Rapport de stage

Analyse exploratoire des données spatialisées en utilisant des processus ponctuels

Clément Dugué

Année 2016–2017

Stage de deuxième année réalisé dans l'entreprise Institut Elie Cartan de Lorraine

Maître de stage : Radu STOICA

Encadrant universitaire : Sébastien DA SILVA

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Dugué, Clément

Élève-ingénieur(e) régulièrement inscrit(e) en 2e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 2406037702x

Année universitaire : 2016–2017

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Analyse exploratoire des données spatialisées en utilisant des processus ponctuels

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Nancy, le 24 août 2017

Signature :

Rapport de stage

Analyse exploratoire des données spatialisées en utilisant des processus ponctuels

Clément Dugué

Année 2016–2017

Stage de deuxième année réalisé dans l'entreprise Institut Elie Cartan de Lorraine

Clément Dugué
46 rue de Laxou
54000, NANCY
06 11 68 21 32
clement.dugue@telecomnancy.net

TELECOM Nancy
193 avenue Paul Muller,
CS 90172, VILLERS-LÈS-NANCY
+33 (0)3 83 68 26 00
contact@telecomnancy.eu

Institut Elie Cartan de Lorraine
Campus Sciences BP 70239
54506, VANDOEUVRE LES NANCY
03 72 74 54 19

Maître de stage : Radu STOICA

Encadrant universitaire : Sébastien DA SILVA

Remerciements

En premier lieu, je tiens à remercier M. Xavier ANTOINE directeur de l'institut Elie Cartan et M. Antoine Lejay responsable de l'équipe de Probabilités Statistiques pour m'avoir accueilli dans le laboratoire.

J'adresse également mes remerciements à mon maître de stage M. Radu STOICA pour ses explications, son soutien et son écoute tout au long du stage, ainsi que pour la confiance qu'il m'a accordée en m'acceptant comme stagiaire.

Je souhaite aussi remercier mon encadrant universitaire M. Sébastien DA SILVA qui a participé au bon déroulement de ce stage.

Je remercie enfin toutes les personnes m'ayant aidé dans l'écriture et la relecture de ce rapport de stage.

Table des matières

Remerciements **v**

Table des matières **vii**

1 Introduction **1**

2 Présentation de l'entreprise d'accueil **2**

2.1 Institut Elie Cartan 2
2.2 L'IECL 2
2.3 L'équipe Probabilités et Statistiques 3

3 Problématique : présentation du problème **4**

3.1 Contexte 4
3.2 Intensité d'un processus ponctuel 4
3.3 L'analyse de processus ponctuels 6
3.4 La fonction K 6
3.4.1 Principe 6
3.4.2 La fonction K empirique 6
3.4.3 La fonction K pour un processus de Poisson stationnaire 7
3.4.4 Interprétation de K 7
3.5 Fonction d'espace vide F 8
3.5.1 Définitions pour un processus de point stationnaire 8
3.5.2 La fonction F pour un processus de Poisson stationnaire 8
3.5.3 Estimation de F 8
3.6 La fonction G du plus proche voisin 9
3.6.1 Définitions pour un processus de point stationnaire 9
3.6.2 Valeurs pour un aléatoire complet 10
3.6.3 Estimation de G 10
3.6.4 Interprétation de G 10

3.7 Fonction J 11
3.8 Effet des bords 11
3.8.1 Principe 11
3.8.2 Correction pour K 12
3.8.3 Correction pour F 12
3.8.4 Correction pour G 12
3.9 Test d'enveloppe 12

4 Réalisation **14**
4.1 Environnement logiciel 14
4.2 Méthode de travail 14
4.3 Structure du code 14
4.3.1 Création des abscisses 15
4.3.2 Les classes 16
4.3.3 Algorithmes de calcul des fonctions 18
4.3.4 Les listes 18
4.3.5 Les tris 19
4.4 Stratégies d'implémentations adaptées au problème 19
4.4.1 Le remplissage des listes des distances 19
4.4.2 Les distances aux carrés 20
4.4.3 Arrêter les calculs et stockage au rayon maximal 20
4.5 Tests d'enveloppe 21
4.6 Intensité 21
4.6.1 Calcul 21
4.6.2 Correction des bords 22

5 Bilan **23**
5.1 Résultats obtenus 23
5.1.1 Validation 23
5.1.2 Visualisation 23
5.1.3 Utilisation 24
5.2 Difficultés et Solutions 24
5.3 Devenir du projet 24

6 Conclusion **25**

Bibliographie / Webographie **27**

Liste des illustrations **29**

Liste des tableaux **31**

Listings **33**

Glossaire **36**

Annexes 38

A Diagramme de Classes **39**

B Algorithmes des calculs des fonctions **41**

C Représentations sous R **47**

Résumé **55**

Abstract **55**

1 Introduction

Lors de l'étude d'un phénomène (localisation d'évènements historiques, de micro-organismes, de populations...), les données spatialisées synthétisent l'information relative à la position. On peut représenter ces données sous forme de points. On obtient ainsi une représentation des données sur laquelle il est plus facile de raisonner. Par l'analyse statistique, on peut alors identifier des tendances selon la densité des points (si les points sont rapprochés ou non). Les résultats de l'analyse peuvent ensuite être interprétés pour en tirer des conclusions, et fournir ainsi des informations objectives utiles pour de nombreux domaines.

L'automatisation de cette analyse se fait par la création de programmes informatiques qui peuvent lire les données fournies et faire des calculs rapides. Cependant il n'y a pas de solution toute faite pour l'analyse statistique de répartition de points, en effet chaque cas doit être analysé en fonction de son contexte (obtention des données, objectif de l'analyse). D'où le besoin de fonctions et de programmes multiples pour cette analyse.

Ce fut le but de ce stage : créer des outils informatiques pour traiter et analyser des données ponctuelles selon des méthodes mathématiques fournies et expliquées par mon tuteur. La réalisation des fonctions se faisait en plusieurs étapes : d'abord la lecture et compréhension de la documentation sur la fonction, puis la conception et l'écriture du code, et enfin une analyse des résultats obtenus en vue d'une amélioration (retour à la 2ème étape). Ainsi, après une présentation globale de l'entreprise d'accueil, la structure de ce rapport suivra l'enchaînement des étapes présentées ci-dessus.

Enfin les termes spécifiques et définitions, seront réunies dans le glossaire (page 30). Dans le texte, un * renvoie à ce glossaire.

2 Présentation de l'entreprise d'accueil

2.1 Institut Elie Cartan

La recherche lorraine en mathématiques est présente en lorraine depuis l'arrivée d'Élie Cartan il y a une centaine d'année. Elle a été marquée par une succession de personnalités de renommée mondiale comme Jean Leray, Laurent Schwartz (Médaille Fields 1950), Jean-Pierre Serre (Médaille Fields 1954), Roger Godement, Jean Delsarte (secrétaire du groupe Bourbaki jusqu'en 1962) ou Jacques-Louis Lions.

Créé en 1953 à l'initiative de Jean Delsarte, l'Institut Élie Cartan de Nancy a été reconnu par le CNRS en 1978 (sous le nom d'Équipe d'Analyse Globale) et il a fusionné avec le Laboratoire de Mathématiques et Applications de Metz au premier janvier 2013. Le nouveau laboratoire est intitulé Institut Elie Cartan de Lorraine (IECL),

2.2 L'IECL

D'après son site internet [5], l'IECL est une unité mixte de recherche (UMR 7502) du CNRS et de l'Université de Lorraine. L'INRIA est un partenaire important et de longue date de l'IECL, notamment à travers les équipes-projets INRIA que nous hébergeons.

Avec environ 120 enseignants-chercheurs et chercheurs permanents, l'Institut Élie Cartan de Lorraine est l'un des grands laboratoires français en mathématiques et le plus grand de l'Est de la France.

Dans plusieurs domaines des mathématiques fondamentales (géométrie, théorie des nombres), des équations aux dérivées partielles ou des probabilités, l'IECL rassemble des spécialistes reconnus internationalement. Cette diversité scientifique favorise les interactions internes et externes, en contribuant à l'attractivité pour les visiteurs étrangers.

2.3 L'équipe Probabilités et Statistiques

L'équipe de Probabilités et Statistique est l'une des quatre équipes de l'IECL. Forte d'une trentaine de membres, elle accueille en son sein les équipes projet INRIA Bigs et Tosca.

Son effectif relativement important (une trentaine de permanents) en fait une des plus grosses équipes du territoire français. Cela lui permet de couvrir l'essentiel du spectre de la recherche contemporaine en probabilités et statistique, comme par exemple l'étude de structures probabilistes discrètes (graphes, arbres), la modélisation et estimation avec des applications par exemple à la biologie, la médecine, la finance, le comportement en temps long des processus stochastiques, les trajectoires rugueuses, la simulation Monte Carlo.

De nombreux membres sont impliqués dans des collaborations inter-disciplinaires en France ou à l'étranger, et/ou industrielles.

3 Problématique : présentation du problème

3.1 Contexte

Étant membre de l'équipe Proba-Stats, mon maître de stage M. Radu Stoica étudie les processus ponctuels* (répartitions de points). Ayant réalisé en langage C et C++ des fonctions simulant des répartitions de points, il souhaitait alors pouvoir analyser ces processus.

Des librairies d'analyse existent sous R comme la librairie spatstat*, mais sont limitées : ces méthodes ne sont pas applicables pour des jeux de données plus larges. Ainsi mon maître de stage souhaitait avoir ces librairies d'analyse en C et C++ afin de pouvoir gérer des données plus complexes. De plus cette implémentation permet d'envisager une extension en trois dimensions qui n'est pas encore mise en place dans la librairie spatstat*. Il sera alors possible d'étudier des modèles plus réalistes.

J'ai eu à lire et interpréter la partie théorique au fur et à mesure des créations des fonctions. Dans un soucis de clarté, je vais commencer par expliquer l'intensité d'un processus ponctuel*.

3.2 Intensité d'un processus ponctuel

Pour un processus ponctuel*, l'intensité donne une mesure du nombre moyen de points sur une surface.

Soit X un processus ponctuel* sur $\mathbb{R}^2$ et soit **x** une réalisation* de X observé dans $W \subseteq \mathbb{R}^2$. Pour chaque sous-région B de W, on s'intéresse à la formule :

$$\mathbb{E}[n(\mathbf{x} \cap B)] = \int_B \lambda(u)\,\mathrm{d}u$$

Ainsi si l'intensité est homogène* sur W, le nombre de points de **x** dans B est censé être proportionnel à l'aire de B :

$$\mathbb{E}[n(\mathbf{x} \cap B)] = \lambda|B|$$

où λ est une constante qu'on appelle "intensité".

L'intensité λ est le nombre de points prévus dans chaque sous-région B. Si X est homogène* on peut alors estimer λ telle que :

$$\hat{\lambda} = \frac{n}{|W|}$$

avec n le nombre de points présents dans W.

Cependant si l'intensité n'est pas homogène*, il faut l'estimer différemment. Une méthode est l'estimation par noyau. Le principe est de quadriller la surface W et pour chaque maille u du quadrillage de chercher l'influence qu'elle amène en étudiant la densité de point dans son voisinage avec une fonction de noyau k :

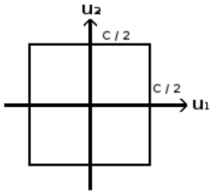
$$\hat{\lambda}(u) = \sum_{i=1}^{n} k(u - x_i)$$

La fonction de noyau k est une densité de probabilité, ainsi pour toute maille u du quadrillage la fonction doit respecter les propriétés :

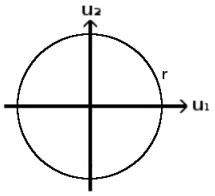
- $k(u) \geq 0$
- $\int_{\mathbb{R}^2} k(u)\,\mathrm{d}u = 1$

Il y a alors plusieurs manières de définir la fonction noyau, on s'intéressera pour l'implémentation aux fonctions suivantes.

Noyau carré : $k(u_1, u_2) = \frac{1}{c^2}\,\mathbb{1}\{|u_1| < c/2\ ,\ |u_2| < c/2\}$

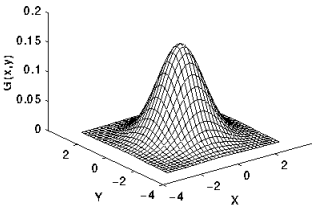


Noyau disque : $k(u_1, u_2) = \frac{1}{\pi r^2}\,\mathbb{1}\left\{\sqrt{u_1^2 + u_2^2} \leq r\right\}$



Noyau gaussien : $k(u_1, u_2) = \frac{1}{2\pi\sigma}\ exp\left[-\frac{u_1^2 + u_2^2}{2\sigma^2}\right]$

(image prise sur le site [7])



Enfin, des défauts d'estimation vont apparaitre lorsque l'on s'approche des bords de W : les effets de bord. En effet les points de **x** en dehors de la surface W étudiée ne seront pas pris en compte. Afin de remédier à ce problème, on peut appliquer la correction de Diggle. En notant, pour chaque point de la surface W, la correction à appliquer :

$$e(x_i) = \int_W K(u - v)\,\mathrm{d}v$$

La correction de Diggle consiste à compenser les effets de bords pour chaque point tel que :

$$\hat{\lambda}(u) = \sum_{i=1}^{n} \frac{1}{e(x_i)} k(u - x_i)$$

3.3 L'analyse de processus ponctuels

Dans le cadre de l'analyse des processus ponctuels*, on identifie de manière générale trois grandes tendances représentées dans la figure 3.1 :

— répulsive (les points se trouvent à une certaine distance les uns des autres) ;

— aléatoire (les points n'ont pas de dépendance) ;

— agrégée (les points forment des groupes de points).

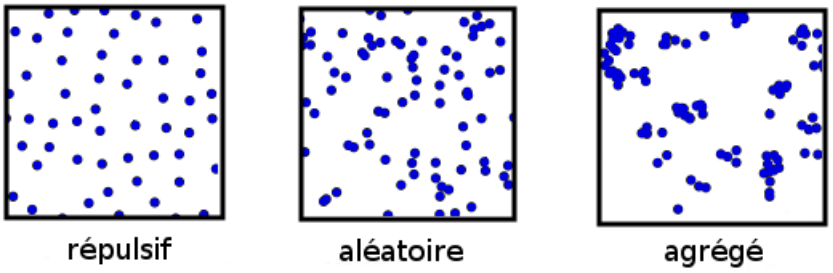


FIGURE 3.1 – Représentation des différentes formes de représentations spatiales de points

Une manière statistique de déterminer si une observation de points correspond à l'un ou l'autre de ces trois cas, est de mesurer son écart avec un processus ponctuel de Poisson*. En effet ce type de processus ponctuel* à la propriété de répartir ses points d'une manière totalement aléatoire.

3.4 La fonction K

3.4.1 Principe

Supposons qu'on se pose une question sur l'espacement entre les points dans un ensemble de points. Il serait alors naturel de regarder les distances entre chaque point. Si ces distances sont plutôt grandes, la répartition des points sera plutôt répulsive. Alors que si elles sont plutôt petites, la répartition des points sera plutôt agrégée.

3.4.2 La fonction K empirique

Soit X un processus ponctuel* stationnaire* sur $\mathbb{R}^2$ d'intensité λ et soit **x** une réalisation* de X observée dans $W \subseteq \mathbb{R}^2$. Les distances $d_{ij} = ||x_i - x_j||$ entre chaque point distinct x_i et x_j de **x** constituent une statistique descriptive*. La fonction K empirique est définie pour chaque distance $r \geq 0$ par la formule :

$$\hat{K}(r) = \frac{1}{n\lambda} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \mathbb{1}\{d_{ij} \leq r\}$$

La fonction indicatrice $\mathbb{1}\{...\}$ vaut 1 si son contenu est vrai et 0 sinon. La somme de ces indicatrices est simplement le nombre de fois où le contenu est vrai.

La fonction K est une moyenne normalisée du nombre de voisins dans une boule de rayon r, sans compter le point lui même. Ces nombres de points voisins sont représentés sur la figure 3.2. La normalisation est représentée dans la formule par le facteur $1/\lambda$. Comme vu dans son paragraphe dédié, l'intensité du processus λ est une mesure du nombre moyen de point dans W. On peut ici l'estimer par |W|/(n-1).

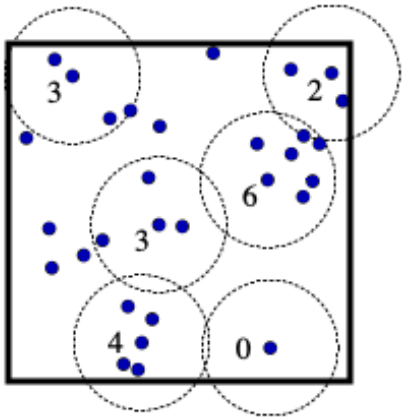


FIGURE 3.2 – Nombre de points voisins dans un certain rayon.

3.4.3 La fonction K pour un processus de Poisson stationnaire

En suivant la partie 7.3.2 du livre *Spatial Point Patterns - Methodology and Applications with R* [8], on peut montrer que la fonction estimée par $\hat{K}(r)$ est donnée par :

$$K_{pois}(r) = \pi r^2$$

pour un processus de Poisson* stationnaire*.

La figure 3.2 montre alors la représentation en fonction de r de la fonction K théorique.

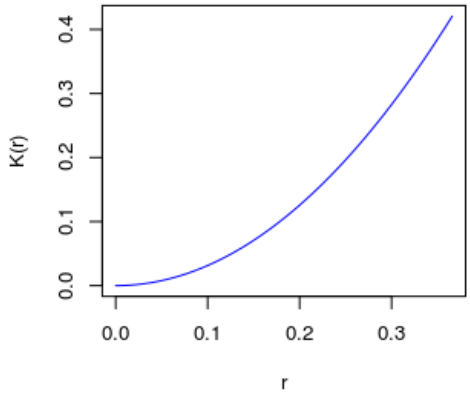


FIGURE 3.3 – Représentation de la fonction K pour une répartition totalement aléatoire.

3.4.4 Interprétation de K

La figure 3.4 ci dessous montre la fonction estimée de K pour les 3 configurations de la section 3.3 (figure 3.1) selon que la configuration est agrégée, aléatoire ou bien répulsive. Les courbes rouges représentent $\hat{K}$ tandis que les courbes bleues discontinues représentent K_{pois}.

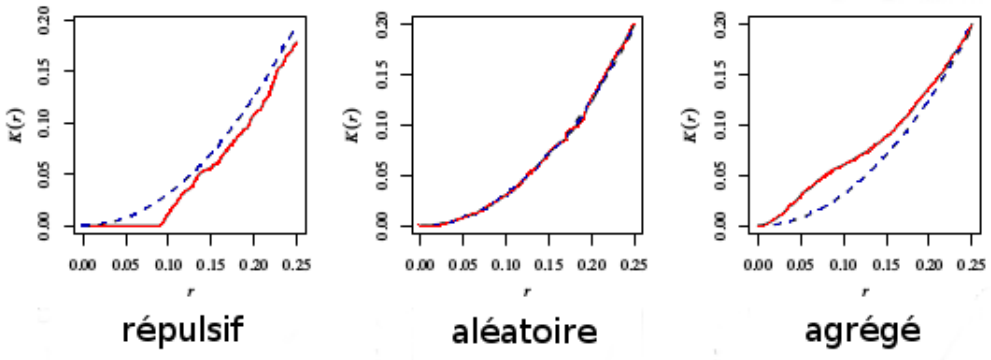


FIGURE 3.4 – Différentes estimations de K.

Sur la gauche, la courbe de la fonction calculée est en dessous de celle pour une répartition totalement aléatoire ($\hat{K}(r) < K_{pois}(r)$). Ainsi, pour une distance r, un point quelconque a en moyenne

moins de voisins qu'il aurait pu espérer avoir avec une répartition totalement aléatoire. Donc la répartition semble plutôt répulsive.
Sur la droite, la courbe de la fonction calculée est au dessus de celle pour une répartition totalement aléatoire ($\hat{K}(r) > K_{pois}(r)$). Ainsi, pour une distance r, un point quelconque a en moyenne plus de voisins qu'il aurait pu espérer avoir avec une répartition totalement aléatoire. Donc la répartition semble plutôt agrégée.

3.5 Fonction d'espace vide F

3.5.1 Définitions pour un processus de point stationnaire

Soit X un processus ponctuel* sur $\mathbb{R}^2$ d'intensité λ et soit **x** une réalisation* de X observée dans $W \subseteq \mathbb{R}^2$. On note d(u,**x**) la distance d'une position $u \in \mathbb{R}^2$ au plus proche point de **x**. Cette fonction est appelé 'distance d'espace vide' et est définie par :

$$d(u, \mathbf{x}) = min\{||u - x_i|| : x_i \in \mathbf{x}\}$$

La fonction F de distance d'espace vide est définie pour toute distance $r \geq 0$ par :

$$F(r) = \mathbb{P}\{d(u, \mathbf{x}) \leq r\}$$

Les valeurs de F(r) sont des probabilités, elles sont comprises entre 0 et 1. Elles définissent les chances de trouver un point dans un disque de rayon r sur la surface étudiée.

3.5.2 La fonction F pour un processus de Poisson stationnaire

Pour un processus de Poisson* stationnaire* d'intensité λ, on peut calculer :

$$\begin{aligned} F_{pois}(r) &= \mathbb{P}\{d(u, \mathbf{x}) \leq r\} \\ &= \mathbb{P}\{n(b(u, r) \cap \mathbf{x}) \neq 0\} \\ &= 1 - \mathbb{P}\{n(b(u, r) \cap \mathbf{x}) = 0\} \\ &= 1 - exp(-\lambda \pi r^2) \end{aligned}$$

car par stationnarité, F(r) ne dépend pas de u. La figure 3.5 montre la représentation en fonction de r de cette fonction théorique.

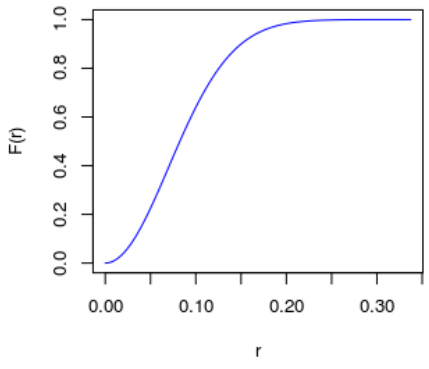


FIGURE 3.5 – Représentation de la fonction de répartition de Poisson pour λ = 40.

3.5.3 Estimation de F

Pour estimer la fonction F à partir de la réalisation* **x** finie, on doit discrétiser pour obtenir les valeurs de u considérées. Ainsi pour n'importe quelle position *u* sur la surface W à analyser, on

mesure la distance *d(u,***x***)* la séparant du point voisin le plus proche (distance d'espace vide). On répète alors cette opération pour un nombre m de positions u_1,...,u_m équitablement réparties sur la surface. On peut alors calculer la fonction empirique comme fonction sur la distance $r \geq 0$:

$$\hat{F}(r) = \frac{1}{m} \sum_{j=1}^{m} \mathbb{1}\{d(u_j, \mathbf{x}) \leq r\}$$

Interprétation de F

La figure 3.6 ci dessous montre la fonction estimé de F pour les 3 configurations de la section 3.3 (figure 3.1) selon que la répartition est agrégée, aléatoire ou bien répulsive. Les traits discontinus sont la représentation graphique de la fonction F théorique pour un aléatoire total.

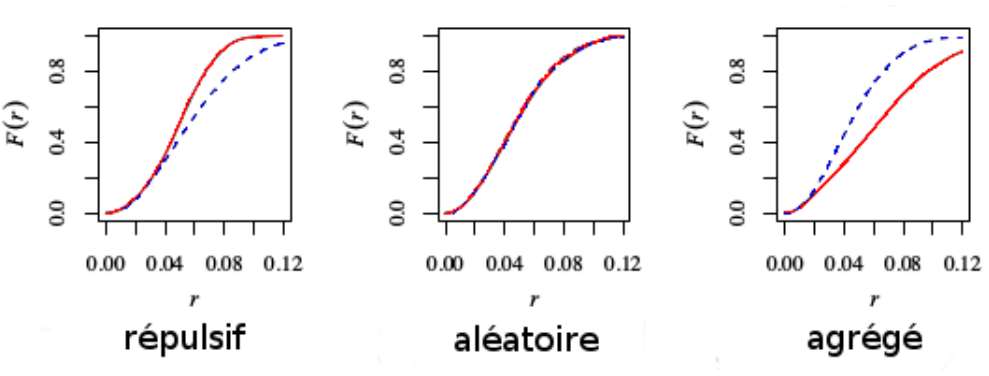


FIGURE 3.6 – Différentes estimations de F.

Sur la gauche, la courbe de la fonction calculée est au dessus de celle pour une répartition totalement aléatoire ($\hat{F}(r) > F_{pois}(r)$). Ainsi, pour une distance r, la probabilité $\mathbb{P}\{d(u, \mathbf{x}) \leq r\}$ est plus grande quelle ne l'est pour une répartition aléatoire, les espaces vides sont donc plus petits que prévu. Donc la répartition semble plutôt répulsive.

Sur la droite, la courbe de la fonction calculée est en dessous de celle pour une répartition totalement aléatoire ($\hat{F}(r) < F_{pois}(r)$). Ainsi, pour une distance r, la probabilité que $\mathbb{P}\{d(u, \mathbf{x}) \leq r\}$ est plus petite quelle ne l'est pour une répartition aléatoire, les espaces vides sont donc plus grands que prévu. Donc la répartition semble plutôt agrégée.

3.6 La fonction G du plus proche voisin

3.6.1 Définitions pour un processus de point stationnaire

Soit X un processus ponctuel* stationnaire* sur $\mathbb{R}^2$ d'intensité λ et soit **x** une réalisation* de X observée dans $W \subseteq \mathbb{R}^2$. La distance du plus proche voisin d'un point x_i de **x** est écrite telle que :

$$d_i = d(x_i, \mathbf{x} \backslash x_i)$$

C'est la plus courte distance de x_i à un autre point de **x** excepté x_i.

La fonction G de distance du plus proche voisin est alors définie pour toute distance $r \geq 0$ telle que :

$$G(r) = \mathbb{P}\{d(u, \mathbf{x}\backslash u) \leq r | \text{ u est un point de } \mathbf{x}\}$$

3.6.2 Valeurs pour un aléatoire complet

Pour un processus de Poisson* stationnaire* homogène* d'intensité λ, il est possible de prouver que la fonction G est égale à la fonction d'espace vide F :

$$F_{pois}(r) = G_{pois}(r) = 1 - exp(-\lambda \pi r^2)$$

Cela n'est vrai que pour une répartition totalement aléatoire, en général F et G seront des fonctions différentes.

3.6.3 Estimation de G

Pour un point x_i de $\mathbf{x}$, on mesure la distance $d_i = d(x_i, \mathbf{x}\backslash x_i)$ la séparant du point voisin le plus proche (distance du plus proche voisin). On répète alors cette opération pour tous les points x_1,...,x_n de la configuration, avec n le nombre des points. On peut alors calculer la fonction empirique pour toute distance $r \geq 0$ telle que :

$$\hat{G}(r) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{d_i \leq r\}$$

3.6.4 Interprétation de G

La figure 3.7 ci-dessous montre la fonction estimée de G pour les 3 configurations de la section 3.3 (figure 3.1) selon que la répartition est agrégée, aléatoire ou bien répulsive. Les traits discontinus sont la représentation graphique de la fonction G théorique pour un aléatoire total.

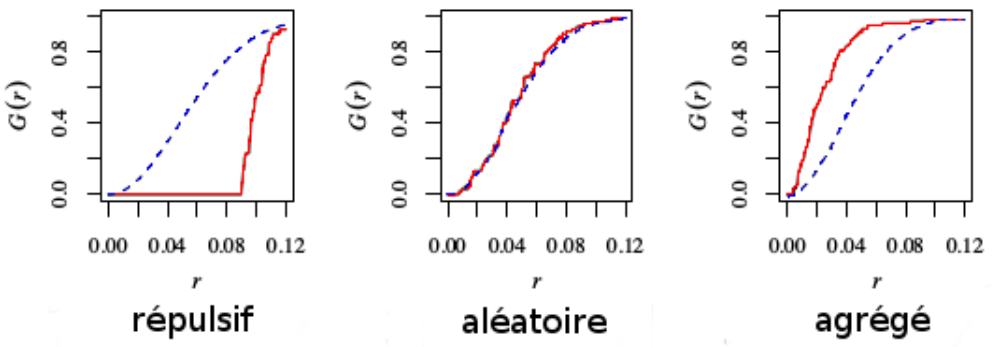


FIGURE 3.7 – Différentes estimations de G.

Sur la gauche, la courbe de la fonction calculée est en dessous de celle pour une répartition totalement aléatoire ($\hat{G}(r) < G_{pois}(r)$). Ainsi, pour une distance r, la probabilité $\mathbb{P}\{d(u, \mathbf{x}\backslash u) \leq$

$r|$ u est un point de $\mathbf{x}\}$ est plus petite quelle ne l'est pour une répartition aléatoire, les points ont des voisins plus éloignés que prévus. Donc la répartition semble plutôt répulsive.

Sur la droite, la courbe de la fonction calculée est au dessus de celle pour une répartition totalement aléatoire ($\hat{G}(r) > G_{pois}(r)$). Ainsi, pour une distance r, la probabilité que $\mathbb{P}\{d(u, \mathbf{x}\backslash u) \leq r|$ u est un point de $\mathbf{x}\}$ est plus grande quelle ne l'est pour une répartition aléatoire, les points ont des voisins plus proches que prévus. Donc la répartition semble plutôt agrégée.

3.7 Fonction J

Les distances du plus proche voisin et les distances d'espace-vide ont la même distribution de probabilité si la répartition des points est totalement aléatoire (pour un processus de Poisson*). Les fonctions F et G ont tendance à s'écarter de la fonction de Poisson de façon opposée : l'une devient plus grande et l'autre plus petite. Cela suggère qu'une combinaison de ces 2 types de distances pourrait être utile pour évaluer une répartition.
Une combinaison pratique de G et F, suggérée par la théorie fondamentale, est la fonction J d'un processus ponctuel* stationnaire* :

$$J(r) = \frac{1-G(r)}{1-F(r)}$$

définie pour tout $r \geq 0$ telle que F(r) < 1. Pour un processus de Poisson* homogène*, $F_{pois} \equiv G_{pois}$ ainsi les valeurs de J(r) > 1 sont cohérentes avec une représentation répulsive, tandis que les valeurs de J(r) < 1 sont cohérentes avec une représentation agrégée.

3.8 Effet des bords

3.8.1 Principe

Les calculs de F, G et K donnés précédemment ne sont pas tout à fait exacts. En effet, ne prenant pas en compte les points qui seraient en dehors d'une réalisation*, les calculs sur les bords de la surface seront erronés.
Une stratégie simple pour corriger ces problèmes d'estimation est la méthode de restriction des bords. Par exemple, lorsqu'on fait une estimation de K(r) pour une distance r, on limite les calculs aux points se trouvant à une distance r du bord de la surface étudiée. Ainsi le cercle de rayon r autour du point loge entièrement dans la surface comme le montre la figure 3.8.

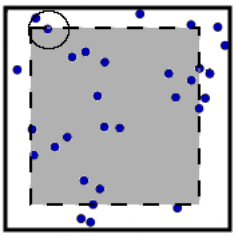


FIGURE 3.8 – Exemple de zone de sûreté pour un certain rayon.

3.8.2 Correction pour K

En prenant en compte que les points sont dans la zone de sûreté pour chaque r, on peut alors réécrire $\hat{K}(r)$. En gardant les mêmes notations qu'au 3.3.2 (La fonction K empirique), on peut écrire la fonction K avec correction des bords $\hat{K}_{bord}(r)$ telle que :

$$\hat{K}_{bord}(r) = \frac{\sum_{i=1}^{n} \mathbb{1}\{b_i \geq r\} \sum_{\substack{j=1 \\ j \neq i}}^{n} \mathbb{1}\{d_{ij} \leq r\}}{\lambda \sum_{i=1}^{n} \mathbb{1}\{b_i \geq r\}}$$

où b_i est la distance d'un point x_i au bord de la surface.

3.8.3 Correction pour F

La même méthode de restriction des bords peut être appliquée à $\hat{F}(r)$. Pour cette fonction, on ne se limite non pas aux points, mais aux positions *u* qui sont à une distance du bord de la surface supérieure à r. Ainsi en gardant les mêmes notations qu'au 3.4.3 (Estimation discrète de F), on peut écrire la fonction F avec correction des bords :

$$\hat{F}_{bord}(r) = \frac{\sum_{j=1}^{m} \mathbb{1}\{d(u_j, \mathbf{x}) \leq r\} \mathbb{1}\{b_j > r\}}{\sum_{j=1}^{m} \mathbb{1}\{b_j > r\}}$$

où b_j est la distance d'une position u_j au bord de la surface.

3.8.4 Correction pour G

Pour appliquer la restriction des bords à $\hat{G}(r)$, on se limite comme pour K aux points qui sont à une distance du bord de la surface supérieure à r. Ainsi en gardant les mêmes notations qu'au 3.5.3 (Estimation discrète de G), on peut écrire la fonction G avec correction des bords :

$$\hat{G}_{bord}(r) = \frac{\sum_{i=1}^{n} \mathbb{1}\{d_i \leq r\} \mathbb{1}\{b_i \geq r\}}{\sum_{i=1}^{n} \mathbb{1}\{b_i \geq r\}}$$

où b_i est la distance d'un point au bord de la surface.

3.9 Test d'enveloppe

Les estimations des fonctions F, G, J et K sont faites à partir de réalisations*. Cela induit un écart entre la valeur théorique et la valeur estimée. La question que l'on se pose est : est-ce que la configuration observée pourrait être considérée comme la réalisation* d'un processus de Poisson* ? Pour cela on fait appel au test d'enveloppe : un outil fournissant une meilleure estimation pour répondre à la question. Son principe est le suivant : on lance les calculs des fonctions F, G, J et K sur un nombre n de configurations de points que l'on sait être aléatoires. On regarde ensuite les valeurs des n courbes obtenues. On peut tracer les courbes représentant les minimums et maximums atteints par les fonctions sur chaque abscisse. On obtient alors une enveloppe de toutes les valeurs obtenues pour les n courbes.

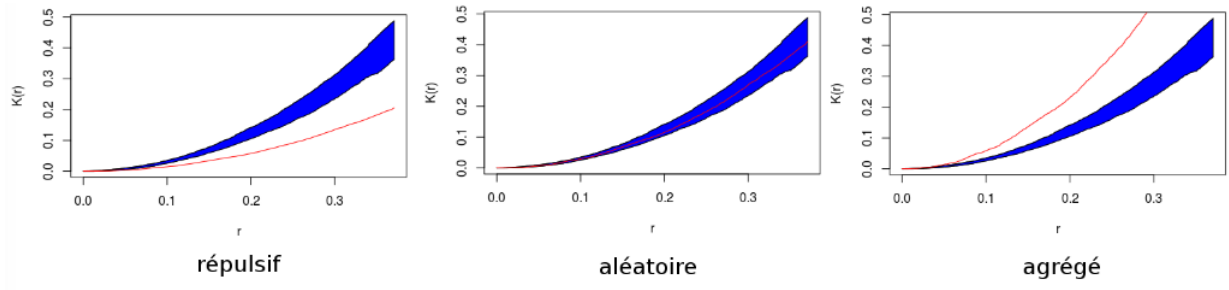


FIGURE 3.9 – Exemple d'un test d'enveloppe sur la fonction K

Ainsi s'il faut analyser une configuration, on peut maintenant comparer sa courbe avec l'enveloppe et faire une analyse plus précise de la répartition. Comme le montre le figure 3.9, pour la fonction K :

- si la courbe est en dessous de l'enveloppe, la répartition sera plutôt répulsive ;
- si la courbe est dans l'enveloppe, la répartition sera plutôt aléatoire ;
- si la courbe est au dessus de l'enveloppe, la répartition sera plutôt agrégée.

Remarque : Il n'est pas obligatoire de prendre le minimum et le maximum de chaque fonction pour faire l'enveloppe, on peut très bien faire par exemple un encadrement entre 2.5% et 97.5% pour éliminer les valeurs extrêmes.

4 Réalisation

4.1 Environnement logiciel

Après l'étude mathématique du problème, il fut possible de se consacrer à l'implémentation. Le code du calcul a été écrit en C++ afin d'utiliser un langage compilé connu et utilisé par mon maître de stage et ainsi faciliter l'insertion de mon code dans ses programmes réalisés ultérieurement. La partie graphique (affichage des courbes résultats) a été réalisée avec le logiciel R qui est très utilisé des statisticiens et qui possède une librairie spatstat* dédiée aux processus ponctuels*. Ainsi j'ai pu comparer mes résultats avec ceux de cette librairie.

4.2 Méthode de travail

Tout au long du stage nous avons procédé étape par étape. En premier lieu, M. Radu Stoica me donnait des explications, puis me donnait de la documentation à lire sur la partie mathématique dans le livre "Spatial Point Patterns - Methodology and Applications with R" [8].

Ensuite après avoir lu et réfléchi sur une méthode de conception, je la partageais avec lui et nous en discutions. Il me proposait lui aussi des idées et nous regardions si elles pouvaient être inclues dans ma conception. De plus nous réfléchissions également pour savoir si nos méthodes étaient viables et rentables en terme de mémoire ou de temps d'exécution. Ensuite je réalisais le code.

Une fois terminé, ou bien lorsque le travail était suffisamment avancé, je retournais voir M. Stoica pour faire le point. Nous regardions alors ce qui n'allait pas et ce qui pouvait être amélioré. Enfin je corrigeais les défauts de mon code et nous passions à une autre fonction en commençant par la partie théorique.

Ainsi j'ai eu à implémenter les fonctions F, G, J et K expliquées dans la partie 3 du rapport. Ensuite j'ai dû prendre en main du code déjà écrit par M. Stoica afin d'y insérer mes fonctions. Enfin j'ai codé le calcul d'intensité.

4.3 Structure du code

Pour réaliser les fonctions j'avais à ma disposition des fichiers texte contenant les valeurs des positions (x,y) d'une répartition de point. J'ai créé un fichier contenant des informations sur des paramètres expérimentaux que l'on pouvait donc modifier d'une expérience à l'autre. Ainsi j'avais

2 fichiers fournisseurs d'informations en entrée. En analysant les calculs des fonctions, je me suis rendu compte que j'avais besoin de générer plusieurs informations :

— la surface de la répartition ;

— un quadrillage sur toute la surface pour F ;

— les distances entre chaque point pour K et G ;

— les distances de chaque maille du quadrillage avec chaque point ;

— une liste d'abscisses de rayons.

Ainsi pour gérer le stockage de ces informations j'ai créé plusieurs classes*. Une fois toutes les classes* instanciées (cf instance*) il a fallu les stocker, puis traiter leur valeurs afin d'avoir le matériel nécessaire pour réaliser les calculs. Dans un souci de présentation, je commencerai par présenter les objets* nécessaires aux calculs, puis les algorithmes des calculs, et enfin je parlerai de points plus précis du code.

4.3.1 Création des abscisses

Pour les calculs des valeurs des fonctions, nous allons faire augmenter un rayon autour de chaque point ou maille. Ce seront les abscisses de nos fonctions. On appelle le dernier rayon, le rayon maximum : "rMAx".

Pour se rapprocher au maximum des courbes fournies par la librairie spatstat*, je suis allé chercher des informations sur la façon dont le code crée la liste des rayons (ses abscisses). J'ai alors trouvé sur le site de librairie spatstat*[1] qu'il fallait créer une liste de taille 513 (512 intervalles), les valeurs allant de 0 à Rmax.

Le calcul de Rmax est différent pour K et les autres fonctions. Ainsi j'ai trouvé qu'il fallait prendre le quart du plus petit des cotés du rectangle de surface comme rayon maximal pour la fonction K. Cependant, je n'ai rien trouvé pour les autres fonctions. J'ai donc fait une analyse de la situation : on a vu dans la partie 3 que pour une configuration aléatoire, les fonctions F et G sont définies par l'équation :

$$F_{poiss} = 1 - exp(-\lambda.pi.r^2)$$

Cette fonction tend vers 1, donc après une certaine abscisse, les valeurs de la fonction seront toutes très proches de 1 et n'ajouteront plus d'information. On cherche donc un rMax qui permettrait de ne garder que les valeurs inférieures à un gros pourcentage de la valeur finale de la fonction (> à 90%) Alors en notant "P" le pourcentage de la valeur finale on peut écrire :

$$P = 1 - exp(-\lambda.\pi.rMax^2)$$

Une transformation rapide de l'équation permet d'exprimer clairement rMax en fonction de P :

$$rMax^2 = \frac{-ln(1-P)}{\pi.\lambda}$$

Or en choisissant $-ln(1-P) = \pi$:

$$\begin{aligned} P &= 1 - exp(-pi) \\ &= 0.9567 \end{aligned}$$

Ainsi avec cette valeur de P on peut avoir un pourcentage de 96% de la valeur finale tout en ayant un calcul simple de rMax :

$$rMax^2 = \frac{1}{\lambda}$$

On a vu également Partie 3, que l'on peut calculer l'intensité λ par le rapport du nombre de points n sur la taille de la surface W.

Ainsi prendre $rMax^2 = \frac{W}{n}$ permet de ne pas avoir de valeur infinie mais de garder la majorité des informations importantes.

4.3.2 Les classes

Comme le montre le diagramme de classe* en Annexe A.1 j'ai créé 4 classes*.

La classe Rectangle

Cette classe* se veut être la représentation de la surface. Elle est définie par 4 valeurs d'encadrement de type double* :

- xMin défini par la plus petite valeur de x de la liste de points ;
- xMax défini par la plus grande valeur de x de la liste de points ;
- yMin défini par la plus petite valeur de y de la liste de points ;
- yMax défini par la plus grande valeur de y de la liste de points.

Ainsi cette classe* permet de retourner l'aire totale de la surface nécessaire pour le calcul de K avec un simple calcul :

```
double Rectangle::getAire() const{
  return((xMax - xMin)*(yMax - yMin));
}
```

Listing 4.1 – Méthode donnant l'aire de la surface

De plus cette classe* permet de renvoyer la distance d'une position à la surface afin de gérer les effets de bord, pour des coordonnées (x,y) :

```
double Rectangle::distanceSurface(double x, double y) const{
  double d = xMax - x;
  if(d >= (x - xMin)){
    d = x - xMin;
  }
  if(d >= (yMax - y)){
    d = yMax - y;
  }
  if(d >= (y - yMin)){
    d = y - yMin;
  }
  return d;
}
```

Listing 4.2 – Méthode donnant la distance d'un point à la surface

La classe Position

Cette classe* ne sera pas instanciée (cf instance*), elle servira de classe* mère pour les classes* Point et Quadrillage. Elle est définie par 2 coordonnées "x" et "y" de type double* et une valeur de distance au bord de la surface "distanceBord" de type double* également.

Les méthodes* de cette classe* permettent d'accéder aux variables x,y et distanceBord.

La classe Point

Cette classe* hérite (cf héritage*) de la classe* Position, elle se veut être la représentation d'un point de la configuration d'entrée. En plus des caractéristiques de sa classe* mère, elle possède la liste "listeDistancePoints" des distances la séparant de chacun des autres points de la configuration.

Cette classe* permet alors de trier sa liste de distance de la plus petite à la plus grande afin de faciliter les calculs à venir, la méthode* utilise le tri du namespace* std qui sera expliquée plus en détail page 19.

```
void Point::trierListe(){
  std::sort(listeDistancePoints.begin(), listeDistancePoints.end());
}
```

Listing 4.3 – Méthode triant la liste de distance

La classe* permet également de recupérer un élément précis de sa liste de distance. Comme la liste aura été triée au préalable, plus on augmente l'indice en entrée de la méthode* plus la valeur retournée sera grande :

```
double Point::getDistanceDuPoint(int i){
  return listeDistancePoints[i];
}
```

Listing 4.4 – Méthode donnant le ième élément de la liste de distances

La classe Quadrillage

Cette classe* hérite (cf héritage*) de la classe* Position, elle se veut être la représentation d'une maille d'un quadrillage de la surface de la configuration de points. Elle sera utile pour le calcul de la fonction F. En plus des caractéristiques de sa classe* mère, elle possède une valeur "distanceVide" de type double* qui représente la distance la séparant du point de la configuration dont elle est le plus proche.

Cette classe* contient ainsi une méthode* permettant de trouver le point le plus proche et pour pouvoir initialiser sa variable :

```
void Quadrillage::setDistanceVide(vector<Point> &point){

  int i=0;
  int n = (int)(point.size());
  double dX = 0;
  double dY = 0;
  double distance = 0;
```

```
8     double dmin = 999999;
9
10    for(i=0;i<n;i++){
11      // calcul distances au écarr
12      dX = x - point[i].getX();
13      dY = y - point[i].getY();
14
15      distance = dX*dX + dY*dY;
16
17      if(dmin > distance){
18        dmin  = distance ;
19      }
20    }
21
22    distanceVide = dmin;
23  }
```

Listing 4.5 – Méthode trouvant le point le plus proche d'une maille du Quadrillage

Cette classe* possède également une méthode* permettant d'accéder à sa valeur "distanceVide".

4.3.3 Algorithmes de calcul des fonctions

Les algorithmes de calcul des fonctions K et F,G,J sont représentés en Annexe B.1 et B.2. On suppose qu'on a préalablement rangé et trié les valeurs des distances à comparer.

Pour K, on procède point par point : on calcule combien chaque point a de voisins pour chaque rayon. Si un point est trop près du bord par rapport à la taille d'un rayon, on n'ajoute pas son nombre de voisins. La liste des points étant préalablement triée par rapport à leur distance au bord, on peut arrêter la boucle sur les points dès qu'un point n'est plus dans la zone de sûreté.
De même en ayant trié les distances entre les points, on peut arrêter la boucle sur les distances aux autres points dès qu'une distance dépasse la valeur de rayon. On peut alors garder la valeur du nombre de voisins, et reprendre le calcul pour rayon suivant plus grand. En effet, comme les listes ont étés préalablement triées, une fois la valeur de rayon dépassée toutes les valeurs suivantes le seront aussi.

Pour F,G et J, on procède point par rayon : pour chaque rayon, on calcule la valeur de F et de G puis on en déduit la valeur de J. Ainsi on compare la valeur de chaque rayon avec les valeurs de distance du plus proche voisin de chaque point pour G et de chaque quadrillage pour F. De plus comme pour K, en triant les listes de Point et de Quadrillage en fonction de leur distance au bord, on peut optimiser les calculs de correction d'effet de bord.

4.3.4 Les listes

Présent à plusieurs endroit du code, l'objet* Vector du namespace* std m'a été très utile. En effet, c'est ce type de liste que j'ai utilisé pour :

- la liste de Point ;
- la liste de Quadrillage ;
- chaque liste de distance dans les objets* Points ;

Comme on peut le voir sur le site cppreference.com[2] l'objet* Vector est un conteneur séquentiel qui encapsule les tableaux de taille dynamique.

Comme les tableaux, l'objet* Vector utilise un stockage contigu, ce qui signifie que les éléments sont accessibles non seulement via les itérateurs, mais aussi à partir des pointeurs classiques sur un élément. A la différence des tableaux, la taille du vecteur peut être modifiée dynamiquement, c'est à dire que son stockage est pris en charge automatiquement, pouvant être augmenté ou diminué au besoin.

Les complexités des opérations courante pout l'objet* Vector sont les suivantes :

— Accès aléatoire - constante O(1) ;

— Insertion ou le retrait d'éléments à la fin - constante amortie O(1) ;

— Insertion ou le retrait d'éléments - linéaire O(n) ;

Au vu du nombre important d'accès aux valeurs des listes pour les calculs, le choix de cette structure m'a paru adapté.

4.3.5 Les tris

Trier les listes définies ci-dessus, permet par la suite d'éviter de nombreux calculs liés aux distances entre les points.

Il a ainsi fallu classer* les listes de Point et de Quadrillage de l'objet* le plus éloigné du bords au plus proche pour les deux listes. De plus pour chaque objet* Point il a fallu trier la liste de distance aux autres points de la plus petite à la plus grande.

La méthode de tri utilisée fut la méthode* "sort" du namespace* std. Selon le site cppreference.com[3] c'est un tri optimisé qui utilise l'algorithme de tri "introsort" hybride entre le tri rapide et tri par tas. Il fournit ainsi une vitesse de calcul rapide pour le cas moyen, et des performances optimales dans le pire des cas. Ce tri possède ainsi une complexité de $n.log(n)$ dans tous les cas.

L'étude[9] menée par une élève de l'ISIMA, à Clermont-Ferrand montre les avantages et efficacité de cette méthode de tri sur l'objet* Vector du namespace* std.

4.4 Stratégies d'implémentations adaptées au problème

4.4.1 Le remplissage des listes des distances

Comme chaque point possède la liste des distances aux autres points, on peut se représenter le tableau suivant :

	point 1	point 2	...	point j	...	point n
point 1	0	d_{12}	...	d_{1j}	...	d_{1n}
point 2	d_{21}	0	...	d_{2j}	...	d_{2n}
...	...	...	...	...	...	...
point i	0	d_{i2}	...	d_{ij}	...	d_{in}
...	...	...	...	...	...	...
point n	d_{n1}	d_{n2}	...	d_{nj}	...	0

Or la distance d'un point i à un point j est la même que celle d'un point j à un point i, c'est à dire que $d_{ij} = d_{ji}$. On n'a besoin de ne calculer que la partie triangulaire supérieure du tableau pour avoir toutes les valeurs.

```
int i;
int j;
double dX = 0;
double dY = 0;
double distance = 0;

for( i = 1; i < n-1; i++){
    for( j = i; j < n; j++){
        dX = point[j].getX() - point[i].getX();
    dY = point[j].getY() - point[i].getY();

    distance  = dX*dX + dY*dY ;

    point[i].ajoutDistance( distance );
        point[j].ajoutDistance( distance );
    }
}
```

Listing 4.6 – Remplissage des listes de distance

Ainsi en appliquant la méthode ci dessus, on peut faire moitié moins d'opérations : pour n points, on économise n(n-1)/2 opérations.

4.4.2 Les distances aux carrés

La majorité des calculs concerne des calculs de distance. Ainsi entre 2 points A et B d'abscisses et d'ordonnées respectives (xA,yA) et (xB,yB) la distance se calcule par la formule de Pythagore :

$$distance = \sqrt{(xB - xA)^2 + (yB - yA)^2}$$

Or le calcul de la racine carrée prend du temps et peut être évité.

En effet, ces distances vont être par la suite seulement comparées, ainsi pour un rayon "r" et une distance entre 2 points "d" :

$$d < r \Leftrightarrow d^2 < r^2$$

Ainsi, il est utile de ne stocker que les valeurs au carré des calculs de distance. En notant ts le temps de calcul d'une racine carrée, on gagne les temps t :

t = (n*(n-1)/2)*ts pour les calculs de distance point/point

t = (m*n)*ts pour les calculs de distance maille/point

4.4.3 Arrêter les calculs et stockage au rayon maximal

Pour le calcul de K, pour chaque point on incrémente un compteur selon le nombre de voisins qu'a le point étudié dans un certain rayon. Tous les points à une distance plus grande que le plus

grand rayon rMax ne seront donc jamais pris en compte. Ainsi lors du remplissage des listes de distance, il est inutile de stocker les distances entre 2 points éloignés de plus de rMax. Cela permet d'économiser beaucoup de mémoire, mais aussi de réduire drastiquement le temps de calcul : moins de temps d'ajout dans les listes, des tris plus rapides et moins de calculs à la fin.

4.5 Tests d'enveloppe

Monsieur Stoica avait par le passé réalisé des programmes en C++ permettant de générer des configurations de points que l'on sait être plutôt répulsifs ou plutôt agrégés. Ses programmes au nombre de deux suivent les méthodes de "Strauss" et de "Area Interaction". Ainsi "Strauss" génère à chaque lancement des configurations de points répulsives, tandis que "Area Interaction" génère des configurations répulsives ou agrégées selon les paramètres fournis.

En itérant un certain nombre de fois ces programmes, on peut alors faire des enveloppes en appliquant à chaque itération les programmes des fonctions F, G, J et K. On change alors la référence Poissonnienne (qui modélise un phénomène aléatoire), on utilise une référence du phénomène de répulsion et une référence du phénomène d'agrégation.

J'ai alors dû explorer le code de M. Stoica, afin d'y insérer et d'y adapter mes programmes. Ainsi en bouclant l'exécution du code obtenu, on peut stocker les valeurs nécessaires pour afficher les enveloppes.

4.6 Intensité

4.6.1 Calcul

Pour le calcul de l'intensité j'ai gardé la classe* Point pour modéliser les points de la réalisation et la classe* Rectangle pour modéliser la surface. Ainsi après une lecture des paramètres, tous les objets* nécessaires à l'implémentation sont disponibles. L'algorithme se trouve en Annexe B.3.

Le principe est donc de se placer sur toutes les positions de matrice quadrillée et de calculer les valeurs des intensités en chacune de ces positions. Les valeurs des intensités sont mesurées en fonction du nombre de points à proximité des positions. Ainsi en utilisant les fonctions noyaux décrites au 3.2 on compare la position avec celle de tous les points de la réalisation. L'implémentation de ces fonctions est représentée ci-dessous :

```
double indicatriceCarre(double u1, double u2, double c){
  return ( (u1 < c) && (-u1 < c) && (u2 < c) && (-u2 < c) ) ? 1 : 0;
}

double indicatriceRond(double u1, double u2, double r){
  return ( (u1*u1 + u2*u2)/4 <= r*r ) ? 1 : 0;
}

double gaussienne(double u1, double u2, double sigma){
  return exp(-(u1*u1 + u2*u2)/(2*sigma*sigma));
}
```

Listing 4.7 – Fonctions des noyaux carré, rond et Gaussien

4.6.2 Correction des bords

Ensuite comme expliqué au 3.2, les estimations sont faussées par des effets de bords. La correction à appliquer est une division que l'on voit apparaître dans l'algorithme en Annexe B.3, et qui est noté "Erreur()". Cette erreur ne dépend que de la position d'un point. Ainsi en début de programme il a fallu calculer ces erreurs.

La formule mathématique pour ce calcul est utilise la fonction noyau. Comme le noyau Gaussien est le plus intéressant des trois, la correction n'a été appliquée que sur ce noyau. Ensuite il y a un calcul d'intégrale (continue), il faut donc utiliser une méthode de discrétisation pour calculer numériquement cette intégrale. Selon le tableau comparatif de cette page Wikipédia[4], la méthode de Rohmberg converge le plus rapidement vers une solution précise.

Cette méthode génère un tableau triangulaire d'estimations numériques de l'intégrale en appliquant l'extrapolation de Richardson* à la méthode des trapèzes*. Cela permet d'améliorer l'ordre de convergence de la méthode des trapèzes* en divisant l'intervalle d'étude et en formant des combinaisons judicieuses des différents termes calculés [6].

Le principe est le suivant : si l'on veut trouver une estimation de $f(x)$ intégrée de a à b on calcule le premier terme $R(0,0)$ par la méthode des trapèzes, puis les termes suivants en sommant des nouveaux termes et en faisant des combinaisons avec les termes calculés précédemment :

$$
\begin{aligned}
R(0,0) &= \frac{1}{2}(b-a)(f(a)+f(b)) \\
R(n,0) &= \frac{1}{2}R(n-1,0) + h\sum_{k=1}^{2^{n-1}} f(a+(2k-1)h) \\
R(n,m) &= \frac{1}{4^m-1}(4^m R(n,m-1) - R(n-1,m-1))
\end{aligned}
$$

avec $h = (b-a)/2^n, n \geq 1, m \geq 1$.

On utilise alors les termes diagonaux $R(n,n)$ comme estimation de l'intégrale avec une erreur en $O(h^{2n+1})$.

L'intégrale étant sur une surface (intégrale double), il a fallu faire deux fonctions pour la méthode de Rohmberg. Une première fonction fait varier une première variable et appelle une deuxième fonction qui, elle, fait varier une deuxième variable et appelle la fonction du noyau.

5 Bilan

5.1 Résultats obtenus

5.1.1 Validation

Une fois les fonctions réalisées, il a fallu s'assurer de la qualité des résultats. Pour cela je me suis aidé de la librairie spatstat* pour comparer les courbes obtenues avec mes calculs avec celles réalisées par leurs algorithmes.

Comme on peut le voir en Annexe C.1 le résultat final correspond bien aux attentes. Il est difficile de distinguer à l'oeil les écarts entre les courbes, sauf pour la fonction J pour laquelle la différence est engendrée par les petits écarts sur les fonctions F et G. On a alors étudié les écarts entre les valeurs calculées par mon programme et celles calculées par librairie spatstat*. Le tableau 5.1 ci-dessous recense, pour ces écarts de résultats, les moyennes, maximums et écarts types pour chaque fonction.

fonction	moyenne	valeur maximale	écart type
f_{ecart}	0.001407488	0.005637747	0.001034606
g_{ecart}	1.500446e-05	0.0003938599	5.073528e-05
j_{ecart}	0.0004730104	0.004047371	0.0007574491
k_{ecart}	0.002176537	0.006467136	0.001847468

TABLE 5.1 – Statistiques sur les écarts de résultats

Ainsi les écarts moyens de résultats ne dépassent pas un ordre de grandeur de 10^{-3}, écarts considérés satisfaisant par M. Stoica. De plus les valeurs maximales de ces écarts indiquent que dans le pire des cas les écarts sont raisonnables. Les écarts types montrent que les valeurs ne s'éloignent pas beaucoup de la moyenne. On peut donc considérer les calculs de fonction comme valides.

5.1.2 Visualisation

Les codes en C++ écrivent des valeurs dans un fichier texte, ces valeurs sont ensuite lues par un programme R qui traite alors les résultats. Ensuite, une fois les résultats rapidement analysés (recherche des minimums et maximums pour le test d'enveloppe), ce même programme affiche les courbes résultantes des valeurs calculées par le premier programme.

En Annexe C.2 sont représentés les résultats des calculs des fonctions F,G,J et K avec leur courbes des valeurs pour un aléatoire théorique pour une observation. Ainsi on peut constater que cette distribution est un peu plus répulsive que ne le serait une distribution complètement aléatoire.

En Annexe C.3 est représenté le résultat des tests d'enveloppe. La dernière courbe analysée y est représentée également en tant qu'exemple.

En Annexe C.4 est représenté le résultat d'un calcul d'intensité avec un noyau carré, rond et Gaussien. La correction des bords n'est montrée que pour le noyau Gaussien car c'est le plus intéressant.

5.1.3 Utilisation

Vous pouvez trouver le projet en libre accès sur github, à l'adresse :
`https://github.com/DugueC/Stage2A`

Les codes fonctionnent à condition d'avoir installé un compilateur c++ et R.
Si un script ne fonctionne pas, essayez de donner les droits au fichier
(chmod 777 nom_script.sh).

Les procédures d'utilisations des programmes sont décrites dans le "READ ME".

5.2 Difficultés et Solutions

Les premières difficultés sont intervenues dès le début du stage. N'étant pas à l'aise avec le langage C++, et ne connaissant pas le langage R, j'ai dû m'adapter et me documenter. J'ai également demandé de l'aide au début pour l'affichage des fonctions réalisé en R. La barrière du langage à donc freiné dans un premier temps l'avancée du projet.

Une fois les fonctions réalisées en procédant simplement par un suivi intuitif des équations des fonctions théoriques, il a fallu accélérer les temps de calculs. En effet mon programme n'était pas très performant dès que les configurations de points étaient trop fournies, car il n'était pas optimisé dans une optique de gain de temps. J'ai donc revu mes algorithmes et trouvé des astuces afin que le temps de calcul des fonctions soit du même ordre que ceux réalisés par la librairie spatstat*. Cela n'a pas toujours été facile notamment pour les changements d'algorithmes.

5.3 Devenir du projet

L'ensemble du projet a été réalisé en 2 dimensions, c'est à dire que les points fournis n'avaient que deux coordonnées spatiales. Une adaptation du code pour une 3ème dimension ouvrirait la porte à un nouveau panel d'expériences qu'il serait possible d'analyser.

M. Radu Stoica s'intéresse à la répartition des positions des galaxies dans notre univers. Au fur et à mesure que les nouveaux outils mathématiques sont proposés, cela demande leur reformatisation. Une perspective immédiate est d'étendre ces développements logiciels au cas des processus ponctuels* inhomogènes*.

6 Conclusion

L'objectif de ce stage à été atteint : les outils informatiques pour traiter les données ponctuelles ont été réalisés (fonctions F,G,J et K) et intégrés au code de mon maître de stage afin d'analyser ces données (test d'eveloppe). Il a été possible également d'implémenter des calculs d'intensité durant ce stage de 8 semaines.

Pour le futur, on peut envisager une extension des fonctions implémentées pour des données en trois dimensions. De même on pourrait continuer ce projet en élargissant son développement pour des processus inhomogènes*.

Finalement cette expérience m'a fait découvrir un monde de la recherche en mathématique et un nouveau mode de travail. Ce fut donc une expérience agréable et utile pour moi puisqu'il m'a permis d'apprendre le langage R et de m'améliorer dans le C++ et d'envisager un nouvel axe de projet professionnel.

Bibliographie / Webographie

[1] Scott Chacon. Spatstat analysing spatial point patterns - faq. http://spatstat.org/FAQ.html. 15

[2] cppreference.com. Description de l'objet std : :vector. http://fr.cppreference.com/w/cpp/container/vector. 19

[3] en.wikipedia.org. Description du tri std : :sort. https://en.wikipedia.org/wiki/Sort_(C%2B%2B)#Complexity_and_implementations. 19

[4] fr.wikipedia.org. Comparatif des méthodes de calcul numérique d'intégral. https://fr.wikipedia.org/wiki/Calcul_num%C3%A9rique_d%27une_int%C3%A9grale#Tableau_comparatif. 22

[5] iecl.univ lorraine.fr. Présentation de l'institut elie cartan. http://www.iecl.univ-lorraine.fr/index.php/presentation/. 2

[6] Hubert Klein. Méthode de romberg. http://www.cinam.univ-mrs.fr/klein/teach/mip/numeriq/node36.html. 22

[7] A. Walker R. Fisher, S. Perkins and E. Wolfart. Image d'une fonction gaussienne en 2 dimensions. https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm. 5

[8] Adrian Baddeley Rolf Turner, Ege Rubak. *Spatial Point Patterns - Methodology and Applications with R*. CRC Press, 2016. 7, 14

[9] Hélène Toussaint. Performances et efficacité du std : :sort. https://www.isima.fr/~toussain/doc/sort_c++.pdf. 19

Liste des illustrations

3.1 Représentation des différentes formes de représentations spatiales de points 6

3.2 Nombre de points voisins dans un certain rayon. 7

3.3 Représentation de la fonction K pour une répartition totalement aléatoire. 7

3.4 Différentes estimations de K. 7

3.5 Représentation de la fonction de répartition de Poisson pour $\lambda = 40$. 8

3.6 Différentes estimations de F. 9

3.7 Différentes estimations de G. 10

3.8 Exemple de zone de sûreté pour un certain rayon. 11

3.9 Exemple d'un test d'enveloppe sur la fonction K 13

A.1 Diagramme de classe pour les calculs des fonctions 39

C.1 Comparaison des fonctions programmées avec celles calculées par spatstat 47

C.2 Comparaison des fonctions programmées avec courbes théoriques pour une répartition aléatoire 49

C.3 Exemple de résultat des tests d'enveloppes 51

C.4 Exemple de résultat des calculs d'intensité 53

Liste des tableaux

5.1 Statistiques sur les écarts de résultats 23

Listings

4.1 Méthode donnant l'aire de la surface 16

4.2 Méthode donnant la distance d'un point à la surface 16

4.3 Méthode triant la liste de distance 17

4.4 Méthode donnant le ième élément de la liste de distances 17

4.5 Méthode trouvant le point le plus proche d'une maille du Quadrillage 17

4.6 Remplissage des listes de distance 20

4.7 Fonctions des noyaux carré, rond et Gaussien 21

B.1 Algorithme fonction K 41

B.2 Algorithme fonction F, G et J 43

B.3 Algorithme calcul de l'intensité 45

Glossaire

classe En programmation orientée objet, la déclaration d'une classe regroupe des membres, classes* et attributs (variables) communs à un ensemble d'objets. Il est possible de restreindre l'ensemble d'objets représenté par une classe A grâce à un mécanisme d'héritageage*. Dans ce cas, on crée une nouvelle classe B liée à la classe A et qui ajoute de nouvelles propriétés. 15–19, 21, 35

double Nombre codés sur 64 bits entre $-1.7 * 10^{-308}$ et $1.7 * 10^{308}$. 16, 17

extrapolation de Richardson En analyse numérique, le procédé d'extrapolation de Richardson est une technique d'accélération de la convergence. Il est ainsi dénommé en l'honneur de Lewis Fry Richardson, qui l'a introduit au début du XXe siècle. 22

homogène Dont les propriétés ne changent pas au cours du temps. 4, 5, 10, 11, 24, 25

héritage En programmation orientée objet, l'héritage est un mécanisme qui permet, lors de la déclaration d'une nouvelle classe*, d'y inclure les caractéristiques d'une autre classe*. 17, 35

instance En programmation orientée objet, on appelle instance d'une classe*, un objet* avec un comportement et un état, tous deux définis par la classe*. Il s'agit donc d'un objet* constituant un exemplaire de la classe*. 15, 17

librairie spatstat Librairie fournissant des méthode de calcul pour l'étude de statistiques spatiales, spatstat est plutôt orienté vers les calculs en 2 Dimensions. 4, 14, 15, 23, 24

méthode En programmation orientée objet, une méthode est une entité informatique dans une classe* qui encapsule une portion de code effectuant un traitement spécifique bien identifié (asservissement, tâche, calcul, etc.). 17–19

méthode des trapèzes Méthode pour le calcul numérique d'une intégrale, qui calcule l'aire des trapèzes successifs sous la courbe d'une fonction. 22

namespace Lieu abstrait conçu pour accueillir des ensembles de termes appartenant à un même répertoire. 17–19

objet En programmation orientée objet, un objet est créé à partir d'une classe*, de laquelle il héritagee* les comportements et les caractéristiques. 15, 18, 19, 21, 35

processus de Poisson Nommé d'après le mathématicien français Siméon Denis Poisson et la loi du même nom, est un processus de comptage classique dont l'équivalent discret est la somme d'un processus de Bernoulli. C'est le plus simple et le plus utilisé des processus modélisant une file d'attente. C'est un processus de Markov, et même le plus simple des processus de naissance et de mort. 7, 8, 10–12, 36

processus ponctuel Type particulier de processus stochastique pour lequel une réalisation est un ensemble de points isolés du temps et/ou de l'espace. Par exemple, la position des arbres dans une forêt peut être modélisée comme la réalisation d'un processus ponctuel. 4, 6, 8, 9, 11, 14, 24, 36, 55

processus ponctuel de Poisson Le processus ponctuel de Poisson est le plus simple et le plus universel des processus ponctuel*. C'est une généralisation spatiale du processus de Poisson* utilisé en théorie des files d'attentes. 6

réalisation La réalisation d'un processus ponctuel est un ensemble de points isolés du temps et/ou de l'espace. Par exemple, la position des arbres dans une forêt peut être modélisée comme la réalisation d'un processus ponctuel. 4, 6, 8, 9, 11, 12

stationnaire Dont les propriétés ne changent pas selon l'endroit où l'on regarde. 6–11

statistique descriptive Technique utilisée pour décrire un ensemble relativement important de données. 6

Annexes

A Diagramme de Classes

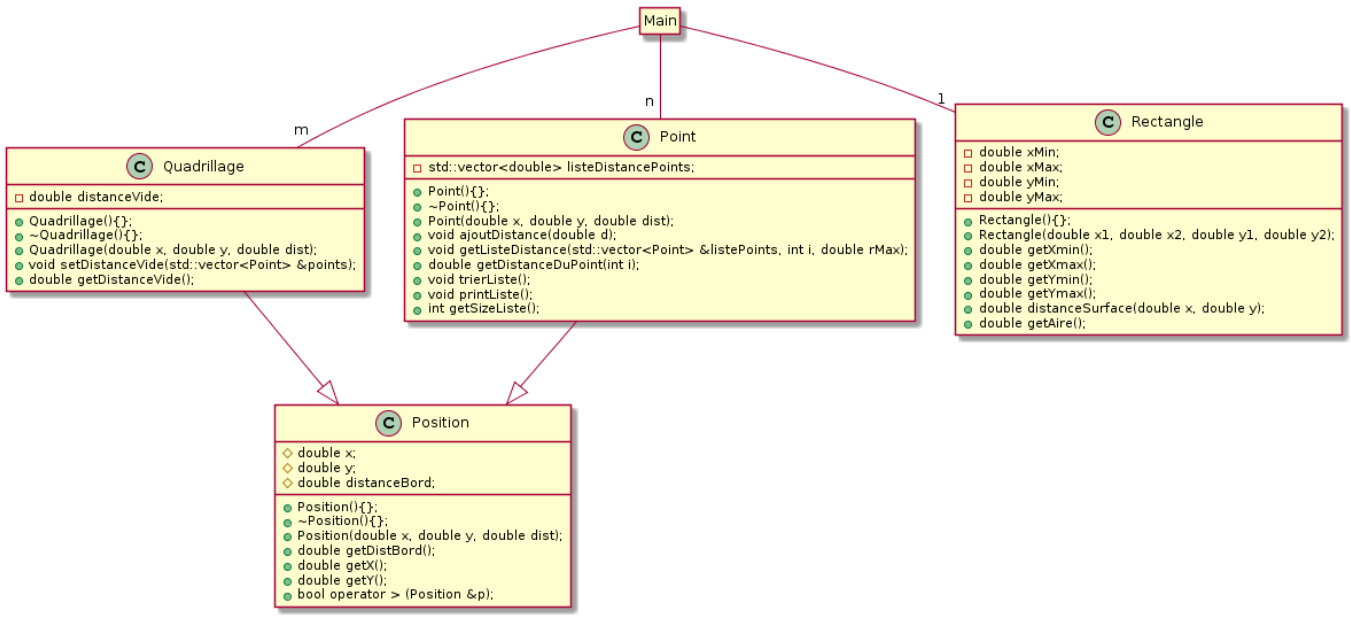


FIGURE A.1 – Diagramme de classe pour les calculs des fonctions

B Algorithmes des calculs des fonctions

```
ENTREE : Liste des Points

CONSTANTES :
    n = nombre de points dans liste
    nR = nombre de rayons à analyser
    aire = aire de la surface couverte par les points de la liste

LISTES :
    NombrePointsConsideres[]
    Kcumule[]

VARIABLES :
    i = 0
    j = 0
    k = 0

POUR ( i de 0 à n )
{
    j = 0

    TANT QUE ( distance du point i au bord  >  au j-eme rayon )
          ET ( j < nR )
    {
        INCREMENTER NombrePointsConsideres[j]

        TANT QUE ( distance du point i au point k < j-eme rayon)
              ET ( k < n - 1 )
        {
            INCREMENTER k
        }

        Kcumule[j] = Kcumule[j] + k

        INCREMENTER j
    }
}

POUR ( j de O à nR )
{
    SAUVEGARDER K[j] = Kcumule[j]*aire/((n-1)*NombrePointsConsideres[j])
}
```

Listing B.1 – Algorithme fonction K

```
ENTREE : Liste des Points

CONSTANTES :
    nP = nombre de points dans liste
    nQ = nombre de quadrillage
    nR = nombre de rayons à analyser
    aire = aire de la surface couverte par les points de la liste

VARIABLES :
    i = 0
    j = 0
    k = 0
    sommeF = 0
    sommeG = 0

POUR ( k de 0 à nR )
{
    i = 0

    TANT QUE ( distance du point i au bord  >=  au k-eme rayon )
          ET ( i < nP )
    {
        SI ( distance du point i au plus proche point voisin <= k-eme rayon)
        {
            INCREMENTER sommeG
        }
        INCREMENTER i
    }

    j = 0

    TANT QUE ( distance du quadrillage j au bord  >  au k-eme rayon )
          ET ( j < nQ )
    {
        SI ( distance du quadrillage j au plus proche point voisin <= k-eme
    rayon)
        {
            INCREMENTER sommeF
        }
        INCREMENTER j
    }

    SAUVEGARDER G = sommeG/i
    SAUVEGARDER F = sommeF/j

    SI ( F==1 )
    {
        SAUVEGARDER J = INFINI
    }
    SINON
    {
        SAUVEGARDER G = (1-G)/(1-F)
    }
}
```

Listing B.2 – Algorithme fonction F, G et J

```
ENTREE : Liste des Points, Liste des Erreurs, fonction Noyau()

CONSTANTES :
    m = nombre pour avoir une matrice de quadrillage de taille m x m
    n = nombre de points
    aire = aire de la surface couverte par le noyau
    sigma = constante pour la fonction Noyau

VARIABLES :
    i = 0
    j = 0
    k = 0
    x = 0
    y = 0
    intensite = 0


POUR ( i de 0 à m )
{

  POUR ( j de 0 à m )
  {
        intensite = 0;

        // pour tous les points
        POUR ( k de 0 à n )
        {
            // calcul de l'ecart entre la position du quadrillage (i,j)
            // et la position du point k stocke dans les variables x et y
            (x,y) = CALCUL_ECART(i,j,k)

            intensite = intensite + Noyau(x,y,sigma)/Erreur(k)
        }

        // écriture des calculs
        SAUVEGARDER intensite/aire
  }
}
```

Listing B.3 – Algorithme calcul de l'intensité

C Représentations sous R

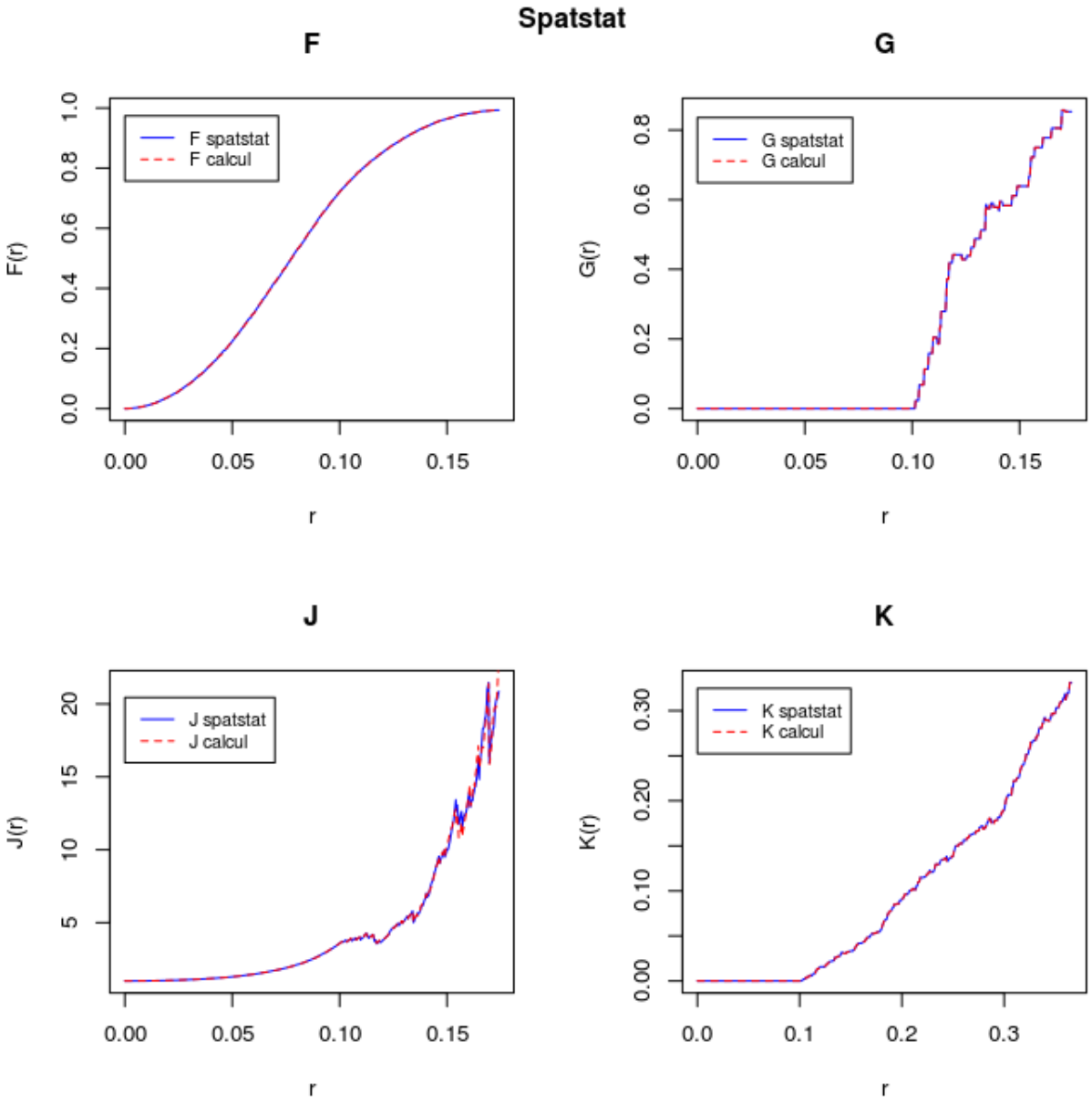


FIGURE C.1 – Comparaison des fonctions programmées avec celles calculées par spatstat

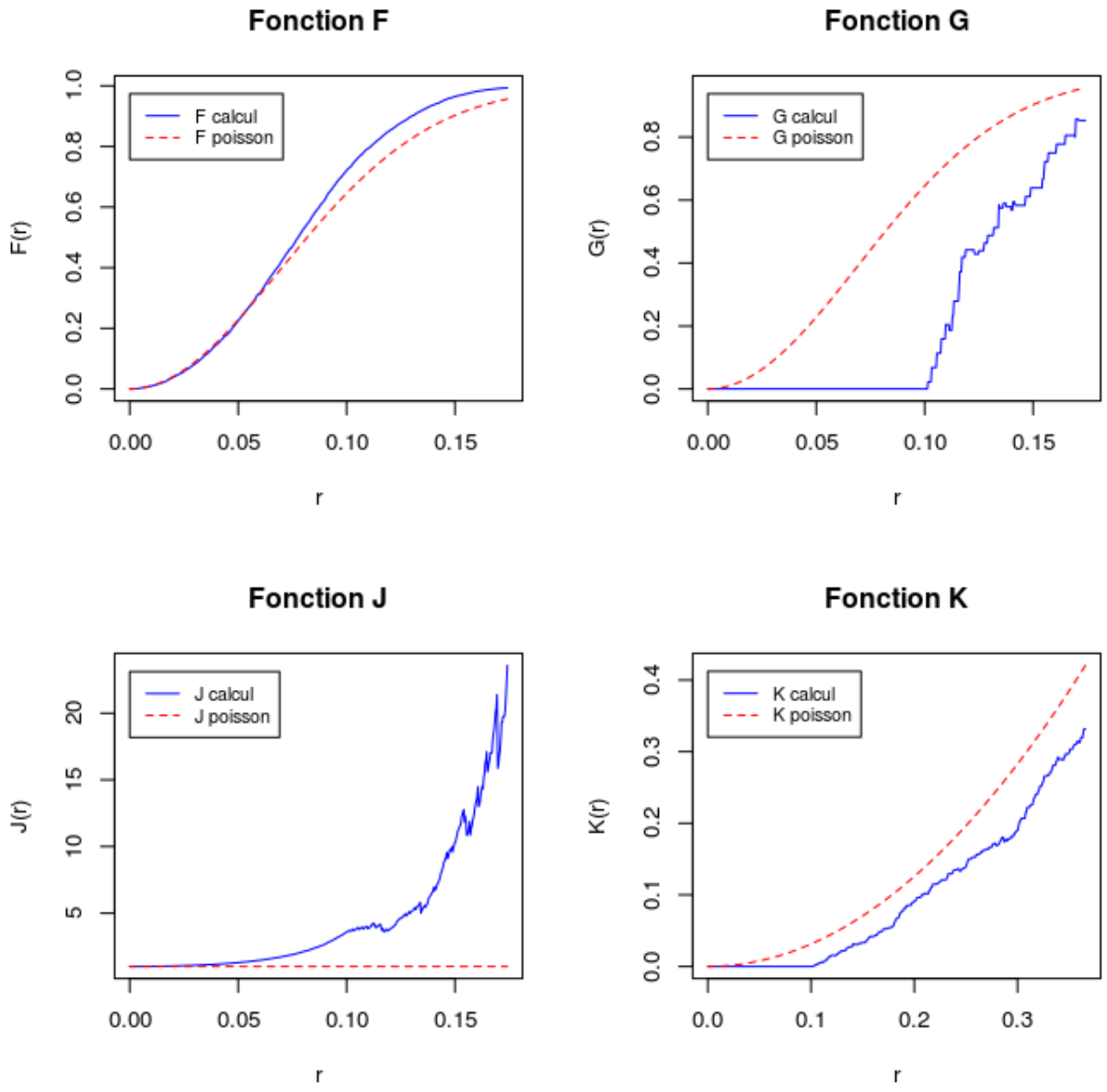


FIGURE C.2 – Comparaison des fonctions programmées avec courbes théoriques pour une répartition aléatoire

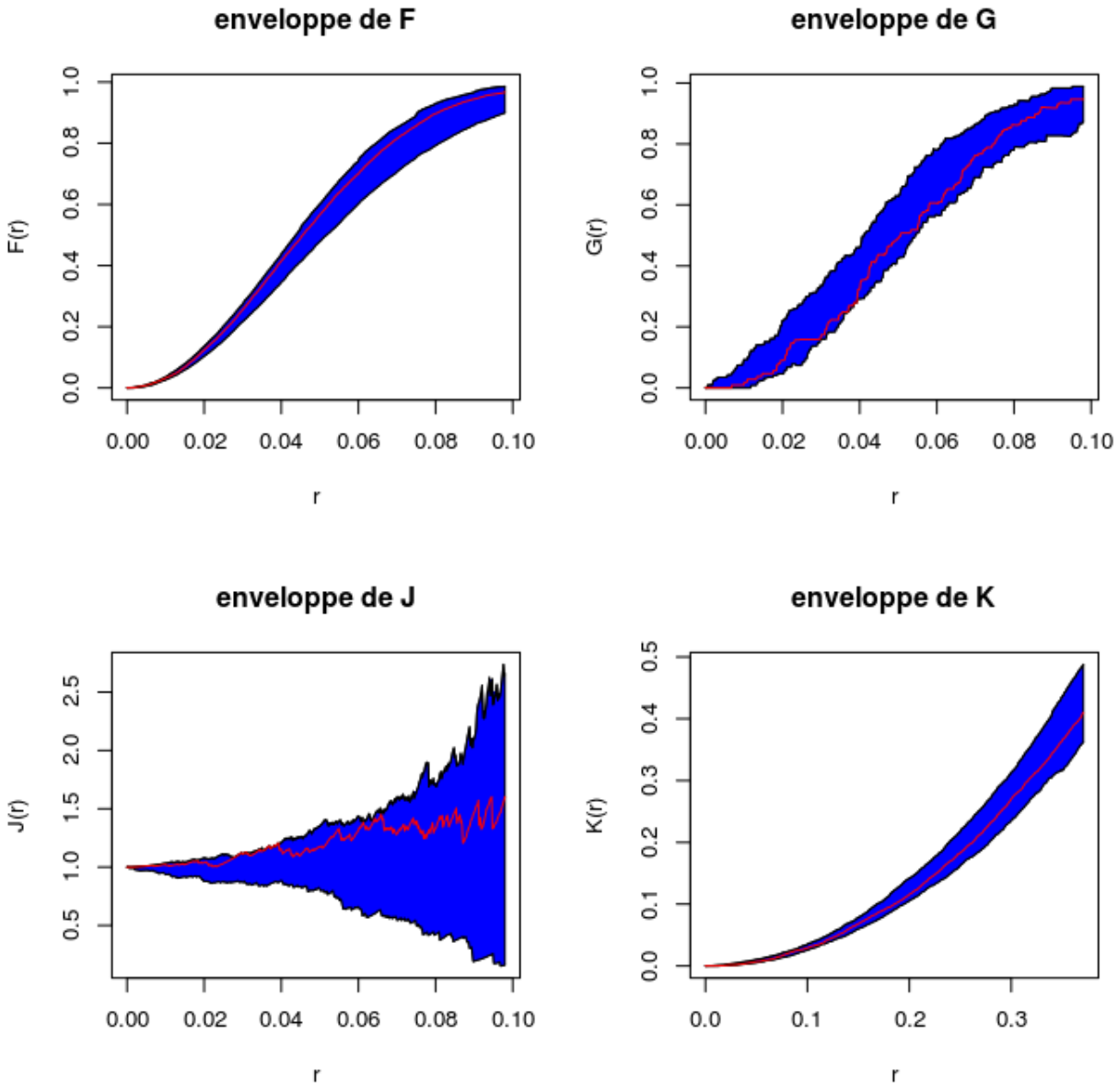


FIGURE C.3 – Exemple de résultat des tests d'enveloppes

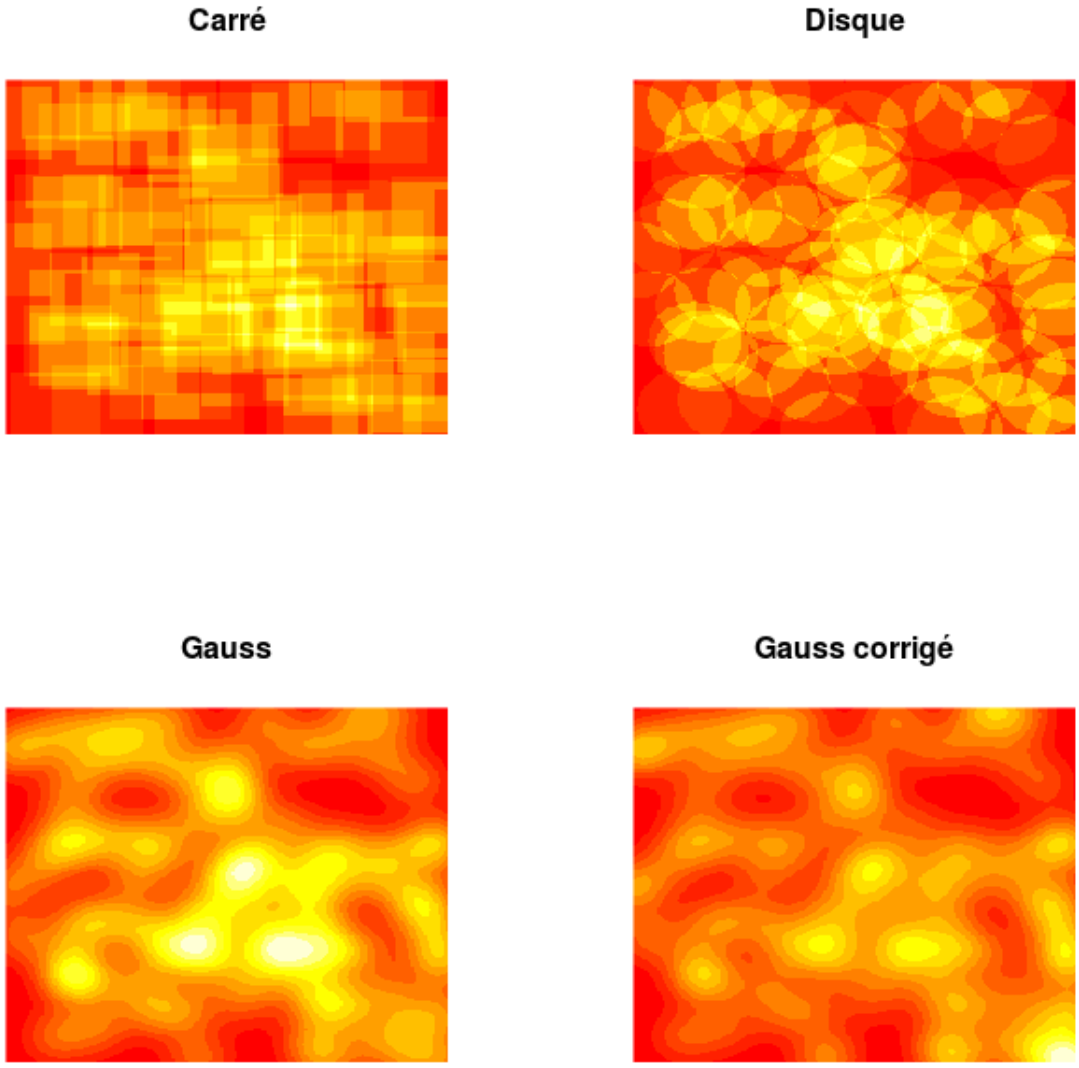


FIGURE C.4 – Exemple de résultat des calculs d'intensité

Résumé

Ce rapport présente le travail de stagiaire effectué dans l'équipe Probabilité-Statistique de l'Institut Elie Cartan de Lorraine. Ce projet s'inscrit dans le cadre d'un stage de 2ème année à TELECOM Nancy. Le stage a pour but la création d'outils informatiques pour traiter et analyser des données spatialisées en utilisant des processus ponctuels*. Les fonctions à implémenter testent si les points d'une configuration sont plus ou moins répartis aléatoirement.

Avec une génération de répartitions ponctuels créé par M. Radu Stoica, il est possible de faire une enveloppe de valeurs qui permet d'avoir une estimation plus précise de la tendance de répartition des points. Ainsi, les résultats de ce projet permettrons à M. Stoica de faire de futures analyses de situations plus précises.

Mots-clés : statistique, analyse, processus ponctuel

Abstract

This report present a trainee work for the Probability-Statistique team of the "Institut Elie Cartan de Lorraine". This project was realised in the framework of a 2nd year trainee in TELECOM Nancy. The goal of the traineeship is to create a computer tool in order to treat and analyze spatial data using point process. The functions to develop intend to estimate the ramdom degree of a point pattern.

With a point pattern generator created by Mr Radu Stoica, it is possible to make a data envelop which allow a more precise estimation of the point repartition trend. Thus, the results of this project would allow Mr Stoica to carry out more precise analysis in the futur.

Keywords : statistic, analysis, point process