Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES MATHÉMATIQUES

Tests d'uniformité de nuages de points en grande dimension basés sur les diagrammes de Voronoi

Doumont, Denis

Award date:
2003

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 02. May. 2026

FUNDP
Faculté des Sciences
Département de Mathématique

Rempart de la Vierge, 8
B-5000 Namur Belgique

Tests d'uniformité de nuages de points en grande dimension basés sur les diagrammes de Voronoi



Mémoire présenté pour l'obtention
du grade de
Licencié en Sciences Mathématiques
par

Denis Doumont

Promoteur : Marcel Rémon

Année académique 2002-2003

Table des matières

Introduction 1

1 Notions préliminaires 2
- 1.1 Processus de Poisson . . . 2
- 1.2 Cellules et diagrammes de Voronoi . . . 6
  - 1.2.1 Définitions . . . 6
  - 1.2.2 Quelques propriétés des diagrammes de Voronoi . . . 11
  - 1.2.3 Quelques propriétés des cellules de Voronoi de Poisson . . . 14

2 Test d'uniformité en une dimension sur base de cellules de Voronoi (partie théorique) 19
- 2.1 Introduction . . . 19
- 2.2 Quelques résultats sur les statistiques d'ordre . . . 20
  - 2.2.1 Préliminaire : la variable Bêta . . . 20
  - 2.2.2 Quelques résultats sur les statistiques d'ordre . . . 21
- 2.3 Caractéristiques des cellules 1-D . . . 22
  - 2.3.1 Étude à partir de la longueur des cellules . . . 23
  - 2.3.2 Densité de la longueur des cellules en terme de Bêta . . . 23
  - 2.3.3 Densité de la longueur des cellules . . . 23
  - 2.3.4 Densité de la longueur des cellules en terme de Gamma . . . 25

3 Problèmes liés à la méthode utilisée dans ce travail 28
- 3.1 Introduction . . . 28
- 3.2 L'approche séquentielle . . . 28
- 3.3 L'approche non séquentielle . . . 29
  - 3.3.1 Le problème des effets de bord . . . 30
  - 3.3.2 Deuxième solution : l'espace torique . . . 31
  - 3.3.3 Le problème de la dépendance des cellules . . . 34
- 3.4 Quadrillage . . . 34
  - 3.4.1 Calcul du nombre de points-grille . . . 34
  - 3.4.2 À propos des distances toriques . . . 36
  - 3.4.3 Précision du quadrillage . . . 37

4 Nombres aléatoires 40
- 4.1 Introduction . . . 40
- 4.2 Générateurs utilisés . . . 40

5 Test d'uniformité en une dimension sur base de cellules de Voronoi (partie pratique) **43**
5.1 Quelques tests 43

6 Explication des algorithmes **47**
6.1 Complexité de l'algorithme `dVOR` 47
6.2 Justification de l'implémentation 48
6.3 Vérifications empiriques du programme 51

7 Test d'uniformité en d dimensions sur base de cellules de Voronoi **53**
7.1 Introduction 53
7.2 Test en 2-D 54
7.3 Test 3-D 55
7.4 Test 4-D 56
7.5 Test 5-D 59
7.6 Test 6-D 60
7.7 Conclusion pour ces tests 61
7.8 Analyse de la précision des résultats 61
7.8.1 Test 2-D 62
7.8.2 Test 3-D 62
7.8.3 Test 4-D 63
7.8.4 Conclusion 64

8 Test de non-uniformité en d dimensions sur base de cellules de Voronoi **65**
8.1 Tests 2-D 65
8.1.1 Caractéristiques des tests 65
8.1.2 Échantillon 2-D en forme de disque centré 67
8.1.3 Échantillon 2-D avec un trou en forme de disque centré 69
8.1.4 Échantillon 2-D avec une couronne centrée 70
8.1.5 Échantillon 2-D avec un trou central en forme de couronne . . 71
8.1.6 Échantillon 2-D formé de deux disques disjoints 73
8.1.7 Échantillon 2-D formé de deux trous en forme de disques disjoints 74
8.1.8 Échantillon 2-D formé de deux bandes horizontales disjointes 75
8.1.9 Conclusions pour les tests 2-D 76
8.2 Test 3-D 77
8.2.1 Caractéristiques des tests 77
8.2.2 Échantillon 3-D en forme de sphère centrée 77
8.2.3 Échantillon 3-D avec un trou en forme de sphère centrée . . . 79
8.2.4 Échantillon 3-D avec une couronne centrée 80
8.2.5 Échantillon 3-D avec un trou central en forme de couronne . . 81
8.2.6 Échantillon 3-D formé de deux sphères disjointes 82
8.2.7 Échantillon 3-D formé de deux trous en forme de sphères disjointes 83
8.2.8 Échantillon 3-D formé de deux bandes horizontales disjointes 84
8.2.9 Conclusions pour les tests 3-D 85

8.3 Test 4-D 86
8.3.1 Caractéristiques des tests 86
8.3.2 Échantillon 4-D en forme d'hypersphère centrée 86
8.3.3 Échantillon 4-D avec un trou en forme d'hypersphère centrée 88
8.3.4 Échantillon 4-D avec une couronne centrée 90
8.3.5 Échantillon 4-D formé de deux hypersphères disjointes 91
8.3.6 Échantillon 4-D formé de deux bandes horizontales disjointes 92
8.4 Test 5-D 93
8.4.1 Échantillon 5-D formé de deux bandes horizontales disjointes 93
8.4.2 Conclusions pour les tests 4-D et 5-D 94

Conclusion **95**

A Listings des programmes utilisés **98**
A.1 Programme Splus pour les tests en une dimension 98
A.2 Programme dVOR en C++ 101

B Détails informatiques **121**
B.1 Informations sur les machines utilisées pour les programmes 121
B.2 Limites définies par l'implémentation 122

Bibliographie **122**

Résumé

Dans ce mémoire, nous proposons d'expérimenter un test d'uniformité insolite pour un nuage de points inclus à $\mathbb{R}^n$. Nous utilisons en effet les caractéristiques des cellules de Voronoi construites à partir de ces points pour réaliser ce test d'uniformité.
Nous effectuons une analyse théorique de ce problème en une dimension, que nous complétons par deux simulations pratiques. Nous expliquons aussi les problèmes liés à cette méthode.
Puis nous effectuons des simulations en deux, trois, quatre, cinq et même six dimensions pour des ensembles de données uniformes et non uniformes. Ces simulations nous permettent de juger la pertinence du test proposé.

Abstract

In this paper we propose an unusual test of uniformity for a set of points in $\mathbb{R}^n$. Indeed, we use the characteristics of the Voronoi cells build form these points to realize this uniformity test.
We first do an theorical analysis of the problem in the 1-dimensional case. Then we complete this by two simulations. We also show what are the problems with this approach.
Next, we realize some simulations in the 2-, 3-, 4-, 5- and 6-dimensional cases for various sets of data. Thanks to these simulations, we can test the feasibility of this approach.

Merci !

Je remercie avant tout mon promoteur, Marcel Rémon, pour sa patience, toutes les corrections qu'il a apportées à ce travail et pour les remarques désobligeantes qu'il ne m'a jamais adressées.

La deuxième personne que je remercie ici est Fabian Bastin, pour toute son « assistance technique », sa patience également, et le prêt de sa machine `sphere` au département que j'ai fait tourner plusieurs dizaines d'heures. Je jure par ailleurs que je lui avais promis la deuxième place dans ma liste de remerciements *avant* de savoir qu'il serait lecteur.

Je remercie également M. Masaharu Tanemura, de l'institut de mathématiques statistiques de Tokyo, Minato-ku, qui par un échange de courriel rapide et très courtois m'a aidé à établir la preuve de la loi statistique Gamma de la longueur des cellules de Voronoi en une dimension.

Je remercie aussi Jean-Yves Pirçon, Julie Vandooren, Caroline Sainvitu, Bertrand Chenal, Stéphane Valk, frère Alexis et Katia Demaseure pour leur contribution directe, même modeste, à ce mémoire. Et s'ils ont oublié pourquoi, moi je le sais.

Je remercie par ailleurs Donald Knuth, Leslie Lamport, et tous les collaborateurs au projet GNU pour les outils magnifiques qu'ils nous *donnent*.

Je remercie également toute la société, car quand on pense à elle, c'est en général pour la critiquer. Éh bien moi je suis bien content d'avoir reçu toute cette éducation et je remercie toutes les personnes qui y ont pris part de près ou de loin.[1]

Enfin, je remercie ma famille, ainsi qu'à ceux et à Celui qui prennent souci de moi (dans tous les sens du terme). Je considère personnellement que les dernières places sont toujours les places d'honneur, c'est pourquoi c'est en dernier que je les cite.

Je dédie ce mémoire à Marie et Joseph : ils savent pourquoi.

[1] À ce propos, que le lecteur ne soit pas étonné de l'orthographe de certains mots, notamment ceux qui auraient perdu dans ce document leur accent circonflexe... C'est que j'ai à peu près écrit ce document en tenant compte des dernières rectifications orthographiques approuvées par l'Académie française en 1990, déjà.

Introduction

La notion de diagramme de Voronoi est connue depuis longtemps. C'est un outil fascinant pour lequel de nouvelles applications sont sans cesse découvertes, aussi bien en géographie qu'en statistiques. Nous allons utiliser ici cet outil pour résoudre un problème simple dans son concept de la théorie statistique : comment savoir si un ensemble de points donnés est réparti uniformément ?

Nous nous attacherons dans ce travail à calculer l'aire, le volume ou l'hypervolume (suivant la dimension) des cellules de Voronoi construites à partir de ce nuage de points. Ensuite, le but sera de déterminer un test non paramétrique qui nous permettra, sur base de cette information, de répondre à la question ci-dessus.

Nous commencerons par donner quelques rappels précis sur la notion de diagramme de Voronoi. Nous donnerons également dans ce chapitre quelques résultats de recherches statisitques effectuées en deux et trois dimensions sur le volume des cellules.

Puis, dans le deuxième chapitre, nous chercherons quelle est la distribution de la longueur des cellules, dans un espace en une dimension. Nous verrons que cette distribution obéit à une loi statistique connue, mais nous constaterons aussi que ces développements nous conduisent déjà à considérer des lois de distributions peu usuelles.

Ensuite nous énoncerons dans le chapitre suivant quelques problèmes directs que nous rencontrons dans la méthode de travail ici choisie. Ces problèmes sont importants, et leur résolution, si elle est simple en théorie, n'est pas toujours évidente à mettre en œuvre en pratique.

Puisque nous avons développé précédemment la théorie concernant des test d'uniformité en une dimension, nous complétons dans le chapitre quatre cela par des réalisations de tests d'uniformité.

Enfin, les deux derniers chapitres sont consacrés à des réalisations de test d'uniformité et de non-uniformité en dimensions supérieures. Nous effectuerons autant que possible une analyse graphique et statistique de chacun de ces tests, et nous éprouverons là en pratique cette idée de départ qui est de tester l'uniformité de données sur base des cellules de Voronoi.

Chapitre 1

Notions préliminaires

1.1 Processus de Poisson

Un processus de points stochastique est une génération probabilistique de points dans un espace selon une fonction de distribution de probabilité définie sur l'espace. Commençons par définir la notion de processus de points binomial. Soit $S \in \mathbb{R}^d$. Notons $m(S)$ la mesure de Lebesgue de la partie S. Dans $\mathbb{R}^d$, on a que $m(S)$ représente la surface de S (si $d = 2$) ou l'hypervolume de S (si $d > 2$, $m(S) < \infty$). Dans un souci de généralité, nous garderons la notation $m(S)$.

Définition 1.1 (processus binomial)
Un processus binomial est un processus dans lequel les points sont générés dans S selon la distribution uniforme.

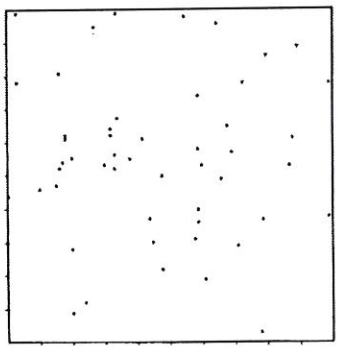


FIG. 1.1: *Un exemple de processus de points binomial ($n = 50$)*

Intuitivement, la distribution uniforme signifie qu'un point n'a pas plus de chances de se trouver là où il est que partout ailleurs. On pourrait aussi dire : tous les points peuvent se trouver partout, à n'importe quel endroit, avec la même probabilité. Mais cela est délicat, car même dans une région bornée de $\mathbb{R}$, il y a une infinité d'« endroits » possibles, ce qui donne une probabilité $1/\infty$.

D'où vient ce nom ? Soit n le nombre total de points dans S générés par un processus binomial, A, un sous-ensemble de S et $N(A)$, le nombre de points dans A.

Alors la probabilité d'avoir un point placé dans A est $\frac{m(A)}{m(S)}$. On trouve aussi :

$$P[N(A) = x] = C_n^x \left(\frac{m(A)}{m(S)}\right)^x \left(1 - \frac{m(A)}{m(S)}\right)^{n-x}. \tag{1.1}$$

La forme de cette probabilité est une distribution binomiale, de paramètres n et $p = \frac{m(A)}{m(S)}$.

Définition 1.2 (processus stochastique)
Un processus stochastique est un processus dans lequel les points sont générés selon une loi de probabilité $P[N(A) = x]$, où $x = 0, 1, \ldots$ $\forall A \subset S$.

Définition 1.3 (processus stochastique simple)
Un processus stochastique simple est tel que la probabilité d'avoir deux points qui coïncident est nulle.

Définition 1.4 (processus stochastique localement fini)
Un processus stochastique localement fini est tel que $\forall A \subset \mathbb{R}^d$ (A région finie), A contient un nombre fini de points.

Dans la suite, on utilisera toujours des processus stochastiques simples et localement finis.

Parmi les différents processus stochastiques simples et localement finis, le plus fondamental est le processus de Poisson.

Définition 1.5 (processus de Poisson)
Un processus stochastique P est un processus de Poisson sur $A \subset S \equiv \mathbb{R}^d$ ssi

(i) $\forall A_1, \ldots, A_n \subset A$ *disjoints, alors les variables aléatoires* $N(A_1), \ldots, N(A_n)$ *sont indépendantes. ($N(A_i)$ est le nombre de points contenus dans A_i.)*

(ii)

$$P[N(A) = k] = \frac{e^{-\mu(A)} (\mu(A))^k}{k!}, \tag{1.2}$$

où μ est une mesure qui donne une masse finie à tout ensemble borné, et $k = 0, 1, 2, \ldots$

Le point *(i)* de la définition veut dire que dans un processus de Poisson, il n'y a pas d'interactions entre les points (en anglais : *independent scattering* – dispersion indépendante).

Définition 1.6 (processus de Poisson homogène)
P est un processus de Poisson homogène (ou stationnaire) sur $A \subset \mathbb{R}^d$ ssi

(i) P est un processus de Poisson sur A

(ii) $\mu(A) = \lambda m(A)$
où $m(A)$ est la mesure de Lebesgue de A, et $\lambda \in \mathbb{R}_0^+$ est l'intensité du processus.

Remarquons que $\mu(A) = \lambda m(A)$ détermine complètement le processus.

De plus, si on remplace la constante λ par une fonction $\lambda(x)$, $x \in \mathbb{R}^d$, alors le processus de Poisson est dit non homogène. λ est aussi appelé la *densité* du processus.

Le processus de Poisson homogène est un modèle naturel pour des points distribués aléatoirement et indépendamment dans l'espace.

Ce processus est également la limite du processus binomial (la limite au sens de l'expansion de la région finie S à une région infinie de même densité, c.-à-d. $n \to \infty$ tel que $\frac{n}{m(S)}$ reste constant, où n est le nombre total de points).

Venons-en à la caractérisation du Processus de Poisson homogène.

Propriété 1. $N(A)$ a une distribution de Poisson de moyenne $\lambda m(A)$, c.-à-d. $N(A) \sim Po(\lambda m(A))$.

Propriété 2 (Uniformité conditionnelle). Si $N(A) = n$, alors les n réalisations sont indépendantes et forment un échantillon aléatoire d'une loi uniforme sur A.

Propriété 3. $0 < P[N(A) = 0] < 1$ si $0 < m(A) < \infty$.

Propriété 4. La distribution de probabilité de $N(A)$ dépend de $m(A)$ et $P[N(A) \geq 1] \to 0$ lorsque $m(A) \to 0$.

Propriété 5.

$$\lim_{m(A)\to 0} \frac{P[N(A) \geq 1]}{P[N(A) = 1]} = 1.$$

La propriété 3 signifie que si une région A a un hypervolume positif, la probabilité d'avoir un point dans A n'est pas nulle.

La propriété 4 est une conséquence directe du fait que $N(A) \sim Po(\lambda m(A))$; elle stipule que $P[N(A) = x]$ ne dépend pas de l'emplacement de A (dans $\mathbb{R}^d$) mais seulement de son hypervolume. Les points n'ont pas tendance à occuper une place particulière dans $\mathbb{R}^d$. Cette propriété est aussi appelée *propriété d'équivariance*, dans le sens où $P[N(A_1) = x] = P[N(A_2) = x]$ si $m(A_1) = m(A_2)$.

Enfin, la propriété 5 revient à dire que le processus est simple.

Notons encore que le processus binomial ne vérifie pas la propriété 3, car si x points sont placés dans A, alors nécessairement $n - x$ points sont placés dans $S \setminus A$.

On peut montrer que P est un processus de Poisson homogène si et seulement si P vérifie la partie *(i)* de la définition 1.5 ainsi que les propriétés 3, 4 et 5.

Dans la suite, nous travaillerons toujours avec des Processus de Poisson homogènes.

Terminons cette section par quelques définitions provenant de [11, p. 33 et ss].

Définition 1.7 (position quadratique générale)
Les points d'un processus stochastique sont en position quadratique générale ssi ils satisfont

- *l'hypothèse de non-colinéarité :* $\nexists$ $(k+1)$ *points se trouvant sur un hyperplan* $(k-1)$*-dimensionnel de* $\mathbb{R}^d$ $(2 \leq k \leq d)$,

- *l'hypothèse de non-cosphéricité :* $\nexists\ (d+2)$ *points se trouvant sur la frontière d'une sphère dans* $\mathbb{R}^d$.

Définition 1.8 (processus stochastique homogène ou stationnaire)
Un processus stochastique homogène est un processus satisfaisant

$$P[N(A_1) = x_1, \ldots, N(A_k) = x_k] = P[N(\varphi_c(A_1)) = x_1, \ldots, N(\varphi_c(A_k)) = x_k], \quad (1.3)$$

où $S \subset \mathbb{R}^d,\ A_i \subset S,\ \varphi_c(A_i) \subset S \quad i \in \underline{k},\ k = 1, 2, \ldots$
$\varphi_c(A) = \{\, \mathbf{x} + \mathbf{c} \mid \mathbf{x} \in A \,\} \quad A \subset S,\ \varphi_c(A) \subset S,$
$\mathbf{c} \in \mathbb{R}^d$ *vecteur constant.*

La fonction $\varphi_c(A)$ correspond à une translation de A dans la direction du vecteur $\mathbf{c}$. Les processus binomial et de Poisson homogène vérifient l'équation (1.3), ils sont donc homogènes par définition. Cela peut se voir intuitivement, à partir du fait que ces processus sont dérivés de la loi uniforme.

Définition 1.9 (processus stochastique isotrope)
Un processus stochastique isotrope est un processus vérifiant

$$P[N(A_1) = x_1, \ldots, N(A_k) = x_k] = P[N(\varphi_\theta(A_1)) = x_1, \ldots, N(\varphi_\theta(A_k)) = x_k], \quad (1.4)$$

où $S \subset \mathbb{R}^d,\ A_i \subset S,\ \varphi_\theta(A_i) \subset S \quad i \in \underline{k},\ k = 1, 2, \ldots$
$\varphi_\theta(A) = \{\, B\,\mathbf{x} \mid \mathbf{x} \in A,\ \det B = 1,\ B^T B = 1 \,\} \quad A \subset S,\ \varphi_\theta(A) \subset S,$
B *matrice* $d \times d$.

La fonction $\varphi_\theta(A)$ correspond à une rotation de A autour de l'origine dans $\mathbb{R}^d$. On peut voir que le processus de Poisson homogène est toujours isotrope.

Une *transformation* φ_v de $A \subset S$ est dite « *volume preserving* » si $m(\varphi_v(A)) = m(A),\ A \subset S,\ \varphi_v(A) \subset S$, où $m(\cdot)$ est la mesure de Lebesgue. (Dans $\mathbb{R}^d$, $m(A)$ représente l'hypervolume de A et est aussi notée dans la littérature par $|A|$). Les rotations et translations sont des transformations « volume preserving ».

Définition 1.10 (processus stochastique « volume preserving »)
Un processus stochastique « volume preserving » est un processus satisfaisant

$$P[N(A_1) = x_1, \ldots, N(A_k) = x_k] = P[N(\varphi_v(A_1)) = x_1, \ldots, N(\varphi_v(A_k)) = x_k], \quad (1.5)$$

où $S \subset \mathbb{R}^d,\ A_i \subset S, \theta_c(A_i) \subset S\ \ i \in \underline{k},\ k = 1, 2, \ldots$
φ_v *est une transformation « volume preserving ».*

Les processus binomial et de Poisson homogène sont « volume preserving ».

Ceci nous amène à mettre en évidence une dernière propriété du processus de Poisson homogène, importante pour la suite.

Propriété 6. Le processus de Poisson homogène est invariant aux translations et $N(A) = k^d N(Y)$ si $A = k^d Y$ car $m(A) = k^d m(Y)$, où d est la dimension de l'espace.

Cette propriété est appelée propriété d'*équivariance.*

1.2 Cellules et diagrammes de Voronoi

1.2.1 Définitions

Nous allons tout d'abord définir la notion de cellules de Voronoi dans le plan, car c'est dans ce cas qu'elle est la plus facile à appréhender. Supposons que l'on dispose d'un ensemble de points dans $\mathbb{R}^2$, au moins deux points mais en nombre fini, et que tous les points soient distincts. Notons ces points $p_1, p_2, \ldots$ Cet ensemble de points donné, on assigne chaque point du plan au point p_i le plus proche. Si un point du plan est aussi proche d'un point p_k que d'un point p_l (ou encore davantage de points p_i), il est assigné à tous ces points à la fois. Bref, pour chaque point p_i, on obtient un ensemble de points du plan (une région) qui sont plus proches de ce point que de tout autre point p_j de l'ensemble de points donnés À la figure 1.2, le point p appartient à la cellule relative au point p_1, car p est plus proche de p_1 que de tout autre point p_i $(i \neq 1)$.

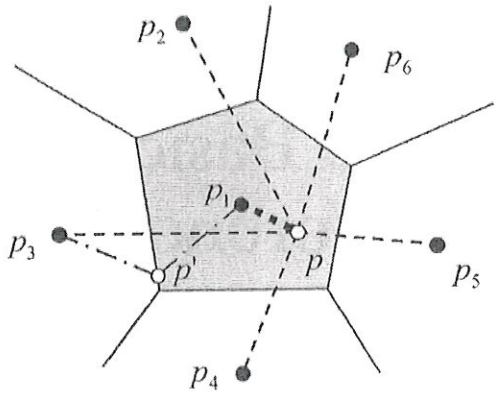


FIG. 1.2: *Définition de la notion de cellule de Voronoi*

L'ensemble des points assignés à deux points p_i ou plus forme les frontières des régions. C'est le cas du point p' à la figure 1.2 : il est à équidistance entre les points p_1 et p_3, il appartient donc aux cellules relatives à ces deux points à la fois, il se trouve sur la frontière commune à ces deux cellules. L'ensemble des régions forme le plan tout entier, puisque chaque point du plan est assigné à un point p_i. De plus, les régions sont disjointes entre elles (excepté à leurs frontières). On dit dès lors que l'ensemble des régions forme une *tessellation*. On appelle ici cette tessellation un *diagramme de Voronoi plan*, et les régions constituant ce diagramme, des *cellules* ou *polygones*[1] *de Voronoi*.

> Une autre manière de considérer les cellules de Voronoi est la suivante.
> Chacun des points est vu comme un noyau à partir duquel se développe une cellule. Ces cellules commencent leur développement au même moment, croissent au même rythme et de manière isotrope. Lorsque deux cellules se « rencontrent », la croissance s'arrête pour ces deux cellules dans la direction de leur rencontre uniquement. Pendant que la croissance continue dans les autres directions, la frontière entre ces deux cellules se propagera perpendiculairement à la direction de leur rencontre. Le processus se termine quand les cellules occupent tout le plan.

[1] Puisqu'il s'agit toujours de polygones ; cela sera mis en évidence dans la définition 1.13.

Cette description est équivalente à la première. En effet, un point p du plan fera partie de la cellule issue du noyau p_i si le temps t_i pris par le noyau i pour atteindre ce point est inférieur à celui que prend tout autre noyau. Soit v la vitesse de croissance et $d(p,p_i)$, la distance euclidienne entre le noyau p_i et le point p. Le temps t_i vaut $t_i = d(p,p_i)/v$. Puisque la vitesse de croissance est constante pour tous les noyaux, on a que $\forall\, i \neq j \quad t_i < t_j \Rightarrow d(p,p_i) < d(p,p_j)$.
Ce processus de croissance peut modéliser certains phénomènes physiques : les cellules d'un tissu, les cristaux d'un minerai, différents types de végétation recouvrant une région, ou tout autre phénomène donnant lieu à une image à structure cellulaire. Dans des espaces de dimension supérieure, il peut représenter des régions textures, etc. Notons que le modèle correspond mieux au monde minéral que végétal. En effet, il est plus réaliste pour différents types de végétation de considérer des vitesses de croissance différentes pour chaque noyau[2]. Mais cela n'entre pas dans le cadre de ce travail.

• **Notations dans $\mathbb{R}^2$.**
Nous considérons un nombre fini n de points dans $\mathbb{R}^2$ $(2 \leq n < \infty)$, notés $p_1, p_2, \ldots, p_n$, avec les coordonnées cartésiennes $(x_{11}, x_{12}), \ldots, (x_{n1}, x_{n2})$, notées aussi sous forme de vecteurs $\mathbf{x_1}, \ldots, \mathbf{x_n}$. Les n points sont distincts : $\mathbf{x_i} \neq \mathbf{x_j} \quad \forall\, i \neq j, \quad i,j \in \{1,\ldots,n\} \stackrel{\text{not}}{=} \underline{n}$. On travaille avec la norme euclidienne : $d(p,p_i) = \|\mathbf{x} - \mathbf{x_i}\| \stackrel{\text{def}}{=} \sqrt{(x_1 - x_{1i})^2 + (x_2 - x_{2i})^2}$.

Définition 1.11 (cellules et diagramme de Voronoi dans $\mathbb{R}^2$)
Soient $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2 \quad (2 \leq n < \infty)$ *et* $\mathbf{x_i} \neq \mathbf{x_j} \quad \forall\, i \neq j, \quad i,j \in \underline{n}$. *La cellule (ou polygone) de Voronoi associée à* p_i *est définie par*

$$V(p_i) = \{\, \mathbf{x} \mid \|\mathbf{x} - \mathbf{x_i}\| \leq \|\mathbf{x} - \mathbf{x_j}\| \quad \forall\, j \in \underline{n},\ j \neq i \,\}. \tag{1.6}$$

Le diagramme (plan) de Voronoi généré par P *est défini par*

$$\mathcal{V} = \{\, V(p_1), \ldots, V(p_n) \,\}. \tag{1.7}$$

On dira que p_i est le *(point) générateur* de $V(p_i)$, la i^{e} cellule de Voronoi, et P l'*ensemble générateur* du diagramme $\mathcal{V}$ de Voronoi.

[2] On peut vraisemblablement supposer que le rythme de croissance des différentes espèces n'est pas le même.

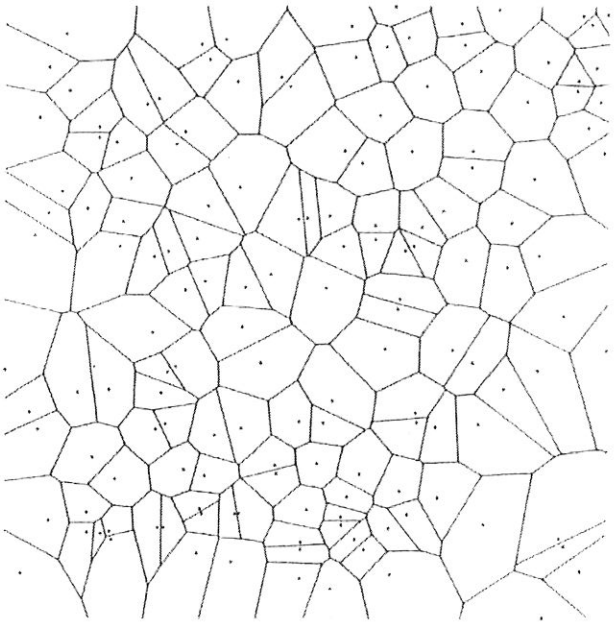


FIG. 1.3: *Une portion de diagramme de Voronoi dans* $\mathbb{R}^2$

Pour la commodité, on écrira V_i au lieu de $V(p_i)$. Notons que dans la définition 1.11, on utilise l'inégalité $\leq$ et non $<$: une cellule de Voronoi est donc un ensemble fermé. La frontière d'une cellule de Voronoi consiste en segments de ligne, demi-lignes ou lignes infinies que nous appellerons les *arêtes* de la cellule. Les arêtes seront notées par e_i. La frontière d'une cellule sera notée $\partial V(p_i)$. On parlera aussi de *sommet* d'une cellule (en anglais *vertex*) : il s'agit du point à l'extrémité d'une arête. On peut aussi définir le sommet par le point intersection de trois cellules de Voronoi ou plus. On notera un sommet par q_i.

Définition 1.12 (diagramme de Voronoi plan dégénéré)
Un diagramme de Voronoi $\mathcal{V}$ est dégénéré lorsqu'il existe au moins un sommet qui est l'intersection de quatre cellules ou plus.

Les deux diagrammes à la figure 1.4 sont dégénérés.

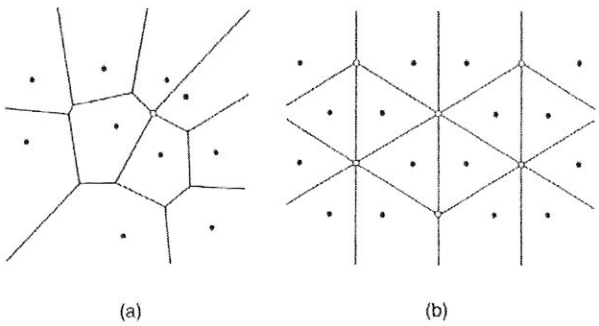


FIG. 1.4: *Deux diagrammes de Voronoi dégénérés*

Lorsque les points générateurs sont régulièrement espacés, comme dans la figure 1.4 (b), on obtient souvent un diagramme dégénéré. Dans la suite, on ne traitera que des diagrammes non-dégénérés. Pour cela, on pose une hypothèse de travail :

Hypothèse 1 (Non dégénérescence)
Tout sommet dans un diagramme de Voronoi est exactement l'intersection de trois arêtes.

Remarquons que dans la définition 1.11, on définit le concept de diagramme de Voronoi dans un plan infini. Dans la pratique, bien évidemment, on travaille sur une région bornée S contenant tous les points générateurs. On considère alors le diagramme borné par S :

$$\mathcal{V}_S = \{ V(p_1) \cap S, \ldots, V(p_n) \cap S \}.$$

Dans la pratique, on utilisera toujours un diagramme de Voronoi « bien formé », c.-à-d. où les cellules ne sont pas déconnectées en plusieurs morceaux ou non convexes (cf. figure 1.5), ce qui exige la convexité de S.

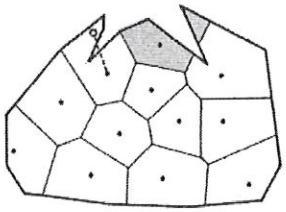


FIG. 1.5: *Diagramme de Voronoi avec cellules non convexes et une cellule « en plusieurs morceaux »*

Introduisons maintenant une autre définition de diagramme de Voronoi. Soient deux points générateurs p_i et p_j. On considère la *médiatrice* entre ces points, notée $b(p_i, p_j)$ (en anglais *bisector*). On a que

$$b(p_i, p_j) \stackrel{\text{def}}{=} \{ \mathbf{x} \mid \|\mathbf{x} - \mathbf{x}_i\| = \|\mathbf{x} - \mathbf{x}_j\| \} \quad (i \neq j). \tag{1.8}$$

La médiatrice $b(p_i, p_j)$ est la droite qui passe par le milieu du segment $[p_i, p_j] \stackrel{\text{not}}{=} \overline{p_i p_j}$, et perpendiculairement à ce segment. Par définition, un point sur cette médiatrice est équidistant aux points générateurs p_i et p_j.

La médiatrice coupe le plan en deux demi-plans ; on appelle

$$H(p_i, p_j) = \{ \mathbf{x} \mid \|\mathbf{x} - \mathbf{x}_i\| \leq \|\mathbf{x} - \mathbf{x}_j\| \} \quad (i \neq j). \tag{1.9}$$

la *région de dominance* de p_i sur p_j.

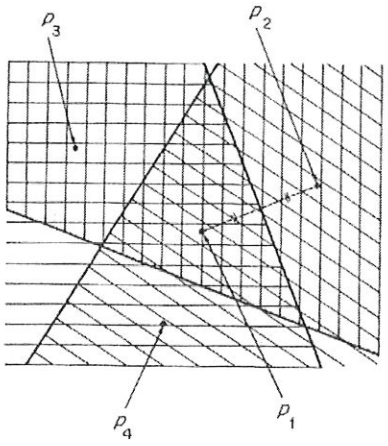


FIG. 1.6: *Une cellule de Voronoi définie par des demi-plans*

À la figure 1.6, on a indiqué les régions de dominance de p_1 sur p_2 (hachure horizontale), de p_1 sur p_3 (diagonale) et de p_1 sur p_4 (verticale). Dans la région $H(p_i, p_j)$, la distance au générateur p_i est plus petite que celle au générateur p_j. Ainsi, $\bigcap_{j=2}^{4} H(p_1, p_j)$ donne le polygone V_1.

Définition 1.13 (cellules et diagramme de Voronoi dans $\mathbb{R}^2$ – autre définition) *Soient* $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2 \quad (2 \leq n < \infty)$ *et* $\mathbf{x}_i \neq \mathbf{x}_j \quad \forall\, i \neq j, \quad i, j \in \underline{n}$. *La cellule (ou polygone) de Voronoi associée à* p_i *est définie par*

$$V(p_i) = \bigcap_{j \in \underline{n} \setminus \{i\}} H(p_i, p_j). \tag{1.10}$$

Le diagramme (plan) de Voronoi généré par P *est défini par*

$$\mathcal{V} = \{V(p_1), \ldots, V(p_n)\}.$$

Les deux définitions sont équivalentes car $\|\mathbf{x} - \mathbf{x}_i\| \leq \|\mathbf{x} - \mathbf{x}_j\|$ ssi $x \in H(p_i, p_j)$ pour $j \neq i$.

On peut maintenant étendre cette définition à l'espace à d dimensions.

Définition 1.14 (cellules et diagramme de Voronoi dans $\mathbb{R}^d$) *Soient* $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d \quad (2 \leq n < \infty)$ *et* $\mathbf{x}_i \neq \mathbf{x}_j \quad \forall\, i \neq j, \quad i, j \in \underline{n}$. *La cellule (ou polyèdre) de Voronoi associée à* p_i *est définie par*

$$V(p_i) = \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{x}_i\| \leq \|\mathbf{x} - \mathbf{x}_j\| \quad \forall\, j \in \underline{n},\ j \neq i\} \tag{1.11}$$

ou équivalemment $V(p_i) = \bigcap_{j \in \underline{n} \setminus \{i\}} H(p_i, p_j),$ (1.12)

où $H(p_i, p_j)$ *est donné par l'équation (1.9). Et l'ensemble* $\mathcal{V} = \{V(p_1), \ldots, V(p_n)\}$ *est appelé le diagramme de Voronoi généré par* P.

Dans $\mathbb{R}^3$, on parlera de *faces* pour les frontières d'une cellule.

Enfin, il nous reste à examiner, pour les besoins de ce travail, le cas des cellules en une dimension. On peut dire que la détermination des cellules en une dimension est un cas particulier du cas en deux dimensions, lorsque tous les points sont alignés.

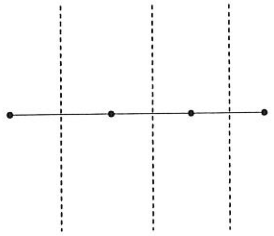


FIG. 1.7: *Diagramme de Voronoi dans $\mathbb{R}^2$ en cas de colinéarité*

Définition 1.15 (cellules de Voronoi en une dimension)
Soient $P = \{x_1, \ldots, x_n\} \subset \mathbb{R}$ un échantillon de n points $(2 < n < \infty)$ distincts $(i \neq j \quad \forall\, i,j \in \underline{n})$. Chaque point x_i est défini par la coordonnée spatiale x_i. La cellule (ou polygone) de Voronoi associée à x_i est définie par

$$V_i = V(x_i) = \{\, x \in \mathbb{R} \mid |x - x_i| \leq |x - x_j| \quad \forall\, j \in \underline{n},\ j \neq i \,\}. \tag{1.13}$$

Le diagramme de Voronoi généré par P est toujours défini par $\mathcal{V} = \{\, V(x_1), \ldots, V(x_n) \,\}$.

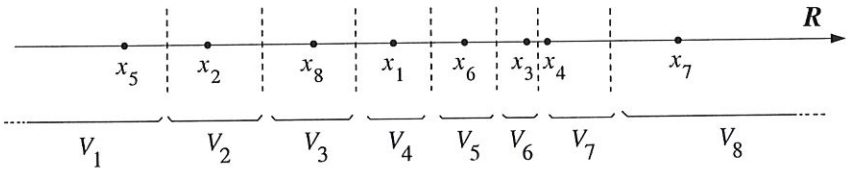


FIG. 1.8: *Exemple de diagramme de Voronoi en une dimension*

Par définition, une cellule est segment de droite fermé, ou une demi-droite fermée. Bien sûr en pratique, on prendra toujours un sous-ensemble $S \in \mathbb{R}$ fermé comme ensemble de départ, de sorte que toutes les cellules soient bornées. La frontière d'une cellule se limite ici à un seul point, qu'on appellera aussi *extrémité* de la cellule (et non plus arête). Les sommets d'une cellule se confondent avec ses extrémités. Toutes les cellules sont disjointes entre elles sauf à leur frontière. La tessellation est toujours unique (pour l'échantillon P). Remarquons encore que par construction, le point x_i n'est pas forcément « au milieu » de la cellule $V(x_i)$.

1.2.2 Quelques propriétés des diagrammes de Voronoi

Venons-en à donner quelques propriétés des diagrammes de Voronoi. On parle bien ici des diagrammes de Voronoi dans un plan infini. La plupart des résultats donnés ici concernent les diagrammes plans, mais certains d'entre eux peuvent être élargis aisément à des diagrammes d-dimensionnels.

Remarquons d'emblée que les cellules de Voronoi sont toujours des polygones[3]. Chaque cellule est un ensemble convexe[4] non vide[5]. Par définition, les cellules sont disjointes entre elles, sauf à leurs frontières et elles forment un recouvrement du plan

[3]Ceci vient du fait que les cellules de Voronoi peuvent alternativement être définies par l'intersection de demi-plans, et donc, elles ne peuvent être que des polygones.

[4]Car un demi-plan est convexe et l'intersection d'ensembles convexes est un convexe.

[5]Puisqu'une cellule contient au moins son point générateur et que ces points sont tous distincts.

euclidien[6]. Tout cela peut se résumer par la Propriété 1 :

Propriété 1. Soient $P = \{ p_1, \ldots, p_n \} \subset \mathbb{R}^2 \quad (2 \leq n < \infty)$ un ensemble de points distincts. Alors

$$V(p_i) = \{ \mathbf{x} \mid \|\mathbf{x} - \mathbf{x}_i\| \leq \|\mathbf{x} - \mathbf{x}_j\| \quad \forall\, j \in \underline{n},\ j \neq i \}$$

est un polygone convexe non vide, et $\mathcal{V} = \{ V(p_1), \ldots, V(p_n) \}$ est tel que

$$\bigcup_{i=1}^{n} V(p_i) = \mathbb{R}^2,$$

$$\{ V(p_i) \setminus \partial V(p_i) \} \cap \{ V(p_j) \setminus \partial V(p_j) \} = \emptyset \qquad i \neq j \quad i, j \in \underline{n}.$$

Le diagramme de Voronoi $\mathcal{V}(P)$ est donc une tessellation unique de $\mathbb{R}^2$ pour P.

Cette propriété est également vraie sur $\mathbb{R}^d$ (en remplaçant le terme « polygone » par « polyèdre »).

Puisque le diagramme de Voronoi recouvre tout le plan euclidien, il est évident qu'il existe au moins une cellule non bornée. La propriété suivante permet de repérer ces cellules non bornées.

Propriété 2. Soit un diagramme de Voronoi généré par un ensemble de points distincts $P = \{ p_1, \ldots, p_n \} \subset \mathbb{R}^2 \quad (2 \leq n < \infty)$, tel que les points p_i ne sont pas colinéaires. Une cellule $V(p_i)$ est non bornée si et seulement si p_i est sur la frontière de l'enveloppe convexe de P.

Si les points étaient colinéaires, alors toutes les cellules de Voronoi seraient non bornées (cf. figure 1.7).

Cette propriété est aussi vraie sur $\mathbb{R}^d$.

Propriété 3. Soit un diagramme de Voronoi généré par un ensemble de points distincts $P = \{ p_1, \ldots, p_n \} \subset \mathbb{R}^2 \quad (2 \leq n < \infty)$. Alors

(i) les arêtes des cellules sont des droites infinies si et seulement si P est colinéaire,

(ii) une arête $e(p_i, p_j)$ $(\neq \emptyset)$ est une demi-droite si et seulement si P est non-colinéaire, et p_i et p_j sont des points générateurs consécutifs de la frontière de l'enveloppe convexe de P,

(iii) une arête $e(p_i, p_j)$ $(\neq \emptyset)$ est un segment si et seulement si le segment de ligne $\overline{p_i p_j}$ n'est pas une arête de l'enveloppe convexe de P.

Propriété 4. Le point générateur le plus proche de p_i génère une arête de $V(p_i)$.

Propriété 5. Le point générateur le plus proche de p_i se trouve parmi les points générateurs dont les polygones ont une arête en commun avec $V(p_i)$.

[6] Puisque tout point de $\mathbb{R}^2$ est assigné à au moins un point générateur.

Les propriétés 4 et 5 concernent le point générateur le plus proche d'un autre point générateur. La propriété suivante concerne le point générateur le plus proche d'un point arbitraire.

Propriété 6. Le point générateur p_i est le point générateur le plus proche d'un point p quelconque si et seulement si $p \in V(p_i)$.

Propriété 7. Pour tout sommet q_i dans un diagramme de Voronoi, il existe un cercle unique C_i centré en q_i qui passe par trois points générateurs ou plus, et ne contient aucun point générateur dans son intérieur. Sous l'hypothèse de non-dégénérescence, C_i passe exactement par trois points générateurs.

On appellera *cercle vide*, un cercle qui ne contient aucun point dans son intérieur (ici, il s'agit des points générateurs).

Introduisons ici une deuxième hypothèse de travail.

Hypothèse 2 (Non-cocircularité)
Soit un ensemble de points $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2 \quad (4 \leq n < \infty)$. *Alors il n'existe pas un cercle* C *tel que* C *passe par les points* $p_{i1}, \ldots, p_{ik} \in P \quad (k \geq 4)$ *et tous les points dans* $P \setminus \{p_{i1}, \ldots, p_{ik}\}$ *sont en dehors de* C.

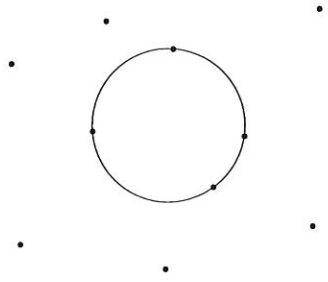


FIG. 1.9: *Un exemple de cocircularité*

À partir de la propriété 7, on peut montrer que l'hypothèse de non-dégénérescence est équivalente à celle de non-cocircularité.

Cette hypothèse peut être étendue à l'espace $\mathbb{R}^d$: on remplace un cercle par une hypersphère et le nombre de 4 points par $d+2$. On parlera alors d'*hypothèse de non-cosphéricité.*

Dans le cas où les points générateurs $P = \{p_1, \ldots, p_n\}$ proviennent d'un processus de Poisson homogène, la configuration de P satisfait à cette hypothèse de non-cosphéricité.

Propriété 8. Le cercle C_i dans la Propriété 7 est le cercle vide le plus large parmi tous les cercles vides centrés en q_i.

Propriété 9. Soient n, n_e et n_v le nombre de points générateurs, le nombre d'arêtes et le nombre de sommets dans un diagramme de Voronoi dans $\mathbb{R}^2$ resp. $(2 \leq n < \infty)$.

Alors $n_v - n_e + n = 1$.

Une extension de cette propriété est donnée dans la propriété suivante.

Propriété 10. Soit n_k le nombre de faces de Voronoi à k dimensions dans un diagramme d-dimensionnel. Alors

$$\sum_{k=0}^{d}(-1)^k n_k = (-1)^d. \tag{1.14}$$

Propriété 11. Le nombre moyen d'arêtes par cellule ne dépasse pas six.

Il y a encore beaucoup d'autres propriétés sur les diagrammes de Voronoi. On en trouvera davantage, ainsi que les justifications de celles énoncées ici, dans [11, section 2.3]. Attardons-nous cependant encore sur un cas particulier de diagrammes de Voronoi.

Les diagrammes de Voronoi de Poisson

On parle uniquement ici de tessellations de Voronoi générées à partir d'un ensemble P de points lui-même généré par un processus de Poisson homogène. D'où les termes *diagramme* et *cellule de Voronoi de Poisson* (« *Poisson Voronoi cells* »). Un diagramme de Voronoi de Poisson, généré par la réalisation du processus de Poisson homogène Θ_P, sera noté $\mathcal{V}_P$. Ces diagrammes possèdent quelques propriétés supplémentaires, héritées de celles de Θ_P.

Propriété P1. Les cellules du diagramme $\mathcal{V}_P$ sont des polyèdres convexes bornés. Cfr. la propriété 2 : $V(p_i)$ est non bornée ssi p_i est sur la frontière de l'enveloppe convexe de $P = \{p_1, \ldots, p_n\}$.

Propriété P2. $\mathcal{V}_P$ est stationnaire et isotrope.

Ces termes rapportés à un diagramme de Voronoi signifient que les caractéristiques de $\mathcal{V}_P$ et celles de ses cellules sont invariantes sous translation et rotation autour de l'origine de $\mathbb{R}^d$. ($\mathcal{V}_P$ est « motion-invariant ».)

Remarque. Nous ne détaillons pas ici comment on peut construire les cellules, l'ensemble de points donné. Il existe plusieurs algorithmes qui calculent ces cellules et en donnent une représentation graphique, en même temps que quelques caractéristiques. Le lecteur peut consulter à ce sujet [11, chapitre 4] et peut-être [2, p. 63–75].

1.2.3 Quelques propriétés des cellules de Voronoi de Poisson

Cette section est grandement inspirée de [11, chapitre 5]. Les auteurs précisent que la plupart des résultats de ce chapitre ne sont vrais que presque sûrement (c.-à-d. avec une probabilité égale à 1).

Afin d'éviter les cas pathologiques, on supposera que l'ensemble P est localement fini et ses points sont en position quadratique générale.

Définition 1.16 ($(d-k)$-face de la tessellation)
Une $(d-k)$-face de la tessellation de Voronoi est

$$F(p_0,\ldots,p_k) = \bigcap_{i=0}^{k} V(p_i) \quad (\neq \emptyset). \tag{1.15}$$

Une cellule est une d-face.

Soit l'hyperplan $G(p_0,\ldots,p_k) = \{\, \mathbf{x} \in \mathbb{R}^d \mid \|\mathbf{x}-\mathbf{x_0}\| = \ldots = \|\mathbf{x}-\mathbf{x_k}\| \,\}$. On a toujours $F(p_0,\ldots,p_k) \subset G(p_0,\ldots,p_k)$.

De plus, l'hypothèse de non-colinéarité est équivalente à l'assertion : $G(p_0,\ldots,p_k)$ est de dimension $(d-k)$ (pour $k=1,\ldots,d$). Et l'hypothèse de non-cosphéricité est équivalente à : $G(p_0,\ldots,p_k) = \emptyset$ si $k > d+1$. Enfin, la position quadratique générale et le caractère localement fini du processus impliquent que $F(p_0,\ldots,p_k)$ est soit vide soit de dimension $(d-k)$.

Venons-en aux moments et distributions des caractéristiques des cellules. Dès le second ordre, la dérivation analytique de ces moments devient très compliquée. Nous donnons dans les tableaux 1.1 et 1.2 quelques moments du premier et du deuxième ordre (les troisième et quatrième colonnes représentent les valeurs exactes et numériques des moments). Dans toute cette section, λ représente l'intensité du processus Θ_P. Précisons encore la notion de *cellule-type* : cela fait référence à une cellule ayant la même distribution qu'une cellule choisie aléatoirement dans $\mathcal{V}$, où le choix est effectué de telle manière que toutes les cellules ont la même probabilité d'être désignée.

moment	notation	v. exacte	v. num.
espérance du nombre de sommets (ou arêtes)	$E[N]$	6	
espérance du nombre de cellules voisines	$E[C]$	4	
espérance de l'aire	$E[A]$	$1/\lambda$	
et son second moment	$E[A^2]$		$1.280\,\lambda^{-2}$

TAB. 1.1: *Moments de premier et second ordre de quelques caractéristiques d'une cellule-type de Voronoi de Poisson dans* $\mathbb{R}^2$

moment	notation	v. exacte	v. num.
espérance du nombre de cellules voisines	$E[C]$	8	8
espérance du volume	$E[V]$	$1/\lambda$	$1/\lambda$
et son second moment	$E[V^2]$		$1.280\,\lambda^{-2}$

TAB. 1.2: *Moments de premier et second ordre de quelques caractéristiques d'une cellule-type de Voronoi de Poisson dans* $\mathbb{R}^3$

Okabe et al. [11, p. 342 et ss] font référence à une quarantaine de chercheurs ayant travaillé sur la distribution de l'aire (sur $\mathbb{R}^2$) et du volume (sur $\mathbb{R}^3$) des cellules de Voronoi de Poisson. Certains auteurs utilisent une fonction gamma généralisée à trois paramètres pour estimer la distribution de l'aire ou du volume des cellules :

$$f_1(x) = \frac{r b^{q/r} x^{q-1} e^{-bx^r}}{\Gamma(\frac{q}{r})} \quad r,\ b,\ q > 0. \tag{1.16}$$

Parmi les auteurs cités dans [11], Hinde et Miles ont estimé l'aire d'une cellule avec la fonction (1.16), en prenant comme paramètres $r = 1.0787$, $b = 3.0328\,\lambda^r$ et $q = 3.3095$ (figure 1.10). L'ajustement semble très bon visuellement, et pourtant il est rejeté par un test χ^2.

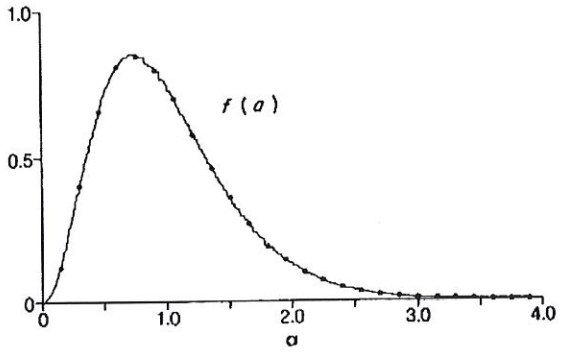


FIG. 1.10: *Histogramme normalisé de l'aire standardisée λA d'une cellule-type de Voronoi de Poisson dans $\mathbb{R}^2$, avec les valeurs de la fonction* (1.16) *d'après les paramètres estimés de Hinde et Miles*

Tanemura (voir [17]) utilise la fonction (1.16) pour estimer le volume de cellules en deux, trois et quatre dimensions. Dans le tableau suivant sont reprises ses valeurs estimées des paramètres. Pour cette étude, Tanemura a pris des échantillons de dix millions de cellules indépendantes en deux dimensions, et cinq millions en trois et quatre dimensions.[7] Les valeurs des paramètres en une dimension sont bien connues.

dimension	r	b	q
1	1 (exacte)	$2\,\lambda$ (exacte)	2 (exacte)
2	1.078 05	3.040 11 λ^r	3.314 98
3	1.163 91	4.063 42 λ^r	4.806 51
4	1.295 53	4.904 93 λ^r	6.497 07

TAB. 1.3: *Valeurs des paramètres proposées par Tanemura* ([17]) *pour la fonction $f_1(x)$, afin d'estimer la taille des cellules de Voronoi de Poisson en deux, trois et quatre dimensions*

[7] Tanemura précise dans son article qu'il y a pas d'autre étude à ce jour utilisant des échantillons de cette taille. Malheureusement, aucune date n'est indiquée dans son article. Tout au plus sait-on que sa référence bibliographique la plus récente date de 1992.

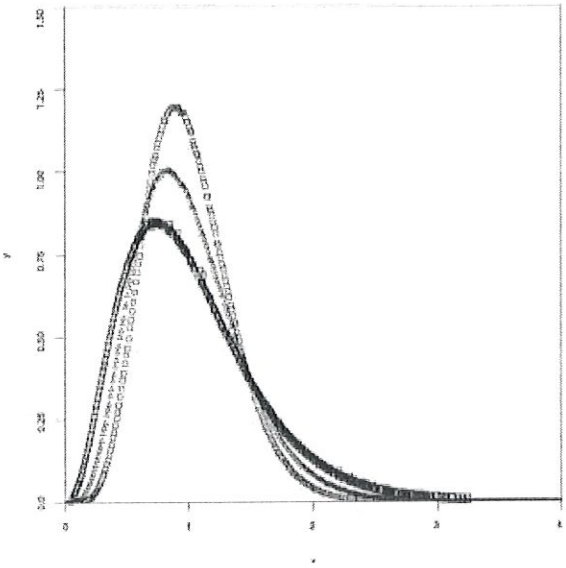


FIG. 1.11: *Histogrammes observés du volume réduit λV d'une cellule-type de Poisson de Voronoi, en deux, trois et quatre dimensions :* • *: 2-D ;* ∇ *: 3-D ;* □ *: 4-D. Espérance du volume :* 1.000 17, 0.999 67 *et* 0.999 91 *en deux, trois et quatre dimensions resp.*

Tanemura ([17]) note également, d'après l'observation des valeurs du tableau 1.3, qu'il peut exister une relation entre les paramètres et la dimension de l'espace ; la tendance serait dissymétrique.

Enfin, Okabe et al. [11, p. 345] citent aussi Tanemura, dans un autre article, où il a trouvé qu'une distribution gamma à trois paramètres donnait un meilleur ajustement du volume qu'une gamma à deux paramètres.

Plusieurs scientifiques (notamment Vaz et Fortes) estiment que bien que les distributions gamma et lognormale soient très proches, la description de l'aire ou du volume d'une cellule par une distribution lognormale n'est pas aussi adéquate qu'avec une distribution gamma. Beaucoup de chercheurs ont proposé une distribution gamma, mais avec deux paramètres : si dans l'équation (1.16) nous posons $r = 1$, nous retrouvons la densité de probabilité d'une variable $\Gamma(b, q)$:

$$f_2(x) = \frac{b^q x^{q-1} e^{-bx}}{\Gamma(q)} \quad b,\ q > 0. \tag{1.17}$$

Quelques valeurs de paramètres pour cette fonction 1.17 sont données dans le tableau suivant (la dernière colonne représente le nombre de cellules considéré pour chaque étude).

dim.	b	q	chercheur(s)	nb. cellules
2	4λ	4	Kiang (1966)	-
	3.61λ	3.61	Weaire, Kermode et Wejchert (1986)	-
			Marthinsen (1996)	100 000
	3.57λ	3.61	Di Cenzo et Wertheim (1989)	$\approx$ 193 500
	3.571λ	3.571	Vaz et Fortes (1988)	-
	3.37λ	3.37	Zaninetti (1992)	-
	3.718λ	3.718	Kumar et Kurtz (1993)	2 000 000
3	6λ	6	Kiang (1966)	12 800
	5.56λ	5.56	Andrade et Fortes (1988)	-
			Vaz et Fortes (1988)	-
			Thorvaldsen (1992)	250 000
			Marthinsen (1996)	$\approx$ 25 000
	5.38λ	5.38	Zaninetti (1992)	-
	5.7869λ	5.7869	Kumar, Kurtz, Banavar, Sharma (1992)	358 000

TAB. 1.4: *Quelques valeurs de paramètres proposées pour la distribution (1.17), afin d'estimer le volume d'une cellule-type de Voronoi de Poisson en deux et trois dimensions*

Notons encore que tous ces ajustements dans le tableau 1.4 sont rejetés par un test de Kolmogorov-Smirnov avec $\alpha = 0.01$. De plus, toutes ces estimations satisfont la condition $E[A] = q/b = \lambda^{-1}$ (ou $E[V] = q/b = \lambda^{-1}$), sauf celles de DiCenzo et Wertheim.

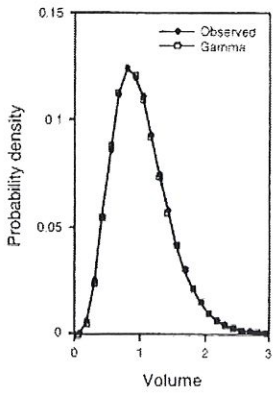


FIG. 1.12: *Histogramme du volume normalisé d'une cellule-type de Voronoi de Poisson dans* $\mathbb{R}^3$ *, avec des points* $\square$ *correspondant à la densité d'une gamma ayant les paramètres estimés de Kumar et al. (1992)*

Chapitre 2

Test d'uniformité en une dimension sur base de cellules de Voronoi (partie théorique)

2.1 Introduction

Soit $A \subset \mathbb{R}$. Nous considérerons ici que A représente toujours un intervalle fermé : $A = [a, b]$. L'hypothèse de départ est que l'échantillon $\{ x_1, \ldots, x_n \}$ est généré par un processus de Poisson (PP) homogène. Par caractérisation du PP homogène, les $N(A) \overset{\text{not}}{=} n$ réalisations sont indépendantes et forment un échantillon aléatoire d'une loi uniforme sur A, conditionnellement à $N(A) = n$.

La démarche utilisée ici est de trouver en quoi l'hypothèse de départ peut influencer les caractéristiques des cellules de Voronoi (construites à partir de l'échantillon réalisé), et ce dans le but d'établir un test d'uniformité en une dimension.

Notons encore que ce n'est pas la position de l'intervalle $A = [a, b]$ qui caractérise le PP homogène, mais seulement sa longueur $m(A) \overset{\text{not}}{=} l(A)$ (propriété 4 du PP homogène). Autrement dit, pour la réalisation de l'échantillon de points, prendre un intervalle $[-2; 10]$ ou un intervalle $[3; 15]$ est tout-à-fait équivalent. Ainsi, on peut se limiter à travailler sur un sous-ensemble $A = [0, b]$ $(l(A) = b)$. De même, tous les résultats seront proportionnels à la longueur, on parle ici d'*équivariance*. Donc, dans la pratique, nous travaillerons sur $[0, 1]$ sans perte de généralité.

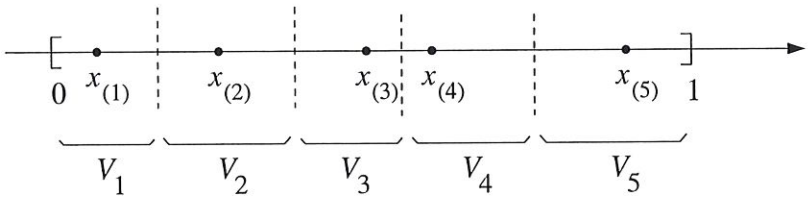


FIG. 2.1: *Exemple de diagramme de Voronoi dans* $A = [0, 1]$

Remarque. Nous considérons maintenant, comme sur la figure 2.1, que V_i représente la cellule du $(i)^{\text{e}}$ point de l'échantillon : $V_i = V(x_{(i)})$, où $\{ x_{(1)}, x_{(2)}, \ldots, x_{(n)} \}$

est l'échantillon de départ ordonné. Ainsi, $x_{(1)}$ est la plus petite valeur de l'échantillon, $x_{(n)}$ la plus grande, $x_{(r)}$ la r^{e} plus petite valeur de l'échantillon ; $x_{(r)}$ est aussi la statistique du r^{e} ordre de l'échantillon. La médiane et les quantiles sont aussi des statistiques d'ordre : la médiane est égale à $x_{(\frac{n}{2})}$ ou $x_{(\frac{n+1}{2})}$.

2.2 Quelques résultats sur les statistiques d'ordre

2.2.1 Préliminaire : la variable Bêta

Nous commencerons tout d'abord par rappeler la définition de la variable aléatoire Bêta, que nous utiliserons par la suite.

Rappelons avant tout la définition de la fonction $\Gamma(\cdot)$. La *fonction* $\Gamma(\cdot)$ est définie par

$$\Gamma : \mathbb{R}^+ \to \mathbb{R}^+ : \alpha \mapsto \Gamma(\alpha) = \int_0^\infty x^{\alpha-1} e^{-x}\, dx. \tag{2.1}$$

La notation $\Gamma(\cdot,\cdot)$ fait référence quant à elle à la *distribution* statistique Gamma à deux paramètres. La condition $\alpha > 0$ assure la convergence de l'intégrale. On voit de suite que $\Gamma(1) = 1$ et $\Gamma(2) = 1$. En fait, on peut mettre en évidence quelques propriétés de la fonction $\Gamma(\cdot)$:

(i) $\forall \alpha > 0 \quad \Gamma(\alpha+1) = \alpha\Gamma(\alpha)$

(ii) Si α est un entier (> 0), alors $\Gamma(\alpha+1) = \alpha!$

(iii) On a la valeur particulière $\Gamma(\frac{1}{2}) = \sqrt{\pi}$.

On peut également définir la fonction Gamma pour $\alpha < 0$, en se servant de la première propriété : $\Gamma(n) = \frac{\Gamma(n+1)}{n}$.

Voici d'autres définitions de la fonction Gamma :

$$\Gamma(x+1) = \lim_{k\to\infty} \frac{1\cdot 2\cdot 3\cdots k}{(x+1)(x+2)\cdots(x+k)} k^x \tag{2.2}$$

$$\frac{1}{\Gamma(x)} = x e^{\gamma x} \prod_{m=1}^{\infty} \left\{ \left(1 + \frac{x}{m}\right) e^{-x/m} \right\}, \tag{2.3}$$

où γ est la constante d'Euler :

$$\gamma = \lim_{n\to\infty} \left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} - \ln n\right) = 0.577\,215\,665. \tag{2.4}$$

L'équation (2.3) représente le développement de la fonction Gamma en produit infini.

Le lecteur trouvera encore d'autres formules sur la fonction $\Gamma(\cdot)$ dans [16, p. 101–102].

La fonction $B(\alpha,\beta)$ est définie par

$$B(\alpha,\beta) \stackrel{\text{def}}{=} \int_0^1 x^{\alpha-1}(1-x)^{\beta-1}\, dx, \tag{2.5}$$

où l'intégrale converge pour $\alpha > 0$ et $\beta > 0$. On peut montrer que

$$B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}. \tag{2.6}$$

Remarquons que $B(\alpha,\beta) = B(\beta,\alpha)$ (fonction symétrique). Cette fonction n'est pas à confondre avec la loi Bêta classique (on parle parfois de loi Bêta de première espèce).

Définition 2.1 (distribution Bêta)
Une variable aléatoire X est dite une variable aléatoire Bêta de paramètres α et β (notée $X \sim \mathcal{B}(\alpha, \beta)$) si sa fonction de fréquence est

$$F_X(x) = \begin{cases} \dfrac{1}{B(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1} & 0 \leq x \leq 1 \quad (\alpha > 0,\ \beta > 0) \\ 0 & x < 0 \end{cases} \tag{2.7}$$

Le domaine de la distribution Bêta est donc $[0, 1]$ et il est facile de vérifier que $\int_0^1 f_X(x)\, dx = 1$. On a également ces résultats-ci :

$$E[X] = \frac{\alpha}{\alpha+\beta}, \tag{2.8}$$

$$var(X) = \frac{\alpha\beta}{(\alpha+\beta+1)(\alpha+\beta)^2}, \tag{2.9}$$

$$\psi_X(t) = \frac{1}{B(\alpha,\beta)} \int_0^1 x^{\alpha-1}(1-x)^{\beta-1} e^{-2\pi i x t}\, dx, \tag{2.10}$$

où $\psi_X(t) \stackrel{\text{def}}{=} E[e^{itX}]$ est la fonction caractéristique.

2.2.2 Quelques résultats sur les statistiques d'ordre

Les développements suivants proviennent de [7, p. 251, 325 et ss].

De manière générale, considérons un échantillon de départ $\{x_1, \ldots, x_n\}$, de taille n, provenant d'une fonction de distribution $F(x)$, avec une fonction de fréquence $f(x)$ continue. Bien sûr, les x_i sont indépendants. La distribution des valeurs est donnée par

$$dG = dF(x_1)dF(x_2)\ldots dF(x_n). \tag{2.11}$$

Si on réarrange les valeurs dans l'ordre : $\{x_{(1)}, x_{(2)}, \ldots, x_{(n)}\}$, ces valeurs $x_{(i)}$ ne sont plus indépendantes ni identiquement distribuées comme les x_i de départ l'étaient.

La distribution d'échantillonnage de $x_{(i)}$ est donnée par G_i, où

$$dG_i = \frac{\left[F(x_{(i)})\right]^{i-1}\left[1-F(x_{(i)})\right]^{n-i}}{B(i, n-i+1)}\, dF(x_{(i)}) \tag{2.12}$$

est une distribution Bêta.

La distribution conjointe de $x_{(i)}$ et $x_{(j)}$ $(i < j)$ est donnée par

$$dG_{i,j} = \frac{\left[F(x_{(i)})\right]^{i-1}\left[F(x_{(j)})-F(x_{(i)})\right]^{j-i-1}\left[1-F(x_{(j)})\right]^{n-j}}{B(i, j-i)B(j, n-j+1)}\, dF(x_{(i)})\, dF(x_{(j)}). \tag{2.13}$$

$x_{(i)}$ et $x_{(j)}$ sont toujours corrélés positivement.

La généralisation à la distribution conjointe de plusieurs statistiques d'ordre est immédiate. Par exemple, pour $x_{(i)}$, $x_{(j)}$, $x_{(k)}$ et $x_{(l)}$ $(i < j < k < l)$, nous obtenons, en écrivant F_i à la place de $F(x_{(i)})$:

$$dG_{i,j,k,l} = \frac{F_i^{i-1}\,(F_j-F_i)^{j-i-1}\,(F_k-F_j)^{k-j-1}\,(F_l-F_k)^{l-k-1}\,(1-F_l)^{n-l}}{B(i, j-i)B(j, k-j)B(k, l-k)B(l, n-l+1)} dF_i\, dF_j\, dF_k\, dF_l. \tag{2.14}$$

Notons que dans le cas où l'échantillon $\{ x_1, \ldots, x_n \}$ est issu d'un PP homogène sur $[0,1]$, $X_1, \ldots, X_n$ sont VAIID de loi $U_{[0,1]}$. Et donc, on a que $F(x) = x (0 \leq x \leq 1)$: dans les équations précédentes, $F(x_{(i)})$ peut être remplacé par $x_{(i)}$, ce qui conduit à des résultats particulièrement simples. Ainsi, on trouve que

$$X_{(i)} \sim \mathcal{B}(i-1, n-i+1) \quad 1 \leq i \leq n. \tag{2.15}$$

On peut montrer également que la distribution conjointe de $y = F(x_{(i)}) - F(x_{(j)})$ et $z = F(x_{(i)})$ est

$$dG_{y,z} = \frac{z^{i-1} y^{j-i-1} (1-y-z)^{n-j}}{B(i, j-i) B(j, n-j+1)} \, dy \, dz \quad 0 \leq y+z \leq 1. \tag{2.16}$$

On trouve donc que y est distribuée selon une loi $\mathcal{B}(j-i, n-j+i+1)$.

Dans le cas d'un échantillon uniforme sur $[0,1]$, on a donc :

$$Y = X_{(j)} - X_{(i)} \sim \mathcal{B}(j-i, n-j+i+1). \tag{2.17}$$

Un cas particulier est donné par

$$Y = X_{(i)} - X_{(i-1)} \sim \mathcal{B}(1, n), \tag{2.18}$$

soit $f_Y(y) = n(1-y)^{n-1}$.

Or, nous trouvons dans [12, p. VI.13] le résultat suivant :

Soient $X_1, \ldots, X_n$ VAIID *de loi* $U_{[0,1]}$, *et* $Y_1 \triangleq \min\limits_{1 \leq i \leq n} X_i$ *et* $Y_2 \triangleq \max\limits_{1 \leq i \leq n} X_i$. *Alors*

$$f_{Y_1,Y_2}(y_1, y_2) = \begin{cases} n(n-1)(y_2 - y_1)^{n-2} & \textit{si } 0 \leq y_1 \leq y_2 \leq 1 \\ 0 & \textit{sinon} \end{cases} \tag{2.19}$$

et

$$f_{Y_2 - Y_1}(t) = \begin{cases} n(n-1)(1-t)t^{n-2} & \textit{si } 0 \leq t \leq 1 \\ 0 & \textit{sinon} \end{cases}. \tag{2.20}$$

Dans le cas où $n = 3$, nous avons $Y \triangleq Y_2 - Y_1 = X_{(n)} - X_{(1)} \sim \mathcal{B}(2, n-1)$ en utilisant l'équation (2.17), ce qui nous donne $f_Y(y) = 6y(1-y)$. On obtient la même réponse en utilisant le résultat (2.20).

Dans le cas où $n = 2$, nous avons $Y \triangleq X_{(2)} - X_{(1)} \sim \mathcal{B}(1, n)$, et donc $f_Y(y) = 2(1-y)$. En utilisant la fonction (2.20), on arrive à la même solution.

2.3 Caractéristiques des cellules 1-D

Essayons de trouver quelques caractéristiques des cellules de Voronoi en une dimension. Les cellules consistant en de simples segments de droite, il n'y a pas beaucoup de caractéristiques les décrivant.

Spontanément, on pensera tout d'abord à la *longueur* des cellules.

On sait aussi que le point i de la cellule V_i n'est pas forcément « au milieu » de cette cellule. Notons $l(V_i) = l_i = l_{i,1} + l_{i,2}$ (cf. figure 2.2). Dès lors, on peut penser utiliser la grandeur $\frac{l_{i,1}}{l_i}$ pour chaque cellule. (Bien sûr, prendre $\frac{l_{i,2}}{l_i}$ est équivalent.).

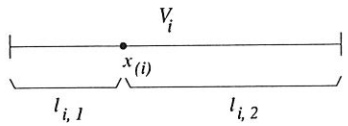


FIG. 2.2: *Définition de deux longueurs supplémentaires pour la cellule V_i*

Éventuellement, la grandeur $\frac{l_{i,1}}{l_{i,2}}$ peut être intéressante à analyser aussi. Ce qui nous donne d'emblée trois caractéristiques sur lesquelles travailler.

2.3.1 Étude à partir de la longueur des cellules

2.3.2 Densité de la longueur des cellules en terme de Bêta

À partir de l'équation (2.17), on peut encore dégager le résultat suivant :

$$T = X_{(i+1)} - X_{(i-1)} \sim \mathcal{B}(2, n-1), \tag{2.21}$$

soit

$$f_T(t) = \begin{cases} n(n-1)t(1-t)^{n-2} & 0 \leq t \leq 1 \\ 0 & \text{ailleurs} \end{cases}. \tag{2.22}$$

Cela correspond de manière surprenante à la fonction (2.20). Or, l, la longueur d'une cellule de Voronoi, est égale à $\frac{X_{(i+1)} - X_{(i-1)}}{2} = \frac{T}{2}$. D'où

$$2L \sim \mathcal{B}(2, n-1). \tag{2.23}$$

2.3.3 Densité de la longueur des cellules

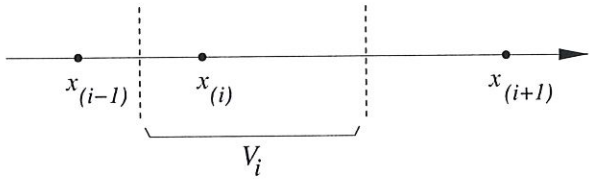


FIG. 2.3: *Calcul de la longueur de la cellule V_i, à partir des points générateurs l'entourant*

La longueur de la cellule V_i à la figure 2.3 est

$$l_i = \frac{x_{(i)} + x_{(i+1)}}{2} - \frac{x_{(i-1)} + x_{(i)}}{2} = \frac{x_{(i+1)} - x_{(i-1)}}{2}. \tag{2.24}$$

C'est la formule générale pour toutes les cellules d'un diagramme comme à la figure 2.1, sauf pour la première et la dernière cellule :

$$l_1 = \frac{x_{(1)} + x_{(2)}}{2} \tag{2.25}$$

$$l_n = 1 - \frac{x_{(n-1)} + x_{(n)}}{2}. \tag{2.26}$$

Soient donc les n points $x_1, \ldots, x_n$ de l'échantillon généré par un PP homogène et soient $X_1, \ldots, X_n$ VAIID de loi $U_{[0,1]}$. D'emblée nous voyons que $x_{(1)}$ est la valeur donnée par une variable aléatoire $Y_1 = \min_{1 \leq i \leq n} X_i$ et $x_{(n)}$ la valeur donnée par une variable $Y_2 = \max_{1 \leq i \leq n} X_i$.[1] On trouve alors, par calculs :

$$f_{Y_1}(x) = \begin{cases} n\,(1-x)^{n-1} & \text{si } 0 \leq x \leq 1 \\ 0 & \text{sinon} \end{cases}$$

et

$$f_{Y_2}(x) = \begin{cases} n\,(x)^{n-1} & \text{si } 0 \leq x \leq 1 \\ 0 & \text{sinon} \end{cases}$$

ainsi que

$$f_{Y_1,Y_2}(y_1, y_2) = \begin{cases} n(n-1)(y_2 - y_1)^{n-2} & \text{si } 0 \leq y_1 \leq y_2 \leq 1 \\ 0 & \text{sinon} \end{cases}$$

et

$$f_{Y_2 - Y_1}(t) = \begin{cases} n(n-1)(1-t)t^{n-2} & \text{si } 0 \leq t \leq 1 \\ 0 & \text{sinon} \end{cases}.$$

On sait (voir 2.22) que la VA longueur $L_i = \dfrac{X_{(i+1)} - X_{(i-1)}}{2}$, notée $2L \sim \mathcal{B}(2, n-1)$, a pour densité

$$f_{2L}(l) = \begin{cases} n(n-1)4l(1-2l)^{n-2} & \text{si } 0 \leq l \leq \frac{1}{2} \\ 0 & \text{sinon} \end{cases}.$$

On voit que cette densité ne dépend pas de i (on suppose ici $i \neq 1$ et $i \neq n$). Pour $n = 3$, nous trouvons

$$f_{2L}(l) = \begin{cases} 24l(1-2l) & \text{si } 0 \leq l \leq \frac{1}{2} \\ 0 & \text{sinon} \end{cases}.$$

[1] Puisque $x_{(1)}$ est le point minimum de l'échantillon et $x_{(n)}$ le point maximum.

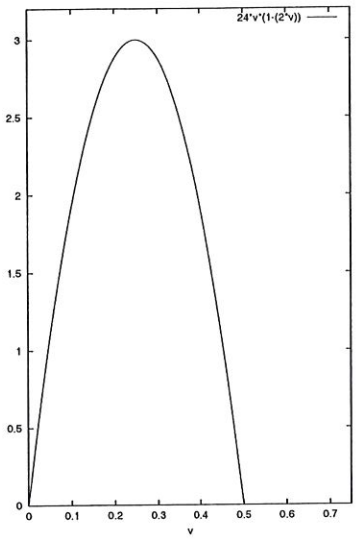


FIG. 2.4: *Graphe de la fonction* $f_L(l)$

On a donc que

$$E[l] = \frac{1}{4}$$
$$var(l) = \frac{21}{80} = 0.2625.$$

De façon générale, pour n points, on a $E[L] = \frac{1}{2}E[T]$, où $T = X_{(i+1)} - X_{(i-1)}$ $(i \neq 1, i \neq n)$. Or $T \sim \mathcal{B}(2, n-1)$, d'où $E[T] = 2/(2+n-1) = 2/(n+1)$. Et donc

$$E[L] = \frac{1}{n+1}.$$

De même, $var(L) = var(T)/4$; or

$$var(T) = \frac{2(n-1)}{(n+1)^2(n+2)},$$

d'où

$$var(L) = \frac{n-1}{2(n+1)^2(n+2)}.$$

Ce qui est logique ! En effet, il faut envisager que la génération de n points uniformément sur $[0,1]$ équivaut à la génération de n points sur un cercle de circonférence égale à 1, mais auxquels il faut ajouter un point correspondant à l'origine qui est également aléatoire. D'où l'espérance de la longueur d'une cellule de Voronoi est de $\frac{1}{n+1}$ pour n points générés sur $[0,1]$ et de $\frac{1}{n}$ pour n points générés sur le cercle de circonférence unité, car alors on prend comme origine $X_{(1)}$.

2.3.4 Densité de la longueur des cellules en terme de Gamma

Dans cette section, nous démontrons que la longueur des cellules suit une loi $\Gamma(2\lambda, 2)$, résultat déjà mis en évidence dans le tableau 1.3, p. 16, des paramètres trouvés numériquement par Tanemura, cf. [17]. Dans le cas de la dimension 1, l'auteur a juste retrouvé numériquement ces valeurs, qui étaient déjà bien connues par

la théorie.

Soit un processus de Poisson homogène Θ_P unidimensionnel sur $B = [0, +\infty[$, de densité λ, donnant lieu à n réalisations. Soit X, une VA représentant la longueur de l'intervalle entre deux points successifs[2]. On peut montrer facilement que la fonction de répartition de X est

$$F_X(x) = P[X \leq x] = 1 - e^{-\lambda x}. \tag{2.27}$$

Et donc, sa fonction de densité est donnée par

$$f(x) = \lambda e^{-\lambda x} \qquad 0 < x < \infty, \tag{2.28}$$

ce qui correspond à une loi $Expo(\lambda)$, ou $\Gamma(\lambda, 1)$.

Prouvons cela. Il s'agit en fait d'un problème classique de processus stochastiques. Ce problème peut être considéré selon un autre point de vue : soit un PP homogène unidimensionnel de densité λ, soient τ_r la durée jusqu'à la r^{e} arrivée (c.-à-d. génération d'un nombre aléatoire) et $n(t)$ le nombre d'arrivées au temps t.
On a que $n(t) \sim Po(\lambda t)$ par hypothèse de PP. De plus, les deux évènements $\{\, n(t) \geq r \,\}$ et $\{\, \tau_r \leq t \,\}$ sont équivalents. Dès lors,

$$F_r(t) = P[\tau_r \leq t] = P[n(t) \geq r] = \sum_{i=r}^{\infty} P[n(t) = i] = \sum_{i=r}^{\infty} \frac{e^{-\lambda t}(\lambda t)^i}{i!}.$$

Ainsi,

$$f_r(t) = \frac{dF_r}{dt}(t) = e^{-\lambda t} \cdot \left\{ -\lambda \sum_{i=r}^{\infty} \frac{(\lambda t)^i}{i!} + \lambda \sum_{i=r}^{\infty} \frac{(\lambda t)^{i-1}}{(i-1)!} \right\}.$$

Si $r = 1$, nous avons

$$f_1(t) = e^{-\lambda t} \left\{ -\lambda(e^{\lambda t} - 1) + \lambda e^{\lambda t} \right\} = \lambda e^{-\lambda t},$$

ce qui est bien la fonction de densité d'une VA $Expo(\lambda)$.
Ceci correspond effectivement à notre problème. Les arrivées successives concordent avec les $x_{(i)}$, croissants ; le cas $r = 1$ signifie qu'on s'intéresse à la durée entre deux arrivées successives[3].
Ajoutons maintenant une remarque. Puisque $X \sim Expo(\lambda)$, on trouve que $E[X] = 1/\lambda$. Or, on a montré que $E[X] = 1/(n+1)$, où $(n+1)$ est le nombre total d'intervalles.
Donc, ici, $\lambda = n + 1$, ainsi $E[X] = 1/(n+1)$ si $X \sim Expo(\lambda)$. Notons que $\lambda = n$ si on travaille sur un cercle, plutôt que sur un segment de droite, comme on l'a dit plus haut.

Considérons maintenant deux intervalles contigus l'un à l'autre, et soient X_1 et X_2 leurs longueurs. Alors, $Y = \frac{1}{2}(X_1 + X_2)$ est une VA représentant la longueur d'une cellule de Voronoi. Notre but ici est bien de déterminer la fonction $f_Y(y)$ de la longueur d'une cellule de Voronoi dans le cas d'un PP homogène.

Pour le processus de Poisson homogène, nous avons que X_1 et X_2 sont indépendants. Nous trouvons donc que

$$f_{X_1,X_2}(x_1, x_2) = f_{X_1}(x_1) \cdot f_{X_2}(x_2) = \lambda^2 e^{-\lambda(x_1+x_2)}, \tag{2.29}$$

[2] Au sens de leurs coordonnées, pas au sens de la génération des points.

[3] Il n'y a pas d'effets mémoire dans le processus de Poisson. Ainsi, la fonction de densité entre la première et la deuxième arrivée est la même que celle entre la r^{e} et la $(r+1)^{\text{e}}$ arrivée.

où $0 < x_1 < \infty$ et $0 < x_2 < \infty$. Par un changement de variable classique, on trouve que

$$f_{X_1,Y}(x_1, y) = 2\lambda^2 e^{-2\lambda y},$$

où $0 < x_1 < 2y$ et $0 < y < \infty$. Donc, on obtient $f_Y(y)$ par calcul :

$$f_Y(y) = \int_0^{2y} f_{X_1,Y}(x_1, y)\, dx_1 = 4y\lambda^2 e^{-2\lambda y}.$$

Ainsi, on a bien $Y \sim \Gamma(2\lambda, 2)$.

Dans le cas particulier où $\lambda = 1$, on a que $f_Y(y) = 4ye^{-2y}$, soit $Y \sim \Gamma(2, 2)$.

Remarquons que $\lambda = \frac{1}{n+1}$ (ou $\frac{1}{n}$) veut dire qu'on attend $n+1$ points (ou n points) sur un segment de taille unité. Le PP homogène considère la génération de points sur $[0, +\infty[$. Mais, vu la décroissance exponentielle de la longueur des intervalles entre points, ou de la longueur des cellules de Voronoi, on peut approximer, pour λ petit et sur $[0, 1]$, les modèles Gamma et les modèles Bêta.

En effet, en posant $\lambda = \frac{1}{n}$, on a, pour $n \to \infty$ et $y = l \to 0$ (où l est la longueur d'une cellule de Voronoi) :

$$Y \sim \Gamma(2\lambda, 2) \Rightarrow f_Y(y) = 4y\lambda^2 e^{-2\lambda} \tag{2.30}$$

$$\text{et } 2L \sim \mathcal{B}(2, n-1) \Rightarrow f_{2L}(l) = n(n-1)4l(1-2l)^{n-2}. \tag{2.31}$$

En posant $Y = L$ et $y = l$, et en se souvenant que $\lambda = \frac{m(B)}{n} = \frac{1}{n}$, on a

$$\lim_{n\to\infty\,;\, l\to 0} f_{2L}(l) = \lim_{n\to\infty\,;\, l\to 0} \frac{1}{n^2} 4l(1-2l)^{n-2} \tag{2.32}$$

$$= \lim_{n\to\infty\,;\, l\to 0} 4\lambda^2 l e^{-2l/n} \tag{2.33}$$

$$= 4y\lambda^2 e^{-2\lambda y}. \tag{2.34}$$

Chapitre 3

Problèmes liés à la méthode utilisée dans ce travail

3.1 Introduction

Dans ce chapitre, comme dans tous les chapitres suivants, il est bien évidemment question de diagrammes de Voronoi *de Poisson* (nous générons des cellules de Voronoi à partir d'un échantillon de points *uniforme*). Pour ces diagrammes il y a essentiellement deux approches de travail possibles :

- on génère un grand nombre de points dans une région bornée B, selon un processus de Poisson homogène Θ_P ; on construit le diagramme $\mathcal{V}_P$ et on mesure les caractéristiques de toutes ses cellules ;
- on génère une séquence de cellules-types[1] de Voronoi de Poisson indépendantes, on mesure les caractéristiques de chacune d'elle, et finalement on les rassemble pour obtenir les distributions recherchées.

La première approche pose deux problèmes :

- le problème des effets de bord dus aux cellules proches de la frontière de B,
- le fait que les caractéristiques de cellules avoisinantes ne sont pas indépendantes.

La deuxième approche, appelée approche *séquentielle*, a comme avantage principal sur la première d'éviter justement les effets de bord.

3.2 L'approche séquentielle

Il y a deux variantes de cette approche. La première consiste à lancer tout d'abord n fois le PP homogène Θ_P, de densité λ, dans une région bornée B.

Puis, on construit pour chaque PP le diagramme $\mathcal{V}_P$, on identifie la cellule centrale[2] et on mesure les caractéristiques de cette cellule-là seulement. Cela implique de réfléchir sur le choix des valeurs de deux paramètres : λ et n Le choix de n est déterminé avant tout par le degré de précision à un niveau de signification demandé pour les estimations.

[1] La définition de ce terme est donnée à la p. 15.

[2] Il suffit de trouver le point générateur le plus proche du centre de B.

Le choix de λ résulte d'un compromis entre deux soucis : si λ est trop petit, la cellule centrale peut « toucher » la frontière de B ou être sujette à des effets de bord ; si λ est trop grand, la procédure générera beaucoup de points qui n'auront aucune influence sur la forme de la cellule centrale. Il y a beaucoup de variations dans les travaux des chercheurs à propos de la valeur la plus appropriée pour λ, même lorsque $B = [0,1]^d$. Par exemple, pour $B \subset \mathbb{R}^2$, Crain (1978) estime que $\lambda = 35$ est suffisant, tandis que Hinde et Miles (1980) utilisent $\lambda = 100$ (cf. [11, p. 308]).

La deuxième variante de l'approche séquentielle utilise la distribution conjointe des distances ordonnées à partir de l'origine dans Θ_P. En deux dimensions, le processus de simulation commence en assignant le premier point, p_0 à l'origine o. Les autres points p_i $(i = 1, 2, \ldots)$ sont ensuite générés aléatoirement tel que leur distance soit toujours croissante par rapport à o. Pour chaque point p_i généré, on dessine la médiatrice entre ces points $b(p_0, p_i)$ (cf. définition à l'équation (1.8)). Après un certain nombre de points p_i ajoutés, p_0 sera entouré par un polygone convexe, P_0. On continue cependant de générer des points, jusqu'à ce que ceux-ci cessent d'influencer la forme de P_0 : cela se produit lorsque la distance entre p_i et l'origine est supérieure au diamètre du plus petit cercle centré en p_0 et qui contient P_0.[3] On mesure alors les caractéristisques de P_0 on recommence toute la procédure pour le polygone suivant. Contrairement à la première variante, cette procédure-ci est essentiellement invariante par rapport à l'échelle utilisée (« *scale invariant* ») : il ne faut plus se soucier de savoir si la cellule est trop grande (effets de bord possibles) ou le nombre de points trop élevé (présence de beaucoup de points inutiles dans la détermination de la forme de la cellule). Bref on n'a plus à se soucier de choisir une valeur pertinente pour le paramètre λ. Cette variante-ci est également moins couteuse que la première, vu que 15 à 20 points suffisent généralement pour produire le polygone P_0.

Tanemura ([17]) par exemple, utilise la deuxième approche, première variante. Il choisit de prendre $\lambda = 200$ en 2-D, $\lambda = 500$ en 3-D et $\lambda = 1\,200$ en 4-D. Puis, il construit la cellule de Voronoi centrale à B au moyen d'un algorithme approprié.[4] Ainsi, quand il affirme utiliser cinq ou dix millions de cellules, il a généré en fait cinq ou dix millions de PP homogènes, c-.à-d. des échantillons de points générateurs, pour lesquels il construit juste la cellule centrale. Il obtient bien ainsi des cellules-types indépendantes.

3.3 L'approche non séquentielle

Le présent travail se situe dans le cadre de la première approche : nous considérons un seul échantillon de points, et nous essayons de vérifier l'uniformité ou non de ces points sur base des caractéristiques des cellules de Voronoi relatives à ces points. Une approche séquentielle ne permet pas de tester l'uniformité d'un nuage de points donnés. Nous devons alors considérer les problèmes inhérents à cette approche.

[3] Il s'agit bien du *diamètre* de ce cercle et non pas de son *rayon*, car chaque segment de P_0 est la médiatrice entre p_0 et un des p_i : un point sur ce segment est à égale distance de p_0 et de ce p_i.

[4] Il me semble évident que Tanemura ne construit pas *tout* le diagramme de Voronoi, mais bien que son algorithme lui permet de ne déterminer que la cellule centrale.

3.3.1 Le problème des effets de bord

Première solution : la suppression des cellules-frontière

Mais quelles sont les cellules-frontière ? Okabe et al. [11, p. 307] proposent d'enlever toutes les cellules de $\mathcal{V}_P$ pour lesquelles un cercle, centré à n'importe quel sommet ν de la cellule et passant par les trois points de Θ_P qui sont équidistants de ν (cf. propriété 7, section 1.2.2), a une intersection non nulle avec la frontière de B. Ainsi, sur la figure 3.1, la cellule A sera enlevée.

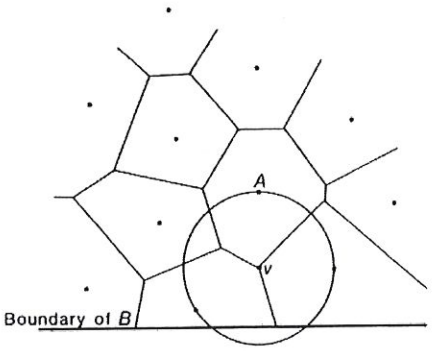


FIG. 3.1: *Une procédure pour identifier les cellules sujettes à des effets de bord*

Ne pas tenir compte des cellules-frontière est la solution utilisée par Vandooren ([18]), excepté qu'elle n'identifie pas les cellules-frontière de la même façon. Son programme place des points-grille sur $B = [0, 1]^3$, et notamment sur la frontière ∂B. Pour chaque point-grille, le programme repère à quelle cellule il appartient, et vérifie si ce point-grille est sur la frontière ∂B. Une cellule est une cellule-frontière si elle contient un ou des points-grille sur la frontière. En pratique, Vandooren ([18, p. 27–28]) constate que le nombre de cellules-frontière augmente fortement avec la dimension de l'espace (cf. tableau 3.1). On en conclut rapidement que toute étude au-delà de la dimension quatre n'a aucun intérêt avec cette méthode.

dimension	**nombre de cellules non-frontière sur 1 000 cellules**
2	893
3	533
4	201
5	30

TAB. 3.1: *Nombre de cellules non-frontière d'après le programme utilisé par Vandooren (voir* [18, p. 28]*)*

De plus, Vandooren, par son critère de sélection de cellules-frontière, n'aurait pas éliminé la cellule A dans la figure 3.1, contrairement à Okabe et al.

3.3.2 Deuxième solution : l'espace torique

Il s'agit d'une solution classique : les effets de bord disparaissent car l'espace (torique) est clos sur lui-même. Commençons par considérer cette solution en une dimension (nous l'utiliserons pour les tests en une dimension).

L'espace torique pour des données unidimensionnelles

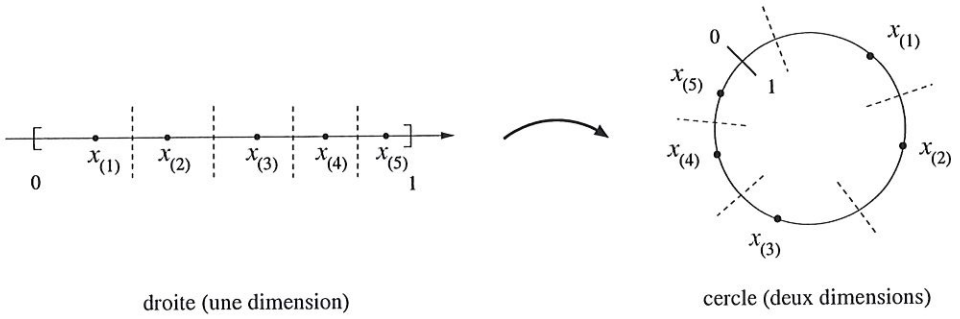


FIG. 3.2: *Passage d'un espace unidimensionnel à un espace torique*

Calculons dès lors les nouvelles longueurs des cellules l_1 et l_n. On trouve

$$l_1 = \frac{x_{(2)} - x_{(n)} + 1}{2} \tag{3.1}$$

$$l_n = \frac{x_{(1)} - x_{(n-1)} + 1}{2}. \tag{3.2}$$

Notons qu'on retrouve la présence du chiffre 1 dans les formules. Ainsi, si on avait pris un intervalle $[0, b]$, les formules seraient $l_1 = \frac{1}{2}\left(x_{(2)} - x_{(n)} + b\right)$ et $l_n = \frac{1}{2}\left(x_{(1)} - x_{(n-1)} + b\right)$: la présence du terme b relève d'un effet d'échelle (équivariance), et pas d'effet de bord.

Une autre solution aux effets de bord est de répéter l'échantillon autour de l'échantillon initial, pour « compléter » les cellules-frontières. Nous montrons ici qu'en une dimension, cette méthode est équivalente au tore. Tout d'abord, on voit bien sur la figure 3.3 que la longueur des cellules V_1 et V_n n'est plus influencée par les bornes 0 et 1.

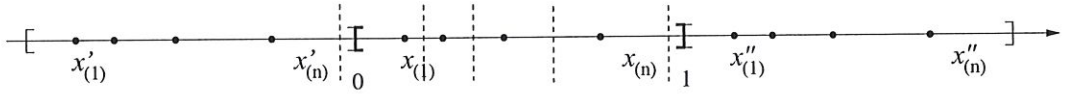


FIG. 3.3: *L'échantillon est répété à gauche et à droite de l'échantillon de départ*

Bien sûr, en pratique, on se contentera juste de « recopier » les points $x'_{(n)}$ et $x''_{(1)}$ à gauche et à droite de l'échantillon (cf. figure 3.4).
Avec cette méthode, la détermination de l_1 et l_n donne :

$$l_1 = \frac{x_{(2)} - x_{(n)} + 1}{2}$$

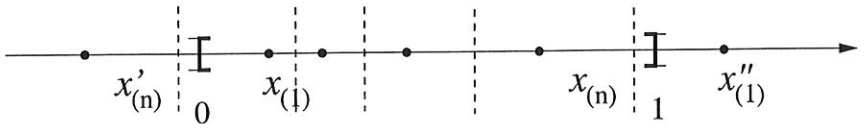


FIG. 3.4: *Seuls les points significatifs de l'échantillon sont répétés à gauche et à droite de l'échantillon de départ*

$$l_n = \frac{x_{(1)} - x_{(n-1)} + 1}{2},$$

c.-à-d. les mêmes résultats qu'avec la méthode du tore. Nous verrons que ceci est encore vrai dans les espaces de dimension supérieure.

L'espace torique en dimensions supérieures

Le principe est de refermer le domaine $B = [0,1]^d$ sur lui-même(d est la dimension de l'espace), en joignant entre elles ces frontières opposées : puisque l'espace est totalement clos sur lui-même, il ne peut plus y avoir d'effets de bord. Par exemple, dans la figure 3.5 (a), les points générateurs sont distribués dans le carré unité. Nous connectons entre eux, en déformant le carré, les côtés $\overline{ab}$ et $\overline{cd}$ (ce qui géométriquement donne un cylindre), puis les côtés $\overline{ac}$ et $\overline{bd}$ (ce qui donne un tore).

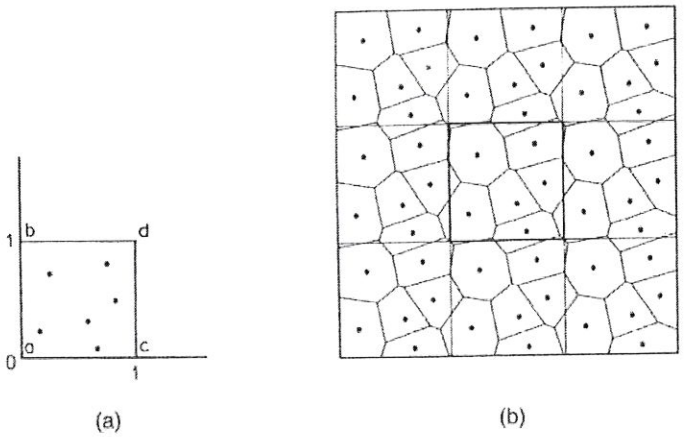


FIG. 3.5: *Un diagramme de Voronoi en $\mathbb{R}^2$ sur un tore*

Ainsi, la distance entre deux points $p_1 = (x_1, y_1)$ et $p_2 = (x_2, y_2)$ sur ce carré déformé est

$$\begin{aligned} d^2_{\text{tore}}(p_1,p_2) = \min \{ & (x_1 - x_2 - 1)^2 + (y_1 - y_2 - 1)^2,\ (x_1 - x_2 - 1)^2 + (y_1 - y_2)^2, \\ & (x_1 - x_2 - 1)^2 + (y_1 - y_2 + 1)^2,\ (x_1 - x_2)^2 + (y_1 - y_2 - 1)^2, \\ & (x_1 - x_2)^2 + (y_1 - y_2)^2,\ (x_1 - x_2)^2 + (y_1 - y_2 + 1)^2, \\ & (x_1 - x_2 + 1)^2 + (y_1 - y_2 - 1)^2,\ (x_1 - x_2 + 1)^2 + (y_1 - y_2)^2, \\ & (x_1 - x_2 + 1)^2 + (y_1 - y_2 + 1)^2 \}. \end{aligned} \tag{3.3}$$

Okabe et al. [11, p. 213] précisent que ce tore obtenu est en fait un « *quasi-tore* » dans le sens où une distance entre deux points sur le tore n'est pas mesurée par la plus courte distance sur ce tore mais par la distance euclidienne sur le rectangle développé à partir du tore.

Nous constatons que nous obtenons la même distance si nous plaçons huit carrés avec les mêmes points générateurs, autour du carré initial, comme à la figure 3.5 (b). Dès lors le diagramme de Voronoi sur le tore est le même que celui dessiné dans le carré central de cette figure. Autrement dit, considérer un tore est équivalent à répliquer l'échantillon de points générateurs autour du domaine B initial.

Conceptuellement, nous ajoutons en dehors de chaque face de B des points qui sont des translations des points juste à l'intérieur de la face opposée. Ces points servent alors à « compléter » les cellules-frontière (cf. figure 3.6).

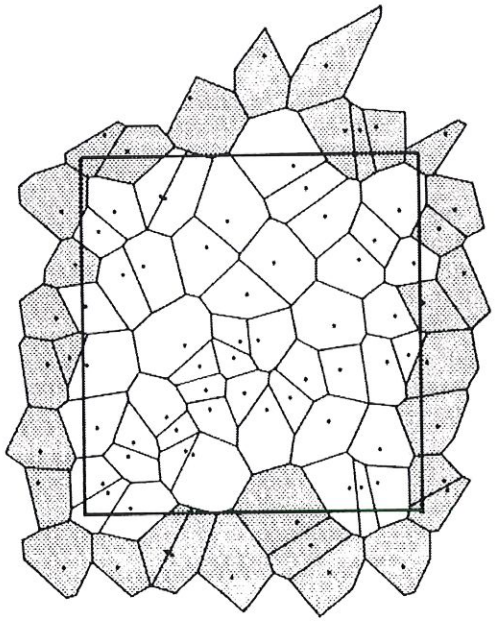


FIG. 3.6: *Répéter des points pour corriger la forme des cellules-frontière et de cette manière annuler les effets de bord*

En pratique, dans l'algorithme, nous calculerons les distances sur le tore au moyen de la formule (3.3) pour un espace bidimensionnel. La généralisation aux dimensions supérieures ne pose pas de problèmes.

Nous remarquons qu'en deux dimensions, il faut répliquer huit carrés autour de l'original, soit neuf carrés au total. Pareillement, en trois dimensions, au total vingt-sept cubes seront nécessaires, et en d dimensions, ce sera 3^d hypercubes. Au niveau de l'implémentation, il faudra comparer entre elles 3^d distances pour trouver la distance sur le tore entre deux points p_1 et p_2.

3.3.3 Le problème de la dépendance des cellules

Soit n, le nombre cellules de Voronoi générées dans $B = [0,1]^d$. Il se fait qu'on n'a jamais n degrés de liberté, puisque

$$\sum_{i=1}^{n} vol(V_i) = vol(B) = 1, \tag{3.4}$$

où $vol(\cdot)$ représente l'hypervolume.

Autrement dit, théoriquement les cellules ne sont jamais indépendantes entre elles. Mais il est évident que si n est assez grand, l'influence de chaque cellule sur les autres (c.-à-d. sur leur forme, leurs caractéristiques) diminue.

P. ex. la cellule 1 n'influencera plus la cellule 124, sauf si la cellule 1 est très grande : dans ce cas, il y aura beaucoup de chance que la cellule 124 soit petite (la cellule 1 influence donc l'hypervolume de la cellule 124). Toutefois, si n est assez élevé, il serait étonnant qu'on trouve encore de grandes cellules.

Donc, en pratique, une solution à ce problème est de considérer un nombre de cellules suffisamment grand pour éviter tout biais.

3.4 Quadrillage

3.4.1 Calcul du nombre de points-grille

Nous proposons de déterminer ici le nombre de points-grille qui se trouvent sur la frontière du domaine $B = [0,1]^d$, tel qu'il en est dans le travail de Vandooren (voir [18]). Dans ce travail, les points-grille sont disposés selon un quadrillage parfait (bornes comprises). Soient donc N, le nombre total de points-grille (choisi par l'utilisateur) et k, le nombre de points-grille par dimension (obligatoirement entier) :

$$k = \left\lfloor \sqrt[d]{N} \right\rfloor .^{5} \tag{3.5}$$

Soit un point-grille $x = (x_1, x_2, \ldots, x_d)$; chaque coordonnée x_j peut prendre une valeur dans l'ensemble

$$A = \left\{ \frac{i}{k-1} \;\middle|\; 0 \leq i < k,\ i \in \mathbb{N} \right\}. \tag{3.6}$$

Notons que $\#A = k$ et que les x_j prennent bien les valeurs 0 et 1.

Le problème est donc de déterminer parmi toutes les points-grille x possibles le nombre de points-grille tel qu'au moins une de leurs composantes x_j ait la valeur 0 ou 1.

Soit u_d, le nombre de points-grille se trouvant à la frontière de $B = [0,1]^d$. On trouve que

$$u_1 = 2 \tag{3.7a}$$

$$u_{d+1} = 2k^d + (k-2)\,u_d\,. \tag{3.7b}$$

Donnons quelques exemples :

$$u_2 = 4k - 4$$
$$u_3 = 6k^2 - 12k + 8$$
$$u_4 = 8k^3 - 24k^2 + 32k - 16\,.$$

[5] La notation $\lfloor x \rfloor$ désigne la troncature du réel x.

Les deux premiers exemples peuvent se vérifier graphiquement. En effet, en deux dimensions, B est un carré de côté 1. Les points-grille qui sont à la frontière sont ceux situés sur les côtés de B. Il y a k points-grille par dimension (et quatre côtés), donc : $4k$ points-grille. Mais l'on a compté chaque sommet du carré deux fois, il faut donc les soustraire une fois : -4 points-frontière. Total : $(4k-4)$ points-frontière. Pour le cube B en trois dimensions, comptons le nombre de points sur les faces (il y a six faces) : $6k^2$ points-frontière. Mais en additionnant les points de chaque face, on a compté une fois de trop chaque arête (il y a douze arêtes) : $-12k$. Concernant les sommets du cube, ceux-ci ont été compté trois fois lorsqu'on a compté les faces ; puis ils ont été soustrait trois fois également lorsqu'on a ôté une fois les points des arêtes[6]. Donc il faut les recompter encore une fois (il y a en huit) : $+8$. Total : $(6k^2-12k+8)$ points-frontière.

En ce qui concerne ce mémoire, nous travaillons toujours dans l'espace torique généré à partir de $B=[0,1]^d$: ceci nous permet d'avoir un espace clos sur lui-même et d'ainsi éviter tout effet de bord pour les cellules (cf. chapitre 3, p. 31 et ss). Autrement dit, il n'y a plus à proprement parler de points-frontière ni de cellules-frontière.

Par ailleurs, dans ce mémoire, les points-grille x prendront leurs valeurs dans l'ensemble

$$A' = \left\{ \frac{i}{k} \;\middle|\; 0 \leq i < k,\ i \in \mathbb{N} \right\}. \tag{3.8}$$

Nous avons encore $\#A = k$, ce qui veut dire qu'il y a toujours bien k points-grille par dimension. Ce choix se justifie par le fait qu'il n'est plus nécessaire d'avoir des composantes x_j prenant la valeur 1 : effectivement, l'espace étant refermé sur lui-même, les valeurs 0 et 1 se confondent.

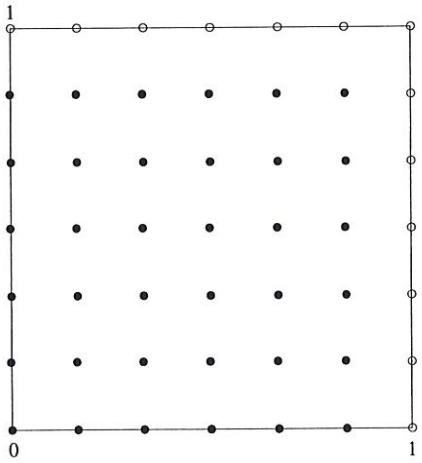


FIG. 3.7: *Éviter les points-grille redondants dans les espaces toriques*

Et donc, posons-nous la question : combien de point-grille économisons-nous lorsque nous passons à un espace torique, pour la même précision du quadrillage ?

Soit $B=[0,1]^d \subset \mathbb{R}^d$ et un quadrillage parfait sur B, avec k^d points-grille. Soit B_{ref}, l'hypercube complètement refermé sur lui-même (espace torique). Alors, pour la même précision du quadrillage[7], on a besoin de $(k-1)^d$ points-grille. Autrement dit, on « économise » $k^d-(k-1)^d$ points-grille.

[6] Puisque chaque sommet est à l'intersection de trois faces et de trois arêtes.

[7] Ces notions de *quadrillage parfait* et *précision du quadrillage* seront définies rigoureusement plus loin. Ici, une idée intuitive de ces notions suffit.

Finalement, lorsque nous utilisons un espace torique, nous économisons le calcul de quelques points-grille, pour la même précision du quadrillage que dans un espace non-torique. Cependant, passer à un espace torique demande de calculer beaucoup plus de distances. En définitive, y gagne-t-on en cout calculatoire à passer à une représentation torique, ou non ? et dans quelles proportions ? C'est ce que nous allons nous attacher à déterminer dans ce qui suit.

3.4.2 À propos des distances toriques

Comme nous l'avons vu à la page 33, le calcul d'une distance torique en d dimensions demande de calculer et de comparer entre elles 3^d distances euclidiennes. On se rend compte que passer d'un espace à d dimensions à un espace à $d+2$ dimensions demande de calculer environ 10 fois plus de distances ! Le tableau 3.2 permet au lecteur de se rendre compte du nombre de distances à calculer.

dimension	nombre de distances
2	9
3	27
4	81
5	243
6	729
7	2 187
8	6 561
9	19 683
10	59 049
12	531 441
15	14 348 907
20	3 486 784 401

TAB. 3.2: *Nombre de distances à calculer et à comparer entre elles pour déterminer* une *distance torique*

Ainsi, nous pouvons affirmer que c'est bien ce calcul des distances qui augmente considérablement le cout de l'algorithme lorsque la dimension augmente. Pourtant les distances toriques sont indispensables au-delà de quatre dimensions ; car lorsque la dimension augmente, le nombre de cellules-frontière augmente énormément, ce qui multiplie les effets de bord (cf. tableau 3.1, p. 30).

Nous pouvons aussi en avoir une idée en examinant le pourcentage de points-grille qui sont des points-frontière. Nous avons vu comment déterminer théoriquement le nombre de points-grille qui sont à la frontière du domaine $B = [0,1]^d$ pour un quadrillage parfait (cf. équations (3.7) et (3.7)). Soient N, le nombre total de points-grille, et k, le nombre de points-grille par dimension. Nous allons examiner ce pourcentage de points-frontière de deux façons : la première, en gardant N constant, la deuxième, en gardant k constant. Pour la première manière, nous prendrons $N = 10\,000\,000$, pour la deuxième, $k = 101$, car c'est la valeur qu'avait choisie Vandooren (cf. [18]) en trois et quatre dimensions.[8] Dans le premier cas, nous calculons k par la formule (3.5),

[8] Vandooren avait pris $k = 51$ en cinq dimensions, ce qui fait en tout : 1 030 301 points en 3-D, 104 060 401 point en 4-D et 345 025 251 points en 5-D.

puis nous recalculons N à partir de k (afin d'avoir bien k entier) : $N_{rec} = k^d$ (finalement le nombre de points n'est donc pas rigoureusement constant, mais il reste toujours dans le même ordre de grandeur). Les résultats sont présentés dans le tableau 3.3.

dimension	k	N_{rec}	points-grille à la frontière (%)
2	3 163	10 004 569	0.13
3	216	10 077 696	2.75
4	57	10 556 001	13.31
5	26	11 881 376	32.98
6	15	11 390 625	57.62

TAB. 3.3: *Quelques ordres de grandeurs sur le pourcentage de points-grille à la frontière ($N \approx 10^6$)*

dimension	k	N	points-grille à la frontière (%)
3	101	1 030 301	5.82
4	101	104 060 401	7.69
5	101	10 510 100 501	9.52
5	51	345 025 251	18.13
6	51	17 596 287 801	21.34

TAB. 3.4: *Quelques ordres de grandeurs sur le pourcentage de points-grille à la frontière ($k = 51$ ou 101)*

d	k	N	points-grille à la frontière (%)
3	50	125 000	11.53
3	75	421 875	7.79
3	100	1 000 000	5.88
3	150	3 375 000	3.95
4	30	810 000	24.12
4	50	6 250 000	15.07
4	100	100 000 000	7.76
5	30	24 300 000	29.18
5	50	312 500 000	18.46
5	75	2 373 046 875	12.64
6	30	729 000 000	33.90
6	50	15 625 000 000	21.72
10	10	10 000 000 000	89.26

TAB. 3.5: *Quelques ordres de grandeurs sur l'accroissement du nombre de points-grille à la frontière par rapport à la dimension*

3.4.3 Précision du quadrillage

Dans ce travail, nous avons toujours utilisé des quadrillages parfaits. Précisons cette notion de *quadrillage parfait* : il s'agit d'un quadrillage tel que les points-grille prennent leurs valeurs dans l'ensemble A défini à l'équation (3.6). Chaque point-grille

se trouve au centre d'un petit carré défini par les médiatrices entre ce point et ses plus proches voisins (cf. fig. 3.8 (a)). Nous appelerons ces carrés les *zones* occupées par les points-grille.

Nous avons naturellement défini la variable `k` comme indication de la précision du quadrillage. À la figure 3.8 (a), nous avons `k` $= 4$: si nous projetons chaque point sur l'abscisse (l'ordonnée), nous obtenons bien au total 4 points sur l'abscisse (l'ordonnée).

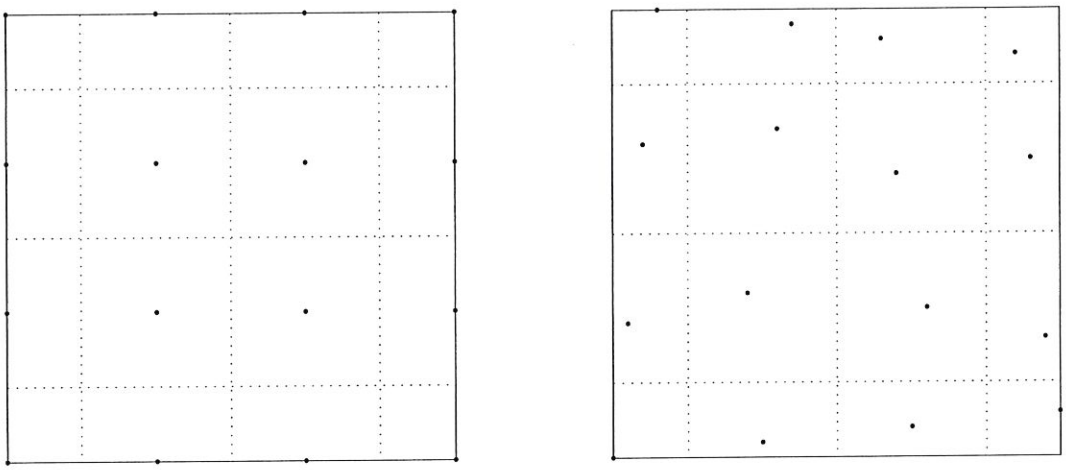


FIG. 3.8: (a) *Quadrillage parfait sur le domaine* $B = [0,1]^2$ (b) *Quadrillage régulier tel qu'aucun point n'est aligné par rapport à un autre*

Cependant, il s'agit bien d'une *indication* de la précision, la variable `k` ne représente pas *toute* la précision. Décalons en effet légèrement chaque point par rapport à sa position initiale, horizontalement et verticalement, de sorte que chaque point ne soit aligné avec aucun autre : dans ce cas, la valeur de `k` sera plus grande, alors qu'en réalité, la précision reste la même : à la figure 3.8 (b), nous avons `k` $= 16$, car lorsque nous projetons chaque point sur l'abscisse ou l'ordonnée, nous obtenons bien 16 points. Nous appelons un quadrillage tel que celui de la figure 3.8 (b), un *quadrillage régulier*, parce qu'il y a toujours un et un seul point-grille dans chaque zone définie par le quadrillage parfait.

En fait, il faudrait plutôt définir la précision du quadrillage par la « place » (l'hypervolume) occupé par chaque point-grille, par rapport au volume total. Effectivement, l'espace occupé par chaque point-grille est le même dans les deux figures 3.8 (a) et 3.8 (b). Autrement dit, la précision du quadrillage sera définie par

$$Pr = \frac{vol(B)}{\texttt{nbgrid}} = \frac{1}{\texttt{nbgrid}}. \tag{3.9}$$

Dès lors, plus `nbgrid` sera élevé, plus la valeur Pr sera faible : cela indiquera la « finesse » de la précision.

Le problème ici est que l'hypervolume total du domaine B vaut toujours 1, puisque $B = [0,1]^d$. Si nous prenions $B = [0,10]^d$, nous constaterions tout de suite l'augmentation du volume avec la dimension : $vol(B)$ vaudrait 10 en une dimension, 100 en 2-D, 1 000 en 3-D, etc. Ici, cette augmentation est cachée dans la dimension.

Remarquons que, dans ce travail, nous n'avons œuvré qu'avec des quadrillages parfaits. On voit facilement comment améliorer le programme... en générant des

points-grille aléatoirement. Mais pas n'importe comment ! Les points-grille seraient attractifs des points générateurs (là où il y a beaucoup de points générateurs, on place davantage de points-grille, pour améliorer la précision des calculs).

Chapitre 4

Nombres aléatoires

4.1 Introduction

Nous avons vu au chapitre 1 comment était caractérisée la distribution uniforme (cf. p. 2) : un point n'a pas plus de chances de se trouver là où il est que partout ailleurs. C'est souvent de cette manière que l'on conçoit la notion de *nombres aléatoires*. En réalité, il y a d'autres sortes de nombres aléatoires, comme par exemple des nombres provenant d'une distribution normale (gaussienne) avec une moyenne et un écart-type spécifiés à l'avance. En pratique, ces autres sortes de nombres aléatoires sont presque toujours générés à partir d'opérations appropriées réalisées sur une ou plusieurs distributions uniformes. Quoi qu'il en soit, la distribution uniforme de points reste la notion la plus intuitive que l'on se fait de nombres aléatoires, et c'est ce qui explique pourquoi nous nous attachons dans ce travail à tester cette distribution précisément, au moyen des diagrammes de Voronoi.

Question importante : comment obtenir un générateur dans $[0,1]^d$?

Si nous voulons générer n points aléatoires uniformes sur $[0,1]^d$, il suffit de générer nd points avec le générateur que nous avons déjà, et de considérer les d premiers nombres comme le premier point, les d nombres suivants comme le deuxième point,... Le résultat sera bien un échantillon de n points aléatoires uniformes sur $\mathbb{R}^d$.

Il est indispensable, pour que cet échantillon sur $\mathbb{R}^d$ soit bien uniforme, que toutes les réalisations du générateur soient bien indépendantes. Dans la conception des générateurs, les informaticiens ont beaucoup œuvré pour éviter justement des corrélations entre les nombres, comme p. ex. la présence de droite sur $\mathbb{R}^2$ (c.-à-d. des points alignés), la présence de plans sur $\mathbb{R}^3$,...
Le résultat théorique suivant assure que l'échantillon obtenu est bien uniforme sur $\mathbb{R}^d$: Soient X et Y VAIID $\sim U_{[a,b]}$; alors $(X,Y) \sim U_{[a,b]^2}$.

4.2 Générateurs utilisés

On utilisera principalement le générateur suivant :

- algorithme de Pierre L'Écuyer : *Efficient and Portable Combined Random Number Generators*, Communications of the ACM, 31 (1988), 742–749 and 774 ; il s'agit de la publication n° 114 sur le site de P. L'Écuyer, http://www.iro.umontreal.ca/~lecuyer/papers.html ;

- implémentation de Fabian Bastin en C, http://www.fundp.ac.be/~fbastin/index.fr.php (page d'accueil), http://www.fundp.ac.be/~fbastin/computing/rangen/index.fr.html (adresse précise pour la génération de nombres aléatoires, librairie `ran-0.6.0`), http://www.fundp.ac.be/~fbastin/computing/languages.fr.html (pour télécharger la librairie `ctools-0.5.1`, nécessaire à la compilation).

La librairie `ran-0.6.0` inclut également le générateur de Park & Miller (utilisé notamment par Vandooren [18]). C'est un générateur classique et qui a passé tous les tests standards en matière de générateurs de nombres aléatoires. C'est ce générateur qui par défaut est utilisé. Malheureusement, j'ai par distraction parfois utilisé ce générateur au lieu de celui de L'Écuyer exclusivement. Dans les tests, je préciserai toujours quel générateur est à l'origine de l'échantillon de points. Nous ne pousserons pas l'analyse des tests à essayer de différencier quel générateur a été utilisé.

Notons que le langage C possède déjà un générateur de nombres aléatoires : la fonction `rand()`, qui calcule une séquence d'entiers compris entre 0 et `RAND_MAX`, défini dans la librairie `<stdlib.h>`. On peut donc produire un nombre aléatoire en virgule flottante, comris entre 0 et 1, par la macro (cf. [8, p. 166]) :

```
#define frand()  ((double) rand() / (RAND_MAX+1))
```

Voici l'implémentation de la fonction `rand()` telle que définie par Kernighan & Ritchie [8, p. 46] (ce livre date de 1997, seconde édition) :

```
unsigned long int suivant = 1;
/* rand: retourne un nombre entier pseudo-aléatoire compris entre
         0 et 32767 */
int rand(void)
{
   suivant = suivant * 1103515245 + 12345;
   return (unsigned int)(suivant/65536) % 32768;
}

/* srand: donne une valeur initiale à rand() */
void srand(unsigned int amroce)
{
   suivant = amorce;
}
```

Kernighan & Ritchie notent que le C ANSI impose que `INT_MAX` $\geq 32\,767$ seulement (cf. [8, p. 263]). Toutefois, cette référence prend en compte le fait que les machines Linux étaient codées sur 16 bits ; donc, pour le compilateur `gcc`, le type `int`, qui est toujours codé sur un mot-mémoire, prenait deux octets, d'où `INT_MAX` était égale à 32 767. C'est notamment ce que reprochent les auteurs de [13, p. 276] au comité C ANSI et aux programmeurs : la fonction `rand()` du C renvoie une valeur de type `int`, qui peut n'être défini qu'avec deux octets, et donc la constante `RAND_MAX` n'est pas très grande. Donc la période du générateur reste très limitée, et cela Dpeut s'avérer catastrophique dans certains cas. Voici un exemple donné par ces auteurs : pour une intégration de Monte-Carlo, on veut évaluer 10^6 points différents. Avec ce

générateur défini ci-dessus, on évaluera les mêmes 32 767 points 30 fois chacun ! Kernighan & Ritchie, les auteurs du C, précisent d'ailleurs que « Si votre bibliothèque fournit déjà une fonction générant des nombres aléatoires en virgule flottante, elle a probablement de meilleures propriétés statistiques que celle-ci [la fonction `frand()` définie plus haut]. » ([8, p. 166])

Sur la plupart des machines actuelles, ce problème de la faible valeur de `RAND_MAX` est corrigé. La machine utilisée pour ce travail est en architecture 32 bits, avec comme système d'exploitation Linux RedHat 8.0 : les objets de type `int` sont donc codés sur quatre octets (soit 32 bits, soit un mot-mémoire) et `RAND_MAX` = 2 147 483 647. Notons cependant que la norme IEEE est encore assez floue à propos du générateur de nombres aléatoires dans les librairies C standard. Donc, son implémentation reste dépendante de l'architecture de la machine utilisée, ce qui peut rendre un programme qui l'utilise moins portable.

> Nous renvoyons le lecteur qui souhaiterait connaitre les valeurs des constantes des types entiers et réels du C/C++, à la consultation des fichiers d'en-tête `values.h>`, `limits.h>` et `<float.h>` sur sa machine. Sur un système d'exploitation de type Linux RedHat 8.0, ces fichiers sont consultables dans les répertoires[1]
> `/usr/include/values.h`
> `/usr/include/limits.h`
> `/usr/lib/gcc-lib/i386-redhat-linux/3.2/include/float.h`
> À titre indicatif, les valeurs de ces constantes, pour la machine utilisée dans ce travail, sont placées en annexe.

Nous pouvons également consulter les pages de manuel à propos des fonctions `rand`, `srand` et `random` (Linux RedHat 8.0). Nous y apprenons que « les versions de `rand()` et `srand()` de la bibliothèque C de Linux utilisent le même générateur de nombres aléatoires que `random()` et `srandom()` », et aussi que « La génération de nombres aléatoires est un domaine complexe. Le livre *Numerical Recipes in C* [cf. [13]] fournit une excellente présentation pratique d'un générateur aléatoire dans le chapitre 1. »

En conclusion, il ne fait pas de doute que les librairies C standard *actuelles* sur Linux (`gcc`) fournissent un générateur de bonne qualité. Néanmoins, utiliser un générateur comme celui de Fabian Bastin offre la garantie de l'indépendance de ce générateur par rapport à l'architecture du système.

Remarquons que dans l'implémentation d'un générateur congruentiel dans un langage de haut niveau, il est fortement recommandé d'utiliser des réels en double précision (cf. [15, p. 46]). C'est le cas pour le générateur de Fabian Bastin.

[1] On peut aussi les localiser au moyen de la commande `locate nom_fichier.h`.

Chapitre 5

Test d'uniformité en une dimension sur base de cellules de Voronoi (partie pratique)

5.1 Quelques tests

Nous utilisons ici le générateur référencé à la page 40. Les tests se font dans un intervalle $B = [0, 1]$. Nous prendrons toujours $n = 1\,000$, où n est le nombre de points (et donc de cellules de Voronoi). Précisons encore que nous travaillons bien dans un espace torique : il n'y a pas d'effets de bord sur la première et la dernière cellule. Tous les graphiques statistiques de ce travail sont donnés par le logiciel Splus 2000 pour Windows, excepté, quand il y en a, les *scatter plot*[1], qui sont générés à partir de GNUplot (version 3.7 patchlevel 2 pour système Linux 2.4.18-14, 19 janvier 2002). Les données à partir desquelles ils sont construits sont bien sûr les longueurs des cellules de Voronoi, calculées par un programme implémenté sous Splus 2000. Ce programme est repris en Annexe. Enfin, l'intensité du PP, λ, vaut :

$$\lambda = \frac{n}{m(B)}, \tag{5.1}$$

c.-à-d. ici $\lambda = n$.

• **Premier échantillon de points.**

[1] Un *scatter plot* est simplement la mise en graphe des données. S'il y a juste une seule colonne de données, le scatter plot met en abscisse le numéro de la coordonnée (1, 2, 3 etc.) et en ordonnée sa valeur. S'il y a deux ou trois colonnes de données, le scatter plot est bi- ou tridimensionnel.

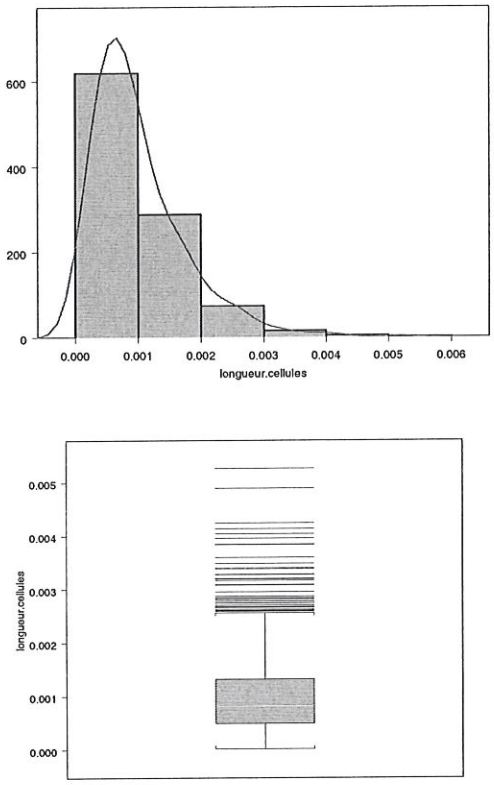


FIG. 5.1: *Histogramme et courbe de densité et box plot des longueurs des cellules*

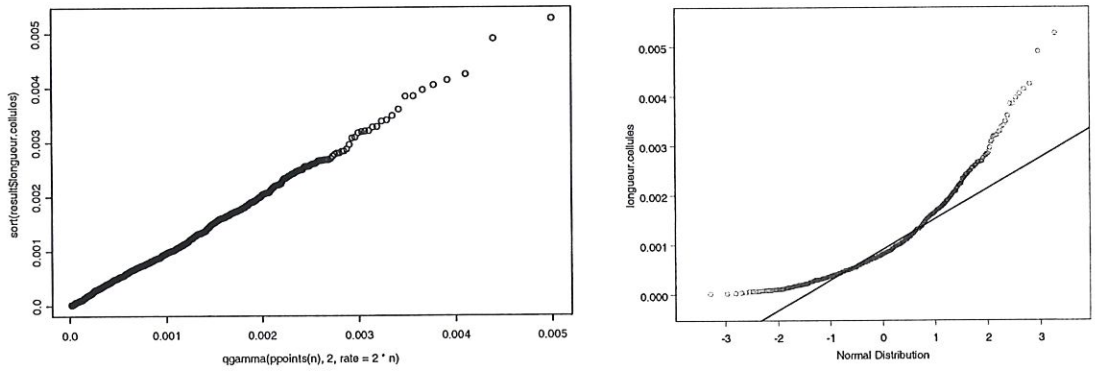


FIG. 5.2: *QQ plot Gamma et QQ plot Normal avec lignes de densité des longueurs des cellules*

Expliquons un peu ces graphiques.
Le premier graphe donne *l'histogramme et la courbe de densité* des longueurs des cellules.
Les *QQ plot* permetent d'estimer si les données suivent une distribution particulière (ou si deux ensembles de données suivent la même distribution). Si c'est le cas, le graphe aura approximativement l'allure d'une droite. Les points extrêmes ont plus de variabilité que les points vers le centre. Un graphe en forme de « U » indique une

distribution biaisée des données par rapport à celle de référence. Un graphe en forme de « S » implique qu'une des deux distributions a des « queues »[2] plus longues que l'autre[3]. Ici, *QQ plot Gamma* indique si les données suivent la distribution $\Gamma(2\lambda, 2)$
Le *QQ plot Normal* est le type de QQ plot le plus fréquemment employé. Il est mis ici de manière facultative[4].
Le *box plot* fournit une « vue d'avion » de la fonction de densité des données : la boite représente 95% de l'aire sous la fonction de densité, la boite plus les moustaches (les segments horizontaux extrêmes) représentent plus de 95% de la fonction de densité, mais pas forcément 100%. Les outliers (ici les points en-deçà du 5^e percentile et au-delà du 95^e percentile) sont mis en évidence ; la petite ligne claire dans la boite représente la médiane. Le box plot montre la localisation et l'étendue des données, mais il peut indiquer aussi la présence de biais.

- **Second échantillon de points.**

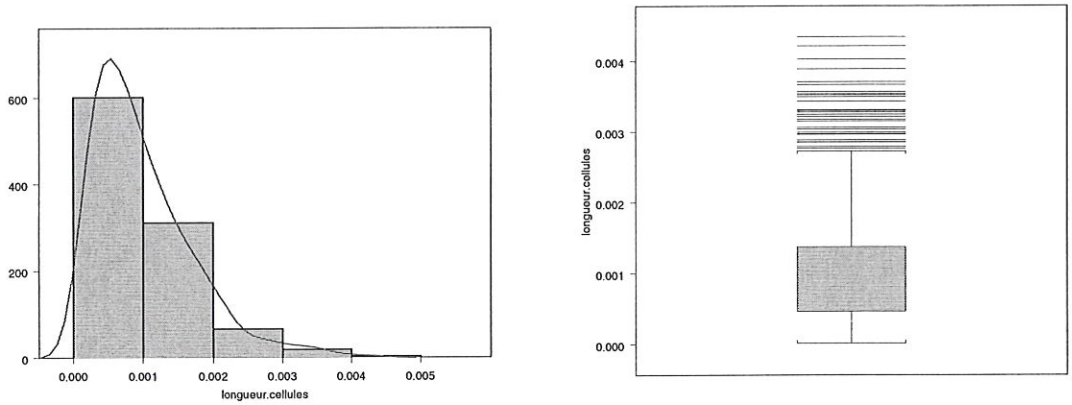


FIG. 5.3: *Histogramme et courbe de densité et box plot des longueurs des cellules*

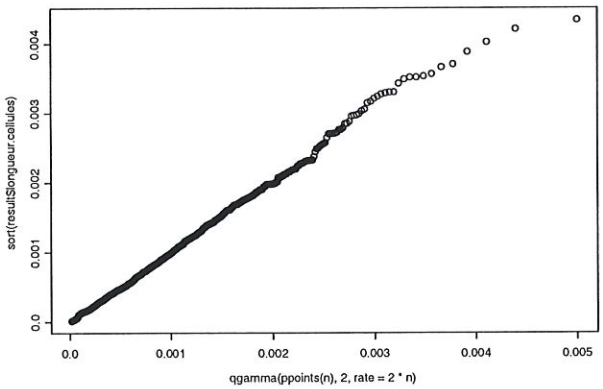


FIG. 5.4: *QQ plot Gamma des longueurs des cellules*

[2] c.-à-d. les régions $P[X \leq x_{0.05}]$ et $P[X \geq x_{0.95}]$, par exemple, où $x_{0.05}$ et $x_{0.95}$ représentent les 5^e et 95^e percentiles.

[3] Dans le QQ plot Normal, un graphe qui est courbé vers le bas à gauche et vers le haut à droite signifie que les données ont des queues plus longues que la loi Normale.

[4] Car on sait bien que les données suivent idéalement une loi $\Gamma(2\lambda, 2)$.

Conclusions

Les tests sont concluants. Les graphes sont parlants : l'allure des fonctions de densité des données ressemble fort à première vue à celle d'une $\Gamma(\cdot,\cdot)$ (mais cela n'est pas suffisant pour confirmer les valeurs des paramètres λ et α) ; les graphes QQ plot $\Gamma(2\lambda, 2)$ et permettent de se rendre compte de la pertinence des valeurs des paramètres ($\alpha = 2$ et $\lambda = 2 \cdot \frac{n}{m(B)} = 2\,000$) ; enfin le boxplot montre bien que la fonction de densité n'est pas symétrique (comme une normale), l'allure du boxplot correspond bien à une fonction de densité de style $\Gamma(\cdot,\cdot)$.

Chapitre 6

Explication des algorithmes

Dans ce chapitre, et dans les suivants, nous utiliserons plus directement le nom de certaines variables du programme, dans un souci de concision. Nous donnons ici les définitions de ces variables.

`Gr[i]`	nombre de points-grille se trouvant dans la cellule `i`
`Vol[i]`	hypervolume de la cellule `i`
`nbcell`	nombre de cellules (égal au nombre de points générateurs)
`nbgrid`	nombre total de points-grille (entré par l'utilisateur et recalculé par le programme de sorte que `k` soit entier
`k`	le nombre de points-grille par dimension
`dmax[i]`	distance entre le point générateur de la cellule `i` et le point-grille appartenant à cette cellule, le plus éloigné du point générateur ($0 \leq$ `i` $<$ `nbcell`)[1]
`dmoy[i]`	distance moyenne entre le point générateur de la cellule `i` et tous les points-grille présents dans cette cellule (ceci est une approximation de l'inertie de la cellule `i`)
`dmin[i]`	distance entre le point générateur de la cellule `i` et le point-grille le plus proche de ce point

6.1 Complexité de l'algorithme `dVOR`

Brièvement, nous pouvons dire que la complexité du programme est *exponentielle*, à cause des distances toriques : on l'a déjà vu précédemment (cf. table 3.2). Grosso modo, la complexité du programme se calcule par

$$3^d \times \texttt{nbgrid} \times \texttt{nbcell}.$$

En effet, pour chaque point-grille, le programme détermine quel est le point générateur le plus proche : il y a donc `nbcell` distances à calculer pour chaque point-grille ; comme on travaille dans un espace torique, il faut évaluer 3^d distances pour déterminer la distance torique entre deux points.

À partir d'un lancement du programme (pas trop court), nous pouvons prévoir le temps que prendront les autres lancements. Prenons le cas de la machine définie à

[1] En C/C++, les tableaux commencent toujours à l'indice 0, et pas 1 comme en Pascal. Aussi, par facilité, nous prendrons cette convention que les indices des cellules vont bien de 0 à `nbcell` -1.

la table B.2. Le test 5-D uniforme fait avec cette machine a pris 53 627 secondes (soit 14 heures, 53 minutes et 47 secondes). On a pris 1 000 cellules et 100 000 points-grille. Le nombre total d'opérations est donc

$$3^5 \times 10^3 \times 10^5 = 24.3\ 10^9.$$

Reste à savoir à combien d'opérations élémentaires, sur cette machine, correspondent ces opérations « programme ». Nous calculons

$$\frac{24.3\ 10^9}{53\,627} = 453\,130,$$

$$\frac{800\ 10^6}{453\,130} = 1\,765.5.$$

Ce dernier résultat est le nombre d'opérations élémentaires que doit effectuer la machine B.2 pour accomplir une opération « programme » (la quantité 800 10^6 est la fréquence du processeur).

Donc, pour un lancement du programme, nous calculons :

$$3^d \times \texttt{nbcell} \times \texttt{nbgrid} \times \frac{1\,765.5}{800\ 10^6},$$

et cela nous donne une estimation du temps (en secondes) que prendra le programmme.

Pour la machine définie à la table B.3, nous avons que le nombre d'opéarations élémentaires pour une opération « programme » est 1 095.5. S'il est moins élevé que dans le cas de la machineB.2, c'est sans doute dû à la mémoire cache deux fois plus élevée de la machine B.3. La machine B.2 reste toutefois plus performante, car on a

$$\frac{1\,765.5}{800\ 10^6} < \frac{1095.5}{333\ 10^6}.$$

Ces estimations peuvent sembler naïves, mais nous avons pu constater combien elles étaient précises en pratique, pour des tests de trois ou quatre heures. Elles semblent par contre être trop optimistes pour des tests de longue durée ($\approx$ 40 heures).

6.2 Justification de l'implémentation

Dans cette section nous justifions certains choix effectués pour l'implémentation en C++ du programme `dVOR`.

Types arithmétiques des variables

Tout d'abord, nous n'utilisons jamais de `short`. Il est vrai que le type `short` est un type entier moins couteux en mémoire que les types `int` et `long`. Cependant, l'ordinateur travaille par « mots-mémoire », et il place deux variables de type `short` sur un seul mot, ce qui rend l'accès à la deuxième variable plus couteux[1] ; le cout en calcul peut être très significatif si des objets de type `short` sont utilisés dans de grandes boucles.

[1] Puisqu'il faut d'abord parcourir la première moitié du mot (la première variable) pour accéder à la deuxième variable.

Faut-il utiliser des entiers de type `int` ou `long` ? Nous savons que toutes les machines 32 bits avec comme système d'exploitation Linux, utilisent quatre octets pour stocker un objet de type `int` (compilateur `gcc`), ce qui fait qu'il n'y pas de différence entre les types `int` et `long`. C'est notamment le cas de la machine utilisée pour ce travail (cf. table B.1).

Pour les réels, il est préférable d'utiliser des nombres en double précision pour la génération de points aléatoires (cf. chapitre 4). Ainsi les points générateurs des cellules seront exprimés en `double`, et, pour la précision des résultats, toutes les variables de type réel seront exprimées également en `double`. L'utilisation d'objets de type `double`, par rapport à des objets de type `float`, donne une meilleure précision pour un cout calculatoire à peine plus grand.

Nous avons également observé certaines erreurs de conversion entre les types `float` et `double`. Dans notre programme, nous utilisons plusieurs fois la fonction prédéfinie du C `pow`, de la librairie `<math.h>` (voir [8, p. 256]) :

```
double pow(double x, double y)
PRE:
POST: x^y (erreur de domaine si x=0 et y<=0,
           ou si x<0 et y n'est pas un entier)
```

Comme on le voit, elle renvoie un objet de type `double`. Au départ, nous utilisions le type `float` pour des variables comme `nbgrid`. Voici un extrait de ce programme (avant qu'on ne redéfinisse les variables réelles comme `double`) :

```
k = (long)ceil(pow(nbgrid, 1.0/((double)dim)));
nbgrid = (float)pow((double)k,(double)dim);
```

Nous avons parfois constaté quelques erreurs quant à la détermination précise de la valeur de `nbgrid`. Quelques petits tests ont mis en évidence la cause de ce problème : cela n'était pas dû à une anomalie de la fonction `pow` (ce qui est *vraiment* peu probable) mais plutôt à la conversion du résultat de cette fonction en un objet de type `float`. Sont-ce des erreurs d'arrondi ? C'est possible. Ces erreurs ne sont pas vraiment aléatoires (les mêmes nombres donneront toujours les mêmes erreurs), mais je n'ai pas pu cerner ce qui caractérisait leur apparition : je n'ai pu que les constater à partir de certains valeurs des variables `k` et `dim`. Leur amplitude, par contre, semble réellement dépendre de la grandeur du nombre traité : cf. tableau 6.1 ; les variables y présentes sont définies par $X = \texttt{k}^{\texttt{dim}}$ (valeur exacte), $Y = \texttt{(float)pow(k, dim)}$; $\frac{|X-Y|}{X}$ représente la proportion (en %) de points en plus ou en moins. Sont-ce des erreurs propres au compilateur `gcc` (cf. tableau B.1) ? à la machine utilisée ? Je n'en sais rien. Néanmoins, toute trace d'erreur disparait lorsque l'on définit la variable `nbgrid` comme `double`.

Nous avons donc préféré l'usage du type `double` également pour cette raison : éviter toute erreur de conversion entre les types réels, même si le calcul de la proportion de points en plus ou en moins (l'erreur relative) nous indique que ces erreurs d'arrondi étaient finalement négligeables.

k	dim	X	Y	$\lvert X-Y\rvert$	$\lvert X-Y\rvert/X$ (en %)
25	6	244 140 625	244 140 624	1	$4.096\ 10^{-7}$
25	7	6 103 515 625	6 103 515 648	23	$3.768\ 10^{-7}$
25	8	152 587 890 625	152 587 894 784	4 159	$2.726\ 10^{-6}$
101	5	10 510 100 501	10 510 100 480	21	$1.998\ 10^{-7}$
51	5	345 025 251	345 025 248	3	$8.695\ 10^{-7}$
51	6	17 596 287 801	17 596 286 976	825	$4.688\ 10^{-6}$

TAB. 6.1: *Valeurs qui faussent la conversion de* `float` *vers* `double` *en C/C++, pour la machine et le compilateur définis à la table* B.1

Mesure du temps d'exécution d'un programme

Il existe en C une fonction prédéfinie pour cela, dans la librairie `<time.h>` :

```
clock_t clock(void)
```

`clock` retourne le temps d'utilisation du processeur par le programme depuis le début de son exécution, ou bien `-1` si cette information n'est pas disponible. Le résultat de l'instruction `clock()/CLOCKS_PER_SEC` est une durée en secondes. Le type `clock_t` est un type arithmétique entier qui représente des tics d'horloge.

Pour la machine définie à la table B.1, nous avons que `INT_MAX` $= 2^{31} - 1 =$ 2 147 483 647 (cf. table B.5) et `CLOCKS_PER_SEC` $= 10^6$.[2] Dès lors, la fonction `clock` peut mesurer un temps d'exécution d'un programme inférieur à

`INT_MAX/CLOCKS_PER_SEC`

, soit 2 147 s, soit un peu plus que 35 minutes. Nous nous rendons compte que nous disposons ici d'une fonction d'une précision remarquable, mais inadéquate pour mesurer le temps d'exécution d'un programme très long, ce qui est le cas pour `dVOR`.

Nous utiliserons donc les fonctions `nlsps_init_timer` et `nlsps_get_timer` écrites par Fabian Bastin :

```
void nlsps_init_timer()
PRE:
POST: initialise le compteur du temps d'exécution du programme

int nlsps_get_timer()
PRE:
POST: le temps (en secondes) écoulé depuis le début du programme
```

La fonction `nlsps_get_timer` peut renvoyer le temps d'exécution en secondes ou en microsecondes. Le compteur ici implémenté n'est pas troublé par du multitâche, car le calcul du temps d'exécution se fait à partir du processeur, non à partir du temps réel. Quel est ici la durée maximale d'exécution d'un programme que peut mesurer cette fonction ? La réponse est `MAX_TIMERVAL` secondes, où la constante `MAX_TIMERVAL` est définie dans l'implémentation de F. Bastin par `UINT_MAX`/100, soit ici $(2^32 - 1)/100$, soit $4.294\,97\ 10^7$ s, soit une année et 132 jours. Nous espérons que cela suffira.

[2] Il est logique que `CLOCKS_PER_SEC` vale 10^6 puisque le processeur fonctionne à une fréquence exprimée en MHz. Ainsi la fonction `clock` a une précision de l'ordre de la μs.

6.3 Vérifications empiriques du programme

Nous confrontons tout d'abord certains résultats théoriques à des résultats empiriques :

- le nombre de points-grille total (`nbgrid`) ;
- le nombre de cellules ne contenant aucun point-grille (`empty_cells`).

En ce qui concerne le nombre de points-grille total, le programme additionne le nombre de points-grille présent(s) dans chaque cellule : le total doit donner le nombre de points-grille donné au départ par l'utilisateur (et recalculé pour avoir `k` entier). Il pourrait y avoir une erreur au cas où un point-grille se trouverait pile sur la frontière entre deux cellules ou plus. Mais cela n'arrive pas avec le programme `dVOR`, car il n'« affecte » jamais un point-grille qu'à une seule cellule. De plus, vu la précision des calculs[3], il est quasi-impossible qu'un point-grille puisse se trouver *pile* sur la frontière entre deux cellules.

En ce qui concerne le nombre de cellules qui ne contiennent aucun point-grille, il vaut toujours mieux que ce nombre soit nul, autrement certaines cellules auraient un volume nul, et le calcul des distances minimales et maximales entre le point générateur et l'ensemble des points-grille de la cellule, serait faussé lui aussi.

Ensuite, nous vérifions les assertions suivantes :

$$\texttt{dmax[i]} \geq \texttt{dmoy[i]} \geq \texttt{dmin[i]} \geq 0, \tag{6.1}$$

$$\texttt{Gr[i]} > 0, \qquad \texttt{Vol[i]} > 0 \qquad \sum_{\texttt{i}=0}^{\texttt{nbcell}-1} \texttt{Gr[i]} = \texttt{nbgrid}, \tag{6.2}$$

$$\sum_{\texttt{i}=0}^{\texttt{nbcell}-1} \texttt{Vol[i]} = 1, \tag{6.3}$$

$$\texttt{dmin[i]} \to 0 \text{ lorsque } \texttt{nbgrid} \to \infty. \tag{6.4}$$

Les équations (6.1) à (6.4) découlent immédiatement des définitions des variables ci-dessus. L'équation (6.3) signifie que la somme des hypervolumes des cellules doit faire l'hypervolume du domaine B. Comme $B = [0,1]^d$, l'hypervolume de B vaut toujours 1, quelle que soit la dimension. En pratique, vu les erreurs d'arrondi, nous vérifierons plutôt que

$$\left| 1 - \sum_{\texttt{i}=0}^{\texttt{nbcell}-1} \texttt{Vol[i]} \right| < \varepsilon. \tag{6.5}$$

[3]On travaille avec des `double`, dont on est assuré que la précision en chiffres décimaux est supérieure à 10 (cf. les annexes).

grandeur	k = 71	k = 78	k = 100	k = 123	k = 142
nbgrid	5 041	6 084	10 000	15 129	20 164
min Gr	1	1	2	3	3
max Gr	31	38	63	103	126
E[Gr]	10.082	12.168	20.000	30.258	40.328
σ(Gr)	5.224	6.232	10.234	15.526	20.585
min dmin	$3.611\ 10^{-8}$	$1.240\ 10^{-7}$	$1.171\ 10^{-8}$	$9.197\ 10^{-8}$	$1.442\ 10^{-8}$
max dmin	$2.689\ 10^{-4}$	$1.107\ 10^{-4}$	$3.353\ 10^{-4}$	$3.320\ 10^{-5}$	$3.355\ 10^{-5}$
E[dmin]	$3.596\ 10^{-5}$	$2.914\ 10^{-5}$	$1.801\ 10^{-5}$	$1.157\ 10^{-5}$	$8.531\ 10^{-6}$
σ(dmin)	$2.792\ 10^{-5}$	$1.842\ 10^{-5}$	$1.787\ 10^{-5}$	$7.275\ 10^{-6}$	$5.633\ 10^{-6}$

TAB. 6.2: *Quelques résultats vérificatifs pour le programme* dVOR *en 2-D pour 500 cellules (les distances sont bien évidemment toriques)*

Au tableau 6.2, nous avons quelques calculs de vérification pour un ensemble de 500 cellules en deux dimensions ; le nombre seed vaut toujours 781 936, ce qui nous assure que les points générateurs seront toujours les mêmes lors des différents lancements du programme, donc les cellules seront toujours les mêmes. Précisons que ces données ont satisfait aux équations (6.1) à (6.3). Les variables supplémentaires servent à la vérification de l'équation (6.4).

La statitique E[Gr] est utile pour avoir une idée de la pertinence de l'approximation de l'hypervolume des cellules. Nous voyons aussi que la grandeur E[dmin] diminue k augmente : cela conforte l'équation (6.4).

Remarquons encore que

$$E[\mathtt{Gr}] = \frac{1}{\mathtt{nbcell}} \sum_{\mathtt{i}=0}^{\mathtt{nbcell}-1} \mathtt{Gr[i]} = \frac{\mathtt{nbgrid}}{\mathtt{nbcell}}. \tag{6.6}$$

Chapitre 7

Test d'uniformité en d dimensions sur base de cellules de Voronoi

7.1 Introduction

Dans les tests suivants, on a juste calculé l'*hypervolume* des cellules (et pas les distances entre chaque point générateur et son point-grille le plus proche, etc.) : c'est la version « light » du programme `dVOR`. C'est donc sur cette seule grandeur que portent les statistiques suivantes.

Tous les tests qui suivent ont satifait les relations (6.2) et (6.5).

Pour le *QQ plot* $\Gamma(\cdot,\cdot)$, on a utilisé les valeurs de Kumar et Kurtz (1993) en 2-D et celles de Kumar, Kurtz, Banavar et Sharma (1992) en 3-D (cf. table 1.4). Voici la syntaxe des commandes Splus qui fournissent ce graphe (paramètres 3-D) :

```
n <- nrow(results)
vol <- sort(results[,3])
plot(qgamma(ppoints(n), 5.7869, rate = 5.7869*n), vol)
```

où `n` est le nombre de cellules (c.-à-d. `nbcell`, c.-à-d. le nombre de points générateurs), et `results` le tableau des résultats– syntaxe sur une ligne :

le n° de la cellule `Gr[i]` `Vol[i]`

Notons encore que $B = [0,1]^d$ donc la densité du PP homogène est $\lambda = \frac{n}{m(B)} = n$ (c'est le λ de la table 1.4). La fonction

```
qgamma(n, shape, rate)
```

donne les quantiles d'une distribution $\Gamma(\cdot,\cdot)$, où `shape` est le paramètre α (q à la formule (1.17) et à la table 1.4) et `rate` est le paramètre λ (b à la formule (1.17) et à la table 1.4).

Bien sûr, puisque nous travaillons dans un espace torique, nous ne devons supprimer aucune cellule qui serait une cellule-frontière.

Dans ce qui suit, tous les intervalles de confiance (IC) sont réalisés au niveau de signification $\alpha = 0.05$.

Notons enfin que la moyenne de la variable `Gr` est déjà donnée par l'équation 6.6. Nous avons également que

$$E[\texttt{Vol}] = \frac{m(B)}{\texttt{nbcell}} = \frac{1}{\texttt{nbcell}}. \tag{7.1}$$

7.2 Test en 2-D

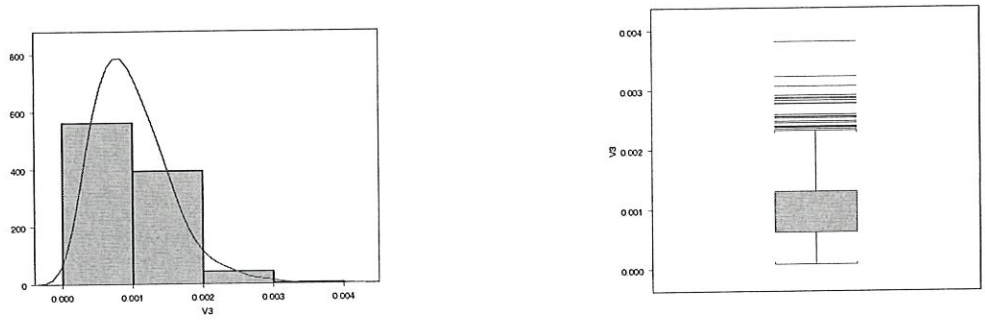


FIG. 7.1: *Histogramme et box plot de l'aire des cellules*

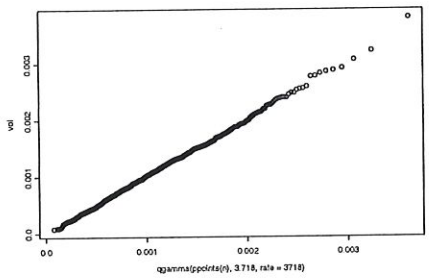


FIG. 7.2: *QQ plot Gamma de l'aire des cellules*

nombre de cellules	1 000
nombre de points-grille	100 489
nombre de points-grille/dimension	317

TAB. 7.1: *Données du test 2-D*

statistiques	`Gr`	`Vol` ($\times 10^{-3}$)
valeur minimale	8	0.079 610 7
premier quartile	62	0.616 983
médiane	92	0.915 523
troisième quartile	131	1.303 63
valeur maximale	383	3.811 36
moyenne	100.489	1.000 00
écart-type	52.597	0.523 414
borne inférieure de l'IC pour la moyenne	97.225	0.967 520
borne supérieure de l'IC pour la moyenne	103.753	1.032 48

TAB. 7.2: *Statistiques élémentaires sur les résultats du test 2-D*

Définition 7.1 (quantile d'ordre ou de niveau p)
Pour une VA *continue X ayant une fonction de répartition strictement croissante, on appelle quantile d'ordre ou de niveau p le nombre x_p qui satisfait la relation*

$$F_X(x_p) = P[X \leq x_p] = p. \tag{7.2}$$

Voici quelques exemples de quantiles :

$x_{0.25}$	premier quartile
$x_{0.3}$	troisième décile
$x_{0.5}$	médiane
$x_{0.75}$	troisième quartile
$x_{0.65}$	soixante-cinquième percentile

La médiane satisfait notamment la relation $P[X \leq x_{0.5}] = P[X \geq x_{0.5}] = 0.5$.

On voit très vite que le QQ plot Gamma et le box plot sont les graphes les plus intéressants. La linéarité du QQ plot Gamma est évidente et montre combien les données sont proches d'une loi $\Gamma(3.178\,\lambda, 3.718)$ (où $\lambda = \frac{m(B)}{n} = \frac{1}{n}$, et n est le nombre de points de l'échantillon, à partir desquels nous générons le diagramme de Voronoi). Le boxplot est également un bon indicateur de la localisation des données (tout de suite nous savons que la fonction de densité n'est pas symétrique – comme c'est le cas pour la loi Gamma), ainsi que le montrent également l'histogramme et la courbe de densité. La courbe de densité a l'allure d'une Gamma.

Le lecteur sera peut-être surpris de ne pas voir ici de tests classiques sur la fonction de fréquence des données, comme les test-χ^2 ou le test de Kolmogorov-Smirnov. C'est que nous n'avons pas réussi à lancer ces tests sous Splus avec la loi Gamma, Splus ne proposant qu'un seul paramètre (`shape1`) pour cette loi, au lieu des deux paramètres habituels.[1]

7.3 Test 3-D

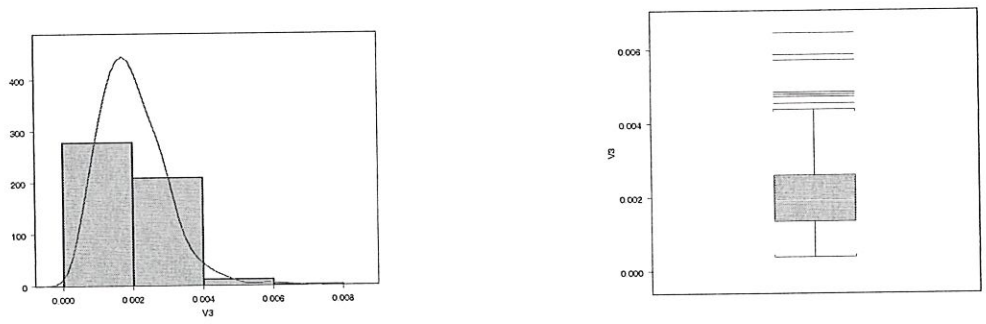


FIG. 7.3: *Histogramme et box plot du volume des cellules*

[1] La syntaxe de Splus définit normalement les deux paramètres `shape` ($= q$) et `rate` ($= b$). On note aussi fréquemment ces paramètres α et λ : la loi $\Gamma(\lambda, \alpha)$, ou $\Gamma(b, q)$. L'espérance est toujours $\frac{\alpha}{\lambda} = \frac{q}{b}$. Notons encore que $\Gamma(\lambda, 1) \equiv Expo(\lambda)$, et $\chi^2_n \equiv \Gamma(\frac{1}{2}, \frac{n}{2})$.

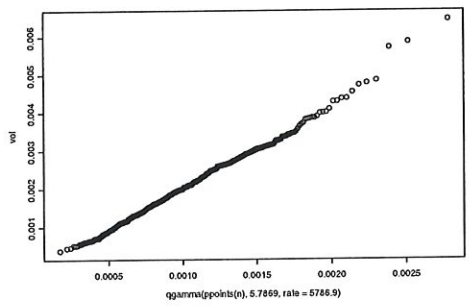


FIG. 7.4: *QQ plot Gamma du volume des cellules*

nombre de cellules	500
nombre de points-grille	103 823
nombre de points-grille/dimension	47
`max_Gr`	669
`min_Gr`	39
`moy_Gr`	207.6

TAB. 7.3: *Données du test 3-D*

statistiques	`Gr`	`Vol` ($\times 10^{-3}$)
valeur minimale	39	0.375 639
premier quartile	139.750	1.346 04
médiane	198	1.907 09
troisième quartile	269	2.590 95
valeur maximale	669	6.443 66
moyenne	207.646	2.000 00
écart-type	93.821	0.903 659
borne inférieure de l'IC pour la moyenne	199.402	1.920 60
borne supérieure de l'IC pour la moyenne	215.890	2.079 40

TAB. 7.4: *Statistiques élémentaires sur les résultats du test 3-D*

À nouveau le QQ plot Gamma est très parlant, la courbe de densité fait songer à une Gamma et le box plot est bien asymétrique. On ne saurait rejeter l'hypothèse que les volumes des cellules suivent en bonne approximation une loi Gamma, à la vue de ces graphes. C'est surtout le QQ plot Gamma qui nous est précieux dans ces tests.

7.4 Test 4-D

Pour les tests de dimension supérieure à trois, nous ne disposons plus d'une théorie sous-jacente ou de recherches empiriques, comme on peut les trouver dans [11]. Pourtant le graphe QQ Gamma nous est précieux. Puisque dans les dimensions un, deux et trois, c'est une loi Gamma qui approxime le mieux la distribution du volume des cellules, nous supposerons par induction que considérer que la distribution de

l'hypervolume des cellules en dimension supérieure suit approximativement une loi Gamma, n'est pas déraisonnable. Mais quels seraient les paramètres d'une telle loi ? Une possibilité serait de chercher les paramètres b et q qui minimiseraient la distance de Kolmogorov (test de Kolmogorov-Smirnov, noté K-S).

Mais nous trouvons également dans la littérature la constatation suivante : le logarithme (népérien) d'une variable aléatoire Gamma est approximativement normal ; et la racine cubique d'une variable aléatoire Gamma est encore une meilleure approximation d'une loi normale (cf. [4, p. 479]). Nous choisirons cette dernière approche. Considérant donc que l'hypervolume des cellules suit approximativement une loi Gamma, nous définissons la variable `zvol` par :

$$\texttt{zvol} = \frac{\sqrt[3]{\texttt{Vol}} - E[\sqrt[3]{\texttt{Vol}}]}{\sigma(\sqrt[3]{\texttt{Vol}})}, \tag{7.3}$$

et nous testerons la normalité centrée réduite de cette variable (nous standardisons $\sqrt[3]{\texttt{Vol}}$ afin de pouvoir comparer cette variable avec une distribution $N(\mathbf{0}, \mathbf{1})$).

Remarquons que nous prenons bien la racine *cubique* de l'hypervolume des cellules, quelle que soit la dimension (et pas la racine d-ième).

Les tests statistiques seront effectués au niveau de signification $\alpha = 0.05$.

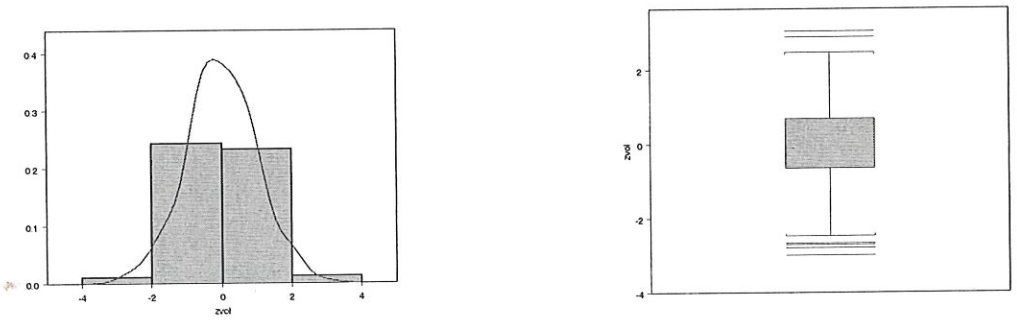


FIG. 7.5: *Histogramme et box plot de l'hypervolume des cellulles (4-D)*

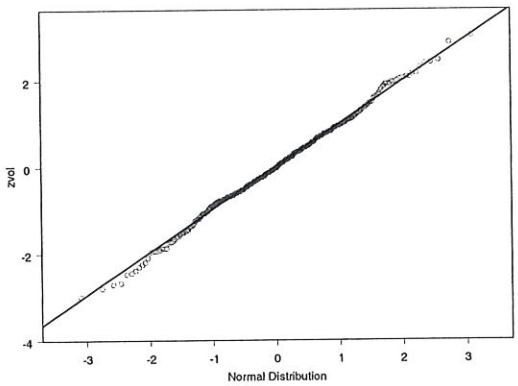


FIG. 7.6: *QQ plot de* `zvol` *(4-D)*

nombre de cellules	500
nombre de points-grille	104 976
nombre de points-grille/dimension	18

TAB. 7.5: *Données pour le test 4-D*

statistiques	`Gr`	`Vol` ($\times 10^{-3}$)
valeur minimale	63	0.600 137
premier quartile	163	1.552 74
médiane	201	1.914 72
troisième quartile	251	2.391 02
valeur maximale	475	4.524 84
moyenne	209.952	2.000 00
écart-type	67.118	0.639 363
borne inférieure de l'IC pour la moyenne	204.055	1.943 82
borne supérieure de l'IC pour la moyenne	215.849	2.056 18

TAB. 7.6: *Statistiques élémentaires sur les résultats du test 4-D*

χ^2	**Kolmogorov-Smirnov**
p-value = 0.719 3	p-value = 0.606 2
$\chi^2_{24} = 19.6$	d = 0.034 1

TAB. 7.7: *Statistiques de tests pour la variable* `zvol` *(4-D)*

On constate que pour le présent test, l'introduction de la variable `zvol` n'est pas dénuée d'intérêt : le QQ plot Gamma « colle » vraiment bien à une distribution $N(0,1)$. La courbe de densité, sans être pleinement régulière, fait cependant songer à une normale sans trop de peine, l'histogramme aussi ; le box plot montre quelques outliers (six, sur cinq cents points) qu'on voyait déjà dans le QQ plot, (on se souviendra de la *règle empirique des 3σ*, qui stipule que 99% des données d'une loi $N(\mu, \sigma)$ se trouvent dans l'intervalle $[\mu - 3\sigma, \mu + 3\sigma]$), mais la boite et les moustaches sont bien symétriques.

Dans les statistiques élémentaires de `zvol` (non réécrites ici), nous avons bien $E[\texttt{zvol}] = 0$ et $var(\texttt{zvol}) = 1$. Mais cela ne nous aide aucunement : il est bien évident que l'on retrouve ces valeurs pour une variable dès qu'on la standardise. Enfin, la distance de Kolmogorov est plutôt faible et le test-χ^2 valide l'hypothèse nulle ($\texttt{zvol} \sim N(0,1)$) pour le seuil de signification $\alpha = 0.05$: $\chi^2_{24,0.05} < 36.4$. Les p-values sont toutes deux supérieures au niveau de signification.

7.5 Test 5-D

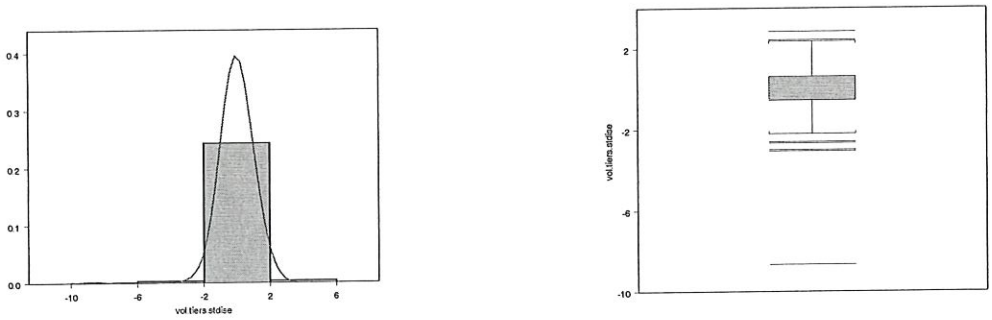


FIG. 7.7: *Histogramme et box plot de l'hypervolume des cellules (5-D)*

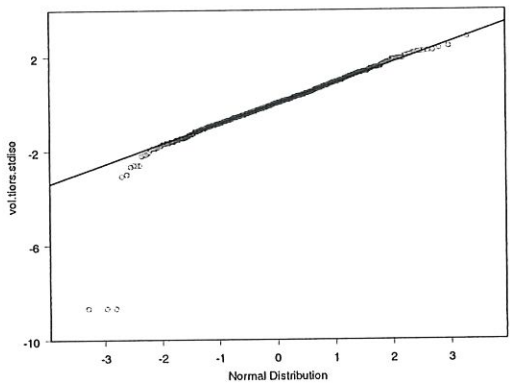


FIG. 7.8: *QQ plot de* `zvol` *(5-D)*

nombre de cellules	1 000
nombre de points-grille	100 000
nombre de points-grille/dimension	10

TAB. 7.8: *Données pour le test 5-D*

χ^2	**Kolmogorov-Smirnov**
p-value = 0	p-value = 0.012 5
$\chi^2_{31} = 76.864$	d = 0.0504

TAB. 7.9: *Statistiques de tests pour la variable* `zvol` *(5-D)*

Nous voyons ici aussi combien la distribution des données (`zvol`) est proche d'une loi $N(0,1)$, à partir du QQ plot Gamma, et, même s'il présente une queue à gauche, le box plot révèle la présence de sept outliers seulement (sur mille points). La boite et les moustaches sont rigoureusment symétriques, la médiane (la petite ligne claire dans la boite) est centrée. La courbe de densité et l'histogramme ont l'allure qu'a une loi normale.

Nous avons dans les tables de la loi χ^2 la valeur $\chi^2_{40,0.05} = 55.8 < 76.864$, et la p-value du test-$\chi^2$ est franchement mauvaise – c'est le moins qu'on puisse dire ! Pourtant la distance de Kolmogorov est assez faible, même si sa p-value est assez petite aussi. L'hypothèse de normalité tient la route, ce sont les quelques outliers qui perturbent ces tests statistiques (on pourrait envisager de relancer ces tests en supprimant par exemple les cinq plus petites valeurs de `zvol`).

7.6 Test 6-D

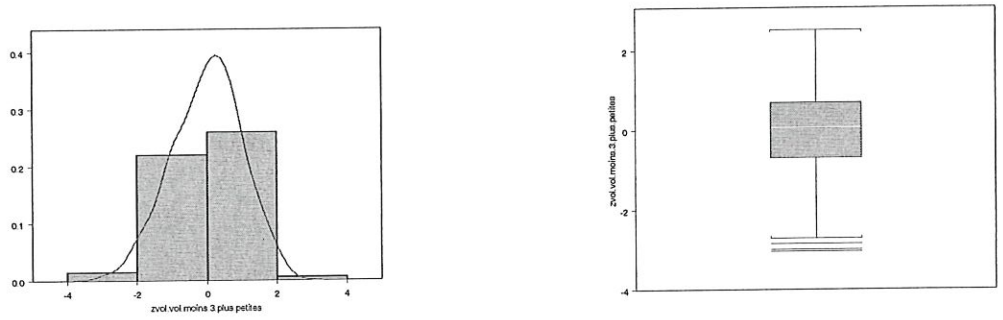


FIG. 7.9: *Histogramme et box plot de l'hypervolume des cellules (6-D)*

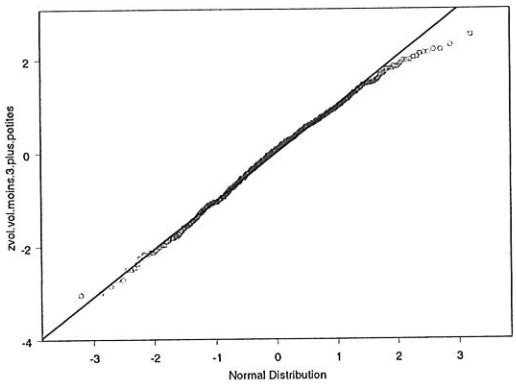


FIG. 7.10: *QQ plot de* `zvol` *(6-D)*

nombre de cellules	750
nombre de points-grille	117 649
nombre de points-grille/dimension	7

TAB. 7.10: *Données pour le test 5-D*

χ^2	**Kolmogorov-Smirnov**
p-value $= 0.062\,5$	p-value $= 0.197\,4$
$\chi^2_{28} = 40.270\,4$	d $= 0.039\,4$

TAB. 7.11: *Statistiques de tests pour la variable* `zvol` *(6-D)*

L'histogramme n'est pas tout à fait symétrique mais s'en approche. La courbe de densité fait tout de même penser à une normale. Le box plot, s'il n'est pas rigoureusement symétrique, ne présente que trois outliers. Enfin, on peut dire que le QQ plot est convaincant en ce qui concerne la normalité de `zvol` (il y a juste une queue à droite non négligeable tout de même). La statistique χ^2 est en-dessous du seuil $\chi^2_{28,0.95} = 37.9$, la distance de Kolmogorov est faible et sa p-value très au-dessus du niveau de signication, celle du test χ^2 restant légèrement supérieure à α également. Bref, on en conclut à la normalité de `zvol`. Il est intéressant de constater ici que même en six dimensions, la variable `zvol` garde un comportement normal. Ceci renforce ici la pertinence du choix de cette variable.

Notons que ce test a pris 171 274 secondes sur la machine B.2, soit 47 heures 34 minutes et 34 secondes.

7.7 Conclusion pour ces tests

Nous disposons ici de deux tests différents : d'une part, le test d'une loi $\Gamma(\cdot,\cdot)$ pour les échantillons en 2-D et 3-D ; d'autre part, le test d'une loi $N(0,1)$ pour les échantillons (variable `zvol`) en dimensions supérieures.

Les tests uniformes 2-D et 3-D sont concluants. Ils confirment la pertinence de l'approximation de l'aire ou du volume des cellules par une distribution Gamma, comme développé dans [11]. Ce sont les graphes QQ Gamma qui nous aident le plus à valider cette hypothèse.

Les tests 4-D et 5-D sont assez intéressants également, on peut dire que le choix de cette variable `zvol` n'est pas sans intérêt. Il est d'ailleurs pertinent de se rappeler que l'hypothèse sous-jacente à ces tests est que l'hypervolume des cellules suit également une loi Gamma. Néanmoins, gardons à l'esprit que nous utilisons là une démarche inductive (même si cela est très courant en statistiques).

7.8 Analyse de la précision des résultats

Puisque le programme `dVOR` est assez gourmand (complexité exponentielle), nous tentons ici d'analyser la précision des résultats pour différentes valeur de `nbgrid` (nombre total de points-grille). Le fait que le générateur de nombres aléatoires de Fabian Bastin fournisse toujours les mêmes nombres pour un même entier `seed`, rend cette tâche très aisée : nous sommes sûrs qu'il s'agit toujours bien du même échantillon. Les *résultats* analysés sont bien sûr ici l'hypervolume des cellules (*y compris* pour le test en quatre dimensions) ; nous ne nous soucions pas de la variable `Gr`, qui est de toutes façons proportionnelle à `Vol` : `Vol[i]` = `Gr[i]/nbgrid`).

Rappelons juste que la *précision* du quadrillage est définie par $Pr = \frac{1}{\texttt{nbgrid}}$; et donc, plus le quadrillage est fin, plus la précision est proche de 0.

7.8.1 Test 2-D

caractéristiques du test	a	b	c	d	e
`nbgrid`	10 000	20 164	40 000	70 225	100 489
`k`	100	142	200	265	317
précision ($\times 10^{-4}$)	1	0.495 93	0.25	0.142 40	0.099 513

TAB. 7.12: *Données pour la précision des résultats (test 2-D)*

statistiques	a ($\times 10^{-3}$)	b ($\times 10^{-3}$)	c ($\times 10^{-3}$)	d ($\times 10^{-3}$)	e ($\times 10^{-3}$)
valeur minimale	0	0.049 593 3	0.050 000	0.071 199 7	0.079 610 7
premier quartile	0.600 000	0.595 120	0.625 000	0.612 318	0.616 983
médiane	0.900 000	0.942 273	0.925 000	0.925 596	0.915 523
troisième quartile	1.300 00	1.289 43	1.300 00	1.295 84	1.303 63
valeur maximale	3.700 00	3.769 09	3.825 00	3.830 55	3.811 36
moyenne	1.000 00	1.000 00	1.000 00	1.000 00	1.000 00
écart-type	0.532 507	0.526 091	0.524 091	0.523 216	0.523 414
b. inf. IC moyenne	0.966 956	0.967 354	0.967 478	0.967 532	0.967 520
b. sup. IC moyenne	1.033 05	1.032 65	1.032 52	1.032 47	1.032 48

TAB. 7.13: *Comparaisons sur la précision des résultats du test 2-D*

Notons que le test a ne vérifie pas la condition 6.2 : $\exists$ `i` tq `Vol[i]` $= 0$.

Analysons la table 7.13. Bien sûr, la moyenne des aires des cellules vaut toujours $\frac{m(B)}{\texttt{nbcell}} = \frac{1}{\texttt{nbcell}}$ – au moins, elle tombe toujours juste dans les résultats ! L'écart-type des aires diminue en général lorque `nbgrid` augmente, mais peu. L'idéal serait de faire un test avec un nombre très élevé de points-grille (p. ex. dix ou cent millions) et de le comparer à ceux-ci. Mais cela nécessite déjà une certaine puissance de calcul. Précisons encore que la différence entre les aires de ces trois tests est toujours de l'ordre de 10^{-5} ou en-deçà.

7.8.2 Test 3-D

caractéristiques du test	a	b	c
nombre de points-grille	27 000	50 653	103 823
nombre de points-grille/dimension	30	37	47
précision ($\times 10^{-5}$)	3.703 70	1.974 22	0.963 178

TAB. 7.14: *Données pour la précision du test 3-D*

statistiques	a ($\times 10^{-3}$)	b ($\times 10^{-3}$)	c ($\times 10^{-3}$)
valeur minimale	0.0370 374	0.335 617	0.375 639
premier quartile	1.333 33	1.357 27	1.346 04
médiane	1.888 89	1.895 25	1.907 09
troisième quartile	2.527 78	2.586 22	2.590 95
valeur maximale	6.444 44	6.416 20	6.443 66
moyenne	2.000 00	2.000 00	2.000 00
écart-type	0.905 250	0.901 027	0.903 659
borne inférieure de l'IC pour la moyenne	1.920 46	1.920 83	1.920 60
borne supérieure de l'IC pour la moyenne	2.079 54	2.079 17	2.079 40

TAB. 7.15: *Statistiques élémentaires sur les résultats du test 3-D*

Les résultats (table 7.15) sont ici plus délicats à analyser. D'une part parce que nous disposons seulement de trois tests, et d'autre part parce que le test c semble moins précis que le test b. Là aussi, la différence entre les volumes des différents tests est de l'ordre de 10^{-5}.

Notons que les statistiques utilisées ici (moyenne, écart-type,...) sont toujours des *estimateurs*, donc *asymptotiquement* exacts. Or, nous analysons ces statistiques sur base de 500 ou 1 000 observations, ce qui est très peu. Il faut donc tenir compte de ce fait ! De plus, nous ne savons pas quelle est la précision des calculs en eux-mêmes : Splus utilise-t-il des réels en simple ou double précision, pour calculer ces statistiques ? De manière réaliste, on peut se douter que le nombre de chiffres significatifs dans les résultats n'est pas très élevé.

7.8.3 Test 4-D

caractéristiques du test	a	b
nombre de points-grille	28 561	104 976
nombre de points-grille/dimension	13	18
précision ($\times 10^{-5}$)	3.501 28	0.952 599

TAB. 7.16: *Données pour la précision du test 4-D*

statistiques	a ($\times 10^{-3}$)	b ($\times 10^{-3}$)
valeur minimale	0.455 166	0.600 137
premier quartile	1.540 56	1.552 74
médiane	1.960 72	1.914 72
troisième quartile	2.380 87	2.391 02
valeur maximale	4.691 71	4.524 84
moyenne	2.000 00	2.000 00
écart-type	0.651 259	0.639 363
b. inf. IC moyenne	1.942 78	1.943 82
b. sup. IC moyenne	2.057 22	2.056 18

TAB. 7.17: *Statistiques élémentaires sur les résultats du test 4-D*

Dans ce dernier test, nécessairement incomplet[2], il est assez frappant de constater que l'augmentation de précision n'est pas très significative par rapport au nombre de points-grille supplémentaire « engagés ».

7.8.4 Conclusion

Cette analyse n'est pas dépourvue d'intérêt. *A priori*, un nombre assez restreint de points-grille serait déjà suffisant pour obtenir des résultats satisfaisants ou, dit autrement, l'augmentation de points-grille n'est pas très payante pour une amélioration de la précision. Mais convenons qu'il s'agit ici de tests encore assez sommaires. Il faudrait, pour se faire une idée plus précise, lancer des tests sur un nombre de cellules bien plus grand (pour diminuer les erreurs des estimateurs), et également réaliser plus de tests pour un même échantillon : peut-être verrait-on dans ce cas apparaitre des oscillations dans les résultats, là où nous croyions gagner systématiquement en précision. Enfin, disposer pour chaque échantillon d'un test (en principe) *très* précis, en entrant un nombre très grand de points-grille, pourrait se révéler intéressant pour la comparaison.

[2]Le temps d'exécution du programme augmentant exponentiellement avec la dimension...

Chapitre 8

Test de non-uniformité en d dimensions sur base de cellules de Voronoi

8.1 Tests 2-D

8.1.1 Caractéristiques des tests

Précisons tout d'abord comment sont générés ces échantillons non-uniformes : le programme génère des nombres aléatoires uniformément distribués dans $B = [0,1]^d$, puis supprime certains de ces nombres de manière à avoir un échantillon en forme d'hypersphère, de trou hypersphérique, etc. L'utilisateur entre le nombre de cellules qu'il désire[1], et le programme, avant de lancer tous les calculs de caractéristiques des cellules, précise combien il restera de cellules après modification de l'échantillon.

Voici quelques précisions sur les différents types d'échantillons non-uniformes présentés ici (tous ces échantillons sont toujours inclus à B, et quand nous disons d'un échantillon qu'il est « centré », c'est par rapport à B bien sûr) :

type d'échantillon	**description**
sphère	échantillon en forme d'hypersphère centrée
trou sphérique	échantillon avec un trou hypersphérique centré
couronne	échantillon en forme de « couronne » centrée
trou-couronne	échantillon avec un trou en forme de couronne centré
deux sphères	échantillon composé de deux hypersphères (dans les tests, elles seront toujours disjointes)
deux trous sphériques	échantillon avec deux trous hypersphériques, pas nécessairement di
deux bandes	échantillon jointes

Dans les tests, sont aussi donnés
tillons. On comprendra rapidement q
sphérique, r_{int} et r_{ext} les rayons int
couronne, c_i et r_i les coordonnées du
et h_i les coordonnée du milieu de la

[1] Ces cellules seront construites à part
nombres aléatoires sont les *points générat*

Pourquoi pas de PPN ici mais qui remplissent tout l'espace analysé ?

Concernant les échantillons « couronne » : en deux dimensions, il s'agit bien d'une couronne ; en trois dimensions, l'échantillon est une sphère centrée dont on a soustrait une autre sphère de même centre plus petite ; pour les dimensions supérieures, on gardera le terme « couronne ».

Concernant les échantillons « deux bandes » : en deux dimensions, il s'agit bien de deux bandes horizontales (disjointes) ; en trois dimensions, l'échantillon est formé de deux parallélépipèdes rectangles ; pour les dimensions supérieures, on applique le même procédé – c.-à-d. qu'on ne garde que les points $p = (p_1, p_2, \ldots, p_d)$ tq $|p_d - m_1| \leq h_1$ et $|p_d - m_2| \leq h_2$.

Remarquons qu'au départ, les points sont générés uniformément. Ainsi, dans le cas des bandes horizontales par exemple, l'échantillon en lui-même (l'ensemble de tous points) n'est plus uniforme, mais dans chacune des bandes, les points restent uniformément distribués. Ainsi, le test sur base des cellules de Voronoi ne devrait pas en conclure que l'échantillon est uniforme. Mais si on retire les plus grandes cellules – celles situées entre les deux bandes –, il ne restera que les cellules à l'intérieur des bandes : autrement dit, c'est comme si l'on avait construit le diagramme de Voronoi à partir des deux bandes rejointes, « recollées », ce qui donne un échantillon uniforme. Donc, si on enlève les plus grandes cellule, le test d'uniformité devrait être satisfait. C'était déjà la méthode qu'utilisait Vandooren (voir [18]).

Voici la définition de ces caractéristiques détaillées dans les tables suivantes :

`nbcell0`	nombre initial de cellules entré par l'utilisateur
`nbcell`	nombre de cellules après modification de l'échantillon
`nbgrid`	nombre de points-grille total (recalculé : cf. section 6.3)
`k`	nombre de points-grille par dimension (toujours entier)
`machine`	référence de la table en annexe où est décrite la machine qui a servi pour ce test
`RNG`	*Random Generator Number* : type de générateur de nombres aléatoires (« É » pour L'Écuyer et « P & M » pour Park & Miller)

échantillon	`nbcell0`	`nbcell`	`nbgrid`	`k`	**machine**	**RNG**
sphère	1 000	545	50 176	224	B.3	É
trou sphérique	1 000	796	50 176	224	B.3	É
couronne	1 000	471	50 176	224	B.3	É
trou-couronne	1 000	656	50 176	224	B.3	É
deux sphères	3 000	784	50 176	224	B.1	É
deux trous sphériques	1 000	754	50 176	224	B.1	É
deux bandes	1 000	769	50 176	224	B.3	É

TAB. 8.1: *Données pour les tests 2-D non uniformes*

échantillon	paramètres
sphère	$r = 0.4$
trou sphérique	$r = 0.25$
couronne	$r_{int} = 0.1,\ r_{ext} = 0.4$
trou-couronne	$r_{int} = 0.15,\ r_{ext} = 0.35$
deux sphères	$c_1 = (0.25, 0.25),\ c_2 = (0.75, 0.75),\ r_1 = r_2 = 0.2$
deux trous sphériques	$c_1 = (0.25, 0.25),\ c_2 = (0.75, 0.75),\ r_1 = r_2 = 0.2$
deux bandes	$m_1 = 0.2,\ m_2 = 0.8,\ h_1 = h_2 = 0.2$

TAB. 8.2: *Paramètres des échantillons pour les tests 2-D non uniformes*

Nous mettons ici le *scatter plot* dans un souci d'offrir une visualisation immédiate du type de données.[2] Le scatter plot des aires des cellules est déjà intéressant pour montrer que la plupart de ces grandeurs sont confinées dans un petit intervalle de valeurs. L'histogramme confirme ici directement cette impression (difficile de faire mieux), la courbe de densité est très raide et très étroite, trop pour une Gamma. Le box plot est lui aussi trop marqué, il y a beaucoup trop d'outliers, la boite est trop fine et les moustaches trop rapprochées, même pour une Gamma. Enfin, le QQ plot Gamma est ici encore le graphe le plus parlant, et est sans équivoque : on ne peut en aucune manière accepter l'hypothèse nulle (à savoir que les données sont uniformes, ce qui implique pour les aires des cellules une distribution très proche de la loi Gamma – pour rappel).

8.1.2 Échantillon 2-D en forme de disque centré

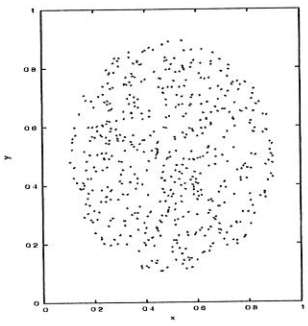


FIG. 8.1: *Scatter plot des points générateurs des cellules*

[2]Même si le lecteur a largement la capacité d'imaginer par lui-même ces données, c'est toujours agréable d'avoir directement sous les yeux un petit dessin les représentant.

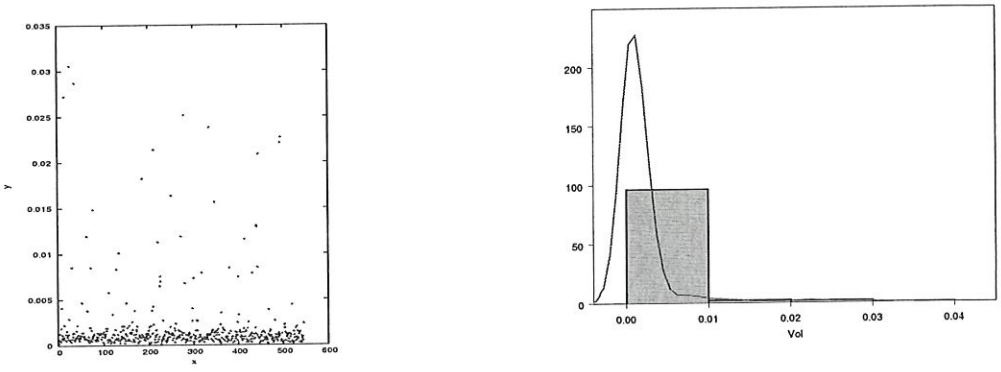


FIG. 8.2: *Scatter plot et histogramme et courbe de densité de l'aire des cellules*

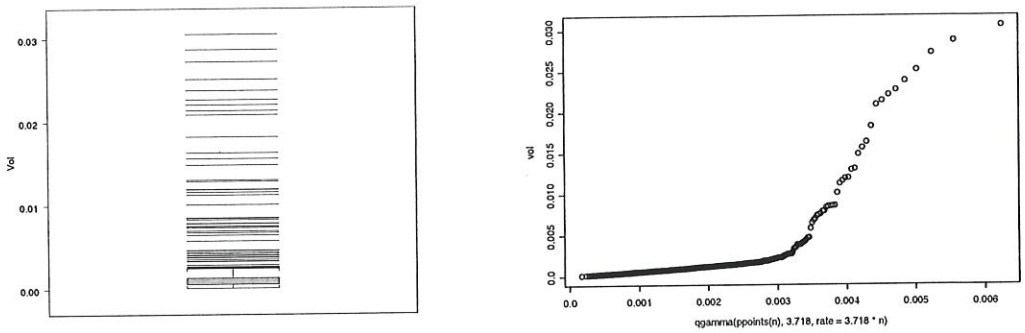


FIG. 8.3: *Box plot et QQ plot Gamma de l'aire des cellules*

statistiques	Gr	Vol ($\times 10^{-3}$)
valeur minimale	6	0.119 579
premier quartile	30	0.597 895
médiane	46	0.916 773
troisième quartile	68	1.355 23
valeur maximale	1 533	30.552 5
moyenne	92.067	1.834 86
écart-type	185.072	3.688 45
borne inférieure de l'IC pour la moyenne	76.494	1.524 51
borne supérieure de l'IC pour la moyenne	107.639	2.145 22

TAB. 8.3: *Statistiques élémentaires sur les résultats du test*

Ici aussi, le scatter plot des valeurs volumes des cellules montre déjà qu'elles sont presque toutes confinées dans un petit intervalle. L'histogramme ressemble au précédent, la fonction de densité aussi. Le box plot est à nouveau trop « extrême » et le QQ plot sans appel.

Reconnaissons s'il est quelque peu embarrassant de ne pas disposer de statistiques sur la fonction de fréquence des données (test-χ^2 et K-S). Toutefois, ce ne sont cependant pas ces données-ci qui nous troubleront à propos de la pertinence de

l'hypothèse nulle. Les graphes suffisent ici amplement à la rejeter, comme dans le cas précédent.

8.1.3 Échantillon 2-D avec un trou en forme de disque centré

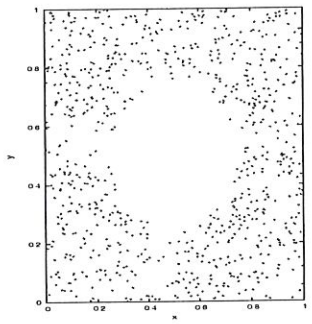


FIG. 8.4: *Scatter plot des points générateurs des cellules*

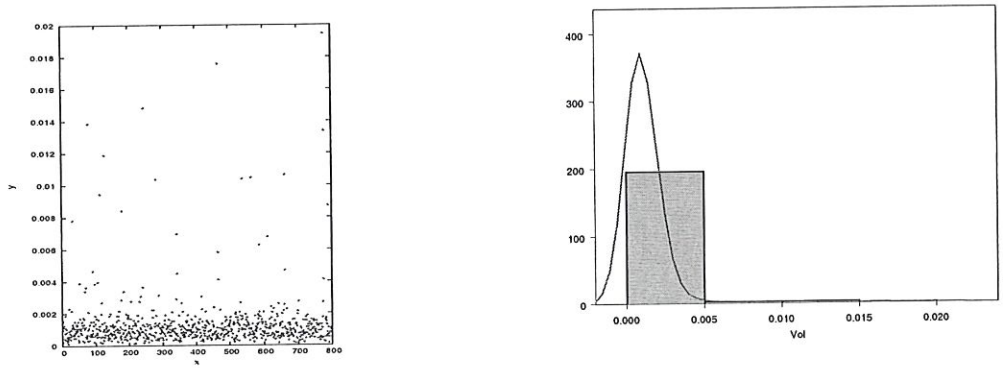


FIG. 8.5: *Scatter plot et histogramme et courbe de densité de l'aire des cellules*

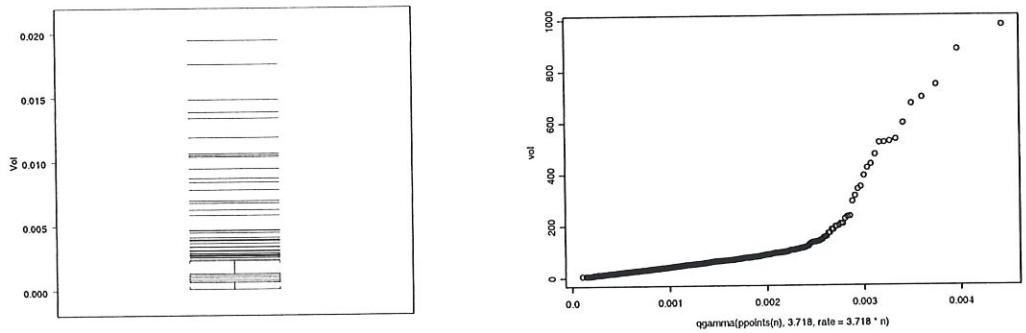


FIG. 8.6: *Box plot et QQ plot Gamma de l'aire des cellules*

statistiques	Gr	Vol ($\times 10^{-3}$)
valeur minimale	6	0.119 579
premier quartile	31	0.617 825
médiane	47	0.936 703
troisième quartile	66	1.315 37
valeur maximale	976	19.451 5
moyenne	63.035	1.256 28
écart-type	83.893	1.671 965
borne inférieure de l'IC pour la moyenne	57.198	1.139 95
borne supérieure de l'IC pour la moyenne	68.872	1.372 61

TAB. 8.4: *Statistiques élémentaires sur les résultats du test*

Les graphes les plus significatifs restent le box plot et le QQ plot Gamma. Ces graphes ressemblent aux précédents, nous ne pensons donc pas surprendre le lecteur si nous délarons rejeter l'hypothèse nulle.

8.1.4 Échantillon 2-D avec une couronne centrée

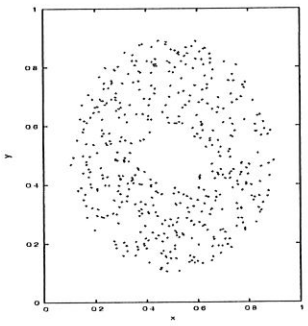


FIG. 8.7: *Scatter plot des points générateurs des cellules*

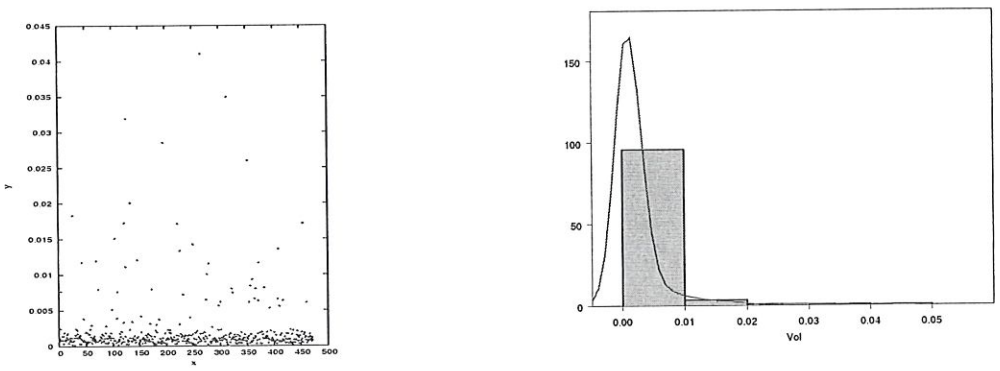


FIG. 8.8: *Scatter plot et histogramme et courbe de densité de l'aire des cellules*

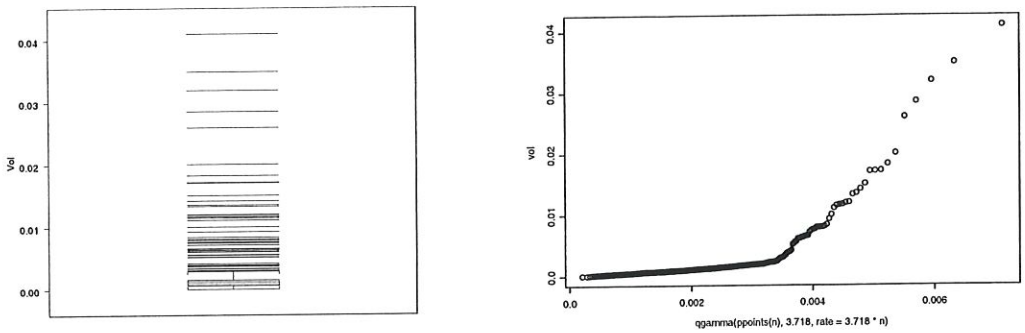


FIG. 8.9: *Box plot et QQ plot Gamma de l'aire des cellules*

statistiques	Gr	Vol ($\times 10^{-3}$)
valeur minimale	3	0.059 789 5
premier quartile	33	0.657 685
médiane	50	0.996 492
troisième quartile	78.5	1.564 49
valeur maximale	2 058	41.015 6
moyenne	106.531	2.123 14
écart-type	212.368	4.232 46
borne inférieure de l'IC pour la moyenne	87.302	1.739 92
borne supérieure de l'IC pour la moyenne	125.759	2.506 36

TAB. 8.5: *Statistiques élémentaires sur les résultats du test*

Ce test-ci ne pose pas de problèmes non plus.

8.1.5 Échantillon 2-D avec un trou central en forme de couronne

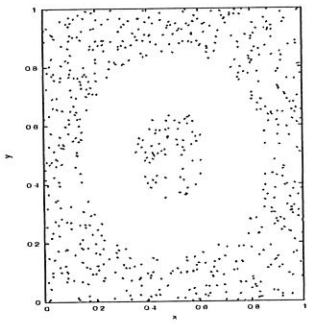


FIG. 8.10: *Scatter plot des points générateurs des cellules*

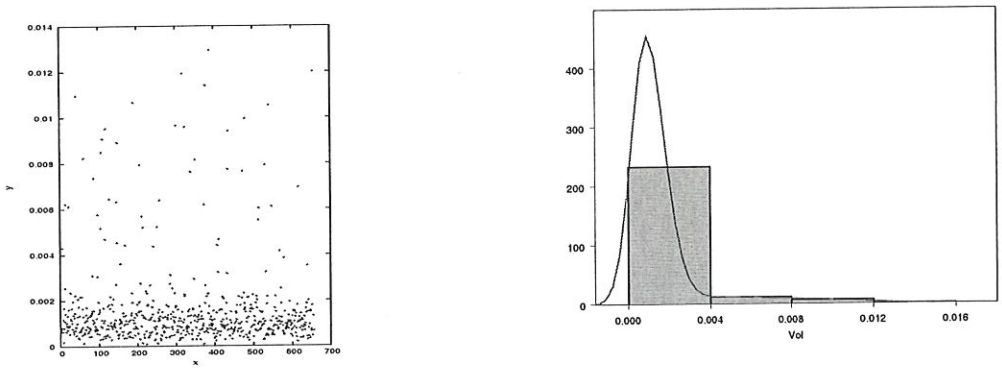


FIG. 8.11: *Scatter plot et histogramme et courbe de densité de l'aire des cellules*

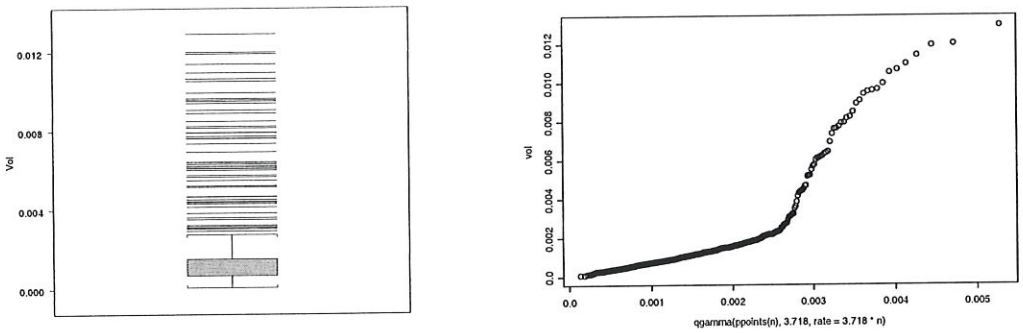


FIG. 8.12: *Box plot et QQ plot Gamma de l'aire des cellules*

statistiques	Gr	Vol ($\times 10^{-3}$)
valeur minimale	4	0.079 719
premier quartile	34	0.677 615
médiane	51	1.016 42
troisième quartile	76	1.514 67
valeur maximale	649	12.934 5
moyenne	76.488	1.524 39
écart-type	92.128	1.836 10
borne inférieure de l'IC pour la moyenne	69.425	1.383 63
borne supérieure de l'IC pour la moyenne	83.551	1.665 16

TAB. 8.6: *Statistiques élémentaires sur les résultats du test*

8.1.6 Échantillon 2-D formé de deux disques disjoints

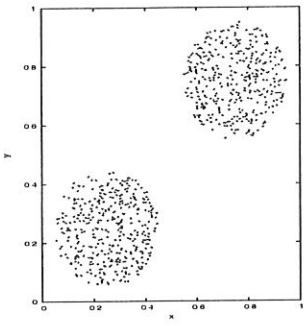


FIG. 8.13: *Scatter plot des points générateurs des cellules*

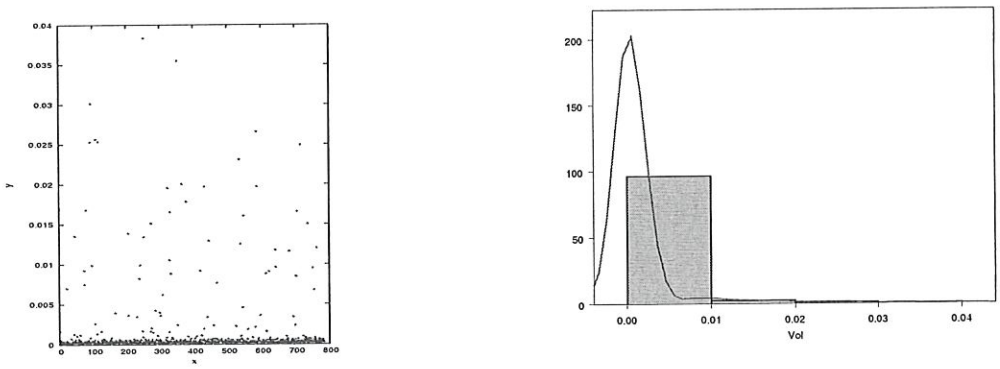


FIG. 8.14: *Scatter plot et histogramme et courbe de densité de l'aire des cellules*

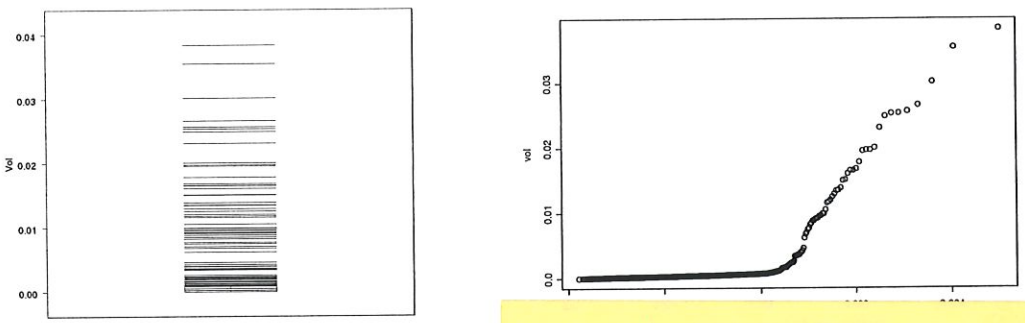


FIG. 8.15: *Box plot et QQ pl*

peut servir de test sur le # de clusters !

statistiques	Gr	Vol ($\times 10^{-3}$)
valeur minimale	0	0
premier quartile	10.750	0.214 246
médiane	16	0.318 878
troisième quartile	24	0.478 316
valeur maximale	1 924	38.345 0
moyenne	64	1.275 51
écart-type	200.246	3.990 88
borne inférieure de l'IC pour la moyenne	49.961	0.995 721
borne supérieure de l'IC pour la moyenne	78.039	1.555 30

TAB. 8.7: *Statistiques élémentaires sur les résultats du test*

8.1.7 Échantillon 2-D formé de deux trous en forme de disques disjoints

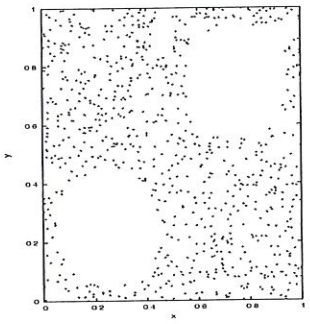


FIG. 8.16: *Scatter plot des points générateurs des cellules*

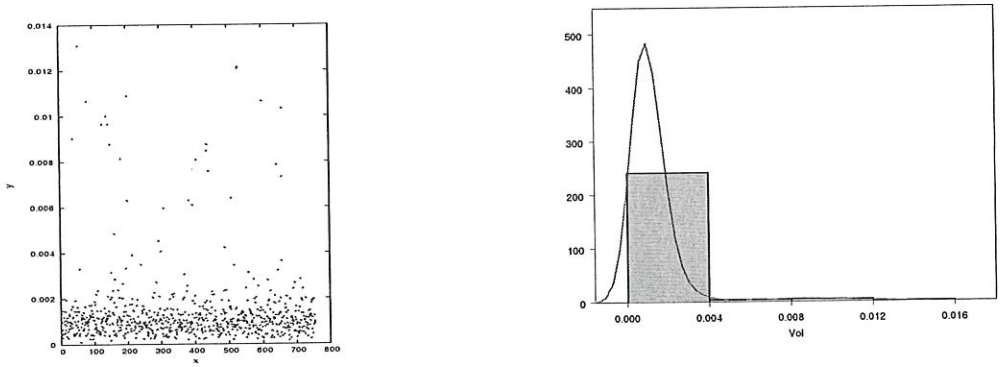


FIG. 8.17: *Scatter plot et histogramme et courbe de densité de l'aire des cellules*

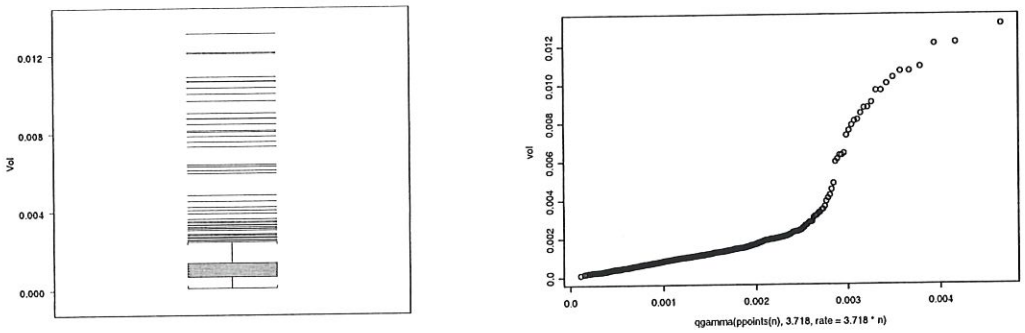


FIG. 8.18: *Box plot et QQ plot Gamma de l'aire des cellules*

statistiques	Gr	Vol ($\times 10^{-3}$)
valeur minimale	5	0.099 649 2
premier quartile	33	0.657 685
médiane	50	0.996 492
troisième quartile	69	1.375 16
valeur maximale	657	13.093 9
moyenne	66.546	1.326 26
écart-type	77.917	1.552 87
borne inférieure de l'IC pour la moyenne	60.976	1.215 24
borne supérieure de l'IC pour la moyenne	72.117	1.437 28

TAB. 8.8: *Statistiques élémentaires sur les résultats du test*

8.1.8 Échantillon 2-D formé de deux bandes horizontales disjointes

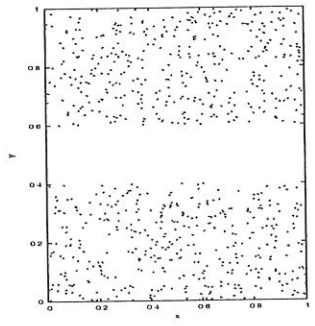


FIG. 8.19: *Scatter plot des points générateurs des cellules*

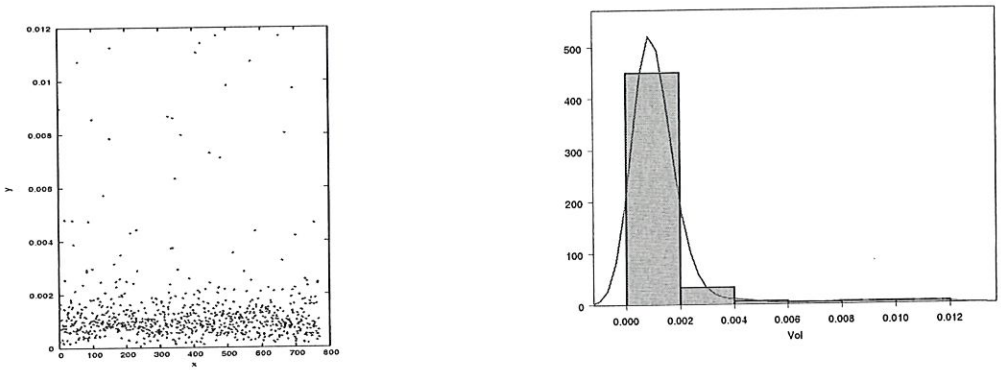


FIG. 8.20: *Scatter plot et histogramme et courbe de densité de l'aire des cellules*

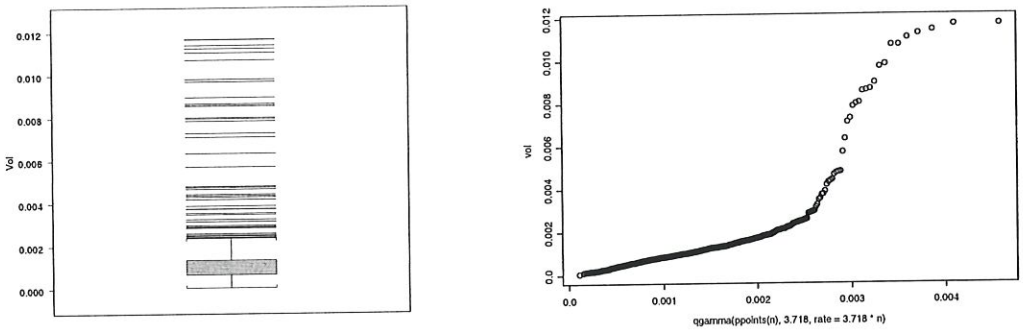


FIG. 8.21: *Box plot et QQ plot Gamma de l'aire des cellules*

statistiques	Gr	Vol ($\times 10^{-3}$)
valeur minimale	3	0.059 789 5
premier quartile	34	0.677 615
médiane	48	0.956 633
troisième quartile	68	1.355 23
valeur maximale	587	11.698 8
moyenne	65.248	1.300 39
écart-type	74.134	1.477 49
borne inférieure de l'IC pour la moyenne	60.000	1.195 80
borne supérieure de l'IC pour la moyenne	70.496	1.404 98

TAB. 8.9: *Statistiques élémentaires sur les résultats du test*

8.1.9 Conclusions pour les tests 2-D

Nous avons analysé sept façons de modifier un échantillon uniforme. Aucune de ces modifications n'a troublé notre test, et les graphes parlent d'eux-mêmes ; les tests sont satisfaisants.

Cependant, ajoutons une limitation : c'est que notre test ne permet ici aucunement de faire la différence entre ces échantillons, de pouvoir analyser leur forme. On constate en effet que tous les graphes se ressemblent.

8.2 Test 3-D

8.2.1 Caractéristiques des tests

échantillon	nbcell0	nbcell	nbgrid	k	machine	RNG
sphère	3 000	810	50 653	37	?	É
trou sphérique	1 000	738	50 653	37	B.3	É
couronne	3 000	773	50 653	37	B.2	É
trou-couronne	1 000	751	50 653	37	B.3	É
deux sphères	8 000	774	50 653	37	B.3	P & M
deux trous sphériques	1 000	789	50 653	37	B.3	P & M
deux bandes	1 000	787	50 653	37	B.3	É

TAB. 8.10: *Données pour les tests 3-D non uniformes*

échantillon	paramètres
sphère	$r = 0.4$
trou sphérique	$r = 0.4$
couronne	$r_{int} = 0.15,\ r_{ext} = 0.4$
trou-couronne	$r_{int} = 0.2,\ r_{ext} = 0.4$
deux sphères	$c_1 = (0.25, 0.25, 0.25),\ c_2 = (0.75, 0.75, 0.75),\ r_1 = r_2 = 0.225$
deux trous sphériques	$c_1 = (0.35, 0.35, 0.35),\ c_2 = (0.65, 0.65, 0.65),\ r_1 = r_2 = 0.3$
deux bandes	$m_1 = 0.2,\ m_2 = 0.8,\ h_1 = h_2 = 0.2$

TAB. 8.11: *Paramètres des échantillons pour les tests 3-D non uniformes*

8.2.2 Échantillon 3-D en forme de sphère centrée

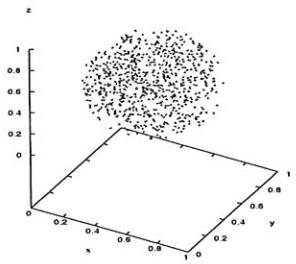


FIG. 8.22: *Scatter plot des points générateurs des cellules*

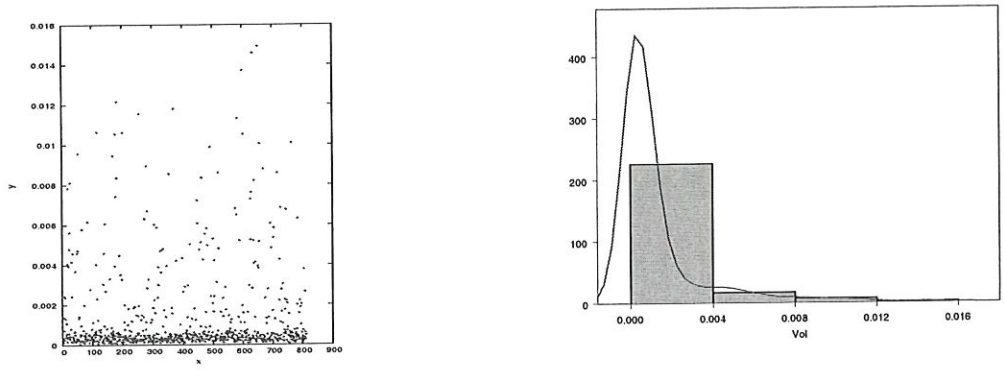


FIG. 8.23: *Scatter plot et histogramme et courbe de densité du volume des cellules*

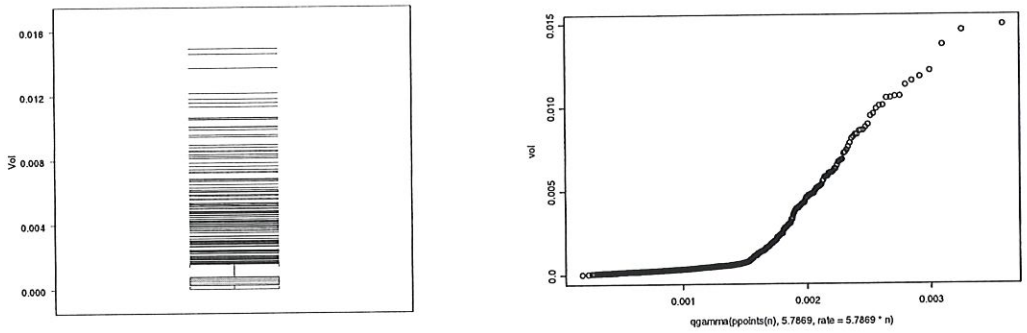


FIG. 8.24: *Box plot et QQ plot Gamma du volume des cellules*

statistiques	Gr	Vol ($\times 10^{-3}$)
valeur minimale	1	0.019 742 2
premier quartile	13	0.256 648
médiane	21	0.414 586
troisième quartile	39	0.769 945
valeur maximale	756	14.925 1
moyenne	62.535	1.234 57
écart-type	109.883	2.169 33
borne inférieure de l'IC pour la moyenne	54.956	1.084 95
borne supérieure de l'IC pour la moyenne	70.113	1.384 19

TAB. 8.12: *Statistiques élémentaires sur les résultats du test*

Les graphes de ce test sont similaires à ceux effectués en deux dimensions. Par conséquent, ce test ne pose pas de souci particulier.

Remarquons qu'il a fallu générer 3 000 cellules pour n'en garder que 810 (table 8.10). C'est que $r < 0.5$ et le volume d'une sphère est proportionnel à r^3, donc la sphère est très petite par rapport au cube $[0,1]^3$. Le rapport de leurs volumes vaut ici 26.81% (alors que pour $r = 0.5$, la sphère est déjà inscrite au cube). Le rapport $\frac{810}{3000}$ vaut 27%, ce qui n'est pas étonnant pour des données uniformes (c.-à-d. réparties partout avec la même probabilité).

8.2.3 Échantillon 3-D avec un trou en forme de sphère centrée

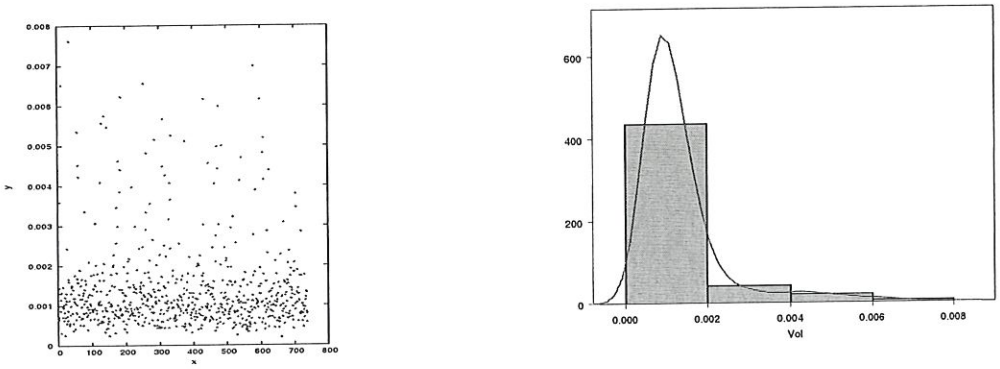


FIG. 8.25: *Scatter plot et histogramme et courbe de densité du volume des cellules*

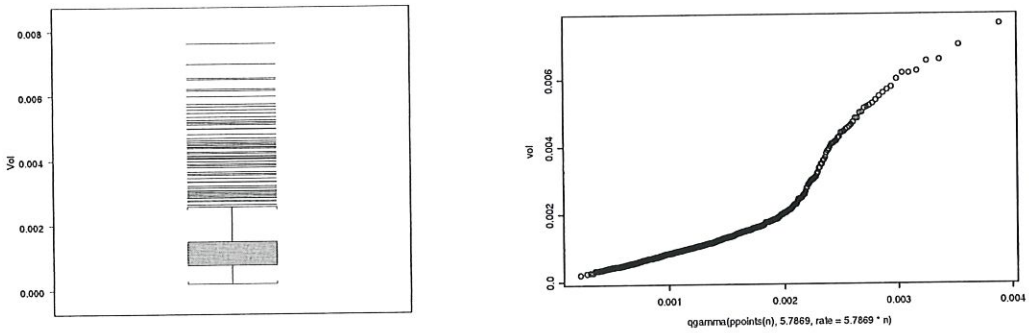


FIG. 8.26: *Box plot et QQ plot Gamma du volume des cellules*

statistiques	Gr	Vol ($\times 10^{-3}$)
valeur minimale	10	0.197 422
premier quartile	39	0.769 945
médiane	53	1.046 34
troisième quartile	76	1.500 41
valeur maximale	386	7.620 48
moyenne	68.636	1.655 01
écart-type	54.085	1.067 76
borne inférieure de l'IC pour la moyenne	64.727	1.277 85
borne supérieure de l'IC pour la moyenne	72.544	1.432 18

TAB. 8.13: *Statistiques élémentaires sur les résultats du test*

Nous n'avons pas mis le *scatter plot* des points générateurs car il n'a pas d'intérêt ici : on voit juste un échantillon de points en 3-D, mais rien ne permet de voir qu'il y a un trou au centre, à cause des points qui sont devant et derrière ce trou, dans l'angle de vue. Ainsi, si nous pouvions directement rejeter l'hypothèse de l'uniformité en deux dimensions, rien que par un brep coup d'œil sur le *scatter plot* des points

générateurs, cela devient déjà plus ardu en trois dimensions. D'où l'intérêt de ce test, évidemment.

Et nous pouvons sans remords rejeter l'hypothèse d'uniformité à la vue de ces graphes. Les scatter plot des volumes est plus disparate que ceux des tests précédents. Et le box plot compte sans doute moins d'outliers. Le QQ plot confirme cela, mais nous sommes toutefois loins de remettre en cause le rejet de l'hypothèse nulle.

8.2.4 Échantillon 3-D avec une couronne centrée

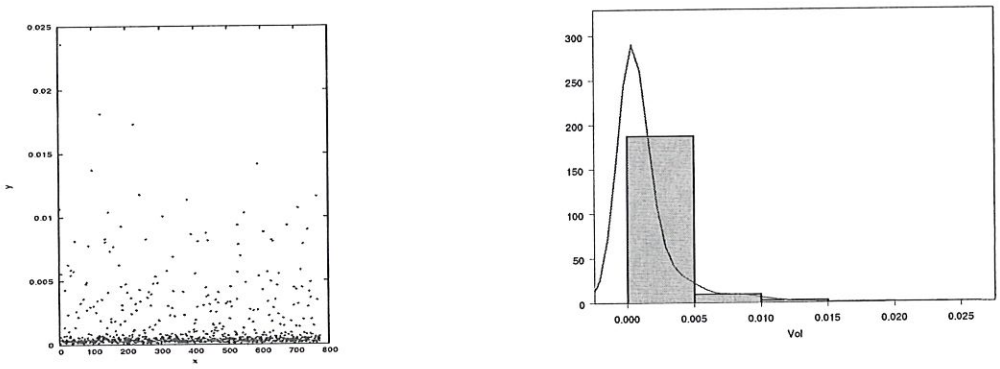


FIG. 8.27: *Scatter plot et histogramme et courbe de densité du volume des cellules*

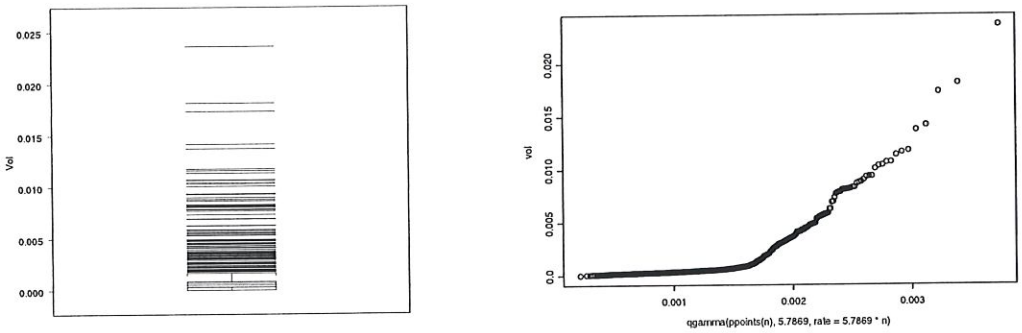


FIG. 8.28: *Box plot et QQ plot Gamma du volume des cellules*

statistiques	Gr	Vol ($\times 10^{-3}$)
valeur minimale	0	0
premier quartile	13	0.256 648
médiane	21	0.414 586
troisième quartile	42	0.829 171
valeur maximale	1 196	23.611 6
moyenne	65.528	1.293 66
écart-type	120.876	2.386 36
borne inférieure de l'IC pour la moyenne	56.993	1.125 17
borne supérieure de l'IC pour la moyenne	74.062	1.462 15

TAB. 8.14: *Statistiques élémentaires sur les résultats du test*

Ici aussi, il ne subsiste que 773 cellules sur 3 000, après modification de l'échantillon. Le rapport des volumes de la couronne et du cube est de 25.39%, et celui du nombre restant de cellules (c.-à-d. de points générateurs) par rapport au nombre initial est de 25.77%. Contrairement au box plot précédent, celui-ci est un record en matière du nombre d'outliers.

Nous rejetterons l'hypothèse nulle sans équivoque.

8.2.5 Échantillon 3-D avec un trou central en forme de couronne

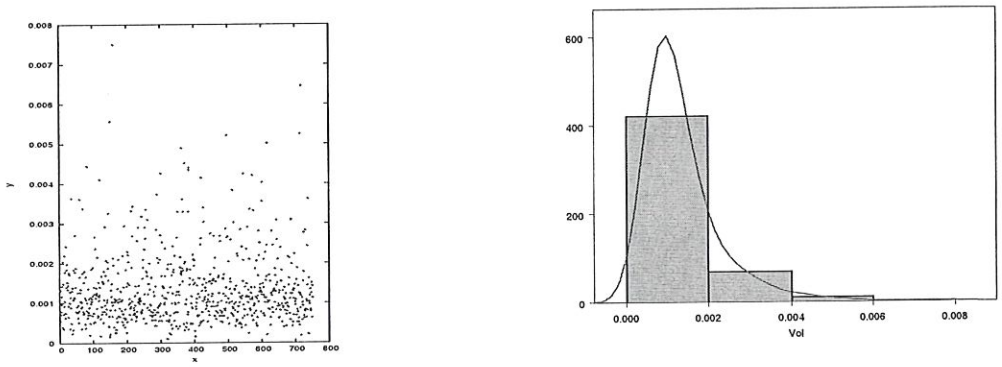


FIG. 8.29: *Scatter plot et histogramme et courbe de densité du volume des cellules*

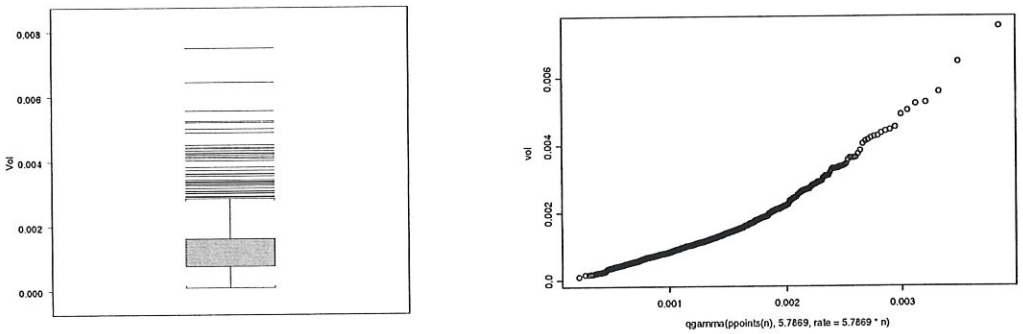


FIG. 8.30: *Box plot et QQ plot Gamma du volume des cellules*

statistiques	Gr	Vol ($\times 10^{-3}$)
valeur minimale	5	0.098 710 8
premier quartile	39	0.769 945
médiane	56	1.105 56
troisième quartile	81.500	1.608 99
valeur maximale	380	7.502 02
moyenne	67.447	4.331 56
écart-type	45.059	0.889 568
borne inférieure de l'IC pour la moyenne	64.220	1.267 83
borne supérieure de l'IC pour la moyenne	70.675	1.395 28

TAB. 8.15: *Statistiques élémentaires sur les résultats du test*

Le satter plot est plus dispersé que les tests précédents, et l'histogramme est un peu moins marqué également. Le box plot présente sans doute moins d'outliers que d'habitude. Ce test reste concluant, même si le QQ plot Gamma est moins « cassant », « anguleux » que dans les précédents tests.

8.2.6 Échantillon 3-D formé de deux sphères disjointes

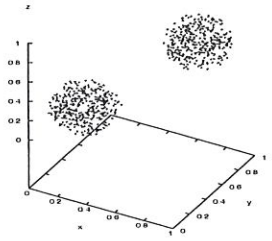


FIG. 8.31: *Scatter plot des points générateurs des cellules*

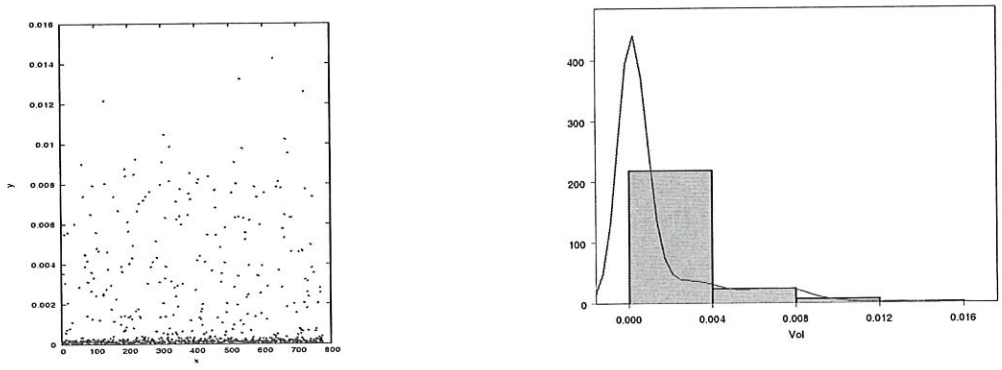


FIG. 8.32: *Scatter plot et histogramme et courbe de densité du volume des cellules*

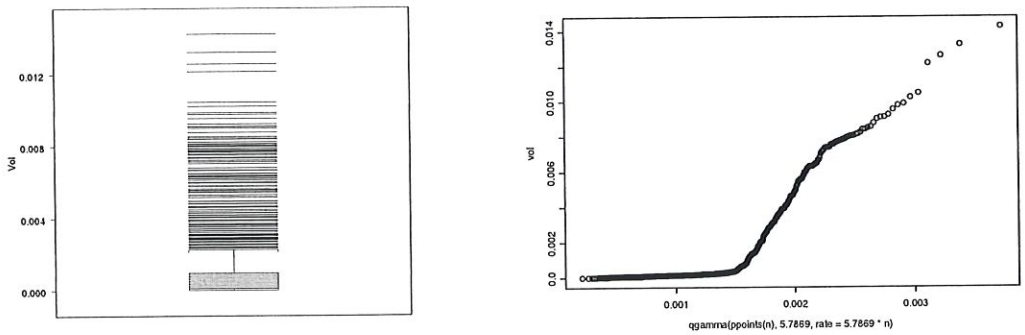


FIG. 8.33: *Box plot et QQ plot Gamma du volume des cellules*

statistiques	Gr	Vol ($\times 10^{-3}$)
valeur minimale	0	0
premier quartile	5	0.098 710 8
médiane	9	0.177 680
troisième quartile	48.75	0.962 431
valeur maximale	722	1.425 38
moyenne	65.443	1.291 99
écart-type	121.029	2.389 38
borne inférieure de l'IC pour la moyenne	56.903	1.123 40
borne supérieure de l'IC pour la moyenne	73.983	1.460 58

TAB. 8.16: *Statistiques élémentaires sur les résultats du test*

Pas de problèmes particuliers avec ce test, on voit bien que les graphes sont assez « extrêmes » – rien que la vue du scatter plot des volumes nous renseigne déjà sur ce point !

8.2.7 Échantillon 3-D formé de deux trous en forme de sphères disjointes

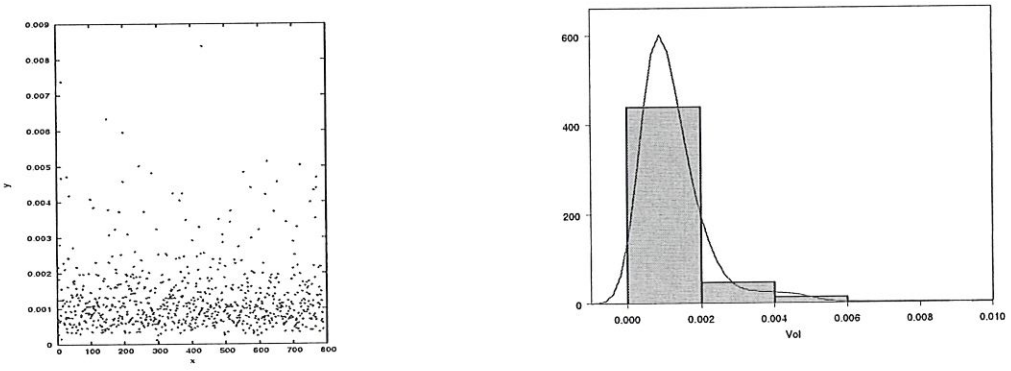


FIG. 8.34: *Scatter plot et histogramme et courbe de densité du volume des cellules*

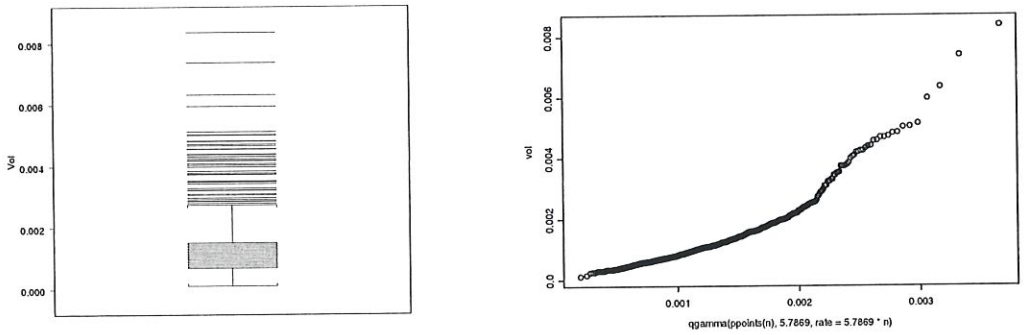


FIG. 8.35: *Box plot et QQ plot Gamma du volume des cellules*

statistiques	Gr	Vol ($\times 10^{-3}$)
valeur minimale	6	0.118 453
premier quartile	35	0.690 976
médiane	53	1.046 34
troisième quartile	77	1.520 15
valeur maximale	424	8.370 68
moyenne	64.199	1.267 42
écart-type	47.148	0.930 813
borne inférieure de l'IC pour la moyenne	60.905	1.202 38
borne supérieure de l'IC pour la moyenne	67.494	1.332 48

TAB. 8.17: *Statistiques élémentaires sur les résultats du test*

Comme pour le cas du trou central en forme de couronne, les graphes ici sont moins marqués que ceux des autres tests. Il faudrait néanmoins beaucoup de talent pour convaincre quelqu'un que les données sont bien uniformes. Mais se pourrait-il que le test réagisse mieux aux formes bien définies qu'aux trous dans l'échantillon ? *A priori*, nous pouvons concevoir qu'un échantillon avec une forme bien définie (une sphère, deux sphères, etc.) génèrerait plus de cellules de grande taille (les outliers) qu'un échantillon avec un trou, où le trou serait « comblé » par peu de cellules de grande taille, ou en tous cas nettement moins que dans le premier cas. Il faut compter avec la forme et la taille du trou. Ainsi, le QQ plot Gamma a une courbure plus douce pour le trou en forme de couronne que pour le trou sphérique : on a gardé 751 cellules (sur 1 000) dans le premier cas contre 738 (sur 1 000) dans le deuxième cas. Le trou sphérique est plus grand et demande plus de cellules de grande taille (d'où davantage d'outliers). La forme de la couronne peut aussi jouer.

Nous ne prétendons pas donner ici une interprétation exacte et rigoureuse, mais plutôt suggérer des pistes de réponses.

8.2.8 Échantillon 3-D formé de deux bandes horizontales disjointes

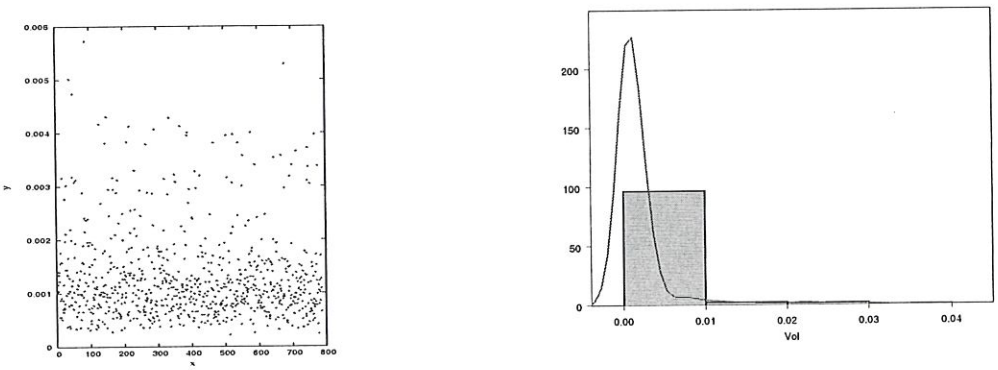


FIG. 8.36: *Scatter plot et histogramme et courbe de densité du volume des cellules*

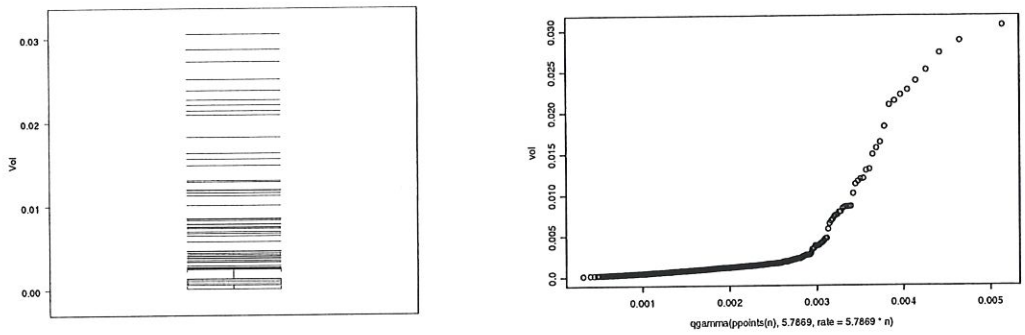


FIG. 8.37: *Box plot et QQ plot Gamma du volume des cellules*

statistiques	Gr	Vol ($\times 10^{-3}$)
valeur minimale	11	0.217 164
premier quartile	39	0.769 945
médiane	53	1.046 34
troisième quartile	75	1.480 66
valeur maximale	290	5.725 23
moyenne	64.362	1.270 65
écart-type	41.317	0.815 677
borne inférieure de l'IC pour la moyenne	61.417	1.213 57
borne supérieure de l'IC pour la moyenne	67.253	1.327 72

TAB. 8.18: *Statistiques élémentaires sur les résultats du test*

Ce test ne pose pas de problèmes.

8.2.9 Conclusions pour les tests 3-D

Les tests sont satisfaisants. Là aussi nous avons essayé sept modifications de l'échantillon. On voit sur les QQ plot Gamma tout d'abord une ligne droite (de la gauche vers la droite) : elle représente les cellules de taille « correcte », la partie des points qui est uniforme, en dehors des trous ou dans des groupes (sphère(s), bandes...). Puis les quantiles s'éloignent fortement de la distribution Gamma, et toujours vers le haut : cela montre que la taille de certaines cellules a fortement augmenté. Le box plot montre très bien cela aussi.

À nouveau, comme pour les tests 2-D, il est impossible à la seule vue de ces graphes (sans le scatter plot des points générateurs !) de deviner la forme initiale de l'échantillon.

8.3 Test 4-D

8.3.1 Caractéristiques des tests

échantillon	nbcell0	nbcell	nbgrid	k	machine	RNG
sphère	5 000	643	50 625	15	B.2	É
trou sphérique	1 000	898	50 625	15	B.1	É
couronne	5 000	608	50 625	15	B.2	É
deux sphères	18 000	553	50 625	15	B.1	É
deux bandes	1 000	587	50 625	15	B.2	É

TAB. 8.19: *Données pour les tests 4-D non uniformes*

échantillon	paramètres
sphère	$r = 0.4$
trou sphérique	$r = 0.4$
couronne	$r_{int} = 0.15,\ r_{ext} = 0.4$
deux sphères	$c_1 = (0.25, 0.25, 0.25, 0.25),\ c_2 = (0.75, 0.75, 0.75, 0.75),\ r_1 = r_2 = 0.23$
deux bandes	$m_1 = 0.3,\ m_2 = 0.7,\ h_1 = h_2 = 0.15$

TAB. 8.20: *Paramètres des échantillons pour les tests 4-D non uniformes*

8.3.2 Échantillon 4-D en forme d'hypersphère centrée

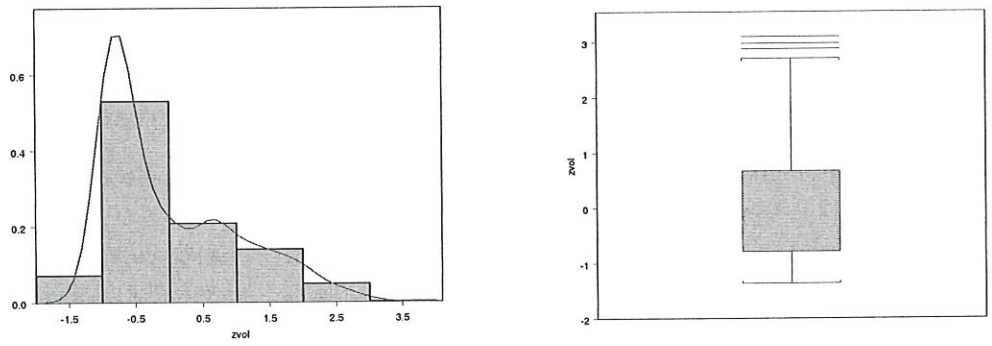


FIG. 8.38: *Histogramme et courbe de densité et box plot de* `zvol` *(4-D sphère)*

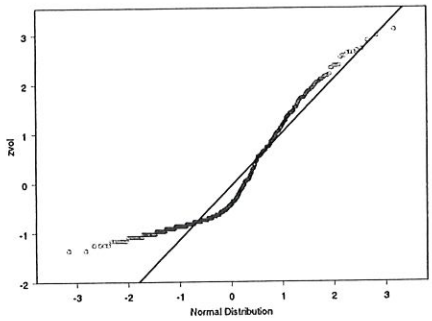


FIG. 8.39: *QQ plot de* `zvol` *(4-D sphère)*

	toutes	**moins 10**	**moins 15**	**moins 20**	**moins 30**
χ^2	496.584 8	522.028 4	471.433 1	470.738 4	441.938
p-value	0	0	0	0	0
K-S	0.169 8	0.1706	0.170 8	0.170 9	0.170 8
p-value	0	0	0	0	0

TAB. 8.21: *Statistiques de tests pour la variable* `zvol` *(4-D sphère)*

	moins 200	**moins 250**	**moins 280**	**moins 300**	**moins 315**
χ^2	181.426 6	100.684 5	116.212 1	133.877 6	151.798 8
p-value	0	0	0	0	0
K-S	0.146 2	0.107 1	0.082 2	0.062 7	0.0642
p-value	0	0.000 2	0.014 7	0.134 7	0.134

TAB. 8.22: *Statistiques de tests pour la variable* `zvol` *(4-D sphère, suite)*

Rappelons que nous testons ici une *autre* statistique, `zvol`, censée suivre une distribution $N(0,1)$ en cas d'uniformité des données.

Nous voyons que le QQ plot est en forme de « S » : cela signifie qu'une des deux distributions a des queues plus longues que l'autre. Puisque les données sont courbées vers le haut à gauche, cela signifie que c'est la distribution empirique des données qui a une queue plus longue. Le box plot nous donne le même renseignement, et la fonction de densité nous le confirme encore. Nous voyons par contre que si la box plot montre bien l'asymétrie de la fonction de fréquence des données, il ne présente que trois outliers. Toutefois, on ne saurait valider en aucun cas l'hypothèse nulle.

Les tests statistiques rejettent aussi l'hypothèse nulle, sans équivoque. Nous pouvons ôter les plus grandes cellules pour tenter de retrouver l'uniformité. À la table ??, « toutes » signifie « toutes les cellules », et « moins x » signifie qu'on a enlevé les x plus grandes cellules. Cependant, nous voyons à la table 8.21 que les valeurs du test-χ^2, au départ énormes, ne diminue que faiblement lorsqu'on enlève peu de cellules ; et la distance de Kolmogorov ne baisse pas non plus. Et nous ne parlons même pas des p-values !

La table 8.22 présente les valeurs les plus basses que nous avons trouvée pour ces statistiques, en soustrayant un certain nombre de cellules. La seule valeur vraiment

intéressante que nous ayons trouvée est d'enlever 300 cellules : le test de Kolmogorov-Smirnov accepte alors l'hypothèse nulle. Cependant, ôter 300 cellules sur 643 est tout de même un peu fort, c'est se priver de la moitié de l'échantillon. On ne peut pas dire qu'enlever la moitié (à peu près) des cellules revient à enlever les « plus grandes cellules », cela fait beaucoup de « plus grandes » cellules !

Cela est peut-être dû à la configuration des données. Nous avons dû prendre 5 000 cellules au départ pour n'en garder que 643 (voir table ??). Nous approximons donc l'hypervolume de l'hypersphère par le rapport de ces nombres, ce qui nous donne 12.86% (l'hypershère occupe seulement 12.86% de l'hypervolume de $B = [0,1]^4$). Donc, nécessairement tous les points se trouvant à l'extérieur de la sphère vont former de grandes cellules, pour « combler le vide ». Il faudrait une formule calculant le rapport de la surface de l'hypersphère sur son volume, en quatre dimensions, pour avoir une idée de la proportion de points qui se trouvent à l'extérieur par rapport à ceux se trouvant à l'intérieur. Remarquons aussi que plus le nombre de données est restreint, plus la précision des statistiques calculées à partir de ces données baisse.

8.3.3 Échantillon 4-D avec un trou en forme d'hypersphère centrée

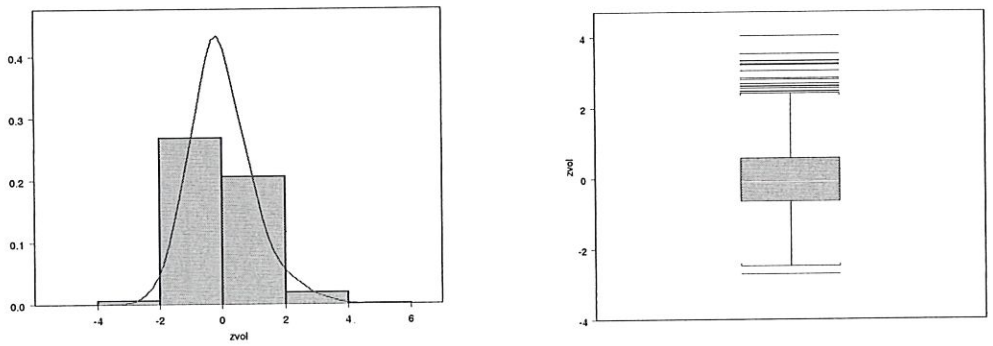


FIG. 8.40: *Histogramme et courbe de densité et box plot de* `zvol` *(4-D trou sphérique)*

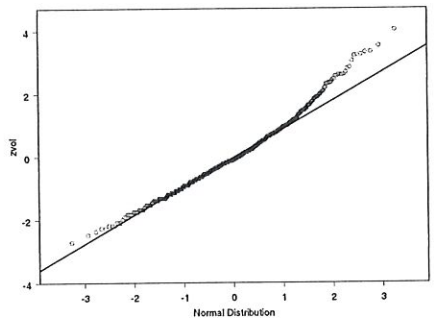


FIG. 8.41: *QQ plot de* `zvol` *(4-D trou sphérique)*

	toutes	moins 10	moins 20	moins 30	moins 40	moins 50
χ^2	60.859 7	82.705	48.398 6	73.336 4	68.573 4	64.452 8
p-value	0.000 7	0	0.018 1	0	0	0.000 2
K-S	0.063 5	0.050 6	0.041 3	0.033 1	0.027 5	0.29 1
p-value	0.001 4	0.021 2	0.100 3	0.298 5	0.534 4	0.469 3

TAB. 8.23: *Statistiques de tests pour la variable* `zvol` *((4-D trou sphérique)*

Les graphes ne permettent pas de conclure que `zvol` suit une loi $N(0,1)$. Le QQ plot a une queue de droite trop importante, et le box plot révèle beaucoup trop d'outliers sur un seul flanc de la fonction de densité des données. Néanmoins la boite et les moustaches restent symétriques (ce qui n'était pas du tout le cas du graphe précédent, ni des deux suivants), et hormis cette trop imposante queue à droite du QQ plot, les données suivent bien la droite sur ce graphe (ce qui n'était pas du tout le cas du graphe précédent non plus). On se souviendra que l'on a enlevé que 10% des cellules pour ce test (table **??**).

Pourquoi si peu ? Parce que le trou hypersphérique doit être « inclus » à $B = [0,1]^4$ (sinon il n'y a plus de données !), et parce que le rapport de l'hypervolume entre l'hypersphère et l'hypercube B diminue lorsque la dimension augmente. Nous ne connaissons pas la formule de l'hypervolume de l'hypersphère en quatre dimensions et plus, mais nous induisons cette assertion à partir de sa véracité dans les dimensions inférieures :

$$\frac{\pi r^2}{m\left([0,1]^2\right)} = \frac{\pi r^2}{1} > \frac{(4/3)\pi r^3}{1} = \frac{(4/3)\pi r^3}{m\left([0,1]^3\right)}$$

(puisque $r \leq \frac{1}{2}$).

Donc, les données n'ont pas été de beaucoup modifiées. Il est justement intéressant de constater que la fait d'ôter peu de données conduit déjà à perturber sérieusement le comportement normal de la variable `zvol`. Souvenons-nous cependant que notre test en quatre dimensions sur un échantillon uniforme présentait déjà quelques outliers (cf. p. 57). Voyons maintenant avec les statistiques.

Le test statistique χ^2 rejette la normalité des données : $\chi^2_{30,0.95} = 43.8 < 60.859\,7$ et la p-value est très faible. Toutefois, la distance de Kolmogorov est assez basse, mais sa p-value est assez petite, bien inférieure à $\alpha = 0.05$. Pour le test 4-D uniforme, les statistiques (table 7.7) confirmaient sans équivoque la normalité des données. Ces tests confirment notre remarque précédente (c.-à-d. que `zvol` serait assez sensible à une modification, même légère, d'une partie des données). Notons toutefois que cette remarque est relative ici au type de modification de l'échantillon (un trou sphérique central dans les données) ; peut-être `zvol` ne réagirait pas de la même manière face à d'autres échantillons modifiés différemment.

Lorsque nous retirons les plus grandes cellules, nous voyons la statistiques χ^2 diminuer, mais pas assez pour être sous la valeur $\chi^2_{30,0.95}$. C'est lorsqu'on enlève 20 cellules que la valeur de ce test est la plus basse.[3] Par contre, le test de Kolmogorov-Smirnov donne de très bons résultats pour un échantillon moins les 40 plus grandes

[3]Relativisons : nous n'avons pas testé toutes les valeurs possibles, mais plutôt certaines valeurs par « tranches » de cellules ôtées : lorsque l'on retire 5 cellules, lorsqu'on en retire 10, 15, 20, 25, 30, 40, 50, 60, 80, 100, 150, 200, 250, 300, 350, plus certaines valeurs différentes d'un test à l'autre selon le comportement des statistiques sur les précédents échantillons.

cellules. Même si la p-value du test-χ^2 est alors nulle, et la valeur de la statistique au-dessus du seuil, nous en concluons cependant, sur base des résultats du test de Kolmogorov-Smirnov, que le fait d'ôter les 40 plus grandes cellules permet de retrouver la normalité pour `zvol` (et donc, par induction

8.3.4 Échantillon 4-D avec une couronne centrée

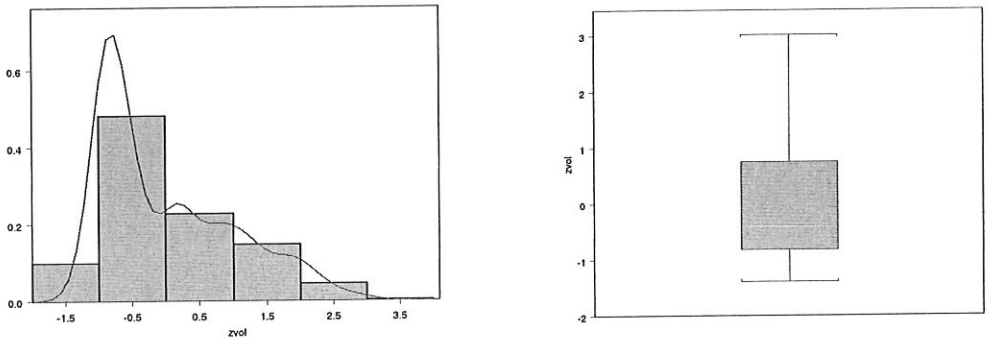


FIG. 8.42: *Histogramme et courbe de densité et box plot de* `zvol` *(4-D couronne)*

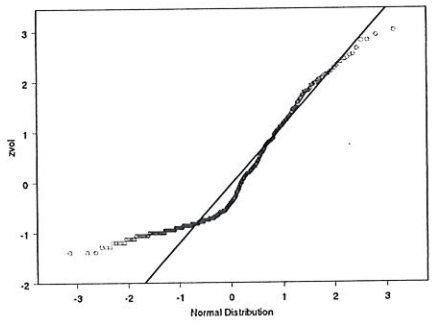


FIG. 8.43: *QQ plot de* `zvol` *(4-D couronne)*

	toutes	**moins 200**	**moins 250**	**moins 270**	**moins 300**	**moins 350**
χ^2	527.960 5	207.137 3	84.458 1	91.816 6	117.194 8	159.558 1
p-value	0	0	0	0	0	0
K-S	0.170 2	0.151 6	0.117 6	0.086 7	0.051	0.097 5
p-value	0	0	0.000 1	0.012 5	0.399 1	0.014 8

TAB. 8.24: *Statistiques de tests pour la variable* `zvol` *(4-D couronne)*

Un examen rapide du QQ plot enlève déjà beaucoup de crédibilité à l'hypothèse nulle. Le box plot ne présente aucun outlier, mais est largement asymétrique. La courbe de densité confirme ces graphes. Les tests eux aussi ne font planer aucun doute à propos du rejet de l'hypothèse nulle (la statistique du test-χ^2 est énorme).

Il faut enlever jusqu'à 200 cellules pour la statistique χ^2 commence à diminuer significativement (même si elle reste complètement au-delà du seuil). C'est lorqu'on enlève 250 cellules qu'elle atteint sa valeur la plus basse. Et pour la statistique de Kolmogorov-Smirnov, les résultats sont très bons lorsque l'on ôte 300 cellules. Mais à nouveau cela revient à ôter la moitié de l'échantillon. Remarquons que la couronne occupe approximativement $\frac{608}{5\,000} = 12.16\%$ du volume total de l'hypercube. Et ça aussi, c'est très peu. Notons que la forme sphérique de l'échantillon est peut-être la cause de l'apparition de nombreuses cellules de grande taille (mais quand même, la moitié...).

8.3.5 Échantillon 4-D formé de deux hypersphères disjointes

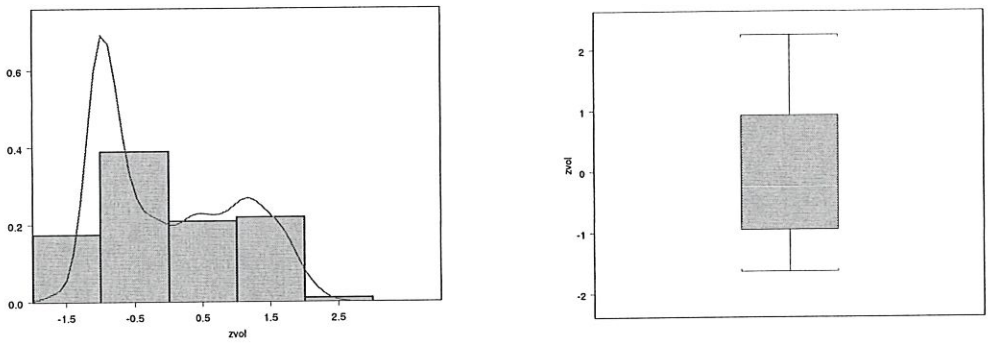


FIG. 8.44: *Histogramme et courbe de densité et box plot de* `zvol` *(4-D deux sphères)*

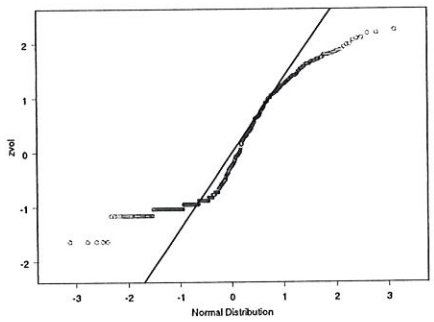


FIG. 8.45: *QQ plot de* `zvol` *(4-D deux sphères)*

	toutes	**moins 40**	**moins 60**	**moins 150**	**moins 200**
χ^2	555.596 7	519.115	477.077 1	353.717 7	310.373 9
p-value	0	0	0	0	0
K-S	0.164 1	0.187 5	0.176 5	0.186 3	0.172 7
p-value	0	0	0	0	0

TAB. 8.25: *Statistiques de tests pour la variable* `zvol` *(4-D deux sphères)*

Les graphes ne laissent planer aucun doute à propos de la non-normalité des données.

Il n'y a pas de problèmes non plus à rejeter l'hypothèse nulle sur bases des valeurs des tests statistiques. Mais ces tests ne sont pas du tout convaincants en ce qui concerne l'idée qu'ôter les plus quelques plus grandes cellules (si elles existent) ramène rapidement la normalité pour la variable `zvol`.

8.3.6 Échantillon 4-D formé de deux bandes horizontales disjointes

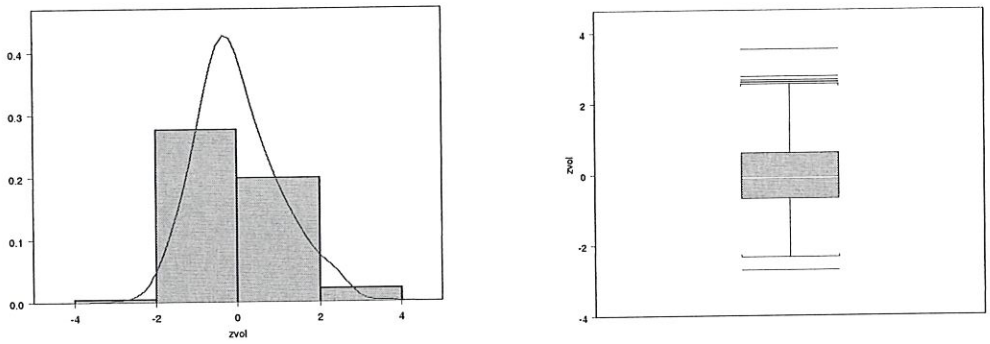


FIG. 8.46: *Histogramme et courbe de densité et box plot de* `zvol` *(4-D deux bandes)*

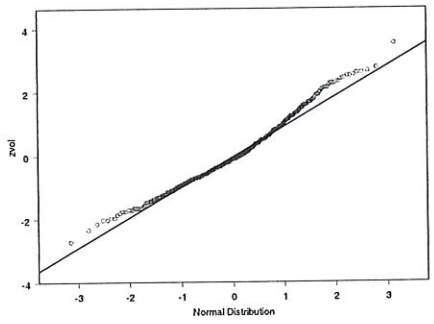


FIG. 8.47: *QQ plot de* `zvol` *(4-D deux bandes)*

	toutes	**moins 10**	**moins 20**	**moins 30**	**moins 40**	**moins 50**
χ^2	47.231 7	41.637 8	27.515	31.944 3	29.736 7	20.402 2
p-value	0.004 6	0.019 7	0.330 7	0.159 6	0.193 6	0.673 7
K-S	0.068 8	0.059 2	0.050 9	0.042 8	0.035 6	0.690 3
p-value	0.007 7	0.034 9	0.105 5	0.259 9	0.492	0.492

TAB. 8.26: *Statistiques de tests pour la variable* `zvol` *(4-D deux bandes)*

L'histogramme est éloigné d'une loi normale, et la courbe de densité s'en ressent : elle n'est pas symétrique (à la limite, elle ferait penser à une fonction entre la loi normale et la loi Gamma). Le box plot n'est pas symétrique (le côté supérieur est

plus grand), mais c'est sans doute l'histogramme qui montre le mieux cette caractéristique. Le box plot présente aussi quelques outliers, six en réalité (nous l'avons vu en effectuant un zoom sur l'image), plus que dans les autres exemples, excepté le trou sphérique. Comme pour le cas du trou sphérique, les données sur le QQ plot « suivent la droite » (ce qui indique leur caractère normal), mais la présence des deux queues relativise ce constat. Pourtant le box plot n'indique que six outliers. Bref, de tous les tests de cette section, c'est sans doute celui qui se rapproche le plus de l'hypothèse de normalité.

Voyons les tests statistiques. Nous avons $\chi^2_{25,0.95} = 37.7$, ce qui fait que le test-χ^2 rejette l'hypothèse de normalité. Mais la valeur du test-χ^2 est beaucoup plus proche du seuil que pour les exemples précédents (excepté toujours le cas du trou sphérique). D'ailleurs, le fait de soustraire les plus grandes cellules est payant sur cet exemple. La statistiques χ^2 tend à montrer qu'il faut ôter les 20 plus grandes cellules, alors que celle de Kolmogorov-Smirnov a sa meilleure valeur pour 40 cellules ôtées. Cela revient à ôter respectivement 3.41% ou 6.81% des cellules.

Notons enfin que la comparaison n'est pas totale avec le trou sphérique : ici on a enlevé à peu près 40% des points initiaux de l'échantillon pour former ces bandes, contre 10% pour le trou sphérique (cf. table **??**). On ne peut pas dire ici que l'échantillon a été *légèrement* modifié.

8.4 Test 5-D

8.4.1 Échantillon 5-D formé de deux bandes horizontales disjointes

échantillon	`nbcell0`	`nbcell`	`nbgrid`	`k`	**machine**	**RNG**
deux bandes	600	475	32 768	8	B.2	P & M

TAB. 8.27: *Données pour le test 5-D non uniforme*

échantillon	**paramètres**
deux bandes	$m_1 = 0.2,\ m_2 = 0.8,\ h_1 = h_2 = 0.2$

TAB. 8.28: *Paramètres des échantillons pour le test 5-D non uniforme*

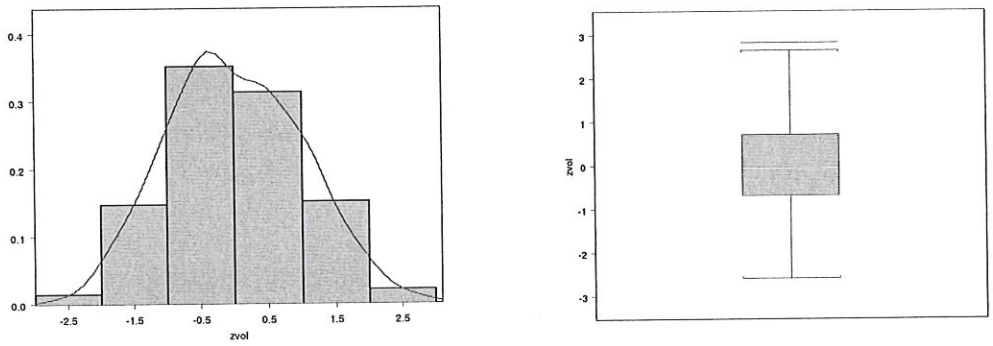


FIG. 8.48: *Histogramme et courbe de densité de* `zvol` *(5-D deux bandes)*

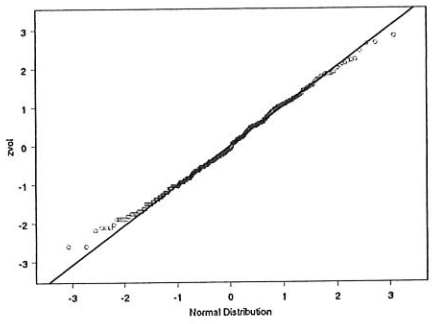


FIG. 8.49: *QQ plot de* `zvol` *(5-D deux bandes)*

	toutes	**moins 5**	**moins 10**	**moins 15**	**moins 20**	**moins 25**
χ^2	33.345 3	35.736 2	35.283 9	56.521 7	39.505 5	63.28
p-value	0.075 2	0.043 9	0.048 7	0.000 1	0.017 4	0
K-S	0.044	0.040 9	0.042 7	0.046 8	0.050 5	0.053 9
p-value	0.316 5	0.412 7	0.364 1	0.266 5	0.196 3	0.146 9

TAB. 8.29: *Statistiques de tests pour la variable* `zvol` *(5-D deux bandes)*

La courbe de densité et l'histogramme ne sont pas très éloignés de la symétrie ; le box plot non plus, et il ne présente qu'un outlier. Les points sur le QQ plot « collent » vraiment bien à la droite y dessinée, sauf aux extrémités mais les queues sont légères (il n'y a qu'un seul outlier dans le box plot). Reconnaissons-le, il est difficile de ne pas valider l'hypothèse de normalité des données. Voyons les tests.

Nous avons la valeur $\chi^2_{23,0.95} = 35.2$, ainsi le test-χ^2 « passe », la p-value étant supérieure à α. La distance de Kolmogorov est plutôt faible et la p-value franchement bonne. Donc, les tests valident l'hypothèse de normalité. On n'est dans ce cas pas étonné de la suite de ces tests : les résultats sont tout de suite plus mauvais lorqu'on enlève des cellules. C'est ici bon signe : cela veut dire que l'on avait la normalité et qu'il n'y a pas de très grandes cellules. Par contre, l'acceptation de l'hypothèse nulle est plus gênante. Nous avons retiré ici 21% des points de l'échantillon initial. Notons que dans les tests 4-D, qui utilisent la même statistique, c'est également sous cette configuration des points que `zvol` se rapprochait le plus de la normalité. Il faudrait faire ici des tests avec d'autres tpyes d'échantillons uniformes modifiés, mais cela est moins aisé que dans les dimensions inférieures, à cause de la complexité exponentielle du programme par rapport à la dimension.

8.4.2 Conclusions pour les tests 4-D et 5-D

Les tests 4-D sont satisfaisants. On s'éloigne bien de la loi normale dès que l'échantillon uniforme est modifié. Le cas du trou central hypersphérique nous laisse penser que la variable `zvol` serait déjà assez sensible à de légères modifications de l'échantillon. Cependant, nous ne savons pas à quel point cette sensibilité est dépendante ou non du type de modification de l'échantillon. Nous voyons d'ailleurs que la

variable `zvol` s'éloigne moins significativement de la normalité dans le cas des deux bandes horizontales.

Pour le test 5-D, comme il n'y en a qu'un seul, tout a déjà été dit à la section s'y rapportant.

Enfin, faisons le point sur notre idée qu'en cas de non-uniformité des points apparaitraient de grandes cellules (et donc, ôter ces cellules permettrait de retrouver rapidement l'uniformité). Deux exemples ici ont fonctionné avec cette approche : le trou sphérique central et les deux bandes (ou vu autrement, un échantillon uniforme avec un trou prismatique central, touchant toutes les frontières « latérales » de l'hypercube). Il nous semble évident que les diagrammes de Voronoi doivent être très différents dans le cas d'un échantillon sphérique (et qui n'occupe approximativement 12% de l'hypercube), et dans celui d'un trou hypersphérique central (qui « occuperait » 10% de l'espace total). Malheureusement nous ne pouvons plus nous représenter mentalement ce type de diagrammes, tout au plus pouvons-nous deviner leur comportement selon la forme de l'échantillon, à partir de nos représentations bi- et tridimensionnelles. Il n'est donc pas exclus de considérer qu'un échantillon comme une hypersphère (nécessairement peu volumineuse puique $r \leq 0.5$) génèrerait, à cause de sa forme, un grand nombre de cellules de grande taille, de sorte qu'on ne puisse plus vraiment dire qu'il y a des cellules « de grande taille », puisque elles constitueraient la majorité des cellules. Si ce raisonnement est pertinent, nous pourrions à l'inverse considérer les cellules de plus petites tailles, pour essayer de deviner la forme des données.

Conclusion et perspectives

Les tests des deux derniers chapitres nous ont montré que l'utilisation des diagrammes de Voronoi, pour tester l'uniformité d'un nuage de points en grande dimension, est intéressante. De manière générale, les tests ont pu confirmer ou infirmer l'uniformité des données originales.

Cependant, nous avons vu que les graphes en cas de non-uniformité se ressemblaient tous (en faisant exception de quelques-uns moins marqués par la non-uniformité des données de départ). Ainsi, il nous est impossible à ce stade de pouvoir deviner la forme de l'échantillon modifié, si nous ne la connaissons pas.

Une perspective à ce travail serait justement de pouvoir détecter la présence de trous dans les données, ou bien de classes de points dans les données – bref, de faire de la classification automatique. On pourrait aussi réaliser d'autres tests de non-uniformité que ceux présentés ici (à savoir un échantillon uniforme dont on enlève une partie des points de manière à créer une forme particulière : sphères, bandes,...). Par exemple, on pourrait lancer la procédure de test sur base d'un échantillon normal.

Notons encore les difficultés auxquelles nous avons dû faire face. En premier, la difficulté théorique de modéliser le comportement des volumes des cellules de Voronoi. Déjà en une dimension, nous parlons de lois Gamma et Bêta. Il n'existe d'ailleurs pas de méthode simple et peu couteuse qui nous permettrait, à partir d'une loi Gamma, de nous ramener à une loi plus connue et plus aisée à manipuler, telle que la loi normale. Pour les dimensions supérieures, le mathématicien doit se contenter de lois Gamma à paramètres non entiers estimés. Il est déjà impossible en deux dimensions d'avoir une expression analytique exacte de la fonction de fréquence des aires des cellules.

Une autre difficulté provient de la complexité exponentielle (par rapport à la dimension de l'espace) de l'algorithme que nous avons développé pour calculer les hypervolumes des cellules. Il est toujours gênant en informatique d'avoir affaire à un algorithme exponentiel, même si les processeurs actuels sont de plus en plus performants. Avec une machine puissante, nous pourrions produire encore un test en six dimensions, en quelques heures. Mais pour un test en dix dimensions ? Cependant, c'est là le prix à payer pour éviter les effets de bord, qui deviennent énormes dès la quatrième dimension, comme Vandooren (voir [18]) l'a montré avec ses tests. Ces effets de bord rendent toute étude impossible au-delà de la troisième dimension. Voilà pourquoi nous ne pouvons nous passer de ces calculs sur espace torique, qui sont à l'origine de la complexité de l'algorithme.

Une constation intéressante dans ce registre, est qu'apparemment un nombre déjà très limité de points-grille suffirait pour les calculs, ou plus exactement : l'accroissement de précision n'est pas très significatif lorsque le nombre de points-grille augmente (nous formulons cela de cette manière car nous ne savons pas quelles sont,

pour un test donné, les valeurs *exactes* des hypervolumes des cellules). Il serait intéressant de pousser plus loin cette analyse de la précision des résultats, ceci afin de pallier à la complexité de l'algorithme.

Mais dans ce but, nous pourrions également retravailler la manière dont nous calculons la distance d'un point à un ensemble de points (dans lequel il est inclus).[4] Nous pourrions implémenter pour cette tâche des méthodes plus subtiles que la comparaison directe à tous les points. Ceci aurait pour effet de diminuer (peut-être même considérablement) la complexité.

Enfin, pour augmenter la précision des résultats (afin de s'approcher davantage des valeurs exactes des volumes), nous pourrions lancer des tests sur un nombre de cellules plus conséquent (par exemple un million). Cela nous permettrait d'avoir de meilleurs estimations sur les statistiques du volume (ou de `zvol`), plutôt que des statistiques sur 500 ou 1 000 données.[5] Une autre manière d'augmenter la précision des résultats serait de considérer un quadrillage du domaine de l'échantillon plus intelligent, avec des points-grille attractifs des points de l'échantillon (points générateurs des cellules).

Notons enfin que dans le cadre de l'utilisation des cellules de Voronoi, nous ne nous sommes intéressés qu'au volume de ces cellules. Il existe dans la littérature (voir par exemple [11]) des études sur d'autres caractéristiques de ces polygones, tout comme il existe des variantes des tessellations de Voronoi. Et il ne faut pas nécessairement aller très loin dans les concepts, rappelons par exemple que tout ce travail a été fait à partir de diagrammes construits sur la distance euclidienne. Déjà en considérant un autre type de distance, nos résultats auraient été différents. Il existe aussi, pour donner un exemple plus évolué, des tessellations courbes, qui offrent comme intérêt d'avoir pour les cellules des arêtes, des contours continument dérivables.

C'est là la force et en même temps le côté perturbant d'un outil comme les diagrammes de Voronoi : il s'agit d'un objet mathématique fondamentalement simple et intuitif, et donc les possibilités inhérentes à cet objet sont énormes. Car un concept simple, basique, est la source de nombreux développements et raffinements, et au contraire une idée pointue reste limitée à ce qu'elle vise. Ainsi toutes les définitions de base des mathématiques sont très simples et générales (la notion de topologie, l'axiomatique de Kolmogorov pour la théorie des probabilités,...). Et le côté perturbant de ce genre d'outil est justement la difficulté de le cerner, d'en mesurer toutes les possibilités. Oui, beaucoup de perspectives sont possibles pour ce travail !

[4] C'est ce que nous faisons dans l'algorithme, afin de déterminer pour chaque point-grille à quelle cellule il appartient, c.-à-d. de quel point générateur il est le plus proche).

[5] Cette perspective est surtout envisageable à partir du moment où l'on a pu réduire la complexité du programme.

Annexe A

Listings des programmes utilisés

A.1 Programme Splus pour les tests en une dimension

```
#==============================================================================
# Auteur:     Denis Doumont
# Cadre:      mémoire 2è licence mathématiques
# Promoteur: Marcel Rémon
# Contenu:    programme S-plus pour les longueurs des cellules de Voronoi (1-D)
#==============================================================================

#******************************************************************************
# PRE:  des nombres aléatoires, générés selon une loi uniforme sur un
#       intervalle B = [0,1], ordonnés par ordre croissant
# POST: la longueur des cellules de Voronoi en 1-D construites à partir de
#       cet échantillon de points
# RMQS: - Pas d'effet de bord (cellules-frontière) car distances toriques
#       - Il est nécessaire d'avoir les points en ordre croissant x_(i) pour
#         le calcul même des longueurs des cellules.
#******************************************************************************
data.restore( "C:\\Mes Documents\\memoire\\le-memoire\\chap3-1D-pratique
\\les-tests\\tests-janvier\\donnees.sdd" )
n <- nrow(donnees)

# INITIALISATION DE LA MATRICE RÉSULTAT
result <- donnees
lambda <- n

# CALCUL DES LONGUEURS DES CELLULES
result[1,2] <- (donnees[2,1]-donnees[n,1]+1)/2
result[n,2] <- (donnees[1,1]-donnees[(n-1),1]+1)/2
for(i in 2:(n-1)) {
result[i,2] <- (donnees[(i+1),1]-donnees[(i-1),1])/2
}

# On peut vérifier que la somme de toutes les longueurs vaut 1:
somme <- sum(result[,2])
# (Effectivement, on a bien somme==1 en pratique.)

# STANDARDISATION DES LONGUEURS: nécessaire ?
# => NON, je ne crois pas !  Car dans la démonstration de li ~ Gamma(2*lambda;2),
# à aucun moment on ne considère des longueurs standardisées !
moyenne <- 1/n
ecartype <- stdev(result[,2])
```

```
#for (i in 1:n) {
# result[i,3] <- (result[i,2]-moyenne)/ecartype
#}

result[1,3] <- n
result[1,4] <- lambda
result[1,5] <- moyenne
result[1,6] <- ecartype
result[1,7] <- somme

# PRÉSENTATION DES RÉSULTATS
guiModify( "double", Name = "result$V1",Precision = "8")
guiModify( "double", Name = "result$X2",Precision = "8")
guiModify( "double", Name = "result$X5",Precision = "8")
guiModify( "double", Name = "result$X6",Precision = "8")
guiModify( "double", Name = "result$X7",Precision = "8")
guiModify( "double", Name = "result$V1",Width = "20")
guiModify( "double", Name = "result$X2",Width = "18")
guiModify( "integer", Name = "result$X3", Width = "15")
guiModify( "double", Name = "result$X5",Width = "14")
guiModify( "double", Name = "result$X6",Width = "14")
guiModify( "double", Name = "result$X7",Width = "15")
guiModify( "double", Name = "result$V1",NewName = "echantillon.ordonne")
guiModify( "double", Name = "result$X2",NewName = "longueur.cellules")
guiModify( "integer", Name = "result$X3", NewName = "nombre.points")
guiModify( "integer", Name = "result$X4", NewName = "lambda")
guiModify( "double", Name = "result$X5",NewName = "moyenne")
guiModify( "double", Name = "result$X6",NewName = "ecart.type")
guiModify( "double", Name = "result$X7",NewName = "somme")

# ÉLABORATION DES GRAPHES (donnée de ces graphes: la longueur des cellules)

# Scatter plot: en abscisse: la référence de chaque point x-(i) (de 1 à n)
# (càd pas la coordonnée de ce point, mais juste son "numéro":(1), (2), ..., (n);
# en ordonnée: la longueur li = l(Vi) = l(V(x_(i))) (longueur de la cellule
# relative au point x_(i) )
guiPlot( PlotType = "Scatter", DataSet = "result", Columns = "longueur.cellules")

# Histogramme + courbe de densité
guiPlot( PlotType = "Histogram Density", DataSet = "result", Columns = "longueur.cellules")

# QQplot Normal
#guiPlot( PlotType = "QQ Normal", DataSet = "result", Columns = "longueur.cellules")

# Boxplot: graphe mettant en évidence les "outliers", càd les données (ici les
# longueurs des cellules) qui dépassent le 95è quantile
guiPlot( PlotType = "Box", DataSet = "result", Columns = "longueur.cellules")

# Comparaison avec une fonction de densité Gamma(2*lambda;2)

# Méthode 1: on génère n points selon une loi Gamma(2*lambda;2) (générateur de
# S-plus); on construit sur un même graphe les fonctions de densité de ces points
# et des longueurs des cellules.
# Inconvénients: on n'a pas la fonction de densité exacte Gamma(2*lambda;2)
# Génération de n points (générateur de S-plus)
#x <- rgamma(n, 2, rate=2*n)
# Graphe de la fonction de densité des longueurs des cellules
#guiPlot( PlotType = "Density", DataSet = "result", Columns = "longueur.cellules")
```

```
# Ajout du graphe de la fonction de densité des points générés selon une loi
# Gamma(2*lambda;2), au graphe de la fonction de densité des longueurs
#guiPlot( PlotType = "Density", DataSet = "x", Columns = "x",
GraphSheet = "GS5", Graph = "1")

# Méthode 2: QQplot entre les longueurs des cellules et une distribution
#            Gamma(2*lambda;2)
plot(qgamma(ppoints(n), 2, rate = 2*n), sort(result$longueur.cellules))

# Méthode 3: comparaison de la fonction de distribution des longueurs avec une supposée
# function de distribtution cumulative
cdf.compare(result$longueur.cellules,dist="gamma", shape=2,rate=2*n)

#-------------------------------------THE END-----------------------------------#
```

A.2 Programme dVOR en C++

Ici est donnée la signification du nom du programme. La lettre d fait référence au fait que c'est un programme gérant des cellules de Voronoi en d dimensions. VOR fait bien sûr référence aux cellules de *Vor*onoi. Enfin, le nom sonne comme « dévore » car il est vrai que c'est un programme coûteux en mémoire et surtout en temps de calcul : il dévore les capacités de la machine !

Précisons encore que pour les tests, nous avons utilisé une version « light » de ce programme : une version qui ne calcule que l'hypervolume des cellules. Ceci afin de gagner du temps à l'exécution du programme.

```
/*============================================================================
  Auteur               : Denis Doumont
  Contenu              : dVOR - programme de calcul de certaines
                         caractéristiques de cellules de Voronoi
  Cadre                : mémoire 2è licence mathématiques
  Promoteur            : Marcel Rémon
  ============================================================================

/*----------------------COMMANDE DE COMPILATION (LINUX)-----------------------
  g++ dVOR_light.cpp -o dVOR_light -lm -lctools -lran -Wall

  NB: -lm: indique au compilateur qu'on utilise la librairie <math.h>
      -lctools et -lran: indique au compilateur qu'on utilise les librairies
       "ctools.h" et "ran.h" (librairies de Fabian Bastin pour la génération
       de nombres aléatoires)
      -Wall: affichage de tous les Warnings (très utile !)
  ----------------------------------------------------------------------------*/

/*--------------------------------REMARQUES-----------------------------------
  - Les commentaires placés dans le code s'appliquent en général à la (ou les)
    ligne(s) en-dessous (sinon on le précise). Exception: les invariants (INV)
    ne sont spécifiés qu'à l'endroit du programme où ils sont vrais.
  - Pour les spécifications de fonctions, on ne met pas dans la pré-
    spécification (PRE) ce que l'utilisateur peut deviner en lisant le
    prototype de la fonction.
  ----------------------------------------------------------------------------*/

#include "dVOR.h"
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <ctype.h>
#include <string.h>
#include <math.h>
#include <values.h>
#include <float.h>
#include <errno.h>
#include "ctools/error.h"
#include "ran.h"
#include <time.h>
#include <sys/time.h>

struct itimerval oldtime, newtime;
```

```
/* maximum value for timerval on Linux systems; quid on other systems?
   The timer resolution is 100 Hz. */
#define MAX_TIMERVAL  (UINT_MAX/100)

#define sqr(x) ((x)*(x))

#define PR 10 // PR pour PRécision (en chiffres décimaux) (PR <= DBL_DIG)
#define MAX 21
#define EPS 1e-6 // Ainsi EPS > 1e-9 >= DBL_EPSILON
#define DIMMAX 50 // DIMMAX <= (long)floor(log(DBL_MAX)/log(3.0))
#define NBCELLMAX 100000
#define NBGRIDMAX FLT_MAX

/*--------------------------VARIABLES GLOBALES-------------------------------
  - dim:    dimension de l'espace euclidien dans lequel on travaille
  - nbcell: nombre de points générateurs (et donc de cellules de Voronoi)
  - nbgene: nombre de nombres aléatoires total à générer (=dim*nbcell)
  - nbgrid: nombre de points-grille total
  ---------------------------------------------------------------------------*/
int dim;
long nbcell, nbgene;
double nbgrid;

//============================PROGRAMME PRINCIPAL=============================
int main() {
  /*************************************************************************
  PRE:  dim:    la dimension de l'espace euclidien
        nbcell: nombre de points générateurs (et donc de cellules de Voronoi)
        nbgrid: nombre de points-grille total
  POST: - Génère nbcell points aléatoires uniformes dans [0,1]^dim.
        - Calcule diverses caractéristiques des cellules de Voronoi générées à
          partir de ces points:
          . volume de la cellule
          . distance entre le point générateur de la cellule et le point-grille
            le plus proche
          . distance entre le point générateur de la cellule et le point-grille
            appartenant à cette cellule et le plus éloigné du point générateur
          . distance moyenne entre le point générateur de la cellule et tous
            les points-grille appartenant à cette cellule.
        - Impression de ces caractéristiques dans le fichier results.dat
  RMQS: - Nous travaillons dans l'espace torique généré à partir de [0,1]^dim
          (ainsi l'espace est fermé sur lui-même et cela évite des effets de
          bord pour les cellules)
        - La syntaxe des résultats est présentée dans le fichier
          miscellaneous.dat.
        - Les résultats ne sont pas exacts, ce sont des APPROXIMATIONS: le
          programme génère des points-grille selon un quadrillage parfait (tous
          les points-grille sont à égales distances les uns des autres).  Ces
          points servent à approximer des grandeurs. P.ex. le volume d'une
          cellule est le rapport entre le nombre de points-grille appartenant à
          cette cellule sur le nombre total de points-grille.  Ainsi, la
          distance moyenne n'est pas calculée par rapport à tous les points de
          la cellule (il y en une infinité -> il faudrait recourir aux matrices
          d'inertie) mais par rapport à tous les points-grille appartenant à
          cette cellule.
  *************************************************************************/

  /*-----------------------DÉCLARATION DES VARIABLES-------------------------
```

```
seed:       semence, graine: entier donné par l'utilisateur pour lancer la
            génération des nombres aléatoires; une même valeur seed donne la
            même suite de nombres
k:          nombre de points-grille par dimension: k = nbgrid^(1/dim) tronqué
Gr[i]:      nombre de points-grille se trouvant dans la cellule i (taille de
            ce tableau: 1 x nbcell)
timexec:    temps total d'exécution du programme
PGe:        tableau des nbcell points générateurs qui sont des dim-uples:
            les dim 1è coordonnées représentent le 1er point, etc.
            (taille:1 x nbgene)
PGr:        tableau des coordonnées du point-grille en cours (1 x dim)
Vol[i]:     hypervolume de la cellule i (1 x nbcell)
dmax[i]:    distance entre le point générateur de la cellule i et le point-
            grille appartenant à la cellule i, le plus éloigné de ce point
            générateur (1 x nbcell)
dmin[i]:    distance entre le point générateur de la cellule i et le point-
            grille le plus proche de ce point (1 x nbcell)
dmoy[i]:    distance moyenne entre le point générateur de la cellule i et tous
            les points-grille présents dans cette cellule (ceci est une
            approximation de l'inertie de la cellule i) (1 x nbcell)
radius:     donnée pour la génération d'un échantillon non-uniforme
            (échantillon uniforme avec trous)
ptsge:      fichier contenant tous les points générateurs des cellules
  ------------------------------------------------------------------------*/
int seed, sample, timexec;
long k, *Gr;
double *PGe0, *PGe, *PGr, *Vol, *dmax, *dmin, *dmoy;
double radius, radius_int, radius_ext, radius1, radius2, *center1, *center2,
  medium1, medium2, half_h1, half_h2;
ofstream ptsge("generator-points.dat");

char secure[MAX];
int out, exit, choix, i, j;
long l, nbcell0;
double times, kprim;

radius = radius_int = radius_ext = radius1 = radius2 = medium1 = medium2 = 0;
half_h1 = half_h2 = 0;
center1 = center2 = NULL;
timexec = 0;

// INITIALISATION DES VARIABLES GLOBALES
system("clear");
cout << "*************************************************************\n"
     << "*            BIENVENUE DANS LE PROGRAMME dVOR_light          *\n"
     << "*************************************************************\n"
     << "\n";
out = 0;
while(out==0) {
  cout << "Voulez-vous une description du programme ?  Si oui, tapez 1.\n"
<< "Sinon tapez 0 (suivi de 'Enter', bien sûr).  ";
  cin >> secure;
  withoutblank(secure);
  if((strcmp(secure,"0")==0)||(strcmp(secure,"1"))==0) out = 1;
}
choix = atoi(secure);
if(choix==1) {
  cout << "-------------Description du programme dVOR_light---------------\n"
```

```
<< "\n"
<< "Ce programme génère des points aléatoires dans [0,1]^d.  Ces\n"
<< "points peuvent suivre une distribution uniforme, ou bien une\n"
<< "distribution uniforme avec 'trous' (sphère, couronne, etc.).\n"
<< "Puis le programme calcule diverses caractéristiques\n"
<< "des cellules de Voronoi construites à partir de ces points.\n"
<< "Pour ce faire, le programme génère des points-grille selon un\n"
<< "quadrillage parfait sur [0,1]^d (tous les points-grille sont à\n"
<< "égales distances les uns des autres).  Ces points servent à\n"
<< "approximer les caractéristiques des cellules de Voronoi:\n"
<< "- l'hypervolume de ces cellules;\n"
<< "- la distance entre le point générateur de la cellule et le\n"
<< "  point-grille le plus proche;\n"
<< "- la distance entre le point générateur de la cellule et le\n"
<< "  point-grille appartenant à cette cellule et le plus éloigné\n"
<< "  du point générateur;\n"
<< "- la distance moyenne entre le point générateur de la cellule\n"
<< "  et tous les points-grille appartenant à cette cellule.\n\n";
   cout << "Les résultats sont transmis dans trois fichiers:\n"
<< "- pts-generateurs.dat: tous les points générés par le\n"
<< "  générateur de nombres aléatoires de ce programme (et qui\n"
<< "  servent à construire les cellules);\n"
<< "- results.dat: caractéristiques calculées pour toutes les "
<< "cellules;\n"
<< "- miscellaneous.dat: quelques résultats et commentaires divers;\n"
<< "  la syntaxe du fichier results.dat y est notamment décrite.\n"
<< "  Jetez-y un coup d'oeil !\n\n";
   cout << "* Note sur l'espace de travail.\n"
<< "  Nous travaillons dans l'espace torique généré à partir de\n"
<< "  [0,1]^dim: ainsi l'espace est clos sur lui-même et cela évite\n"
<< "  des effets de bord pour les cellules).  Il n'y a donc pas à\n"
<< "  proprement parler de 'cellules-frontière' ici.";
   cout << "* Note sur le nombre de points-grille\n"
<< "  Le programme recalcule le nombre de points-grille entré par\n"
<< "  l'utilisateur de sorte que k soit bien un entier.  Le nombre\n"
<< "  de points-grille recalculé est toujours supérieur ou égal au\n"
<< "  nombre initialement entré par l'utilisateur (ainsi\n"
<< "  l'utilisateur ne doit pas se soucier d'entrer un nombre de\n"
<< "  points-grille qui 'tombe' juste avec la dimension).\n\n";
   cout << "* Note sur les résultats du programme\n"
<< "  Nous insistons ici sur le fait que les résultats ne sont pas\n"
<< "  exacts, ce sont des APPROXIMATIONS: le programme génère des\n"
<< "  points-grille selon un quadrillage parfait (tous les points-\n"
<< "  grille sont à égales distances les uns des autres).  Ces\n"
<< "  points servent à approximer des grandeurs. P.ex. le volume\n"
<< "  d'une cellule est le rapport entre le nombre de points-grille\n"
<< "  appartenant à cette cellule sur le nombre total de points-\n"
<< "  grille.  Ainsi, la distance moyenne n'est pas calculée par\n"
<< "  rapport à tous les points de la cellule (il y en une infinité\n"
<< "  -> il faudrait recourir aux matrices d'inertie) mais par\n"
<< "   rapport à tous les points-grille appartenant à cette cellule."
<< "\n\n";
   cout << "* Note sur la génération des nombres aléatoires\n"
<< "  La génération aléatoire uniforme est ici effectuée d'après\n"
<< "  l'algorithme de Pierre L'Écuyer, implémenté par Fabian\n"
<< "  Bastin."
<< "  Les différentes réalisations du générateur de nombres sont\n"
<< "  supposées indépendantes (pas de corrélation entre les\n"
```

```
<< "  points).\n"
<< "  L'entier 'seed' (semence, graine) est un nombre donné par\n"
<< "  l'utilisateur pour lancer la génération des nombres.\n"
<< "  L'avantage de cette implémentation réside dans le fait qu'une\n"
<< "  même valeur seed génère les mêmes nombres: cela permet de\n"
<< "  retrouver des nombres précédemment générés.  Si l'utilisateur\n"
<< "  entre une valeur négative, le programme prendra seed=0.\n\n"
<< "  Références:\n"
<< "  - http://www.iro.umontreal.ca/~lecuyer/papers.html\n"
<< "    Publications de Pierre L'Écuyer, nř114: 'Efficient and\n"
<< "    Portable Combined Random Number Generators', Communications\n"
<< "    of the ACM, 31 (1988), 742-749 and 774.\n"
<< "  - http://www.fundp.ac.be/~fbastin/computing/rangen/index.fr."
<< "html\n"
<< "    http://www.fundp.ac.be/~fbastin/computing/languages.fr.html\n"
<< "    Librairie de génération de nombres aléatoires de Fabian\n"
<< "    Bastin (fichiers ran-0.6.0 et ctools-0.5.1 respectivement)\n"
<< "\n";
 }
 cout << "--------------------Lancement du programme-----------------------\n"
      << "\n";

 out = 0;
 while(out==0) {
   cout << "Veuillez entrer la dimension de l'espace euclidien (un entier\n"
<< "positif inférieur à " << DIMMAX << ", suivi de 'Enter'):\n";
   cin >> secure;
   withoutblank(secure);
   i = 0;
   out = 1;
   while(secure[i]!='\0') {
     if(isdigit(secure[i])==0) out = 0;
     i++;
   }
   dim = atoi(secure);
   if((dim==0)||(dim>DIMMAX)) out = 0;
   if(out==0) cout << "\t\t\t!!!! Erreur !!!!\n";
 }
 // INV: dim a été correctement entré (entier positif < DIMMAX)

 out = 0;
 while(out==0) {
   cout << "Veuillez entrer le nombre de cellules de Voronoi souhaité (un\n"
<< "entier positif inférieur à " << NBCELLMAX
<< ", suivi de 'Enter'):\n";
   cin >> secure;
   withoutblank(secure);
   i = 0;
   out = 1;
   while(secure[i]!='\0') {
     if(isdigit(secure[i])==0) out = 0;
     i++;
   }
   nbcell = atol(secure);
   if((nbcell==0)||(nbcell>NBCELLMAX)) out = 0;
   if(out==0) cout << "\t\t\t!!!! Erreur !!!!\n";
 }
 // INV: nbcell a été correctement entré (entier positif)
```

```
 out = 0;
 while(out==0) {
   cout << "Veuillez entrer le nombre de points-grille souhaité (un\n"
<< "entier positif inférieur à " << NBGRIDMAX
<< ", suivi de 'Enter'):\n";
   cin >> secure;
   withoutblank(secure);
   i = 0;
   out = 1;
   while(secure[i]!='\0') {
     if(isdigit(secure[i])==0) out = 0;
     i++;
   }
   nbgrid = atof(secure);
   if((nbgrid==0.0)||(nbgrid>NBGRIDMAX)) out = 0;
   if(out==0) cout << "\t\t\t!!!! Erreur !!!!\n";
 }
 // INV: nbgrid a été correctement entré (entier positif)

 cout << "\nDimension de l'espace: " << dim << "\nNombre de cellules:  "
      << nbcell << "\nNombre de points-grille total (original): "
      << nbgrid << "\n";

 nbcell0 = nbcell;
 nbgene = dim*nbcell;
 k = (long)ceil(pow(nbgrid, 1.0/((double)dim)));
 nbgrid = pow((double)k,(double)dim);
 cout << "Nombre de points-grille total (recalculé): " << setprecision(PR)
      << nbgrid << "\nNombre de points-grille/dimension: " << k << endl
      << "Nombre moyen de points-grille par cellule: " << setprecision(PR)
      << nbgrid/(double)nbcell << endl;

 PGe0 = (double *)malloc(nbgene*sizeof(double));
 PGe = (double *)malloc(nbgene*sizeof(double));
 PGr = (double *)malloc(dim*sizeof(double));
 Vol = (double *)malloc(nbcell*sizeof(double));
 dmax = (double *)malloc(nbcell*sizeof(double));
 dmin = (double *)malloc(nbcell*sizeof(double));
 dmoy = (double *)malloc(nbcell*sizeof(double));
 Gr = (long *)malloc(nbcell*sizeof(long));

 // INITIALISATION DES VARIABLES
 init(Gr, dmax, dmin, dmoy);

 // Génération des points générateurs des cellules de Voronoi
 cout << "Entrez seed, please (un entier): ";
 cin >> seed;
 generator(seed, nbgene, PGe0);
 cout << "\nQuel type d'échantillon voulez-vous ?\n"
      << "(Tous ces échantillons sont inclus dans B = [0,1]^dim; quand\n"
      << "on dit 'centré', c'est par rapport au centre de B, bien sûr.)\n\n"
      << "  0.Échantillon uniforme\n"
      << "Échantillon uniforme modifié:\n"
      << "  1. une hypersphère centrée\n"
      << "  2. un 'trou' hypersphérique centré\n"
      << "  3. une 'couronne' centrée\n"
      << "  4. un 'trou' en forme de 'couronne' centrée\n"
```

```
      << "  5. deux hypersphères dans deux 'coins' de B\n"
      << "  6. deux 'trous' en forme d'hypersphères dans deux coins de B\n"
      << "  7. deux bandes horizontales\n\n";
 cout << "Votre choix (tapez le chiffre à gauche de l'option choisie): ";
 cin >> sample;
 if(sample==0) PGe = PGe0;
 if(sample==1) {
   cout << "Entrez le rayon de l'hypersphère (0 < r <= 0.5)\n"
<< "(si r=0.5, il s'agit de l'hypersphère inscrite dans B): ";
   cin >> radius;
   center_sphere(PGe0, PGe, radius);
 }
 else if(sample==2) {
   cout << "Entrez le rayon du trou hypersphérique (0 < r <= 0.5)\n"
<< "(si r=0.5, le trou 'touche' les frontières de B): ";
   cin >> radius;
   center_spheric_hole(PGe0, PGe, radius);
 }
 else if(sample==3) {
   cout << "Entrez les rayons extérieur et intérieur de la 'couronne'\n"
<< "(0 < r_int < r_ext <= 0.5).\n"
<< "Rayon intérieur: ";
   cin >> radius_int;
   cout << "Rayon extérieur: ";
   cin >> radius_ext;
   center_ring(PGe0, PGe, radius_int, radius_ext);
 }
 else if(sample==4) {
   cout << "Entrez les rayons extérieur et intérieur de la 'couronne'\n"
<< "(0 < r_int < r_ext <= 0.5).\n"
<< "Rayon intérieur: ";
   cin >> radius_int;
   cout << "Rayon extérieur: ";
   cin >> radius_ext;
   center_ring_hole(PGe0, PGe, radius_int, radius_ext);
 }
 else if(sample==5) {
   center1 = (double *)malloc(dim*sizeof(double));
   center2 = (double *)malloc(dim*sizeof(double));
   cout << "Entrez les coordonnées du centre de la première sphère\n"
<< "(chaque coordonnée suivie de <Enter>):\n";
   for(i=0; i<dim; i++) {
     cout << "center1[" << i << "] = ";
     cin >> center1[i];
   }
   cout << "Entrez le rayon de la première sphère\n"
<< "(0 < radius1 et tq la sphère soit incluse dans B): ";
   cin >> radius1;
   cout << "Entrez les coordonnées du centre de la deuxième sphère\n"
<< "(chaque coordonnée suivie de <Enter>):\n";
   for(i=0; i<dim; i++) {
     cout << "center2[" << i << "] = ";
     cin >> center2[i];
   }
   cout << "Entrez le rayon de la deuxième sphère\n"
<< "(0 < radius2 et tq la sphère soit incluse dans B): ";
   cin >> radius2;
```

```
  two_corner_spheres(PGe0, PGe, radius1, radius2, center1, center2);
}
else if(sample==6) {
  center1 = (double *)malloc(dim*sizeof(double));
  center2 = (double *)malloc(dim*sizeof(double));
  cout << "Entrez les coordonnées du centre du premier trou sphérique\n"
<< "(chaque coordonnée suivie de <Enter>):\n";
  for(i=0; i<dim; i++) {
    cout << "center1[" << i << "] = ";
    cin >> center1[i];
  }
  cout << "Entrez le rayon du premier trou sphérique\n"
<< "(0 < radius1 et tq le trou sphérique soit 'inclus' dans B): ";
  cin >> radius1;
  cout << "Entrez les coordonnées du centre du deuxième trou sphérique\n"
<< "(chaque coordonnée suivie de <Enter>):\n";
  for(i=0; i<dim; i++) {
    cout << "center2[" << i << "] = ";
    cin >> center2[i];
  }
  cout << "Entrez le rayon du deuxième trou sphérique\n"
<< "(0 < radius2 et tq le trou soit'inclus' dans B): ";
  cin >> radius2;

  two_corner_spheric_holes(PGe0, PGe, radius1, radius2, center1, center2);
}
else if(sample==7) {
  cout << "Entrez la coordonnée du milieu de la première bande\n"
<< "(0 < medium1 < 1): ";
  cin >> medium1;
  cout << "Entrez la demi-hauteur de la première bande\n"
<< "(0 < half_h1 < 1): ";
  cin >> half_h1;
  cout << "Entrez la coordonnée du milieu de la deuxième bande\n"
<< "(0 < medium2 et tq les deux bandes seront disjointes): ";
  cin >> medium2;
  cout << "Entrez la demi-hauteur de la deuxième bande\n"
<< "(0 < half_h2 et tq les deux bandes sont disjointes): ";
  cin >> half_h2;
  two_horizontal_bands(PGe0, PGe, medium1, medium2, half_h1, half_h2);
}

cout << "Nombre de cellules (recalculé):" << nbcell << endl
     << "Nombre moyen de points-grille par cellule: " << setprecision(PR)
     << nbgrid/(double)nbcell << endl;

// Affichage des nombres aléatoires générés (les points générateurs)
for(l=0;l<nbgene;l++) {
  ptsge << l+1 << "\t" << setprecision(PR) << PGe[l] << "\n";
}
ptsge.close();
cout << ".....Calculs en cours !.....\n";
nlsps_init_timer();

// CORPS DU PROGRAMME: GÉNÉRATION ET TRAITEMENT DES POINTS-GRILLE
for(i=0; i<dim; i++) PGr[i] = 0.0;
treatPGr(PGe, PGr, nbgene, Gr, dmax, dmin, dmoy);
times = 1.0;
```

```
  kprim = (double)k;
  while(times<nbgrid) {
    // BINV: times = nombre de points-grille déjà traités
    exit = 0;
    j = dim-1;
    while(exit==0) {
      if(fmod(times,pow(kprim,(double)j))==0.0) {
PGr[j] += 1.0/kprim;
for(i=0; i<j; i++) PGr[i] = 0.0;
treatPGr(PGe, PGr, nbgene, Gr, dmax, dmin, dmoy);
times += 1.0;
// cout << times << ". PGr = ";
// for(i=0;i<dim; i++) cout << PGr[i] << "  ";
// cout << endl;
exit = 1;
      }
      else j--;
    }
    // INV: times a été "incrémenté" une fois (càd on a traité un point-
    //      grille, parmi les nbgrid possibles)
  }

  // Calcul du volume de chaque cellule
  // Calcul de la distance moyenne entre le point générateur de la cellule i
  // et tous les points-grille présents dans cette cellule
  // (pour chaque cellule)
  // Calculs de vérification
  for(l=0; l<nbcell; l++) {
    Vol[l] = ((double)Gr[l])/nbgrid; // Division réelle
    dmoy[l] /= (double)Gr[l];
  }

  // IMPRESSION DES RÉSULTATS
  //  timexec = clock()/CLOCKS_PER_SEC;
  timexec = nlsps_get_timer();

  print(seed,Gr,Vol,dmax,dmin,dmoy,k,timexec,nbcell0,sample,radius,radius_int,
radius_ext,radius1,radius2,center1,center2,medium1,medium2,half_h1,
        half_h2);

  cout << "\nTemps d'exécution des calculs: " << timexec << " s\n";
  //  timexec = clock()/CLOCKS_PER_SEC;
  //  timexec = nlsps_get_timer();
  //  cout << "Temps d'exécution des calculs + impression des résultats: "
  //       << timexec << " s\n";

  // CLÔTURE DU PROGRAMME
  free(PGe0);
  free(PGe);
  free(PGr);
  free(Vol);
  free(dmax);
  free(dmin);
  free(dmoy);
  free(Gr);
  cout << "********************** Fin du programme **********************\n";
  return 0;
}
```

```
//=======================FIN DU PROGRAMME PRINCIPAL==========================

void withoutblank (char chr[]) {
  /*****************************************************************************
  PRE:
  POST: - enlève les caractères blancs au début et à la fin de chr
          (s'il y en a)
        - ajoute un '\0' après le dernier caractère de chr ainsi modifiée
  *****************************************************************************/
  int i, j, l, lenchr;
  i = lenchr = 0;

  lenchr=strlen(chr);
  while (chr[i] == ' ') i++;
  // INV: i = position du premier caractère de chr différent de ' '
  j=lenchr-1;
  while ((chr[j] == ' ') && (j>i)) j--;
  // INV: j = position du dernier caractère de chr différent de ' '
  lenchr=j-i+1; // longueur de la string sans le(s) caractère(s) blanc(s)
  for (l=0;l<lenchr;l++) {
    chr[l]=chr[i];
    i++;
  }
  chr[lenchr]='\0';
}
/*
  double sqr(const double x) {
  //  PRE:
  //  POST: renvoie x^2
  return x*x;
  }
*/
void nlsps_init_timer() {
  /*****************************************************************************
  PRE:
  POST: initialise le compteur du temps d'exécution du programme
  *****************************************************************************/
  int r;
  /* Set up interval timer; we start the timer now */
  oldtime.it_value.tv_sec = MAX_TIMERVAL;
  oldtime.it_value.tv_usec = 0;
  oldtime.it_interval.tv_sec = oldtime.it_interval.tv_usec = 0;
  r = setitimer(ITIMER_VIRTUAL, &oldtime, NULL);

  /* adjust the value of oldtime */
  getitimer(ITIMER_VIRTUAL, &newtime);
  oldtime.it_value.tv_sec = newtime.it_value.tv_sec;
  oldtime.it_value.tv_usec = newtime.it_value.tv_usec;

  if(r == EINVAL || r == EFAULT)
    c_warning("Unable to set interval timer.\n");
}

int nlsps_get_timer() {
  /*****************************************************************************
  PRE:
  POST: le temps (en secondes) écoulé depuis le début du programme
  *****************************************************************************/
```

```
  struct itimerval tmp;

  memset(&tmp, 0, sizeof(tmp));
  getitimer(ITIMER_VIRTUAL, &newtime);
  timersub(&(oldtime.it_value), &(newtime.it_value), &(tmp.it_value));
  // champ du temps écoulé en secondes: tmp.it_value.tv_sec
  // champ du temps écoulé en microsecondes: tmp.it_value.tv_usec
  // On arrondit à la seconde la plus proche
  return tmp.it_value.tv_sec + ((tmp.it_value.tv_usec >= 500000) ? 1 : 0);
}

void generator(int seed, int n, double *P) {
  /************************************************************************
  PRE:  seed: entier; si seed<0, le programe prendra seed=0.
  POST: Renvoie (dans le tableauP) n nombres aléatoires générés uniformément
        dans [0,1].
  RMQS: - La génération aléatoire uniforme est ici effectuée d'après
          l'algorithme de Pierre L'Écuyer, implémenté par Fabian Bastin.
        - Les différentes réalisations du générateur de nombres sont supposées
  indépendantes (pas de corrélation entre les points).
        - L'entier seed (semence, graine) est un nombre donné par l'utilisateur
  pour lancer la génération des nombres aléatoires. L'avantage de cette
          implémentation réside dans le fait qu'une même valeur seed génère les
          mêmes nombres (cela permet de retrouver des nombres précédemment
          générés).
  ************************************************************************/
  int i;
  Random *r;

  r = ran_random_new(&seed);
  // Attention: pour la version ran-0.6.0 !
  ran_random_set_generator(r, RAN_ECUYER);
  for(i=0;i<n;i++) {
    P[i] = ran_random_get_val(r);
  }
  ran_random_free(r);
}

void center_sphere(const double *PGe0, double *PGe, const double radius) {
  /************************************************************************
  PRE:  PGe0 initialisé (on a déjà lancé le générateur)
        0 < radius <= 0.5
  POST: supprime des points de l'échantillon de manière à avoir des points en
        forme d'hypersphère centrée dans l'hypercube B=[0,1]^dim; résultats
        dans le tableau PGe
  RMQ:  1. radius=0.5 correspond à l'hypersphère inscrite dans l'hypercube
        2. Cette fonction modifie les variables globales nbcell et nbgene
  ************************************************************************/
  int i;
  long j, l;
  double *center, dcenter;
  center = (double *)malloc(dim*sizeof(double));

  l = 0;
  for(i=0; i<dim; i++) center[i] = 0.5;

  for(j=0; j<nbgene; j+=dim) {
    dcenter = 0.0;
```

```
    for(i=0; i<dim; i++) dcenter += sqr(PGe0[j+i]-center[i]);
    dcenter = sqrt(dcenter);
    if(dcenter<=radius) {
      for(i=0; i<dim; i++) PGe[l+i] = PGe0[j+i];
      l += dim;
    }
    else nbcell--;
  }
  nbgene = nbcell*dim;
  free(center);
}

void center_spheric_hole(const double *PGe0, double *PGe, const double
 radius) {
  /*****************************************************************************
  PRE:  PGe0 initialisé (on a déjà lancé le générateur)
        0 < radius <= 0.5
  POST: supprime des points de l'échantillon de manière à avoir un 'trou' de
        forme hypersphérique, centré dans l'hypercube B=[0,1]^dim; résultats
        dans le tableau PGe
  RMQ:  1. Si radius=0.5, le trou sphérique 'touche' les frontières de B
        2. Cette fonction modifie les variables globales nbcell et nbgene
  *****************************************************************************/
  int i;
  long j, l;
  double *center, dcenter;
  center = (double *)malloc(dim*sizeof(double));

  l = 0;
  for(i=0; i<dim; i++) center[i] = 0.5;

  for(j=0; j<nbgene; j+=dim) {
    dcenter = 0.0;
    for(i=0; i<dim; i++) dcenter += sqr(PGe0[j+i]-center[i]);
    dcenter = sqrt(dcenter);
    if(dcenter>=radius) {
      for(i=0; i<dim; i++) PGe[l+i] = PGe0[j+i];
      l += dim;
    }
    else nbcell--;
  }
  nbgene = nbcell*dim;
  free(center);
}

void center_ring(const double *PGe0, double *PGe, const double radius_int,
 const double radius_ext) {
  /*****************************************************************************
  PRE:  PGe0 initialisé (on a déjà lancé le générateur)
        0 < radius_int < radius_ext <= 0.5
  POST: supprime des points de l'échantillon de manière à avoir des points en
        forme de 'couronne', centrée dans l'hypercube B=[0,1]^dim; résultats
        dans le tableau PGe
  RMQ:  1. Si radius_ext=0.5, l'hypercouronne est inscrite dans B
        2. Cette fonction modifie les variables globales nbcell et nbgene
  *****************************************************************************/
  int i;
  long j, l;
```

```
 double *center, dcenter;
 center = (double *)malloc(dim*sizeof(double));

 l = 0;
 for(i=0; i<dim; i++) center[i] = 0.5;

 for(j=0; j<nbgene; j+=dim) {
   dcenter = 0.0;
   for(i=0; i<dim; i++) dcenter += sqr(PGe0[j+i]-center[i]);
   dcenter = sqrt(dcenter);
   if(dcenter>=radius_int && dcenter<=radius_ext) {
     for(i=0; i<dim; i++) PGe[l+i] = PGe0[j+i];
     l += dim;
   }
   else nbcell--;
 }
 nbgene = nbcell*dim;
 free(center);
}

void center_ring_hole(const double *PGe0, double *PGe, const double radius_int,
      const double radius_ext) {
 /*****************************************************************************
 PRE:  PGe initialisé (on a déjà lancé le générateur)
       0 < radius_int < radius_ext <= 0.5
 POST: supprime des points de l'échantillon de manière à avoir un 'trou' en
       forme de 'couronne', centrée dans l'hypercube B=[0,1]^dim; résultats
       dans le tableau PGe
 RMQ:  1. Si radius_ext=0.5, le trou en forme de couronne 'touche' les
          frontières de B
       2. Cette fonction modifie les variables globales nbcell et nbgene
 *****************************************************************************/
 int i;
 long j, l;
 double *center, dcenter;
 center = (double *)malloc(dim*sizeof(double));

 l = 0;
 for(i=0; i<dim; i++) center[i] = 0.5;

 for(j=0; j<nbgene; j+=dim) {
   dcenter = 0.0;
   for(i=0; i<dim; i++) dcenter += sqr(PGe0[j+i]-center[i]);
   dcenter = sqrt(dcenter);
   if(dcenter<=radius_int || dcenter>=radius_ext) {
     for(i=0; i<dim; i++) PGe[l+i] = PGe0[j+i];
     l += dim;
   }
   else nbcell--;
 }
 nbgene = nbcell*dim;
 free(center);
}

void two_corner_spheres(const double *PGe0, double *PGe, const double radius1,
const double radius2, const double *center1, const double *center2) {
 /*****************************************************************************
 PRE:  PGe initialisé (on a déjà lancé le générateur)
```

```
      0 < radius1, radius2 et tq les sphères sont incluses dans B=[0,1]^dim
 POST: supprime des points de l'échantillon de manière à avoir des points en
      forme de deux hypersphères placées dans deux "coins" de B
 RMQ:  Cette fonction modifie les variables globales nbcell et nbgene
 ****************************************************************************/
 int i;
 long j, l;
 double dcenter1, dcenter2;

 l = 0;

 for(j=0; j<nbgene; j+=dim) {
   dcenter1 = dcenter2 = 0.0;
   for(i=0; i<dim; i++) {
     dcenter1 += sqr(PGe0[j+i]-center1[i]);
     dcenter2 += sqr(PGe0[j+i]-center2[i]);
   }
   dcenter1 = sqrt(dcenter1);
   dcenter2 = sqrt(dcenter2);
   if(dcenter1<=radius1 || dcenter2<=radius2) {
     for(i=0; i<dim; i++) PGe[l+i] = PGe0[j+i];
     l += dim;
   }
   else nbcell--;
 }
 nbgene = nbcell*dim;
}

void two_corner_spheric_holes(const double *PGe0, double *PGe, const double
      radius1, const double radius2, const double *center1, const double *center2) {
  /****************************************************************************
  PRE:  PGe initialisé (on a déjà lancé le générateur)
        0 < radius1, radius2 et tq les 'trous' sphériques sont inclus dans
        B=[0,1]^dim
  POST: supprime des points de l'échantillon de manière à avoir deux 'trous'
        sphériques placés dans deux "coins" de B
  RMQ:  Cette fonction modifie les variables globales nbcell et nbgene
  ****************************************************************************/
  int i;
  long j, l;
  double dcenter1, dcenter2;

  l = 0;

  for(j=0; j<nbgene; j+=dim) {
    dcenter1 = dcenter2 = 0.0;
    for(i=0; i<dim; i++) {
      dcenter1 += sqr(PGe0[j+i]-center1[i]);
      dcenter2 += sqr(PGe0[j+i]-center2[i]);
    }
    dcenter1 = sqrt(dcenter1);
    dcenter2 = sqrt(dcenter2);
    if(dcenter1>=radius1 && dcenter2>=radius2) {
      for(i=0; i<dim; i++) PGe[l+i] = PGe0[j+i];
      l += dim;
    }
    else nbcell--;
  }
```

```
  nbgene = nbcell*dim;
}

void two_horizontal_bands(const double *PGe0, double *PGe, double medium1,
  double medium2, double half_h1, double half_h2) {
  /*****************************************************************************
  PRE:  PGe initialisé (on a déjà lancé le générateur)
        coordonnées pour les bandes tq les bandes soient disjointes
  POST: supprime des points de l'échantillon de manière à avoir des points en
        forme de deux bandes horizontales dans B=[0,1]^dim
  RMQ:  Cette fonction modifie les variables globales nbcell et nbgene
  *****************************************************************************/
  int i;
  long j, l;
  double d1, d2;

  l = 0;

  for(j=0; j<nbgene; j+=dim) {
    d1 = fabs(PGe0[j+dim-1]-medium1);
    d2 = fabs(PGe0[j+dim-1]-medium2);
    if(d1<=half_h1 || d2<=half_h2) {
      for(i=0; i<dim; i++) PGe[l+i] = PGe0[j+i];
      l += dim;
    }
    else nbcell--;
  }
  nbgene = nbcell*dim;
}

void init(long *Gr, double *dmax, double *dmin, double *dmoy) {
  /*****************************************************************************
  PRE:  Toutes ces variables non initialisées.
  POST: Toutes ces variables initialisées.
  *****************************************************************************/
  long i;
  for(i=0; i<nbcell; i++) {
    Gr[i] = 0;
    dmax[i] = dmoy[i] = 0.0;
    dmin[i] = 1.0e9;
  }
}

inline double dtore2(const double x[], const double y[]) {
  /*****************************************************************************
  PRE:  x et y: des dim-uples appartenant à B=[0,1]^dim.
  POST: La distance euclidienne au carré entre x et y sur le tore développé à
        partir de B.
  *****************************************************************************/
  // Initialisation
  int i,j, exit;
  double times, d, dtemp;
  double *a;
  a = (double *)malloc(dim*sizeof(double));
  dtemp = 0.0;
  for(i=0; i<dim; i++) a[i] = -1.0;
  for(i=0; i<dim; i++) dtemp += sqr(x[i]-y[i]+a[i]);
  times = 1.0;
```

```
  while(times<pow(3.0,(double)dim)) {
    // BINV: times = nombre de distances déjà calculées
    exit = 0;
    j = dim-1;
    while(exit==0) {
      if(fmod(times,pow(3.0,(double)j))==0.0) {
a[j] += 1.0;
for(i=0; i<j; i++) a[i] = -1.0;
d = 0.0;
for(i=0; i<dim; i++) d += sqr(x[i]-y[i]+a[i]);
if(d<dtemp) dtemp = d;
times += 1.0;
exit = 1;
      }
      else j--;
    }
    // INV: times a été "incrémenté" une fois (càd on a calculé une
    //      distance, parmi les 3^n possibles)
  }
  // INV: dtemp = distance euclidienne au carré entre x et y sur le tore
  //      développé à partir de B
  free(a);
  return dtemp;
}

void treatPGr(const double *PGe, const double *PGr, const long nbgene,
      long *Gr, double *dmax, double *dmin, double *dmoy) {
  /*************************************************************************
  PRE:  PGr: le point-grille traité actuellement dans la boucle principale du
             programme.
  POST: Mise à jour de dmax, dmin, dmoy et Gr (selon le point-grille PGr).
  *************************************************************************/

  /*-----------------------Déclaration des variables--------------------------
    front: vaudra 0 si PGr n'est pas sur la frontière de B=[0,1]^dim
                  1 sinon
  ------------------------------------------------------------------------*/
  int i, front;
  long j, label;
  double dtemp, d;
  // double PGetmp[dim]; => FAUX ! (que vaut dim ?)  Mais c'est correct si dim
  // est passé en argument de la fonction (ds ce cas dim est bien défini) !
  double *PGetmp;
  PGetmp = (double *)malloc(dim*sizeof(double));
  front = 0;

  // PGr est-il sur la frontière de B ?
  for(i=0; i<dim; i++) {
    if( PGr[i]==0.0 || (1.0-PGr[i])<EPS ) front = 1;
  }

  // Calcul des distances entre PGr et tous les points générateurs
  // Initialisation: premier point générateur
  j = label = 0;
  dtemp = 1.0e9;
  while(j<nbgene) {
    for(i=0; i<dim; i++) PGetmp[i] = PGe[j+i];
```

```
    d = dtore2(PGr,PGetmp);
    // distance euclidienne classique (pas sur le tore):
    // d = 0.0;
    // for(i=0; i<dim; i++) d += sqr(PGr[i]-PGetmp[i]);

    // INV: d = distance euclidienne torique au carré entre le point-grille
    //      PGr et le (j+1)è point générateur
    if(d<dtemp) {
      dtemp = d;
      label = j/dim; // division entière
    }
    j += dim;
  }
  /* INV:
     label = indice (0<=label<nbcell) du point générateur le plus proche du
     point-grille PGr (=> indice de la cellule à laquelle PGr
     appartient)
     dtemp = distance torique entre le point-grille PGr et le point générateur
     le plus proche de PGr */

  // Mise à jour des distances dmax, dmin, dmoy, et de Gr
  if(dmax[label]<dtemp) dmax[label] = dtemp;
  if(dmin[label]>dtemp) dmin[label] = dtemp;
  dmoy[label] += dtemp;
  Gr[label]++;

  free(PGetmp);
}

void print(const int seed, const long *Gr, const double *Vol,
   const double *dmax, const double *dmin, const double *dmoy,
           const long k, const int timexec, const long nbcell0, const int
           sample, const double radius, const double radius_int, const double
           radius_ext, const double radius1, const double radius2, const
           double *center1, const double *center2, const double medium1, const
           double medium2, const double half_h1, const double half_h2) {
  /*************************************************************************
  PRE:
  POST: Impression de tous les résultats dans les fichiers results.dat et
        miscellaneous.dat
  *************************************************************************/

  /*------------------------Déclaration des variables-------------------------
    results.dat:      fichier contenant les résultats du programme pour toutes
    les cellules; syntaxe du fichier: sur une ligne:
    nř-cellule Gr[i] Vol[i] dmax[i] dmin[i] dmoy[i]
    miscellaneous.dat: fichier contenant des commentaires divers sur les
    résultats du programme
    ------------------------------------------------------------------------*/

  ofstream result("results.dat");
  ofstream misc("miscellaneous.dat");
  long i;

  for(i=0; i<nbcell; i++) {
    result << (i+1) << "\t\t"
   << Gr[i] << "\t\t"
   << setprecision(PR) << Vol[i] << "\t\t"
```

```
 << setprecision(PR) << dmax[i] << "\t\t"
 << setprecision(PR) << dmin[i] << "\t\t"
 << setprecision(PR) << dmoy[i] << endl;
}
result.close();
misc << "------------------Fichier miscellaneous.dat------------------\n\n"
     << "=============================================================\n"
     << "Auteur:                Denis Doumont\n"
     << "Contenu:               dVOR light - programme de calcul de\n"
     << "                       certaines caractéristiques de cellules de\n"
     << "                       Voronoi (version 'light': juste le volume)\n"
     << "Cadre:                 mémoire 2è licence mathématiques\n"
     << "Promoteur:             Marcel Rémon\n"
     << "Dernière mise à jour: 25/05/03\n"
     << "=============================================================\n";
misc << "\n";
misc << "-----------Description du programme dVOR_light----------------\n"
     << "\n"
     << "Ce programme génère des points aléatoires dans [0,1]^d.\n"
     << "Ces points peuvent suivre une distribution uniforme, ou bien une\n"
     << "distribution uniforme avec 'trous' (sphère, couronne, etc.).\n"
     << "Puis le programme calcule diverses caractéristiques\n"
     << "des cellules de Voronoi construites à partir de ces points.\n"
     << "Pour ce faire, le programme génère des points-grille selon un\n"
     << "quadrillage parfait sur [0,1]^d (tous les points-grille sont à\n"
     << "égales distances les uns des autres).  Ces points servent à\n"
     << "approximer les caractéristiques des cellules de Voronoi:\n"
     << "- l'hypervolume de ces cellules;\n"
     << "- la distance entre le point générateur de la cellule et le\n"
     << "  point-grille le plus proche;\n"
     << "- la distance entre le point générateur de la cellule et le\n"
     << "  point-grille appartenant à cette cellule et le plus éloigné du\n"
     << "  point générateur;\n"
     << "- la distance moyenne entre le point générateur de la cellule et\n"
     << "  tous les points-grille appartenant à cette cellule.\n\n";
misc << "Les résultats sont transmis dans trois fichiers:\n"
     << "- pts-generateurs.dat: tous les points générés par le générateur\n"
     << "  de nombres aléatoiresde ce programme (et qui servent à\n"
     << "  construire les cellules);\n"
     << "- results.dat: caractéristiques calculées pour toutes les "
     << "cellules;\n"
     << "- miscellaneous.dat: quelques résultats et commentaires divers;\n"
     << "  la syntaxe du fichier results.dat y est notamment décrite.\n"
     << "  Vous êtes précisément en train de le lire !\n\n";
misc << "* Note sur l'espace de travail.\n"
     << "  Nous travaillons dans l'espace torique généré à partir de\n"
     << "  [0,1]^dim: ainsi l'espace est clos sur lui-même et cela évite\n"
     << "  des effets de bord pour les cellules).  Il n'y a donc pas à\n"
     << "  proprement parler de 'cellules-frontière' ici.\n";
misc << "* Note sur le nombre de points-grille\n"
     << "  Le programme recalcule le nombre de points-grille entré par\n"
     << "  l'utilisateur de sorte que k soit bien un entier.  Le nombre\n"
     << "  de points-grille recalculé est toujours supérieur ou égal au\n"
     << "  nombre initialement entré par l'utilisateur (ainsi\n"
     << "  l'utilisateur ne doit pas se soucier d'entrer un nombre de\n"
     << "  points-grille qui 'tombe' juste avec la dimension).\n\n";
misc << "* Note sur les résultats du programme\n"
     << "  Nous insistons ici sur le fait que les résultats ne sont pas\n"
```

```
     << "  exacts, ce sont des APPROXIMATIONS: le programme génère des\n"
     << "  points-grille selon un quadrillage parfait (tous les points-\n"
     << "  grille sont à égales distances les uns des autres).  Ces\n"
     << "  points servent à approximer des grandeurs. P.ex. le volume\n"
     << "  d'une cellule est le rapport entre le nombre de points-grille\n"
     << "  appartenant à cette cellule sur le nombre total de points-\n"
     << "  grille.  Ainsi, la distance moyenne n'est pas calculée par\n"
     << "  rapport à tous les points de la cellule (il y en une infinité\n"
     << "  -> il faudrait recourir aux matrices d'inertie) mais par\n"
     << "   rapport à tous les points-grille appartenant à cette cellule."
     << "\n\n";
misc << "* Note sur la génération des nombres aléatoires\n"
     << "  La génération aléatoire uniforme est ici effectuée d'après\n"
     << "  l'algorithme de Pierre L'Écuyer, implémenté par Fabian Bastin.\n"
     << "  Les différentes réalisations du générateur de nombres sont\n"
     << "  supposées indépendantes (pas de corrélation entre les points).\n"
     << "  L'entier 'seed' (semence, graine) est un nombre donné par\n"
     << "  l'utilisateur pour lancer la génération des nombres.\n"
     << "  L'avantage de cette implémentation réside dans le fait qu'une\n"
     << "  même valeur seed génère les mêmes nombres: cela permet de\n"
     << "  retrouver des nombres précédemment générés.  Si l'utilisateur\n"
     << "  entre une valeur négative, le programme prendra seed=0.\n"
     << "  Références:\n"
     << "  - http://www.iro.umontreal.ca/~lecuyer/papers.html\n"
     << "    Publications de Pierre L'Écuyer, nř114: 'Efficient and\n"
     << "    Portable Combined Random Number Generators', Communications\n"
     << "    of the ACM, 31 (1988), 742-749 and 774.\n"
     << "  - http://www.fundp.ac.be/~fbastin/computing/rangen/index.fr."
     << "html\n"
     << "    et http://www.fundp.ac.be/~fbastin/computing/languages.fr."
     << "html\n"
     << "    Librairies de génération de nombres aléatoires de Fabian\n"
     << "    Bastin (fichiers ran-0.6.0 et ctools-0.5.1 respectivement)\n\n";
misc << "------Quelques données et résultats de l'application lancée------\n"
     << "Nombre seed: " << seed << endl
     << "Dimension de l'espace: " << dim << endl
     << "Nombre de cellules de Voronoi original: " << nbcell0 << endl
     << "Nombre de cellules recalculé (si échantillon modifié): "
     << nbcell << endl
     << "Nombre (théorique) de points-grille total: "
     << setprecision(PR) << nbgrid << endl
     << "Nombre de points-grille/dimension: " << k << endl
     << "Nombre de points-grille/cellule en moyenne: " << setprecision(PR)
     << nbgrid/(double)nbcell << endl << endl;
misc << "Type d'échantillon: ";
if(sample==0) misc << "uniforme sur B=[0,1]^" << dim << ".\n";
else if(sample==1)
  misc << "une hypersphère centrée en B=[0,1]^" << dim
<< " où les points\nsont répartis de façon uniforme.\n"
<< "Rayon de la sphère: " << radius << endl;
else if(sample==2)
  misc << "uniforme avec un trou hypersphérique au centre de B=[0,1]^"
<< dim << ".\nRayon du trou sphérique: " << radius << endl;
else if(sample==3)
  misc << "en forme de 'couronne' centrée dans B=[0,1]^" << dim << ",\n"
<< "couronne dans laquelle les points sont répartis de façon uniforme"
<< "\nRayon intérieur de la couronne: " << radius_int
<< "\nRayon extérieur de la couronne: " << radius_ext << endl;
```

```
 else if(sample==4)
   misc << "uniforme avec un trou en forme de 'couronne' centrée dans\n"
<< "B=[0,1]^" << dim <<".\n"
<< "Rayon intérieur du trou en forme de couronne: " << radius_int
<< "\nRayon extérieur du trou en forme de couronne: " << radius_ext
<< endl;
 else if(sample==5) {
   misc << "deux hypersphères incluses à\nB=[0,1]^" << dim
<< ", dans lesquelles les points sont répartis uniformément.\n";
   misc << "Centre de la première sphère:\n";
   for(i=0; i<dim; i++) misc << center1[i] << "  ";
   misc << "\nRayon de la première sphère: " << radius1 << endl;
   misc << "Centre de la deuxième sphère:\n";
   for(i=0; i<dim; i++) misc << center2[i] << "  ";
   misc << "\nRayon de la deuxième sphère: " << radius2 << endl;
 }
 else if(sample==6) {
   misc << "uniforme avec deux trous en forme d'hypersphères\n"
<< "incluses à B=[0,1]^" << dim <<".\n";
   misc << "Centre du premier trou sphérique:\n";
   for(i=0; i<dim; i++) misc << center1[i] << "  ";
   misc << "\nRayon du premier trou sphérique: " << radius1 << endl;
   misc << "Centre du deuxième trou sphérique:\n";
   for(i=0; i<dim; i++) misc << center2[i] << "  ";
   misc << "\nRayon du deuxième trou sphérique: " << radius2 << endl;
 }
 else if(sample==7) {
   misc << "deux bandes horizontales disjointes incluses\n"
<< "à B=[0,1]^" << dim << ", dans lesquelles les points sont\n"
<< "répartis uniformément.\n";
   misc << "Coordonnée du milieu de la première bande: " << medium1
<< "\nDemi-hauteur de la première bande: " << half_h1 << endl;
   misc << "Coordonnée du milieu de la deuxième bande: " << medium2
<< "\nDemi-hauteur de la deuxième bande: " << half_h2 << endl;
 }
 misc << "\nTemps d'exécution des calculs: " << timexec << " s\n";
 misc << "\nTous les autres résultats sont présentés dans le fichier\n"
      << "results.dat\n\n";
 misc << "----------------Syntaxe du fichier results.dat----------------\n";
 misc << "Sur une ligne se trouvent les grandeurs:\n"
      << "nř-de-la-cellule  Gr[i]  Vol[i]  dmax[i]  dmin[i]"
      <<  "  dmoy[i]\n\n"
      << "Gr[i]\tnombre de points-grille se trouvant dans la cellule i\n"
      << "Vol[i]\thypervolume de la cellule i\n"
      << "dmax[i]\tdistance entre le point générateur de la cellule i et le\n"
      << "\tpoint-grille appartenant à la cellule i, le plus éloigné de ce\n"
      << "\tpoint générateur\n"
      << "dmin[i]\tdistance entre le point générateur de la cellule i et le\n"
      << "\tpoint-grille le plus proche de ce point\n"
      << "dmoy[i]\tdistance moyenne entre le point générateur de la cellule i"
      << "\n\tet tous les points-grille présents dans cette cellule (ceci est"
      << "\n\tune approximation de l'inertie de la cellule i)\n\n"
      << "\t\t\t\tTHE END\n\n";
  misc.close();
}
/*================================THE END================================*/
```

Annexe B

Détails informatiques

B.1 Informations sur les machines utilisées pour les programmes

Type de processeur	Intel Pentium II Celeron (Mendocino)
Fréquence du CPU	300.708A MHz
Mémoire vive	256 Mo
Mémoire cache	128 ko sur le processeur et 128 ko sur la carte mère
Date de l'AMIBIOS	15 juillet 1995
Système d'exploitation	Linux RedHat 8.0
Version du compilateur `gcc`	v3.2 20020903 (3 septembre 2002)

TAB. B.1: *Quelques informations sur une des machines utilisées*

Type de processeur	Intel Pentium III (Coppermine)
Fréquence du CPU	803.417 MHz
Mémoire vive	512 Mo
Mémoire cache	256 ko
Système d'exploitation	Linux Suse 7.0 maisa avec un noyau 2.4.20
Version du compilateur `gcc`	v2.95.3 20010315 (15 mars 2001)

TAB. B.2: *Quelques informations sur une des machines utilisées (machine* `sphere` *du département de mathématiques)*

Type de machine	Intel Pentium II (Deschutes)
Fréquence du CPU	334.096 MHz
Mémoire vive	256 Mo
Mémoire cache	512 ko
Système d'exploitation	Linux Suse noyau 2.4.10
Version du compilateur `gcc`	v2.95.3 20010315 (15 mars 2001)

TAB. B.3: *Quelques informations sur une des machines utilisées (machines* `math-asa` *du département de mathématiques)*

B.2 Limites définies par l'implémentation

Voici quelques informations sur les constantes pour les tailles des types entiers et réels du C/C++.

Taille en octets du type `char`	1
Taille en octets du type `short`	2
Taille en octets du type `int`	4
Taille en octets du type `long`	4
Taille en octets du type `float`	4
Taille en octets du type `double`	8
Taille en octets du type `long double`	12

TAB. B.4: *Les différentes tailles mémoire des types entiers et réels en C/C++ sur la machine décrite à la table B.1*

Les deux tableaux suivants sont issus de [8, p. 264]. Les champs « v. machine » et « v. min. » signifient « valeur sur la machine définie à la table B.1 » et « valeur minimale acceptable (par le C ANSI) ». On notera que les constantes `UCHAR_MIN`, `UINT_MIN`, `ULONG_MIN` et `USHRT_MIN` n'existent pas. C'est que, par définition, la borne inférieure pour les types entiers non signés est toujours 0.

nom	v. machine	description	v. min.
`CHAR_BIT`	8	nombre de bits par caractère	8
`CHAR_MAX`	127	valeur maximale d'un `char`	127 ou 255
`CHAR_MIN`	- 128	valeur minimale d'un `char`	−127 ou 0
`INT_MAX`	2 147 483 648	valeur maximale d'un `int`	32 767
`INT_MIN`	- 2 147 483 647	valeur minimale d'un `int`	−32 767
`LONG_MAX`	2 147 483 647	valeur maximale d'un `long`	2 147 483 647
`LONG_MIN`	- 2 147 483 648	valeur minimale d'un `long`	−2 147 483 647
`SCHAR_MAX`	127	valeur maximale d'un `signed char`	127
`SCHAR_MIN`	- 128	valeur minimale d'un `signed char`	−127
`SHRT_MAX`	32 767	valeur maximale d'un `short`	32 767
`SHRT_MIN`	- 32 768	valeur minimale d'un `short`	−32 767
`UCHAR_MAX`	255	valeur maximale d'un `unsigned char`	255
`UINT_MAX`	4 294 967 295	valeur maximale d'un `unsigned int`	65 535
`ULONG_MAX`	4 294 967 295	valeur maximale d'un `unsigned long`	4 294 967 295
`USHRT_MAX`	65 535	valeur maximale d'un `unsigned short`	65 535

TAB. B.5: *Constantes pour les tailles des types entiers du C/C++, définies dans le fichier d'en-tête* `<limits.h>`

Bibliographie

[1] *S-plus 2000, aide en ligne.* MathSoft, 1988-1999.

[2] ARCHAMBAULT Sylvain. *Tests basés sur des opérateurs morphologiques pour l'ajustement d'images.* Département de Mathématiques appliquées, École polytechnique de Montréal, Université de Montréal, 1990. Thèse présentée en vue de l'obtention du grade de Philosophiæ Doctor (Ph. D.) (Mathématiques appliquées).

[3] BAYART Benjamin. *Joli manuel pour LaTeX 2ε.* Disponible sur la page http://www.fundp.ac.be/~bcolson, 1995.

[4] GELMAN Andrew, CARLIN John B., STERN Hal S., and RUBIN Donald B. *Bayesian Data Analysis.* Chapman & Hall, London, 1995. ISBN 0-412-03991-5.

[5] HARDY André. *Statistiques.* Librairie des Sciences, Namur, 2001. Cours de deuxième candidature en Sciences mathématiques FUNDP.

[6] HARDY André. *Aspects statistiques de la classification.* Librairie des Sciences, Namur, 2002. Cours de deuxième licence en Sciences mathématiques FUNDP.

[7] KENDALL Maurice G. and STUART Alan. *The advanced theory of statistics*, volume 1 : *Distribution theory.* Charles Griffin & Company limited, London, troisième edition, 1969. Édition en trois volumes, SBN 85264-141-9.

[8] KERNIGHAN Brian W. and RITCHIE Denis M. *Le langage C – Norme ANSI.* Masson, Paris, deuxième edition, 1997. ISBN 2-225-83035-5.

[9] MICHELOUD Marylène and RIEDER Medard. *Programmation orientée objets en C++ : une approche évolutive.* Presses polytechniques et universitaires romandes, Lausanne, seconde edition, 2002. Édition entièrement revue et augmentée, ISBN 2-88074-504-7.

[10] OETIKER Tobias, PARTL Hubert, HYNA Irene, and SCHLEGL Elisabeth. *Une courte (?) introduction à LaTeX 2ε.* http://www.univ-lille1.fr/labo-stat-proba/df/doc/dfshort.pdf, 2003. Traduit par Mathieu Herrb, adaptation et compléments par Daniel Flipo, version 4.0c, document libre.

[11] OKABE Atsuyuki, BOOTS Barry, SUGIHARA Kokichi, and CHIU Sung Nok. *Spatial Tessellations : Concepts and Applications of Voronoi Diagrams.* John Wiley & sons Ltd, Baffins Lane, Chichester, West Sussex, England, 2000. ISBN 0-471-98635-6.

[12] PARIS José. *Probabilités–Statistiques.* SICI, Louvain-la-Neuve, 1997. Cours de deuxième candidature ingénieur civil UCL.

[13] PRESS William H., TEULOSKY Saul A., VETTERLING William T., and FLANNERY Brian P. *Numerical Recipes in C – The Art of Scientific Computing.* Cam-

bridge University Press, Cambridge, seconde edition, 1992. http://www.nr.com, ISBN 0-521-43108-5.

[14] RASSON Jean-Paul. *Calcul des probabilités, théorie des erreurs d'observation et éléments de statistiques.* Namur, 2000. Cours de première licence en Sciences mathématiques FUNDP.

[15] RIPLEY Brian D. *Stochastic Simulation.* John Wiley & Sons, New York, 1987. ISBN 0-471-81884-4.

[16] SPIEGEL Murray R. *Formules et tables de mathématiques.* McGraw-Hill, New-York, Paris pour la traduction française, 1974. ISBN France 2-7042-0025-4.

[17] TANEMURA Masaharu. "Random Voronoi cells of higher dimensions". http://members.tripod.com/vismath7/proceedings/tanemura.htm, Tanemura est statisticien à l'Institute of Statistical Mathematics, Tokyo, Japon.

[18] VANDOOREN Julie. *Tests d'uniformité et tessellations de Voronoi.* Département de Mathématique, Faculté des Sciences, FUNDP Namur, 2002. Mémoire présenté pour l'obtention du grade de Licencié en Sciences Mathématiques.

Liste des errata

Denis Doumont

26 juin 2003

Le présent document a pour but de relever et de corriger *tous* les *errata* présents dans la version officielle du mémoire, intitulé *Tests d'uniformité de nuages de points en grande dimension basés sur les diagrammes de Voronoi.* Ne sont pas relevées ici les fautes de frappe, d'orthographe et de grammaire.

p. 4

- Notons encore que le processus binomial ne vérifie pas la *partie (i) de la définition 1.5*, car si x points [...]

p. 13

- **Hypothèse 2 (Non-cocircularité)**
  [...] *tel que C passe par les points $p_{i_1}, \ldots, p_{i_k} \in P \quad (k \geq 4)$ et tous les points dans $P \setminus \{p_{i_1}, \ldots, p_{i_k}\}$ sont en dehors de C.*
  [Les notations $p_{i_1}, \ldots, p_{i_k}$ nous semblent plus explicites que $p_{i1} \ldots, p_{ik}$.]

p. 16

- Légende de la figure 1.10 : *Histogramme normalisé de l'aire* réduite *d'une cellule-type de Voronoi de Poisson dans $\mathbb{R}^2$;* les points représentent *des valeurs de la fonction (1.16) avec les paramètres estimés de Hinde et Miles* [L'aire *réduite* est définie par $\alpha = \lambda A$, où λ est l'intensité du processus Θ_P et A l'aire de la cellule.]
- Les valeurs des paramètres en une dimension sont bien connues *(voir chapitre suivant).*

p. 17

- Légende de la figure 1.11 : [...] $\circ$: 2-D (courbe la plus haute) ; $\triangle$: 3-D (courbe au milieu) ; $\square$: 4-D (courbe du bas). [...]

p. 18

- Légende de la figure 1.12 : *Histogramme* normalisé *du volume d'une cellule-type de Voronoi de Poisson* [...]

p. 23

- Section 2.3.1 : Étude à partir de la longueur des cellules
  Section 2.3.1.1 : Densité de la longueur des cellules en terme de Bêta
  Section 2.3.1.2 : Densité de la longueur des cellules
- Cela correspond à la fonction (2.20) *lorsque $n = 3$ (ce qui est logique).*

- Or, l, la longueur d'une cellule de Voronoi, est égale à $\frac{X_{(i+1)}-X_{(i-1)}}{2} = \frac{T}{2}$ *($i \neq 1$, $i \neq n$, cf. équation (2.24), plus loin).*
- **Ajout.** *La fonction (2.22) est donc la fonction de densité correspondant à deux fois la longueur d'une cellule de Voronoi de Poisson en une dimension. On remarque que cette densité ne dépend pas de i.*
  Cherchons alors la fonction de densité de la VA $L = \frac{T}{2}$. *Nous trouvons par calculs :*

$$f_L(l) = \begin{cases} 4n(n-1)l(1-2l)^{n-2} & 0 \leq l \leq \frac{1}{2} \\ 0 & \text{ailleurs} \end{cases}. \quad (1)$$

p. 24

- On sait *(voir équation (2.23))* que, pour la VA L, longueur d'une cellule, $2L \sim \mathcal{B}(2, n-1)$. *Pour $n = 3$, nous obtenons*

$$f_{2L}(t) = \begin{cases} 6t(1-t) & 0 \leq t \leq 1 \\ 0 & \textit{ailleurs} \end{cases}.$$

  Par calculs, nous trouvons

$$f_L(l) = \begin{cases} 24l(1-2l) & 0 \leq l \leq \frac{1}{2} \\ 0 & \textit{ailleurs} \end{cases}.$$

  Cela correspond d'ailleurs, sans surprise, à l'équation (1)*, avec $n = 3$.*

p. 25

- Section 2.3.1.3 : Densité de la longueur des cellules en terme de Gamma

p. 27

- Remarquons que $\lambda = \mathbf{n+1}$ (ou $\mathbf{n}$) veut dire qu'on attend $n+1$ points (ou n points) sur un segment de taille unité. [Effectivement, puisque $\mathrm{E}[Y] = 1/\lambda$, que Y représente la longueur d'une cellule, et qu'il y a autant de cellules que de points générateurs (points de l'échantillon).]
- [...] En effet, *nous avons* :

$$Y \sim \Gamma(2\lambda, 2) \Rightarrow f_Y(y) = 4y\lambda^2 e^{-2\lambda y}$$
$$\text{et } 2L \sim \mathcal{B}(2, n-1) \Rightarrow f_L(l) = 4n(n-1)l(1-2l)^{n-2}.$$

  En posant $Y = L$ et $y = l$ (l est la longueur d'une cellule de Voronoi), en se souvenant que $\lambda = \mathbf{n}$, on a, pour $n \to \infty$ et $l \to 0$:

$$\lim_{n\to\infty\,;\, l\to 0} f_L(l) = \lim_{n\to\infty\,;\, l\to 0} 4\mathbf{n^2} l(1-2l)^{n-2}$$

p. 57

- Légende de la figure 7.5 : *Histogramme et box plot de* `zvol` *(4-D)*

p. 58

- [...] et le test-χ^2 valide l'hypothèse nulle (`zvol` $\sim N(0,1)$) pour le seuil de signification $\alpha = 0.05$: $\chi^2_{24,\,0.95} = 36.4 > \chi^2_{\mathbf{obs}}$.

p. 59

- Légende de la figure 7.7 : *Histogramme et box plot de* `zvol` *(5-D)*

p. 60

- Nous avons dans les tables de la loi χ^2 la valeur $\chi^2_{40,\,0.95} = 55.8 < \chi^2_{\text{obs}}$ [...]
- Légende de la figure 7.9 : *Histogramme et box plot de* `zvol` *(6-D)*

p. 61

- La statistique χ^2 est en dessous du seuil $\chi^2_{28,\,0.95} = \mathbf{41.3}$ [...]

p. 67

- Le paragraphe « Nous mettons ici... » doit être placé *après* le titre « 8.1.2 Échantillon 2-D en forme de disque centré » : il se rapporte en effet à ce test.

p. 68

- Le paragraphe « Ici aussi,... » doit être placé *après* le titre « 8.1.3 Échantillon 2-D avec un trou en forme de disque centré » : il se rapporte en effet à ce test.

p. 70

- Le paragraphe « Les graphes les plus significatifs... » doit être placé *après* le titre « 8.1.4 Échantillon 2-D avec une couronne centrée » : il se rapporte en effet à ce test.

p. 87

- À la table *8.21*, « toutes » signifie « toutes les cellules » [...]

p. 88

- Nous avons dû prendre 5 000 cellules au départ pour n'en garder que 643 (voir table *8.19*).

p. 89

- On se souviendra que l'on n'a enlevé que 10% des cellules pour ce test (table *8.19*).

p. 90

- [...] que le fait d'ôter les 40 plus grandes cellules permet de retrouver la normalité pour `zvol` (et donc, par induction, *une loi Gamma pour l'hypervolume des cellules*). [Mais cela n'est pas de première importance : nous travaillons exclusivement avec la variable `zvol` pour les dimensions supérieures à trois, ainsi l'essentiel est de montrer que l'assertion « `zvol` $\sim N(0,1)$ » est un bon candidat pour un test d'uniformité dans ces dimensions.]

p. 93

- Notons enfin que la comparaison n'est pas totale avec le trou sphérique : [...] contre 10% pour le trou sphérique (cf. table *8.19*).