choissisant l'option de transformer les résultats obtenus en instructions LaTeX.

Des exemples détaillés se trouvent aux chapitres 3 et 4. En outre, nous allons tenter d'effectuer nous même une étude similaire sur la comparaison des tests pour une distribution de Laplace dont la démarche sera présentée au chapitre 4.

Chapitre 2

TESTS D'HYPOTHÈSES

2.1. RAPPELS THÉORIQUES

En statistique, un test d'hypothèse est un outil permettant d'évaluer une hypothèse statistique à partir d'un jeu de données (échantillon). Plus précisement, les tests d'hypothèses consistent en un outil d'aide à la décision pour éventuellement valider une assertion d'intérêt H_1 (appelée plus communément hypothèse « alternative »). Cette décision statistique sera prise à partir d'une règle de décision construite intuitivement à partir d'une statistique de test T (dépendant d'un échantillon d'observations).

On commence par introduire les deux hypothèses contraires à tester :

- l'hypothèse nulle : habituellement notée H_0 dans le problème de test que l'on étudie ;
- l'hypothèse alternative : habituellement notée H_1 dans le problème de test que l'on étudie ; il s'agit de l'hypothèse contraire à H_0.

Un point fondamental concernant les tests d'hypothèses est la probabilité que l'on a de se tromper. Il y a deux façons de se tromper lors de l'utilisation d'un test statistique :

- une erreur de première espèce : rejeter H_0 à tort (rejeter à tort l'hypothèse nulle lorsqu'elle est vraie).
- une erreur de deuxième espèce : accepter H_0 à tort (ne pas rejeter l'hypothèse nulle alors qu'elle est fausse).

  Un risque (probabilité d'erreur) est associé à chacune de ces erreurs.

De plus, le seuil de signification α du test qui garantit un risque maximal de première espèce faible est toujours fixé à l'avance (en général 5%). Lorsque nous appliquons la règle de décision sur la base d'un jeu de données observées et que nous sommes conduits à accepter H_1, il est intéressant de se demander jusqu'à quel seuil de signification l'acceptation de H_1 aurait été maintenue. De manière alternative, lorsque nous sommes conduits à la non-acceptation de H_1 à un risque α préspécifié, il est aussi intéressant de savoir à partir de quel seuil de signification (le plus petit) on est conduit à accepter H_1. Dans ces deux cas de figure, la solution est donnée par ce que l'on appelle la valeur-p (aussi appelée p-valeur), ainsi définie comme le risque (maximal) à encourir pour accepter H_1. Cette valeur-p est fournie par tout logiciel de statistique, et l'utilisateur saura alors comparer ce risque à un seuil de signification α fixé. Son interpretation est simple : plus la valeur-p est faible, plus la décision (d'accepter l'assertion d'intérêt H_1) est fiable.

Dans le même contexte, nous pouvons mentionner la notion de puissance d'un test : elle est donnée par la quantité $1 - \beta$, où β est le risque de commettre une erreur de deuxième espèce. Plus un test est puissant, plus il est fiable.

2.2. TESTS D'ADÉQUATION

Dans ce document, nous nous intéressons plus particulièrement aux tests d'adéquation. En général, le test d'adéquation consiste à définir une règle de décision concernant la validité d'une hypothèse relative à l'accord global d'une distribution empirique avec une distribution théorique.

Voici quelques examples de tests d'adéquation :

- Test de Shapiro-Wilk : il est conçu spécialement pour étudier la non-normalité d'une variable continue X. C'est l'un des tests les plus puissants pour tester la normalité d'une distribution.
- Test du χ^2 d'ajustement : il s'emploie lorsque l'on veut montrer qu'une variable qualitative ne suit pas une loi théorique donnée. Il permet éventuellement de tester si la loi d'une variable quantitative ne suit pas une loi théorique donnée, mais il faut au préalable regrouper ses observations en classes pour la rendre qualitative.

– Test de Kolmogorov-Smirnov : le but de ce test est le même que celui du khi-deux d'ajustement. Il s'agit de comparer une distribution empirique à une distribution théorique entièrement spécifiée.
– Test de Kolmogorov-Smirnov pour deux échantillons : ce test permet de comparer deux distributions.
– etc.

Pour faciliter l'utilisation de tels tests d'adéquation, nous avons créé un nouveau progiciel pour le logiciel R, dont nous détaillons les buts et fonctionnalités dans les chapitres suivants.

2.3. MÉTHODE DE MONTE CARLO

La méthode que nous exposons dans ce chapitre consiste à examiner les propriétés en échantillon fini des statistiques de test en utilisant les expériences Monte Carlo. Le terme « Monte Carlo » est employé dans de nombreuses disciplines et fait référence aux procédures où les quantités d'intérêt sont approximées en générant de nombreuses réalisations aléatoires d'un certain processus stochastique puis en calculant une moyenne empirique des valeurs obtenus.

Entre deux tests correctement bâtis, celui qui a la plus grande puissance devrait être préféré en pratique. Voici une procédure qui montre comment évaluer la puissance d'un test d'adéquation $H_0 : X \sim \mathcal{L}$ versus $H_1 : X \not\sim \mathcal{L}$, par la méthode de simulation de Monte Carlo :

(1) Nous commençons par déterminer la région critique d'un test sous l'hypothèse nulle H_0. Il est important de rappeler que la région critique est un ensemble des valeurs de la variable de décision qui conduisent à écarter H_0 au profit de H_1. Soit M_1 le nombre de répétitions que nous voulons effectuer pour trouver la région critique. Pour chaque répétition ($1 \leq j \leq M_1$) nous procédons comme suit :

(i) Générer un échantillon de taille n_1 sous H_0

$$X_1, X_2, \ldots, X_{n_1} \text{ i.i.d.} \sim \mathcal{L}.$$

(ii) Calculer la valeur observée de la statistique de test T_j.

(iii) Ensuite il faut ordonner les statistiques de test calculées :

$$T_{(1)} \leq T_{(2)} \leq \ldots \leq T_{(M_1)}.$$

La région conduisant au rejet de l'hypothèse nulle s'appelle la région critique, et les limites de cette région s'appellent les valeurs critiques du test. Selon le niveau de signification α choisi, les valeurs critiques se trouvent parmi les statistiques d'ordre $T_{(j)}$ où $j = 1, 2, \ldots, M_1$.

Nous rappelons ici la définition de la fonction quantile d'une loi de probabilité, elle est l'inverse de sa fonction de répartition. Si F désigne la fonction de répartition, alors la fonction quantile Q est la fonction qui à $u \in]0, 1[$ associe :

$$Q(u) = inf\{x \text{ tel que } F(x) \geq u\}.$$

La fonction quantile empirique d'un échantillon est la fonction quantile de sa distribution empirique.

Soit $T = (T_1, T_2, \ldots, T_{M_1})$ un échantillon et $(T_{(1)}, T_{(2)}, \ldots, T_{(M_1)})$ le vecteur de ses statistiques d'ordre. La fonction quantile empirique de l'échantillon est la fonction $\hat{Q}$ qui, pour tout $j = 1, 2, \ldots, M_1$, vaut $T_{(j)}$ sur l'intervalle $]\frac{j-1}{M_1}, \frac{j}{M_1}[$:

$$\forall u \in]\tfrac{j-1}{M_1}, \tfrac{j}{M_1}[, \hat{Q}(u) = T_{(j)}.$$

Les $T_{(j)}$ sont distincts car nous les avons calculés avec différents échantillons aléatoires.

Les valeurs critiques obtenues sont sauvegardées pour être utilisées plus tard.

(2) Soit M_2 le nombre de répétitions que nous voulons effectuer pour calculer la puissance sous une certaine hypothèse alternative donnée H_1. Pour chaque répétition ($1 \leq j \leq M_2$) nous procédons comme suit :

(i) Générer un échantillon de taille n_2 sous H_1

$$X_1, X_2, \ldots, X_{n_2} \text{ i.i.d.} \sim \tilde{\mathcal{L}}.$$

où $\tilde{\mathcal{L}}$ est une certaine distribution différente de $\mathcal{L}$.

(ii) Calculer la valeur observée de la statistique de test T_j.

(3) Compter le nombre de fois où une statistique T_j, où $j = 1, 2, \ldots, M_2$, se trouve dans la région critique déterminée en (1). Cela donnera une évaluation empirique de la puissance du test.

En utilisant la méthode de simulation de Monte Carlo, le progiciel **PoweR** que nous avons développé permet de calculer les puissances empiriques de différents tests d'adéquation.

Remarque 2.3.1.

- *Les nombres de répétitions* M_1 *et* M_2 *peuvent être identiques.*
- *Les tailles d'échantillons* n_1 *et* n_2 *doivent être identiques.*
- *Considérons chaque itération de simulation de Monte Carlo comme une expérience de loi Bernoulli, où chaque essai donne lieu à un « succès » (*H_0 *est rejetée) avec probabilité* p*, ou à un « échec » (*H_0 *n'est pas rejetée) avec probabilité* $1-p$*. Alors la variable aléatoire* x *qui compte le nombre de succès obtenus en* M *essais (*M *répétitions de Monte Carlo) suit la loi binomiale* $\mathcal{B}(M,p)$*. Il est essentiel que les essais soient indépendants les uns des autres.*

  Une estimation de la puissance p *du test est alors donnée par la proportion* $\hat{p} = \dfrac{x}{M}$.

  Cependant, lorsque la taille d'échantillon est suffisamment grande, il est possible d'avoir recours au théorème central limite qui permet d'approximer la loi de $\hat{p}$ *par une loi* normale*. Plus précisément, la moyenne expérimentale d'une répétition d'expériences identiques converge, quand* M *augmente, vers une variable aléatoire de loi* normale*. Il en résulte que, pour* M *suffisamment grand, nous pouvons considérer que* $\hat{p}$ *suit une loi* normale *de moyenne* p *et d'écart-type* $\sqrt{\dfrac{p(1-p)}{M}}$.

  Les commandes `binom.test()` *et* `prop.test()` *dans R peuvent nous permettre de calculer un intervalle de confiance de* p.

Chapitre 3

POWER : BUTS ET FONCTIONNALITÉS

3.1. PROBLÉMATIQUE

L'une des conditions de validité les plus importantes de nombreuses procédures statistiques est la normalité des données. Pourtant, la vérification des hypothèses de normalité est souvent négligée. Par exemple, la normalité des erreurs d'une régression linéaire est rarement testée. Nous pouvons alors remettre en question la validité des résultats obtenus.

Il existe différents tests de normalité et certains sont implémentés directement dans les logiciels statistiques :

- SPSS : le test de Kolmogorov-Smirnov, le test de Shapiro-Wilk, le diagramme Quantile-Quantile (Q-Q plot).
- SAS : le test de Kolmogorov-Smirnov, le test de Shapiro-Wilk (taille de l'échantillon ≤ 2000), le test d'Anderson-Darling, et le test de Cramér-von Mises sont inclus dans la procédure PROC UNIVARIATE.
- R : le test de Shapiro-Wilk est intégré dans la base de R. On peut l'utiliser en tapant l'instruction : `shapiro.test()`.
- etc.

Pour aider l'utilisateur de R, nous avons decidé dans un premier temps de créer un nouveau progiciel, nommé **PoweR**, qui contient plusieurs tests de normalité. Nous avons aussi ajusté quelques tests d'adéquation à une distribution de Laplace, et des tests d'adéquation d'uniformité.

Dans cette section, nous présentons les fonctionnalités principales du progiciel **PoweR** :

① Générer des données selon les lois de probabilité (usuelles ou non).

② Effectuer des tests de normalité sur les données, calculer les quantiles empiriques, les puissances et les valeurs-p des tests utilisés.

③ Possibilités de :

(i) rajouter des lois de probabilité ;

(ii) rajouter des tests d'adéquation.

④ Permettre de reproduire des résultats d'articles publiés, de détecter des erreurs dans des tables de simulation de puissance ainsi que de proposer de nouvelles études de comparaison de puissances entre les tests.

⑤ Faciliter l'usage du progiciel par un interface d'utilisateur (GUI).

⑥ Afficher des résultats en instructions LaTeX.

Comme une image vaut mille mots, nous présentons le diagramme UML 3.1 dans la page suivante, qui illustre les fonctions de base du progiciel **PoweR**. Vous pouvez observer les fonctions principales comme : générer les données, calculer une ou plusieurs statistiques de test, ou calculer les puissances des tests. Par exemple, si l'utilisateur veut calculer les valeurs critiques des tests d'adéquation, il peut appeler la fonction `many.crit()`. Ensuite, cette fonction appelle la fonction `compquant()` qui calcule les valeurs critiques pour chaque statistique de test, et cela appelle encore la fonction `gensample()` pour la génération des données. Les exemples détaillés seront présentés à la fin de ce chapitre.

De plus, quand un chercheur cherche à construire un nouveau test d'adéquation, il voudrait pouvoir comparer la puissance de son test par rapport aux autres tests déjà existants, ainsi que son fonctionnement contre les distributions alternatives. Nous procédons donc comme suit :

– avant de rentrer dans le détail des fonctionnalités du progiciel, nous commençons par lister tous les matériels essentiels du progiciel (liste des lois de probabilité, liste des tests d'adéquation, etc) ;

– ensuite, pour chaque fonctionnalité principale nous présentons une brève description des commandes utiles et de leur usage dans R ;
– nous terminons en montrant la procédure à suivre pour rajouter une nouvelle loi ou un nouveau test dans le progiciel, et présentons un manuel des outils de débogage usuels.

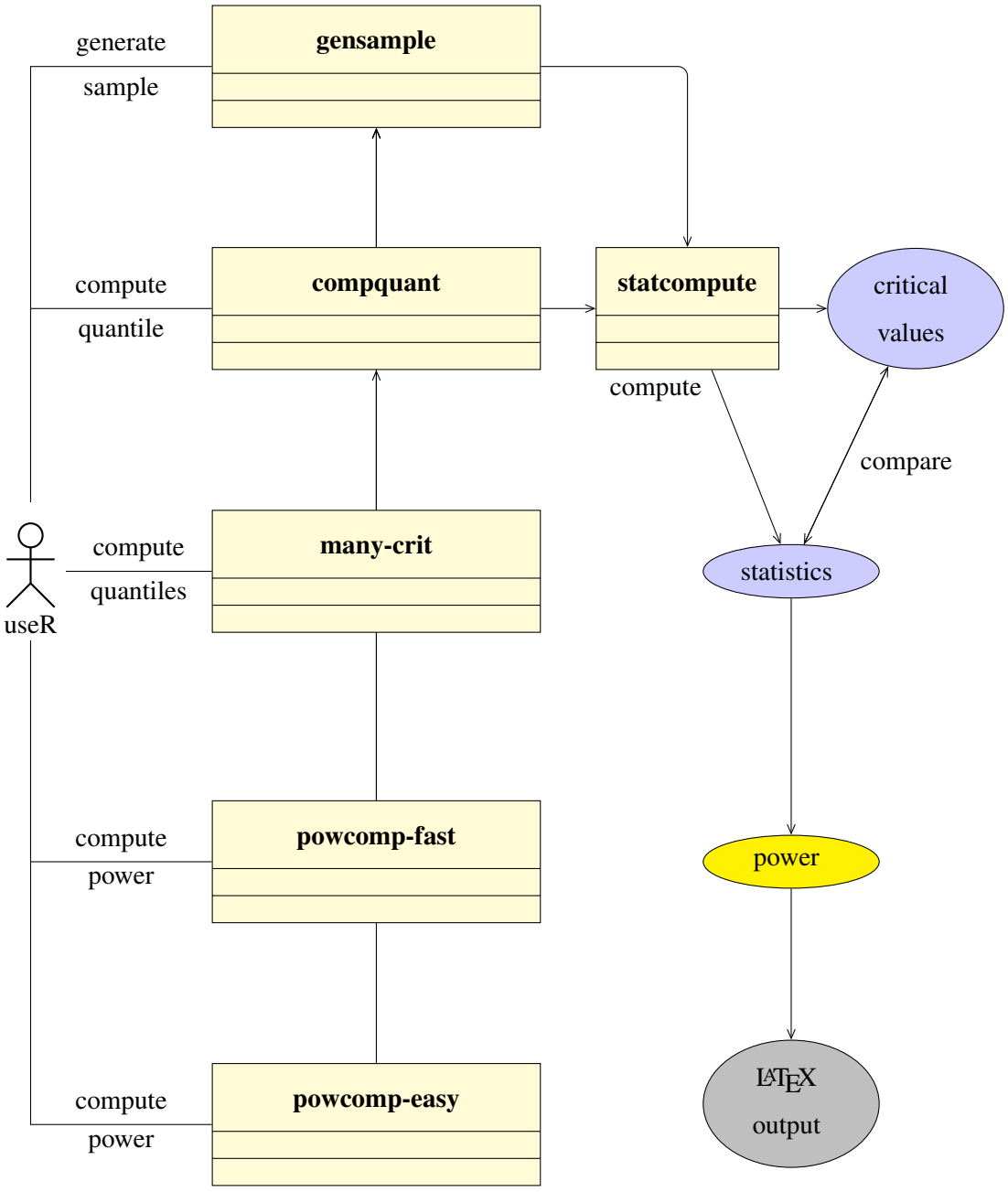

FIGURE 3.1. Diagramme UML du progiciel PoweR

Remarque 3.1.1.

Une des caractéristiques intéressantes du logiciel R est qu'il permet aux utilisateurs de charger et d'appeler des routines codées en C/C++ ou en Fortran de façon dynamique. Cela permet d'améliorer grandement la rapidité d'exécution de nos programmes codés en R.

Ainsi, nous avons décidé de coder une grande partie de notre package en langage C/C++, notamment toutes les lois et tous les tests. Voici les 3 étapes principales de la démarche d'interface de R et C/C++ ou Fortran :

- *Compilation du code C/C++ ;*
- *Chargement en R du code C/C++ compilé ;*
- *Appel en R des fonctions programmées en C/C++.*

3.2. MATÉRIELS ESSENTIELS DU PROGICIEL **POWER**

3.2.1. Liste des lois de probabilité

Le tableau 3.1 présente une liste de 37 lois de probabilité déjà incluses dans notre progiciel. Il s'agit une des listes les plus complètes parmi tous les logiciels permettant la génération de données suivant une loi de probabilité. Nous donnons, pour chaque loi, son index, son nom, sa densité, un moyen pour en générer des observations, son espérance et sa variance.

Nous commençons par donner une liste de références associées à certains indices de lois de probablité :

- 12, 19, 20, 22 : Kallenberg et Ledwina (1997) ;
- 13, 15, 16 : Quesenberry et Miller (1977) ;
- 14 : Quesenberry et Miller (1977) Bates (1955) ;
- 17 : Johnson (1949) ;
- 18 : Joiner et Rosenblatt (1971)
- 21 : Azzalini (2005) ;
- 23, 26, 28 : Coles (2001) ;
- 24 : Nadarajah (2005) ;
- 25 : Chambers et coll. (1976) ;
- 27 : Castillo et coll. (2005) ;

– 29 : Feller (1968, 1971) ;
– 30 : Leone et coll. (1961) ;
– 31, 32, 33 : Romão et coll. (2010) ;
– 34 : Desgagné et coll. (2013) ;
– 36 : Kotz et coll. (2001) ;
– 37 : Atkinson (1982).

TABLE 3.1. Liste de lois de probabilité

	Loi	Notation	Densité	Génération	Espérance	Variance
1	Laplace	$Lp(\mu, b)$	$\frac{1}{2b}\exp\left(-\frac{\lvert x-\mu\rvert}{b}\right)$	$\mu - b.sgn\{U-\frac{1}{2}\}\ln(1-2\lvert U-\frac{1}{2}\rvert)$	μ	$2b^2$
2	Normal	$N(\mu,\sigma)$	$(\sqrt{2\pi}\sigma)^{-1}\exp^{-\frac{x^2}{2\sigma^2}}$	$\sigma Z+\mu$	μ	σ^2
3	Cauchy	$Cauchy(l,s)$	$\frac{1}{\pi s(1+(\frac{x-l}{s})^2)}$	rcauchy(location,scale)	indéfinie	indéfinie
4	Logistic	$Lg(\mu,s)$	$\frac{1}{s}e^{-\frac{x-\mu}{s}}(1+e^{-\frac{x-\mu}{s}})^{-2}$	$\mu + s\ln\left(\frac{U}{1-U}\right)$	μ	$\frac{\pi^2}{3}s^2$
5	Gamma	$Gamma(a,b)$	$\frac{1}{(1/b)^a\Gamma(a)}x^{a-1}e^{-xb}, X\geq 0, a,b>0$	rgamma(a,1/b)	$\frac{a}{b}$	$\frac{a}{b^2}$
6	Beta	$Beta(\alpha,\beta)$	$\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}x^{\alpha-1}(1-x)^{\beta-1}, 0\leq x\leq 1$	rbeta(α, β)	$\frac{\alpha}{\alpha+\beta}$	$\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$
7	Uniform	$U(a,b)$	$(b-a)^{-1}, a\leq x\leq b$	$\frac{U}{(b-a)}+a$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
8	Student	$Student-t(k)$	$(\sqrt{k\pi})^{-1}\frac{\Gamma\left(\frac{k+1}{2}\right)}{\Gamma(\frac{k}{2})}\left(1+\frac{x^2}{k}\right)^{-\frac{k+1}{2}}, k>0$	$\frac{Z}{\sqrt{\chi^2(k)/k}}$	k=1: indéfinie k>1: 0	$k\leq 2$: ∞ $k>2$: $\frac{k}{k-2}$
9	Chi-squared	$\chi^2(k)$	$2^{-k/2}\Gamma(k/2)^{-1}x^{k/2-1}e^{-x/2}, x>0$	$\sum_{i=1}^{k}Z_i^2$	k	$2k$
10	Log Normal	$LN(\mu,\sigma)$	$\frac{1}{x\sigma\sqrt{2\pi}}e^{-\frac{(\ln x-\mu)^2}{2\sigma^2}}$	$\exp\{N(\mu,\sigma^2)\}$	$e^{\mu+\sigma^2/2}$	$(e^{\sigma^2}-1)e^{2\mu+\sigma^2}$
11	Weibull	$W(\lambda,k)$	$\frac{\lambda}{k}\left(\frac{x}{k}\right)^{\lambda-1}e^{-(x/k)^\lambda}$	$k(-\ln(U))^{1/\lambda}$	$\mu = k\Gamma\left(1+\lambda^{-1}\right)$	$k^2\Gamma\left(1+\frac{2}{\lambda}\right)-\mu^2$
12	Shifted Exponential	$SE(l,b)$	$b\exp\{-(x-l)b\}, x\geq l$	$\frac{-\ln U}{b}+l$	$l+\frac{1}{b}$	$\frac{1}{b^2}$
13	Power Uniform	U^{1+j}	$\frac{1}{1+j}x^{-\frac{j}{j+1}}$	U^{1+j}	$\frac{1}{j+2}$	$\frac{1}{2j+3}\frac{(j+1)^2}{(j+2)^2}$
14	Average Uniform	$AveUnif(k,a,b)$	$\frac{k^k}{(k-1)!}\sum_{j=0}^{\lfloor k\frac{x-a}{b-a}\rfloor}(-1)^j\binom{k}{j}(\frac{x-a}{b-a}-\frac{j}{k})^{k-1}$ pour $a\leq x\leq b$	mean(runif(k,a,b))	$\frac{1}{2}(a+b)$	$\frac{1}{12k}(b-a)^2$
15	UUniform	$UUnif(j)$	$(2(1+j))^{-1}(x^{-j/(1+j)}+(1-x)^{-j/(1+j)})$	$SU^{j+1}+(1-S)(1-U^{j+1})$	$\frac{1}{2}$	$\frac{2j^2+3j+2}{2(2j+3)(2j+4)}$
16	VUniform	$VUnif(j)$	indéfinie	si $Z_{j+1}<0.5 : Z_{j+1}+0.5$ sinon : $Z_{j+1}-0.5$, où $Z_{j+1}=AveUnif(j+1)$	$\frac{1}{2}$	voir remarque 3.2.1
17	Johnson SU	$JSU(\mu,\sigma,\nu,\tau)$	$\frac{1}{c\sigma\frac{1}{\tau}}\frac{1}{\sqrt{z^2+1}}\frac{1}{\sqrt{2\pi}}e^{-r^2/2}$	$\mu+c\sigma\sqrt{w}\sinh(\omega)+c\sigma\sinh\left(\frac{1}{\tau}(Z+\nu)\right)$ $r=-\nu+\tau\sinh^{-1}(z)$ $z=\frac{x-(\mu+c\sigma\sqrt{w}\sinh(\omega))}{c\sigma}$	μ	σ^2

Continuer à la page suivante

TABLE 3.1 – Suite de la page précédente

	Law	Notation	Densité	Génération	Espérance	Variance
				$c = ((w-1)(w\cosh(2\omega)+1)/2)^{-1/2}$ $w = e^{(\frac{1}{\tau})^2}$ et $\omega = -\nu\frac{1}{\tau}$		
18	Symmetrical Tukey	$TU(l)$	indéfinie	$\frac{U^l-(1-U)^l}{l}, -1 \le X \le 1$	0	$\frac{2}{l^2}\left(\frac{1}{2l+1} - \frac{\Gamma^2(l+1)}{\Gamma(2l+2)}\right)$
19	Location Contaminated	$LoConN(p,m)$	$\frac{1}{\sqrt{2\pi}}\left[pe^{-\frac{(x-m)^2}{2}} + (1-p)e^{-\frac{x^2}{2}}\right]$	U = runif(0,1) ; si (U<p) x = rnorm(m,1) ; sinon x = rnorm(0,1)	pm	$1-(pm)^2+pm^2$
20	Johnson SB	$JSB(g,d)$	$\frac{d}{\sqrt{2\pi}}\frac{1}{x(1-x)}e^{-\frac{1}{2}\left(g+d\ln\frac{x}{1-x}\right)^2}, d>0$	$\left(1+e^{-\frac{Z-g}{d}}\right)^{-1}, 0<X<1$	indéfinie	indéfinie
21	Skew Normal	$SkewN(\xi,\omega,\alpha)$	$\left(\frac{2}{\omega}\right)\phi\left(\frac{x-\xi}{\omega}\right)\Phi\left(\alpha\left(\frac{x-\xi}{\omega}\right)\right), \omega>0$	$\xi+\omega Y$ si $(U_0 \ge 0)$ $Y = U_1$; sinon $Y = -U_1$ U_0, V indépendant de $N(0,1)$ $U_1 = \delta U_0 + \sqrt{1-\delta^2}V$ $\delta = \alpha/\sqrt{1+\alpha^2}$	$\xi+\omega\sqrt{2/\pi}\delta$	$\omega^2(1-2\delta^2/\pi)$
22	Scale Contaminated	$ScConN(p,d)$	$\frac{1}{\sqrt{2\pi}}\left[\frac{p}{d}e^{-\frac{x^2}{2d^2}} + (1-p)e^{-\frac{x^2}{2}}\right]$	U = runif(0,1) ; si (U<p) x = rnorm(0,d) ; sinon x = rnorm(0,1)	0	pd^2+1-p
23	Generalized Pareto	$GP(\mu,\sigma,\xi)$	si $\xi>0$: $\mathbb{1}[0 \le x \le \mu+\frac{\sigma}{\xi}]\frac{1}{\sigma}(1-\xi\frac{x-\mu}{\sigma})^{(1-\xi)/\xi}$ sinon : $\mathbb{1}[-\mu \le x \le \infty]\frac{1}{\sigma}(1-\xi\frac{x-\mu}{\sigma})^{(1-\xi)/\xi}$	$\mu - \frac{\sigma(U^{\xi}-1)}{\xi}$	$\mu+\frac{\sigma}{1+\xi}$ $(\xi<1)$	$\frac{\sigma^2}{(1+\xi)^2(1+2\xi)}$ $(\xi<1/2)$
24	Generalized Error Distribution	$GED(\mu,\sigma,p)$	$\frac{p}{2\sigma\Gamma(1/p)}e^{-(\lvert x-\mu\rvert/\sigma)^p}$	$\mu+\sigma\left(\frac{G_p}{p}\right)^{(1/p)}\text{sign}(U-1/2)$	μ	$\frac{\sigma^2\Gamma(3/p)}{\Gamma(1/p)}$
25	Stable	$S(\alpha,\beta,c,\mu)$	indéfinie $0<\alpha\le 2, -1\le\beta\le 1$ $c>0$ et $\mu\in\mathbb{R}$	si ($\alpha=1$ et $\beta=0$) tmp = rcauchy(0,1) $x = tmp*c+\mu$; sinon voir fonction **rstable()**	μ si $\alpha>1$ indéfinie sinon	$2c^2$ si $\alpha=2$ ∞ sinon

Continuer à la page suivante

TABLE 3.1 – Suite de la page précédente

	Law	Notation	Densité	Génération	Espérance	Variance
				dans package **stabledist**		
26	Gumbel	$\mathtt{Gumbel}(\mu,\sigma)$	$\frac{1}{\sigma}\exp\left\{-\exp\left[-\left(\frac{x-\mu}{\sigma}\right)\right]-\left(\frac{x-\mu}{\sigma}\right)\right\}$	$\mu-\sigma\ln(E)$	$\mu+\sigma(-\Gamma'(1))$	$\frac{\pi^2}{6}\sigma^2$
27	Frechet	$\mathtt{Frechet}(\mu,\sigma,\alpha)$	$\frac{\alpha}{\sigma}\left(\frac{x-\mu}{\sigma}\right)_+^{-\alpha-1}\exp\left\{-\left(\frac{x-\mu}{\sigma}\right)^{-\alpha}\right\}$	$\mu+\sigma E^{-1/\alpha}$	si $\alpha>1$: $\mu+\sigma\Gamma(1-\frac{1}{\alpha})$; sinon ∞	si $\alpha>2$: $\sigma^2(\Gamma(1-\frac{2}{\alpha})-(\Gamma(1-\frac{1}{\alpha}))^2)$; sinon ∞
28	Generalized Extreme Value	$\mathtt{GEV}(\mu,\sigma,\xi)$	$\xi\neq 0 : [1+z]^{-\frac{1}{\xi}-1}\exp\left\{-[1+z]^{-\frac{1}{\xi}}\right\}/\sigma$ avec $z=\xi\frac{x-\mu}{\sigma}$, pour $1+z>0$; $\xi=0$: Gumbel	si $\xi=0 : \mu-\sigma\ln(E)$; sinon : $\mu+\sigma(E^{-\xi}-1)/\xi$	si $\xi\neq 0, \xi<1$: $\mu+\sigma\frac{\Gamma(1-\xi)-1}{\xi}$; $\mu+\sigma\gamma$ si $\xi=0$; ∞ si $\xi\geq 1$; γ : constante d'Euler	si $\xi\neq 0, \xi<\frac{1}{2}$: $\sigma^2\frac{(g_2-g_1^2)}{\xi^2}$; $\sigma^2\frac{\pi^2}{6}$ si $\xi=0$; ∞ si $\xi\geq\frac{1}{2}$; $g_k=\Gamma(1-k\xi)$
29	Generalized Arcsine	$\mathtt{GArcSine}(\alpha)$	$\frac{\sin(\pi\alpha)}{\pi}x^{-\alpha}(1-x)^{\alpha-1}$ pour $0\leq x\leq 1$ et $0<\alpha<1$	$\mathtt{rbeta}(1-\alpha,\alpha)$	$1-\alpha$	$(1-\alpha)\alpha/2$
30	Folded Normal	$\mathtt{FoldN}(\mu,\sigma)$	dnorm(x,mu,sigma)+dnorm(-x,mu,sigma) pour $x\geq 0$	$\lvert N(\mu,\sigma^2)\rvert$	$\sigma\sqrt{\frac{2}{\pi}}e^{-\frac{\mu^2}{2\sigma^2}}+\mu[1-2\Phi(-\frac{\mu}{\sigma})]$	$\mu^2+\sigma^2-\left\{\sigma\sqrt{\frac{2}{\pi}}e^{-\frac{\mu^2}{2\sigma^2}}+\ldots\right\}\{\ldots+\mu[1-2\Phi(-\frac{\mu}{\sigma})]\}^2$
31	Mixture Normal	$\mathtt{MixN}(p,m,d)$	p*dnorm(x,m,d)+(1-p)*dnorm(x)	U = runif(0,1) ; si (U<p) x = rnorm(m,d) ; sinon x[i] = rnorm(0,1)	mp	$(1-p)(1+pm^2)+pd^2$
32	Truncated Normal	$\mathtt{TruncN}(a,b)$	$\frac{\exp(-x^2/2)}{\sqrt{2\pi}(\Phi(b)-\Phi(a))}\mathbb{I}[a\leq x\leq b]$	Z = rnorm(0,1) tant que ((Z<a) ‖ (Z>b)) Z = rnorm(0,1) x = Z	$\frac{\phi(a)-\phi(b)}{\Phi(b)-\Phi(a)}$	$1+\frac{a\phi(a)-b\phi(b)}{\Phi(b)-\Phi(a)}-\left(\frac{\phi(a)-\phi(b)}{\Phi(b)-\Phi(a)}\right)^2$
33	Normal with outliers	$\mathtt{Nout}(a)$	indéfinie	$\mathbf{a}\in\{1,2,3,4,5\}$	0	1

Continuer à la page suivante

TABLE 3.1 – Suite de la page précédente

	Law	Notation	Densité	Génération	Espérance	Variance
				x = rnorm(0,1) avec **a** aberrantes		
34	Generalized Exponential Power	GEP(t1, t2, t3)	si $\|x\| \geq z_0$: $p(x;\gamma,\delta,\alpha,\beta,z_0) \propto e^{-\delta\|x\|^\gamma}\|x\|^{-\alpha}(\log\|x\|)^{-\beta}$ si $\|x\| < z_0$: $p(x;\gamma,\delta,\alpha,\beta,z_0)$	voir fonction **law34.cpp** dans **PoweR**	indéfinie	indéfinie
35	Exponential	$\mathrm{Exp}(\lambda)$	$f(x) = \lambda e^{-\lambda x}$ pour $x \geq 0$	$\mathrm{rexp}(\frac{1}{\lambda})$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$
36	Asymmetric Laplace	ALp(μ, b, k)	pour $x \leq \mu$: $f(x) = \frac{\sqrt{2}}{b}\frac{k}{1+k^2}\exp\left(-\frac{\sqrt{2}}{b*k}\|x-\mu\|\right)$ pour $x > \mu$: $f(x) = \frac{\sqrt{2}}{b}\frac{k}{1+k^2}\exp\left(-\frac{\sqrt{2}*k}{b}\|x-\mu\|\right)$	$\mu + b * \log(\frac{\mathtt{runif(n)}^k}{\mathtt{runif(n)}^{1/k}})/\sqrt{2}$	$\mu + b * \frac{(\frac{1}{k}-k)}{\sqrt{2}}$	$b^2 * \frac{1+k^4}{2*k^2}$
37	Normal-inverse Gaussian	NIG(α, β, δ, μ)	$\frac{\alpha\delta K_1(\alpha\sqrt{\delta^2+(x-\mu)^2})}{\pi\sqrt{\delta^2+(x-\mu)^2}} e^{\delta\gamma+\beta(x-\mu)}$ $\gamma = \sqrt{\alpha^2-\beta^2}$ K_1 : fonction de Bessel de deuxième espèce	voir fonction **rnig**() dans package **fBasics**	$\mu + \frac{\beta\delta}{\gamma}$	$\frac{\delta\alpha^2}{\gamma^3}$

Remarque 3.2.1.

- U *signifie la loi* $\mathrm{Uniforme}(0,1)$, E *représente la loi* $\mathrm{Exponentielle}$, U *et* E *sont indépendantes.*
- S *signifie une loi indépendante de* U *et telle que* $P[S=0] = P[S=1] = 1/2$.
- Z *signifie la loi* $\mathrm{Gaussienne}$ *et* G_p *représente la loi* $\mathrm{Gamma}(1/p, p)$.
- *La loi* $\mathrm{Average\ Uniform}$ *s'appelle aussi loi* $\mathrm{Bates}(k,a,b)$. *Dans Quesenberry et Miller (1977), c'est* $\mathrm{AveUnif}(k+1,0,1)$.
- *Nous passons de* $\mathrm{Generalized\ Pareto}(\mu,\sigma,\xi)$ *à* $\mathrm{Pareto}(a,k)$ *en posant* $\mu = k$, $\xi = a^{-1}$ *et* $\sigma = ka^{-1}$.
- *Nous passons de* $\mathrm{Generalized\ Pareto}(\mu,\sigma,\xi)$ *à* Pareto *décentrée (Shifted Pareto) en posant* $\mu = 0$, $\xi = 1/2$ *et* $\sigma = 1/2$.
- *Nous passons de* $\mathrm{Johnson\ SU}(\mu,\sigma,\nu,\tau)$ *à* $\mathrm{Johnson\ SB}(g,d)$ *en posant* $\tau = d$, $\nu = -g$, $\sigma = c^{-1} = \left[(e^{d^2}-1)(e^{d^2}\cosh(2g/d)+1)/2\right]^{-1/2}$ *et* $\mu = -\sqrt{e^{d^2}}\sinh(g/d)$.
- *Nous passons de* $\mathrm{Generalized\ Error\ Distribution}(\mu,\sigma,p)$ *à* $\mathrm{Generalized\ Error\ Distribution}(\lambda)$ *en posant* $\mu = 0$, $p = \lambda$ *et* $\sigma = \dfrac{1}{\lambda^{1/\lambda}\sigma}$ *avec* $C_\lambda = \sqrt{\Gamma(3\lambda^{-1})/\Gamma(\lambda^{-1})}$.
- *La variance de la loi* $\mathrm{VUnif}(j)$ *s'écrit de façon suivante :*

$$\mathrm{Var}(Y_j) = -\frac{1}{4} + \frac{1}{12(j+1)} + \frac{1}{j!}\sum_{k=0}^{j+1}(-1)^k\binom{j+1}{k} * \left\{(-1)^{j+1}\frac{k^{j+2}}{(j+1)^2(j+2)} + \mathrm{sign}(\frac{k}{j+1} - 0.5)\left(\frac{j+1}{2}-k\right)^{(j+1)}\left[\frac{0.5-k/(j+1)}{j+2} + \frac{k}{(j+1)^2}\right]\right\}.$$

où $\mathrm{sign}(0) = 1$.

3.2.2. Liste des tests d'adéquation présents dans le progiciel

Dans cette section, nous présentons les tests de compatibilité à la loi normale, à une distribution de Laplace, et à la loi uniforme qui sont inclus dans notre progiciel **PoweR**.

3.2.2.1. *Tests de normalité*

Le tableau 3.2 présente une liste des tests de normalité que nous avons inclus dans notre progiciel. L'ordre de présentation est chronologique. Nous donnons, pour chaque test, son index, son nom et sa référence.

TABLE 3.2. Liste des tests de normalité

	Test	Référence
1	Test de Lilliefors $K-S$	Lilliefors (1967)
2	Test d'Anderson-Darling AD^*	D'Agostino et Stephens (1986)
3	Test de Zhang-Wu Z_C	Zhang et Wu (2005)
4	Test de Zhang-Wu Z_A	Zhang et Wu (2005)
5	Test de Glen-Leemis-Barr P_S	Glen et coll. (2001)
6	Test de D'Agostino-Pearson K^2	D'Agostino et Pearson (1973, 1974)
7	Test de Jarque-Bera JB	Jarque et Bera (1987)
8	Test de Doornik-Hansen DH	Doornik et Hansen (2008)
9	Test de Gel-Gastwirth RJB	Gel et Gastwirth (2008)
10	Test de Hosking T_{Lmom}	Hosking (1990)
11	Test de Hosking $T^{(1)}_{Lmom}$	Hosking (1990)
12	Test de Hosking $T^{(2)}_{Lmom}$	Hosking (1990)
13	Test de Hosking $T^{(3)}_{Lmom}$	Hosking (1990)
14	Test de Bontemps-Meddahi BM_{3-4}	Bontemps et Meddahi (2005)
15	Test de Bontemps-Meddahi BM_{3-6}	Bontemps et Meddahi (2005)
16	Test de Brys-Hubert-Struyf T_{MC-LR}	Brys et coll. (2008)
17	Test de Bonett-Seier T_w	Bonett et Seier (2002)
18	Combinaison de T_{MC-LR} & T_w	Brys et coll. (2008); Bonett et Seier (2002)
19	Test de Cabana-Cabana $T_{S,l}$	Cabaña et Cabaña (1994)
20	Test de Cabana-Cabana $T_{K,l}$	Cabaña et Cabaña (1994)
21	Test de Shapiro-Wilk W	Shapiro et Wilk (1965)
22	Test de Shapiro-Francia W'	Shapiro et Francia (1972)
23	Test de Shapiro-Wilk modifié $\tilde{W}$	Rahman et Govindarajulu (1997)
24	Test de D'Agostino D	D'Agostino (1971)
25	Test de Filliben r	Filliben (1975)
26	Test de Chen-Shapiro CS	Chen et Shapiro (1995)
27	Test de Zhang Q	Zhang (1999)
28	Test de Zhang $Q-Q^*$	Zhang (1999)
29	Test de Barrio-Cuesta Albertos	Barrio et coll. (1999)

Continuer à la page suivante

TABLE 3.2 – Suite de la page précédente

	Test	Référence
	-Matran-Rodriguez BCMR	
30	Test de Coin β_3^2	Coin (2008)
31	Test de Epps-Pulley $T^*(\alpha)$	Epps et Pulley (1983)
32	Test de Martinez-Iglewicz I_n	Martinez et Iglewicz (1981)
33	Test de Gel-Miao-Gastwirth R_{sJ}	Gel et coll. (2007)
34	Test de Zhang Q^*	Zhang (1999)
35	Test de Desgagné-Micheaux-Leblanc P_1	Desgagné et coll. (2009)
36	Test de Desgagné-Micheaux-Leblanc P_2	Desgagné et coll. (2009)
37	Test de Desgagné-Micheaux-Leblanc P_3	Desgagné et coll. (2009)
38	Test de Desgagné-Micheaux-Leblanc P_4	Desgagné et coll. (2009)
39	Test de Desgagné-Micheaux-Leblanc P_5	Desgagné et coll. (2009)
40	Test de Desgagné-Micheaux-Leblanc	Desgagné et coll. (2013)
41	Test de Spiegelhalter S	Spiegelhalter (1977)

3.2.2.2. *Tests pour une distribution de Laplace*

Le tableau 3.3 présente une liste des tests pour une distribution de Laplace. Il contient également leur indices et les références pour les définitions de ces tests.

TABLE 3.3. Liste des tests pour une distribution de Laplace

	Test	Référence
42	Test de Anderson-Darling A^2	Yen et Moore (1988)
43	Test de Cramér-von Mises W^2	Yen et Moore (1988)
44	Test de Watson U^2	Puig et Stephens (2000)
45	Test de Kolmogorov-Smirnov $\sqrt{n}D$	Puig et Stephens (2000)
46	Test de Kuiper V	Puig et Stephens (2000)
47	Test de Meintanis $T_{n,a}^{(1)}$ - MO	Meintanis (2004)
48	Test de Meintanis $T_{n,a}^{(1)}$ - ML	Meintanis (2004)
		Continuer à la page suivante

TABLE 3.3 – Suite de la page précédente

	Test	Référence
49	Test de Meintanis $T_{n,a}^{(2)}$ - MO	Meintanis (2004)
50	Test de Meintanis $T_{n,a}^{(2)}$ - ML	Meintanis (2004)
51	Test de Choi-Kim $T_{m,n}^{V}$	Choi et Kim (2006)
52	Test de Choi-Kim $T_{m,n}^{E}$	Choi et Kim (2006)
53	Test de Choi-Kim $T_{m,n}^{C}$	Choi et Kim (2006)
54	Test de Desgagné-Micheaux-Leblanc $\hat{G}_n$	Desgagné et coll. (2012)
55	Test de Rayner-Best V_3	Rayner et Best (1989)
56	Test de Rayner-Best V_4	Rayner et Best (1989)
57	Test de Langholz-Kronmal K_1	Langholz et Kronmal (1991)
58	Test de Kundu T	Kundu (2005)
59	Test de Gulati Z	Gulati (2011)
60	Test de Gel K	Gel (2010)
61	Test de Lafaye de Micheaux LM	Lafaye de Micheaux (2013)

3.2.2.3. *Tests d'uniformité*

Le tableau 3.4 présente une liste des tests d'uniformité, leur indices ainsi que les références associées.

TABLE 3.4. Liste des tests d'uniformité

	Test	Référence
63	Test de Kolmogorov D_n	Kolmogorov (1933)
64	Test de Cramér-von Mises W_n^2	Anderson et Darling (1954)
65	Test de Anderson-Darling A_n^2	Anderson et Darling (1954)
66	Test de Durbin C_n	Durbin (1969)
67	Test de Kuiper K_n	Brunk (1962)
68	Test de Hegazy-Green T_1	Hegazy et Green (1975)
69	Test de Hegazy-Green T_2	Hegazy et Green (1975)
70	Test de Greenwood $G(n)$	Greenwood (1946)

Continuer à la page suivante

TABLE 3.4 – Suite de la page précédente

	Test	Référence
71	Test de Quesenberry-Miller Q	Quesenberry et Miller (1977)
72	Test de Read-Cressie $2nI^{\lambda}$	Read et Cressie (1988)
73	Test de Moran $M(n)$	Moran (1951)
74	Test de Cressie $L_n^{(m)}$	Cressie (1978)
75	Test de Cressie $S_n^{(m)}$	Cressie (1979)
76	Test de Vasicek $H(m, n)$	Vasicek (1976)
77	Test de Swartz $A^*(n)$	Swartz (1992)
78	Test de Morales $D_{n,m}(\phi_\lambda)$	Morales et coll. (2003)
79	Test de Pardo $E_{m,n}$	Pardo (2003)
80	Test de Marhuenda $T^{\lambda}_{n,m}$	Marhuenda et coll. (2005a)
81	Test de Zhang Z_A	Zhang (2002)
82	Test de Zhang Z_C	Zhang (2002)

3.3. FONCTIONNALITÉS DU PROGICIEL **POWER**

Pour faciliter l'utilisation du progiciel, nous avons créé une interface d'utilisateur (GUI). Pour le lancer, il faut taper dans la console de R :

```
require(PoweR)
power.gui()
```

Vous devriez voir apparaître la fenêtre présentée à la figure 3.2. Cette interface est composée de 5 onglets différents :

(1) Generate sample : générer des échantillons aléatoires d'une loi présente dans le progiciel.

(2) Compute statistic : calculer la valeur de la statistique de test pour une valeur donnée de test.

(3) Critical values : calcul des valeurs critiques de plusieurs statistiques de tests.

(4) Compute power : calculs de puissances des tests.

(5) Examples : reproduire des exemples dans certains articles publiés.

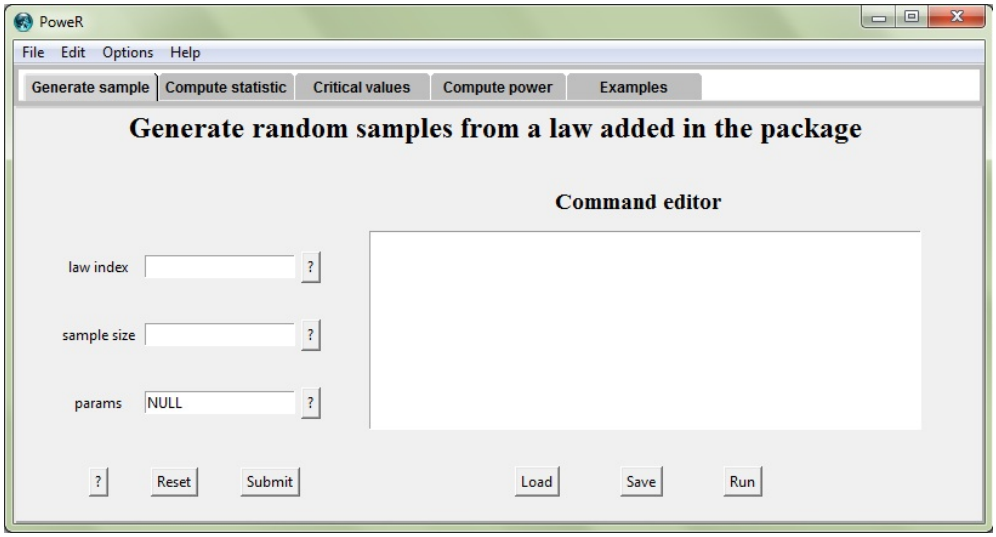

FIGURE 3.2. Interface graphique du progiciel PoweR.

La fenêtre sous chaque onglet est divisée en deux parties : la partie gauche et la partie droite

– La partie gauche contient des champs à remplir (par exemple, les numéros des lois, des tests, ou encore des paramètres, ect), un bouton `Reset` pour remettre les champs à leur valeur initiale et un bouton `Submit` qui sert à envoyer les commandes à un éditeur de texte dans la partie droite de la fenêtre. Par ailleurs, les boutons `?` nous donnent de l'aide pour remplir les champs, comme cela est illustré sur la figure 3.3.

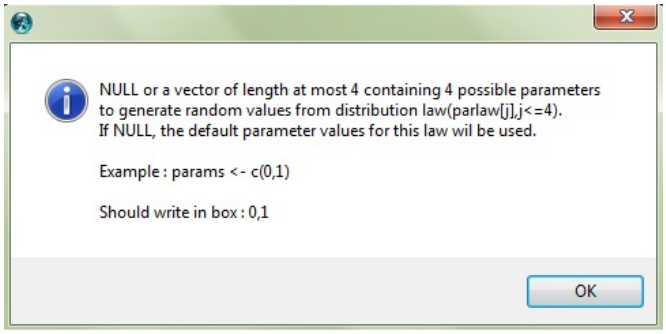

FIGURE 3.3. Fenêtre d'aide via le bouton `?`.

Le bouton `?` tout en bas à gauche nous renvoie à la documentation de la fonction correspondant à cet onglet.

– La partie droite contient un éditeur de texte qui reçoit les commandes de la partie gauche. Il est ensuite facile de les modifier, de les sauvegarder avec le bouton `Save`, de charger un fichier déjà créé par le bouton `Load`, et d'exécuter les commandes avec le bouton `Run`.

Remarque 3.3.1.

Nous avons codé l'interface de ***PoweR*** *avec* ***tcltk*** *(voir Dalgaard (2001, 2002)), un progiciel intégré dans la base du logiciel R. Les utilisateurs n'ont donc pas besoin d'installer d'autres packages supplémentaires.*

3.3.1. Génération de données

Le progiciel **PoweR** permet de générer des données selon diverses lois de probabilité. Plus précisément, chaque loi est codée comme une fonction C/C++ séparée et elle sera appelée par la fonction `gensample()` pour la génération.

```
gensample(law.index,n,params)
```

où :

– `law.index` : indice d'une loi donné par la fonction `getindex()` ou par le Tableau 3.1 à la page 24.

– `n` : nombre d'observations à générer.

– `params` : vecteur des paramètres de la loi. La longueur de ce paramètre ne doit pas dépasser 4. La valeur par défaut de `params` est `NULL`.

La sortie de cette fonction nous donne l'échantillon généré, le nom de la loi choisie, ainsi que ses paramètres.

Exemple 3.3.1.

Supposons que nous voulions générer un échantillon de taille 10 provenant de la loi $\mathtt{Laplace}(0,1)$. *La commande à utiliser sera alors :*

```
# law.index = 1 : loi Laplace
# n = 10 : échantillon généré de taille 10
# params = c(0,1) : les paramètres de la Laplace sont 0 et 1
> gensample(law.index=1,n=10,params=c(0,1))
$sample
 [1]  2.2540146 -0.1748581 -0.7996351 -0.6973653 -0.1489884
```

```
 [6] -1.4606257  0.3161178  2.9227767 -1.7754742 -0.8113752
$law
 [1] "Laplace(mu,b)"
$params
 [1] 0 1
```

De plus, nous pouvons toujours utiliser le GUI de ***PoweR*** *de la façon suivante, comme indiqué sur la figure 3.4 :*

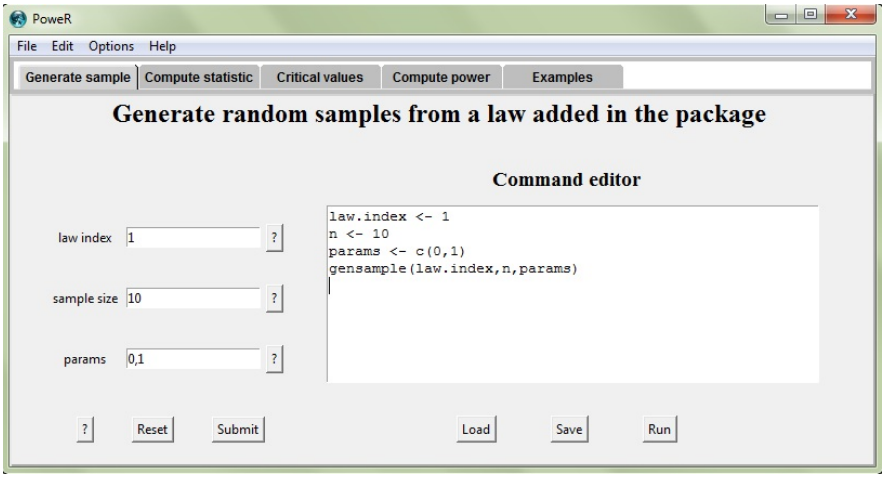

FIGURE 3.4. Génération de données d'une loi $\mathtt{Laplace}(0,1)$.

3.3.2. Calcul de la statistique d'un test

Une autre fonctionnalité du package **PoweR** est de permettre le calcul de la statistique de tests d'adéquation. Plus précisément, chaque test est codé comme une fonction C/C++ séparée et celle-ci sera appelée par la fonction `statcompute()` :

```
statcompute(stat.index,x,level,critvalL,critvalR,alter,paramstat)
```

où :

- `stat.index` : indice d'un test donné par la fonction `getindex()`, ou vous pouvez voir les tableaux 3.2, 3.3, et 3.4 pour une liste complète des tests ajoutés dans le package.
- `x` : échantillon (généré ou réel) à partir duquel calculer la statistique du test.
- `level` : vecteur des niveaux désirés pour le calcul de la statistique du test.
- `critvalL` : `NULL` ou un vecteur de valeurs critiques à gauche.
- `critvalR` : `NULL` ou un vecteur de valeurs critiques à droite.

– `alter` : type de test,

$$\texttt{alter} = \begin{cases} 0 & \text{pour un test bilatéral avec } \mathcal{R}_\alpha = \{T_n < c_L(\alpha)\} \cup \{T_n > c_R(\alpha)\}; \\ 1 & \text{pour un test unilatéral avec } \mathcal{R}_\alpha = \{T_n < c_\alpha\}; \\ 2 & \text{pour un test unilatéral avec } \mathcal{R}_\alpha = \{T_n > c_\alpha\}; \\ 3 & \text{pour un test bilatéral avec } \mathcal{R}_\alpha = \{T_n > c_\alpha\}; \\ 4 & \text{pour un test bilatéral avec } \mathcal{R}_\alpha = \{T_n < c_\alpha\}. \end{cases}$$

– `paramstat` : vecteur des paramètres du test. Si `NULL`, les valeurs des paramètres par défaut pour ce test seront allouées.

La sortie de cette fonction nous donne la valeur de la statistique du test, la valeur-p (s'il y a lieu) du test, la valeur de `decision` (0 : nous « acceptons » H_0 et 1 : nous rejetons H_0), la valeur de `alter`, ainsi que les paramètres du test.

Exemple 3.3.2.

Supposons que nous voulions évaluer la normalité d'un échantillon de taille 50 suivant la loi $\mathrm{Normale}(0,1)$ *par le test de normalité de Shapiro-Wilk Shapiro et Wilk (1965). Pour cela, nous commençons par générer un échantillon de taille 50 suivant la loi* $\mathrm{Normale}(0,1)$ *; les données sont sauvegardées dans* x*. Ensuite, l'indice du test de normalité de Shapiro-Wilk est 21, celui-ci peut se trouver à l'aide de la fonction* `getindex()` *ou du tableau 3.2. Il nous reste à fixer les valeurs critiques tandis que* `paramstat` *est fixé à la valeur* `NULL`*. Les niveaux utilisés sont 0.05 et 0.01. Les commandes à utiliser seront alors :*

```
# On commence par générer un échantillon de taille 50
# suivant la loi Normale(0,1)
> x <- gensample(2,50,c(0,1))$sample
# On calcule la statistique du test de Shapiro-Wilk
# et sa valeur-p correspondante
> statcompute(21,x,level=c(0.05,0.01),critvalL=NULL,
              critvalR=NULL,alter=4,paramstat=NULL)
$statistic
[1] 0.9774978
$pvalue
[1] 0.4519347
$decision
[1] 0 0
$alter
```

```
[1] 4
$paramstat
[1] NA
```

Nous observons que la statistique du test vaut 0.977 et la valeur-p est égale à 0.451. Il est alors impossible d'affirmer, de façon significative, que les données ne sont pas normales (la valeur de `decision` *est 0 aux niveaux de 0.05 et 0.01). Veuillez noter aussi que la valeur de* `alter` *est égale à 4, c'est-à-dire que le test est bilatéral et que nous rejetons* H_0 *seulement pour les petites valeurs de la statistique du test. Aucun paramètre n'a été utilisé (*`paramstat` *est* `NA`*). De plus, nous pouvons toujours utiliser le GUI de* ***PoweR*** *de la façon suivante :*

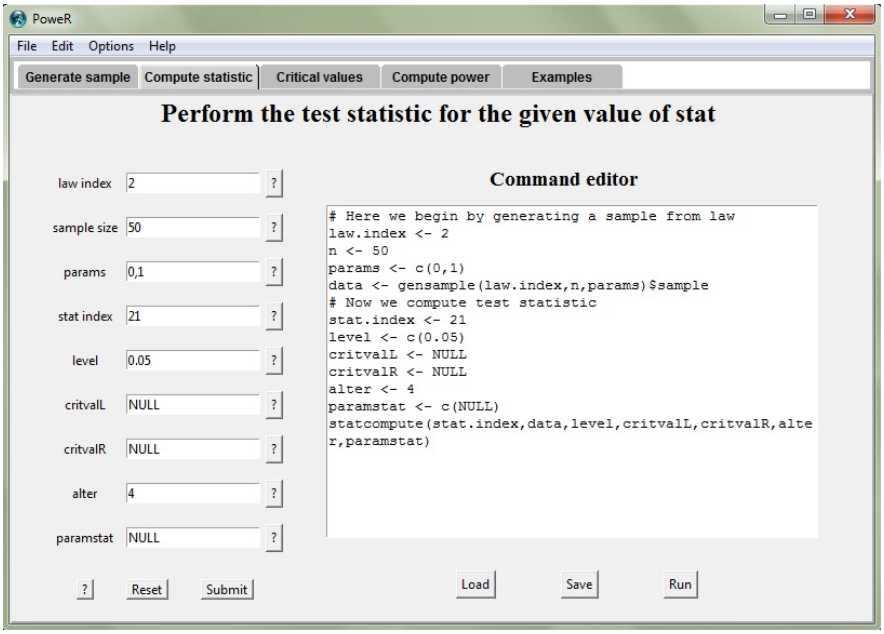

FIGURE 3.5. Calcul de la statistique du test de Shapiro-Wilk pour des données suivant la loi $\mathtt{Normal}(0,1)$.

Remarque 3.3.2.

- *Dans l'exemple ci-dessus, nous avons simulé un échantillons de taille 50 suivant la loi* $\mathtt{Normale}(0,1)$ *avec la fonction* `gensample()` *du package* ***PoweR****. Notez que vous pouvez utiliser n'importe quel générateur de données (par exemple :* `rnorm(50)`*), ou encore des données réelles.*
- *Par ailleurs, nous avons mis les valeurs critiques (*`critvalL` *et* `critvalR`*) à la valeur* `NULL`*. Il est conseillé de mettre des valeurs critiques dans la commande*

si vous les connaissez. Sinon, les fonctions `compquant()` *et* `many.crit()` *dans notre progiciel* ***PoweR*** *permettent de calculer les quantiles / valeurs critiques empiriques.*

3.3.3. Calcul des quantiles empiriques

Une des fonctions les plus importantes du progiciel **PoweR** est de permettre le calcul des quantiles / valeurs critiques empiriques de tests d'adéquation. Chaque test est codé comme une fonction C/C++ séparée et elle sera appelée par les fonctions `compquant()` et `many.crit()`. La fonction `compquant()` est un cas particulier de la fonction `many.crit()` puisqu'elle calcule les quantiles empiriques pour un seul test et une seule taille d'échantillon.

```
many.crit(law.index,M,vectn,stat.indices,level,
                              alter,parlaw,parstat)
```

où :

- `law.index` : indice d'une loi donnée par la fonction `getindex()` ou par le tableau 3.1 à la page 24.
- `M` : nombre de répétitions de Monte Carlo à utiliser.
- `vectn` : vecteur des tailles d'échantillons souhaitées à générer.
- `stat.indices` : vecteur d'indices des tests. Vous pouvez trouver les indices par la fonction `getindex()`, ou par les tableaux 3.2, 3.3, et 3.4 qui donnent une liste complète des tests ajoutés dans le package.
- `level` : vecteur des niveaux désirés pour le calcul des valeurs critiques des tests.
- `alter` : type de test,

$$\texttt{alter} = \begin{cases} 0 & \text{pour un test bilatéral avec } \mathcal{R}_\alpha = \{T_n < c_L(\alpha)\} \cup \{T_n > c_R(\alpha)\}; \\ 1 & \text{pour un test unilatéral avec } \mathcal{R}_\alpha = \{T_n < c_\alpha\}; \\ 2 & \text{pour un test unilatéral avec } \mathcal{R}_\alpha = \{T_n > c_\alpha\}; \\ 3 & \text{pour un test bilatéral avec } \mathcal{R}_\alpha = \{T_n > c_\alpha\}; \\ 4 & \text{pour un test bilatéral avec } \mathcal{R}_\alpha = \{T_n < c_\alpha\}. \end{cases}$$

- `parlaw` : NULL ou un vecteur contenant au plus 4 paramètres pour la loi des valeurs à générer.

– `parstat` : liste de valeurs de paramètres pour chaque statistique de test. Les noms de la liste doit être `statj`, `j` prise en `stat.indices`. Si NULL, les valeurs des paramètres par défaut pour cette statistique du test seront alloués.

La sortie de cette fonction nous donne une liste dont chaque composant contient le nom de la statistique du test, les valeurs critiques calculées correspondant aux niveaux et aux tailles d'échantillons.

Exemple 3.3.3.

Supposons que nous voulions calculer les valeurs critiques des tests d'uniformité de Quesenberry-Miller Quesenberry et Miller (1977) et de Pardo Pardo (2003) aux niveaux de 0.05 et de 0.01. Pour cela, nous avons d'abord simulé des observations suivant la loi Uniforme$(0,1)$*, donc la valeur de* `law.index` *vaut 7 (voir le tableau 3.1). Ensuite, les indices correspondant aux deux tests d'uniformité de Quesenberry-Miller et de Pardo sont 71 et 79, ceux-ci peuvent se trouver à l'aide de la fonction* `getindex()` *ou du tableau 3.4. À partir de ces observations, la méthode de simulation de Monte Carlo (voir section 2.3) est utilisée pour calculer des valeurs critiques. Il nous reste alors à fixer les valeurs des paramètres de la fonction* `many.crit()`*. Les tailles d'échantillon à simuler sont égales à 50 et 100 ; les niveaux sont 0.05 et 0.01 ; les valeurs de* `parlaw` *de la loi* Uniforme *et* `model` *sont* `NULL` *; la valeur du paramètre* `m` *du test de Pardo vaut 2 alors que le test de Quesenberry-Miller n'a pas de paramètre. Les commandes à utiliser seront alors :*

```
# law.index = 7 : loi Uniforme
# M = 100000 : nombre de répétitions de Monte Carlo
# vectn = c(50,100) : tailles d'échantillon à simuler
# stat.indices = c(71,79) : Quesenberry-Miller et Pardo
# level = c(0.05,0.01) : niveaux utilisés
# parstat=2 pour le test de Pardo
# parlaw = model = NULL : valeurs par défaut sont utilisées
> (critvalues <- many.crit(law.index=7,M=100000,vectn=c(50,100),
                 stat.indices=c(71,79),level=c(0.05,0.01),
                 alter=list(stat71=3,stat79=3),parlaw=NULL,
                 parstat=list(stat71=NA,stat79=2),model=NULL))
$stat71
    n level param critL      critR
1  50  0.05    NA    NA 0.06755384
2 100  0.05    NA    NA 0.03312536
```

```
3  50  0.01    NA    NA 0.07488685
4 100  0.01    NA    NA 0.03549023
$stat79
    n level param critL    critR
1  50  0.05     2    NA 1.866913
2 100  0.05     2    NA 1.641259
3  50  0.01     2    NA 2.325975
4 100  0.01     2    NA 1.898914
```

Nous observons que le nombre de répétitions `M` *est égal à 100000. Veuillez noter aussi que les valeurs de* `alter` *des deux tests sont égales à 3, c'est-à-dire que les tests sont bilatéraux et que nous rejetons* H_0 *seulement pour les grandes valeurs. De plus, nous pouvons utiliser le GUI de* ***PoweR*** *de la manière suivante :*

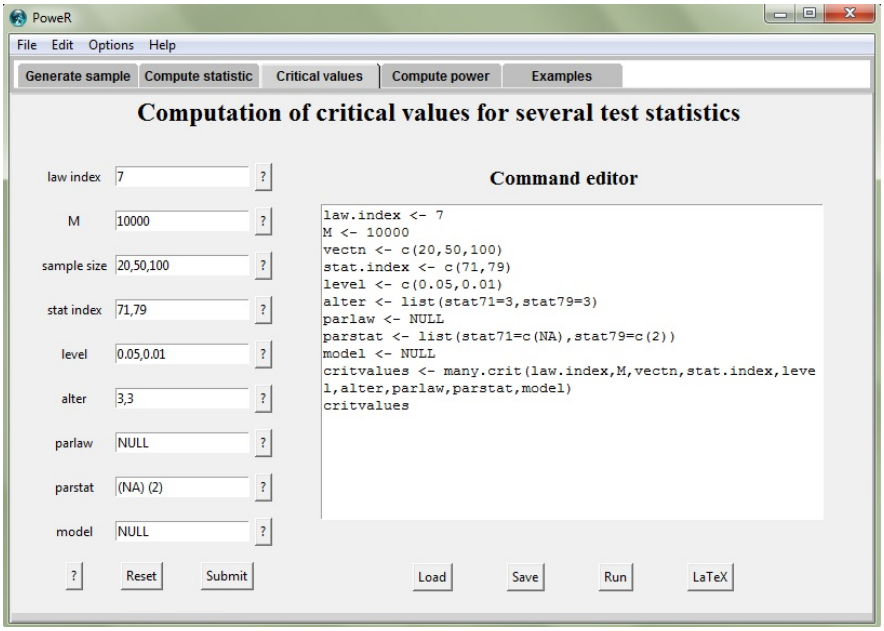

FIGURE 3.6. Calcul des valeurs critiques des tests d'uniformité de Quesenberry-Miller et de Pardo.

Remarque 3.3.3.

– *Si la valeur de* `parlaw` *est* `NULL`*, le progiciel choisit les paramètres par défaut, c'est-à-dire la loi* $\mathtt{Uniforme}(0,1)$*. Idem pour les valeurs de* `parstat` *et* `model`*.*

– *Dans l'exemple ci-dessus, les résultats pourrons être utilisés pour calculer la puissance des ces tests en appelant la fonction* `powcomp.fast()` *dont nous parlons dans la section suivante.*
– *D'autre part, le bouton* `LaTeX` *en bas à droite de l'onglet* `Critical values` *permet de transformer les résultats obtenus en instructions LaTeX. Si vous préférez travailler dans la console de R, il vous suffit de taper :*

```
print.critvalues(critvalues,digits=3,latex.output=TRUE)
```

où `critvalues` *est le résultat obtenu par la fonction* `many.crit()` *dans l'exemple ci-dessus, et* `digits` *est le nombre de chiffres après la virgule. Le tableau 3.5 présente la sortie (après compilation) de la commande ci-dessus :*

TABLE 3.5. Valeurs critiques des tests Q et $E_{m,n}$

		Goodness-of-fit tests	
Sample size	*Level*	Q	$E_{m,n}$*(m=2)*
50	*0.05*	*0.068*	*1.867*
100	*0.05*	*0.033*	*1.641*
50	*0.01*	*0.075*	*2.326*
100	*0.01*	*0.035*	*1.899*

Des exemples détaillés seront présentés au Chapitre 4.

3.3.4. Calculs de puissances de tests

La fonctionnalité la plus importante du progiciel **PoweR** est le calcul de la puissance empirique de tests d'adéquation. Chaque test est codé comme une fonction C/C++ séparée et elle sera appelée par les fonctions `powcomp.fast()` où les valeurs critiques sont fournies par la fonction `many.crit()`.

```
powcomp.fast(vectn,M,law.indices,stat.indices,levels,critval,
                  alter,parlaws,parstats,latex.output,nbclus)
```

où :

– `vectn` : vecteur des tailles d'échantillons souhaitées à générer.
– `M` : nombre de répétitions de Monte Carlo à utiliser.

- `law.indices` : vecteur d'indices des lois alternatives. Vous pouvez trouver les indices par la fonction `getindex()`, ou par le tableau 3.1.
- `stat.indices` : vecteur d'indices des tests. Vous pouvez trouver les indices par la fonction `getindex()`, ou par les tableaux 3.2, 3.3, et 3.4 qui donnent une liste complète des tests ajoutés dans le package.
- `levels` : vecteur des niveaux désirés pour le calcul des valeurs critiques des tests et des puissances des tests.
- `critval` : liste des valeurs critiques pour chaque statistique de test. Les noms de la liste doivent être `statj`, où l'indice `j` est pris dans `stat.indices`. Les valeurs de `critval` peuvent être fournies par la fonction `many.crit()`.
- `alter` : type de test,

$$\texttt{alter} = \begin{cases} 0 & \text{pour un test bilatéral avec } \mathcal{R}_\alpha = \{T_n < c_L(\alpha)\} \cup \{T_n > c_R(\alpha)\}; \\ 1 & \text{pour un test unilatéral avec } \mathcal{R}_\alpha = \{T_n < c_\alpha\}; \\ 2 & \text{pour un test unilatéral avec } \mathcal{R}_\alpha = \{T_n > c_\alpha\}; \\ 3 & \text{pour un test bilatéral avec } \mathcal{R}_\alpha = \{T_n > c_\alpha\}; \\ 4 & \text{pour un test bilatéral avec } \mathcal{R}_\alpha = \{T_n < c_\alpha\}. \end{cases}$$

- `parlaws` : liste de valeurs de paramètres pour chaque loi à simuler. Les noms de la liste doivent être `lawj`, `j` pris dans `law.indices`. La longueur du vecteur `lawj` ne doit pas être supérieure à 4. Si `NULL`, les paramètres par défaut seront utilisés.
- `parstats` : liste de valeurs de paramètres pour chaque statistique de test. Les noms de la liste doivent être `statj`, `j` pris dans `stat.indices`. Si NULL, les valeurs des paramètres par défaut pour cette statistique du test seront alloués.
- `latex.output` : peut prendre la valeur `TRUE` ou `FALSE`. Si `TRUE`, les instructions LaTeX seront affichées pour construire des tableaux de puissances.
- `nbclus` : nombre de processeurs pour le calcul parallèle.

La sortie de cette fonction nous donne une liste contenant des tests utilisés, des alternatives choisies, ainsi que des puissances de ces tests, etc.

Exemple 3.3.4.

Supposons que nous voulions calculer les puissances des tests (pour une distribution de Laplace) de Cramér-von Mises Yen et Moore (1988), de Watson Puig et Stephens

(2000), d'Anderson-Darling Yen et Moore (1988), de Kolmogorov-Smirnov Puig et Stephens (2000) et de Kuiper Puig et Stephens (2000) au niveau de 0.05, et contre les alternatives Normale$(0,1)$, Cauchy$(0,1)$ *et* Logistic$(0,1)$. *Pour cela, nous commençons par calculer les valeurs critiques pour chaque test d'adéquation (voir la section de la fonction* `many.crit()`*) en utilisant la méthode de simulation de Monte Carlo (voir section 2.3). Ensuite, pour chaque test, nous calculons la statistique de test à partir d'un échantillon parmi les alternatives, et la comparons avec la région critique obtenue auparavant. Une simulation de Monte Carlo a été mise en place pour trouver les puissances de cinq tests d'adéquation. Il nous reste à fixer les paramètres de la fonction* `powcomp.fast()` *:*

- *le nombre de répétitions de Monte Carlo* `M` *vaut 50000 ;*
- *les tailles d'échantillon à simuler sont 20, 50 et 100 ;*
- *les indices des tests pour une distribution de Laplace de Cramér-von Mises, de Watson, d'Anderson-Darling, de Kolmogorov-Smirnov et de Kuiper sont 43, 44, 42, 45 et 46 respectivement. Ceux-ci peuvent se trouver à l'aide de la fonction* `getindex()` *ou du tableau 3.3 ;*
- *le niveau des tests est fixé à 5% ;*
- *les valeurs de* `parlaws`, `parstats` *et* `model` *sont NULL ;*
- *la valeur de* `nbclus` *vaut 1.*
- *les puissances sont en pourcentage.*

Les commandes à utiliser seront alors :

```
> M <- 50000
> vectn <- c(20,50,100)
> stat.indices <- c(43,44,42,45,46)
> level <- c(0.05)
> alter <- list(stat43=3,stat44=3,stat42=3,stat45=3,stat46=3)
> law.indices <- c(2,3,4)
> critval <- many.crit(law.index=1,M,vectn,stat.indices,
                   level,alter,parlaw=NULL,parstat=NULL)
> pow <- powcomp.fast(vectn,M,law.indices,stat.indices,level,
           critval,alter,parlaws=NULL,parstats=NULL,nbclus=1)
> print.power(pow,digits=0,latex.output=FALSE)
            law   n level $W^2$ $U^2$ $A^2$ $\\sqrt{n}D$ $V$
```

```
  1    Normal(0,1)   20   0.05      8     11      7            9   10
  2                  50   0.05     17     33     15           19   29
  3                 100   0.05     41     69     36           38   61
  4    Cauchy(0,1)   20   0.05     55     63     56           51   61
  5                  50   0.05     86     92     87           83   90
  6                 100   0.05     99    100     99           98   99
  7 Logistic(0,1)   20   0.05      7      7      6            7    7
  8                  50   0.05     11     17      9           12   15
  9                 100   0.05     19     36     15           20   32
```

Nous observons que les valeurs de `alter` *des 5 tests sont égales à 3, c'est-à-dire que les tests sont bilatéraux et que nous rejetons* H_0 *seulement pour les grandes valeurs. De plus, nous pouvons utiliser le GUI de* ***PoweR*** *de la manière suivante :*

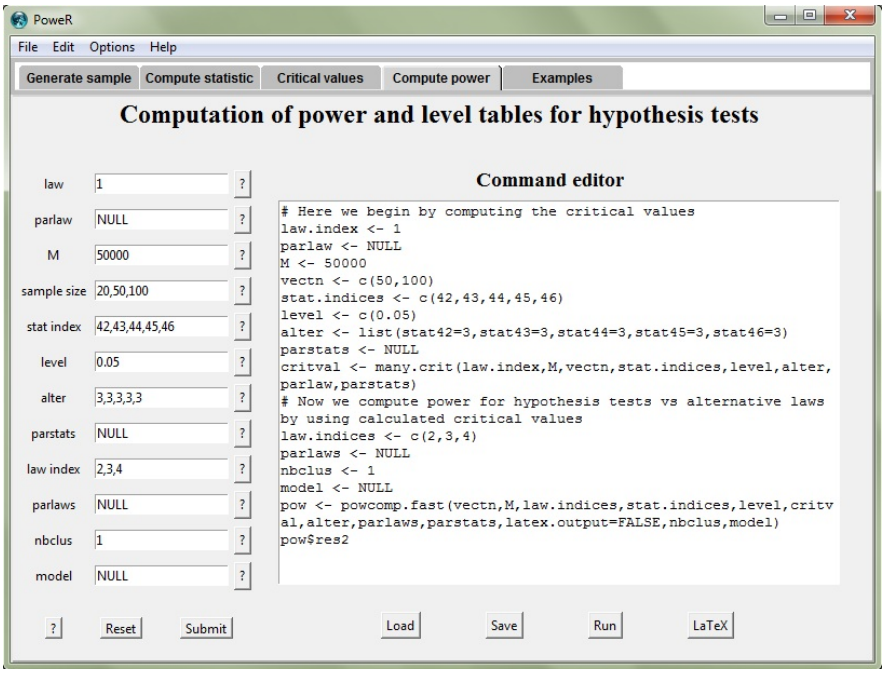

FIGURE 3.7. Calcul des puissances des tests pour une distribution de Laplace W^2, U^2, A^2, $\sqrt{n}D$ et V contre quelques alternatives.

Remarque 3.3.4.

– *Si la valeur de* `parlaws` *est* `NULL`*, le progiciel choisit les paramètres par défaut, c'est-à-dire la loi* $\mathrm{Normal}(0,1)$, $\mathrm{Cauchy}(0,1)$ *et* $\mathrm{Logistic}(0,1)$. *Idem pour la valeur de* `parstats`.

– *Les valeurs critiques* `critval` *sont fournies par la fonction* `many.crit()`. *Vous pouvez aussi utiliser vos propres valeurs.*
– *Pour diminuer le temps de calcul, il est conseillé de mettre la valeur de* `nbclus` *égale au nombre de processeurs de votre ordinateur moins 1 (pour le calcul parallèle).*
– *Le bouton* `LaTeX` *en bas à droite de l'onglet* `Compute power` *permet de transformer les résultats obtenus en instructions LaTeX. Si vous préférez travailler dans la console de R, il vous suffit de taper :*

```
print.power(pow,digits=0,latex.output=TRUE)
```

où `pow` *est le résultat obtenu dans l'exemple ci-dessus, et* `digits` *est le nombre de chiffres après la virgule. Le tableau 3.6 présente la sortie (après compilation) de la fonction* `latex.power()` *:*

TABLE 3.6. Puissances des tests (en %) pour une distribution de Laplace W^2, U^2, A^2, $\sqrt{n}D$ et V

			Goodness-of-fit tests				
Alternative	n	α	W^2	U^2	A^2	$\sqrt{n}D$	V
Normal(0,1)	*20*	*0.05*	*8*	*11*	*7*	*9*	*10*
	50	*0.05*	*17*	*33*	*15*	*19*	*29*
	100	*0.05*	*41*	*69*	*36*	*38*	*61*
Cauchy(0,1)	*20*	*0.05*	*55*	*63*	*56*	*51*	*61*
	50	*0.05*	*86*	*92*	*87*	*83*	*90*
	100	*0.05*	*99*	*100*	*99*	*98*	*99*
Logistic(0,1)	*20*	*0.05*	*7*	*7*	*6*	*7*	*7*
	50	*0.05*	*11*	*17*	*9*	*12*	*15*
	100	*0.05*	*19*	*36*	*15*	*20*	*32*

Des exemples détaillés seront présentés au Chapitre 4.

3.4. INFORMATIONS UTILES CONCERNANT L'UTILISATION DU PROGICIEL **POWER**

3.4.1. Liste des fonctions dans le progiciel PoweR

Vous trouvez dans le tableau 3.7 une liste complète des fonctions dans notre progiciel **PoweR**.

TABLE 3.7. Liste des fonctions du progiciel **PoweR**

Nom	Description
`calcFx`	Calculer Fxi de la matrice de valeurs-p obtenue par la fonction `many.pval()`
`comquant()`	Calculer des quantiles pour une seule statistique de test
`create.alter()`	Obtenir une liste de valeurs de alter des statistiques de test
`density()`	Fonctions de densité de lois de probabilité dans le progiciel
`gensample()`	Générer des échantillons aléatoires selon une loi dans le progiciel
`getindex()`	Obtenir des indices des lois et des statistiques
`getnbparstats()`	Obtenir des nombres de paramètres des statistiques
`graph()`	« P value plots », « P value discrepancy plots » et « Size-power curves »
`help.law()`	Ouvrir la documentation pour une loi à l'aide de son index
`help.stat()`	Ouvrir la documentation pour un test à l'aide de son index
`many.crit()`	Calculer des valeurs critiques de plusieurs statistiques de tests, pour plusieurs tailles d'echantillon et plusieurs valeurs de seuil de signification, pour une distribution donnée
`many.pval()`	Calculer des valeurs-p pour plusieurs statistiques de test
`plot.pvalue()`	« P value plots »
`plot.discrepancy()`	« P value discrepancy plots »
`plot.sizepower()`	« Size-power curves »
`powcomp.easy()`	Calculer la puissance de tests d'hypothèses (version « lente »)
`powcomp.fast()`	Calculer la puissance de plusieurs statistiques de tests, pour plusieurs tailles d'echantillon et plusieurs niveaux,
Continuer à la page suivante	

TABLE 3.7 – Suite de la page précédente

Nom	Description
	(version « rapide »)
`power.gui()`	Lancer le GUI du progiciel
`print.critvalues()`	Transformer les résultats de la fonction `many.crit()` en instructions LaTeX
`print.power()`	Transformer les résultats de la fonction `powcomp.fast()` en instructions LaTeX
`pvalueMC()`	Calculer une valeur de p pour un test statistique au moyen de simulations de Monte Carlo
`statcompute()`	Calculer la statistique d'un test ajouté dans le progiciel
`veriflaw()`	Vérifier le comportement d'un nouveau générateur de données aléatoires qui vient d'être rajouté dans le progiciel

3.4.2. Procédure pour rajouter une nouvelle loi ou un nouveau test dans le progiciel PoweR

Une fois construit un nouveau test d'adéquation, le chercheur veut pouvoir le comparer aux autres tests déjà existants dans la littérature. Il suffit donc de l'ajouter à notre progiciel et d'utiliser la fonction `powcomp.fast()` pour effectuer cette comparaison. Nous présentons dans cette section les étapes à mettre en oeuvre pour rajouter une nouvelle loi de probabilité ou un nouveau test dans le progiciel :

① Se rendre sur le site `http://cran.r-project.org/web/packages/` pour télécharger la progiciel **PoweR**_*version*.**tar.gz**.

② Rajouter la définition de la nouvelle loi ou du nouveau test dans le fichier **def-laws-stats.cpp** et la déclarer aussi dans **register.cpp**. Les deux fichiers se trouvent à "/Power/src/laws-stats/".

③ Créer le fichier **loi***j*.**cpp** dans "/Power/src/laws-stats/laws" ou **stat***j*.**cpp** dans "/Power/src/laws-stats/stats" qui contiendra le code source en C++ de cette loi ou de cette statistique de test, où *j* correspond au numéro de loi ou de statistique. Il est important de créer un fichier **pvalue***j*.**cpp** dans le dossier

"/Power/src/laws-stats/stats/pvalues" qui permettra le calcul de la valeur-p du test numéro *j*.

④ Recompiler la fonction **calcpuiss.cpp** dans le dossier "/Power/src/" en utilisant les commandes suivantes :

```
Sous Windows:
g++ -c calcpuiss.cpp -o calcpuiss.o ^
    -I"C:\Program Files\R\R-2.15.2\include"
g++ -shared -o calcpuiss.so calcpuiss.o ^
    -I"C:\Program Files\R\R-2.15.2\include" ^
    -L"C:\Program Files\R\R-2.15.2\bin\i386" -lR
```

```
Sous Linux:
g++ -c calcpuiss.cpp -o calcpuiss.o ^
    -I"/usr/lib/R/include"
g++ -shared -o calcpuiss.so calcpuiss.o ^
    -I"/usr/lib/R/include" -L"/usr/lib" -lR
```

⑤ Supprimer les deux fichiers objets **calcpuiss.o** et **calcpuiss.so** qui se trouvent dans le dossier "/Power/src/".

⑥ Recompiler le progiciel **PoweR**.

```
Sous Windows:
R CMD check PoweR
R CMD INSTALL --build PoweR
```

```
Sous Linux:
R CMD build PoweR
```

Remarque 3.4.1.

– *Après avoir recompilé le progiciel, il est fortement conseillé d'utiliser la fonction* `veriflaw()` *pour vérifier si la génération d'observations de la nouvelle loi se comporte correctement. De plus, un fichier d'aide correspondant à la nouvelle loi sera utile pour l'utilisateur.*

- *Si vous avez des problèmes de simulation avec votre nouvelle loi ou votre nouveau test, vous pouvez trouver dans l'annexe B des informations sur des outils de débogage usuels. Celles-ci peuvent vous aider à identifier des erreurs dans votre programmation.*
- *Si vous voulez compiler ce progiciel sous MAC, veuillez consulter les deux sites suivants :*
  `http://www.personality-project.org/R/makingpackages.html`
  `http://www.math.yorku.ca/~hkj/Software/macpac.pdf`.

Chapitre 4

APPLICATIONS

Dans le chapitre précédent, nous avons abordé les notions de base du progiciel **PowerR**, ainsi que ses fonctionnalités principales. Maintenant, nous vous présentons comment utiliser le progiciel pour reproduire les résultats de simulation de certains articles publiés. Autrement dit, vous pouvez l'utiliser comme un outil de vérification de calcul, de détection d'erreurs dans les tables, ou encore de comparaison entre des tests d'adéquation. À la dernière section de ce chapitre, nous allons aussi montrer comment utiliser notre progiciel pour évaluer la robustesse d'un test de Student à la non-normalité.

Nous présentons ici quelques codes permettant de reproduire des résultats d'articles. Vous trouverez l'ensemble complet des codes dans les annexes C, D et E.

4.1. REPRODUIRE LES RÉSULTATS D'ARTICLES PUBLIÉS

Puig et Stephens (2000) ont publié un article sur les tests d'adéquation à une distribution de Laplace. Dans cet article, le tableau 1 présente les valeurs critiques du test de Cramér-von Mises (voir aussi Yen et Moore (1988)) aux différents niveaux et aux différentes tailles d'échantillon. Nous retrouvons les valeurs du tableau 1 en exécutant les commandes suivantes :

```
1 # Reproduire le tableau 1 de Puig (2000) (page 419)
2 # Ici on considère n=1000 comme INFINI
3
4 # Indice de loi Laplace = 1
5 law.index <- 1
```

```
# Nombre de répétitions de Monte Carlo = 10000
M <- 10000
# Vecteur de tailles d'échantillon
vectn <- c(10,15,20,35,50,75,100,1000)
# Indice du test de Cramér-von Mises = 43
stat.indices <- 43
# Vecteur de niveaux
level <- c(0.50,0.25,0.10,0.05,0.025,0.01)
# Calcul des valeurs critiques du test W^2
table1 <- many.crit(law.index,M, vectn, stat.indices, level, alter=list(
    stat43=3),parlaw=NULL, parstat=NULL)
# Transfomer les résultats obtenus en instructions LaTeX
print.critvalues(table1, digits=3,latex.output=TRUE)
```

La fonction `print.critvalues()` transforme les résultats obtenus par la fonction `many.crit()` en instructions LaTeX, comme affiché dans le tableau 4.1 ci-dessous. Nous observons que nos valeurs sont identiques à celles de Puig et Stephens (2000).

TABLE 4.1. Valeurs critiques du test de Cramér-von Mises W^2 pour une distribution de Laplace

	Significance level					
Sample size	0.5	0.25	0.1	0.05	0.025	0.01
10	0.049	0.069	0.096	0.118	0.142	0.175
15	0.053	0.079	0.113	0.145	0.172	0.206
20	0.051	0.074	0.105	0.13	0.155	0.193
35	0.054	0.08	0.118	0.144	0.169	0.211
50	0.052	0.075	0.111	0.138	0.164	0.208
75	0.053	0.081	0.115	0.143	0.176	0.205
100	0.053	0.078	0.115	0.141	0.172	0.208
1000	0.054	0.079	0.116	0.143	0.173	0.213

Prenons un autre exemple avec la statistique du test de normalité W proposé par Shapiro et Wilk (1965). Dans cet article, le tableau 6 présente les valeurs critiques du test de Shapiro-Wilk pour différents niveaux et différentes tailles d'échantillon. Nous exécutons les codes suivants pour retrouver les valeurs du tableau 6 :

```
# Reproduire le tableau 6 de Shapiro (1965) (page 605)

# Indice de loi normale = 2
law.index <- 2
# Nombre de répétitions de Monte Carlo = 10000
M <- 10000
# Vecteur de tailles d'échantillon
vectn <- seq(4,50,1)
# Indice du test de Shapiro-Wilk = 21
stat.indices <- 21
# Vecteur de niveaux
level <- c(0.01,0.02,0.05,0.10,0.50,0.90,0.95,0.98,0.99)
# Test de Shapiro-Wilk est bilatéral et on rejette H_0 pour les
    petites valeurs de la statistique du test : alter=4
alter <- list(stat21=4)
# Calcul des valeurs critiques
table6 <- many.crit(law.index,M, vectn, stat.indices, level, alter,
    parlaw=NULL, parstat=NULL, model=NULL)
# Transformer les résultats obtenus en instructions LaTeX
print.critvalues(table6, digits=3, latex.output=TRUE)
```

Le tableau 4.2 affiche seulement les 10 premières lignes de la sortie des commandes ci-dessus. Il s'agit des quantiles empiriques du test de normalité de Shapiro et Wilk (1965). Nous observons qu'il y a une petite variation pour les petites valeurs de n par rapport à celles obtenues par Shapiro et Wilk. Lorsque n augmente, les deux tableaux sont approximativement les mêmes.

Nous opérons de la même manière pour le calcul de la puissance d'un test d'adéquation. Prenons l'exemple provenant de l'article de Meintanis (2004), où il a introduit de nouvelles statistiques de test $T_{n,a}^{(1)}$ et $T_{n,a}^{(2)}$ pour une distribution de Laplace. Les codes

TABLE 4.2. Valeurs critiques du test de normalité de Shapiro-Wilk W

	Significance level								
Sample size	0.01	0.02	0.05	0.1	0.5	0.9	0.95	0.98	0.99
4	0.691	0.718	0.761	0.799	0.913	0.983	0.992	0.996	0.998
5	0.714	0.732	0.775	0.814	0.915	0.976	0.983	0.991	0.995
6	0.719	0.753	0.793	0.828	0.919	0.972	0.981	0.988	0.991
7	0.744	0.766	0.808	0.843	0.925	0.971	0.979	0.986	0.99
8	0.767	0.79	0.827	0.853	0.927	0.971	0.978	0.985	0.988
9	0.777	0.799	0.836	0.861	0.931	0.97	0.978	0.984	0.987
10	0.78	0.811	0.845	0.872	0.935	0.971	0.978	0.984	0.988
11	0.803	0.824	0.856	0.878	0.938	0.972	0.978	0.984	0.987
12	0.812	0.83	0.864	0.884	0.942	0.973	0.979	0.983	0.986
13	0.82	0.844	0.87	0.891	0.944	0.974	0.979	0.985	0.987

suivants nous donnent les valeurs de puissance dans le tableau 4 de Meintanis pour $n = 50$. La valeur de α est égale à 0.10.

```
# Reproduire le tableau 4 de Meintanis (2004) (page 945)
# Pour n = 50 SEULEMENT

# Indice de loi Laplace = 1
law.index <- 1
# Nombre de répétitions de Monte Carlo = 10000
M <- 10000
# Taille d'échantillon = 50
vectn <- 50
# Seuil de significativité = 0.10
level <- 0.10
# Vecteur d'indices des tests : Cramér-von Mises = 43, Anderson-
    Darling = 42, Kolgomorov-Smirnor = 45, Watson = 44, Meintanis T1-
    ML = 48, Meintanis T2-ML = 50
# Les indices 48 et 50 sont répétés car ces tests prennent diffé
    rents paramètres
stat.indices <- c(43,42,45,44,48,50,48,50,48,50,48,50,48,50,48,50)
```

```
# Vecteur d'indices des alternatives : normale = 2, cauchy = 3,
    logistic = 4, uniforme = 7, exp = 35, gamma = 5, beta = 6,
    Laplace asymmétrique = 36
law.indices <- c(2,3,4,7,35,5,6,36,36)
# Tous les tests sont bilatéraux et on rejette H_0 seulement pour
    les grandes valeurs de la statistique du test : alter=3
alter <- list(stat43=3,stat42=3,stat45=3,stat44=3,
              stat48=3,stat50=3,stat48=3,stat50=3,
              stat48=3,stat50=3,stat48=3,stat50=3,
              stat48=3,stat50=3,stat48=3,stat50=3)
# Paramètres des tests
parstat <- list(stat43=NA,stat42=NA,stat45=NA,stat44=NA,
                stat48=0.5,stat50=0.5,stat48=1.0,stat50=1.0,
                stat48=2.0,stat50=2.0,stat48=4.0,stat50=4.0,
                stat48=5.0,stat50=5.0,stat48=10.0,stat50=10.0)
# Paramètres des alternatives
parlaws <- list(law2=c(0,1),law3=c(0,1),law4=c(0,1),
                law7=c(0,1),law35=c(1),law5=c(2,1),
                law6=c(2,2),law36=c(0,1,2),law36=c(0,1,3))
# Calcul des valeurs critiques des tests
critval <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat)
# Calcul des puissances des tests
table4 <- powcomp.fast(vectn,M,law.indices,stat.indices,level,
    critval,alter,parlaws,parstats=parstat,nbclus=1)
# Transformer les résultats obtenus en instructions LaTeX
print.power(table4,digits=0,latex.output=TRUE)
```

Le tableau 4.3 présente les puissances des tests de Cramér-von Mises W^2, d'Anderson-Darling A^2 (voir Yen et Moore (1988)), de Kolmogorov $\sqrt{n}D$, de Watson U^2 (voir Puig et Stephens (2000)), et de Meintanis $T^{(1)}_{n,a}$ et $T^{(2)}_{n,a}$ (voir Meintanis (2004)), contre plusieurs alternatives, avec $\alpha = 0.01$. Nous observons que nos valeurs sont identiques à celles de Meintanis.

TABLE 4.3. Puissances des tests (en %) de Cramér-von Mises W^2, d'Anderson-Darling A^2, de Kolmogorov $\sqrt{n}D$, de Watson U^2 et de Meintanis $T^{(1)}_{n,a}$ et $T^{(2)}_{n,a}$ contre plusieurs alternatives

			Goodness-of-fit tests															
Alternative	n	α	W^2	A^2	$\sqrt{n}D$	U^2	$T^{(1)}_{n,0.5}$	$T^{(2)}_{n,0.5}$	$T^{(1)}_{n,1}$	$T^{(2)}_{n,1}$	$T^{(1)}_{n,2}$	$T^{(2)}_{n,2}$	$T^{(1)}_{n,4}$	$T^{(2)}_{n,4}$	$T^{(1)}_{n,5}$	$T^{(2)}_{n,5}$	$T^{(1)}_{n,10}$	$T^{(2)}_{n,10}$
Normal(0,1)	50	0.1	31	27	31	47	12	50	25	51	48	20	30	14	17	13	11	13
Cauchy(0,1)	50	0.1	90	91	87	94	57	92	80	94	92	93	93	85	90	79	67	59
Logistic(0,1)	50	0.1	20	17	20	27	11	29	19	24	29	12	16	12	12	12	10	11
Uniform(0,1)	50	0.1	90	93	77	98	27	99	82	100	99	92	98	29	58	28	22	25
Exponential(1)	50	0.1	96	99	98	97	56	99	91	99	99	94	99	88	95	88	92	90
Gamma(2,1)	50	0.1	72	86	68	76	18	86	54	84	86	68	84	59	71	60	68	66
Beta(2,2)	50	0.1	62	63	54	83	13	89	45	94	87	57	78	21	33	20	16	18
ALaplace(0,1,2)	50	0.1	96	99	98	97	55	99	91	98	99	94	99	88	95	88	92	90
ALaplace(0,1,3)	50	0.1	96	99	99	97	54	99	91	99	99	94	99	88	95	87	92	90

4.2. DÉTECTER DES ERREURS DANS LES RÉSULTATS PUBLIÉS

À côté des articles où les résultats de simulation sont vraiment précis, nous avons trouvé pas mal d'erreurs dans les autres articles. Nous pouvons utiliser le progiciel **PoweR** comme un outil de détection d'erreur. En outre, nous distinguons les erreurs selon deux sources possibles.

4.2.1. Erreurs dues au manque de puissance des outils informatiques

Dans les années 80 et avant, les ordinateurs étaient moins puissants. Prenons un exemple de valeurs critiques provenant de l'article de Filliben (1975). Dans cet article, le tableau 1 présente les valeurs critiques du test de Filliben pour différents niveaux et différentes tailles d'échantillon.

```
# Reproduire le tableau 1 de Filliben (1995) (page 113)

# Indice de loi normale = 2
law.index <- 2
# Nombre de répétitions de Monte Carlo = 100000
M <- 100000
# Vecteur de tailles d'échantillon
vectn <- c(seq(4,50,1),seq(55,100,5))
# Indice du test de Filliben = 25
stat.indices <- 25
# Vecteur de niveaux
level <- c(0,0.005,0.01,0.025,0.05,0.10,0.25,
           0.50,0.75,0.90,0.95,0.975,0.99,0.995)
# Test de Filliben est bilatéral et on rejette H_0 seulement pour
    les petites valeurs de la statistique du test : alter = 4
alter <- list(stat25=4)
# Calcul des valeurs critiques
table1 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL,model=NULL)
# Transformer les résultats obtenus en instructions LaTeX
print.critvalues(table1,digits=3,latex.output=TRUE)
```

TABLE 4.4. Valeurs critiques du test de normalité de Filliben

	Significance level													
Sample size	0	0.005	0.01	0.025	0.05	0.1	0.25	0.5	0.75	0.9	0.95	0.975	0.99	0.995
4	0.785	0.811	0.823	0.845	0.867	0.893	0.931	0.958	0.979	0.992	0.996	0.998	0.999	1
5	0.734	0.804	0.824	0.855	0.878	0.902	0.934	0.96	0.977	0.987	0.992	0.995	0.997	0.998
6	0.727	0.816	0.837	0.865	0.887	0.91	0.94	0.962	0.977	0.986	0.99	0.993	0.996	0.997
7	0.706	0.828	0.848	0.875	0.897	0.917	0.944	0.964	0.978	0.986	0.99	0.992	0.995	0.996
8	0.711	0.84	0.859	0.885	0.904	0.923	0.948	0.966	0.979	0.986	0.99	0.992	0.994	0.995
9	0.696	0.85	0.869	0.892	0.911	0.929	0.952	0.968	0.98	0.987	0.99	0.992	0.994	0.995
10	0.686	0.859	0.877	0.9	0.917	0.934	0.955	0.97	0.981	0.987	0.99	0.992	0.994	0.995
11	0.749	0.868	0.885	0.906	0.922	0.938	0.957	0.972	0.982	0.987	0.99	0.992	0.994	0.995
12	0.719	0.874	0.892	0.912	0.927	0.941	0.96	0.973	0.982	0.988	0.99	0.992	0.994	0.995
13	0.751	0.883	0.897	0.917	0.931	0.944	0.962	0.975	0.983	0.988	0.991	0.992	0.994	0.995
14	0.713	0.888	0.903	0.921	0.935	0.947	0.963	0.976	0.984	0.989	0.991	0.993	0.994	0.995
15	0.758	0.895	0.907	0.925	0.938	0.95	0.965	0.977	0.984	0.989	0.991	0.993	0.994	0.995
16	0.76	0.897	0.912	0.929	0.941	0.952	0.967	0.978	0.985	0.989	0.991	0.993	0.994	0.995
17	0.794	0.903	0.915	0.932	0.943	0.954	0.968	0.979	0.986	0.99	0.992	0.993	0.994	0.995
18	0.753	0.907	0.92	0.935	0.945	0.956	0.97	0.98	0.986	0.99	0.992	0.993	0.995	0.995

Le tableau 4.4 à la page 58 affiche seulement les 15 premières lignes de la sortie des commandes ci-dessus. Il s'agit des quantiles empiriques du test de normalité de Filliben (1975). Nous observons que les valeurs de la première colonne du tableau 1 de l'article ne sont pas correctes. Pour les autres valeurs, nous retrouvons les mêmes résultats que ceux de Filliben.

Considérons un autre exemple de l'article de Martinez et Iglewicz (1981). Nous essayons de reproduire les quantiles empiriques du test de Martinez-Iglewicz dans cet article avec les commandes suivantes. Il faut aussi noter que nous avons pris $M = 80000$ répétitions pour toute la simulation où Martinez et Iglewicz ont choisi $M = 800000/n$.

```
# Reproduire le tableau 1 de Martinez (1981) (page 332)

# Indice de loi normale = 2
law.index <- 2
# Nombre de répétitions de Monte Carlo = 80000
M <- 80000
# Vecteur de tailles d'échantillon
vectn <- c(seq(10,50,5),seq(60,100,10),150,200,300)
# Indice du test de Martinez-Iglewicz = 32
stat.indices <- 32
# Vecteur de niveaux
level <- 1 - c(0.90,0.95,0.975,0.99)
# Test de Martinez-Iglewicz est bilatéral et on rejette H_0
    seulement pour les grandes valeurs de la statistique du test :
    alter = 3
alter <- list(stat32=3)
# Calcul des valeurs critiques
table1 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL,model=NULL)
# Transformer les résultats obtenus en instructions LaTeX
print.critvalues(table1,digits=3,latex.output=TRUE)
```

TABLE 4.5. Valeurs critiques du test de normalité de Martinez-Iglewicz

	Significance level			
Sample size	90%	95%	97.5%	99%
10	1.376	1.698	2.165	2.796
15	1.253	1.442	1.676	2.049
20	1.202	1.334	1.492	1.747
25	1.163	1.266	1.394	1.564
30	1.143	1.228	1.323	1.472
35	1.126	1.197	1.277	1.389
40	1.112	1.175	1.245	1.344
45	1.103	1.158	1.218	1.31
50	1.094	1.143	1.2	1.269
60	1.082	1.125	1.166	1.231
70	1.07	1.107	1.147	1.2
80	1.062	1.096	1.133	1.177
90	1.057	1.086	1.117	1.156
100	1.052	1.078	1.106	1.146
150	1.036	1.056	1.077	1.104
200	1.027	1.043	1.059	1.079
300	1.017	1.029	1.041	1.055

En comparant les valeurs dans le tableau 4.5 avec celles de Martinez et Iglewicz, nous remarquons les erreurs pour les petites valeurs de $\mathfrak{n}$ (10, 15, 20 et 25). Quand $\mathfrak{n}$ augmente, les deux tableaux sont proches.

Pour un exemple de puissances, prenons l'article d'Epps et Pulley (1983). Dans cet article, Epps et Pulley a proposé un nouveau test de normalité, qui a la puissance élevée contre de nombreuses distributions alternatives. Ce test utilise une intégrale pondérée du module au carré de la différence entre les fonctions caractéristiques de l'échantillon et de la distribution normale. Nous essayons de reproduire le tableau 2 de cet article qui présente les puissances des tests de normalité d'Epps-Pulley, de Shapiro et Wilk (1965)) et de Filliben (1975), contre plusieurs alternatives. Les codes à exécuter pour $\mathfrak{n} = 50$ sont les suivants :

```
# Reproduire le tableau 2 de Epps (1983) (page 725)
# Pour n = 50 SEULEMENT

# Indice de loi normale = 2
law.index <- 2
# Nombre de répétitions de Monte Carlo = 10000
M <- 10000
# Taille d'échantillon
vectn <- 50
# Vecteur d'indices des tests : Epps-Pulley = 31, Shapiro-Wilk = 21,
    Filliben = 25
stat.indices <- c(31,31,21,25)
# Seuil de significativité de 5%
level <- 0.05
# Le test d'Epps-Pulley est bilatéral et on rejette H_0 pour les
   grandes valeurs de la statistique du test : alter = 3; pour les 2
    autres tests : alter = 4
alter <- list(stat31=3,stat31=3,stat21=4,stat25=4)
# Vecteur d'indices des alternatives : beta = 6, JSB = 20, JSU = 17,
    chi-2 = 9, lognormal = 10, student-t = 8, Laplace = 1, cauchy =3
law.indices <- c(6,20,17,9,10,6,8,1,17,3)
# Paramètres des tests
parstats <- list(stat31=c(0.7),stat31=c(1.0),stat21=NA,stat25=NA)
# Paramètres des alternatives
parlaws <- list(law6=c(2,1),law20=c(1,1),law17=c(-1,2),
                law9=c(4),law10=c(0,1),law6=c(1,1),
                law8=c(10),law1=c(0,1),law17=c(0,1),law3=c(0,1))
# Nombre de processeurs = 1
nbclus <- 1
# Calcul des valeurs critiques
critval <- many.crit(law.index,M,vectn,stat.indices,level,alter,
   parlaw=NULL,parstat=parstats)
# Calcul des puissances des tests
table2 <- powcomp.fast(vectn,M,law.indices,stat.indices,level,
   critval,alter,parlaws,parstats,latex.output=FALSE,nbclus)
# Transformer les résultats obtenus en instructions LaTeX
```

```
print.power(table2, digits=0, latex.output=TRUE)
```

Nous trouvons les erreurs dans le tableau 4.6 des puissances de tests $T^*(0.7)$ et $T^*(1.0)$. Il est donc impossible de conclure que ces deux tests sont puissants contre les distributions symétriques.

TABLE 4.6. Puissances de tests (en %) $T^*(0.7)$, $T^*(1.0)$, W, et r

			Goodness-of-fit tests			
Alternative	n	α	$T^*(0.7)$	$T^*(1.0)$	W	r
Beta(2,1)	50	0.05	0	0	83	66
JSB(1,1)	50	0.05	0	0	79	68
JSU(-1,2,0,0.5)	50	0.05	3	3	20	26
$\chi^2(4)$	50	0.05	0	0	95	92
Lognormal(0,1)	50	0.05	0	0	100	100
Beta(1,1)	50	0.05	0	0	74	40
Student-t(10)	50	0.05	4	4	15	19
Laplace(0,1)	50	0.05	1	1	52	60
JSU(0,1,0,0.5)	50	0.05	3	3	21	27
Cauchy(0,1)	50	0.05	0	0	100	100

D'ailleurs vous trouverez des variations des résultats dans les articles de Shapiro et Wilk (1965), de Jarque et Bera (1987), etc. L'incertitude peut être causée par le nombre de répétitions parfois très petit ($M = 200$).

4.2.2. Erreur dans les équations ou dans la programmation

Commençons en prenant un exemple tiré de l'article de Choi et Kim (2006) où ils ont introduit la nouvelle statistique de test $T^E_{m,n}$. Dans le tableau 3 de cet article, Choi et Kim ont fait varier la valeur de m de 1 à 9. Nous avons fixé $m = 6$ et exécuté les commandes suivantes :

```
# Reproduire le tableau 3 de Choi (2006) (page 524)
# Pour m = 6 SEULEMENT

# Indice de loi Laplace = 1
```

```
law.index <- 1
# Nombre de répétitions de Monte Carlo = 100000
M <- 100000
# Vecteur de tailles d'échantillon
vectn <- c(seq(14,20,2),seq(25,50,5))
# Seuil de significativité de 5%
level <- 0.05
# Indice du test de Choi–Kim T_{m,n}^{E} = 52
stat.indices <- 52
# Ce test est bilatéral et on rejette H_0 seulement pour les petites
    valeurs de la statistique du test : alter = 4
alter <- list(stat52=4)
# Paramètre du test : m = 6
parstat <- list(stat52=6)
# Calcul des valeurs critiques
table3 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
   parlaw=NULL,parstat)
# Transformer les résultats obtenus en instructions LaTeX
print.critvalues(table3,digits=3,latex.output=TRUE)
```

TABLE 4.7. Valeurs critiques du test de Choi-Kim $T^E_{m,n}$ avec $\alpha = 0.05$

	Significance level
Sample size	$T^E_{6,n}$
14	5.706
16	6.218
18	6.648
20	6.987
25	7.619
30	8.025
35	8.314
40	8.529
45	8.7
50	8.848

Les valeurs dans le tableau 4.7 sont entièrement différentes de celles de Choi et Kim. Nous constatons soit une erreur dans la formule de la statistique du test, soit une erreur dans la programmation. Par conséquent, tous les résultats concernant la statistique de test $T^{E}_{m,n}$ dans cet article ne sont pas corrects.

Prenons maintenant le tableau 3 de l'article de Best et coll. (2008) où ils ont comparé les puissances des tests pour une distribution de Laplace, contre plusieurs alternatives. Les codes à utiliser sont les suivants :

```
# Reproduire le tableau 3 de Best (2008) (page 5341)

# Indice de loi Laplace = 1
law.index <- 1
# Nombre de répétitions de Monte Carlo = 10000
M <- 10000
# Vecteur de tailles d'échantillon
vectn <- c(50,100)
# Seuil de significativité de 5%
level <- 0.05
# Vecteur d'indices des tests : Best-Rayner V3 = 55, Best-Rayner V4
    = 56, Anderson-Darling = 42, Langholz-Kronmal = 57, Meintanis T_{
    n,a}^{(1)} = 47 et Choi-Kim T_{m,n}^{V} = 51
stat.indices <- c(55,56,42,57,47,51)
# Vecteur d'indices des alternatives : GEV = 28, Logistic = 4,
    Cauchy = 3, Tukey = 18, Normale = 2, Laplace = 1
law.indices <- c(28,4,3,18,2,1)
# Les tests de Best-Rayner sont bilatéraux et on rejette H_0
    seulement pour les grandes valeurs de la statistique du test :
    alter = 3; pour les autres tests : Anderson-Darling, Langholz-
    Kronmal, Meintanis : alter = 3 et Choi-Kim : alter = 4
alter <- list(stat55=3,stat56=3,stat42=3,stat57=3,stat47=3,stat51=4)
# Paramètres des alternatives
parlaws <- list(law28=c(0,1,0),law4=c(0,1),law3=c(0,1),law18=c(0.5),
    law2=c(0,1),law1=c(0,1))
# Calcul des puissances des tests
for (n in vectn) {
```

```
21    critval <- many.crit(law.index,M,n,stat.indices,level,alter,parlaw
          =NULL,parstat=NULL)
22    table3 <- powcomp.fast(n,M,law.indices,stat.indices,level,critval,
          alter,parlaws,parstats=NULL,nbclus=1)
23    # Transformer les résultats obtenus en instructions LaTeX
24    print(print.power(table3,digits=2,latex.output=TRUE))
25  }
```

TABLE 4.8. Puissances des tests (en %) de Best-Rayner V_3 et V_4, d'Anderson-Darling A^2, de Langholz-Kronmal K_1, de Meintanis $T^{(1)}_{n,a}$, et de Choi-Kim $T^{V}_{m,n}$ pour une distribution de Laplace

			Goodness-of-fit tests					
Alternative	n	α	V_3	V_4	A^2	K_1	$T^{(1)}_{n,a}$	$T^{V}_{m,n}$
GEV(0,1,0)	50	0.05	22.78	4.41	36.69	45.24	55.3	66.8
Logistic(0,1)	50	0.05	1.34	0.97	8.06	12.06	15.69	21.24
Cauchy(0,1)	50	0.05	41.34	89.24	85.88	92.24	91.92	59.38
Tukey(0.5)	50	0.05	0	0	46.31	74.9	79.09	98.12
Normal(0,1)	50	0.05	0.07	0	14.2	28.37	32.78	47.33
Laplace(0,1)	50	0.05	5.36	5.45	4.81	4.9	5.38	4.98
GEV(0,1,0)	100	0.05	44.55	4.3	78.21	83.55	90.02	91.52
Logistic(0,1)	100	0.05	1.18	0.68	14.92	27.48	30.39	25.85
Cauchy(0,1)	100	0.05	46.35	98.54	98.74	99.67	99.56	95.55
Tukey(0.5)	100	0.05	0	0	93.21	99.07	99.25	99.99
Normal(0,1)	100	0.05	0	0	35.85	67.26	65.4	65.71
Laplace(0,1)	100	0.05	5.14	5.34	4.83	4.85	4.87	5.06

Nous observons une différence entre nos résultats dans les deux colonnes V_3 et V_4 par rapport à ceux de Best et coll. (2008). Pourtant, comme nous avons trouvé les mêmes quantiles empiriques pour les statistiques de test V_3 et V_4 ainsi que les mêmes puissances pour les autres tests dans le tableau 4.8 ci-dessus, cela nous amène à conclure qu'il y a une erreur dans la partie programmation.

4.3. PROPOSER DE NOUVELLES COMPARAISONS

Dans cette section, nous proposons une comparaison entre les tests d'adéquation à une distribution de Laplace déjà codés dans notre progiciel **PoweR** contre diverses distributions alternatives. Les commandes à utiliser seront les suivantes :

```
rm(list=ls())
# Indice de loi Laplace = 1
law.index <- 1
# Nombre de répétitions de Monte Carlo = 100000
M <- 100000
# Vecteur de tailles d'échantillon
vectn <- c(20,50,100,200,500)
# Seuil de significativité de 5% et de 1%
level <- c(0.05,0.1)
# Vecteur d'indices des tests
stat.indices <- c(42:61)
# Vecteur d'indices des alternatives
law.indices <- c(1:37)
# Liste de alter
list.alter <- c(rep(3,50-42+1),rep(4,53-51+1),rep(3,60-54+1),0)
eval(parse(text=paste("alter <- list(",paste("stat",stat.indices,"=c
    (",list.alter,")",sep="",collapse=","),")",sep="")))
parstats <- NULL
parlaws <- NULL
# Calcul des valeurs critiques
critval <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=parstats)
# Calcul des puissances des tests de Laplace
table <- powcomp.fast(vectn,M,law.indices,stat.indices,level,critval
    ,alter,parlaws,parstats,latex.output=FALSE,nbclus=1)
# Transformer les résultats obtenus en instructions LaTeX
save(table,file="results-Laplace.RData")
print(latex.power(table,digits=0,latex.output=FALSE))
options(width=1000)
print(print.power(table,digits=0,latex.output=TRUE))
```

Le tableau 4.9 présente les résultats de simulation de puissances des tests de Laplace contre quelques distributions alternatives.

TABLE 4.9. Comparaison de puissances des tests (en %) de Laplace

Alternative	n	α	Goodness-of-fit tests																			
			A^2	W^2	U^2	$\sqrt{n}D$	V	$T_{n,a}^{(1)}$ (MO)	$T_{n,a}^{(1)}$ (ML)	$T_{n,a}^{(2)}$ (MO)	$T_{n,a}^{(2)}$ (ML)	$T_{m,n}^{V}$	$T_{m,n}^{E}$	$T_{m,n}^{C}$	$\hat{G}_n$	V_3	V_4	K_1	T	Z	K	LM
Laplace(0,1)	10	0.05	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
	20	0.05	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
	50	0.05	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
	100	0.05	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
	200	0.05	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
Normal(0,1)	10	0.05	5	5	5	5	5	6	8	7	9	9	4	11	3	1	1	5	12	5	1	7
	20	0.05	7	7	11	8	10	11	14	12	15	19	7	19	7	0	0	9	27	8	0	15
	50	0.05	15	17	34	20	29	31	36	32	38	46	14	43	35	0	0	29	67	31	0	47
	100	0.05	36	40	69	38	60	66	68	66	70	66	27	65	79	0	0	67	94	72	0	84
	200	0.05	79	81	96	70	92	96	95	96	96	88	51	86	99	0	0	97	100	98	0	99
Cauchy(0,1)	10	0.05	36	34	39	33	38	36	33	33	32	17	18	9	45	24	40	39	1	41	44	40
	20	0.05	56	54	63	51	61	61	56	60	55	36	55	31	71	33	62	64	0	68	68	67
	50	0.05	87	86	92	83	90	92	89	92	89	60	88	61	96	41	89	93	0	95	93	95
	100	0.05	99	99	100	98	99	100	99	100	99	96	99	93	100	45	99	100	0	100	100	100
	200	0.05	100	100	100	100	100	100	100	100	100	100	100	100	100	47	100	100	0	100	100	100
Logistic(0,1)	10	0.05	4	5	5	4	5	5	6	6	7	7	4	9	3	2	2	4	9	5	2	5
	20	0.05	5	6	7	6	7	7	9	8	9	11	6	12	5	2	1	6	16	6	1	9
	50	0.05	8	10	18	12	16	15	19	16	19	21	8	19	16	1	1	13	34	14	1	23
	100	0.05	15	19	36	20	32	30	35	31	36	26	12	25	39	1	1	28	56	33	0	47
	200	0.05	34	40	68	38	60	60	65	61	65	36	19	34	74	1	0	58	82	67	0	78
Gamma(2,1)	10	0.05	13	12	10	10	9	14	12	15	12	17	28	16	7	13	5	12	11	8	5	8

Continuer à la page suivante

TABLE 4.9 – Suite de la page précédente

Alternative	n	α	Goodness-of-fit tests																			
			A^2	W^2	U^2	$\sqrt{n}D$	V	$T_{n,a}^{(1)}$ (MO)	$T_{n,a}^{(1)}$ (ML)	$T_{n,a}^{(2)}$ (MO)	$T_{n,a}^{(2)}$ (ML)	$T_{m,n}^{V}$	$T_{m,n}^{E}$	$T_{m,n}^{C}$	$\hat{G}_n$	V_3	V_4	K_1	T	Z	K	LM
	20	0.05	27	21	21	19	19	34	27	35	26	49	54	48	8	19	6	27	18	10	6	12
	50	0.05	72	55	62	51	57	86	76	86	75	97	91	95	14	41	6	72	27	21	7	18
	100	0.05	99	92	95	93	94	100	99	100	99	100	100	100	22	74	6	98	36	44	11	27
	200	0.05	100	100	100	100	100	100	100	100	100	100	100	100	38	98	7	100	50	78	23	41
Beta(1,1)	10	0.05	11	12	15	10	13	20	24	22	27	29	8	31	9	0	0	15	29	10	0	15
	20	0.05	25	25	44	25	36	50	59	53	63	77	41	77	27	0	0	41	72	31	0	43
	50	0.05	82	77	95	63	88	96	98	96	98	100	94	100	91	0	0	93	100	95	0	95
	100	0.05	100	99	100	94	100	100	100	100	100	100	100	100	100	0	0	100	100	100	0	100
	200	0.05	100	100	100	100	100	100	100	100	100	100	100	100	100	0	0	100	100	100	0	100
Uniform(0,1)	10	0.05	11	12	16	10	13	20	25	23	28	29	9	31	9	0	0	15	29	10	0	15
	20	0.05	25	25	44	25	36	50	58	53	63	78	41	78	28	0	0	41	72	31	0	43
	50	0.05	82	77	95	63	88	96	98	97	98	100	94	100	91	0	0	94	100	95	0	95
	100	0.05	100	100	100	94	100	100	100	100	100	100	100	100	100	0	0	100	100	100	0	100
	200	0.05	100	100	100	100	100	100	100	100	100	100	100	100	100	0	0	100	100	100	0	100
Student-t(1)	10	0.05	36	34	39	33	38	36	33	33	32	17	18	9	45	24	40	39	1	41	44	41
	20	0.05	56	54	63	51	61	61	56	60	56	37	55	31	71	33	62	64	0	68	68	67
	50	0.05	87	86	92	83	91	92	89	92	89	60	88	61	96	41	89	93	0	95	93	95
	100	0.05	99	99	100	98	99	100	99	100	99	96	99	93	100	45	99	100	0	100	100	100
	200	0.05	100	100	100	100	100	100	100	100	100	100	100	100	100	47	100	100	0	100	100	100
			...																			

Avec notre progiciel, nous pouvons aussi simuler des tests avec des échatillons de grandes tailles ($n > 50$), par exemple pour les statistiques de test de Choi et Kim (2006), car dans l'article de Choi et Kim, ils ont seulement simulé avec n petits. Cela nous donne une idée de comment ces tests fonctionnent dans les différentes situations.

```
# Refaire le tableau 7 de Choi (2006) (page 525)
# Pour n >= 50
# Indice de loi Laplace = 1
law.index <- 1
# Nombre de répétitions de Monte Carlo = 50000
M <- 50000
# Vecteur de tailles d'échantillon
vectn <- c(50,100,200,300,500)
# Seuil de significativité de 5%
level <- 0.05
# Vecteur d'indices des tests : Choi-Kim T_{m,n}^{V} = 51, T_{m,n}^{
    C} = 53, T_{m,n}^{E} = 52, Cramér-von Mises = 43, Watson = 44,
    Anderson-Darling = 42, Kolgomorov-Smirnov = 45, Kuiper = 46
stat.indices <- c(51,53,52,43,44,42,45,46)
# Vecteur d'indices des alternatives : chi^2 = 9, lognormal = 10,
    Weibull = 11, GEV = 28
law.indices <- c(9,10,11,28)
# Liste de alter
alter <- list(stat51=4,stat53=4,stat52=4,stat43=3,stat44=3,stat42=3,
    stat45=3,stat46=3)
# Paramètres des alternatives
parlaws <-list(law9=c(2),law10=c(0,0.5),law11=c(2,1),law28=c(0,1,0))
# Calcul des valeurs critiques
critval <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL)
# Calcul des puissances des tests
toto <- powcomp.fast(vectn,M,law.indices,stat.indices,level,critval,
    alter,parlaws,parstats=NULL,nbclus=1)
# Transformer les résultats obtenus en instructions LaTeX
print.power(toto,digits=1,latex.output=TRUE)
```

Le tableau 4.10 ci-dessous montre les puissances des tests de Choi et Kim (2006), de Cramer-von Mises, de Watson, d'Anderson-Darling (voir Yen et Moore (1988)), de Kolgomorov-Smirnov et de Kuiper (voir Puig et Stephens (2000)), contre les alternatives, en prenant les grandes valeurs de tailles d'échantillon. Nous constatons que, à partir de $n > 500$, les tests de Choi et Kim $T^V_{m,n}$, $T^C_{m,n}$ et $T^E_{m,n}$ donnent une puissance très faible, inversement pour les petites valeurs de n. Par conséquent, nous pouvons avoir une idée sur la validité de ces tests.

TABLE 4.10. Puissances des tests (en %) $T^V_{m,n}$, $T^C_{m,n}$, $T^E_{m,n}$, W^2, U^2, A^2, $\sqrt{n}D$ et V

			Goodness-of-fit tests							
Alternative	n	α	$T^V_{m,n}$	$T^C_{m,n}$	$T^E_{m,n}$	W^2	U^2	A^2	$\sqrt{n}D$	V
$\chi^2(2)$	50	0.05	100	100	100	88.7	93.2	97.2	95.8	95.8
	100	0.05	100	100	100	99.9	100	100	100	100
	200	0.05	100	100	100	100	100	100	100	100
	300	0.05	100	100	0	100	100	100	100	100
	500	0.05	0	0	0	100	100	100	100	100
Lognormal(0,0.5)	50	0.05	87.5	85.3	79.7	45.6	50.4	64	40.7	44.2
	100	0.05	99.7	99.2	97.2	83.5	87	96.9	81.9	82.3
	200	0.05	100	100	100	99.8	99.8	100	99.8	99.6
	300	0.05	100	100	0	100	100	100	100	100
	500	0.05	0	100	0	100	100	100	100	100
Weibull(2,1)	50	0.05	82.2	80	50	30.4	49.8	34.1	26.6	41.2
	100	0.05	98.7	98	82.3	69	87.3	78.4	53.5	77.9
	200	0.05	100	100	98.5	98.3	99.7	99.7	89.5	98.6
	300	0.05	100	100	0	100	100	100	99.2	100
	500	0.05	0	100	0	100	100	100	100	100
GEV(0,1,0)	50	0.05	64.5	62	47.5	28	35.5	37.7	22.2	29.6
	100	0.05	91.3	88.8	73.8	60.9	71	78.2	47.4	60.8
	200	0.05	99.7	99.5	94.3	95.2	97.1	99.5	86.3	93.1
	300	0.05	100	100	0	99.8	99.8	100	98.6	99.4
	500	0.05	0	0.7	0	100	100	100	100	100

4.4. ROBUSTESSE DU TEST T DE STUDENT À LA NON-NORMALITÉ

À l'aide du progiciel, nous pouvons facilement évaluer la robustesse d'un test à la non-normalité. Nous présentons d'abord une définition générale :

Définition 4.4.1. *Un test est robuste s'il reste valable alors que les hypothèses d'application ne sont pas toutes réunies. Ce peut être une taille d'échantillon un peu faible ou une loi de probabilité (loi normale pour les tests paramétriques par exemple) qui n'est pas satisfaite par les données.*

Le test t de Student pour un échantillon est à appliquer pour tester si un échantillon gaussien a une moyenne différente d'une valeur donnée. Le principe du test t de Student est le suivant : on veut déterminer si la valeur inconnue d'espérance μ d'une population de distribution normale et d'écart type σ non connu est différente d'une valeur de référence μ_0. Pour ce faire, on tire de cette population un échantillon de taille n dont on calcule la moyenne $\overline{x}$ et l'écart-type empirique s.

Considérons les hypothèses suivantes :

$$H_0 : \mu = \mu_0 \text{ versus } H_1 : \mu \neq \mu_0.$$

Sous l'hypothèse nulle,

- la distribution d'échantillonnage de la moyenne $\overline{x}$ est normale avec un écart type $\dfrac{s}{\sqrt{n}}$;
- et la statistique $T = \dfrac{\overline{x} - \mu_0}{s/\sqrt{n}}$ suit alors une loi de Student avec $n - 1$ degrés de liberté.

Si la valeur-p est suffisamment faible (ou si la statistique du test t se trouve dans la région critique), nous pouvons rejeter l'hypothèse nulle H_0.

Nous nous intéressons à la robustesse du test t à la non-normalité en termes de la

$$\begin{aligned} \texttt{Puissance} &= P(\text{décider } H_1 \mid H_1 \text{ vraie}) \\ &= P(|T| > c_{\alpha/2} \mid \mu \neq \mu_0). \end{aligned}$$

Pour cela, nous avons incorporé le test t de Student dans le progiciel **PoweR** (fichier **stat83.cpp**). Le choix des alternatives est le suivant : une distribution `Normale`, une distribution symétrique `Laplace`, une distribution asymétrique comme la loi `Gamma` ou la loi `Exponentielle` décentrée (Shifted Exponential). Les paramètres de chaque loi sont choisis afin que les moyennes soient égales et les variances soient aussi égales.

Ensuite, nous calculons les puissances du test t de Student contre ces alternatives en faisant varier la valeur de μ_0 autour de la moyenne des distributions. Dans cette étude, le test t est bilatéral, néanmoins dans le cas d'un test t unilatéral, le principe reste toujours le même. Le seuil de significativité est fixé à 5%.

Nous exécutons le code suivant dans R en considérant que le nombre de répétitions de simulation de Monte Carlo $M = 100000$ est suffisant. Le tableau 4.11 en page 74 présente la sortie, après compilation puis exécution de ces commandes.

```
# Nombre de répétitions de Monte Carlo vaut 100000
M <- 100000
# Vecteur de tailles d'échantillons souhaitées
vectn <- c(10,20,50,100,200)
# Seuil de significativité de 5%
level <- c(0.05)
# Indices du test t de Student dans le package
stat.indices <- c(83,83,83,83,83,83)
# Indices des alternatives : Normale=2, Laplace=1, Gamma=5, Shifted
    Exp.=12
law.indices <- c(2,1,5,12)
# Test-t est bilatéral : alter=0
alter <- list(stat83=0,stat83=0,stat83=0,stat83=0,stat83=0,stat83=0)
# Paramètres du test-t correspondant aux valeurs de mu
parstats <- list(stat83=c(1.0),stat83=c(1.01),stat83=c(1.02),stat83=
    c(1.05),stat83=c(1.10),stat83=c(1.20))
# Paramètres des alternatives
parlaws <- list(law2=c(1,sqrt(5)/2),law1=c(1,sqrt(5/2)/2),law5=c
    (0.8,0.8),law12=c(1-sqrt(5)/2,2/sqrt(5)))
# Calcul de puissances du test t de Student en utilisant des valeurs
    -p
table <- powcomp.fast(vectn,M,law.indices,stat.indices,level,critval
    =NULL,alter,parlaws,parstats,nbclus=1)
# Afficher les résultats obtenus en instructions LaTeX
print.power(table,digits=0,latex.output=TRUE)
```

TABLE 4.11. Puissance (en %) du test t de Student pour un échantillon

			Goodness-of-fit tests					
			μ_0	μ_0	μ_0	μ_0	μ_0	μ_0
Alternative	n	α	1.0	1.01	1.02	1.05	1.10	1.20
Normal(1,1.118)	10	0.05	5	5	5	5	6	8
	20	0.05	5	5	5	5	7	12
	50	0.05	5	5	5	6	10	24
	100	0.05	5	5	5	7	14	43
	200	0.05	5	5	6	10	24	71
Laplace(1,0.791)	10	0.05	4	4	4	5	5	8
	20	0.05	5	5	5	5	7	13
	50	0.05	5	5	5	6	10	25
	100	0.05	5	5	5	7	15	44
	200	0.05	5	5	6	10	25	71
Gamma(0.8,0.8)	10	0.05	11	12	12	13	16	21
	20	0.05	9	9	10	12	15	23
	50	0.05	7	7	8	10	16	31
	100	0.05	6	7	7	11	20	46
	200	0.05	5	6	7	13	28	70
SE(-0.118,0.894)	10	0.05	10	10	11	12	14	20
	20	0.05	8	9	9	11	14	22
	50	0.05	6	7	7	10	15	30
	100	0.05	6	6	7	10	19	45
	200	0.05	5	6	7	12	28	70

Nous observons que pour $\mu_0 = 1.0$ (la vraie moyenne des échantillons) le test t contre les alternatives symétriques respecte bien le seuil de significativité de 5%, même pour les petites valeurs de n. Pour les distributions asymétriques comme la loi Gamma et la loi Exponentielle décentrée, nous avons besoin d'une grande taille d'échantillon ($n \geq 200$). Cela démontre une robustesse du test t de Student limitée pour des distributions asymétriques.

De plus, plus l'écart entre μ et μ_0 augmente, plus la puissance du test t augmente. À partir de $\mu_0 = 1.05$, cela est encore plus visible à $n = 50$ pour les distributions alternatives. En outre, pour les grandes valeurs de n, la puissance du test t de Student est

identique pour les quatre alternatives considérées. Il y a donc une certaine robustesse du test t de Student pour les grandes valeurs de n.

Remarque 4.4.1.

Tous les exemples mentionnés dans ce chapitre, ainsi que beaucoup d'autres exemples se trouvent dans les annexes C, D et E. Vous pouvez aussi utiliser le GUI de notre progiciel, sous l'onglet `Examples`*, pour retrouver les résultats de certains articles publiés.*

CONCLUSION

Dans ce mémoire, nous avons d'abord fait un rappel concernant les termes importants de la notion de « recherche reproductible », ainsi que son importance dans la recherche computationnelle de nos jours. Ensuite, nous avons introduit un nouveau progiciel R, nommé **PoweR**, un outil de recherche reproductible pour faciliter les calculs de puissance de certains tests d'hypothèses au moyen de simulations de Monte Carlo. Il est également important de noter que notre progiciel permet de reproduire les résultats numériques d'articles publiés, de détecter les erreurs dans les résultats s'il y a lieu, et de proposer de nouvelles comparaisons. Enfin, le GUI de notre progiciel et la possiblité de transformer les résultats obtenus en instructions LaTeX rendent son usage plus facile.

Dans le futur, nous nous intéressons à élagir le progiciel **PoweR** en ajoutant d'autres tests d'adequation présents dans la littérature. Un autre chemin à considérer est de généraliser notre progiciel dans le cas d'un test d'ajustement pour les erreurs d'un modèle paramétrique. Supposons par exemple le modèle linéaire simple suivant :

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i \tag{1}$$

où ϵ_i est l'erreur, et les ϵ_i sont indépendants et identiquement distribués selon une loi de probabilité $\mathcal{L}$. Nous procédons comme suit :

- Générer les ϵ_i selon la loi de probabilité $\mathcal{L}$.
- Fabriquer les valeurs de y_i en utilisant l'équation (1). Les valeurs des β_i et du vecteur x_i seront fournies par l'utilisateur.
- Estimer les paramètres β_0 et β_1 par $\widehat{\beta}_0$ et $\widehat{\beta}_1$.
- Simuler les résidus estimés par l'equation suivante :

$$\widehat{\epsilon}_i = y_i - \widehat{\beta}_0 - \widehat{\beta}_1 x_i \tag{2}$$

– Calculer la statistique de test en utilisant les valeurs $\hat{\epsilon}_i$.

De plus, Kundu (2005) Kundu (2005) a introduit une statistique de test qui distingue une distribution normale de celle de la Laplace en utilisant le rapport des maximums de vraisemblance. Nous pourrions tenter de créer une nouvelle fonction dans notre progiciel permettant de choisir, entre plusieurs distributions, la distribution originale d'un échantillon de données observées.

Bibliographie

Anderson, T. W. et Darling, D. A. (1954). A test of goodness of fit. *Journal of the American Statistical Association*, **49**, 765–769.

Atkinson, A. C. (1982). The simulation of generalized inverse Gaussian and hyperbolic random variables. *Society for Industrial and Applied Mathematics. Journal on Scientific and Statistical Computing*, **3**, 502–515. URL http://dx.doi.org/10.1137/0903033.

Azzalini, A. (2005). The skew-normal distribution and related multivariate families. *Scandinavian Journal of Statistics. Theory and Applications*, **32**, 159–200. URL http://dx.doi.org/10.1111/j.1467-9469.2005.00426.x. With discussion by Marc G. Genton and a rejoinder by the author.

Barrio, E. d., Cuesta-Albertos, J., Matran, C. et Rodriguez-Rodriguez, J. (1999). Tests of goodness-of-fit based on the l_2-wasserstein distance. *The Annals of Statistics*, **27**, 1230–1239.

Bates, G. E. (1955). Joint distributions of time intervals for the occurrence of successive accidents in a generalized Pólya scheme. *Annals of Mathematical Statistics*, **26**, 705–720.

Best, D. J., Rayner, J. C. W. et Thas, O. (2008). Comparison of some tests of fit for the Laplace distribution. *Computational Statistics & Data Analysis*, **52**, 5338–5343. URL http://dx.doi.org/10.1016/j.csda.2008.05.023.

Bonett, D. G. et Seier, E. (2002). A test of normality with high uniform power. *Computational Statistics & Data Analysis*, **40**, 435–445. URL http://dx.doi.org/10.1016/S0167-9473(02)00074-9.

Bontemps, C. et Meddahi, N. (2005). Testing normality : a GMM approach. *Journal of Econometrics*, **124**, 149–186. URL http://dx.doi.org/10.1016/

j.jeconom.2004.02.014.

Brunk, H. D. (1962). On the range of the difference between hypothetical distribution function and Pyke's modified empirical distribution function. *Annals of Mathematical Statistics*, **33**, 525–532.

Brys, G., Hubert, M. et Struyf, A. (2008). Goodness-of-fit tests based on a robust measure of skewness. *Computational Statistics*, **23**, 429–442. URL http://dx.doi.org/10.1007/s00180-007-0083-7.

Cabaña, A. et Cabaña, E. M. (1994). Goodness-of-fit and comparison tests of the Kolmogorov-Smirnov type for bivariate populations. *The Annals of Statistics*, **22**, 1447–1459. URL http://dx.doi.org/10.1214/aos/1176325636.

Castillo, E., Hadi, A. S., Balakrishnan, N. et Sarabia, J. M. (2005). *Extreme value and related models with applications in engineering and science*. Wiley Series in Probability and Statistics. Hoboken, NJ : Wiley-Interscience [John Wiley & Sons].

Chambers, J. M., Mallows, C. L. et Stuck, B. W. (1976). A method for simulating stable random variables. *Journal of the American Statistical Association*, **71**, 340–344.

Chen, L. et Shapiro, S. S. (1995). An alternative test for normality based on normalized spacings. *Journal of Statistical Computation and Simulation*, **53**, 269–288.

Chen, Z., Barrenetxea, G. et Vetterli, M. (2011). Distributed successive refinement of multi-view images using broadcast advantage. *IEEE Transaction of Image Processing*.

Choi, B. et Kim, K. (2006). Testing goodness-of-fit for laplace distribution based on maximum entropy. *Statistics*, **40**, 517–531. URL http://www.tandfonline.com/doi/abs/10.1080/02331880600822473.

Coin, D. (2008). A goodness-of-fit test for normality based on polynomial regression. *Computational Statistics & Data Analysis*, **52**, 2185–2198. URL http://dx.doi.org/10.1016/j.csda.2007.07.012.

Coles, S. (2001). *An introduction to statistical modeling of extreme values*. Springer Series in Statistics. London : Springer-Verlag London Ltd.

Cressie, N. (1978). Power results for tests based on high order gaps. *Biometrika*, **65**, 214–218.

Cressie, N. (1979). An optimal statistic based on higher order gaps. *Biometrika*, **66**, 619–627. URL http://dx.doi.org/10.1093/biomet/66.3.619.

D'Agostino, R. B. (1971). An omnibus test of normality for moderate and large size samples. *Biometrika*, **58**, 341–348.

D'Agostino, R. B. et Pearson, E. S. (1973). Tests for departure from normality. Empirical results for the distributions of b_2 and $\sqrt{b_1}$. *Biometrika*, **60**, 613–622.

D'Agostino, R. B. et Pearson, E. S. (1974). Correction to : "Tests for departure from normality. Empirical results for the distributions of b_2 and $\sqrt{b_1}$" (Biometrika **60** (1973), 613–622). *Biometrika*, **61**, 647.

D'Agostino, R. B. et Stephens, M. A. (1986). *Goodness-of-Fit Techniques*. New York : Marcel Dekker.

Dalgaard, P. (2001). A Primer on the R-Tcl/Tk package. *R News*, **1/3**, 27–31.

Dalgaard, P. (2002). Changes to the R-Tcl/Tk package. *R News*, **2/3**, 25–27.

Desgagné, A., Lafaye de Micheaux, P. et Leblanc, A. (2009). P-kurtosis and goodness-of-fit tests for normality. Unpublished.

Desgagné, A., Lafaye de Micheaux, P. et Leblanc, A. (2012). Goodness-of-fit tests for the Laplace distribution. Unpublished.

Desgagné, A., Lafaye de Micheaux, P. et Leblanc, A. (2013). Test of normality against generalized exponential power alternatives. *Communications in Statistics - Theory and Methods*, **42**, 164–190. URL http://www.tandfonline.com/doi/abs/10.1080/03610926.2011.577548.

Donoho, D. L., Maleki, A., Rahman, I. U., Shahram, M. et Stodden, V. (2009). Reproducible research in computational harmonic analysis. *Computing in Science and Engineering*, **11**, 8–18. URL http://link.aip.org/link/?CSX/11/8/1.

Doornik, J. A. et Hansen, H. (2008). An omnibus test for univariate and multivariate normality*. *Oxford Bulletin of Economics and Statistics*, **70**, 927–939. URL http://dx.doi.org/10.1111/j.1468-0084.2008.00537.x.

Durbin, J. (1969). Tests for serial correlation in regression analysis based on the periodogram of least-squares residuals. *Biometrika*, **56**, 1–15.

Epps, T. W. et Pulley, L. B. (1983). A test for normality based on the empirical characteristic function. *Biometrika*, **70**, 723–726. URL http://dx.doi.org/10.

1093/biomet/70.3.723.

Feller, W. (1968). *An introduction to probability theory and its applications. Vol. I.* Third edition. New York : John Wiley & Sons Inc.

Feller, W. (1971). *An introduction to probability theory and its applications. Vol. II.* Second edition. New York : John Wiley & Sons Inc.

Filliben, J. J. (1975). The probability plot correlation coefficient test for normality. *Technometrics*, **17**, 111–117.

Fomel, S. et Claerbout, J. (2009). Guest editors' introduction : Reproducible research. *Computing in Science and Engineering*, **11**, 5–7.

Gel, Y. R. (2010). Test of fit for a Laplace distribution against heavier tailed alternatives. *Computational Statistics & Data Analysis*, **54**, 958–965. URL http://dx.doi.org/10.1016/j.csda.2009.10.008.

Gel, Y. R. et Gastwirth, J. L. (2008). A robust modification of the Jarque-Bera test of normality. *Economics Letters*, **99**, 30–32. URL http://dx.doi.org/10.1016/j.econlet.2007.05.022.

Gel, Y. R., Miao, W. et Gastwirth, J. L. (2007). Robust directed tests of normality against heavy-tailed alternatives. *Computational Statistics & Data Analysis*, **51**, 2734–2746. URL http://dx.doi.org/10.1016/j.csda.2006.08.022.

Glen, A. G., Leemis, L. M. et Barr, D. R. (2001). Order statistics in goodness-of-fit testing. *IEEE Transactions on Reliability*, **50**, 209–213.

Greenwood, M. (1946). The statistical study of infectious diseases. *Journal of Royal Statistical Society Series A*, **109**, 85–110.

Gulati, S. (2011). Goodness of fit test for the Rayleigh and the Laplace distributions. *International Journal of Applied Mathematics & Statistics*, **24**, 74–85.

Hegazy, Y. A. S. et Green, J. R. (1975). Some new goodness-of-fit tests using order statistics. *Applied Statistics*, **24**, 299–308.

Hosking, J. R. M. (1990). L-moments : analysis and estimation of distributions using linear combinations of order statistics. *Journal of the Royal Statistical Society. Series B. Methodological*, **52**, 105–124. URL http://links.jstor.org/sici?sici=0035-9246(1990)52:

1<105:AAEODU>2.0.CO;2-7&origin=MSN.

Jarque, C. M. et Bera, A. K. (1987). A test for normality of observations and regression residuals. *International Statistical Review. Revue International de Statistique*, **55**, 163–172. URL http://dx.doi.org/10.2307/1403192.

Johnson, N. L. (1949). Systems of frequency curves generated by methods of translation. *Biometrika*, **36**, 149–176.

Joiner, B. L. et Rosenblatt, J. R. (1971). Some properties of the range in samples from tukey's symmetric lambda distributions. *Journal of the American Statistical Association*, **66**, 394–399. URL http://www.tandfonline.com/doi/abs/10.1080/01621459.1971.10482275.

Kallenberg, W. C. M. et Ledwina, T. (1997). Data driven smooth tests for composite hypotheses : comparison of powers. *Journal of Statistical Computation and Simulation*, **59**, 101–121. URL http://dx.doi.org/10.1080/00949659708811850.

Kolmogorov, A. N. (1933). Sulla determinazione empirica di una legge di distibuziane. *Giornale dell'Istituta Italiano degli Attuari*, **4**, 83–91.

Kotz, S., Kozubowski, T. J. et Podgórski, K. (2001). *The Laplace distribution and generalizations*. Boston, MA : Birkhäuser Boston Inc. URL http://dx.doi.org/10.1007/978-1-4612-0173-1. A revisit with applications to communications, economics, engineering, and finance.

Kundu, D. (2005). Discriminating between normal and Laplace distributions. In *Advances in ranking and selection, multiple comparisons, and reliability*, Stat. Ind. Technol., 65–79. Boston, MA : Birkhäuser Boston. URL http://dx.doi.org/10.1007/0-8176-4422-9_4.

Lafaye de Micheaux, P. (2013). Unpublished.

Lafaye de Micheaux, P., Drouilhet, R. et Liquet, B. (2010). *The R Software : Fundamentals of Programming and Statistical Analysis*. Springer.

Langholz, B. et Kronmal, R. A. (1991). Tests of distributional hypotheses with nuisance parameters using Fourier series methods. *Journal of the American Statistical Association*, **86**, 1077–1084. URL http://links.jstor.org/sici?sici=0162-1459(199112)86:

416<1077:TODHWN>2.0.CO;2-6&origin=MSN.

Leisch, F. (2002a). Sweave : Dynamic generation of statistical reports using literate data analysis. In *Compstat 2002 — Proceedings in Computational Statistics* (eds. W. Härdle et B. Rönz), 575–580. Physica Verlag, Heidelberg. URL http://www.stat.uni-muenchen.de/~leisch/Sweave. ISBN 3-7908-1517-9.

Leisch, F. (2002b). Sweave, part I : Mixing R and LaTeX. *R News*, **2**, 28–31. URL http://CRAN.R-project.org/doc/Rnews/.

Leisch, F. (2003). Sweave, part II : Package vignettes. *R News*, **3**, 21–24. URL http://CRAN.R-project.org/doc/Rnews/.

Lenth, R. V. et Højsgaard, S. (2007). Sasweave : Literate programming using sas. *Journal of Statistical Software*, **19**, 1–20. URL http://www.jstatsoft.org/v19/i08.

Leone, F. C., Nelson, L. S. et Nottingham, R. B. (1961). The folded normal distribution. *Technometrics*, **3**, 543–550.

LeVeque, R. J. (2009). Python tools for reproducible research on hyperbolic problems. *Computing in Science and Engineering*, **11**, 19–27. URL http://link.aip.org/link/?CSX/11/19/1.

Lilliefors, H. (1967). On the Kolmogorov-Smirnov test for normality with mean and variance unknown. *Journal of the American Statistical Association*, **62**, 399–402.

Marhuenda, M. A., Marhuenda, Y. et Morales, D. (2005a). Uniformity tests under quantile categorization. *Kybernetes*, **34**, 888–901. URL http://dx.doi.org/10.1108/03684920510595553.

Marhuenda, Y., Morales, D. et Pardo, M. C. (2005b). A comparison of uniformity tests. *Statistics. A Journal of Theoretical and Applied Statistics*, **39**, 315–328. URL http://dx.doi.org/10.1080/02331880500178562.

Martinez, J. et Iglewicz, B. (1981). A test for departure from normality based on a biweight estimator of scale. *Biometrika*, **68**, 331–333. URL http://dx.doi.org/10.1093/biomet/68.1.331.

Meintanis, S. G. (2004). A class of omnibus tests for the Laplace distribution based on the empirical characteristic function. *Communications in Statistics. Theory and Methods*, **33**, 925–948. URL http://dx.doi.org/10.1081/

STA-120028735.

Morales, D., Pardo, L., Pardo, M. C. et Vajda, I. (2003). Limit laws for disparities of spacings. *Journal of Nonparametric Statistics*, **15**, 325–342. URL http://dx.doi.org/10.1080/1048525031000120206.

Moran, P. A. P. (1951). The random division of an interval. II. *Journal of the Royal Statistical Society. Series B. Methodological*, **13**, 147–150.

Nadarajah, S. (2005). A generalized normal distribution. *Journal of Applied Statistics*, **32**, 685–694. URL http://dx.doi.org/10.1080/02664760500079464.

Pardo, M. C. (2003). A test for uniformity based on informational energy. *Statistical Papers*, **44**, 521–534. URL http://dx.doi.org/10.1007/BF02926008.

Peng, R. D. et Eckel, S. P. (2009). Distributed reproducible research using cached computations. *Computing in Science and Engineering*, **11**, 28–34. URL http://link.aip.org/link/?CSX/11/28/1.

Pinto, F. et Vetterli, M. (2009). Space/time-frequency processing of acoustic wave fields : Theory, algorithms, and applications. *IEEE Transactions on Signal Processing*.

Puig, P. et Stephens, M. A. (2000). Tests of fit for the Laplace distribution, with applications. *Technometrics*, **42**, 417–424. URL http://dx.doi.org/10.2307/1270952.

Quesenberry, C. P. et Miller, F. L. J. (1977). Power studies of some tests for uniformity. *Journal of Statistical Computation and Simulation*, **5**, 169–191.

Rahman, M. M. et Govindarajulu, Z. (1997). A modification of the test of Shapiro and Wilk for normality. *Journal of Applied Statistics*, **24**, 219–235. URL http://dx.doi.org/10.1080/02664769723828.

Rayner, J. C. W. et Best, D. J. (1989). *Smooth tests of goodness of fit*. New York : The Clarendon Press Oxford University Press.

Read, T. R. C. et Cressie, N. A. C. (1988). *Goodness-of-fit statistics for discrete multivariate data*. Springer Series in Statistics. New York : Springer-Verlag. URL http://dx.doi.org/10.1007/978-1-4612-4578-0.

Romão, X., Delgado, R. et Costa, A. (2010). An empirical power comparison of univariate goodness-of-fit tests for normality. *Journal of Statistical Computation and Simulation*, **80**, 545–591. URL http://dx.doi.org/10.1080/00949650902740824.

Schwab, M., Karrenbach, M. et Claerbout, J. (2000). Making scientific computations reproducible. *Computing in Science and Engineering*, **2**, 61–67. URL http://dx.doi.org/10.1109/5992.881708.

Shapiro, S. S. et Francia, R. (1972). An approximation analysis of variance test for normality. *Journal of the American Statistical Association*, **67**, 215–216.

Shapiro, S. S. et Wilk, M. B. (1965). An analysis of variance test for normality : Complete samples. *Biometrika*, **52**, 591–611.

Spiegelhalter, D. J. (1977). A test for normality against symmetric alternatives. *Biometrika*, **64**, 415–418.

Swartz, T. (1992). Goodness-of-fit tests using Kullback-Leibler information. *Communications in Statistics. Simulation and Computation*, **21**, 711–729. URL http://dx.doi.org/10.1080/03610919208813046.

Vasicek, O. (1976). A test for normality based on sample entropy. *Journal of the Royal Statistical Society. Series B. Methodological*, **38**, 54–59.

Yen, V. C. et Moore, A. H. (1988). Modified goodness-of-fit test for the laplace distribution. *Communications in Statistics - Simulation and Computation*, **17**, 275–281.

Zhang, J. (2002). Powerful goodness-of-fit tests based on the likelihood ratio. *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, **64**, 281–294. URL http://dx.doi.org/10.1111/1467-9868.00337.

Zhang, J. et Wu, Y. (2005). Likelihood-ratio tests for normality. *Computational Statistics & Data Analysis*, **49**, 709–721. URL http://dx.doi.org/10.1016/j.csda.2004.05.034.

Zhang, P. (1999). Omnibus test of normality using the Q statistic. *Journal of Applied Statistics*, **26**, 519–528. URL http://dx.doi.org/10.1080/02664769922395.

Annexe A

EXEMPLE DE RECHERCHE REPRODUCTIBLE AVEC SWEAVE

Dans cette section, nous présentons un exemple pour vous montrer comment créer un rapport reproductible à l'aide de **Sweave**. Le principe est simple : nous allons créer un fichier **toto.Rnw** (Rnw signifie R-noweb) qui contient les documentations LaTeX et les codes R emballés dans les blocs de code (*code chunks*). Ensuite, il suffira de lancer R et de taper les commandes suivantes :

```
setwd(choose.dir())
Sweave("toto.Rnw")
library(tools)
texi2dvi("toto.tex", pdf = TRUE)
```

Un fichier minimal de **Sweave** est présenté ci-dessous ; celui-ci contient deux blocs de code incorporés dans un document LaTeX simple.

```
%---------- FICHIER toto.Rnw ----------%

\documentclass[a4paper]{article}
\usepackage[utf8]{inputenc}
\usepackage{tabularx}
\usepackage{booktabs}
\usepackage{multirow}

\title{Un exemple de Sweave : toto}
\author{Viet Anh Tran}
```

A-ii

```
\begin{document}
\maketitle
Une expérience a été menée afin de déterminer si la température a un
impact important sur la densité de différents types de briques.
Une expérience complètement aléatoire conduit aux données suivantes:
\begin{table}[htbp]
  \centering
  {\footnotesize
  \caption{Densité des briques selon les températures}
    \begin{tabular}{cccccc}
    \toprule
    Température & \multicolumn{5}{c}{Densité}       \\
    \midrule
    100         & 21.8 & 21.9 & 21.7 & 21.6 & 21.7  \\
    125         & 21.7 & 21.4 & 21.5 & 21.4 &       \\
    150         & 21.9 & 21.8 & 21.8 & 21.6 & 21.5  \\
    175         & 21.9 & 21.7 & 21.8 & 21.4 &       \\
    \bottomrule
    \end{tabular}}
\end{table}
En utilisant la fonction ANOVA du logiciel R, nous obtenons:

% Début du premier bloc de code R
<<>>=
temperature <- as.factor(c("100","100","100","100","100","125","125",
                           "125","125","150","150","150","150","150",
                           "175","175","175","175"))
densite <- c(21.8,21.9,21.7,21.6,21.7,21.7,21.4,21.5,21.4,
     21.9,21.8,21.8,21.6,21.5,21.9,21.7,21.8,21.4)
densite.aov <- aov(densite ~ temperature)
summary(densite.aov)
@
% Fin du premier bloc de code R

La valeur-p est grande. Donc nous ne pouvons pas rejeter $H_{0}$,
par suite il est impossible de conclure que la température a une
influence importante sur la densité de différents types de briques.
Nous traçons aussi le graphique des résidus:

% Début du deuxième bloc de code R
```

```
\begin{center}
<<fig=TRUE,height=4.3,width=8,echo=FALSE>>=
qqnorm(resid(densite.aov),pch=16,xlab="quantiles theoriques",
                                  ylab="quantiles empiriques")
qqline(resid(densite.aov),lwd=2 ,col="red")
@
% Fin du deuxième bloc de code R

\end{center}
\end{document}
```

Sweave traduit ce fichier **toto.Rnw** en un document LaTeX nommé **toto.tex** qui contient des environnements d'entrée et de sortie de R pour les deux blocs de code ci-dessus, à l'aide du fichier **Sweave.sty** (automatiquement chargé). Par ailleurs, les autres textes sont copiés sans aucune modification du fichier **toto.Rnw** au fichier **toto.tex**.

```
%---------- FICHIER toto.tex ----------%

\documentclass[a4paper]{article}
\usepackage[utf8]{inputenc}
\usepackage{tabularx}
\usepackage{booktabs}
\usepackage{multirow}
\title{Un exemple de Sweave : toto}
\author{Viet Anh Tran}
\usepackage{Sweave}
\begin{document}
\maketitle
Une expérience a été menée afin de déterminer si la température a un
impact important sur la densité de différents types de briques.
Une expérience complètement aléatoire conduit aux données suivantes:
\begin{table}[htbp]
  \centering
  {\footnotesize
  \caption{Densité des briques selon les températures}
    \begin{tabular}{cccccc}
    \toprule
    Température & \multicolumn{5}{c}{Densité}       \\
    \midrule
    100         & 21.8 & 21.9 & 21.7 & 21.6 & 21.7  \\
```

```
    125         & 21.7 & 21.4 & 21.5 & 21.4 &       \\
    150         & 21.9 & 21.8 & 21.8 & 21.6 & 21.5  \\
    175         & 21.9 & 21.7 & 21.8 & 21.4 &       \\
    \bottomrule
    \end{tabular}}
\end{table}
En utilisant la fonction ANOVA du logiciel R, nous obtenons:

% Début du premier bloc de code R
\begin{Schunk}
\begin{Sinput}
> temperature <- as.factor(c("100","100","100","100","100","125","125",
+                           "125","125","150","150","150","150","150",
+                           "175","175","175","175"))
> densite <- c(21.8,21.9,21.7,21.6,21.7,21.7,21.4,21.5,21.4,
+       21.9,21.8,21.8,21.6,21.5,21.9,21.7,21.8,21.4)
> densite.aov <- aov(densite ~ temperature)
> summary(densite.aov)
\end{Sinput}
\begin{Soutput}
            Df Sum Sq Mean Sq F value Pr(>F)
temperature  3 0.1561 0.05204   2.024  0.157
Residuals   14 0.3600 0.02571
\end{Soutput}
\end{Schunk}
% Fin du premier bloc de code R

La valeur-p est grande. Donc nous ne pouvons pas rejeter $H_{0}$,
par suite il est impossible de conclure que la température a une
influence importante sur la densité de différents types de briques.
Nous traçons aussi le graphique des résidus:

% Début du deuxième bloc de code R
\begin{center}
\includegraphics{toto-002}
\end{center}
% Fin du deuxième bloc de code R

\end{document}
```

Le travail de **Sweave** se fait sur les blocs de code de manière suivante. Le premier bloc n'a pas de nom, le comportement par défaut de **Sweave** est alors observé : transfert à la fois des commandes de R et de leur sortie respectivement dans le fichier LaTeX.

```
<<>>=
...
densite.aov <- aov(densite ~ temperature)
summary(densite.aov)
@
```

Le deuxième bloc montre une des extensions de **Sweave** : le nom du bloc peut être utilisé pour contrôler la sortie finale.

```
<<fig=TRUE,height=4.3,width=8,echo=FALSE>>=
...
@
```

- **fig=TRUE** informe **Sweave** de créer les fichiers images EPS et PDF correspondants aux lignes de commande de R dans le deuxième bloc. Par conséquent, une commande `\includegraphics{toto-002}` sera automatiquement insérée dans le fichier **.tex**.
- **height=4.3,width=8** donnent la taille de la zone de figure.
- **echo=FALSE** indique que les lignes de commandes de R ne seront pas incluses dans le document final.

Voilà une image du document final créé :

Un exemple de Sweave : toto

Viet Anh Tran

June 16, 2013

Une expérience a été menée afin de déterminer si la température a un impact important sur la densité de différents types de briques. Une expérience complètement aléatoire conduit aux données suivantes:

Table 1: Densité des briques selon les températures

Température	Densité				
100	21.8	21.9	21.7	21.6	21.7
125	21.7	21.4	21.5	21.4	
150	21.9	21.8	21.8	21.6	21.5
175	21.9	21.7	21.8	21.4	

En utilisant la fonction ANOVA du logiciel R, nous obtenons:

```
> temperature <- as.factor(c("100","100","100","100","100","125","125",
+                            "125","125","150","150","150","150","150",
+                            "175","175","175","175"))
> densite <- c(21.8,21.9,21.7,21.6,21.7,21.7,21.4,21.5,21.4,
+              21.9,21.8,21.8,21.6,21.5,21.9,21.7,21.8,21.4)
> densite.aov <- aov(densite ~ temperature)
> summary(densite.aov)

            Df Sum Sq Mean Sq F value Pr(>F)
temperature  3 0.1561 0.05204   2.024  0.157
Residuals   14 0.3600 0.02571
```

La valeur-p est grande. Donc nous ne pouvons pas rejeter H_0, par suite il est impossible de conclure que la température a une influence importante sur la densité de différents types de briques. Nous traçons aussi le graphique des résidus:

FIGURE A.1. Document final créé par LaTeX : **toto.pdf**.

Annexe B

DÉBOGAGE DE FONCTIONS

Si vous voulez ajouter de nouvelles fonctions à notre progiciel **PoweR**, à un moment ou à un autre vous devrez effectuer des opérations de débogage. Cette section vous donnera les idées principales sur les outils de débogage usuels. Il y a deux types d'erreurs dans ce cas : une erreur dans un code R et une erreur dans un code C/C++.

B.1. ERREUR DANS UN CODE R

Si l'erreur se trouve dans un fichier .R, il suffit d'installer, puis de charger en mémoire le progiciel **debug**. Ensuite, la fonction `mtrace()` de ce progiciel sera utilisée de la façons suivante :

```
require(debug)
mtrace(powcomp.fast)
powcomp.fast(...)
```

Ouvrir la fenêtre `tcltk` en plein écran ou la mettre à côté de la fenêtre de **R**. Ensuite, cliquer sur la touche `ENTER` pour commencer le débogage pas à pas. Finalement, taper l'instruction suivante pour terminer le débogage :

```
mtrace.off()
```

B.2. ERREUR DANS UN CODE C/C++

Nous présentons maintenant comment il est possible d'effectuer le même genre d'opération de débogage, pour une partie de code écrite en langage C/C++ ou Fortran. Il s'agira essentiellement d'utiliser l'option de compilation `-g` pour rajouter des informations sur le code source dans le fichier `DLL` créé, et ensuite d'utiliser un outil de débogage spécialisé.

Voici une procédure de débogage avec **GDB** et **Emacs** :

B-ii

(1) Créer un fichier nommé **Makevars.win** dans un sous-dossier nommé **.R/** de son répertoire **$HOME$**. Ce fichier devra contenir les lignes suivantes :

```
## for C++ code
CXXFLAGS=-g
```

Normalement ce fichier se trouve à l'addresse "C :/Users/NomPC/.R/".

(2) Vérifier si le fichier **NAMESPACE** se trouvant dans "/PoweR/" contient la ligne suivante :

```
useDynLib(PKG)
```

(3) Recompiler le progiciel et le réinstaller en tapant les instructions suivantes :

```
R CMD check PoweR
R CMD INSTALL --debug --build PoweR
R CMD INSTALL PoweR.zip
```

(4) Ouvrir **Emacs** et taper simultanément ALT et X

```
M-x R [RET] [RET]
M-x gdb [RET] gdb -i=mi --annotate=3 [RET]
```

où `[RET] = ENTER`.

Votre fenêtre **Emacs** devrait alors être séparée en deux, avec **R** en haut et **GDB** en bas. Si ce n'est pas le cas, allez dans le menu `File/Split Window` ou `File/New Window Below (C-x 2)`, puis allez dans le menu `Buffers` pour sélectionner `*R* *`.

(5) Il faut ensuite récupérer le numéro de processus de **R**. Pour cela, on utilise la combinaison de touches `CTRL+ALT+Suppr` pour ouvrir le gestionnaire des tâches. On sélectionne ensuite l'onglet des `Processus`.

Dans le menu `Affichage/Sélectionner des colonnes...`, on coche la boîte `PID` (identificateur de processus), ce qui aura pour effet de rajouter une colonne `PID` dans le gestionnaire des tâches. On cherche alors le numéro du processus (`PID`) dont le nom est `Rterm.exe *32` (par exemple 5404).

On tape ensuite dans **Emacs** les instructions suivantes :

```
(gdb) attach 5404 [RET]
(gdb) signal 0 [RET]
```

(6) Cliquer sur le panneau (appelée `Buffer` dans **Emacs**) intitulé `*R*`, et entrer les instructions suivantes :

```
> setwd("/R/R-2.15.2/library/PoweR/libs/i386/")
> dyn.load("PoweR.dll")
```

ou encore

```
> require(PoweR)
```

⑦ Cliquer sur la sous-fenêtre inférieure (`Buffer *gud*`)

```
Ctrl+C Ctrl+C
(gdb) b statj.cpp:1 [RET]
(gdb) c [RET]
```

⑧ Cliquer sur la sous-fenêtre supérieure (`Buffer *R*`)

```
> compquant(n=10,law=1,stat=52,quant=0.05,
            M=10^5,parlaw=NULL,model=NULL)$quant
M-x gdb-many-windows
```

⑨ Passer la fenêtre d'**Emacs** en plein écran. La fenêtre **Emacs** devrait maintenant être divisée en six panneaux.

⑩ Cliquer maintenant sur la fenêtre `statj.cpp`. À partir de ce moment, nous pouvons cliquer sur le symbole de `Next Line` (à droite du `GO`) pour effectuer le code C/C++ pas à pas. Les valeurs des paramètres se trouvent dans la fenêtre en haut à droite.

Remarque B.2.1.

Il est conseillé de supprimer les deux fichiers ***PoweR.dll*** *qui se trouvent dans le dossier ".../R/R-2.15.2/library/PoweR/libs/" avant de recompiler le progiciel pour eviter des erreurs potentielles. De plus, vous pourrez consulter avec profit le livre* The R Software *de Lafaye de Micheaux et coll. (2010) pour un guide de débogage plus complet et détaillé.*

Annexe C

REPRODUIRE CERTAINS RÉSULTATS POUR DES TESTS DE NORMALITÉ DANS DES ARTICLES PUBLIÉS

C.1. LIKELIHOOD-RATIO TESTS FOR NORMALITY - ZHANG AND WU (2005)

```
# Regenerate table 1 from zhang2005 (page 712)

law.index <- c(2)
M <- 1000000
vectn <- c(5:10,seq(12,20,2)
    ,25,30,40,50,70,100,150,200,300,500,1000)
stat.indices <- c(4)
level <- 1 - c(0.001,0.01,0.05,seq(0.10,0.90,0.10),0.95,0.99,0.999)
alter <- list(stat4=3)
table1 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL,model=NULL)
print.critvalues(table1,digits=3,latex.output=FALSE)
```

```
# Regenerate table 2 from zhang2005 (page 713)

law.index <- c(2)
M <- 1000000
```

C-ii

```
vectn <- c(5:10,seq(12,20,2)
    ,25,30,40,50,70,100,150,200,300,500,1000)
stat.indices <- c(3)
level <- 1 - c(0.001,0.01,0.05,seq(0.10,0.90,0.10),0.95,0.99,0.999)
alter <- list(stat3=3)
table2 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL,model=NULL)
print.critvalues(table2,digits=3,latex.output=FALSE)
```

C.2. A TEST FOR NORMALITY OF OBSERVATIONS AND REGRESSION RESIDUALS - JARQUE AND BERA (1987)

```
# Regenerate table 1 from jarque1987 (page 168)

law.index <- c(2)
M <- 250
vectn <- c(20,50)
stat.indices <- c(24,21,22,7)
level <- c(0.10)
alter <- list(stat24=0,stat21=4,stat22=4,stat7=3)
law.indices <- c(6,8,5,10)
parlaws <- list(law6=c(3,2),law8=c(5),law5=c(2,1),law10=c(0,1))
for (n in vectn) {
  critval <- many.crit(law.index,M,n,stat.indices,level,alter,parlaw
      =NULL,parstat=NULL)
  table1 <- powcomp.fast(n,M,law.indices,stat.indices,level,critval=
      critval,alter,parlaws,parstats=NULL,nbclus=1)
  print(print.power(table1,digits=2,latex.output=FALSE))
}
```

```
# Regenerate table 2 from jarque1987 (page 169)

```

```
law.index <- c(2)
M <- 10000
vectn <- c(20,30,40,50,75,100,125,150,200,250,300,400,500,800,1000)
stat.indices <- c(7)
level <- c(0.10,0.05)
alter <- list(stat7=3)
table2 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL,model=NULL)
print.critvalues(table2,digits=2,latex.output=FALSE)
```

C.3. A ROBUST MODIFICATION OF THE JARQUE-BERA TEST OF NORMALITY - GEL AND GASTWIRTH (2008)

```
# Regenerate table 1 from gel2008 (page 31)

law.index <- c(2)
M <- 10000
vectn <- c(30,50,100)
stat.indices <- c(21,7,9)
level <- c(0.05)
alter <- list(stat21=4,stat7=3,stat9=3)
law.indices <- c(2,8,8,4,1,22,35)
parlaws <- list(law2=c(0,1),law8=c(3),law8=c(5),law4=c(0,1),law1=c
    (0,1),law22=c(0.5,0),law35=c(1))
critval <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL)
table1 <- powcomp.fast(vectn,M,law.indices,stat.indices,level,
    critval=critval,alter,parlaws,parstats=NULL,nbclus=1)
print.power(table1,digits=2,latex.output=FALSE)
```

C.4. An analysis of variance test for normality : Complete samples - Shapiro and Wilk (1965)

```
# Regenerate table 6 from shapiro1965 (page 605)

law.index <- c(2)
M <- 10000
vectn <- seq(4,50,1)
stat.indices <- c(21)
level <- c(0.01,0.02,0.05,0.10,0.50,0.90,0.95,0.98,0.99)
alter <- list(stat21=4)
table6 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL,model=NULL)
print.critvalues(table6,digits=3,latex.output=FALSE)
```

```
law.index <- c(2)
M <- 10000
vectn <- c(20)
stat.indices <- c(21,1,2)
level <- c(0.05)
alter <- list(stat21=4,stat1=3,stat2=3)
law.indices <- c(9,9,9,9,10,3,7,4,6,1)
parlaws <- list(law9=c(1),law9=c(2),law9=c(4),law9=c(10),law10=c
    (0,1),law3=c(0,1),law7=c(0,1),law4=c(0,1),law6=c(2,1),law1=c(0,1)
    )
critval <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL)
table7 <- powcomp.fast(vectn,M,law.indices,stat.indices,level,
    critval=critval,alter,parlaws,parstats=NULL,nbclus=1)
print.power(table7,digits=0,latex.output=FALSE)
```

C.5. A MODIFICATION OF THE TEST OF SHAPIRO AND WILK FOR NORMALITY - RAHMAN AND GOVINDARAJULU (1997)

```
# Regenerate table 2 from rahman1997 (page 226)

law.index <- c(2)
M <- 20000
vectn <- c(seq(4,99,1),seq(100,200,10),seq(250,1000,50),seq
    (1500,5000,500))
stat.indices <- c(23)
level <- c(0.01,0.02,0.05,0.10,0.50,0.90,0.95,0.98,0.99)
alter <- list(stat23=4)
table2 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL,model=NULL)
print.critvalues(table2,digits=3,latex.output=FALSE)
```

```
# Regenerate table 4 from rahman1997 (page 232)

law.index <- c(2)
M <- 5000
vectn <- c(10,20,35,50,75,99)
stat.indices <- c(21,22,23)
level <- c(0.01,0.05,0.10)
alter <- list(stat21=4,stat22=4,stat23=4)
law.indices <- c(7,4,10,9,9,9,9,3,1,6)
parlaws <- list(law7=c(0,1),law4=c(0,1),law10=c(0,1),law9=c(1),law9=
    c(2),law9=c(4),law9=c(10),law3=c(0,1),law1=c(0,1),law6=c(2,1))
for (lv in level) {
  critval <- many.crit(law.index,M,vectn,stat.indices,lv,alter,
      parlaw=NULL,parstat=NULL)
  table2 <- powcomp.fast(vectn,M,law.indices,stat.indices,lv,critval
      =critval,alter,parlaws,parstats=NULL,nbclus=1)
  print(print.power(table2,digits=0,latex.output=FALSE))
}
```

C.6. THE PROBABILITY PLOT CORRELATION COEFFICIENT TEST FOR NORMALITY - FILLIBEN (1975)

```
# Regenerate table 1 from filliben1995 (page 113)

law.index <- c(2)
M <- 100000
vectn <- c(seq(4,50,1),seq(55,100,5))
stat.indices <- c(25)
level <- c(0.000,0.005,0.01,0.025,0.05,0.10,0.25,
           0.50,0.75,0.90,0.95,0.975,0.99,0.995)
alter <- list(stat25=4)
table1 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL,model=NULL)
print.critvalues(table1,digits=3,latex.output=FALSE)
```

```
# Regenerate table 2 from filliben1995 (page 114) (partially)
# Here M = 200 is too small to be exact

law.index <- c(2); M <- 200
vectn <- c(20,50,100)
stat.indices <- c(21,22,1,25)
level <- c(0.05)
alter <- list(stat21=4,stat22=4,stat1=3,stat25=4)
law.indices <- c(20,18,18,7,20,18,18,18,20)
parlaws <- list(law20=c(0,0.5),law18=c(1.5),law18=c(1.25),law7=c
    (0,1),law20=c(0,0.7071),law18=c(0.75),law18=c(0.5),law18=c(0.25),
    law20=c(0,2))
critval <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL)
```

```
table2 <- powcomp.fast(vectn,M,law.indices,stat.indices,level,
    critval=critval,alter,parlaws,parstats=NULL,nbclus=1)
print.power(table2,digits=0,latex.output=FALSE)
```

C.7. A GOODNESS-OF-FIT TEST FOR NORMALITY BASED ON POLYNOMIAL REGRESSION - COIN (2008)

```
# Regenerate table 1 from coin2008 (page 2188)

law.index <- c(2)
M <- 200000
vectn <- c(seq(10,100,10),seq(150,500,50),seq(600,1000,100))
stat.indices <- c(30)
level <- 1 - c(0.90,0.95,0.99,0.995,0.999)
alter <- list(stat30=3)
table1 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL,model=NULL)
print.critvalues(table1,digits=6,latex.output=FALSE)
```

```
law.index <- c(2)
M <- 20000
vectn <- c(20,50,200)
stat.indices <- c(1,2,4,7,17,21,24,30)
level <- c(0.05)
alter <- list(stat1=3,stat2=3,stat4=3,stat7=3,stat17=0,stat21=4,
    stat24=0,stat30=3)
law.indices <- c(19,19,19,19,19,19,6,7,32,6,32,32,2,8,4,
                 22,22,22,22,22,22,22,22,22,18,1,8,8,8,
                 6,32,6,32,32,11,11,11,11,
                 19,19,19,19,19,19,19,19,19,
                 9,35,9,10,11)
```

```
parlaws <- list(law19=c(0.5,10),law19=c(0.5,5),law19=c(0.5,4),law19=
    c(0.5,3),law19=c(0.5,2),law19=c(0.5,1),
                    law6=c(0.5,0.5),law7=c(0,1),law32=c(-1,1),law6=c
                        (2,2),law32=c(-2,2),law32=c(-3,3),
          law2=c(0,1),law8=c(9),law4=c(0,1),
          law22=c(0.05,3),law22=c(0.05,5),law22=c(0.05,7),
          law22=c(0.1,3),law22=c(0.1,5),law22=c(0.1,7),
          law22=c(0.2,3),law22=c(0.2,5),law22=c(0.2,7),
          law18=c(10),law1=c(0,1),law8=c(5),law8=c(3),law8=c(1),
          law6=c(2,1),law32=c(-2,1),law6=c(3,2),law32=c(-3,1),law32=c
              (-3,2),law11=c(4,1),law11=c(3.6,1),
          law11=c(2.2,1),law11=c(2,1),
          law19=c(0.2,3),law19=c(0.2,5),law19=c(0.2,7),
          law19=c(0.1,3),law19=c(0.1,5),law19=c(0.1,7),
          law19=c(0.05,3),law19=c(0.05,5),law19=c(0.05,7),
          law9=c(4),law35=c(1),law9=c(1),law10=c(0,1),law11=c(0.5,1))
for (n in vectn) {
  critval <- many.crit(law.index,M,n,stat.indices,level,alter,parlaw
      =NULL,parstat=NULL)
  tables345 <- powcomp.fast(n,M,law.indices,stat.indices,level,
      critval=critval,alter,parlaws,parstats=NULL,nbclus=1)
  print(print.power(tables345,digits=2,latex.output=FALSE))
}
```

```
# Regenerate tables 6,7 and 8 from coin2008 (page 2194)

law.index <- c(2); M <- 20000
vectn <- c(20,50,200)
stat.indices <- c(1,2,4,7,17,21,24,30)
level <- c(0.01)
alter <- list(stat1=3,stat2=3,stat4=3,stat7=3,stat17=0,stat21=4,
    stat24=0,stat30=3)
law.indices <- c(19,19,19,19,19,19,6,7,32,6,32,32,2,8,4,
                 22,22,22,22,22,22,22,22,22,18,1,8,8,8,
                 6,32,6,32,32,11,11,11,11,
```

```
                    19,19,19,19,19,19,19,19,19,
                    9,35,9,10,11)
parlaws <- list(law19=c(0.5,10),law19=c(0.5,5),law19=c(0.5,4),law19=
    c(0.5,3),law19=c(0.5,2),law19=c(0.5,1),
                    law6=c(0.5,0.5),law7=c(0,1),law32=c(-1,1),law6=c
                        (2,2),law32=c(-2,2),law32=c(-3,3),
          law2=c(0,1),law8=c(9),law4=c(0,1),
          law22=c(0.05,3),law22=c(0.05,5),law22=c(0.05,7),
          law22=c(0.1,3),law22=c(0.1,5),law22=c(0.1,7),
          law22=c(0.2,3),law22=c(0.2,5),law22=c(0.2,7),
          law18=c(10),law1=c(0,1),law8=c(5),law8=c(3),law8=c(1),
          law6=c(2,1),law32=c(-2,1),law6=c(3,2),law32=c(-3,1),law32=c
              (-3,2),law11=c(4,1),law11=c(3.6,1),
          law11=c(2.2,1),law11=c(2,1),
          law19=c(0.2,3),law19=c(0.2,5),law19=c(0.2,7),
          law19=c(0.1,3),law19=c(0.1,5),law19=c(0.1,7),
          law19=c(0.05,3),law19=c(0.05,5),law19=c(0.05,7),
          law9=c(4),law35=c(1),law9=c(1),law10=c(0,1),law11=c(0.5,1))
for (n in vectn) {
  critval <- many.crit(law.index,M,n,stat.indices,level,alter,parlaw
      =NULL,parstat=NULL)
  tables678 <- powcomp.fast(n,M,law.indices,stat.indices,level,
      critval=critval,alter,parlaws,parstats=NULL,nbclus=1)
  print(print.power(tables678,digits=2,latex.output=FALSE))
}
```

C.8. A TEST OF NORMALITY BASED ON EMPIRICAL CHARACTERISTIC FUNCTION - EPPS AND PULLEY (1983)

```
# Regenerate table 1 from epps1983 (page 725)

law.index <- c(2); M <- 10000
vectn <- c(seq(4,12,2),200)
stat.indices <- c(31,31)
```

C-x

```
level <- c(0.025,0.050,0.950,0.975)
alter <- list(stat31=3,stat31=3)
parstat <- list(stat31=c(0.7),stat31=c(1.0))
table1 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat,model=NULL)
print.critvalues(table1,digits=2,latex.output=FALSE)
```

```
# Regenerate table 2 from epps1983 (page 725)

law.index <- c(2)
M <- 10000
vectn <- c(20,50)
stat.indices <- c(31,31,21,25)
level <- c(0.05)
alter <- list(stat31=3,stat31=3,stat21=4,stat25=4)
law.indices <- c(6,20,17,9,10,6,8,1,17,3)
parstats <- list(stat31=c(0.7),stat31=c(1.0),stat21=NA,stat25=NA)
parlaws <- list(law6=c(2,1),law20=c(1,1),law17=c(-1,2),law9=c(4),
    law10=c(0,1),law6=c(1,1),law8=c(10),law1=c(0,1),law17=c(0,1),law3
    =c(0,1))
for (n in vectn) {
  critval <- many.crit(law.index,M,n,stat.indices,level,alter,parlaw
      =NULL,parstat=parstats)
  table2 <- powcomp.fast(n,M,law.indices,stat.indices,level,critval=
      critval,alter,parlaws,parstats,nbclus=1)
  print(print.power(table2,digits=0,latex.output=FALSE))
}
```

C.9. A TEST FOR DEPARTURE FROM NORMALITY BASED ON A BIWEIGHT ESTIMATOR OF SCALE - MARTINEZ AND IGLEWICZ (1981)

```
# Regenerate table 1 from martinez1981 (page 332)

law.index <- c(2)
M <- 80000
vectn <- c(seq(10,50,5),seq(60,100,10),150,200,300)
stat.indices <- c(32)
level <- 1 - c(0.90,0.95,0.975,0.99)
alter <- list(stat32=3)
table1 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL,model=NULL)
print.critvalues(table1,digits=3,latex.output=FALSE)
```

```
# Regenerate table 2 from martinez1981 (page 333) (partially)

law.index <- c(2)
M <- 1000
vectn <- c(10,50)
stat.indices <- c(41,21,24,32)
level <- c(0.10,0.05)
alter <- list(stat41=3,stat21=4,stat24=0,stat32=3)
law.indices <- c(10,10)
parlaws <- list(law10=c(0,0.25),law10=c(0,0.5))
for (i in 1:2) {
  critval <- many.crit(law.index,M,vectn[i],stat.indices,level[i],
      alter,parlaw=NULL,parstat=NULL)
  table2 <- powcomp.fast(vectn[i],M,law.indices,stat.indices,level[i
      ],critval=critval,alter,parlaws,parstats=NULL,nbclus=1)
  print(print.power(table2,digits=1,latex.output=FALSE))
}
```

C.10. Robust directed tests of normality against heavy-tailed alternatives - Gel et al. (2007)

```
# Regenerate tables 2,3 and 4 from gel2007 (page 2741)

law.index <- c(2)
M <- 10000
vectn <- c(20,50,100)
stat.indices <- c(21,7,33,17,1)
level <- c(0.05)
alter <- list(stat21=4,stat7=3,stat33=3,stat17=2,stat1=3)
law.indices <- c(8,35,4,1,7,6,6,6,3,10,31)
parlaws <- list(law8=c(3),law35=c(1),law4=c(0,1),law1=c(0,1),law7=c
    (0,1),law6=c(2,2),law6=c(3,3),law6=c(4,4),law3=c(0,1),law10=c
    (0,1),law31=c(0.1,0,3))
for (n in vectn) {
  critval <- many.crit(law.index,M,n,stat.indices,level,alter,parlaw
      =NULL,parstat=NULL)
  tables234 <- powcomp.fast(n,M,law.indices,stat.indices,level,
      critval=critval,alter,parlaws,parstats=NULL,nbclus=1)
  print(print.power(tables234,digits=0,latex.output=FALSE))
}
```

C.11. A test for Normality against symetric alternatives - Spiegelhalter (1977)

```
# Regenerate table 1 from spiegelhalter1977 (page 417)

law.index <- c(2)
M <- 3000
vectn <- c(5,10,15,20,50,100)
stat.indices <- c(41)
level <- c(0.05,0.10)
alter <- list(stat41=3)
```

```
table1 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL,model=NULL)
print.critvalues(table1,digits=3,latex.output=FALSE)
```

```
# Regenerate table 2 from spiegelhalter1977 (page 417) (partially)

law.index <- c(2)
M <- 1000
vectn <- c(20)
stat.indices <- c(41,21)
level <- c(0.10)
alter <- list(stat41=3,stat21=4)
law.indices <- c(7,18,18,4,1,3)
parlaws <- list(law7=c(0,1),law18=c(0.7),law18=c(0.1),law4=c(0,1),
    law1=c(0,1),law3=c(0,1))
critval <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL)
table2 <- powcomp.fast(vectn,M,law.indices,stat.indices,level,
    critval=critval,alter,parlaws,parstats=NULL,nbclus=1)
print.power(table2,digits=0,latex.output=FALSE)
```

Annexe D

REPRODUIRE CERTAINS RÉSULTATS POUR DES TESTS POUR UNE DISTRIBUTION DE LAPLACE DANS DES ARTICLES PUBLIÉS

D.1. MODIFIED GOODNESS-OF-FIT TEST FOR THE LAPLACE DISTRIBUTION - YEN AND MOORE (1988)

```
# Regenerate table I from yenn1988 (page 277)

require(PoweR)
law.index <- 1
M <- 5000
vectn <- seq(5,50,5)
stat.indices <- c(42,43)
level <- c(0.20,0.15,0.10,0.05,0.01)
alter <- list(stat42=3,stat43=3)
table1 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,model=NULL)
print.critvalues(table1,digits=3,latex.output=FALSE)
```

```
# Regenerate table II from yenn1988 (page 279)

law.index <- c(1); M <- 5000
vectn <- c(5,15,25,50)
```

D-ii

```
stat.indices <- c(42,43)
level <- c(0.01)
alter <- list(stat42=3,stat43=3)
law.indices <- c(1,2,11,7,3,5,35)
parlaws <- list(law1=c(0,1),law2=c(0,1),law11=c(3,1),law7=c(0,1),
    law3=c(0,1),law5=c(6,1),law35=c(1))
critval <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL)
table2 <- powcomp.fast(vectn,M,law.indices,stat.indices,level,
    critval=critval,alter,parlaws,parstats=NULL,nbclus=1)
print.power(table2,digits=1,latex.output=FALSE)
```

```
# Regenerate table III from yenn1988 (page 279)

law.index <- c(1)
M <- 5000
vectn <- c(5,15,25,50)
stat.indices <- c(42,43)
level <- c(0.05)
alter <- list(stat42=3,stat43=3)
law.indices <- c(1,2,11,7,3,5,35)
parlaws <- list(law1=c(0,1),law2=c(0,1),law11=c(3,1),law7=c(0,1),
    law3=c(0,1),law5=c(6,1),law35=c(1))
critval <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL)
table3 <- powcomp.fast(vectn,M,law.indices,stat.indices,level,
    critval=critval,alter,parlaws,parstats=NULL,nbclus=1)
print.power(table3,digits=1,latex.output=FALSE)
```

D.2. Tests of fit for the Laplace distribution, with applications - Puig and Stephens (2000)

```
# Regenerate tables 1,2,3 from puig2000 (page 419)

Here we consider n=1000 as INFINITE
law.index <- c(1)
M <- 10000
vectn <- c(10,15,20,35,50,75,100,1000)
level <- c(0.50,0.25,0.10,0.05,0.025,0.01)
table1 <- many.crit(law.index,M,vectn,stat.indices=c(43),level,alter
    =list(stat43=3),parlaw=NULL,parstat=NULL)
table2 <- many.crit(law.index,M,vectn,stat.indices=c(44),level,alter
    =list(stat44=3),parlaw=NULL,parstat=NULL)
table3 <- many.crit(law.index,M,vectn,stat.indices=c(42),level,alter
    =list(stat42=3),parlaw=NULL,parstat=NULL)
print.critvalues(table1,digits=3,latex.output=FALSE)
print.critvalues(table2,digits=3,latex.output=FALSE)
print.critvalues(table3,digits=3,latex.output=FALSE)
```

```
# Regenerate tables 4,5 from puig2000 (page 420)

law.index <- c(1)
M <- 10000
vectn <- c(10,15,20,35,50,75,100)
level <- c(0.50,0.25,0.10,0.05,0.025,0.01)
table4 <- many.crit(law.index,M,vectn,stat.indices=c(45),level,alter
    =list(stat45=3),parlaw=NULL,parstat=NULL)
table5 <- many.crit(law.index,M,vectn,stat.indices=c(46),level,alter
    =list(stat46=3),parlaw=NULL,parstat=NULL)
print.critvalues(table4,digits=3,latex.output=FALSE)
print.critvalues(table5,digits=3,latex.output=FALSE)
```

```
# Regenerate table 6 from puig2000 (page 424)

law.index <- c(1)
```

D-iv

```
M <- 50000
vectn <- c(10,15,20,35,50,75,100)
level <- c(0.05)
stat.indices <- c(43,44,42,45,46)
law.indices <- c(2,3,4)
alter <- list(stat43=3,stat44=3,stat42=3,stat45=3,stat46=3)
critval <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL)
table6 <- powcomp.fast(vectn,M,law.indices,stat.indices,level,
    critval=critval,alter,parlaws=NULL,parstats=NULL,nbclus=1)
print.power(table6,digits=0,latex.output=FALSE)
```

D.3. A Class of Omnibus Tests for the Laplace Distribution Based on the Empirical Characteristic Function - Meintanis (2004)

```
# Regenerate table 1 from meintanis2004 (page 941)

law.index <- c(1)
M <- 100000
vectn <- c(20,50)
stat.indices <- c(47,49,47,49,47,49,47,49,47,49,47,49)
level <- c(0.05,0.10)
alter <- list(stat47=3,stat49=3,stat47=3,stat49=3,stat47=3,stat49=3,
    stat47=3,stat49=3,stat47=3,stat49=3,stat47=3,stat49=3)
parstat <- list(stat47=0.5,stat49=0.5,stat47=1.0,stat49=1.0,stat47
    =2.0,stat49=2.0,stat47=4.0,stat49=4.0,stat47=5.0,stat49=5.0,
    stat47=10.0,stat49=10.0)
table1 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat)
```

```
# Regenerate table 2 from meintanis2004 (page 941)

law.index <- c(1)
M <- 100000
vectn <- c(20,50)
stat.indices <- c(48,50,48,50,48,50,48,50,48,50,48,50)
level <- c(0.05,0.10)
alter <- list(stat48=3,stat50=3,stat48=3,stat50=3,stat48=3,stat50=3,
    stat48=3,stat50=3,stat48=3,stat50=3,stat48=3,stat50=3)
parstat <- list(stat48=0.5,stat50=0.5,stat48=1.0,stat50=1.0,stat48
    =2.0,stat50=2.0,stat48=4.0,stat50=4.0,stat48=5.0,stat50=5.0,
    stat48=10.0,stat50=10.0)
table2 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat)
```

```
# Regenerate table 3 from meintanis2004 (page 944)

law.index <- c(1); M <- 10000
vectn <- c(20,50)
level <- c(0.10)
stat.indices <- c(47,49,47,49,47,49,47,49,47,49,47,49)
law.indices <- c(2,3,4,7,35,5,6,36,36)
alter <- list(stat47=3,stat49=3,stat47=3,stat49=3,stat47=3,stat49=3,
    stat47=3,stat49=3,stat47=3,stat49=3,stat47=3,stat49=3)
parstat <- list(stat47=0.5,stat49=0.5,stat47=1.0,stat49=1.0,stat47
    =2.0,stat49=2.0,stat47=4.0,stat49=4.0,stat47=5.0,stat49=5.0,
    stat47=10.0,stat49=10.0)
parlaws <- list(law2=c(0,1),law3=c(0,1),law4=c(0,1),law7=c(0,1),
    law35=c(1),law5=c(2,1),law6=c(2,2),law36=c(0,1,2),law36=c(0,1,3))
for (n in vectn) {
  critval <- many.crit(law.index,M,n,stat.indices,level,alter,parlaw
      =NULL,parstat)
  table3 <- powcomp.fast(n,M,law.indices,stat.indices,level,critval,
      alter,parlaws,parstats=parstat,nbclus=1)
```

```
  print(print.power(table3,digits=0,latex.output=FALSE))
}
```

```
# Regenerate table 4 from meintanis2004 (page 945)

law.index <- c(1)
M <- 10000
vectn <- c(20,50)
level <- c(0.10)
stat.indices <- c(43,42,45,44,48,50,48,50,48,50,48,50,48,50,48,50)
law.indices <- c(2,3,4,7,35,5,6,36,36)
alter <- list(stat43=3,stat42=3,stat45=3,stat44=3,stat48=3,stat50=3,
    stat48=3,stat50=3,stat48=3,stat50=3,stat48=3,stat50=3,stat48=3,
    stat50=3,stat48=3,stat50=3)
parstat <- list(stat43=NA,stat42=NA,stat45=NA,stat44=NA,stat48=0.5,
    stat50=0.5,stat48=1.0,stat50=1.0,stat48=2.0,stat50=2.0,stat48
    =4.0,stat50=4.0,stat48=5.0,stat50=5.0,stat48=10.0,stat50=10.0)
parlaws <- list(law2=c(0,1),law3=c(0,1),law4=c(0,1),law7=c(0,1),
    law35=c(1),law5=c(2,1),law6=c(2,2),law36=c(0,1,2),law36=c(0,1,3))
for (n in vectn) {
  critval <- many.crit(law.index,M,n,stat.indices,level,alter,parlaw
      =NULL,parstat)
  table4 <- powcomp.fast(n,M,law.indices,stat.indices,level,critval,
      alter,parlaws,parstats=parstat,nbclus=1)
  print(print.power(table4,digits=0,latex.output=FALSE))
}
```

D.4. Testing goodness-of-fit for Laplace distribution based on maximum entropy - Choi and Kim (2006)

```
# Regenerate table 1 from choi2006 (page 523)

```

```
law.index <- c(1)
M <- 100000
vectn <- c(c(4:10),seq(12,20,2),seq(25,50,5))
level <- c(0.05)
stat.indices <- c(51,51,51,51,51,51,51,51,51)
alter <- list(stat51=4,stat51=4,stat51=4,stat51=4,stat51=4,stat51=4,
    stat51=4,stat51=4,stat51=4)
parstat <- list(stat51=1,stat51=2,stat51=3,stat51=4,stat51=5,stat51
    =6,stat51=7,stat51=8,stat51=9)
table1 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat)
print.critvalues(table1,digits=3,latex.output=FALSE)
```

```
# Regenerate table 2 from choi2006 (page 523)

law.index <- c(1)
M <- 100000
vectn <- c(c(4:10),seq(12,20,2),seq(25,50,5))
level <- c(0.05)
stat.indices <- c(53,53,53,53,53,53,53,53,53)
alter <- list(stat53=4,stat53=4,stat53=4,stat53=4,stat53=4,stat53=4,
    stat53=4,stat53=4,stat53=4)
parstat <- list(stat53=1,stat53=2,stat53=3,stat53=4,stat53=5,stat53
    =6,stat53=7,stat53=8,stat53=9)
table2 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat)
print.critvalues(table2,digits=3,latex.output=FALSE)
```

```
# Regenerate table 3 from choi2006 (page 524)

law.index <- c(1)
M <- 100000
vectn <- c(c(4:10),seq(12,20,2),seq(25,50,5))
```

```
level <- c(0.05)
stat.indices <- c(52,52,52,52,52,52,52,52,52)
alter <- list(stat52=4,stat52=4,stat52=4,stat52=4,stat52=4,stat52=4,
    stat52=4,stat52=4,stat52=4)
parstat <- list(stat52=1,stat52=2,stat52=3,stat52=4,stat52=5,stat52
    =6,stat52=7,stat52=8,stat52=9)
table3 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat)
print.critvalues(table3,digits=3,latex.output=FALSE)
```

```
# Regenerate table 5 from choi2006 (page 525)

law.index <- c(1)
M <- 50000
vectn <- c(10,20,35,50)
level <- c(0.05)
stat.indices <- c(51,53,52)
law.indices <- c(1)
alter <- list(stat51=4,stat53=4,stat52=4)
l <- list(l1=c(1,0.5),l2=c(1,2),l3=c(1,4))

for (i in 1:length(l)) {

  # cat("(mu,sigma)=",l[[i]],"\n")
  cat(paste("mu=",l[[i]][1],",","sigma=",l[[i]][2],"\n",sep=""))
  eval(parse(text=paste("parlaws <- list(",paste("law",law.indices,"
      =c(",paste(l[[i]],collapse=","),")",sep="",collapse=","),")",
      sep="")))

  for (n in vectn) {
    critval <- many.crit(law.index,M,n,stat.indices,level,alter,
        parlaw=NULL,parstat=NULL)
    table5 <- powcomp.fast(n,M,law.indices,stat.indices,level,
        critval,alter,parlaws,parstats=NULL,nbclus=1)
    print(print.power(table5,digits=1,latex.output=FALSE))
```

```
  }
}
```

```
# Regenerate table 6 from choi2006 (page 525)

law.index <- c(1)
M <- 50000
vectn <- c(10,20,35,50)
level <- c(0.05)
stat.indices <- c(51,53,52,43,44,42,45,46)
law.indices <- c(2,8,4,3,7)
alter <- list(stat51=4,stat53=4,stat52=4,stat43=3,stat44=3,stat42=3,
    stat45=3,stat46=3)
parlaws <- list(law2=c(0,1),law8=c(10),law4=c(0,1),law3=c(0,1),law7=
    c(0,1))

for (n in vectn) {
  critval <- many.crit(law.index,M,n,stat.indices,level,alter,parlaw
      =NULL,parstat=NULL)
  table6 <- powcomp.fast(n,M,law.indices,stat.indices,level,critval,
      alter,parlaws,parstats=NULL,nbclus=1)
  print(print.power(table6,digits=1,latex.output=FALSE))
}
```

```
# Regenerate table 7 from choi2006 (page 525)

law.index <- c(1)
M <- 50000
vectn <- c(10,20,35,50)
level <- c(0.05)
stat.indices <- c(51,53,52,43,44,42,45,46)
law.indices <- c(9,10,11,28)
```

```
alter <- list(stat51=4,stat53=4,stat52=4,stat43=3,stat44=3,stat42=3,
    stat45=3,stat46=3)
parlaws <- list(law9=c(2),law10=c(0,0.5),law11=c(2,1),law28=c(0,1,0)
    )

for (n in vectn) {
  critval <- many.crit(law.index,M,n,stat.indices,level,alter,parlaw
      =NULL,parstat=NULL)
  table7 <- powcomp.fast(n,M,law.indices,stat.indices,level,critval,
      alter,parlaws,parstats=NULL,nbclus=1)
  print(print.power(table7,digits=1,latex.output=FALSE))
}
```

D.5. Tests for the Goodness-of-fit of the Laplace distribution - Chen (2002)

```
# Regenerate table 8 from chen2002 (page 173)

law.index <- c(1)
M <- 100000
vectn <- c(25,50,100)
level <- c(0.10)
stat.indices <- c(45,43,42)
law.indices <- c(2,3,5,6,7)
alter <- list(stat45=3,stat43=3,stat42=3)
parlaws <- list(law2=c(0,100),law3=c(0,1),law5=c(2,1),law6=c(2,2),
    law7=c(0,1))
critval <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL)
table8 <- powcomp.fast(vectn,M,law.indices,stat.indices,level,
    critval,alter,parlaws,parstats=NULL,nbclus=1)
print.power(table8,digits=0,latex.output=FALSE)
```

D.6. GOODNESS OF FIT TEST FOR THE RAYLEIGH AND THE LAPLACE DISTRIBUTIONS - GULATI (2011)

```
# Regenerate table 6 from gulati2011

law.index <- c(1)
M <- 40000
vectn <- seq(20,100,10)
level <- c(0.10,0.05,0.025)
stat.indices <- c(59)
law.indices <- c(1)
alter <- list(stat59=3)
critval <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL)
table6 <- powcomp.fast(vectn,M,law.indices,stat.indices,level,
    critval,alter,parlaws=NULL,parstats=NULL,nbclus=1)
print.power(table6,digits=2,latex.output=FALSE)
```

```
# Regenerate table 7 from gulati2011

law.index <- c(1)
M <- 10000
vectn <- seq(20,100,10)
level <- c(0.05)
stat.indices <- c(59)
law.indices <- c(2,7,3,4,35,5,28,6)
alter <- list(stat59=3)
parlaws <- list(law2=c(0,1),law7=c(0,1),law3=c(0,1),law4=c(0,1),
    law35=c(1),law5=c(2,1),law28=c(0,1,0),law6=c(2,2))
critval <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL)
table7 <- powcomp.fast(vectn,M,law.indices,stat.indices,level,
    critval,alter,parlaws,parstats=NULL,nbclus=1)
print.power(table7,digits=2,latex.output=FALSE)
```

```
# Regenerate table 8 from gulati2011

law.index <- c(1)
M <- 10000
vectn <- c(20,50)
level <- c(0.10)
stat.indices <- c(59,43,42,45,44,48,50)
law.indices <- c(2,3,4,35,5,6,7)
alter <- list(stat59=3,stat43=3,stat42=3,stat45=3,stat44=3,stat48=3,
    stat50=3)
parlaws <- list(law2=c(0,1),law3=c(0,1),law4=c(0,1),law35=c(1),law5=
    c(2,1),law6=c(2,2),law7=c(0,1))
for (n in vectn) {
  critval <- many.crit(law.index,M,n,stat.indices,level,alter,parlaw
      =NULL,parstat=NULL)
  table8 <- powcomp.fast(n,M,law.indices,stat.indices,level,critval,
      alter,parlaws,parstats=NULL,nbclus=1)
  print(print.power(table8,digits=2,latex.output=FALSE))
}
```

D.7. Test of fit for a Laplace distribution against heavier tailed alternatives - Gel (2010)

```
# Regenerate table 2 from gel2010 (page 963)

law.index <- c(1)
M <- 100000
vectn <- c(20,50,100)
level <- c(0.05)
stat.indices <- c(43,44,42,45,46,60)
law.indices <- c(1,2,4,37,37,8,3,35,37,37)
```

```
alter <- list(stat43=3,stat44=3,stat42=3,stat45=3,stat46=3,stat60=3)
parlaws <- list(law1=c(0,1),law2=c(0,1),law4=c(0,1),law37=c
    (0.4,0,1,0.6),law37=c(0.7,0,1,0.2),law8=c(3),law3=c(0,1),law35=c
    (1),law37=c(1,0.5,1,0.43),law37=c(0.5,0.2,1,0.5))
critval <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL)
table2 <- powcomp.fast(vectn,M,law.indices,stat.indices,level,
    critval,alter,parlaws,parstats=NULL,nbclus=1)
print.power(table2,digits=2,latex.output=FALSE)
```

Annexe E

REPRODUIRE CERTAINS RÉSULTATS POUR DES TESTS D'UNIFORMITÉ DANS DES ARTICLES PUBLIÉS

E.1. Test for serial correlation in Regression Analysis based on the periodogram of least-squares residuals - Durbin (1969)

```
# Regenerate table 1 from durbin1969 (page 4)

law.index <- c(7)
M <- 10000
vectn <- c(4:100)
stat.indices <- c(66)
level <- c(0.10,0.05,0.025,0.01,0.005)*2
alter <- list(stat66=3)
table1 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL)
print.critvalues(table1,digits=3,latex.output=FALSE)
```

E.2. Some new goodness-of-fit tests using order statistics - Hegazy and Green (1975)

E-ii

```
# Regenerate table 2 from hegazy1975 (page 303)
# For T_1 and T_2 ONLY

law.index <- c(7)
M <- 10000
vectn <- c(3,8,18,38,78)
stat.indices <- c(68,69)
level <- c(0.01,0.025,0.05,0.1,0.15)
alter <- list(stat68=3,stat69=3)
table2 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL)
print.critvalues(table2,digits=4,latex.output=FALSE)
```

```
# Regenerate table 4 from hegazy1975 (page 304) (partially)

law.index <- c(7)
M <- 1000
vectn <- c(5,10,20,40,80)
level <- c(0.05)
stat.indices <- c(68,69)
law.indices <- c(2,1,35,3)
alter <- list(stat68=3,stat69=3)
critval <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL)
table4 <- powcomp.fast(vectn,M,law.indices,stat.indices,level,
    critval,alter,parlaws=NULL,parstats=NULL,nbclus=1)
print.power(table4,digits=1,latex.output=FALSE)
```

E.3. A TEST FOR UNIFORMITY BASED ON INFORMATIONAL ENERGY - PARDO (2003)

```
# Regenerate table 1 from pardo2003 (page 527)

law.index <- c(7)
M <- 10000
vectn <- c(10,20,30,40,50,100)
level <- c(0.1,0.05,0.01)
list.m <- list(n10=c(2,3,4),n20=c(2,3,4,5,6,7,8,9),n30=c
    (2,3,4,5,6,7,8,9,10),
                n40=c(2,3,4,5,6,7,8,9,10,15),n50=c
                    (2,3,4,5,6,7,8,9,10,15,20),
          n100=c(2,3,4,5,6,7,8,9,10,15,20,30,40))
for (i in 1:length(vectn)) {
  n <- vectn[i]
  stat.indices <- rep(79,length(list.m[[i]]))
  list.alter <- rep(3,length(list.m[[i]]))
  eval(parse(text=paste("alter <- list(",paste("stat",stat.indices,"
      =c(",list.alter,")",sep="",collapse=","),")",sep="")))
  eval(parse(text=paste("parstat <- list(",paste("stat",stat.indices
      ,"=c(",list.m[[i]],")",sep="",collapse=","),")",sep="")))
  table1 <- many.crit(law.index,M,n,stat.indices,level,alter,parlaw=
      NULL,parstat)
  print(print.critvalues(table1,digits=3,latex.output=FALSE))
}
```

E.4. Powerful goodness-of-fit tests based on the likelihood ratio - Zhang (2002)

```
# Regenerate table 1 from zhang2002 (page 290)

law.index <- c(7)
M <- 1000000
vectn <- c(5:10,seq(12,20,2)
    ,25,30,40,50,70,100,150,200,300,500,1000)
```

```
stat.indices <- c(81)
level <- 1 - c(0.001,0.01,0.05,seq(0.10,0.90,0.10),0.95,0.99,0.999)
alter <- list(stat81=3)
table1 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL,model=NULL)
print.critvalues(table1,digits=2,latex.output=FALSE)
```

```
# Regenerate table 2 from zhang2002 (page 291)

law.index <- c(7)
M <- 1000000
vectn <- c(5:10,seq(12,20,2)
    ,25,30,40,50,70,100,150,200,300,500,1000)
stat.indices <- c(82)
level <- 1 - c(0.001,0.01,0.05,seq(0.10,0.90,0.10),0.95,0.99,0.999)
alter <- list(stat82=3)
table2 <- many.crit(law.index,M,vectn,stat.indices,level,alter,
    parlaw=NULL,parstat=NULL,model=NULL)
print.critvalues(table2,digits=1,latex.output=FALSE)
```