Méthodes d'analyse spatiale : un grand bol d'R

Philippe Apparicio Jérémy Gelb

2025-05-18

Table des matières

Liste des figures

Listes des tableaux

Préface

Résumé : Ce livre vise à décrire une panoplie de méthodes d'analyse spatiale avec le logiciel ouvert R. Le contenu est pensé pour être accessible à tous et toutes, même à ceux et celles n'ayant presque aucune base en analyse de données spatiales ou en statistiques. La philosophie de ce livre est de donner toutes les clefs de compréhension et de mise en œuvre des méthodes abordées dans R. La présentation des méthodes est basée sur une approche compréhensive et intuitive plutôt que mathématique, sans pour autant négliger la rigueur statistique.

Remerciements : Ce manuel a été réalisé avec le soutien de la fabriqueREL. Fondée en 2019, la fabriqueREL est portée par divers établissements d'enseignement supérieur du Québec et agit en collaboration avec les services de soutien pédagogique et les bibliothèques. Son but est de faire des ressources éducatives libres (REL) le matériel privilégié en enseignement supérieur au Québec.

Maquette de la page couverture et identité graphique du livre : Andrés Henao Florez.

Mise en page : Philippe Apparicio et Marie-Hélène Gadbois Del Carpio.

Révision linguistique : Denise Latreille.

Pour citer cet ouvrage : Apparicio P. et J. Gelb (2024). *Méthodes d'analyse spatiales : un grand bol d'R*. Université de Sherbrooke, Département de géomatique appliquée. fabriqueREL. Licence CC BY-SA.

Un manuel sous la forme d'une ressource éducative libre

Pourquoi un manuel sous licence libre?

Les logiciels libres sont aujourd'hui très répandus. Comparativement aux logiciels propriétaires, l'accès au code source permet à quiconque de l'utiliser, de le modifier, de le dupliquer et de le partager. Le logiciel R, dans lequel sont mises en œuvre les méthodes d'analyse spatiale décrites dans ce livre, est d'ailleurs à la fois un langage de programmation et un logiciel libre (sous la licence publique générale GNU GPL2). Par analogie aux logiciels libres, il existe aussi des **ressources éducatives libres (REL)** « dont la licence accorde les permissions désignées par les 5R (**Retenir — Réutiliser — Réviser — Remixer — Redistribuer**) et donc permet nécessairement la modification » (***fabriqueREL***). La licence de ce livre, CC BY-SA (figure 1), permet donc de :

- **Retenir**, c'est-à-dire télécharger et imprimer gratuitement le livre. Notez qu'il aurait été plutôt surprenant d'écrire un livre payant sur un logiciel libre et donc gratuit. Aussi, nous aurions été très embarrassés que des personnes étudiantes avec des ressources financières limitées doivent payer pour avoir accès au livre, sans pour autant savoir préalablement si le contenu est réellement adapté à leurs besoins.
- **Réutiliser**, c'est-à-dire utiliser la totalité ou une section du livre sans limitation et sans compensation financière. Cela permet ainsi à d'autres personnes enseignantes de l'utiliser dans le cadre d'activités pédagogiques.
- **Réviser**, c'est-à-dire modifier, adapter et traduire le contenu en fonction d'un besoin pédagogique précis puisqu'aucun manuel n'est parfait, tant s'en faut! Le livre a d'ailleurs été écrit intégralement dans R avec Quatro. Quiconque peut ainsi télécharger gratuitement le code source du livre sur github et le modifier à sa guise (voir l'encadré intitulé *Suggestions d'adaptation du manuel*).
- **Remixer**, c'est-à-dire « combiner la ressource avec d'autres ressources dont la licence le permet aussi pour créer une nouvelle ressource intégrée » (***fabriqueREL***).
- **Redistribuer**, c'est-à-dire distribuer, en totalité ou en partie le manuel ou une version révisée sur d'autres canaux que le site Web du livre (par exemple, sur le site Moodle de votre université ou en faire une version imprimée).

La licence de ce livre, CC BY-SA (figure 1), oblige donc à :

- Attribuer la paternité de l'auteur dans vos versions dérivées, ainsi qu'une mention concernant les grandes modifications apportées, en utilisant la formulation suivante : Apparicio Philippe et Jérémy Gelb (2024). *Méthodes d'analyses spatiales : un grand bol d'R*. Université de Sherbrooke. fabriqueREL. Licence CC BY-SA.
- Utiliser la même licence ou une licence similaire à toutes versions dérivées.

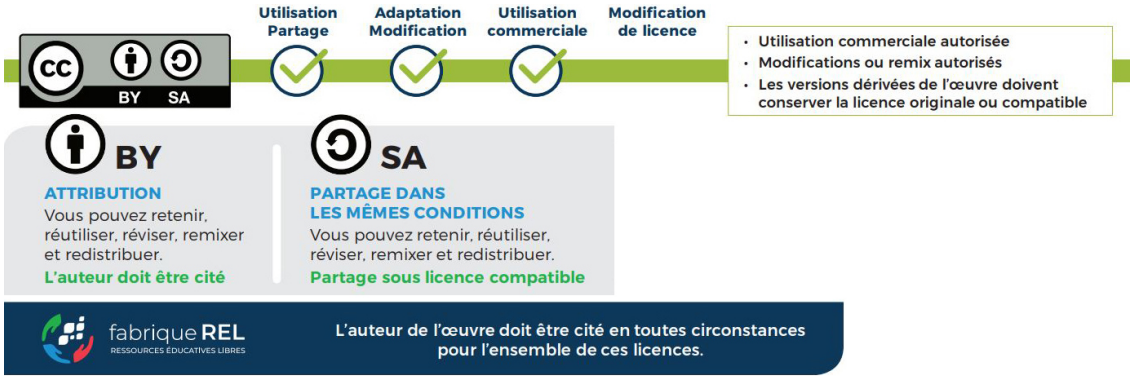


FIGURE 1 – Licence Creative Commons du livre

Astuce

Suggestions d'adaptation du manuel
Pour chaque méthode d'analyse spatiale abordée dans le livre, une description détaillée et une mise en œuvre dans R sont disponibles. Par conséquent, plusieurs adaptations du manuel sont possibles :

- Conserver uniquement les chapitres sur les méthodes ciblées dans votre cours.
- En faire une version imprimée et la distribuer aux personnes étudiantes.
- Modifier la description d'une ou de plusieurs méthodes en effectuant les mises à jour directement dans les chapitres.
- Insérer ses propres jeux de données dans les sections intitulées *Mise en œuvre dans R*.
- Modifier les tableaux et figures.
- Ajouter une série d'exercices.
- Modifier les quiz de révision.
- Rédiger un nouveau chapitre.
- Modifier des syntaxes R. Plusieurs *packages* R peuvent être utilisés pour mettre en œuvre telle ou telle méthode. Ces derniers évoluent aussi très vite et de nouveaux *packages* sont proposés fréquemment! Par conséquent, il peut être judicieux de modifier une syntaxe R du livre en fonction de ses habitudes de programmation dans R (utilisation d'autres *packages* que ceux utilisés dans le manuel par exemple) ou de bien mettre à jour une syntaxe à la suite de la parution d'un nouveau *package* plus performant ou intéressant.
- Toute autre adaptation qui permet de répondre au mieux à un besoin pédagogique.

Comment lire ce manuel?

Le livre comprend plusieurs types de blocs de texte qui en facilitent la lecture.

Package

Bloc *packages*
Habituellement localisé au début d'un chapitre, il comprend la liste des *packages* R utilisés pour un chapitre.

Objectif

Bloc objectif
Il comprend une description des objectifs d'un chapitre ou d'une section.

Note

Bloc notes
Il comprend une information secondaire sur une notion, une idée abordée dans une section.

Aller plus loin

Bloc pour aller plus loin
Il comprend des références ou des extensions d'une méthode abordée dans une section.

Astuce

Bloc astuce
Il décrit un élément qui vous facilitera la vie : une propriété statistique, un *package*, une fonction, une syntaxe R.

Attention

Bloc attention
Il comprend une notion ou un élément important à bien maîtriser.

Exercice

Bloc exercice
Il comprend un court exercice de révision à la fin de chaque chapitre.

Comment utiliser les données du livre pour reproduire les exemples?

Ce livre comprend des exemples détaillés et appliqués dans R pour chacune des méthodes abordées. Ces exemples se basent sur des jeux de données structurés et mis à disposition avec le livre. Ils sont disponibles sur le *repo github* dans le sous-dossier `data`, à l'adresse https://github.com/SerieBoldR/MethodesAnalyseSpatiale/tree/main/data.

Une autre option est de télécharger le *repo* complet du livre directement sur *github* (https://github.com/SerieBoldR/MethodesAnalyseSpatiale) en cliquant sur le bouton `Code`, puis le bouton `Download ZIP` (figure 2). Les données se trouvent alors dans le sous-dossier nommé `data`.

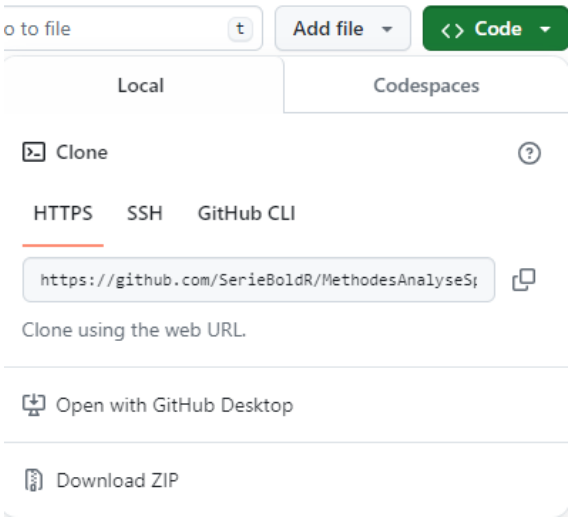


FIGURE 2 – Téléchargement de l'intégralité du livre

Liste des packages utilisés

Dans ce livre, nous utilisons de nombreux *packages* R que vous pouvez installer avec le code ci-dessous.

```
# Liste des packages
ListePackages <- c("classInt", "cluster", "ClustGeo", "concaveman", "dbscan", "dplyr", "factoextra",
                   "foreign", "future", "geocmeans", "ggplot2", "ggpubr", "ggthemes",
                   "gifski", "Gmedian", "gpx", "gtfstools", "igraph", "lubridate", "MASS",
                   "metR", "mgcv", "osmextract", "r5r", "raster", "RColorBrewer", "rgdal",
                   "rgeoda", "rgl", "rmapshaper", "sf", "sparr", "spatialreg", "spatstat",
                   "spdep", "spgwr", "spNetwork", "terra", "tmap", "viridis", "xlsx")
# Packages non installés dans la liste
PackagesNonInstalles <- ListePackages[!(ListePackages %in% installed.packages()[,"Package"])]
# Installation des packages manquants
if(length(new.packages)) install.packages(PackagesNonInstalles)
```

Structure du livre

Le livre est organisé autour de cinq grandes parties.

Partie 1. Données spatiales et R. Dans cette première partie, nous voyons comment importer, manipuler, cartographier et exporter des données spatiales dans R, principalement avec les *packages* `sf` pour les données vectorielles, `terra` pour les données matricielles (images) et `tmap` pour la cartographie (chapitre 1). Maîtriser les notions abordées dans ce chapitre constitue une étape préalable et indispensable à tout projet d'analyse spatiale. D'une part, avant d'analyser des données spatiales, il convient de les structurer (importation et manipulation) et de les explorer (cartographie). D'autre part, une fois la ou les méthodes d'analyse spatiale mises en œuvre, il convient de cartographier les résultats finaux et de les exporter au besoin dans un format de données géographiques (shapefile (`shp`), GeoPackage (`GPKG`), GeoJSON (`geojson`), sqlite (`sqlite`), GeoTiff, etc.).

Partie 2. Autocorrélation spatiale. L'autocorrélation spatiale est une notion fondamentale en analyse spatiale permettant de vérifier si les entités proches ou voisines ont tendance à être (dis)semblables en fonction d'un phénomène donné (Anselin 2019). Dans le chapitre 2, nous décrivons une panoplie de méthodes permettant de définir des matrices de pondération spatiale qui sont utilisées pour évaluer la dépendance spatiale d'une ou de plusieurs variables, soit les mesures d'autocorrélation spatiale globale et locale. La compréhension de la notion de dépendance spatiale et des différentes mesures d'autocorrélation spatiale est primordiale puisqu'elles sont largement mobilisées dans d'autres méthodes d'analyses spatiales décrites dans les chapitres suivants, notamment les régressions spatiales (chapitre 7) et les méthodes de classification spatiale (chapitre 8).

Partie 3. Méthodes de répartition ponctuelle et de détections d'agrégats spatiaux. Cette troisième partie comprend deux chapitres. Dans le chapitre 3, nous abordons les principales méthodes qui permettent de décrire un semis de points dans un espace géographique donné : fréquence et densité des points, analyse centrographique, arrangement spatial du semis de points (méthode du plus proche voisin, méthode des quadrats), cartographie de la densité (notamment l'estimation de la densité par noyau). Dans le chapitre 4, nous abordons deux familles de méthodes de détection d'agrégats spatiaux et spatio-temporels qui s'appliquent à des géométries différentes : les méthodes de classification basées sur la densité des points (couche de points), principalement les algorithmes DBSCAN et ST-DBSCAN, et les méthodes de balayage de Kulldorff (couche de polygones).

Partie 4. Analyses de données avec des réseaux de transport. Cette quatrième partie comprend aussi deux chapitres. Dans le chapitre 5, nous voyons comment construire un réseau multimode (voiture, marche, vélo, transport en commun) afin de calculer différentes mesures d'accessibilité spatiale dans R avec le *package* `R5R`. Dans le chapitre 6, nous décrivons

plusieurs méthodes permettant de décrire la distribution spatiale d'évènements localisés sur un réseau de rues avec le *package* `spNetwork`.

Partie 5. Méthodes de régression spatiale et de classification spatiale. Cette cinquième et dernière partie comprend deux chapitres. Le chapitre 7 est consacré à plusieurs méthodes de régression intégrant l'espace : modèles d'économétrie spatiale, modèles généralisés additifs (GAM) avec une *spline* bivariée sur les coordonnées géographiques, régressions géographiquement pondérées. Dans le chapitre 8, nous décrivons des méthodes permettant de regrouper des entités spatiales d'une région en plusieurs classes en fonction de leurs caractéristiques évaluées à partir de plusieurs variables. Nous distinguons deux types de méthodes de classification non supervisée spatiale : les méthodes de classification non supervisée avec contrainte spatiale visant à regrouper des entités spatiales en plusieurs régions avec une absence de mitage (algorithmes AZP, SKATER, REDCAP, etc.); les méthodes de classification non supervisée avec une dimension spatiale (méthode *ClustGeo* et classification floue c-moyennes spatiale).

Remerciements

De nombreuses personnes ont contribué à l'élaboration de ce manuel.

Ce projet a bénéficié du soutien pédagogique et financier de la ***fabriqueREL*** (ressources éducatives libres). Les différentes rencontres avec le comité de suivi nous ont permis de comprendre l'univers des ressources éducatives libres (REL) et notamment leurs fameux 5R (Retenir — Réutiliser — Réviser — Remixer — Redistribuer), de mieux définir le besoin pédagogique visé par ce manuel, d'identifier des ressources pédagogiques et des outils pertinents pour son élaboration. Ainsi, nous remercions chaleureusement les membres de la *fabriqueREL* pour leur soutien inconditionnel :

— Myriam Beaudet, bibliothécaire à l'Université de Sherbrooke.
— Marianne Dubé, coordonnatrice de la fabriqueREL, Université de Sherbrooke.
— Serge Piché, conseiller pédagogique, Université de Sherbrooke.
— Claude Potvin, conseiller en formation, Service de soutien à l'enseignement, Université Laval.

Nous remercions chaleureusement les personnes étudiantes du cours **GMQ405 - Modélisation et analyse spatiale** du Baccalauréat en géomatique appliquée à l'environnement et du Microprogramme de 1er cycle en géomatique appliquée du Département de géomatique appliquée de l'Université de Sherbrooke de la session d'été 2023 : Hermann B. Beaudin, Jérémie Durand, Marie-Hélène Gadbois Del Carpio, David Lapointe, Marc-Antoine Lecours-Toutloff, Rosemarie Légaré, Frédéric Malo, Anthony Mandeville, Gurwen Meret, Sarah Pion, Anne-Sophie Roy, William Scrive, Aldin Evrad Tampe et Kévin Therrien. Vos suggestions et commentaires nous ont permis d'améliorer grandement la version préliminaire de ce manuel qui a été utilisée dans le cadre de ce cours.

Nous remercions aussi les membres du comité de révision pour leurs commentaires et suggestions très constructifs. Ce comité est composé de quatre personnes étudiantes du Département de géomatique appliquée de l'Université de Sherbrooke :

— Gaël Machemin et Loek Pascaud, étudiants à la maîtrise en géomatique appliquée et télédétection (type recherche).
— Rosemarie Légaré, étudiante au baccalauréat en géomatique appliquée à l'environnement.

Marie-Hélène Gadbois Del Carpio, étudiante de troisième année au baccalauréat en géomatique appliquée à l'environnement a participé activement à la mise en page du manuel; elle est désormais une grande spécialiste des feuilles de style en cascade (CSS) et de Quarto.

Aussi, les discussions enrichissantes avec Geneviève Crevier et Amélie Fréchette ont largement contribué à définir et à bonifier la table des matières.

Finalement, nous remercions Denise Latreille, réviseure linguistique et chargée de cours à l'Université Sherbrooke, pour la révision du manuel.

À propos des auteurs

Philippe Apparicio est professeur titulaire au Département de géomatique appliquée de l'Université de Sherbrooke. Il y enseigne aux programmes de 1er et 2e cycles de géomatique les cours *Transport et mobilité durable*, *Modélisation et analyse spatiale* et *Géomatique appliquée à la gestion urbaine*. Durant les dernières années, il a offert plusieurs formations aux Écoles d'été du Centre interuniversitaire québécois de statistiques sociales (CIQSS). Géographe de formation, ses intérêts de recherche incluent la justice et l'équité environnementale, la mobilité durable, les pollutions atmosphérique et sonore, et le vélo en ville. Il a publié une centaine d'articles scientifiques dans différents domaines des études urbaines et de la géographie mobilisant la géomatique et l'analyse spatiale.

Jérémy Gelb a obtenu un doctorat en études urbaines à l'INRS en 2022 (*L'exposition des cyclistes aux pollutions atmosphérique et sonore en milieu urbain : comparaison empirique de plusieurs villes à travers le monde*), sous la supervision de Philippe Apparicio. Il utilise quotidiennement des systèmes d'information géographique (SIG) et des méthodes d'analyses spatiales. Il est tombé dans la marmite de l'*open source* avec le triptyque QGIS, R et Python au début de sa maîtrise. Il a développé deux *packages* : `geocmeans` et `spNetwork`, permettant respectivement d'effectuer des analyses de classification floue non supervisée pondérée spatialement et des estimations de densité par kernel sur réseau. Il travaille actuellement comme conseiller en science des données à l'Autorité régionale de transport métropolitain qui gère la planification du transport collectif dans la région de Montréal. Ces travaux portent sur la qualité des milieux urbains, l'accessibilité spatiale, le transport, l'équité environnementale, les SIG et l'analyse spatiale.

Philippe et Jérémy travaillent étroitement ensemble depuis plusieurs années. Avec d'autres collègues, ils ont copublié une vingtaine d'articles scientifiques et un manuel intitulé *Méthodes quantitatives en sciences sociales : un grand bol d'R*, avec le soutien de la fabriqueREL.

Partie 1. Données spatiales et R

1 Manipulation des données spatiales dans R

Dans ce chapitre, nous décrivons comment importer, manipuler et cartographier des données spatiales dans R. Pour une description plus détaillée du langage de programmation R – objets et expression, opérateurs, structures de données (vecteurs, matrices, *arrays*, *DataFrame*), importation et manipulation de données –, lisez le chapitre intitulé *Prise en main avec R* (Apparicio et Gelb 2022).

Package

Liste des *packages* utilisés dans ce chapitre

- Pour importer et manipuler des fichiers géographiques :
  - `sf` pour importer et manipuler des données vectorielles.
  - `rmapshaper` pour simplifier des géométries en conservant la topologie.
  - `terra` pour importer et manipuler des données *raster*.
  - `gpx` pour importer des coordonnées GPS au format *GPS eXchange Format*.
  - `foot` pour créer des enveloppes orientées sur les géométries.
  - `concaveman` pour créer des enveloppes concaves.
- Pour cartographier des données :
  - `ggplot2` est un *package* pour construire des graphiques qui peut être aussi utilisé pour visualiser des données spatiales.
  - `tmap` pour construire des cartes thématiques.
  - `RColorBrewer` pour construire une palette de couleur.
  - `ggpurb` pour combiner des graphiques et des cartes.
- Pour importer des tables attributaires :
  - `foreign` pour importer des fichiers *dBase*.
  - `xlsx` pour importer des fichiers Excel.

Attention

Pas de panique!
Ce document comprend de nombreuses notions sur l'importation, la manipulation et la cartographie de données spatiales dans R, soit des opérations que vous avez l'habitude de réaliser dans ArcGIS Pro ou QGIS.
Prenez le temps de lire ce premier chapitre à tête reposée et assurez-vous de bien comprendre chaque notion avant de passer à la suivante. L'objectif n'est pas de maîtriser parfaitement la syntaxe R pour toutes les opérations dès la première semaine!
Vous allez manipuler de nombreuses données spatiales avec R au fil de la lecture du livre. Par conséquent, n'hésitez pas à revenir sur ce chapitre lorsque nécessaire; considérez-le comme un aide-mémoire.

1.1 Importation de données géographiques

Note

Quels *packages* choisir pour importer et manipuler des données spatiales?
Pour les données vectorielles, il existe deux principaux *packages* (équivalent d'une librairie dans Python) : `sp` (Pebesma et Bivand 2005; Bivand, Pebesma et Gomez-Rubio 2013) et `sf` (Pebesma 2018). Puisque le *package* `sp` est progressivement délaissé par R, il est donc fortement conseillé d'utiliser `sf`.
Pour les données *raster*, il est possible d'utiliser les *packages* `raster` (Hijmans 2022a) et `terra` (Hijmans 2022b), dont le dernier, plus récent, semblerait plus rapide.
Cette transition de `sp` à `sf` et de `raster` à `terra` est assez récente et encore en cours durant l'écriture de ce livre. Il existe encore de nombreux *packages* basés sur `sp` et `raster`. Il est donc possible que vous ayez à les utiliser, car leur transition n'est peut-être pas encore effectuée. Notez que la façon dont ces anciens *packages* intègrent les données vectorielles et matricielles dans R est très différente de celle des nouveaux *packages*. À titre d'exemple, la fonction `sp::readOGR` lit un fichier *shapefile*, tout comme la fonction `sf::st_read`, mais la première produit un objet de type `SpatialDataFrame`, alors que la seconde produit un `tbl_df`. Dans le premier cas, les géométries et les données sont stockées dans deux éléments séparés, alors que dans le second cas, le `tbl_df` est un `data.frame` avec une colonne contenant les géométries.
Pour les personnes intéressées aux motivations ayant conduit à cette transition, consultez cet excellent article de blog. Il existe deux raisons principales : le mainteneur des *packages* `rgdal` et `rgeos` servant de fondation à `raster` et `sp` a pris sa retraite. À cela s'ajoutent le côté « vieille école » de ces *packages* (ayant plus de 20 ans!) et l'apparition de *packages* plus modernes. Il s'agit d'un bon exemple de ce qui peut arriver dans une communauté *open source* et des évolutions constantes de l'environnement R.
En résumé, privilégiez l'utilisation de `sf` et de `terra`.
Il convient d'installer les deux *packages*. Notez que l'installation d'un *package* requiert une connexion Internet, car R accède au répertoire de *packages CRAN* pour télécharger le *package* et l'installer sur votre ordinateur. Cette opération est réalisée avec la fonction `install.packages("nom du package")`. Notez qu'une fois que le *package* est installé, il est enregistré localement sur votre ordinateur et y reste à moins de le désinstaller avec la fonction `remove.packages("nom du package")`.
Autrement dit, il n'est pas nécessaire de les installer à chaque ouverture de R! Pour utiliser les fonctions d'un *package*, vous devez préalablement le charger avec la fonction `library("Nom du package")` (équivalent à la fonction `import` de Python).
Pour plus d'informations sur l'installation et le chargement de *packages*, consultez la section suivante (Apparicio et Gelb 2022).

1.1.1 Importation de données vectorielles

La fonction `st_read` de `sf` permet d'importer une multitude de formats de données géographiques, comme des fichiers *shapefile* (`shp`), *GeoPackage* (`GPKG`), *GeoJSON* (`geojson`), *sqlite* (`sqlite`), *geodatabase d'ESRI* (`FileGDB`), *Geoconcept* (`gxt`), *Keyhole Markup Language* (`kml`), *Geography Markup Language* (`gml`), etc.

1.1.1.1 Importation d'un fichier shapefile

Le code R ci-dessous permet d'importer des couches géographiques au format *shapefile*. Notez que la fonction `list.files(pattern = ".shp")` renvoie préalablement la liste des couches *shapefile* présentes dans le dossier de

travail.

```
## Chargement des packages
library("sf")
library("terra")
library("tmap")
library("ggplot2")
library("ggpubr")
library("foreign")
library("xlsx")
library("rmapshaper")
library("RColorBrewer")
## Obtention d'une liste des shapefiles dans le dossier de travail
list.files(path = "data/chap01/shp", pattern = ".shp")
```

```
 [1] "AbidjanPtsGPS.shp"
 [2] "AbidjanSegRue.shp"
 [3] "Arrondissements.shp"
 [4] "IncidentsSecuritePublique.shp"
 [5] "Installations_sportives_et_recreatives.shp"
 [6] "Pistes_cyclables.shp"
 [7] "PolyX.shp"
 [8] "PolyY.shp"
 [9] "Quebec.shp"
[10] "Segments_de_rue.shp"
```

```
## Importation des shapefiles avec sf
Arrondissements <- st_read("data/chap01/shp/Arrondissements.shp", quiet=TRUE)
InstallationSport <- st_read("data/chap01/shp/Installations_sportives_et_recreatives.shp", quiet=TRUE)
PistesCyclables <- st_read("data/chap01/shp/Pistes_cyclables.shp", quiet=TRUE)
Rues <- st_read("data/chap01/shp/Segments_de_rue.shp", quiet=TRUE)
```

Regardons à présent la structure des couches importées. Pour ce faire, nous utilisons la fonction `head(nom du DataFrame, n=2)`; notez que le paramètre `n` permet de spécifier le nombre des premiers enregistrements à afficher. Les informations suivantes sont ainsi disponibles :

- `6 fields` : six champs attributaires (`TYPE`, `DETAIL`, `NOM`, `SURFACE`, `ECLAIRAGE`, `OBJECTID`).
- `Geometry type POINT` : le type de géométrie est **point**.
- `Bounding box:  xmin: -8009681 ymin: 5686891 xmax: -8001939 ymax: 5696536` : les quatre coordonnées définissant l'enveloppe de la couche.
- `Projected CRS: WGS 84 / Pseudo-Mercator` : la projection cartographique. Ici, une projection cartographique utilisée par Google Maps et OpenStreetMap.
- La géométrie est enregistrée dans le champ `geometry`. Pour le premier enregistrement, nous avons la valeur `POINT (-8001939 5686891)`, soit un point avec les coordonnées géographiques (x,y) entre parenthèses.

```
head(InstallationSport, n=2)   # Visualisation des deux premiers enregistrements
```

```
Simple feature collection with 2 features and 6 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -8009681 ymin: 5686891 xmax: -8001939 ymax: 5696536
Projected CRS: WGS 84 / Pseudo-Mercator
   TYPE DETAIL                      NOM SURFACE ECLAIRAGE OBJECTID
1 Aréna   <NA>    Aréna Eugène-Lalonde    <NA>      <NA>        1
2 Aréna   <NA> Aréna Philippe-Bergeron    <NA>      <NA>        2
                  geometry
1 POINT (-8001939 5686891)
2 POINT (-8009681 5696536)
```

```
names(InstallationSport)       # Noms de champs (colonnes)
```

```
[1] "TYPE"      "DETAIL"    "NOM"       "SURFACE"   "ECLAIRAGE" "OBJECTID"
[7] "geometry"
```

```
View(InstallationSport)        # Afficher l'ensemble de la table attributaire
```

Explorons les types de géométries et la projection des autres couches avec le code ci-dessous. En résumé, les types de géométries sont :

- Des géométries simples
  - `point` : un seul point.
  - `linestring` : une séquence de deux points et plus formant une ligne.
  - `polygon` : un seul polygone formé par une séquence de points pouvant contenir un ou plusieurs polygones intérieurs formant des trous.
- Des géométries multiples
  - `multipoint` : plusieurs points pour une même observation.
  - `multilinestring` : plusieurs lignes pour une même observation.
  - `multipolygon` : plusieurs polygones pour une même observation.
- Une collection de géométries (`Geometrycollection`) qui peut contenir différents types de géométries décrites ci-dessus pour une même observation.

```
head(PistesCyclables, n=2)
```

```
Simple feature collection with 2 features and 3 fields
Geometry type: MULTILINESTRING
Dimension:     XY
Bounding box:  xmin: -8010969 ymin: 5666202 xmax: -7997972 ymax: 5697954
Projected CRS: WGS 84 / Pseudo-Mercator
```

```
                      NOM OBJECTID SHAPE__Len                       geometry
1     Axe de la Massawippi        1   13944.09 MULTILINESTRING ((-8010969 ...
2 Axe de la Saint-François        2   19394.28 MULTILINESTRING ((-8001909 ...
```

```
head(Rues, n=2)
```

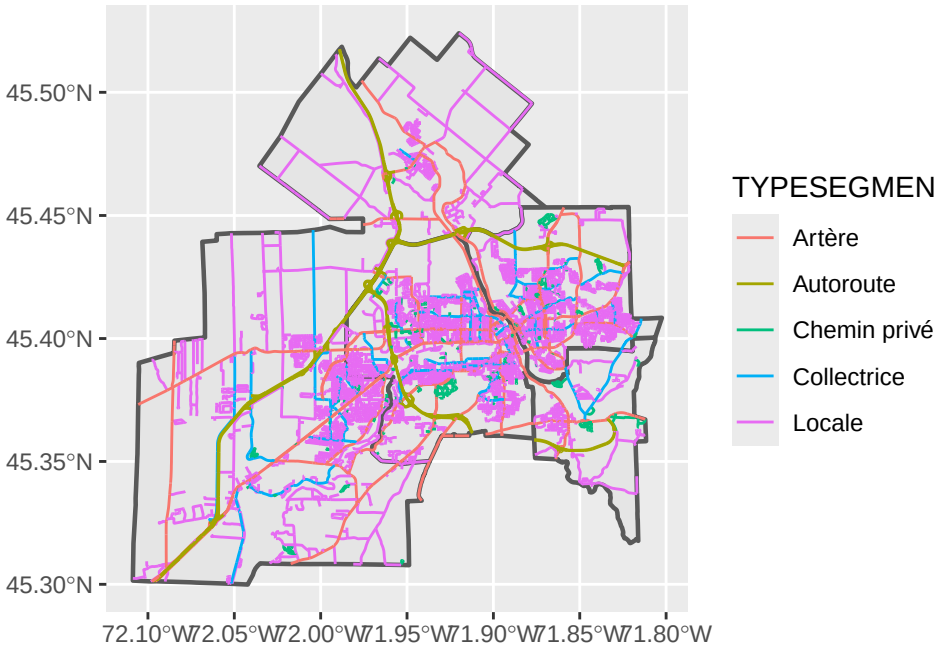
```
Simple feature collection with 2 features and 16 fields
Geometry type: MULTILINESTRING
Dimension:     XY
Bounding box:  xmin: -8013896 ymin: 5681299 xmax: -8008810 ymax: 5695980
Projected CRS: WGS 84 / Pseudo-Mercator
  ID         TOPONYMIE NOROUTE NUMEROCIVI NUMEROCI_1 NUMEROCI_2 NUMEROCI_3
1  1 Rue Oliva-Turgeon      NA          0          0          0          0
2  9      Rue Melville      NA          0          0          0          0
     NOMGENERIQ TYPERUE TYPESEGMEN VITESSE         TYPESENSUN MUNICIPALI
1 OLIVA-TURGEON     Rue     Locale      50 Pas de sens unique      43027
2      MELVILLE     Rue     Locale      50 Pas de sens unique      43027
  OBJECTID SHAPE__Len              CIRCULATIO                       geometry
1        1   114.7781 Interdite en tout temps MULTILINESTRING ((-8008810 ...
2        2   114.4441 Interdite en tout temps MULTILINESTRING ((-8013782 ...
```

```
head(Arrondissements, n=2)
```

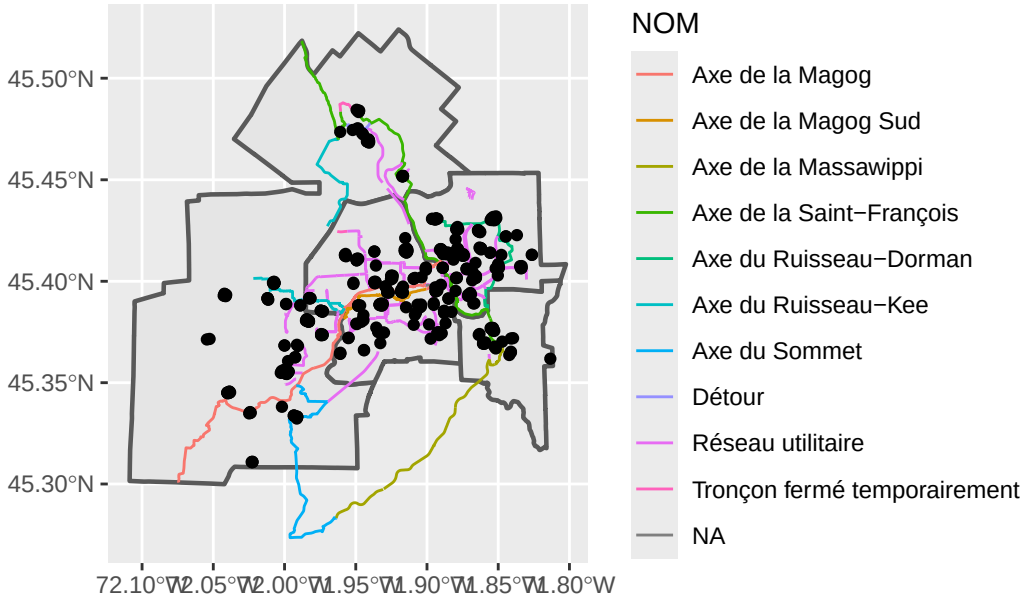
```
Simple feature collection with 2 features and 2 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: -8027109 ymin: 5668860 xmax: -8000502 ymax: 5704391
Projected CRS: WGS 84 / Pseudo-Mercator
  NUMERO                                                        NOM
1      1 Arrondissement de Brompton-Rock Forest-Saint-Élie-Deauville
2      4                                 Arrondissement des Nations
                       geometry
1 POLYGON ((-8005013 5702777,...
2 POLYGON ((-8005680 5690860,...
```

Visualisons quelques couches importées avec `ggplot()`.

```
## Arrondissements et rues
ggplot()+ geom_sf(data = Arrondissements, lwd = .8)+
  geom_sf(data = Rues, aes(colour = TYPESEGMEN))
```




```
## Arrondissements, pistes cyclables et installations sportives
ggplot()+ geom_sf(data = Arrondissements, lwd = .8)+
  geom_sf(data = PistesCyclables, aes(colour = NOM), lwd = .5)+
  geom_sf(data = InstallationSport)
```



1.1.1.2 Importation d'une couche dans un GeoPackage

Pour importer une couche stockée dans un *GeoPackage* (GPKG), vous devez spécifier le fichier et la couche avec respectivement les paramètres `dsn` et `layer` de la fonction `st_read`. Le code ci-dessous permet d'importer les secteurs

de recensement de la région métropolitaine de recensement de Sherbrooke pour l'année 2021. Notez que la fonction `st_layers(dsn)` permet d'obtenir la liste des couches contenues dans le fichier GPKG, avec le type de géométrie, les nombre d'entités spatiales et de champs, et la projection cartographique pour chacune d'elles.

```
## Nom du fichier GPKG
fichierGPKG <- "data/chap01/gpkg/Recen2021Sherbrooke.gpkg"
## Liste des couches dans le GPKG
st_layers(dsn=fichierGPKG, do_count = TRUE)
```

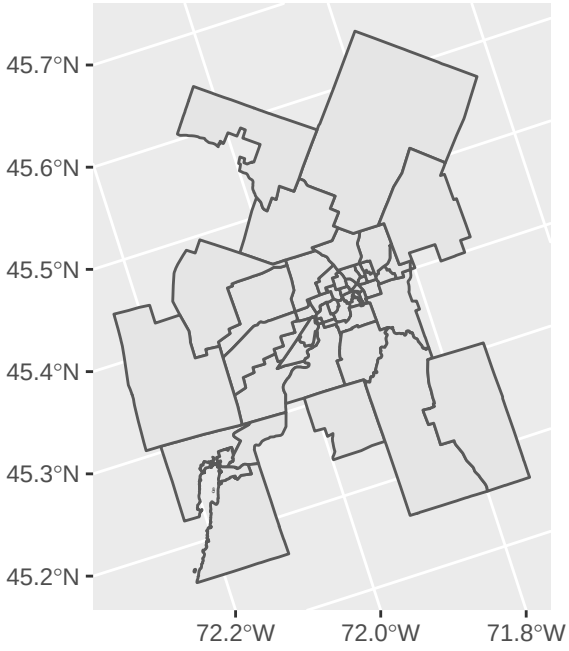
```
Driver: GPKG
Available layers:
           layer_name geometry_type features fields
1             SherbRMR Multi Polygon        1     12
2             SherbSDR Multi Polygon       11     12
3              SherbSR Multi Polygon       50     10
4              SherbAD Multi Polygon      333     10
5              SherbIL Multi Polygon     2734     12
6         RegionEstrie Multi Polygon        1      5
7           sdr_Estrie Multi Polygon       89      6
8         DREstrie2021 Multi Polygon        7      6
9 DivisionsRecens2021 Multi Polygon       98      6
                          crs_name
1 NAD83 / Statistics Canada Lambert
2 NAD83 / Statistics Canada Lambert
3 NAD83 / Statistics Canada Lambert
4 NAD83 / Statistics Canada Lambert
5 NAD83 / Statistics Canada Lambert
6 NAD83 / Statistics Canada Lambert
7 NAD83 / Statistics Canada Lambert
8 NAD83 / Statistics Canada Lambert
9 NAD83 / Statistics Canada Lambert
```

```
## Importation d'une couche
SR.RMRSherb <- st_read(dsn = fichierGPKG,
                       layer = "SherbSR", quiet=TRUE)
## Affichage des deux premiers enregistrements
head(SR.RMRSherb, n=2)
```

```
Simple feature collection with 2 features and 10 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 7762066 ymin: 1271201 xmax: 7765357 ymax: 1274082
Projected CRS: NAD83 / Statistics Canada Lambert
                IDUGD      SRIDU   SRNOM SUPTERRE PRIDU SRpop_2021 SRtlog_2021
1 2021S05074330001.00 4330001.00 0001.00   3.1882    24       5637        2918
```

```
2 2021S05074330002.00 4330002.00 0002.00   0.8727    24       1868        1169
  SRrhlog_2021 RMRcode   HabKm2                            geom
1         2756     433 1768.082 MULTIPOLYGON (((7764998 127...
2         1063     433 2140.484 MULTIPOLYGON (((7763361 127...
```

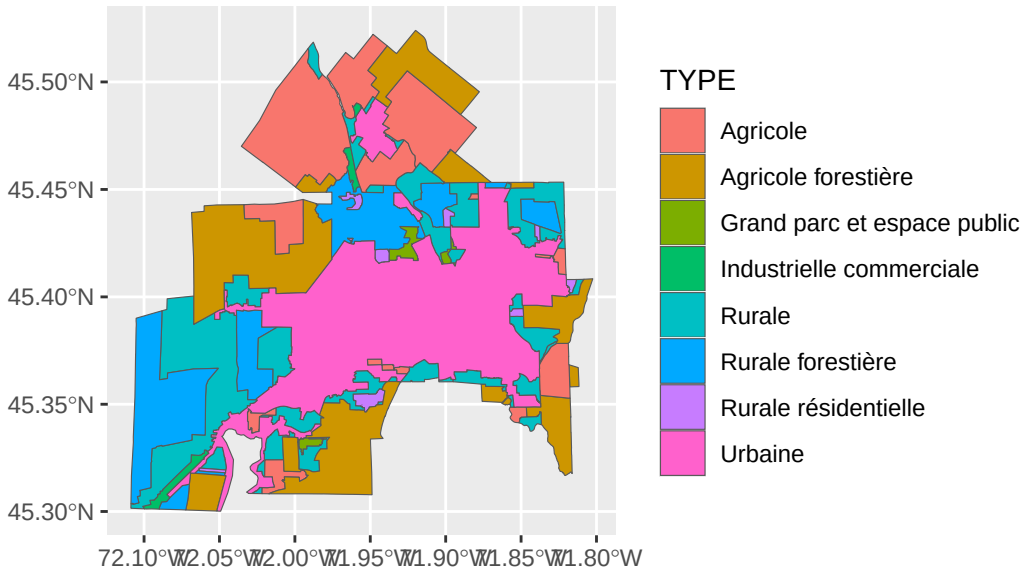
```
## Visualisation rapide des secteurs avec ggplot
ggplot()+ geom_sf(data = SR.RMRSherb, lwd = .5)
```



1.1.1.3 Importation d'une couche dans une geodatabase d'ESRI

La logique est la même qu'avec un *GeoPackage*, nous spécifions le chemin de la *geodatabase* et la couche avec les paramètres `dsn` et `layer`.

```
AffectDuTerritoire <- st_read(dsn = "data/chap01/geodatabase/Sherbrooke.gdb",
                              layer = "AffectationsDuTerritoire", quiet=TRUE)
## Visualisation des affectations du sol avec ggplot
ggplot()+ geom_sf(data = AffectDuTerritoire, aes(fill = TYPE), lwd = .2)
```



1.1.1.4 Importation de données GPS

En géomatique appliquée, il est fréquent de collecter des données sur le terrain avec un appareil GPS. Les données ainsi collectées peuvent être enregistrées dans différents formats de données dépendamment de l'appareil GPS utilisé : *GPS eXchange Format* (GPX), *Garmin's Flexible and Interoperable Data Transfer* (FIT), *Training Center XML* (TCX), etc.

Importation de coordonnées GPS longitude/latitude au format csv

Une personne ayant collecté des données sur le terrain pourrait aussi vous les transmettre dans un fichier *csv* (fichier texte délimité par des virgules). Il convient d'importer le fichier de coordonnées GPS dans R dans un *DataFrame* (avec la fonction `read.csv`). Une fois importé, nous constatons qu'il comprend trois champs :

- `id` : un champ identifiant avec des valeurs uniques.
- `lon` : longitude.
- `lat` : latitude.

Les points sont projetés en longitude/latitude (WGS84 long/lat, EPSG : 4326).

```
## Importation du fichier csv
PointsGPS <- read.csv("data/chap01/gps/pointsGPS.csv")
head(PointsGPS)
```

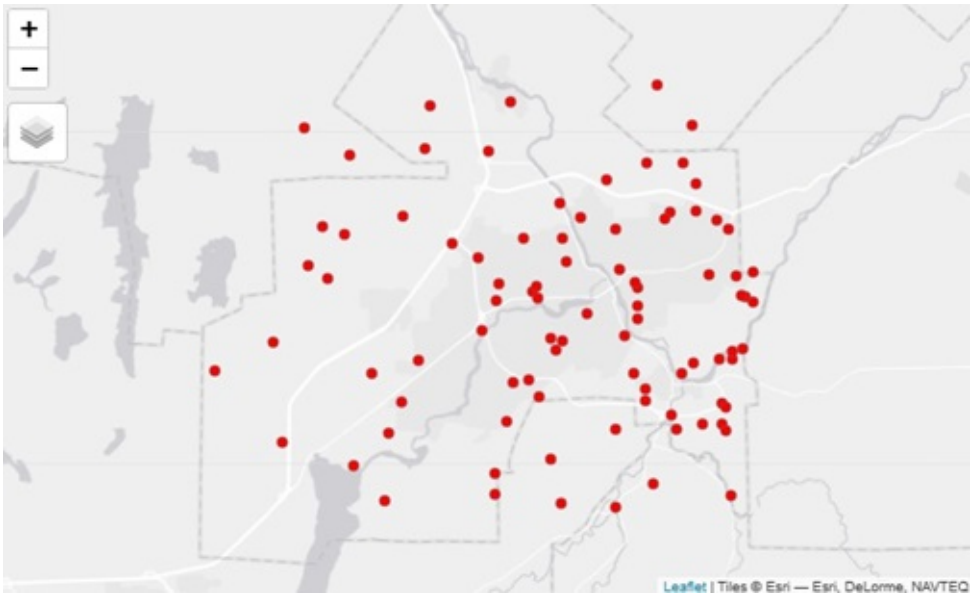
```
  id       lon      lat
1  1 -71.99985 45.36010
2  2 -71.99096 45.37535
3  3 -71.98444 45.46964
4  4 -72.09873 45.37126
5  5 -72.04880 45.41035
6  6 -71.95000 45.32570
```

Pour convertir le *DataFrame* en un objet `sf`, nous utilisons la fonction `st_as_sf` en spécifiant les champs pour les coordonnées et la projection cartographique.

```
## Importation du fichier csv
PointsGPS <- st_as_sf(PointsGPS, coords = c("lon","lat"), crs = 4326)
```

Les points ainsi créés sont localisés dans la ville de Sherbrooke.

```
## Affichage des points avec le package tmap
library("tmap")
tmap_mode("view") ## Mode actif de tmap
tm_shape(PointsGPS)+
  tm_dots(size = 0.05, shape = 21, col = "red")
```



Importation de coordonnées GPS au format GPX

Le format GPX est certainement le format de stockage et d'échange de coordonnées GPS le plus utilisé. Les informations géographiques (x,y) et temporelles (date et heure) sont respectivement enregistrées en degrés longitude/latitude (projection WSG) (WGS84, EPSG : 4326) et en temps universel coordonné (UTC, format ISO 8601).

Pour importer un fichier GPX, nous utilisons le *package* `gpx`. S'il n'est pas installé sur votre ordinateur, lancez la commande `install.packages("gpx")` dans la console de R; n'oubliez pas de le charger avec `library("gpx")`! Ensuite, importez le fichier GPX avec la fonction `read_gpx`, enregistrez la trace GPS dans un *DataFrame* et convertissez-la en objet `sf`.

```
library("gpx")
## Importation du fichier GPX
TraceGPS <- read_gpx("data/chap01/gps/TraceGPS.gpx")
## Cette trace GPS comprend trois listes : routes, tracks et waypoints
## Les points sont stockés dans tracks
names(TraceGPS)
```

```
[1] "routes"    "tracks"    "waypoints"
```

```
## Pour visualiser les données, il suffit de lancer la ligne
## ci-dessous (mise en commentaire car le résultat est un peu long...)
# head(TraceGPS)
TraceGPS <- TraceGPS$tracks$`ID1_PA_2021-12-03_TRAJET01.gpx`
## Conversion du DataFrame en objet sf
TraceGPS <- st_as_sf(TraceGPS, coords = c("Longitude","Latitude"), crs = 4326)
## Visualisation des premiers enregistrements
head(TraceGPS, n=2)
```

```
Simple feature collection with 2 features and 4 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -4.022827 ymin: 5.327383 xmax: -4.022825 ymax: 5.327387
Geodetic CRS:  WGS 84
  Elevation                Time extensions Segment ID
1      40.8 2021-12-03 08:38:49         NA          1
2      40.6 2021-12-03 08:38:50         NA          1
                    geometry
1 POINT (-4.022827 5.327387)
2 POINT (-4.022825 5.327383)
```

La trace GPS correspond à un trajet réalisé à vélo à Abidjan (Côte d'Ivoire) le 3 décembre 2021. Cette trace a été obtenue avec une montre Garmin et comprend un point chaque seconde.

```
tmap_mode("view")
tm_basemap(leaflet::providers$OpenStreetMap)+
tm_shape(TraceGPS)+
  tm_dots(size = 0.001, shape = 21, col = "red")
```


Le Plateau
Ambassade
Blockauss
codjro
Jérusalem
Treichville
Marcory
Zone 3
Pont Général-de-Gaulle
Gare Abidjan-Lagune
Gare de Treichville
Plateau-Centre
Plateau Sud
Station Service
Leaflet | © OpenStreetMap contributors

Aller plus loin

La structure de la classe `sf`

La classe `sf` est composée de trois éléments (figure 1.1) :

- L'objet **`simple feature geometry (sfg)`** est la géométrie d'une observation. Tel que vu plus haut, elle est une géométrie simple (`point`, `linestring`, `polygon`), multiple (`multipoint`, `multilinestring`, `multipolygon`) ou une collection de géométries différentes (`Geometrycollection`). Pour définir chacune de ces géométries, nous utilisons les méthodes `st_point()`, `st_linestring()`, `st_polygon()`, `st_multipoint()`, `st_multilinestring()`, `st_multipolygon()` et `Geometrycollection()`.
- L'objet **`simple feature column (sfc)`** est simplement une liste de `simple feature geometry (sfg)`. Elle représente la colonne `geometry` d'une couche vectorielle `sf`.
- L'objet **`data.frame`** correspond à la table attributaire.

Une **`simple feature`** correspond ainsi à une observation (ligne) d'un objet `sf`, soit une entité spatiale comprenant l'information sémantique (attributs) et l'information spatiale (géométrie).

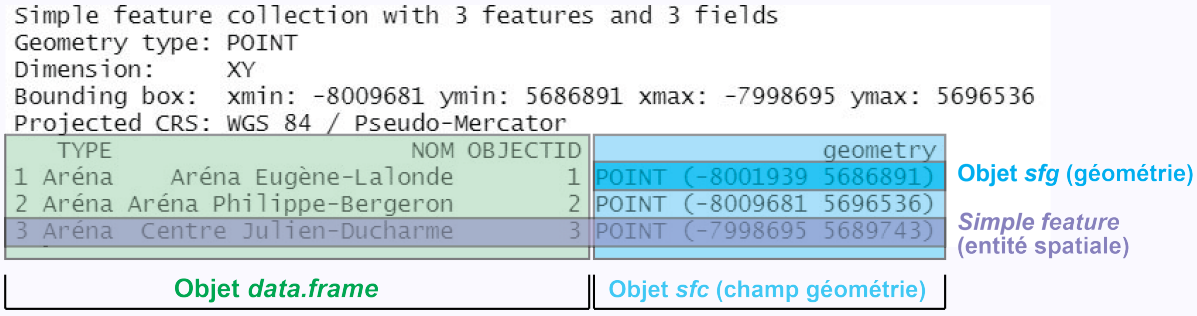


FIGURE 1.1 – Structure de la classe `sf`

Voyons un exemple concret : créons une couche `sf` comprenant les trois entités spatiales décrites dans la figure 1.1.

```
## Création des géométries : simple feature geometry (sfg)
point1 = st_point(c(-8001939, 5686891))
point2 = st_point(c(-8009681, 5696536))
point3 = st_point(c(-7998695, 5689743))
## Création d'une liste de géométries : simple feature geometry (sfc)
## avec la projection cartographique EPSG 3857
points_sfc = st_sfc(point1, point2, point3, crs = 3857)
## Création de la table attributaire : objet data.frame
table_attr = data.frame(TYPE = c("Aréna", "Aréna","Aréna"),
                        NOM = c("Aréna Eugène-Lalonde",
                                "Aréna Philippe-Bergeron",
                                "Centre Julien-Ducharme"),
                        OBJECTID = c(1, 2, 3))
## Création de l'objet sf
Arena_sf = st_sf(table_attr, geometry = points_sfc)
## Le résultat est bien identique à celui de la figure ci-dessus
head(Arena_sf)
```

```
Simple feature collection with 3 features and 3 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -8009681 ymin: 5686891 xmax: -7998695 ymax: 5696536
Projected CRS: WGS 84 / Pseudo-Mercator
   TYPE                     NOM OBJECTID                  geometry
1 Aréna    Aréna Eugène-Lalonde        1 POINT (-8001939 5686891)
2 Aréna Aréna Philippe-Bergeron        2 POINT (-8009681 5696536)
```

1.1.2 Importation de données raster

La fonction `terra::rast` permet d'importer des images de différents formats (GeoTiff, ESRI, ENVI, ERDAS, BIN, GRID, etc.). Nous importons ci-dessous cinq feuillets de modèles numériques d'altitude (MNA) à l'échelle du 1/20000 couvrant la ville de Sherbrooke. La figure 1.2 présente l'un d'entre eux.

```
## Liste des fichiers GeoTIFF dans le dossier
list.files(path="data/chap01/raster", pattern = ".tif")
```

```
 [1] "f21e05_101.tif"         "f21e05_101.tif.aux.xml" "f21e05_201.tif"
 [4] "f21e05_201.tif.aux.xml" "f21e12_101.tif"         "f21e12_101.tif.aux.xml"
 [7] "f31h08_102.tif"         "f31h08_102.tif.aux.xml" "f31h08_202.tif"
[10] "f31h08_202.tif.aux.xml"
```

```
## Importation des fichiers
f21e05_101 <- terra::rast("data/chap01/raster/f21e05_101.tif")
f21e05_201 <- terra::rast("data/chap01/raster/f21e05_201.tif")
f31h08_102 <- terra::rast("data/chap01/raster/f31h08_102.tif")
f31h08_202 <- terra::rast("data/chap01/raster/f31h08_202.tif")
f21e12_101 <- terra::rast("data/chap01/raster/f21e12_101.tif")
## Visualisation des informations sur l'image f21e05_101
f21e05_101
```

```
class       : SpatRaster
dimensions  : 1409, 2798, 1  (nrow, ncol, nlyr)
resolution  : 9e-05, 9e-05  (x, y)
extent      : -72.00095, -71.74913, 45.24907, 45.37588  (xmin, xmax, ymin, ymax)
coord. ref. : lon/lat NAD83 (EPSG:4269)
source      : f21e05_101.tif
name        : f21e05_101
min value   :   143.4273
max value   :   423.5806
```

```
# Visualisation de l'image
terra::plot(f21e05_101,
     main="Modèle numérique d'altitude à l'échelle de 1/20 000 (f21e05_101)")
```

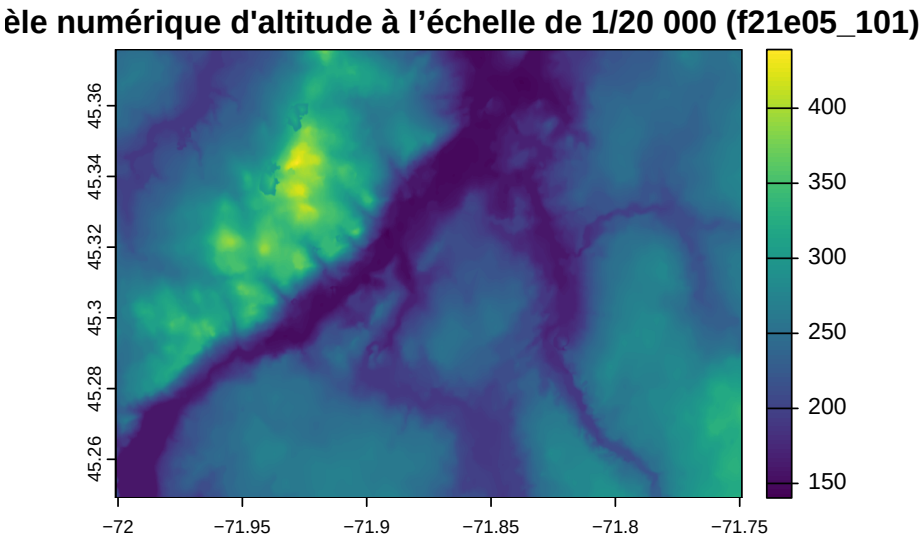


Figure 1.2 – Modèle numérique d'élévation au 1/20000 (feuillet f21e05_101)

1.2 Manipulation de données vectorielles

Objectif

***Package* `sf` et opérations géométriques**

Le *package* `sf` est une librairie extrêmement complète permettant de réaliser une multitude d'opérations géométriques sur des couches vectorielles comme dans un système d'information géographique (SIG). Notre objectif n'est pas de toutes les décrire, mais d'aborder les principales. Au fil de vos projets avec `sf`, vous apprendrez d'autres fonctions. Pour ce faire, n'hésitez pas à consulter :

- Une belle *Cheatsheet* sur `sf`. Allez y jeter un œil, cela vaut la peine!
- Sur le site CRAN de `sf`, vous trouverez plusieurs vignettes explicatives (exemples de code documentés).
- La documentation complète en PDF.

La syntaxe `methods(class = 'sfc')` renvoie la liste des méthodes implémentées dans le *package* `sf`. Pour accéder à l'aide en ligne de l'une d'entre elles, écrivez simplement `?Nom de la fonction` (ex. : `?st_buffer`).

```
methods(class = 'sfc')
```

```
 [1] [                            [<-
 [3] as.data.frame                c
 [5] coerce                       format
 [7] fortify                      identify
 [9] initialize                   ms_clip
[11] ms_dissolve                  ms_erase
[13] ms_explode                   ms_filter_islands
[15] ms_innerlines                ms_lines
[17] ms_points                    ms_simplify
[19] Ops                          points
[21] print                        rep
[23] scale_type                   show
[25] slotsFromS3                  st_area
[27] st_as_binary                 st_as_grob
[29] st_as_s2                     st_as_sf
[31] st_as_text                   st_bbox
[33] st_boundary                  st_break_antimeridian
[35] st_buffer                    st_cast
[37] st_centroid                  st_collection_extract
[39] st_concave_hull              st_convex_hull
[41] st_coordinates               st_crop
[43] st_crs                       st_crs<-
[45] st_difference                st_exterior_ring
[47] st_geometry                  st_inscribed_circle
[49] st_intersection              st_intersects
[51] st_is                        st_is_full
[53] st_is_valid                  st_line_merge
[55] st_m_range                   st_make_valid
[57] st_minimum_rotated_rectangle st_nearest_points
[59] st_node                      st_normalize
[61] st_point_on_surface          st_polygonize
[63] st_precision                 st_reverse
[65] st_sample                    st_segmentize
[67] st_set_precision             st_shift_longitude
[69] st_simplify                  st_snap
[71] st_sym_difference            st_transform
[73] st_triangulate               st_triangulate_constrained
```

1.2.1 Fonctions relatives à la projection cartographique

Les trois principales fonctions relatives à la projection cartographique des couches vectorielles sont :

- `st_crs(x)` pour connaître la projection géographique d'un objet `sf`.
- `st_transform(x, cr)` pour modifier la projection cartographique.
- `st_is_longlat(x)` pour vérifier si les coordonnées sont en degrés longitude/latitude.

```
## Importation d'un shapefile pour la province de Québec
ProvinceQc <- st_read("data/chap01/shp/Quebec.shp", quiet = TRUE)
## La projection est EPSG:3347 - NAD83 / Statistics Canada Lambert,
## soit la projection conique conforme de Lambert
st_crs(ProvinceQc)
```

```
Coordinate Reference System:
  User input: NAD83 / Statistics Canada Lambert
  wkt:
PROJCRS["NAD83 / Statistics Canada Lambert",
    BASEGEOGCRS["NAD83",
        DATUM["North American Datum 1983",
            ELLIPSOID["GRS 1980",6378137,298.257222101,
                LENGTHUNIT["metre",1]]],
        PRIMEM["Greenwich",0,
            ANGLEUNIT["degree",0.0174532925199433]],
        ID["EPSG",4269]],
    CONVERSION["Statistics Canada Lambert",
        METHOD["Lambert Conic Conformal (2SP)",
            ID["EPSG",9802]],
        PARAMETER["Latitude of false origin",63.390675,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8821]],
        PARAMETER["Longitude of false origin",-91.8666666666667,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8822]],
        PARAMETER["Latitude of 1st standard parallel",49,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8823]],
        PARAMETER["Latitude of 2nd standard parallel",77,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8824]],
        PARAMETER["Easting at false origin",6200000,
            LENGTHUNIT["metre",1],
            ID["EPSG",8826]],
        PARAMETER["Northing at false origin",3000000,
            LENGTHUNIT["metre",1],
            ID["EPSG",8827]]],
    CS[Cartesian,2],
```

```
        AXIS["(E)",east,
            ORDER[1],
            LENGTHUNIT["metre",1]],
        AXIS["(N)",north,
            ORDER[2],
            LENGTHUNIT["metre",1]],
    USAGE[
        SCOPE["Topographic mapping (small scale)."],
        AREA["Canada - onshore and offshore - Alberta; British Columbia; Manitoba; New Brunswick; Newfoundland a
        BBOX[38.21,-141.01,86.46,-40.73]],
    ID["EPSG",3347]]
```

```
## Reprojection de la couche en WGS84 long/lat (EPSG:4326)
ProvinceQc.4326 <- st_transform(ProvinceQc, crs = 4326)
## longitude/latitude?
st_is_longlat(ProvinceQc)
```

```
[1] FALSE
```

```
st_is_longlat(ProvinceQc.4326)
```

```
[1] TRUE
```

La figure 1.3 démontre bien que les deux couches sont projetées différemment.

```
Map1 <- ggplot()+geom_sf(data = ProvinceQc)+coord_sf(crs = st_crs(ProvinceQc))+
        labs(subtitle = "Conique conforme de Lambert (EPSG : 3347)")

Map2 <- ggplot()+geom_sf(data = ProvinceQc.4326)+coord_sf(crs = st_crs(ProvinceQc.4326))+
        labs(subtitle = "WGS84 long/lat (EPSG : 4326)")

comp_plot <- ggarrange(Map1, Map2, ncol = 2, nrow = 1)
annotate_figure(comp_plot,
                top = text_grob("Province de Québec",
                                face = "bold", size = 12, just = "center")
                )
```

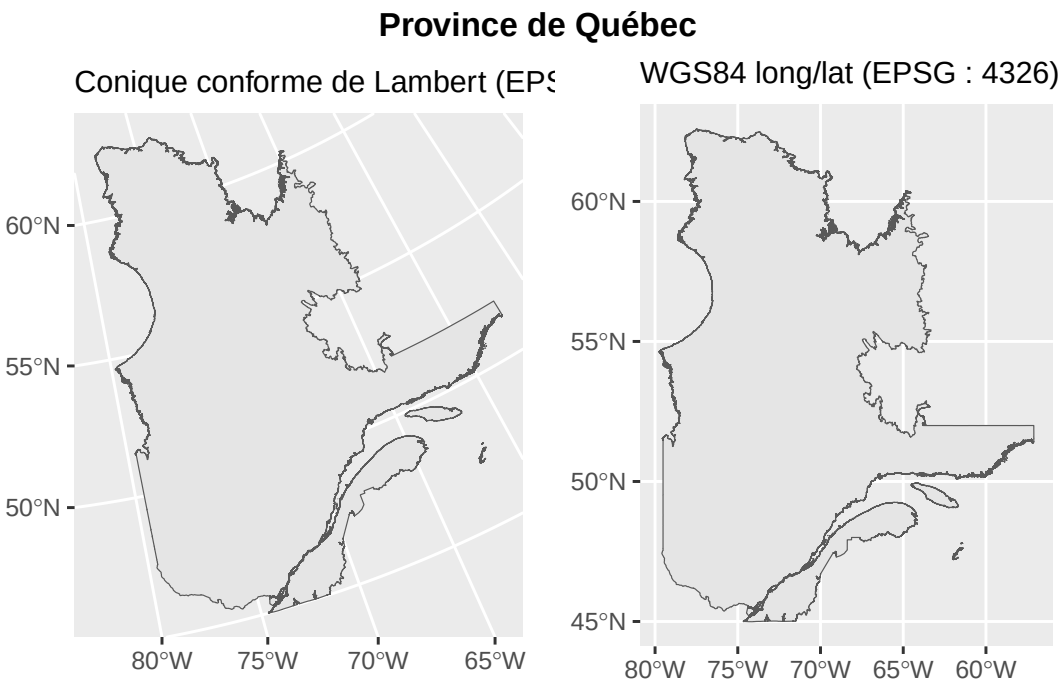


FIGURE 1.3 – Deux projections cartographiques

1.2.2 Fonctions d'opérations géométriques sur une couche

Il existe une quinzaine de fonctions d'opérations géométriques sur une couche dans le *package* `sf` dont le résultat renvoie de nouvelles géométries (voir la documentation suivante). Nous décrivons ici uniquement celles qui nous semblent les plus utilisées :

- `st_bbox(x)` renvoie les coordonnées minimales et maximales des géométries d'un objet `sf`. Pour créer l'enveloppe d'un objet `sf`, il suffit donc d'écrire `st_as_sfc(st_bbox(x))`.
- `st_boundary(x)` renvoie les limites (contours) des géométries d'un objet `sf`.
- `st_convex_hull(x)` crée l'enveloppe convexe des géométries d'un objet `sf`.
- `st_combine(x)` regroupe les géométries d'un objet `sf` en une seule géométrie, sans les réunir ni résoudre les limites internes.
- `st_union(x)` fusionne les géométries d'un objet `sf` en une seule géométrie.
- `st_buffer(x, dist, endCapStyle = c("ROUND", "FLAT", "SQUARE"), joinStyle = c("ROUND", "MITRE", "BEVEL"))` crée des zones tampons d'une distance définie avec le paramètre `dist`. Cette fonction s'applique à des points, à des lignes et à des polygones.
- `st_centroid(x)` crée des points au centre de chaque géométrie d'un objet `sf`. Elle s'applique donc à des lignes et à des polygones.
- `st_point_on_surface(x)` crée un point au centre de chaque polygone d'un objet `sf` .
- `st_simplify(x, dTolerance)` simplifie les contours de géométries (lignes ou polygones) avec une tolérance exprimée en mètres (paramètre `dTolerance`) d'un objet `sf` .
- `st_voronoi(x, bOnlyEdges = TRUE)` crée des polygones de Thiessen, appelés aussi polygones de Voronoï pour des points. Attention, le paramètre `bOnlyEdges = TRUE` renvoie des lignes tandis que `bOnlyEdges = FALSE` renvoie des polygones.

1.2.2.1 Enveloppe et union d'une couche

Le code ci-dessous crée une enveloppe (en bleu) et un polygone fusionné (en rouge) pour les arrondissements de la ville de Sherbrooke (figure 1.4). La couche résultante de l'opération `st_as_sfc(st_bbox(x))` est ainsi l'équivalent des outils `Emprise` de QGIS et `Minimum Bounding Geometry (Geometry Type = Envelope)` d'ArcGIS Pro.

```
## Enveloppe sur les arrondissements de la ville de Sherbrooke
Arrond.Enveloppe <- st_as_sfc(st_bbox(Arrondissements))
## Fusionne les géométries en une seule en résolvant les limites internes
Arrond.Union <- st_union(Arrondissements)
```

```
tmap_mode("plot")
tm_shape(Arrond.Enveloppe) + tm_borders(col = "blue", lwd=2)+
  tm_shape(Arrond.Union) + tm_borders(col = "red", lwd=2)+
  tm_layout(frame = FALSE)+
  tm_scale_bar(c(0,5,10))
```

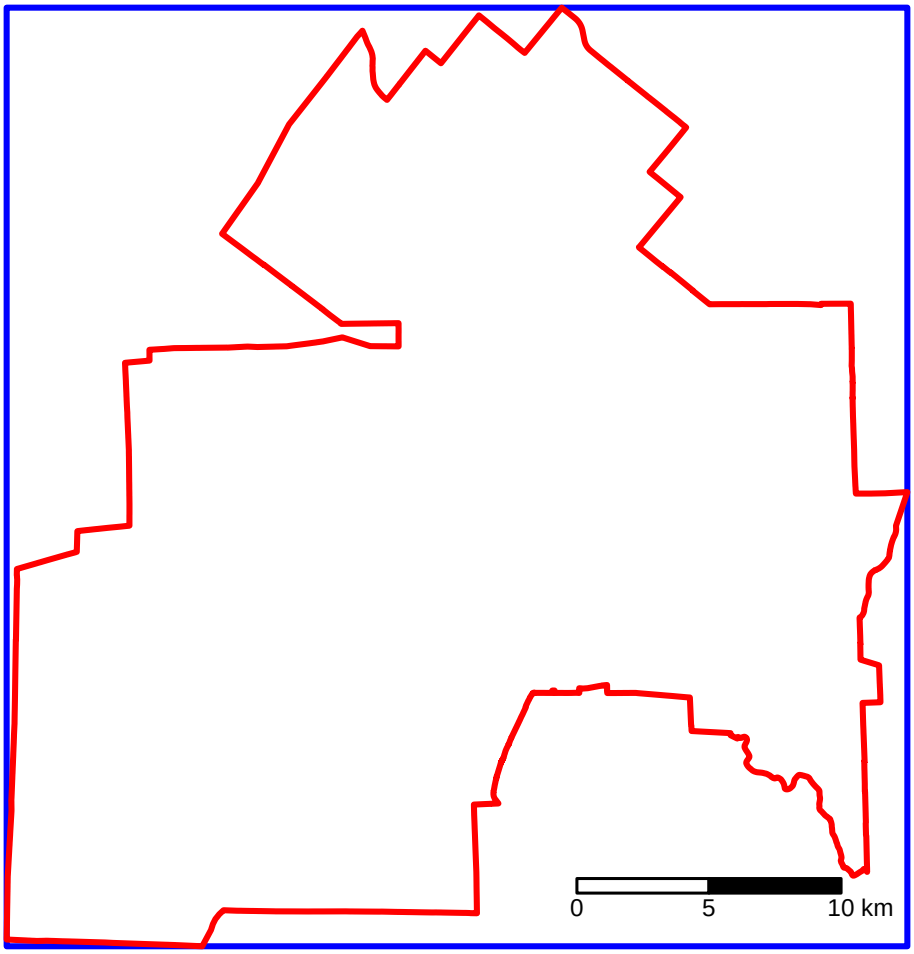


Figure 1.4 – Enveloppe sur une couche

1.2.2.2 Enveloppe orientée

La fonction `st_bbox` de `sf` produit des rectangles englobant des géométries qui sont orientées nord-sud. Il est possible de générer des rectangles orientés autour de géométries pour minimiser leur emprise et ainsi mieux représenter l'orientation de la géométrie initiale. Il n'existe pas de fonction dans `sf` pour le faire, mais le *package* `foot` offre une implémentation facile d'utilisation. Notez que `foot` n'est pas déposé sur CRAN et doit être téléchargé depuis *Github* avec la ligne de code ci-dessous.

```
devtools::install_github("wpgp/foot", build_vignettes = FALSE)
```

La couche résultante de l'opération `fs_mbr(x, returnShape = TRUE)` (figure 1.5, b) est ainsi l'équivalent des outils `Emprise orientée minimale (OMBB)` de QGIS et `Minimum Bounding Geometry (Geometry Type = Rectangle by area)` d'ArcGIS Pro.

```
# devtools::install_github("wpgp/foot", build_vignettes = TRUE)
library(foot, quietly = TRUE)
## Rectangles (enveloppes) orientés
rectangles_oriented <- fs_mbr(Arrondissements, returnShape = TRUE)
rectangles_oriented <- st_as_sf(rectangles_oriented,
                                crs = st_crs(Arrondissements))
rectangles_oriented$NOM <- Arrondissements$NOM
## Rectangles non orientés (nord-sud)
st_bbox_by_feature = function(x) {
  x = st_geometry(x)
  f <- function(y) st_as_sfc(st_bbox(y))
  do.call("c", lapply(x, f))
}
rectangles <- st_as_sf(st_bbox_by_feature(Arrondissements),
                       crs = st_crs(Arrondissements))
rectangles$NOM <- Arrondissements$NOM
```

1.2.2.3 Centroïdes et centre de surface

Le code ci-dessous extrait les centres géométriques, c'est-à-dire les centroïdes (en bleu) et les points à l'intérieur des polygones (en rouge) pour les arrondissements de la ville de Sherbrooke (figure 1.6). Ces deux opérations correspondent aux outils `centroïdes` et `Point dans la surface` de QGIS et `Feature to Point (avec l'option Inside)` d'ArcGIS Pro.

```
## Centroïdes et points dans les polygones sur les arrondissements
Arrond.centroide <- st_centroid(Arrondissements)
Arrond.pointpoly <- st_point_on_surface(Arrondissements)
```

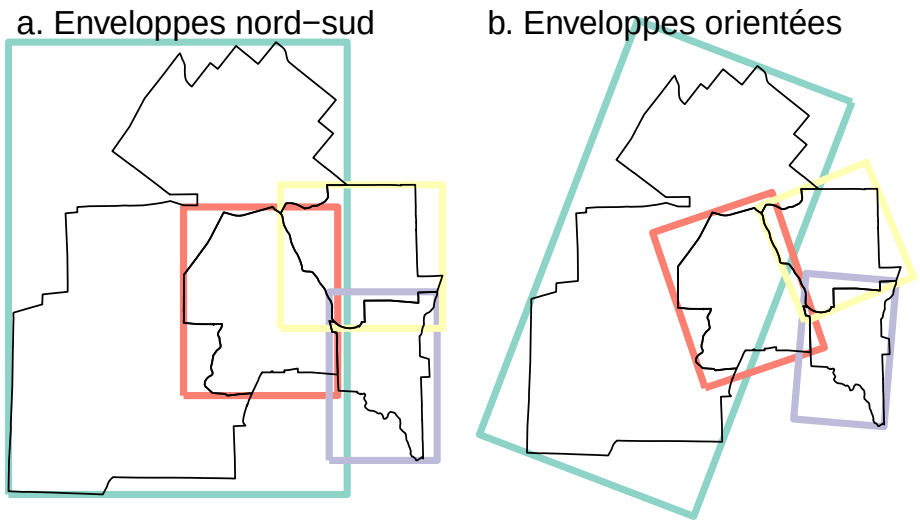


FIGURE 1.5 – Enveloppes classiques et orientées

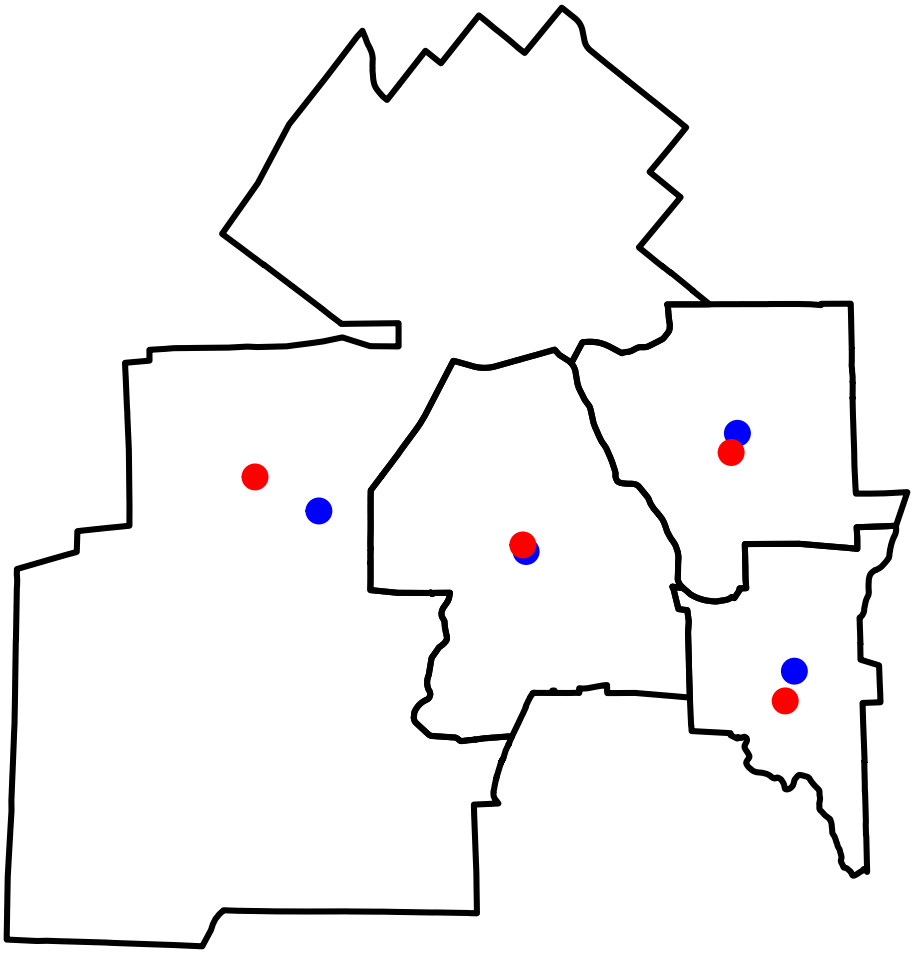


FIGURE 1.6 – Centroïdes et points à l'intérieur des polygones

1.2.2.4 Zone tampon (buffer)

Une simple ligne de code permet de créer des zones tampons (équivalent des outils `Analyse vectorielle/Tampon` dans QGIS et `Buffer` dans ArcGIS Pro). Une fois les zones créées, utilisez la fonction `st_union` pour fusionner les tampons en un polygone (figure 1.7).

```
## Zones tampons de 1000 mètres autour des installations sportives et récréatives
InstSports.buffer <- st_buffer(InstallationSport, dist = 1000)
## Si vous souhaitez fusionner les zones tampons, utilisez la fonction st_union
InstSports.bufferUnion <- st_union(InstSports.buffer)
## Zones tampons de 500 mètres autour des lignes
PistesCyclables.buffer <- st_buffer(PistesCyclables, dist = 500)
PistesCyclables.bufferUnion <- st_union(PistesCyclables.buffer)
```

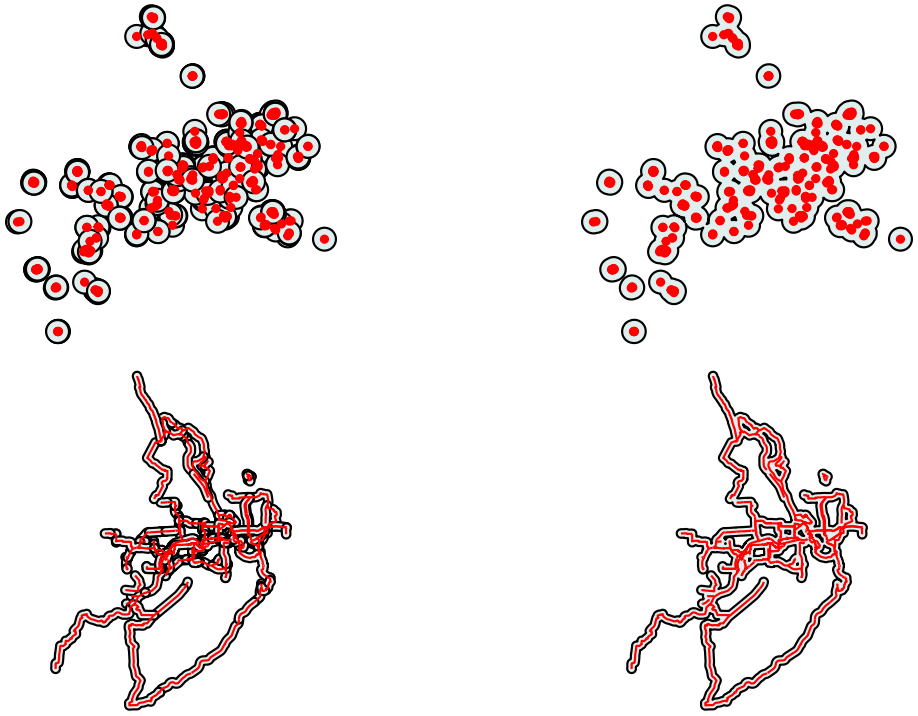


Figure 1.7 – Zones tampons

Notez que pour des polygones, il est possible de créer des polygones intérieurs comme suit : `st_buffer(x, dist = -Valeur)`. Par exemple, le code ci-dessous crée des polygones de 200 mètres autour et à l'intérieur du parc du Mont-Bellevue de la ville de Sherbrooke (figure 1.8).

```
## Importation de la couche des aires aménagées de la ville de Sherbrooke
AiresAmenag <- st_read(dsn = "data/chap01/geodatabase/Sherbrooke.gdb",
                       layer = "AiresAmenagees", quiet = TRUE)
## Sélection du parc du Mont-Bellevue
MontBellevue <- subset(AiresAmenag, NOM == "Parc du Mont-Bellevue")
## Création d'une zone tampon autour du parc
```

```
MontBellevue.ZTA500 <- st_buffer(MontBellevue, dist = 200)
## Création d'une zone tampon à l'intérieur du parc
MontBellevue.ZTI500 <- st_buffer(MontBellevue, dist = -200)
```

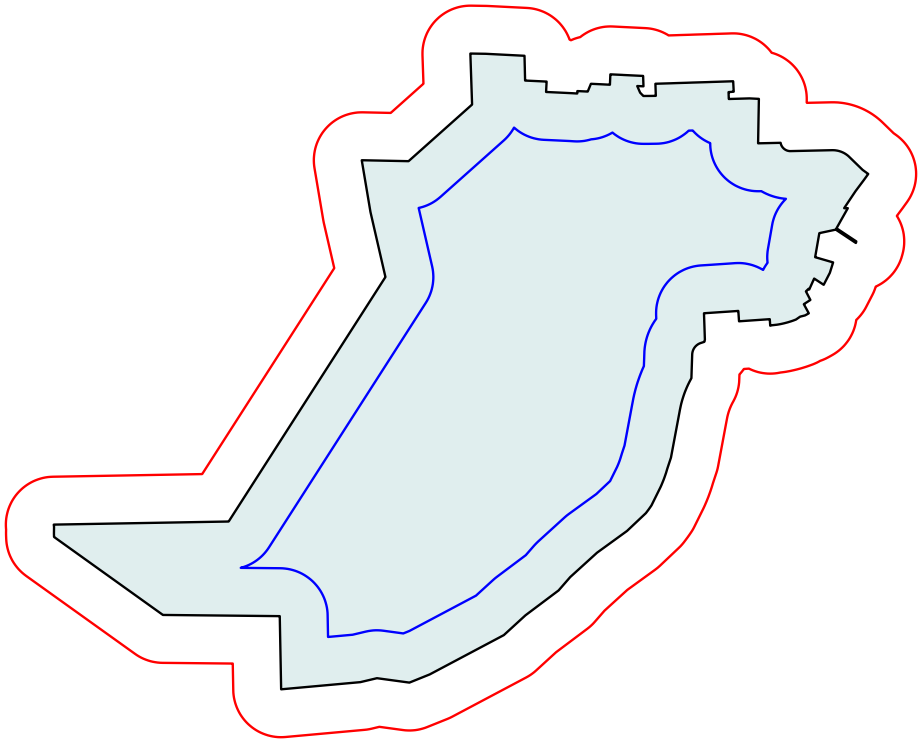


FIGURE 1.8 – Zone tampon intérieure et zone tampon extérieure

1.2.2.5 Simplification de géométries

La simplification ou généralisation d'une couche de lignes ou de polygones permet de supprimer des sommets tout en gardant le même nombre de géométries dans la couche résultante. Cette opération peut être réalisée dans QGIS avec l'outil `simplifier` et dans ArcGIS Pro avec l'outil `Generalize`. Deux raisons principales peuvent motiver le recours à cette opération :

- La réduction de la taille du fichier, surtout si la couche est utilisée pour de la cartographie interactive sur Internet avec des formats vectoriels comme le SVG (*Scalable Vector Graphics*), le KML ou le GeoJSON.
- L'utilisation de la couche à une plus petite échelle cartographique nécessitant la suppression de détails.

Le code suivant permet de simplifier les contours des arrondissements de la ville de Sherbrooke avec des tolérances de 250, 500, 1000 et 2000 mètres. Plus la valeur de la tolérance est élevée, plus les contours sont simplifiés (figure 1.9). Notez que l'algorithme de Douglas-Peucker (Douglas et Peucker 1973) a été implémenté dans la fonction `st_simplify`. Bien qu'intéressant, cet algorithme ne conserve pas les frontières entre les polygones.

```
## Simplification des contours avec différentes distances de tolérance
Arrond.simplify250m <- st_simplify(Arrondissements,
```

```
                                  preserveTopology = TRUE,
                                  dTolerance = 250)
Arrond.simplify500m <- st_simplify(Arrondissements,
                                  preserveTopology = TRUE,
                                  dTolerance = 500)
Arrond.simplify1000m <- st_simplify(Arrondissements,
                                   preserveTopology = TRUE,
                                   dTolerance = 1000)
Arrond.simplify2000m <- st_simplify(Arrondissements,
                                   preserveTopology = TRUE,
                                   dTolerance = 2000)
```

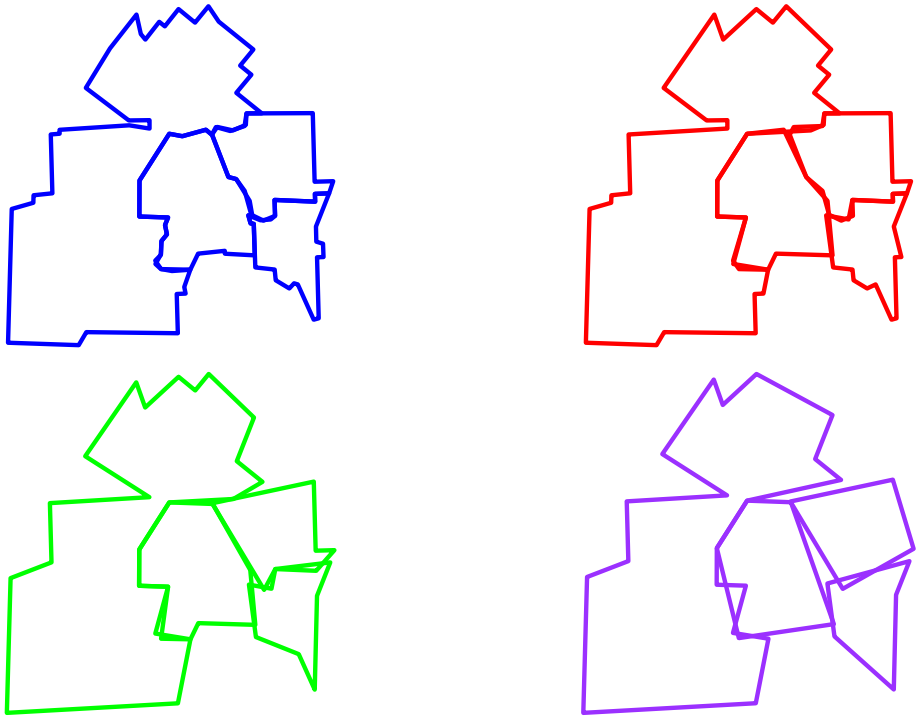


FIGURE 1.9 – Simplification des contours de géométries

Pour remédier au problème des frontières non conservées, utilisez l'algorithme de Visvalingam et Whyatt (1993) avec la fonction `ms_simplify` du *package* `rmapshaper` (figure 1.10), tel qu'illustré dans le code ci-dessous. À titre de rappel, pour l'installer et le charger sur votre ordinateur, tapez dans la console : `install.packages("rmapshaper")` et `library("rmapshaper")`. Le paramètre `keep` permet de définir la proportion de points à retenir : plus sa valeur est faible, plus la simplification est importante.

1.2.2.6 Enveloppe convexe (convex hull)

Le code ci-dessous permet de créer l'enveloppe convexe pour des points (figure 1.11). Notez que cette fonction peut également s'appliquer à des lignes et à des polygones. Elle correspond aux outils `Enveloppe convexe` de QGIS et `Feature to Point (avec l'option Convex hull)` d'ArcGIS Pro.

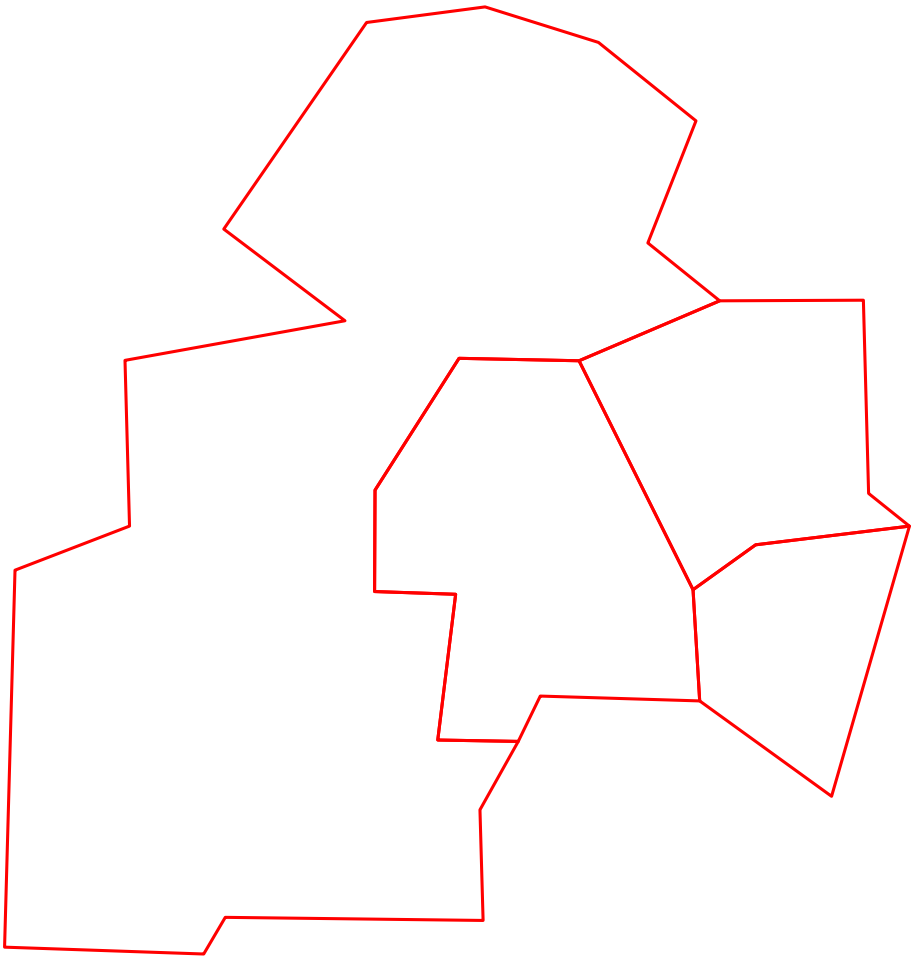


FIGURE 1.10 – Simplification des contours avec l'algorithme de Visvalingam–Whyatt

```
## Enveloppe convexe autour des points GPS
PointsGPS.Convexhull <- st_convex_hull(st_union(PointsGPS))
```

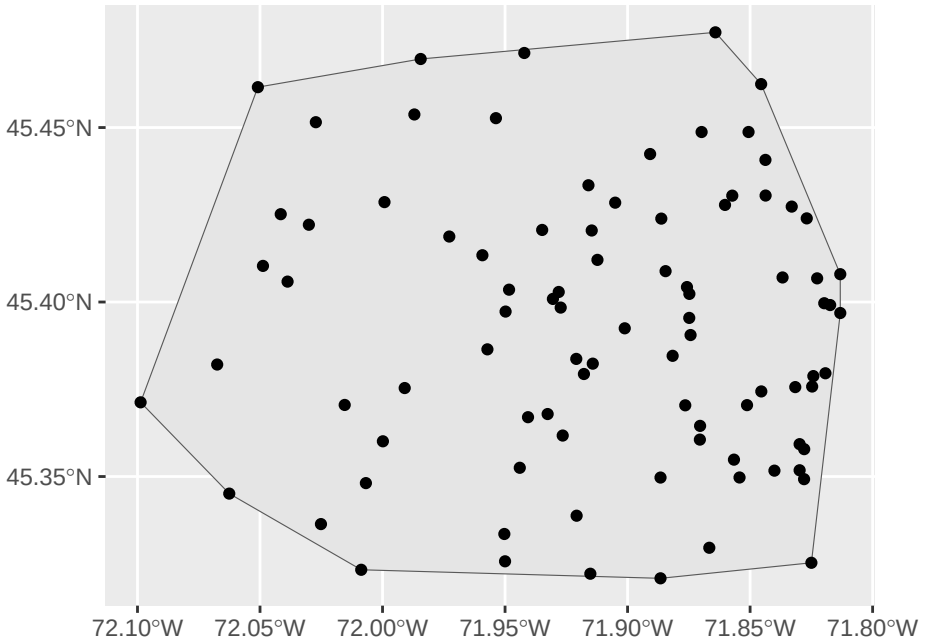


Figure 1.11 – Enveloppe convexe autour de points

1.2.2.7 Enveloppe concave (concave hull)

Une extension possible du polygone convexe est le polygone concave qui a une superficie plus réduite. Il n'existe pas de fonction dans `sf` qui l'implémente. Il faut donc installer un *package* supplémentaire, soit `concaveman`.

```
library(concaveman)
## Convex hull autour des points GPS
PointsGPS.Concavhull <- concaveman(PointsGPS)
```

Notez que comparativement au polygone convexe (figure 1.12), le polygone concave peut auvoir plus d'une solution possible (lire l'article suivant). Plus spécifiquement, il faut choisir son degré de concavité. Dans la fonction `concaveman::concaveman`, le paramètre `concavity` prend une valeur numérique, qui, si elle tend vers l'infini, produit un polygone convexe (figure 1.13).

```
library(ggpubr)
# test avec plusieurs valeur de concavité
concav_values <- c(1,1.5,3,8)
plots <- lapply(concav_values, function(i){
  Concavhull <- concaveman(PointsGPS, concavity = i)
  this_plot <- ggplot()+
    geom_sf(data = Concavhull)+
```

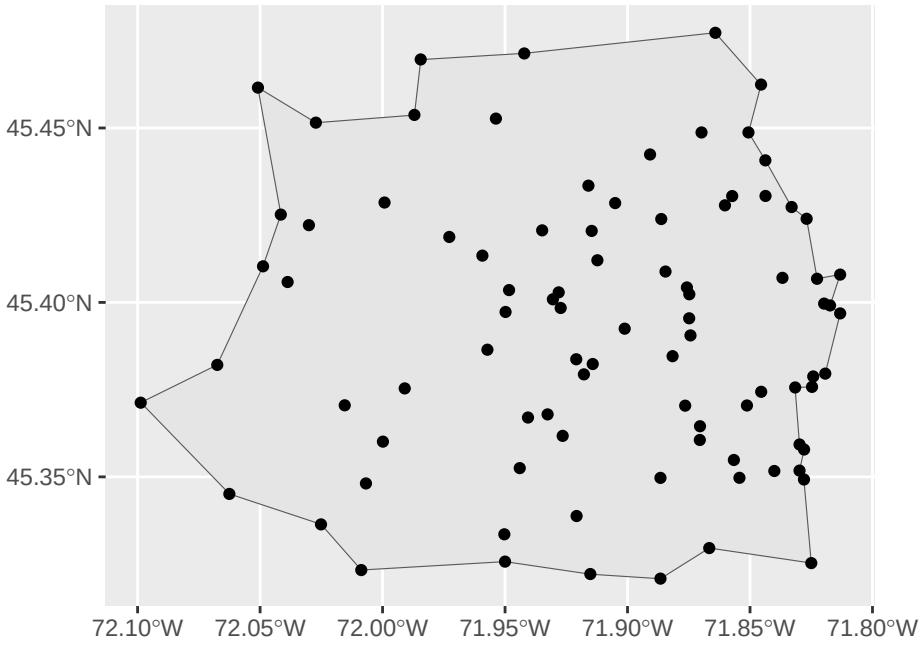


FIGURE 1.12 – Enveloppe concave autour de points

```
    geom_sf(data = PointsGPS)+
    labs(subtitle = paste0("Concavité : ",i))+
      theme(axis.text.x = element_blank(),
            axis.text.y = element_blank(),
            axis.ticks = element_blank())
  return(this_plot)
})
ggarrange(plotlist = plots)
```

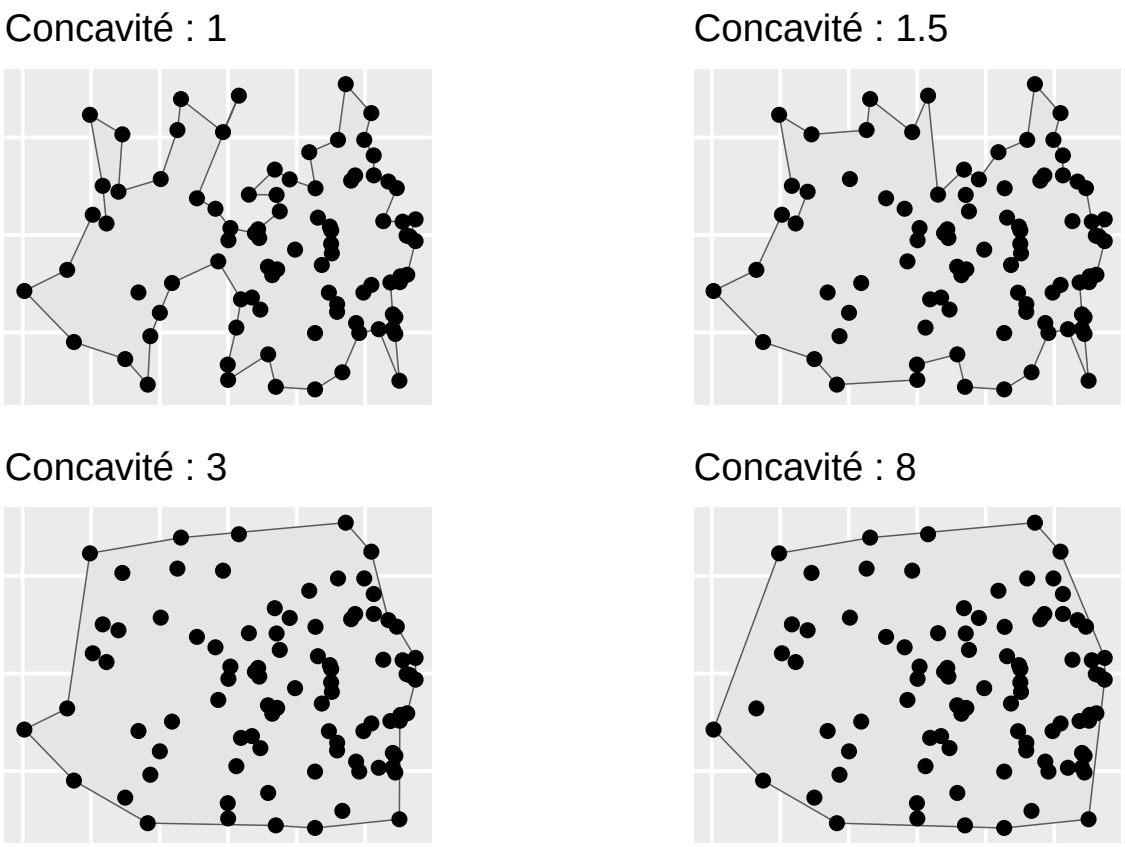


Figure 1.13 – Enveloppe concave autour de points

Dans QGIS, il existe plusieurs *plugins* permettant de générer des enveloppes concaves, ainsi qu'une fonction installée de base avec GRASS (`v.concave.hull`).

1.2.3 Fonctions d'opérations géométriques entre deux couches

Les opérations entre deux couches sont bien connues et largement utilisées dans les SIG. Bien entendu, plusieurs fonctions de ce type sont disponibles dans `sf` et renvoient une nouvelle couche géographique `sf` :

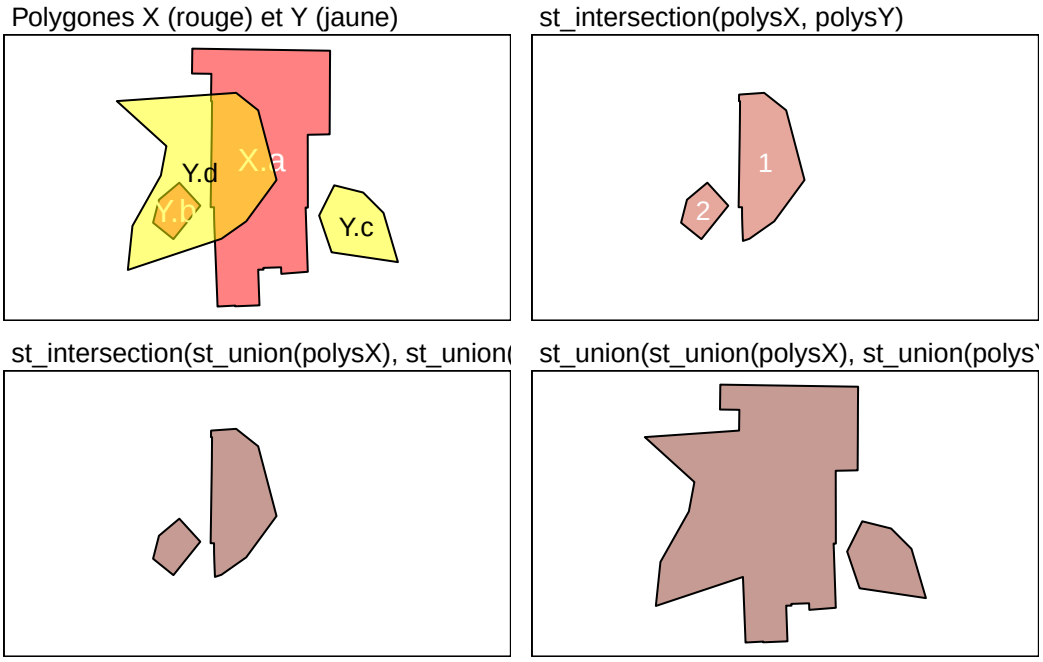
- `st_intersection(x, y)` génère l'intersection entre les géométries de deux couches. À ne pas confondre avec la fonction `st_intersects(x, y)` qui permet de construire une requête spatiale.
- `st_union(x, y)` génère l'union entre les géométries de deux couches.
- `st_difference(x, y)` crée une géométrie à partir de `x` qui n'est pas en intersection avec `y`.
- `st_sym_difference(x, y)` crée une géométrie représentant les portions des géométries `x` et `y` qui ne s'intersectent pas.
- `st_crop(x, y, xmin, ymin, xmax, ymax)` extrait les géométries de `x` comprises dans un rectangle.

En guise de comparaison, toutes ces fonctions sont disponibles dans la boîte à outils de traitement de QGIS (dans le groupe `recouvrement de vecteur`) et les outils de la catégorie `Overlay du Geoprocessing` d'ArcGIS Pro. Le code ci-dessous illustre comment réaliser des intersections et des unions entre deux couches polygonales.

```
## Importation des deux couches
polysX <- st_read("data/chap01/shp/PolyX.shp", quiet = TRUE)
polysY <- st_read("data/chap01/shp/PolyY.shp", quiet = TRUE)
## Intersection des deux couches
## Les géométries récupèrent les attributs des deux couches
Inter.XY <- st_intersection(polysX, polysY)
head(Inter.XY)
```

```
Simple feature collection with 2 features and 2 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: -8006904 ymin: 5684822 xmax: -8006602 ymax: 5685184
Projected CRS: WGS 84 / Pseudo-Mercator
  X_id Y_id                       geometry
1  X.a  Y.d POLYGON ((-8006753 5684838,...
2  Y.b  Y.d POLYGON ((-8006788 5684908,...
```

```
## Intersection entre deux couches préalablement fusionnées :
## Le résutat est une seule géométrie
Inter.XYUnion <- st_intersection(st_union(polysX), st_union(polysY))
## Union des deux couches
Union.XY <- st_union(st_union(polysX), st_union(polysY))
```



La fonction `st_intersection` peut aussi être utilisée comme la méthode `clip` dans un SIG (ArcGIS Pro ou QGIS). En guise d'exemple, dans le code ci-dessous, nous extrayons les points GPS localisés sur le territoire de la ville de Sherbrooke (figure 1.14).

```
# Nous nous assurons que les deux couches ont la même projection
PointsGPS <- st_transform(PointsGPS, st_crs(Arrond.Union))
# Extraction des points
PointsGPS.Sherb <- st_intersection(PointsGPS, Arrond.Union)
# Visualisation avant et après
Carte1 <- ggplot()+geom_sf(data = Arrond.Union)+geom_sf(data = PointsGPS)+
          labs(subtitle = "Avant l'intersection")+
          theme(axis.text.x = element_blank(),
                axis.text.y = element_blank(),
                axis.ticks = element_blank())
Carte2 <- ggplot()+geom_sf(data = Arrond.Union)+geom_sf(data = PointsGPS.Sherb)+
          labs(subtitle = "Après l'intersection")+
          theme(axis.text.x = element_blank(),
                axis.text.y = element_blank(),
                axis.ticks = element_blank())
ggarrange(Carte1, Carte2, ncol = 2, nrow = 1)
```

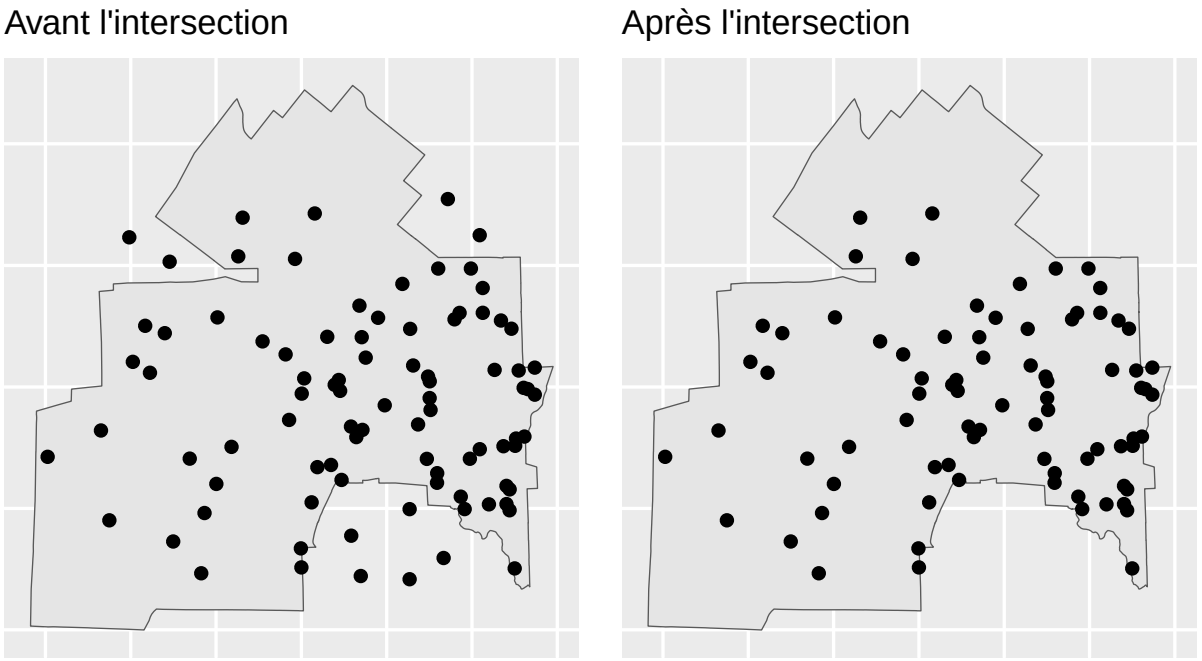


FIGURE 1.14 – Fonction `st_intersection()` équivalente à la méthode clip dans un SIG

Quelques lignes de code suffisent pour générer les différences de superposition entre les géométries de couches géographiques (figure 1.15).

```
## Différences entre deux couches
Diff.XY <- st_difference(st_union(polysX), st_union(polysY))
Diff.YX <- st_difference(st_union(polysY), st_union(polysX))
Diff.symXY <- st_sym_difference(st_union(polysY), st_union(polysX))
```

```
tmap_mode("plot")
MapInterU <- tm_shape(Zone)+tm_fill(col = NA, alpha = 0, border.col = NA, lwd = 0)+
tm_shape(Inter.XYUnion)+tm_polygons(col = "tomato4", alpha = .5,
                            border.col ="black", lwd = 1)

MapDiffXY <- tm_shape(Zone)+tm_fill(col = NA, alpha = 0, border.col = NA, lwd = 0)+
             tm_shape(Diff.XY)+
             tm_polygons(col = "red", alpha = .5,border.col ="black", lwd = 1)+
             tm_layout(main.title = "st_difference(st_union(polysX), st_union(polysY))",
                       main.title.size = .8)

MapDiffYX <- tm_shape(Zone)+tm_fill(col = NA, alpha = 0, border.col = NA, lwd = 0)+
             tm_shape(Diff.YX)+
             tm_polygons(col = "yellow", alpha = .5,border.col ="black", lwd = 1)+
             tm_layout(main.title = "st_difference(st_union(polysY), st_union(polysX))",
                       main.title.size = .8)

MapSymDiffYX <- tm_shape(Zone)+tm_fill(col = NA, alpha = 0, border.col = NA, lwd = 0)+
             tm_shape(Diff.symXY)+
             tm_polygons(col = "green", alpha = .5,border.col ="black", lwd = 1)+
             tm_layout(main.title = "st_sym_difference(st_union(polysY), st_union(polysX))",
                       main.title.size = .8)

tmap_arrange(MapZone, MapDiffXY, MapDiffYX, MapSymDiffYX, ncol=2, nrow=2)
```

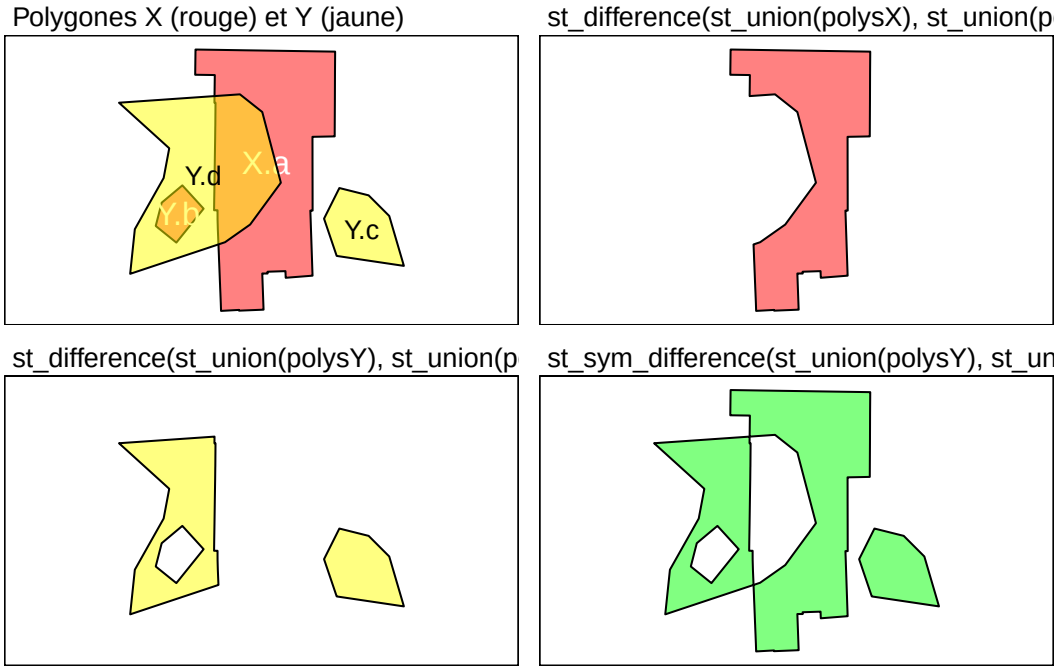


FIGURE 1.15 – Différences de superposition entre des géométries de différentes couches

1.2.4 Fonctions de mesures géométriques et de récupération des coordonnées géographiques

Les principales fonctions de mesures géométriques et de coordonnées géographiques sont :

- `st_area(x)` calcule la superficie des polygones ou des multipolygones d'une couche `sf` .
- `st_length(x)` calcule la longueur des lignes ou des polylignes d'une couche `sf` .
- `st_distance(x, y)` calcule la distance 2D entre deux objets `sf`, exprimée dans le système de coordonnées de référence.
- `st_coordinates(x)` renvoie les coordonnées géographiques de géométries.

Ci-dessous, nous affichons les superficies des quatre arrondissements, puis nous enregistrons les superficies en m^2 et en km^2 dans deux nouveaux champs dénommés `SupM2` et `SupKm2`.

```
## Superficie des polygones des arrondissements
st_area(Arrondissements)
```

```
Units: [m^2]
[1] 477791738 119343215  58289370  87034244
```

```
## Ajout de champs de superficie dans la table attributaire
Arrondissements$SupM2 <- as.numeric(st_area(st_transform(Arrondissements, crs = 2949)))
Arrondissements$SupKm2 <- as.numeric(st_area(st_transform(Arrondissements, crs = 2949)))/1000000
head(Arrondissements, n=2)
```

```
Simple feature collection with 2 features and 4 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: -8027109 ymin: 5668860 xmax: -8000502 ymax: 5704391
Projected CRS: WGS 84 / Pseudo-Mercator
  NUMERO                                                        NOM
1      1 Arrondissement de Brompton-Rock Forest-Saint-Élie-Deauville
2      4                                  Arrondissement des Nations
                        geometry     SupM2    SupKm2
1 POLYGON ((-8005013 5702777,... 235580454 235.58045
2 POLYGON ((-8005680 5690860,...  58861606  58.86161
```

De manière très semblable, calculons la longueur de géométries étant des lignes ou des multilignes.

```
## Longueurs en mètres
PistesCyclables$longMetre <- as.numeric(st_length(st_transform(PistesCyclables, crs = 2949)))
PistesCyclables$longKm <- as.numeric(st_length(st_transform(PistesCyclables, crs = 2949)))/10000
head(PistesCyclables, n=2)
```

```
Simple feature collection with 2 features and 5 fields
Geometry type: MULTILINESTRING
```

```
Dimension:     XY
Bounding box:  xmin: -8010969 ymin: 5666202 xmax: -7997972 ymax: 5697954
Projected CRS: WGS 84 / Pseudo-Mercator
                      NOM OBJECTID SHAPE__Len                       geometry
1     Axe de la Massawippi        1   13944.09 MULTILINESTRING ((-8010969 ...
2 Axe de la Saint-François        2   19394.28 MULTILINESTRING ((-8001909 ...
  longMetre    longKm
1  9807.769 0.9807769
2 13602.324 1.3602324
```

Pour calculer la longueur d'un périmètre, il faut préalablement récupérer son contour avec la méthode `st_boundary`, puis calculer la longueur avec `st_length`.

```
## Conversion des polygones en lignes
Arrond.Contour <- st_boundary(Arrondissements)
## Calcul de la longueur et enregistrement dans deux nouveaux champs
Arrondissements$PerimetreMetre <- as.numeric(st_length(Arrond.Contour))
Arrondissements$PerimetreKm <- as.numeric(st_length(Arrond.Contour)) / 1000
head(Arrondissements)
```

```
Simple feature collection with 4 features and 6 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: -8027109 ymin: 5668860 xmax: -7993013 ymax: 5704391
Projected CRS: WGS 84 / Pseudo-Mercator
  NUMERO                                                        NOM
1      1 Arrondissement de Brompton-Rock Forest-Saint-Élie-Deauville
2      4                                  Arrondissement des Nations
3      3                               Arrondissement de Lennoxville
4      2                                Arrondissement de Fleurimont
                        geometry     SupM2    SupKm2 PerimetreMetre PerimetreKm
1 POLYGON ((-8005013 5702777,... 235580454 235.58045      143771.63   143.77163
2 POLYGON ((-8005680 5690860,...  58861606  58.86161       50476.65    50.47665
3 POLYGON ((-7993443 5684778,...  28776861  28.77686       43531.03    43.53103
4 POLYGON ((-7999483 5693167,...  42882506  42.88251       44172.25    44.17225
```

Calculons désormais la distance 2D (euclidienne) entre les centres des arrondissements. Nous utilisons donc la fonction `st_distance(x)`, puisque nous avons une seule couche (`x = Arrond.pointpoly`).

```
## Longueurs en mètres
st_distance(Arrond.pointpoly)
```

```
Units: [m]
         1         2         3         4
1     0.00 10458.989 21787.479 18047.846
```

```
2 10458.99     0.000 11555.203  8627.962
3 21787.48 11555.203     0.000  9622.735
4 18047.85  8627.962  9622.735     0.000
```

Admettons que nous souhaitons calculer la distance entre les centres des quatre arrondissements et l'hôtel de ville de Sherbrooke dont les coordonnées en degrés (WGS84, EPSG : 4326) sont les suivantes : -71.89306, 45.40417. Nous utilisons alors la fonction `st_distance(x, y)` dans laquelle les paramètres `x` et `y` sont les arrondissements et l'hôtel de ville. Quelques lignes de code suffisent à créer une couche pour l'hôtel de ville, à calculer les distances et à les stocker dans un nouveau champ attributaire de la couche arrondissement.

```
## Création d'un objet sf pour l'hôtel de ville
HotelVille <- data.frame(ID = 1,
                         Nom = "Hôtel de ville",
                         lon = -71.89306,
                         lat = 45.40417)
HotelVille <- st_as_sf(HotelVille, coords = c("lon","lat"), crs = 4326)
head(HotelVille)
```

```
Simple feature collection with 1 feature and 2 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -71.89306 ymin: 45.40417 xmax: -71.89306 ymax: 45.40417
Geodetic CRS:  WGS 84
  ID            Nom                   geometry
1  1 Hôtel de ville POINT (-71.89306 45.40417)
```

```
## Nous nous assurons que les deux couches ont la même projection
HotelVille <- st_transform(HotelVille, st_crs(Arrond.pointpoly))
## Calcul des distances
Arrondissements$DistHVMetre <- as.numeric(st_distance(Arrond.pointpoly,HotelVille))
Arrondissements$DistHVKm <- as.numeric(st_distance(Arrond.pointpoly,
                                                   HotelVille)) / 1000
head(Arrondissements)
```

```
Simple feature collection with 4 features and 8 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: -8027109 ymin: 5668860 xmax: -7993013 ymax: 5704391
Projected CRS: WGS 84 / Pseudo-Mercator
  NUMERO                                                        NOM
1      1 Arrondissement de Brompton-Rock Forest-Saint-Élie-Deauville
2      4                                  Arrondissement des Nations
3      3                               Arrondissement de Lennoxville
4      2                                Arrondissement de Fleurimont
                       geometry     SupM2    SupKm2 PerimetreMetre PerimetreKm
```

```
1 POLYGON ((-8005013 5702777,... 235580454 235.58045      143771.63   143.77163
2 POLYGON ((-8005680 5690860,...  58861606  58.86161       50476.65    50.47665
3 POLYGON ((-7993443 5684778,...  28776861  28.77686       43531.03    43.53103
4 POLYGON ((-7999483 5693167,...  42882506  42.88251       44172.25    44.17225
  DistHVMetre  DistHVKm
1   14661.518 14.661518
2    4662.164  4.662164
3    9058.677  9.058677
4    4050.374  4.050374
```

Il est fréquent de vouloir enregistrer les coordonnées géographiques dans des champs attributaires. Dans le code ci-dessous, nous créons deux champs (`x` et `y`) dans lesquels nous enregistrons les coordonnées géographiques des points au centre de la surface de chaque arrondissement. Pour ce faire, nous utilisons la méthode `st_coordinates` .

```
## Coordonnées des centres de la surface des polygones
xy <- st_coordinates(st_point_on_surface(Arrondissements))
head(xy)
```

```
           X       Y
[1,] -8017707 5686628
[2,] -8007570 5684053
[3,] -7997637 5678149
[4,] -7999683 5687552
```

```
## Enregistrement dans la couche Arrondissements. Notez que :
## xy[,1] signale de récupérer toutes les valeurs de la première colonne, soit X
## xy[,2] signale de récupérer toutes les valeurs de la deuxième colonne, soit Y
Arrondissements$X <- xy[,1]
Arrondissements$Y <- xy[,2]
```

1.2.5 Jointures spatiales

En géomatique, il est fréquent de réaliser des jointures spatiales, soit une opération qui consiste à joindre les attributs d'une couche géographique à une autre à partir d'une relation spatiale. Prenons deux exemples construits avec les installations sportives et récréatives (couche `InstallationSport`) et les arrondissements de la ville de Sherbrooke (`Arrondissements`).

Premièrement, pour les installations sportives et récréatives (couche `InstallationSport`), nous souhaitons ajouter dans la table attributaire les champs `NUMERO` et `NOM` issus de la couche des arrondissements de la ville de Sherbrooke (`Arrondissements`). Grâce à ces deux champs, nous pouvons connaître dans quel arrondissement chaque installation sportive est située.

```
## Jointure spatiale avec le paramètre st_intersects
InstallS.join <- st_join(InstallationSport, Arrondissements, join = st_intersects)
## Visualisation des deux premiers enregistrements
head(InstallS.join, n=2)
```

```
Simple feature collection with 2 features and 16 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -8009681 ymin: 5686891 xmax: -8001939 ymax: 5696536
Projected CRS: WGS 84 / Pseudo-Mercator
   TYPE DETAIL                    NOM.x SURFACE ECLAIRAGE OBJECTID NUMERO
1 Aréna   <NA>     Aréna Eugène-Lalonde    <NA>      <NA>        1      2
2 Aréna   <NA> Aréna Philippe-Bergeron    <NA>      <NA>        2      1
                                                      NOM.y     SupM2
1                                Arrondissement de Fleurimont  42882506
2 Arrondissement de Brompton-Rock Forest-Saint-Élie-Deauville 235580454
     SupKm2 PerimetreMetre PerimetreKm DistHVMetre  DistHVKm        X       Y
1  42.88251       44172.25    44.17225    4050.374  4.050374 -7999683 5687552
2 235.58045      143771.63   143.77163   14661.518 14.661518 -8017707 5686628
                  geometry
1 POINT (-8001939 5686891)
2 POINT (-8009681 5696536)
```

```
## Suppression des champs utiles
InstallS.join[c("SupM2", "SupKm2", "PerimetreMetre",
                      "PerimetreKm", "DistHVMetre",  "DistHVKm")] <- list(NULL)
## Modification des noms de champs : NOM.x et NOM.y
names(InstallS.join)[names(InstallS.join) == "NOM.x"] <- "NomInstallation"
names(InstallS.join)[names(InstallS.join) == "NOM.y"] <- "NomArrondissement"
head(InstallS.join, n=2)
```

```
Simple feature collection with 2 features and 10 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -8009681 ymin: 5686891 xmax: -8001939 ymax: 5696536
Projected CRS: WGS 84 / Pseudo-Mercator
   TYPE DETAIL         NomInstallation SURFACE ECLAIRAGE OBJECTID NUMERO
1 Aréna   <NA>     Aréna Eugène-Lalonde    <NA>      <NA>        1      2
2 Aréna   <NA> Aréna Philippe-Bergeron    <NA>      <NA>        2      1
                                        NomArrondissement        X       Y
1                                Arrondissement de Fleurimont -7999683 5687552
2 Arrondissement de Brompton-Rock Forest-Saint-Élie-Deauville -8017707 5686628
                  geometry
1 POINT (-8001939 5686891)
2 POINT (-8009681 5696536)
```

Deuxièmement, une autre jointure classique consiste à dénombrer les points compris dans des polygones, soit une opération SIG communément appelée *POINT-IN-POLYGON*.

```
## Sélection des points dans les polygones des arrondissements
## Notez que la relation spatiale pour la jointure est st_contains
## Nous aurions pu aussi utiliser st_intersects
Arrondissements$NbInstall = lengths(st_contains(Arrondissements, InstallationSport))
head(Arrondissements$NbInstall)
```

```
[1] 125 166  29 116
```

🚀 Aller plus loin

Autres relations spatiales à appliquer lors de la jointure spatiale
Avec le paramètre `join` de la méthode `st_join`, il est possible de spécifier la jointure spatiale avec différentes méthodes : `st_contains_properly`, `st_contains`, `st_covered_by`, `st_covers`, `st_crosses`, `st_disjoint`, `st_equals_exact`, `st_equals`, `st_is_within_distance`, `st_nearest_feature`,`st_overlaps`, `st_touches` et `st_within`.
N'hésitez pas à consulter la documentation de la fonction en tapant`?st_join` dans la console R.

1.2.6 Requêtes spatiales

Dans un logiciel SIG, la sélection d'entités spatiales par localisation est une opération courante, équivalente à *Select By Location* dans ArcGis Pro ou *Sélection par localisation* dans QGIS.

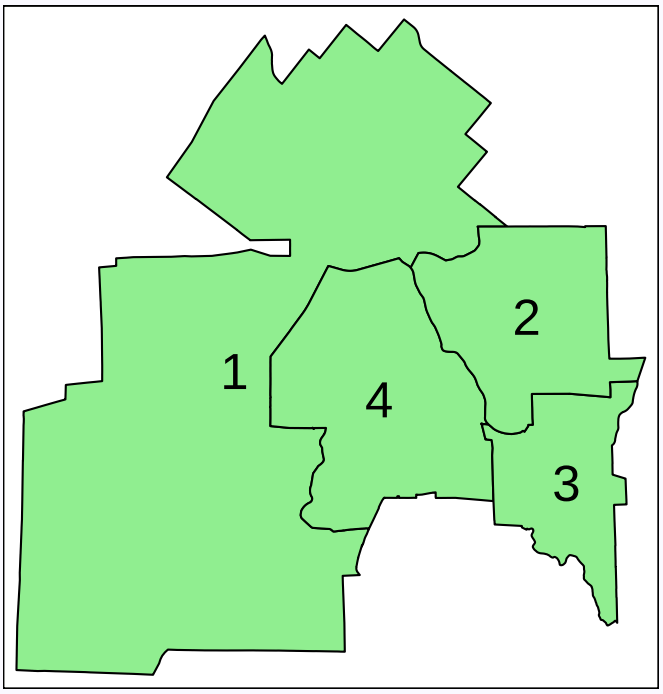
Le *package* `sf` permet de réaliser des requêtes spatiales avec notamment les méthodes suivantes :

- `st_contains(x, y)` renvoie les géométries de `x` qui contiennent celles de `y`. Cette fonction est donc l'inverse de `st_within`.
- `st_disjoint(x, y)` renvoie les géométries de `x` qui ne partagent aucune portion de celles de `y`. Cette fonction est donc l'inverse de `st_intersects(x, y)`.
- `st_equals(x, y)` renvoie les géométries de `x` qui sont identiques à celles de `y`.
- `st_intersects(x, y)` renvoie les géométries de `x` qui partagent au moins une partie de celles de `y`. Elle est donc l'inverse de `st_disjoints(x, y)`.
- `st_nearest_feature(x, y)` renvoie, pour chaque géométrie `x`, la géométrie de `y` qui est la plus proche.
- `st_overlaps(x, y)` cette fonction est très semblable à `st_intersects(x, y)`. Toutefois, les types de géométries de `x` et de `y` doivent être identiques, c'est-à-dire deux couches de lignes ou de couches de polygones. Aussi, une géométrie ne peut pas contenir complètement l'autre comme avec `st_within(x, y)` et `st_contains(x, y)`.
- `st_touches(x, y)` renvoie les géométries de `x` qui sont tangentes à celles de `x` sans qu'elles se chevauchent. Par exemple, deux arrondissements peuvent se toucher, c'est-à-dire qu'ils partagent une frontière commune sans que l'un chevauche l'autre.
- `st_within(x, y)` renvoie les géométries de `x` qui sont comprises intégralement dans celles de `y`. Cette fonction est donc l'inverse de `st_contains(x, y)`.
- `st_within_distance(x, y, dist =)` renvoie les géométries de `x` qui sont situées à une certaine distance euclidienne de celles de `y`.

Astuce

Modification de l'affichage du résultat de la requête spatiale : le paramètre `sparse`
Par défaut, le résultat d'une requête spatiale renvoie une liste d'indices pour les géométries `x` et `y`. Il est aussi possible de renvoyer la matrice complète entre `x` et `y`, avec les valeurs `TRUE` quand la relation spatiale est vérifiée et `FALSE` pour une situation inverse.
Prenons deux exemples pour illustrer le tout.
La figure ci-dessous représente les quatre arrondissements de la ville de Sherbrooke. Notez que les numéros correspondent aux indices des géométries.

```
  NUMERO                                                    NOM
1      1 Arrondissement de Brompton–Rock Forest–Saint–Élie–Deauville
2      4                                 Arrondissement des Nations
3      3                               Arrondissement de Lennoxville
4      2                                Arrondissement de Fleurimont
```



Appliquons une requête spatiale entre les arrondissements avec `st_intersects` et `sparse = TRUE`. Pour chaque arrondissement, nous obtenons une liste des arrondissements qui l'intersectent.

```
st_intersects(Arrondissements, Arrondissements, sparse = TRUE)
```

```
Sparse geometry binary predicate list of length 4, where the predicate
was `intersects'
 1: 1, 2, 4
 2: 1, 2, 3, 4
 3: 2, 3, 4
 4: 1, 2, 3, 4
```

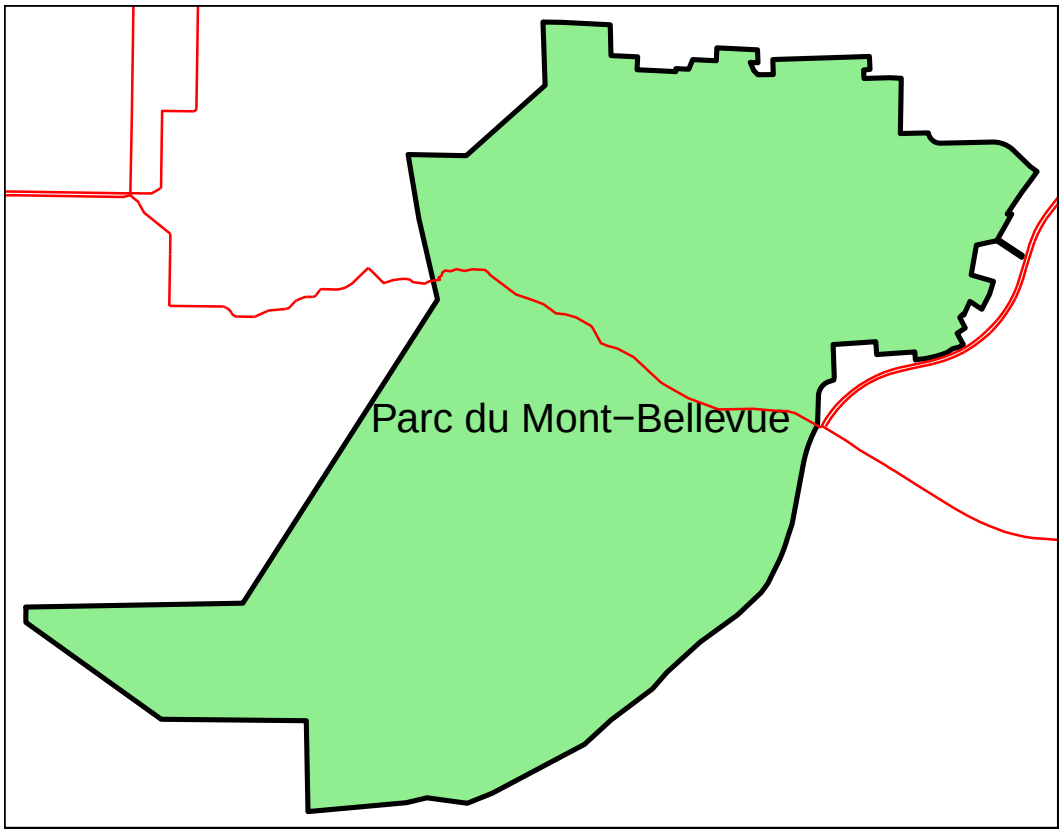
Avec `sparse = FALSE`, nous obtenons une matrice complète de dimension 4 X 4 arrondissements. Nous constatons que l'arrondissement `1` intersecte lui-même (évidemment!) et les arrondissements `2` et `4`, mais il n'intersecte pas le `3`.


```
st_intersects(Arrondissements, Arrondissements, sparse = FALSE)
```

```
      [,1] [,2]  [,3] [,4]
[1,]  TRUE TRUE FALSE TRUE
```

Construisons des requêtes plus complexes comprenant deux couches.

Premièrement, écrivons une requête spatiale pour sélectionner les segments des pistes cyclables qui intersectent le parc du Mont-Bellevue. Pour ce faire, nous utilisons la fonction `st_intersects` avec l'argument `sparse = FALSE` et enregistrons le résultat dans un nouveau champ dénommé `ParcMB.intersect` qui prendra les valeurs `TRUE` ou `FALSE`.




```
## Intersection
RequeteSpatiale <- st_intersects(PistesCyclables, MontBellevue, sparse = FALSE)
head(RequeteSpatiale)
```

```
      [,1]
[1,] FALSE
[2,] FALSE
[3,] FALSE
[4,] FALSE
[5,] FALSE
[6,] FALSE
```

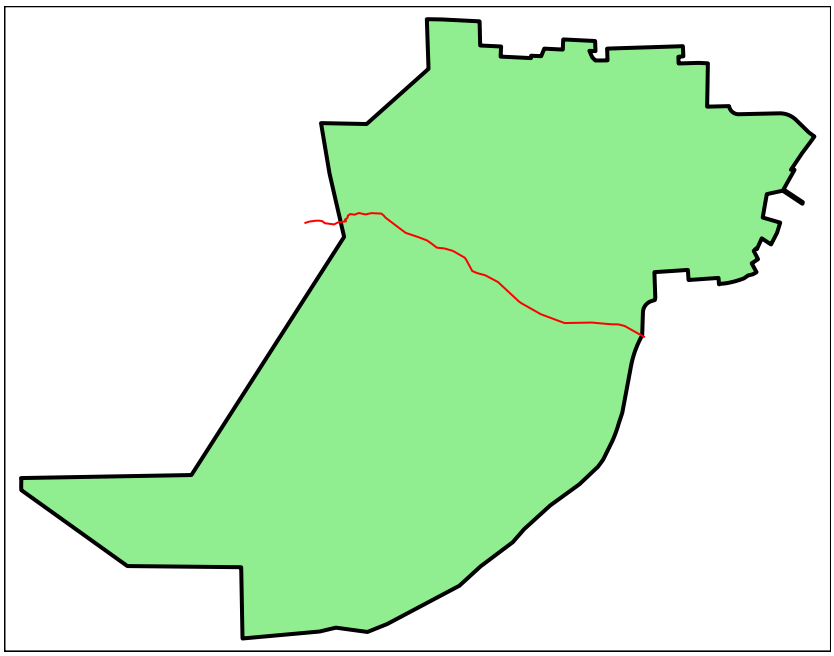
```
## Création d'un nouveau champ
PistesCyclables$ParcMB.intersect <- RequeteSpatiale[, 1]
head(PistesCyclables)
```

```
Simple feature collection with 6 features and 6 fields
Geometry type: MULTILINESTRING
Dimension:     XY
Bounding box:  xmin: -8010969 ymin: 5666202 xmax: -7997216 ymax: 5697954
Projected CRS: WGS 84 / Pseudo-Mercator
                       NOM OBJECTID  SHAPE__Len                       geometry
1     Axe de la Massawippi        1 13944.08678 MULTILINESTRING ((-8010969 ...
2 Axe de la Saint-François        2 19394.27693 MULTILINESTRING ((-8001909 ...
3   Axe du Ruisseau-Dorman        3 16337.23985 MULTILINESTRING ((-7999121 ...
4        Réseau utilitaire        4   467.23254 MULTILINESTRING ((-8000179 ...
5        Réseau utilitaire        5    15.57987 MULTILINESTRING ((-8004036 ...
6        Réseau utilitaire        6   823.83428 MULTILINESTRING ((-8003649 ...
    longMetre      longKm ParcMB.intersect
1  9807.76890 0.980776890            FALSE
2 13602.32404 1.360232404            FALSE
3 11469.13476 1.146913476            FALSE
4   327.46928 0.032746928            FALSE
5    10.95083 0.001095083            FALSE
6   578.59143 0.057859143            FALSE
```

```r
## Nous constatons qu'un seul segment intersecte le parc
table(PistesCyclables$ParcMB.intersect)
```

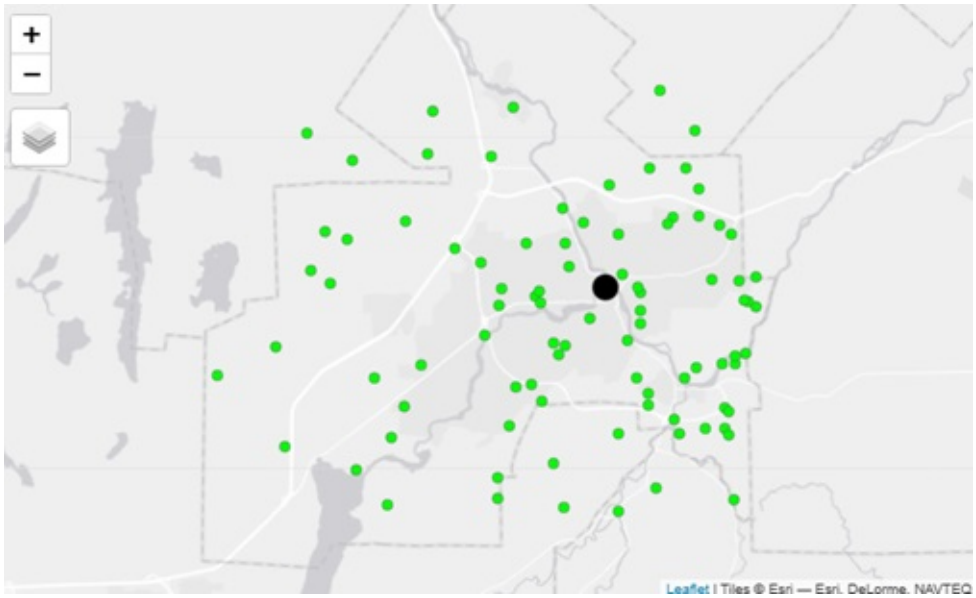
```
FALSE  TRUE
  272     1
```

```r
## Création d'une nouvelle couche pour la sélection
PistesCyclables.Selection <- PistesCyclables[PistesCyclables$ParcMB.intersect== TRUE, ]
## Visualisation
tm_shape(MontBellevue) + tm_fill(col="lightgreen")+ tm_borders(col = "black", lwd=2)+
tm_shape(PistesCyclables.Selection)+tm_lines(col="red", lwd=1)
```



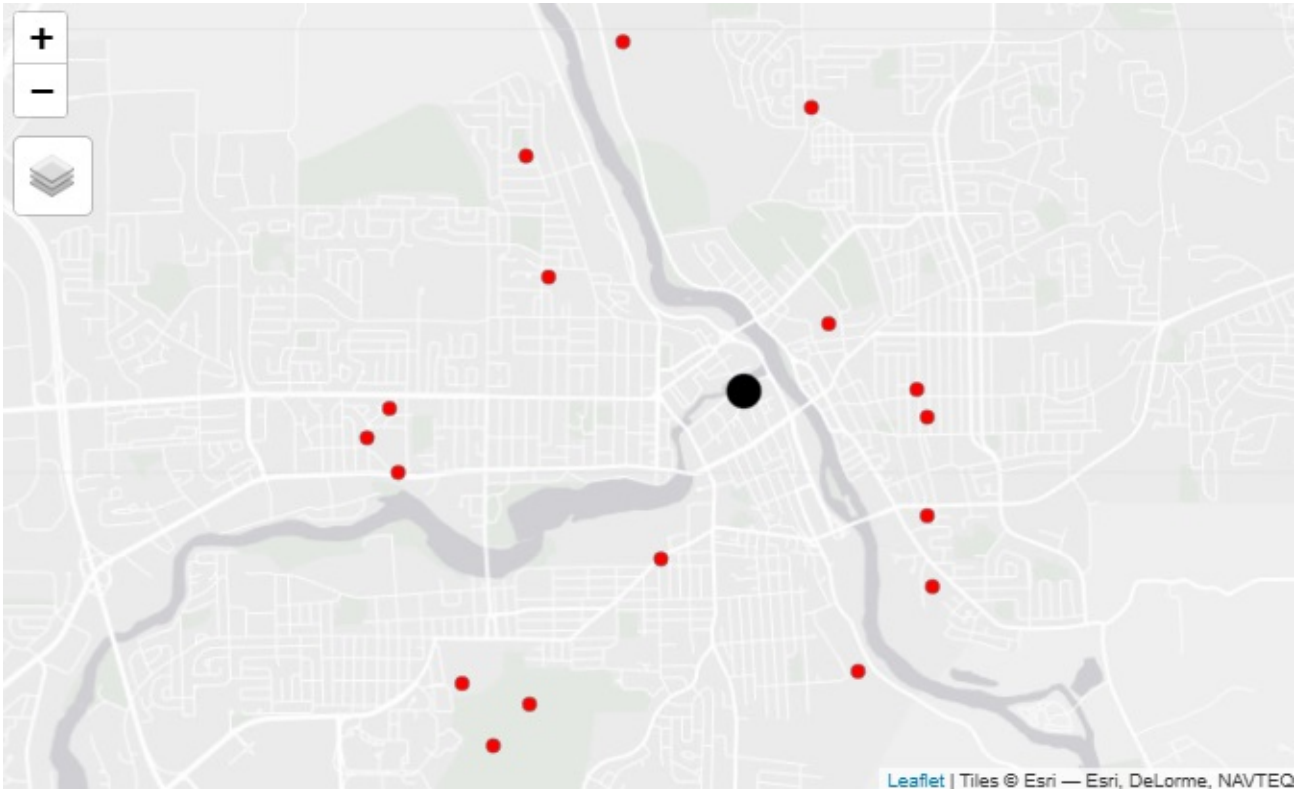
Créons une deuxième requête spatiale pour sélectionner les points GPS situés à moins de cinq kilomètres de l'hôtel de ville de Sherbrooke avec la méthode `st_is_within_distance`.

```
tmap_mode("view")
tm_shape(PointsGPS) + tm_dots(col="green", size = .05) +
tm_shape(HotelVille)+tm_dots(col="black", size = .25)
```




```
## Requête spatiale
RequeteSpatiale <- st_is_within_distance(PointsGPS, HotelVille,
                                         5000, sparse = FALSE)
```

```
## Ajout d'un champ pour la requête
PointsGPS$HotelVille2km <- RequeteSpatiale[, 1]
## Nous constatons que 17 points GPS sont à moins de 5 km
table(PointsGPS$HotelVille2km)
## Création d'une nouvelle couche pour la sélection
PointsGPS.selection <- PointsGPS[PointsGPS$HotelVille2km == TRUE, ]
## Visualisation
tm_shape(PointsGPS.selection) + tm_dots(col="red", size = .05)+
tm_shape(HotelVille)+tm_dots(col="black", size = .25)
```



Finalement, avec la méthode `st_within`, nous constatons que seuls deux points GPS sont situés dans le parc du Mont-Bellevue.

```
## Requête spatiale
RequeteSpatiale <- st_within(st_transform(PointsGPS, st_crs(MontBellevue)),
                             MontBellevue, sparse = FALSE)
table(RequeteSpatiale[,1])
```

```
FALSE  TRUE
   87     2
```

1.2.7 Manipulation des données attributaires

Dans cette section, nous verrons comment importer une table attributaire, puis la joindre à une couche géographique, ajouter et calculer de nouveaux champs et réaliser des requêtes attributaires.

1.2.7.1 Importation d'une table attributaire

Objectif

Joindre les attributs d'une table externe à une couche vectorielle `sf`
Dans un SIG, joindre une table à une couche géographique vectorielle est une opération courante. Par exemple, il est fréquent de joindre des données socioéconomiques issues d'un recensement à une couche géographique (divisions de recensement, subdivisions de recensement, secteurs de recensement, aires de diffusion, etc.).
Pour ce faire, vous devez importer les données dans un *DataFrame* de R. Ces données peuvent être stockées dans différents formats de fichiers (texte délimité par des virgules (extension *csv*), *dBase* (*dbf*), Excel (*xlsx*)) ou dans des fichiers provenant de logiciels statistiques commerciaux comme Stata, SAS et SPSS (*dta*, *sas7bdat*, *sav*).
Dans cette section, nous voyons seulement l'importation de fichiers texte délimités par des virgules, de fichiers Excel et *dBase*. Concernant ce dernier type de fichier, notez que la table attributaire d'une couche Esri Shapefile est stockée dans un fichier *dBase*! Il peut être intéressant d'importer la table sans les géométries.
Pour une description détaillée de l'importation d'autres fichiers (entre autres Stata, SAS et SPSS), consultez la section intitulée *Manipulation d'un DataFrame* (Apparicio et Gelb 2022).

Dans le code ci-dessous, nous voyons comment importer trois types de fichiers :

- `read.csv(file)` pour importer un fichier délimité par des virgules. Cette fonction est de base avec R, ce qui signifie qu'elle ne nécessite pas l'installation d'un *package*.
- `read.dbf(file)` pour importer un fichier *dBase*. Cette fonction est rattachée au *package* `foreign` que vous devez installer si ce n'est pas déjà fait (commande `install.packages("foreign")`) et le charger (commande `library("foreign")`).
- `read.xlsx(file)` pour importer un fichier Excel. Cette fonction est rattachée au *package* `xlsx` que vous devez installer si ce n'est pas déjà fait (commande `install.packages("xlsx")`) et le charger (commande `library("xlsx")`).

```
library("xlsx")      # package pour importer des fichiers Excel
library("foreign")   # package pour importer des fichiers dBase
## Importation du fichier csv
t1 <- Sys.time()
dfCSV <- read.csv(file = "data/chap01/tables/SRQC2021.csv",
                  header = TRUE,
                  dec = ".",    # séparateur de décimales qui peut être remplacé par ,
                  sep = ","     # séparateur des champs qui peut être remplacé par ;
                  )
t2 <- Sys.time()
cat("temps de traitement (CSV) : ",
    as.numeric(difftime(t2,t1,units="secs")),
    " secondes")
```

```
temps de traitement (CSV) :  0.0197618  secondes
```

```
## Importation d'un fichier Excel avec le nom de fichier et de la feuille Excel
## sheetIndex = 1 signale l'importation de la première feuille Excel
t1 <- Sys.time()
dfExcel <- read.xlsx(file = "data/chap01/tables/ADSRQC2021.xlsx",
                     sheetIndex = 2)
t2 <- Sys.time()
cat("temps de traitement (Excel) : ",
    as.numeric(difftime(t2,t1,units="secs")),
    " secondes")
```

```
temps de traitement (Excel) :  9.996  secondes
```

```
## Importation du fichier dBase
t1 <- Sys.time()
dfDbf <- read.dbf(file = "data/chap01/tables/ADQC2021.dbf")
t2 <- Sys.time()
cat("temps de traitement (dBase) : ",
    as.numeric(difftime(t2,t1,units="secs")),
    " secondes")
```

```
temps de traitement (dBase) :  0.159807  secondes
```

Astuce

Exportation du fichier Excel dans un fichier texte
Le temps nécessaire pour importer un fichier Excel est bien plus long que pour des fichiers texte et *dBase*! Par conséquent, si vous travaillez avec Excel, il est vivement conseillé de l'exporter vers un fichier texte (dans Excel, `Fichier/Enregistrer sous/type de fichier CSV`).

Quelques lignes suffisent pour explorer la structure des données importées avec les fonctions `nrow`, `ncol`, `colnames` (respectivement le nombre de lignes, le nombre de colonnes et les noms des colonnes du `dataframe`).

```
## Nombre de lignes et de colonnes
nrow(dfCSV)
```

```
[1] 2245
```

```
ncol(dfCSV)
```

```
[1] 40
```

```
cat("le DataFrame dfCSV a", nrow(dfCSV), "lignes (observations)",
    'et', ncol(dfCSV), "colonnes\n")
```

```
le DataFrame dfCSV a 2245 lignes (observations) et 40 colonnes
```

```
## Noms des champs
colnames(dfCSV)
```

```
 [1] "SRIDU"                     "PopTotAge"
 [3] "Pop0_14"                   "Pop15_64"
 [5] "Pop65plus"                 "TotalLog"
 [7] "MaisonIndiv"               "MaisonJumulee"
 [9] "MaisonRangee"              "AppartDuplex"
[11] "AppartMoins5E"             "Appart5EtPlus"
[13] "AutreMaisonIndivAttenante" "LogementMobile"
[15] "TotalMenag"                "Menage1pers"
[17] "Menage2pers"               "Menage3pers"
[19] "Menage4pers"               "Menage5pPlus"
[21] "RevMedMenage"              "PopTotMFRApI"
[23] "PopTotMFR"                 "PopTotMFRPct"
[25] "TotalMenag2"               "Proprietaire"
[27] "Locataire"                 "TotalLog2"
[29] "Log1960ouAv"               "Log1961_80"
[31] "Log1981_90"                "Log1991_00"
[33] "Log2001_05"                "Log2006_10"
[35] "Log2011_15"                "Log2016_21"
[37] "ValeurMedLog"              "ValeurMoyLog"
[39] "LoyerMedian"               "LoyerMoyen"
```

```
## Affichage des deux premières observations
head(dfCSV, n=2)
```

```
                                                  SRIDU PopTotAge Pop0_14
1 4470001.01 (SR), Drummondville (RMR) (4470001.01) (00000)      5080     810
2 4470001.02 (SR), Drummondville (RMR) (4470001.02) (00000)      3400     175
  Pop15_64 Pop65plus TotalLog MaisonIndiv MaisonJumulee MaisonRangee
1     3285       985     2280        1290           185          210
2     1305      1920     1815         155            75           85
  AppartDuplex AppartMoins5E Appart5EtPlus AutreMaisonIndivAttenante
1           70           415             0                        15
2           15          1485             0                         5
  LogementMobile TotalMenag Menage1pers Menage2pers Menage3pers Menage4pers
1            100       2285         705         895         325         225
2              0       1810         985         670         100          45
```

```
  Menage5pPlus RevMedMenage PopTotMFRApI PopTotMFR PopTotMFRPct TotalMenag2
1          130        69000         5085       520         10.2        2295
2           15        47600         2885       545         18.9        1820
  Proprietaire Locataire TotalLog2 Log1960ouAv Log1961_80 Log1981_90 Log1991_00
1         1445       850      2295         115        590        535        485
2          485      1335      1820          50        310        375        405
  Log2001_05 Log2006_10 Log2011_15 Log2016_21 ValeurMedLog ValeurMoyLog
1        155         70         75        265       250000       250200
2        215        200        120        140       250000       305000
  LoyerMedian LoyerMoyen
1         695        742
2         740        737
```

1.2.7.2 Jointure attributaire avec la couche géographique `sf`

Les données importées dans la table attributive proviennent du recensement de Statistique Canada de 2021 et sont ancrées au niveau des secteurs de recensement (SR) des régions métropolitaines de recensement (RMR) et des agglomérations de recensement (AR) du Québec. Pour les SR de la RMR de Sherbrooke, les données de la couche géométrique sont importées à partir d'un fichier shapefile. Aussi, nous constatons que les deux sources de données ont un champ commun `SRIDU`, soit l'identifiant unique des SR, mais que l'information y est présentée différemment :

- Dans la couche géographique `SR.RMRSherb` (objet *sf*), nous avons une observation avec la valeur `4470001.01`, soit un champ avec dix caractères.
- Dans la table attributaire `dfCSV` (*DataFrame*), nous avons une observation avec la valeur `4470001.01 (SR), Drummondville (RMR) (4470001.01) (00000)`.

Par conséquent, avant d'appliquer une jointure, nous modifions le champ `SRIDU` de ce *DataFrame* afin qu'il ait aussi dix caractères avec la ligne de code suivante : `dfCSV$SRIDU <- substr(dfCSV$SRIDU, 1, 10)`. De la sorte, nous récupérons uniquement les dix premiers caractères.

Finalement, la jointure est réalisée avec la fonction `merge` avec laquelle nous spécifions le résultat de la jointure (`SR.RMRSherbDonnees`), la couche géographique (`SR.RMRSherb`), la table attributaire (`dfCSV`) et le champ commun aux deux avec l'option (`by ="SRIDU"`) :

`SR.RMRSherbDonnees <- merge(SR.RMRSherb, dfCSV, by = "SRIDU").`

```
## Importation des SR de la RMR de Sherbrooke
SR.RMRSherb <- st_read(dsn = "data/chap01/gpkg/Recen2021Sherbrooke.gpkg",
                       layer = "SherbSR", quiet=TRUE)
## Visualisation des premiers enregistrements
head(as.data.frame(SR.RMRSherb), n=2)
```

```
                IDUGD      SRIDU   SRNOM SUPTERRE PRIDU SRpop_2021 SRtlog_2021
1 2021S05074330001.00 4330001.00 0001.00   3.1882    24       5637        2918
2 2021S05074330002.00 4330002.00 0002.00   0.8727    24       1868        1169
  SRrhlog_2021 RMRcode   HabKm2                          geom
1         2756     433 1768.082 MULTIPOLYGON (((7764998 127...
2         1063     433 2140.484 MULTIPOLYGON (((7763361 127...
```

```
head(dfCSV, n=2)
```

```
                                                    SRIDU PopTotAge Pop0_14
1 4470001.01 (SR), Drummondville (RMR) (4470001.01) (00000)      5080     810
2 4470001.02 (SR), Drummondville (RMR) (4470001.02) (00000)      3400     175
  Pop15_64 Pop65plus TotalLog MaisonIndiv MaisonJumulee MaisonRangee
1     3285       985     2280        1290           185          210
2     1305      1920     1815         155            75           85
  AppartDuplex AppartMoins5E Appart5EtPlus AutreMaisonIndivAttenante
1           70           415             0                        15
2           15          1485             0                         5
  LogementMobile TotalMenag Menage1pers Menage2pers Menage3pers Menage4pers
1            100       2285         705         895         325         225
2              0       1810         985         670         100          45
  Menage5pPlus RevMedMenage PopTotMFRApI PopTotMFR PopTotMFRPct TotalMenag2
1          130        69000         5085       520         10.2        2295
2           15        47600         2885       545         18.9        1820
  Proprietaire Locataire TotalLog2 Log1960ouAv Log1961_80 Log1981_90 Log1991_00
1         1445       850      2295         115        590        535        485
2          485      1335      1820          50        310        375        405
  Log2001_05 Log2006_10 Log2011_15 Log2016_21 ValeurMedLog ValeurMoyLog
1        155         70         75        265       250000       250200
2        215        200        120        140       250000       305000
  LoyerMedian LoyerMoyen
1         695        742
2         740        737
```

```
## Modification du champ SRIDU du DataFrame dfCSV
dfCSV$SRIDU <- substr(dfCSV$SRIDU, 1, 10)
## Jointure attributaire avec la fonction merge
SR.RMRSherbDonnees <- merge(SR.RMRSherb,
                            dfCSV,
                            by = "SRIDU")
```

Astuce

Jointure avec deux champs ayant des noms différents
En résumé, une jointure attributaire s'écrit :
`NouvelObjetSf <- merge(X, Y, by = "Nom du champ commun pour la jointure")`
avec `X` et `Y` étant respectivement l'objet `sf` (couche géographique) et la table attributaire à joindre. Si les champs pour la jointure ont des noms différents, il est possible d'écrire :
`NouvelObjetSf <- merge(X, Y, by.x = "Champ X pour la jointure", by.y = "Champ Y pour la jointure")`
Ce type de jointure conserve uniquement les observations qui sont communes à la couche géographique et à la table attributaire. Concrètement, si une couche comprend 100 entités spatiales et la table attributaire uniquement 80 observations, la couche résultante (`NouvelObjetSf`) aura uniquement 80 entités spatiales (bien entendu, quand les valeurs concordent…).
Lorsque vous souhaitez quand même conserver toutes les entités spatiales de la couche géographique de départ, écrivez :
`NouvelObjetSf <- merge(X, Y, by = "Nom du champ commun pour la jointure", all.x = TRUE)`
Dans la nouvelle couche `Sf`, les entités spatiales de `X` qui n'ont pas été appariées avec les observations de la table attributaire `Y` auront des valeurs nulles (`NA`) pour les champs de `X` ajoutés au `NouvelObjetSf`.
Pour obtenir plus d'informations sur les différentes variantes d'une jointure, tapez `?merge` dans la console.

1.2.7.3 Ajout et calcul de champs

Dans la section 1.2.4, nous avons vu comment ajouter des champs relatifs à la géométrie (aire, longueur, distance, coordonnées de centroïdes). Dans un SIG, il est courant de calculer de nouveaux champs à partir de champs existants dans la table attributaire (par exemple, avec les outils *Calculatrice de champ* dans QGIS ou *Calculate Field* dans ArcGIS Pro).

Ce type de traitement est aussi très simple à réaliser dans R. Pour ce faire, nous utilisons des opérateurs mathématiques, relationnels et logiques comme dans n'importe quel logiciel de SIG. En guise d'exemple, nous calculons ci-dessous les pourcentages d'enfants de moins de 15 ans et de locataires. Ces pourcentages sont arrondis à deux décimales avec la fonction `round`.

```
## Création et cacul de nouveau champs
SR.RMRSherbDonnees$PctPop0_14 <- round(SR.RMRSherbDonnees$Pop0_14 /
                                       SR.RMRSherbDonnees$PopTotAge * 100, 2)

SR.RMRSherbDonnees$PctLocataire <- round(SR.RMRSherbDonnees$Locataire /
                                       SR.RMRSherbDonnees$TotalMenag2 * 100, 2)
```

1.2.7.4 Requêtes attributaires

Dans un SIG, il est fréquent de réaliser une requête attributaire pour explorer les données (par exemple, avec les outils *Select By Attributes* dans ArcGIS Pro et *Sélection avec expression* dans QGIS) et exporter le résultat de la requête dans une nouvelle couche (*Export Features* dans ArcGIS Pro et *Sauvegarder les entités sélectionnées sous…*).

Dans le code ci-dessous, vous trouverez plusieurs exemples de requêtes attributaires. Remarquez que les résultats des requêtes sont enregistrés dans de nouveaux objets `sf` (couches géographiques) dénommés `Requete1` à `Requete5`.

```
## Sélection de l'axe cyclable de la Magog
################################################
# Affichage des valeurs uniques pour le champ NOM de la couche PistesCyclables
unique(PistesCyclables$NOM)
```

```
 [1] "Axe de la Massawippi"         "Axe de la Saint-François"
 [3] "Axe du Ruisseau-Dorman"       "Réseau utilitaire"
 [5] "Tronçon fermé temporairement" "Détour"
 [7] "Axe de la Magog"              "Axe du Ruisseau-Kee"
 [9] "Axe de la Magog Sud"          NA
[11] "Axe du Sommet"
```

```
## Requête attributaire et enregistrement du résultat dans un nouvel objet sf
Requete1 <- subset(PistesCyclables, NOM == "Axe de la Magog")
cat(nrow(Requete1), "enregistrements sélectionnés sur", nrow(PistesCyclables))
```

```
15 enregistrements sélectionnés sur 273
```

```
## Si vous souhaitez connaître uniquement le nombre d'enregistrements sélectionnés
## sans créer un nouvel objet sf, il suffit d'écrire :
nrow(subset(PistesCyclables, NOM == "Axe de la Magog"))
```

```
[1] 15
```

```
## Sélection des SR dont la moitié ou plus des logements sont en location
################################################################################
## Sommaire statistique sur le champ pourcentage de locataires
summary(SR.RMRSherbDonnees$PctLocataire)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   6.92   18.03   41.63   43.81   65.95   93.83
```

```
## Requête attributaire et enregistrement du résultat dans un nouvel objet sf
Requete2 <- subset(SR.RMRSherbDonnees, PctLocataire >= 50)
cat(nrow(Requete2), "enregistrements sélectionnés sur", nrow(SR.RMRSherbDonnees))
```

```
20 enregistrements sélectionnés sur 50
```

```
## Sélection des installations sportives avec un éclairage dans
## l'arrondissement des Nations (deux critères dans la requête)
################################################################################
unique(InstallS.join$NomArrondissement)
```

```
[1] "Arrondissement de Fleurimont"
[2] "Arrondissement de Brompton–Rock Forest–Saint-Élie–Deauville"
[3] "Arrondissement des Nations"
[4] "Arrondissement de Lennoxville"
```

```
table(InstallS.join$ECLAIRAGE)
```

```
Non Oui
 46  85
```

```
## Requête attributaire avec un opérateur AND (&)
Requete3 <- subset(InstallS.join,
                   NomArrondissement == "Arrondissement des Nations" &
                   ECLAIRAGE == "Oui")
cat(nrow(Requete3), "enregistrements sélectionnés sur", nrow(InstallS.join))
```

```
30 enregistrements sélectionnés sur 436
```

```
## Sélection des SR avec deux critères et un opérateur OR (|)
################################################################################
## Sommaires statistiques sur deux champs
summary(SR.RMRSherbDonnees$LoyerMoyen)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  570.0   679.5   729.0   768.4   847.5  1200.0
```

```
summary(SR.RMRSherbDonnees$ValeurMedLog)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 200000  235500  250000  272520  300000  476000
```

```
## Requête attributaire avec un opérateur OR
Requete4 <- subset(SR.RMRSherbDonnees,
                   LoyerMoyen < 700 | ValeurMedLog < 250000)
cat(nrow(Requete4), "enregistrements sélectionnés sur", nrow(SR.RMRSherbDonnees))
```

```
27 enregistrements sélectionnés sur 50
```

```
## Sélection de différents types d'installations sportives
################################################################################
unique(InstallS.join$TYPE)
```

```
 [1] "Aréna"                         "Tir à l'arc"
 [3] "Pétanque"                      "Jeu de galets"
 [5] "Planche à roulettes"           "Préau et plancher de danse"
 [7] "Patinoire à bandes mobiles"    "Surface, anneau ou étang glacé"
 [9] "Patinoire à bandes fixes"      "Plage"
[11] "Jeu d'eau"                     "Piste multifonctionnelle"
[13] "Glissade sur tube"             "Jeu de fers"
[15] "Piscine"                       "Tennis"
[17] "Baseball"                      "Basketball"
[19] "Football"                      "Volleyball"
[21] "Ultimate frisbee"              "Pickleball"
[23] "Soccer"                        "Jeu modulaire"
```

```
## Requête attributaire avec un opérateur %in%
Requete5 <- subset(InstallS.join,
                           TYPE %in% c("Aréna", "Piscine", "Jeu d'eau"))
cat(nrow(Requete5), "enregistrements sélectionnés sur", nrow(InstallS.join))
```

```
26 enregistrements sélectionnés sur 436
```

1.3 Manipulation de données matricielles (raster)

En géomatique, les données matricielles (*raster*) sont une représentation de l'information spatiale sous forme d'une grille rectangulaire composée de cellules élémentaires de taille identique appelées pixels, soit une image. Chaque pixel a une valeur pour une caractéristique spécifique, comme l'altitude, la température, l'utilisation du sol, etc. Les données matricielles sont couramment employées dans les systèmes d'information géographique (SIG) pour la cartographie, l'analyse et la prise de décision en géomatique.

Dans cette section, nous abordons uniquement des fonctions simples de manipulation de données matricielles, notamment le mosaïquage et découpage d'images, et les requêtes attributaires sur des images.

1.3.1 Mosaïquage et découpage d'images

Une fois plusieurs images importées, il est fréquent de vouloir les fusionner. Pour ce faire, nous utilisons deux méthodes du *package* `terra` :

- `terra::merge` fusionne plusieurs images (objets de type `SpatRasters`) pour former un nouvel objet `SpatRasters` dont l'étendue est recalculée en fonction des images fusionnées. Par contre, quand les images se chevauchent, les valeurs des pixels dans les zones de chevauchement seront prises dans le même ordre que les images.
- `terra::mosaic` fusionne aussi plusieurs images. Toutefois, dans les zones de chevauchement, les moyennes des pixels sont calculées. Selon la documentation de `terra`, cette méthode serait plus rapide que la précédente. Dans le code ci-dessous, nous fusionnons les feuillets de modèles numériques d'altitude (MNA) importés dans la section 1.1.2.

```
## Les GeoTIFF importés avec terra sont bien des SpatRaster
class(f21e05_101)
```

```
[1] "SpatRaster"
attr(,"package")
[1] "terra"
```

```
## Création d'une liste pour les cinq feuillets SpatRaster
rlist <- list(f21e05_101, f21e05_201, f31h08_102,
              f31h08_202, f21e12_101)
rsrc <- sprc(rlist)
## Création de la mosaïque
MosaicSherb <- mosaic(rsrc)
MosaicSherb
```

```
class       : SpatRaster
dimensions  : 4187, 5575, 1  (nrow, ncol, nlyr)
resolution  : 9e-05, 9e-05  (x, y)
extent      : -72.25087, -71.74913, 45.24907, 45.6259  (xmin, xmax, ymin, ymax)
coord. ref. : lon/lat NAD83 (EPSG:4269)
source(s)   : memory
varname     : f21e05_101
name        : f21e05_101
min value   :   123.7184
max value   :   845.0474
```

Vous constatez ci-dessus que la projection des images est lon/lat NAD83 (EPSG:4269).

D'autres fonctions permettent de découper une image en fonction d'une autre image (objet `SpatRaster` de `terra`) ou d'un objet `terra` vectoriel (`SpatVector`) :

- `crop(x, y)` découpe une image `x` en prenant l'étendue de `y`.
- `mask(x, y)` découpe une image `x` en prenant la zone (pixels avec des valeurs non nulles ou objets vectoriels) de `y`. Les pixels en dehors de cette zone auront nulle comme valeur (`NA` dans R).

En guise d'exemple, découpons la mosaïque avec le polygone de la ville de Sherbrooke en utilisant la méthode `mask`. Attention, les deux sources de données doivent avoir la même projection et il faut préalablement convertir l'objet `sf` en objet `SpatVector` de `terra`.

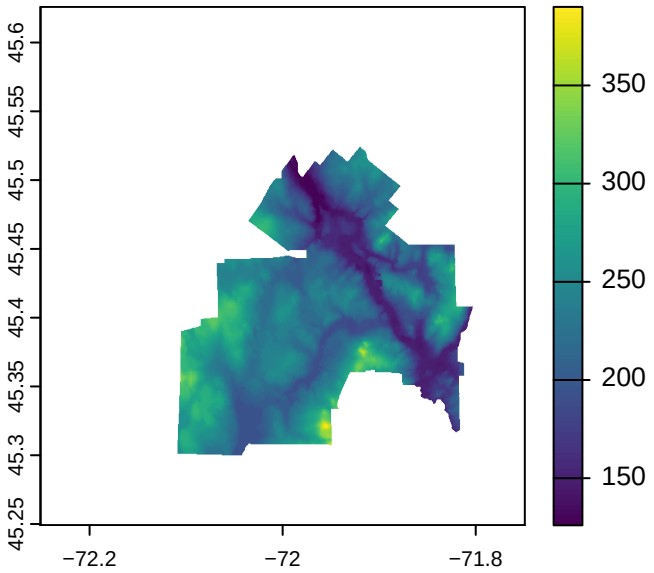
```
## Changement de projection pour le polygone de la ville de Sherbrooke
## Application de la même projection que celle de la mosaïque
VilleSherb.EPSG4269 <- st_transform(Arrond.Union, crs(MosaicSherb))
# Convertir l'objet sf en un objet SpatVector de terra
VilleSherb.SpatVector = vect(VilleSherb.EPSG4269)
## Découpage de la mosaïque avec le polygone de la ville de Sherbrooke
MosaicSherbCrop <- terra::mask(MosaicSherb, VilleSherb.SpatVector)
MosaicSherbCrop
```

```
class       : SpatRaster
dimensions  : 4187, 5575, 1  (nrow, ncol, nlyr)
resolution  : 9e-05, 9e-05  (x, y)
extent      : -72.25087, -71.74913, 45.24907, 45.6259  (xmin, xmax, ymin, ymax)
coord. ref. : lon/lat NAD83 (EPSG:4269)
source(s)   : memory
varname     : f21e05_101
name        : f21e05_101
min value   :        126
max value   :        390
```

```
## Constatez ci-dessus que le nom de l'image est f21e05_101.
## Pour le changer, utilisez la fonction names()
names(MosaicSherbCrop) = "Elevation"
MosaicSherbCrop
```

```
class       : SpatRaster
dimensions  : 4187, 5575, 1  (nrow, ncol, nlyr)
resolution  : 9e-05, 9e-05  (x, y)
extent      : -72.25087, -71.74913, 45.24907, 45.6259  (xmin, xmax, ymin, ymax)
coord. ref. : lon/lat NAD83 (EPSG:4269)
source(s)   : memory
varname     : f21e05_101
name        : Elevation
min value   :       126
max value   :       390
```

```
## Visualisation du résultat
plot(MosaicSherbCrop)
```



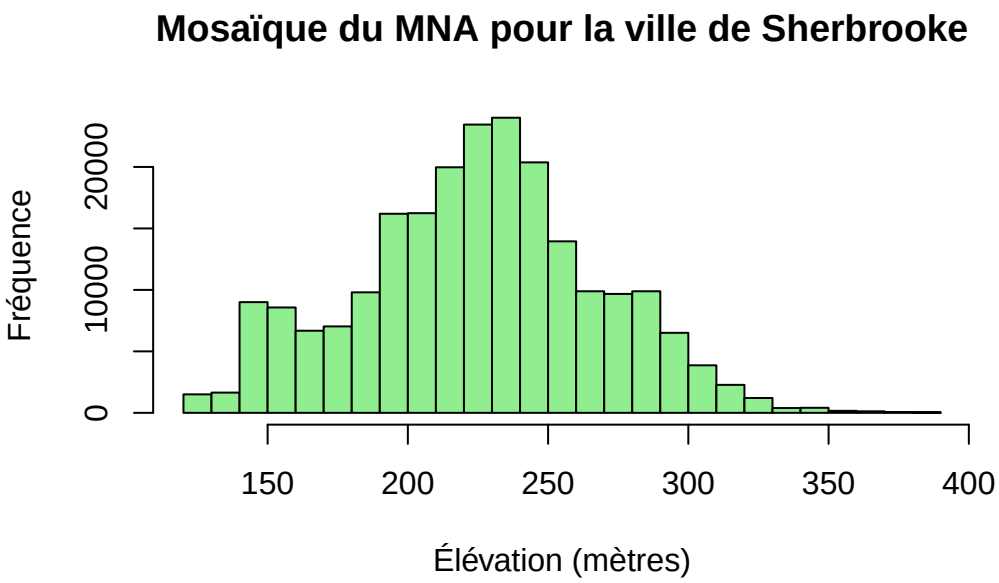
1.3.2 Requêtes attributaires sur des images

Avant d'effectuer une requête, il est judicieux d'explorer les valeurs des pixels de l'image avec un histogramme et la fonction `summary(Nom de l'image)` (valeurs minimales, maximales, quartiles, moyenne et valeurs nulles – `NA`).

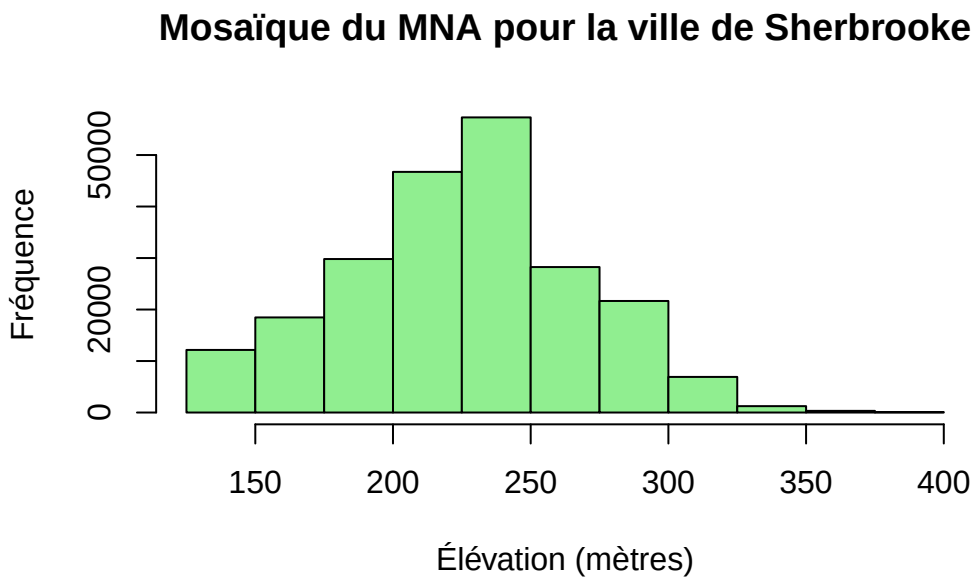
```
## Sommaire statistique des valeurs
summary(MosaicSherbCrop)
```

```
   Elevation
 Min.   :126.0
 1st Qu.:197.7
 Median :227.6
 Mean   :225.4
 3rd Qu.:251.4
 Max.   :389.9
 NA's   :78040
```

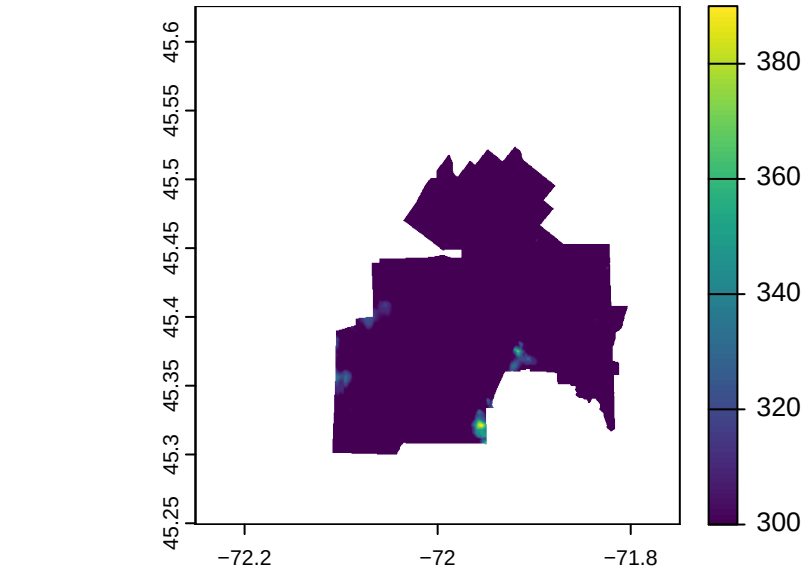
```
## Histogramme
hist(MosaicSherbCrop,
     main = "Mosaïque du MNA pour la ville de Sherbrooke",
     xlab = "Élévation (mètres)", ylab = "Fréquence",
     col = "lightgreen")
```



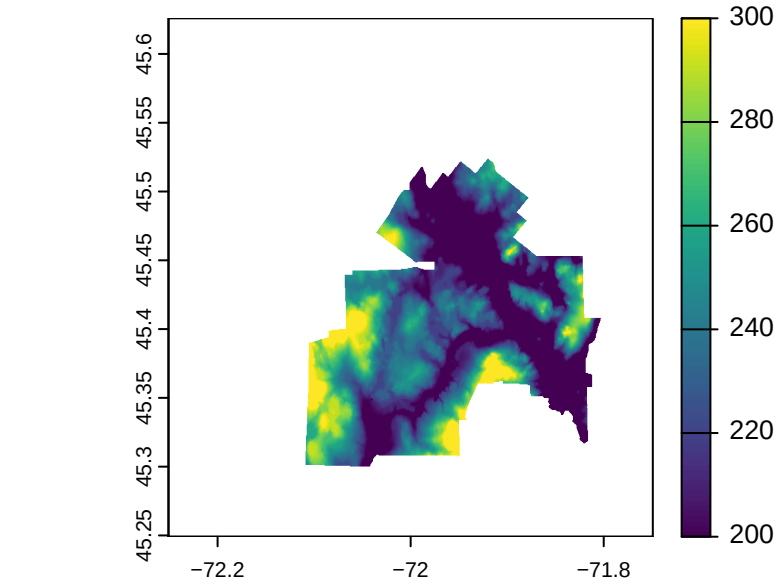

```
## Histogramme en barre de 125 à 400 avec un saut de 25 mètres
hist(MosaicSherbCrop,
     main = "Mosaïque du MNA pour la ville de Sherbrooke",
     xlab = "Élévation (mètres)", ylab = "Fréquence",
     breaks = seq(from = 125, to = 400, by = 25),
     col = "lightgreen")
```




```
## Sélection des pixels avec une élévation d'au moins 300 mètres
MosaicSherbCrop300 = clamp(MosaicSherbCrop, lower = 300)
plot(MosaicSherbCrop300,
     main = "Pixels avec une élévation d'au moins 300 mètres")
```




```
## Sélection des pixels avec une élévation de 200 à 300 mètres
MosaicSherbCrop200_300 = clamp(MosaicSherbCrop, lower = 200, upper = 300)
plot(MosaicSherbCrop200_300,
     main = "Pixels avec une élévation de 200 à 300 mètres")
```



1.4 Exportation de données spatiales de R vers des formats géographiques

1.4.1 Exportation de données vectorielles `sf`

Objectif

Pourquoi exporter des objets `sf` vers différents formats géographiques?
Plusieurs méthodes d'analyse de données spatiales ne sont pas implémentées dans les logiciels de SIG comme ArcGIS Pro ou QGIS d'où l'intérêt de recourir à R ou à Python. La démarche méthodologique classique comprend alors trois étapes :

- Importer des données géographiques.
- Réaliser des analyses avancées dans R ou Python.
- Exporter les résultats finaux vers différents formats géographiques (*shapefile*, *GeoPackage*, *geodatabase* d'ESRI, etc.).

Trois raisons majeures motivent l'exportation des données :

- Cartographier les résultats finaux dans votre logiciel SIG préféré.
- Partager les données avec des personnes n'utilisant pas R.
- Réaliser éventuellement d'autres analyses dans votre logiciel de SIG préféré.

Dans la section 1.1, nous avons vu que la fonction `st_read()` du *package* `sf` permet d'importer une multitude de formats géographiques. Pour exporter avec `sf`, utilisez simplement la fonction `st_write()`. Le code ci-dessous illustre comment exporter des objets `sf` aux formats *shapefile* (`shp`), *GeoPackage* (`GPKG`), *Keyhole Markup Language* (`kml`) et *GeoJSON*. Par défaut, `st_write()` n'écrase pas un fichier existant; pour l'écraser, ajoutez le paramètre `append = FALSE`.

```
## Exportation au format shapefile
st_write(PointsGPS, # couche sf
         "data/chap01/export/PointsGPS.shp",  # chemin et nom du fichier
```

```
         append = FALSE, # pour écraser le fichier s'il existe
         driver = "ESRI Shapefile")
```

```
Deleting layer `PointsGPS' using driver `ESRI Shapefile'
Writing layer `PointsGPS' to data source
  `data/chap01/export/PointsGPS.shp' using driver `ESRI Shapefile'
Writing 89 features with 1 fields and geometry type Point.
```

```
## Exportation dans une couche dans GPKG
st_write(PointsGPS,
         dsn = "data/chap01/export/Data.gpkg",
         layer = "PointsGPS",
         append = FALSE,
         driver = "GPKG")
```

```
Deleting layer `PointsGPS' using driver `GPKG'
Writing layer `PointsGPS' to data source
  `data/chap01/export/Data.gpkg' using driver `GPKG'
Writing 89 features with 1 fields and geometry type Point.
```

```
## Exportation vers un fichier KML
st_write(PointsGPS,
         dsn = "data/chap01/export/PointsGPS.kml",
         append = FALSE,
         driver="KML")
```

```
Writing layer `PointsGPS' to data source
  `data/chap01/export/PointsGPS.kml' using driver `KML'
Writing 89 features with 1 fields and geometry type Point.
```

```
## Exportation vers un fichier GeoJSON
st_write(PointsGPS,
         dsn = "data/chap01/export/PointsGPS.geojson",
         append = FALSE,
         driver="GeoJSON")
```

```
Deleting layer not supported by driver `GeoJSON'
Deleting layer `PointsGPS' failed
Writing layer `PointsGPS' to data source
  `data/chap01/export/PointsGPS.geojson' using driver `GeoJSON'
Updating existing layer PointsGPS
Writing 89 features with 1 fields and geometry type Point.
```

Le paramètre `driver` de la fonction `st_write` permet de spécifier le format du fichier. Pour obtenir la liste des formats qu'il est possible d'importer et d'exporter, tapez dans la console `st_drivers()` ou consultez le tableau 1.1.

Tableau 1.1 – Liste des formats avec le *package* `sf` (`st_drivers`)

	Nom	Description	Écriture	Si vecteur	Si raster
PCIDSK	PCIDSK	PCIDSK Database File	TRUE	TRUE	TRUE
netCDF	netCDF	Network Common Data Format	TRUE	TRUE	TRUE
PDS4	PDS4	NASA Planetary Data System 4	TRUE	TRUE	TRUE
VICAR	VICAR	MIPL VICAR file	TRUE	TRUE	TRUE
JP2OpenJPEG	JP2OpenJPEG	JPEG-2000 driver based on JP2OpenJPEG library	FALSE	TRUE	TRUE
PDF	PDF	Geospatial PDF	TRUE	TRUE	TRUE
MBTiles	MBTiles	MBTiles	TRUE	TRUE	TRUE
BAG	BAG	Bathymetry Attributed Grid	TRUE	TRUE	TRUE
EEDA	EEDA	Earth Engine Data API	FALSE	FALSE	TRUE
OGCAPI	OGCAPI	OGCAPI	FALSE	TRUE	TRUE
ESRI	ESRI	ESRI Shapefile	TRUE	FALSE	TRUE
Shapefile	Shapefile				
MapInfo File	MapInfo File	MapInfo File	TRUE	FALSE	TRUE
UK .NTF	UK .NTF	UK .NTF	FALSE	FALSE	TRUE
LVBAG	LVBAG	Kadaster LV BAG Extract 2.0	FALSE	FALSE	TRUE
OGR_SDTS	OGR_SDTS	SDTS	FALSE	FALSE	TRUE
S57	S57	IHO S-57 (ENC)	TRUE	FALSE	TRUE
DGN	DGN	Microstation DGN	TRUE	FALSE	TRUE
OGR_VRT	OGR_VRT	VRT - Virtual Datasource	FALSE	FALSE	TRUE
Memory	Memory	Memory	TRUE	FALSE	TRUE
CSV	CSV	Comma Separated Value (.csv)	TRUE	FALSE	TRUE
GML	GML	Geography Markup Language (GML)	TRUE	FALSE	TRUE
GPX	GPX	GPX	TRUE	FALSE	TRUE
KML	KML	Keyhole Markup Language (KML)	TRUE	FALSE	TRUE
GeoJSON	GeoJSON	GeoJSON	TRUE	FALSE	TRUE
GeoJSONSeq	GeoJSONSeq	GeoJSON Sequence	TRUE	FALSE	TRUE
ESRIJSON	ESRIJSON	ESRIJSON	FALSE	FALSE	TRUE
TopoJSON	TopoJSON	TopoJSON	FALSE	FALSE	TRUE
OGR_GMT	OGR_GMT	GMT ASCII Vectors (.gmt)	TRUE	FALSE	TRUE
GPKG	GPKG	GeoPackage	TRUE	TRUE	TRUE
SQLite	SQLite	SQLite / Spatialite	TRUE	FALSE	TRUE
ODBC	ODBC	Open Database Connectivity (ODBC)	FALSE	FALSE	TRUE
WAsP	WAsP	WAsP .map format	TRUE	FALSE	TRUE
PGeo	PGeo	ESRI Personal GeoDatabase	FALSE	FALSE	TRUE
MSSQLSpatial	MSSQLSpatial	Microsoft SQL Server Spatial Database	TRUE	FALSE	TRUE
PostgreSQL	PostgreSQL	PostgreSQL/PostGIS	TRUE	FALSE	TRUE
MySQL	MySQL	MySQL	TRUE	FALSE	TRUE
OpenFileGDB	OpenFileGDB	ESRI FileGDB	TRUE	TRUE	TRUE
DXF	DXF	AutoCAD DXF	TRUE	FALSE	TRUE
CAD	CAD	AutoCAD Driver	FALSE	TRUE	TRUE

Tableau 1.1 – Liste des formats avec le *package* `sf` (`st_drivers`)

	Nom	Description	Écriture	Si vecteur	Si raster
FlatGeobuf	FlatGeobuf	FlatGeobuf	TRUE	FALSE	TRUE
Geoconcept	Geoconcept	Geoconcept	TRUE	FALSE	TRUE
GeoRSS	GeoRSS	GeoRSS	TRUE	FALSE	TRUE
VFK	VFK	Czech Cadastral Exchange Data Format	FALSE	FALSE	TRUE
PGDUMP	PGDUMP	PostgreSQL SQL dump	TRUE	FALSE	TRUE
OSM	OSM	OpenStreetMap XML and PBF	FALSE	FALSE	TRUE
GPSBabel	GPSBabel	GPSBabel	TRUE	FALSE	TRUE
OGR_PDS	OGR_PDS	Planetary Data Systems TABLE	FALSE	FALSE	TRUE
WFS	WFS	OGC WFS (Web Feature Service)	FALSE	FALSE	TRUE
OAPIF	OAPIF	OGC API - Features	FALSE	FALSE	TRUE
EDIGEO	EDIGEO	French EDIGEO exchange format	FALSE	FALSE	TRUE
SVG	SVG	Scalable Vector Graphics	FALSE	FALSE	TRUE
Idrisi	Idrisi	Idrisi Vector (.vct)	FALSE	FALSE	TRUE
XLS	XLS	MS Excel format	FALSE	FALSE	TRUE
ODS	ODS	Open Document/ LibreOffice / OpenOffice Spreadsheet	TRUE	FALSE	TRUE
XLSX	XLSX	MS Office Open XML spreadsheet	TRUE	FALSE	TRUE
Elasticsearch	Elasticsearch	Elastic Search	TRUE	FALSE	TRUE
Carto	Carto	Carto	TRUE	FALSE	TRUE
AmigoCloud	AmigoCloud	AmigoCloud	TRUE	FALSE	TRUE
SXF	SXF	Storage and eXchange Format	FALSE	FALSE	TRUE
Selafin	Selafin	Selafin	TRUE	FALSE	TRUE
JML	JML	OpenJUMP JML	TRUE	FALSE	TRUE
PLSCENES	PLSCENES	Planet Labs Scenes API	FALSE	TRUE	TRUE
CSW	CSW	OGC CSW (Catalog Service for the Web)	FALSE	FALSE	TRUE
VDV	VDV	VDV-451/VDV-452/INTREST Data Format	TRUE	FALSE	TRUE
MVT	MVT	Mapbox Vector Tiles	TRUE	FALSE	TRUE
NGW	NGW	NextGIS Web	TRUE	TRUE	TRUE
MapML	MapML	MapML	TRUE	FALSE	TRUE
GTFS	GTFS	General Transit Feed Specification	FALSE	FALSE	TRUE
PMTiles	PMTiles	ProtoMap Tiles	TRUE	FALSE	TRUE
JSONFG	JSONFG	OGC Features and Geometries JSON	TRUE	FALSE	TRUE
MiraMonVector	MiraMonVector	MiraMon Vectors (.pol, .arc, .pnt)	TRUE	FALSE	TRUE
TIGER	TIGER	U.S. Census TIGER/Line	FALSE	FALSE	TRUE
AVCBin	AVCBin	Arc/Info Binary Coverage	FALSE	FALSE	TRUE
AVCE00	AVCE00	Arc/Info E00 (ASCII) Coverage	FALSE	FALSE	TRUE
HTTP	HTTP	HTTP Fetching Wrapper	FALSE	TRUE	TRUE

1.4.2 Exportation de données raster

L'exportation d'objets `SpatRasters` de `terra` est très simple avec la méthode `terra::writeRaster`. En guise d'exemple, le code ci-dessous exporte la mosaïque de MNA dans un fichier GeoTIFF. Notez que le paramètre `filetype` permet de

spécifier d'autres formats d'images de la liste qui est disponible au lien suivant : *https://gdal.org/drivers/raster/index.html*. En guise d'exemple, les paramètres `EIR`, `ENVI`, `RST`, `ERS` et `GRASS` permettent d'exporter vers les logiciels de télédétection ERDAS, ENVI, Idrisi, ERMapper et GRASS, tandis que le paramètre `GPKG` permet d'exporter vers un *GeoPackage* raster.

```
terra::writeRaster(MosaicSherbCrop, "data/chap01/export/MosaicSherb.tif",
                   filetype = "GTiff",
                   overwrite = TRUE)
```

1.5 Cartographie avec R

Objectif

Pourquoi cartographier des données dans R?
Vous avez certainement un logiciel de SIG préféré pour construire une carte thématique (QGIS ou ArcGIS Pro par exemple). Puisqu'en quelques clics de souris, il est facile de réaliser une carte dans un SIG, quel est donc l'intérêt d'écrire des lignes de code pour afficher une carte dans R? Autrement dit, pourquoi devriez-vous vous compliquer la vie à apprendre de la syntaxe R pour produire une simple carte? Savoir cartographier dans R a plusieurs avantages :

- Cartographier rapidement les résultats d'une analyse dans R permet d'éviter des allers-retours (exportation et importation de données) entre R et un logiciel de SIG. Or, la cartographie fait partie intégrante d'une démarche méthodologique d'analyse ou de modélisation spatiale. Vous restez ainsi dans le même environnement de travail (R) jusqu'à l'obtention de vos résultats finaux. Une fois ces derniers obtenus, vous pouvez les exporter et construire une carte très élaborée dans un logiciel de SIG.
- La syntaxe R n'est pas si compliquée. Quelques lignes de code écrites pour une première analyse peuvent être réutilisées, modifiées et bonifiées pour une autre analyse. Au fil de vos projets, vous construirez des cartes de plus en plus élaborées. Autrement dit, après quelques heures d'investissement, vous deviendrez une personne experte en cartographie dans R!

Package

Quels *packages* utiliser pour la cartographie dans R?
Il existe plusieurs *packages* R pour la cartographie, notamment :

- `ggplot2` est certainement le meilleur *package* R pour réaliser des graphiques (Wickham 2016). Il permet désormais de construire des cartes.
- `cartography` permet de construire efficacement des cartes thématiques (Giraud et Lambert 2016). Pour avoir une idée de son potentiel, consultez cette *Cheatsheet*.
- `tmap` (Tennekes 2018) est actuellement l'un des *packages* les plus complets et les plus utilisés pour construire des cartes thématiques.
- Des *packages* spécifiques permettent de créer des cartes interactives sur Internet, notamment `mapview`, `mapdeck` et `leaflet`. Ce dernier est basé sur la librairie JavaScript, largement utilisée dans le domaine de la cartographie sur Internet.

Dans cette section, nous utilisons uniquement `tmap` dont plusieurs ressources sont disponibles sur Internet :

- Sur le site CRAN de `tmap`, une excellente vignette intitulée *tmap: get started!*
- Un article dans Journal of Statistical Software de Martijn Tennekes, créateur du *package* `tmap`.
- La documentation complète en PDF.

1.5.1 Manipulation des couches géométriques

1.5.1.1 Principales fonctions de représentation de couches vectorielles et matricielles

Il existe trois catégories de fonctions pour paramétrer l'affichage de couches géographiques (tableau 1.2).

Tableau 1.2 – Principales fonctions pour manipuler des couches vectorielles et matricielles

Fonction	Description	Points	L
Fonction principale			
tm_shape	Crée un élément tmap à partir d'une couche géographique vectorielle (sf) ou matricielle (raster)	X	
Fonctions de base de manipulation			
tm_polygons	Dessine des polygones (couleur et contour)		
tm_symbols	Dessine des symboles	X	
tm_lines	Dessine des lignes		
tm_text	Dessine des étiquettes à partir d'un champ	X	
tm_raster	Affiche un raster		
Autres fonctions de manipulation			
tm_fill	Dessine l'intérieur de polygones		
tm_border	Dessine les contours		
tm_bubbles	Dessine des cercles (notamment proportionnels)	X	
tm_squares	Dessine des carrés (notamment proportionnels)	X	
tm_dots	Dessine des points	X	
tm_markers	Dessine des icones avec étiquettes	X	

Construction d'une carte simple avec une couche vectorielle et une couche matricielle

Le code ci-dessous permet d'afficher deux couches avec la fonction `tm_shape` : l'une vectorielle, l'autre matricielle (figure 1.16).

```
tmap_mode("plot")
# 1er objet tmap pour une couche raster
tm_shape(MosaicSherbCrop)+
  tm_raster(palette = terrain.colors(10))+
# 1er objet tmap pour une couche vectorielle
tm_shape(Arrondissements)+
  tm_borders(col = "black", lwd = 3)+ # contour noir avec une épaisseur de trois points
  tm_text("NUMERO") # Étiquettes identifiant l'arrondissement
```

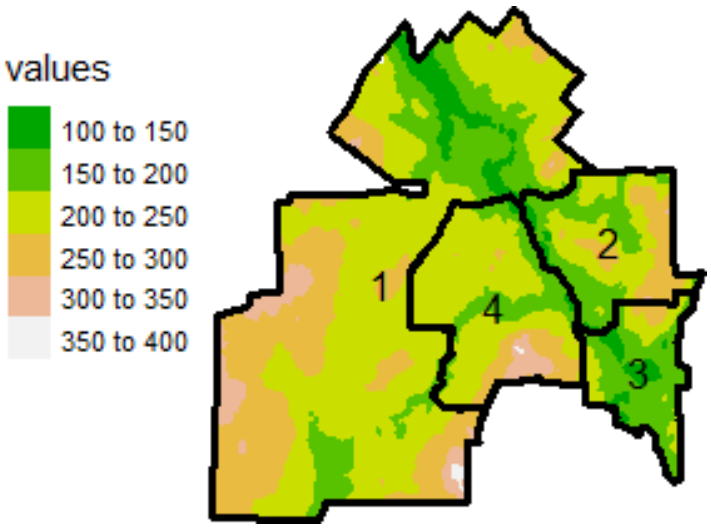


FIGURE 1.16 – Exemple de carte construite avec le *package* `tmap` avec une couche polygonale et une image

⚠ **Attention**

Ordre et hiérarchie des couches avec `tmap`.
Vous avez compris qu'une couche est affichée avec la fonction `tm_shape` et que le `+` permet d'ajouter une ou plusieurs fonctions d'habillage à cette couche (`tm_polygons`, `tm_lines`, `tm_text`, `tm_raster`, etc.).
Il est possible d'en superposer en utilisant plusieurs `tm_shape` comme suit :
`tm_shape(Nom de la première couche)+ ... paramètres de la couche + tm_shape(Nom de la seconde couche)+ ... paramètres de la couche`
Notez que la première couche est celle avec laquelle la projection et l'étendue de la carte sont définies. Il est toutefois possible de changer le tout en utilisant l'argument `is.master = TRUE` dans le `tm_shape` d'une couche donnée.

Construction d'une carte avec plusieurs couches vectorielles

Les lignes de code suivantes permettent de construire la figure 1.17 avec trois couches `sf`.

```
tmap_mode("plot")
## Polygones
tm_shape(Arrondissements)+
  tm_text("NUMERO")+ # Étiquettes identifiant l'arrondissement
  tm_polygons(col="wheat", border.col = "black", lwd = 3)+
## Lignes
tm_shape(Rues)+
  tm_lines(col= "gray", lwd = 1)+
## Points
tm_shape(PointsGPS.Sherb)+
  tm_dots(shape=21, col="blue", size=.3)
```

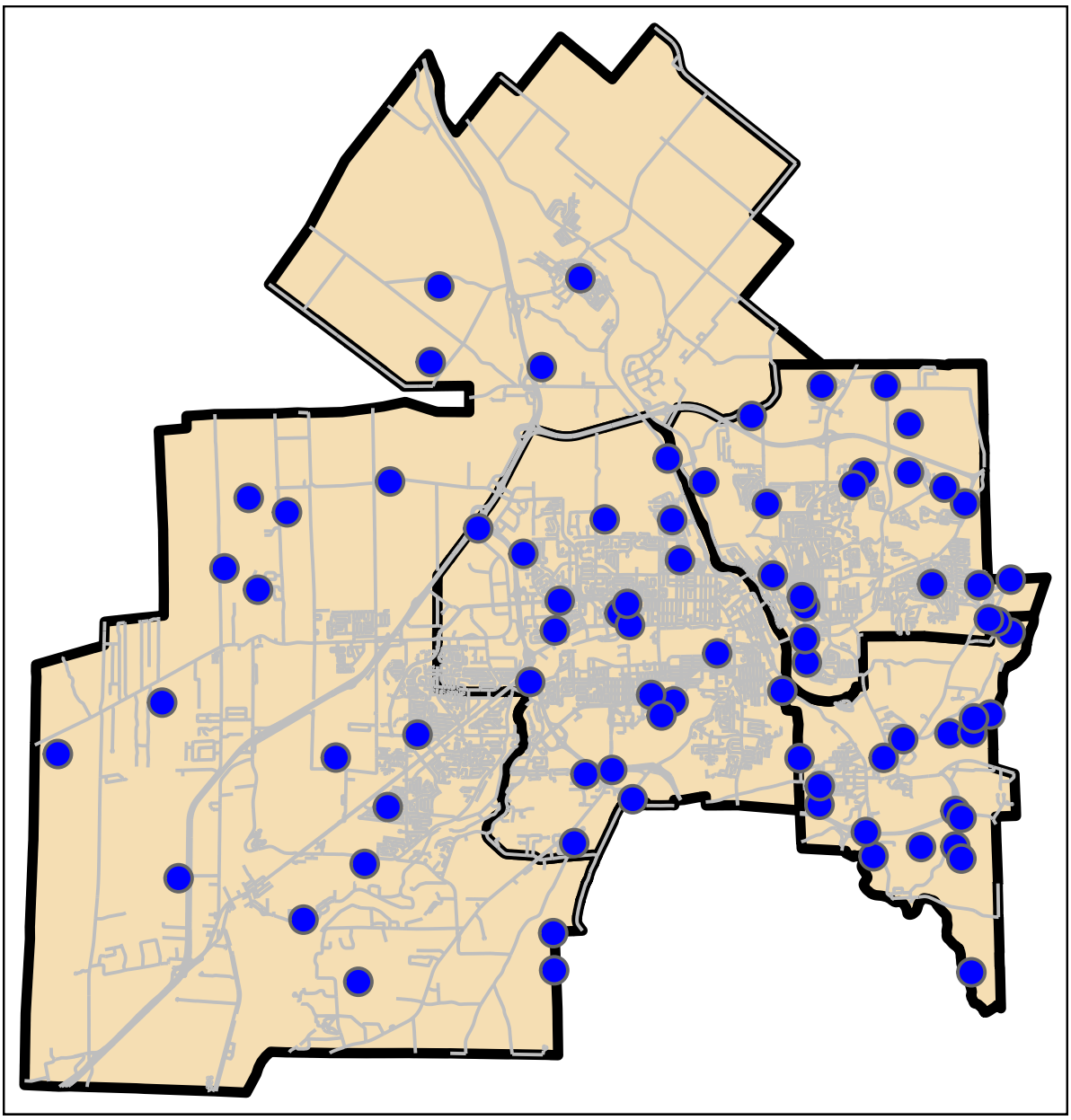


FIGURE 1.17 – Exemple de carte construite avec le *package* `tmap` avec plusieurs couches vectorielles (polygones, lignes, points)

La figure 1.18 illustre la différence entre les fonctions `tm_dots` et `tm_markers`.

```
## Points avec tm_dots()
CartePoints <-
  tm_shape(Arrondissements) + tm_polygons(col="wheat", border.col = "black") +
  tm_shape(PointsGPS.Sherb) + tm_dots(shape=21, col="blue", size=.3)
## Icones avec tm_markers()
CarteMarkers <-
```

```
  tm_shape(Arrondissements) + tm_polygons(col="wheat", border.col = "black") +
  tm_shape(PointsGPS.Sherb) + tm_markers(size = 0.2, border.col = rgb(0,0,0,0))
## Combinaison des deux cartes
tmap_arrange(CartePoints, CarteMarkers, ncol=2, nrow=1)
```

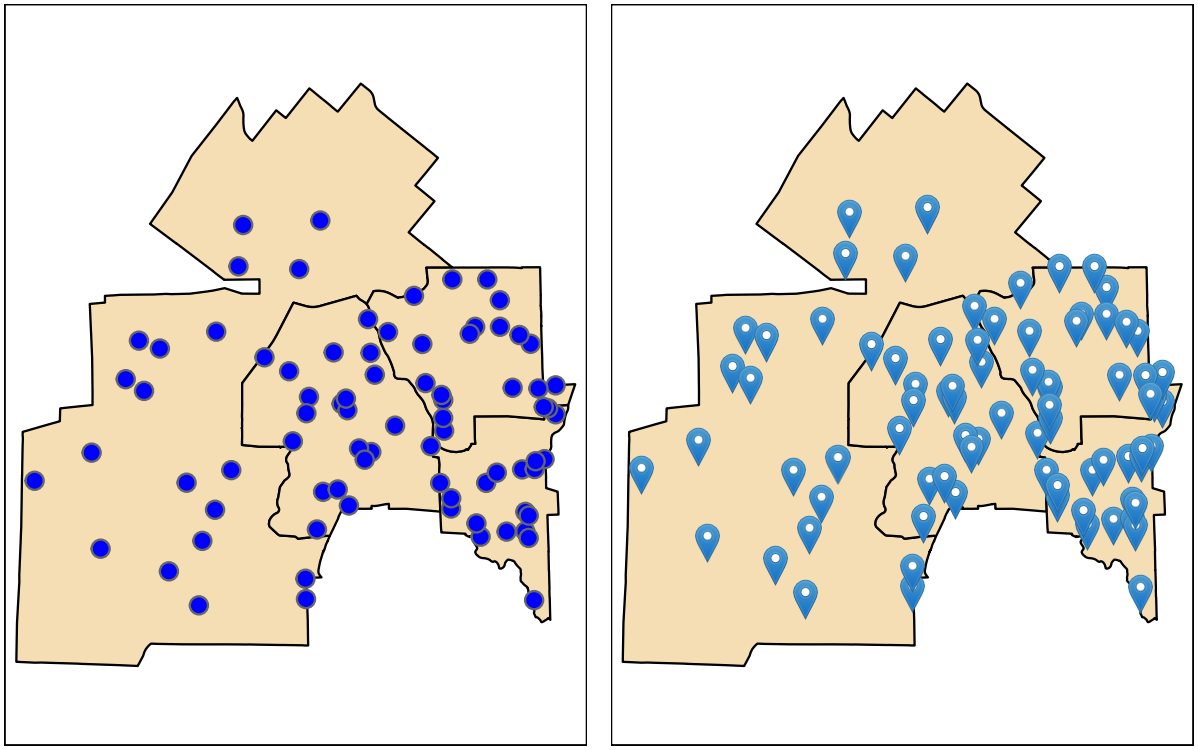


FIGURE 1.18 – Exemple de carte `tmap` avec `tm_dots` et `tm_markers`

1.5.1.2 Couleurs uniques et palette de couleurs dans `tmap`

Vous avez remarqué plus haut que plusieurs fonctions comprennent l'argument `col` pour spécifier une couleur. Pour connaître les trois manières de spécifier une couleur dans R – nom de la couleur R (`lightblue` par exemple), code hexadécimal (`#f03b20` par exemple) ou notation RVBA (`rgb(0.2, 0.4, 0.4, 0)` par exemple) –, consultez la section suivante (Apparicio et Gelb 2022).

Pour spécifier une palette de couleurs sur un champ dans différentes fonctions (entre autres, `tm_polygons`, `tm_lines`, `tm_fill`, `tm_dots`), il suffit d'utiliser deux arguments dans la fonction, soit `col="Nom du champ"` et `palette="nom de la palette de couleurs"`.

Le *package* `tmap` intègre les palettes de deux autres *packages* : `viridisLite` (Garnier et al. 2021) et `RColorBrewer` (Neuwirth 2022). Le premier propose cinq palettes de couleurs : `viridis`, `magma`, `plasma`, `inferno`, `cividis`. Le second intègre une série de palettes de couleurs proposées par la géographe et cartographe Cynthia Brewer et ses collègues (Harrower et Brewer 2003; Brewer, Hatchard et Harrower 2003). Vous avez probablement déjà exploré leur site Internet où il est possible de sélectionner une palette en fonction du nombre de classes, de la nature des données et de la codification des couleurs (HEX, RGB, CMYK). Succinctement, `RColorBrewer` propose plusieurs palettes regroupées selon trois catégories :

— Palettes qualitatives à appliquer à une variable qualitative nominale comme son nom l'indique (figure 1.19). Pour afficher les palettes et connaître leurs noms, tapez `display.brewer.all(type="qual")` dans la console.
— Palettes séquentielles pour une variable continue avec des valeurs faibles à fortes (figure 1.20). Tapez `display.brewer.all(type="seq")` dans la console.
— Palettes divergentes à appliquer à une variable continue dont les valeurs aux deux extrémités s'opposent (figure 1.21). Tapez `display.brewer.all(type="div")` dans la console.

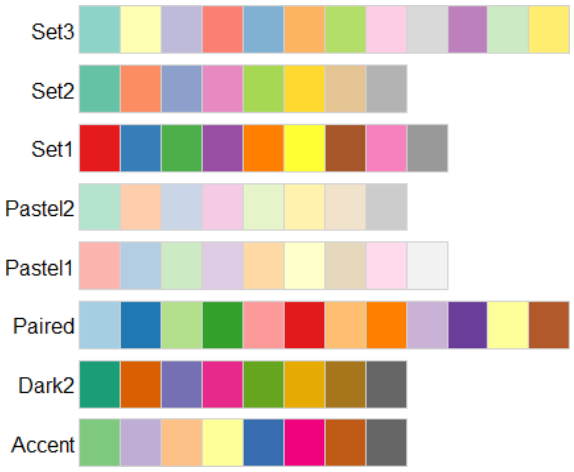


FIGURE 1.19 – Palettes de couleurs qualitatives du *package* `RColorBrewer`

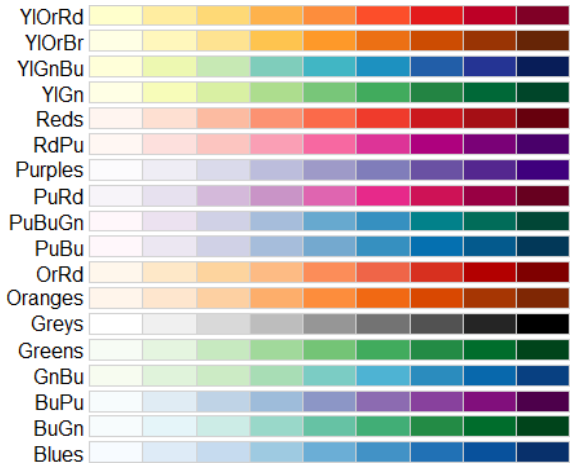


FIGURE 1.20 – Palettes de couleurs séquentielles du *package* `RColorBrewer`

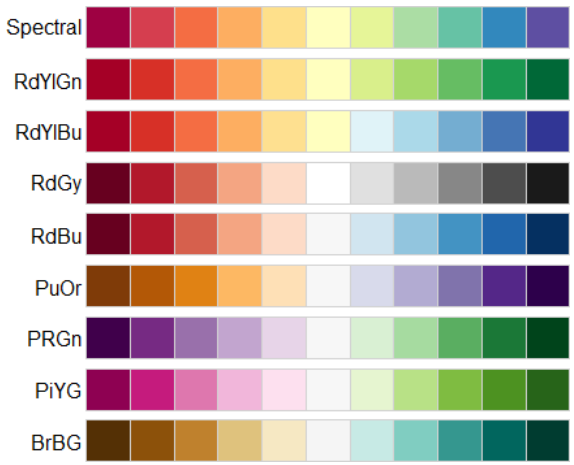


FIGURE 1.21 – Palettes de couleurs divergentes du *package* `RColorBrewer`

Astuce

Comparaison de palettes avec un nombre de classes défini

Si vous connaissez le nombre de classes, mais que vous hésitez à choisir telle ou telle palette de couleurs, tapez dans la console :

- `display.brewer.all(n=5, type="seq", exact.n=TRUE)`
- `display.brewer.all(n=5, type="div", exact.n=TRUE)`
- `display.brewer.all(n=5, type="qual", exact.n=TRUE)`

D'autres arguments peuvent être ajoutés comme `colorblindFriendly=TRUE` qui renvoie uniquement des palettes de couleurs adaptées aux personnes daltoniennes. En guise d'exemple, avec cinq classes, il est possible de comparer neuf palettes divergentes et six autres adaptées aux personnes daltoniennes (figure 1.22).

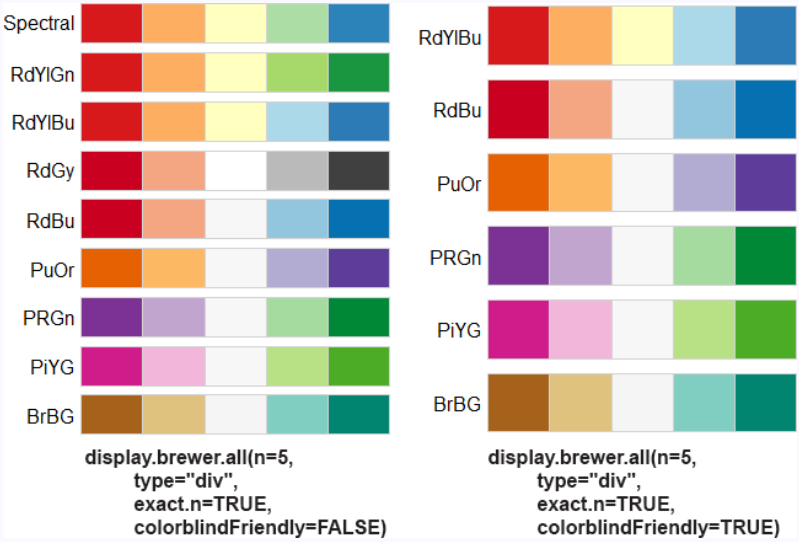


FIGURE 1.22 – Palettes de couleurs divergentes du *package* `RColorBrewer` avec cinq classes

Vous hésitez encore à choisir une palette de couleurs? Tapez la syntaxe ci-dessous dans la console pour afficher l'ensemble des palettes des *packages* `RColorBrewer` et `viridisLite`.

```
tmaptools::palette_explorer()
```

Pour inverser les couleurs d'une palette, vous devez précéder le nom de la palette par un signe moins (exemple : `-Greens`).

1.5.1.3 Cartographie d'une variable qualitative : valeurs uniques

Application à une couche de points

Le code ci-dessous illustre comment construire une carte thématique avec des couleurs appliquées à une variable qualitative nominale (champ `TYPE` de la couche `InstallationSport`) d'une couche de points (figure 1.23).

```
## Carte
tm_shape(Arrondissements)+
  tm_borders()+
tm_shape(InstallationSport)+
  tm_dots(shape = 21,
          size=.3,
          col= "TYPE",
          palette = "Set1",
          title ="Type d'installation")+
tm_layout(main.title = "Installations sportives",
          frame=FALSE,
          legend.position = c("left", "top"),
          legend.outside=TRUE)
```

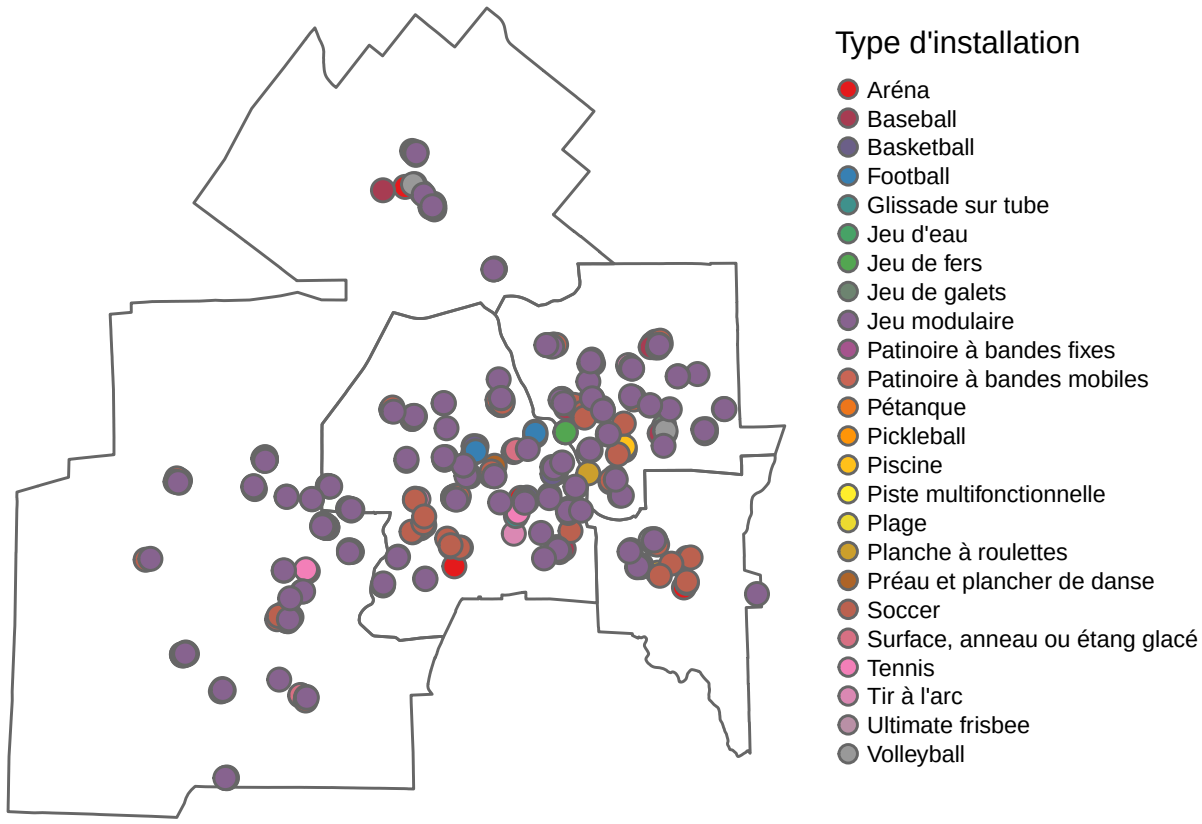


FIGURE 1.23 – Exemple de cartographie d'une variable qualitative sur des points

Application à une couche de lignes

Le code ci-dessous illustre comment construire une carte thématique avec des couleurs appliquées à une variable qualitative nominale (champ `TYPESEGMEN`) d'une couche de lignes (figure 1.24).

```
## Listes des valeurs uniques
table(Rues$TYPESEGMEN)
```

```
      Artère    Autoroute Chemin privé  Collectrice       Locale 
        1459          380          467          579         4844 
```

```
## Lignes
tmap_mode("plot")
tm_shape(Rues)+
  tm_lines(col= "TYPESEGMEN",
           palette = c("red", "brown4", "cornsilk1", "lightpink", "gainsboro"),
           lwd = 2
           )
```

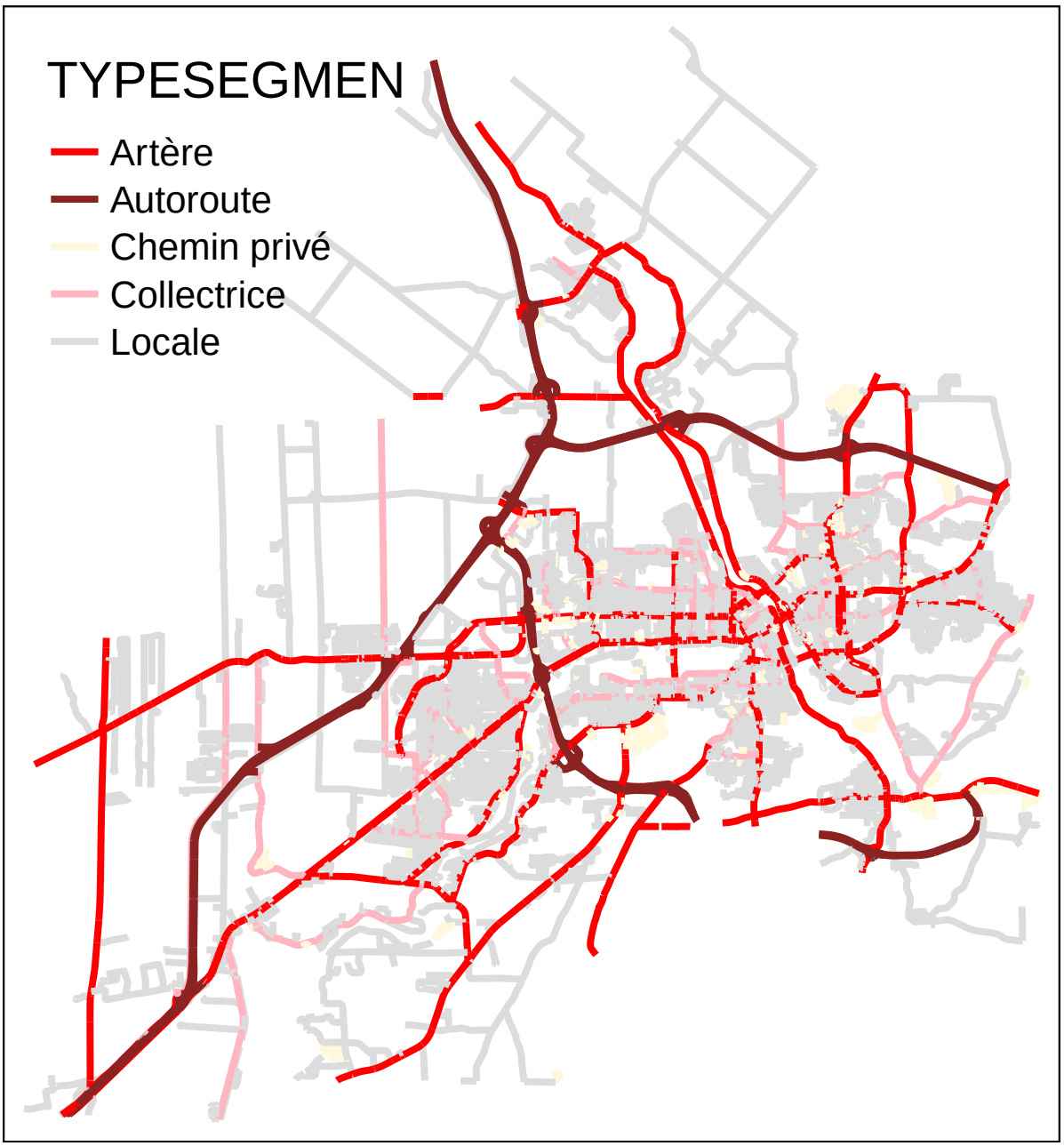


FIGURE 1.24 – Exemple de cartographie d'une variable qualitative sur des lignes

Application à une couche de polygones

Le code ci-dessous illustre comment construire une carte thématique avec des couleurs appliquées à une variable qualitative nominale (champ `SDRNOM` de la couche `AD2021`) d'une couche de polygones (figure 1.25).

```
## Importation de la couche des aires de diffusion de 2021 pour la RMR de Sherbrooke
AD2021 <- st_read(dsn = "data/chap01/gpkg/Recen2021Sherbrooke.gpkg",
                  layer = "SherbAD",
```

```
                  quiet = TRUE)
## Carte
tmap_mode("plot")
tm_shape(AD2021)+
  tm_fill(col= "SDRNOM",
          palette = "Set2",
          lwd = 1,
          title ="Municipalité")+
  tm_borders(col="black")+
tm_layout(main.title = "Aires de diffusion de 2021",
          frame =FALSE,
          legend.position = c("left", "top"),
          legend.outside=TRUE)
```

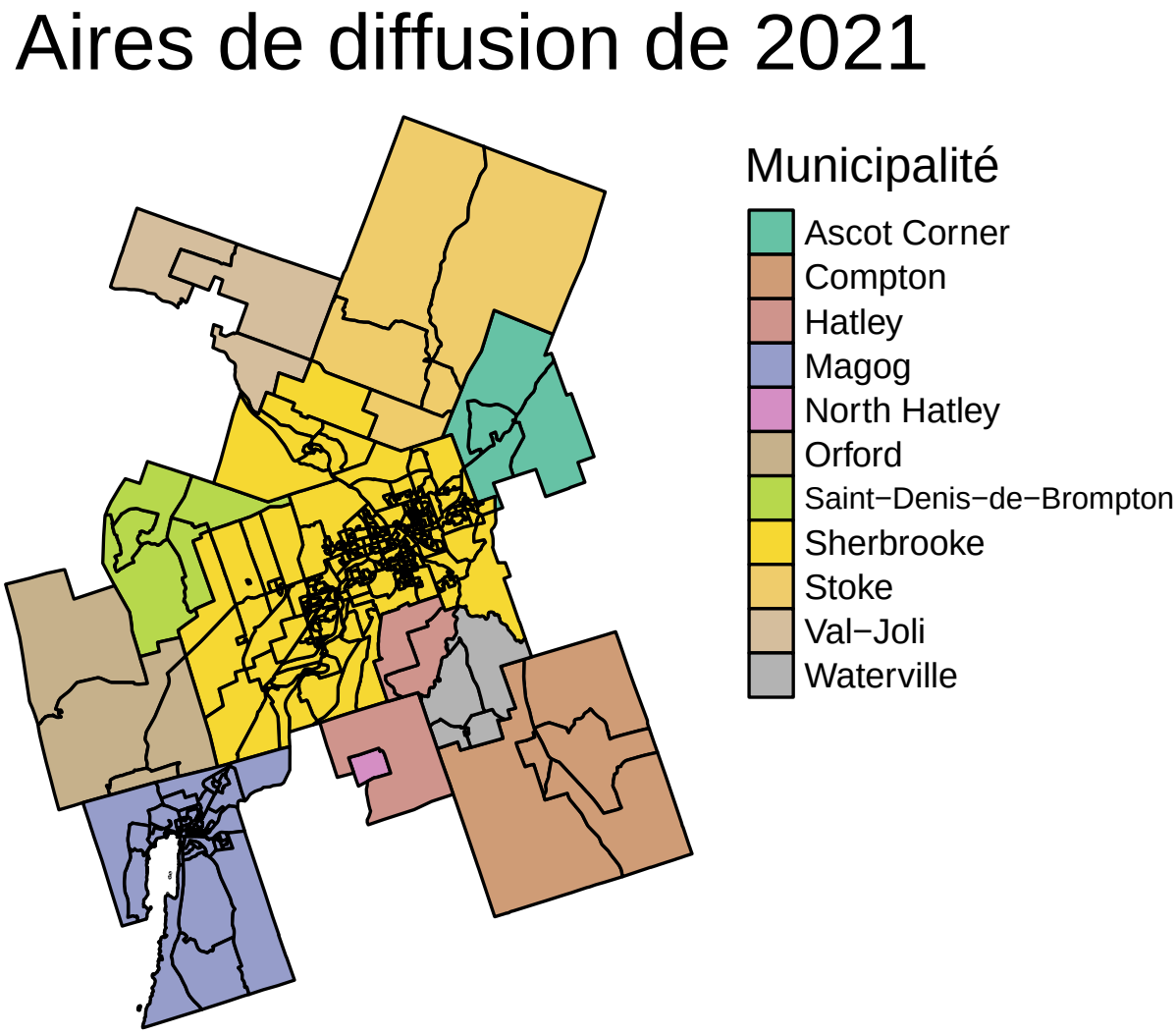


Figure 1.25 – Exemple de cartographie d'une variable qualitative sur des polygones

1.5.1.4 Cartographie d'une variable discrète : cercles proportionnels

La syntaxe ci-dessous permet de créer une carte avec des cercles proportionnels pour les municipalités de la région administrative de l'Estrie (figure 1.26).

```
## Importation des municipalités (subdivisions de recensements - SDR) de l'Estrie
SDR.Estrie <- st_read(dsn = "data/chap01/gpkg/Recen2021Sherbrooke.gpkg",
                  layer = "sdr_Estrie", quiet = TRUE)
## Importation des MRC (divisions de recensements - DR) de l'Estrie
DR.Estrie <- st_read(dsn = "data/chap01/gpkg/Recen2021Sherbrooke.gpkg",
                  layer = "DREstrie2021",  quiet = TRUE)
## Importation des données sur la population
PopSDR <- read.csv("data/chap01/tables/SDR_Estrie.csv")
PopSDR$SDRidu <- as.character(PopSDR$SDRidu)
## Fusion des données
SDR.Estrie <- merge(SDR.Estrie, PopSDR, by.x = "SDRIDU", by.y = "SDRidu")
## Construction de la carte
tmap_mode("plot")
tm_shape(SDR.Estrie)+
  tm_polygons(col="whitesmoke", border.col = "grey30", lwd = 1)+
  tm_bubbles(size = "SDRpop_2021",
             border.col = "black",
             col = "tomato1",
             title.size = "Population",
             scale = 3)+ # facteur multiplicateur pour la taille du cercle
tm_shape(DR.Estrie)+
  tm_borders(col="black", lwd = 2)
```

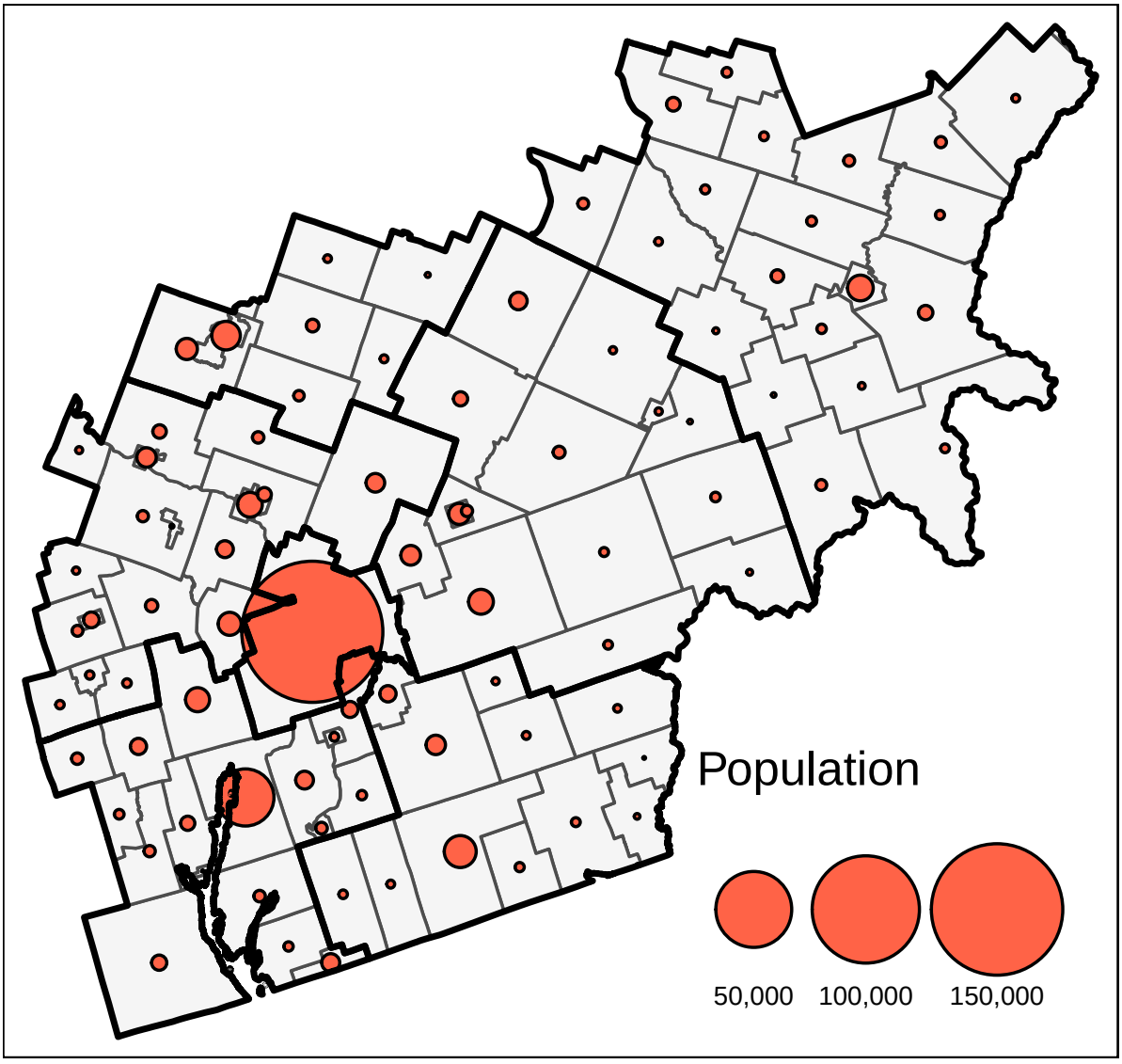


FIGURE 1.26 – Exemple de carte avec des cercles proportionnels

1.5.1.5 Cartographie d'une variable continue : cartes choroplèthes et méthodes de discrétisation

L'argument `style`, qui est commun à plusieurs fonctions (`tm_polygons`, `tm_fill`, `tm_lines`, `tm_dots`, etc.), permet de choisir une méthode de discrétisation dont les principales sont :

- `fixed`: intervalles fixés par l'analyste.
- `equal`: intervalles égaux.
- `pretty`: intervalles arrondis aux nombres entiers.
- `quantile`: selon les quantiles (même nombre d'observations dans chaque classe).
- `jenks`: selon la méthode de Jenks.
- `sd`: selon l'écart-type.

D'autres méthodes peuvent être utilisées comme `kmeans`, `hclust`, `bclust`, `fisher`, `dpih`, `headtails` et `log10_pretty`. En guise d'exemple, la figure 1.28 présente une discrétisation en cinq classes selon la méthode des quantiles. Notez aussi qu'il est possible de réaliser une carte avec un dégradé continu avec `style = "cont"` tel qu'illustré ci-dessous (figure 1.27).

```
## Sélection des aires de diffusion de Sherbrooke
AD2021.sherb <- subset(AD2021, SDRNOM == "Sherbrooke")
## Carte
tmap_mode("plot")
tm_shape(AD2021.sherb)+
  tm_fill(col= "HabKm2",
          palette = "Reds",
          style = "cont",
          title ="Hab./km2")+
  tm_borders(col="black")
```

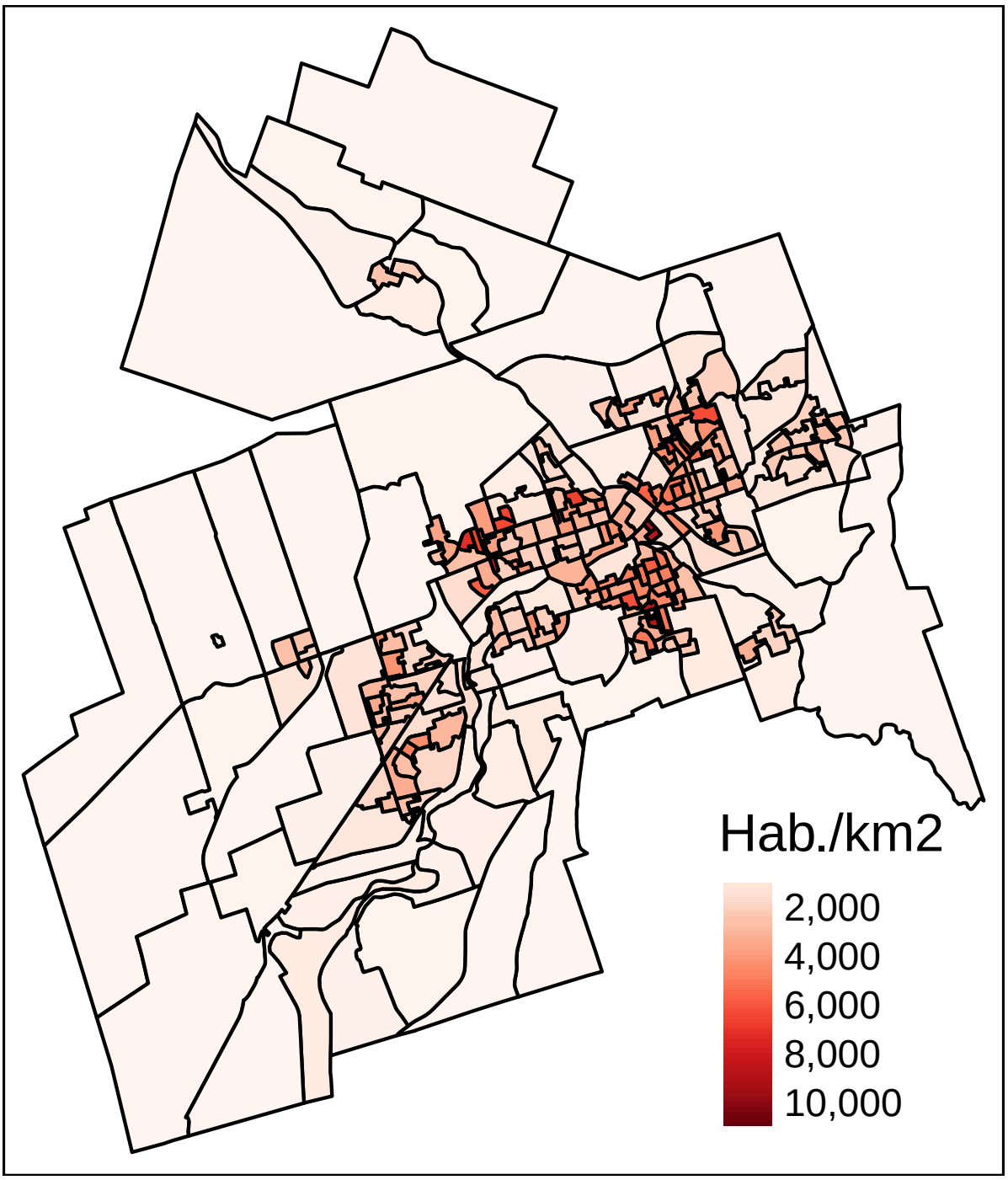


Figure 1.27 – Exemple de carte choroplèthe avec une palette continue

La figure 1.28 utilise une discrétisation selon la méthode de quantiles avec cinq classes. Autrement dit, chaque classe comprend 20 % des aires de diffusion de la ville de Sherbrooke.

```
tmap_mode("plot")
tm_shape(AD2021.sherb)+
```

```
  tm_fill(col= "HabKm2",
          palette = "Reds",
          n = 5, # nombre de classes
          style = "quantile",
          legend.format = list(text.separator = "à"),
          title ="Hab./km2")+
  tm_borders(col="black", lwd = .5)
```

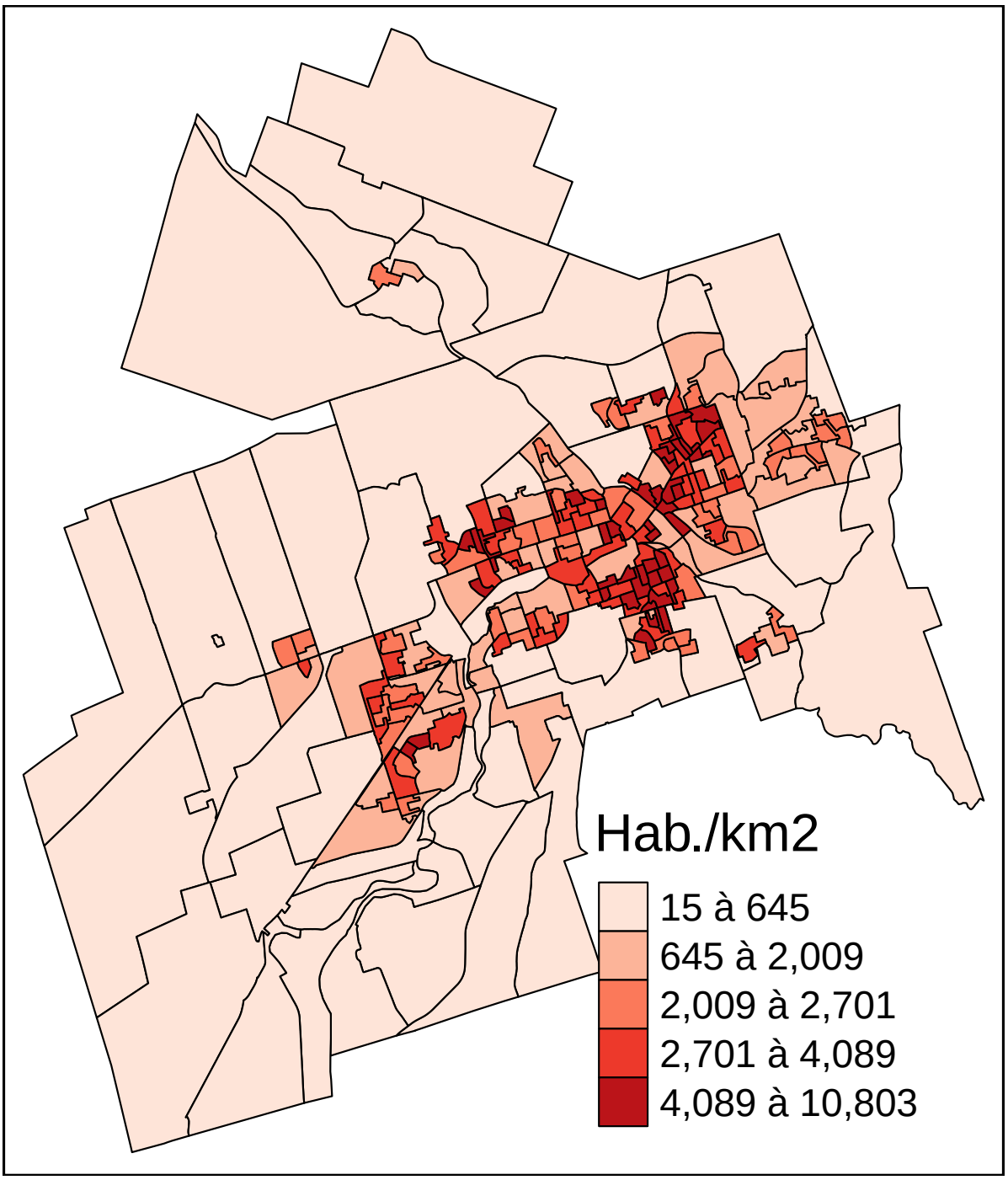


Figure 1.28 – Exemple de carte choroplèthe avec une discrétisation selon les quantiles

La figure 1.29 présente quatre méthodes de discrétisation différentes appliquées au revenu médian des ménages par secteur de recensement dans la région métropolitaine de recensement de Sherbrooke en 2021.

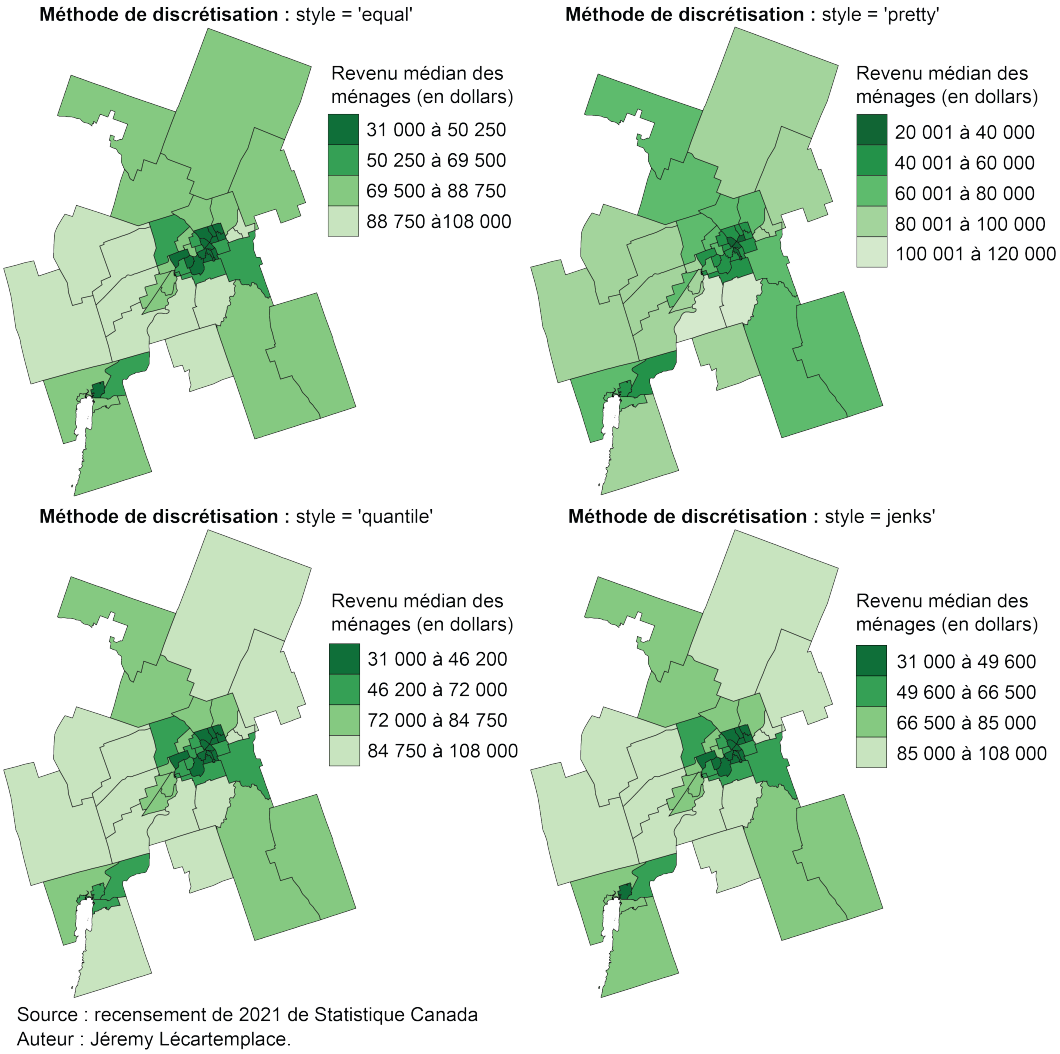


Figure 1.29 – Différentes méthodes de discrétisation

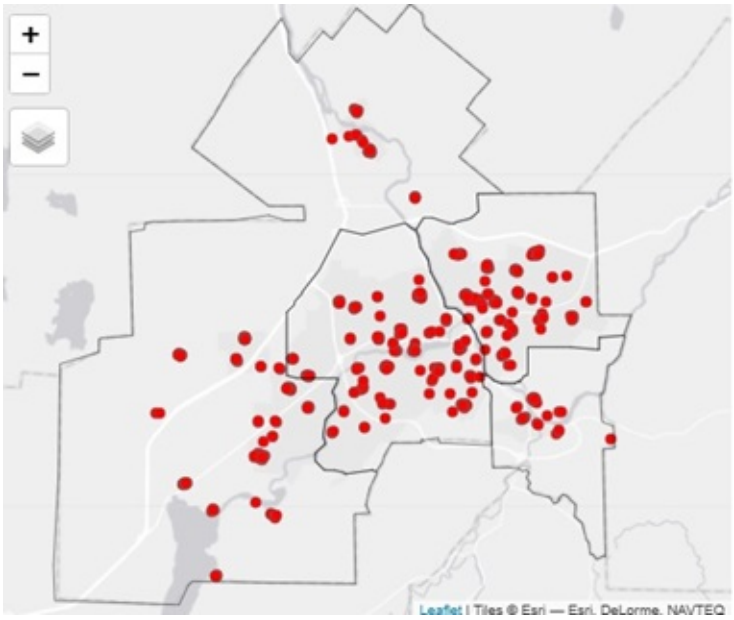
1.5.2 Cartes interactives

Avec la fonction `tmap_mode`, il est possible de choisir l'un des deux modes de visualisation suivants :

- statique avec `tmap_mode("plot")`.
- interactif avec `tmap_mode("view")`.

Vous constaterez ci-dessous que par défaut, trois fonds de carte sont disponibles dans la carte interactive, soit dans l'ordre `Esri.WorldGrayCanvas`, `OpenStreetMap` et `Esri.WorldTopoMap`.

```
## Mode active tmap
tmap_mode("view")
## Importation des couches
Arrond.sf = read_sf("data/chap01/shp/Arrondissements.shp")
InstallSR.sf = read_sf("data/chap01/shp/Installations_sportives_et_recreatives.shp")
## Carte
tm_shape(InstallSR.sf)+ tm_dots(size = 0.05, shape = 21, col = "red")+
tm_shape(Arrond.sf)+ tm_borders(col="black", lwd= .5)
```



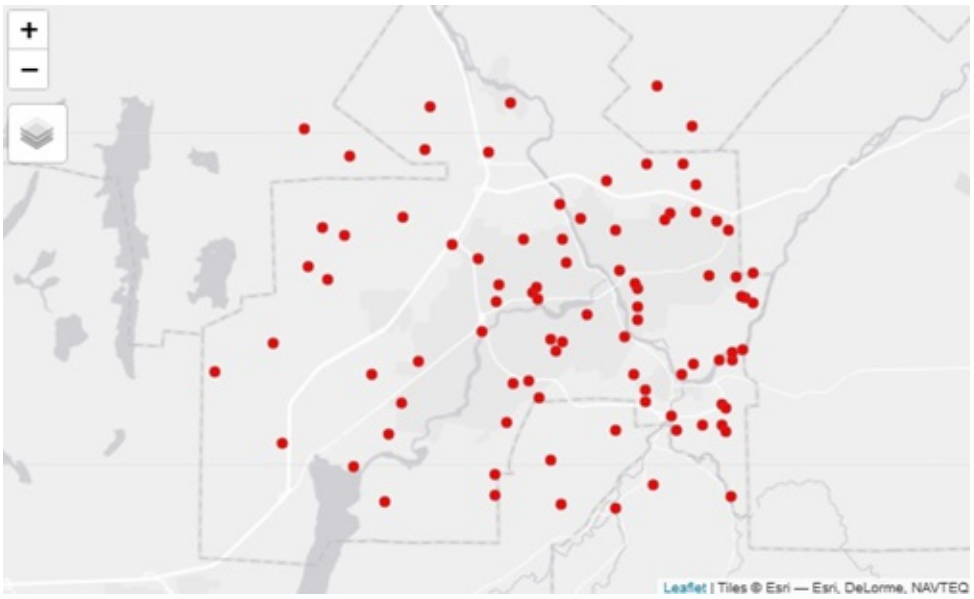
Il est aussi possible de changer les fonds de carte avec la fonction `tm_basemap` tandis que la fonction `tm_tiles` permet de superposer une tuile (pour la toponymie par exemple) (tableau 1.3).

Tableau 1.3 – Fonctions pour des cartes interactives

Fonction	Description
`tmap_mode`	Choisir le mode statique ou interactive
`tm_basemap`	Spécifier un fond de carte
`tm_tiles`	Spécifier une tuile de fond

Dans le code ci-dessous, nous utilisons uniquement deux fonds de carte. Remarquez les lignes avec l'argument `popup.vars` qui permet de définir les champs visibles dans la fenêtre surgissante (*pop-up*). Cliquez sur une installation sportive pour activer la fenêtre surgissante.

```
## Carte
tm_basemap(c("OpenStreetMap", "Esri.WorldTopoMap"))+
tm_shape(InstallSR.sf)+
  tm_dots(size = 0.05, shape = 21, col = "red",
          # définition pour le pop-up (clic sur une installation)
          popup.vars=c("Nom : "="NOM",
                       "Type : " = "TYPE",
                       "Éclairage : " = "ECLAIRAGE",
                       "Éclairage : " = "SURFACE"),
          id = "OBJECTID")+
tm_shape(Arrond.sf)+ tm_borders(col="black", lwd= .5)
```



Astuce

Où trouver des fonds de carte?
Une liste des fonds de carte **Leaflet** est disponible au lien suivant.

1.5.3 Mise en page d'une carte

Les principales fonctions de mise en page d'une carte sont présentées au tableau 1.4.

Tableau 1.4 – Fonctions d'habillage d'une carte

Fonction	Description	Principaux arguments
`tm_facets`	Créer un élément `tmap` avec plusieurs vignettes	`by`: groupé par colonne. `nrow` et `ncol`: nombres de lignes et de colonnes
`tmap_arrange`	Fusionner plusieurs cartes dans une mise en page	`nrow` et `ncol`: nombre de lignes et de colonnes
`tm_grid`	Ajouter une grille de lignes de coordonnées (ex. long/lat)	`x` et `y`: vecteurs pour les coordonnées
`tm_credits`	Créer un texte pour spéficier l'auteur.e ou la source de la carte	`text`: texte. `size`: taille du texte. `fontfamily`: police du texte
`tm_scale_bar`	Créer une échelle	`break`: vecteur numérique pour l'échelle. `position`: position de l'échelle avec les valeurs `left`, `center`, `right`, `bottom`, `top`. Par exemple c('left', 'bottom')
`tm_compass`	Créer une flèche du nord	`type`: type de flèche du Nord ('arrow', '4star', '8star', 'radar', 'rose')
`tm_logo`	Ajouter un logo à une carte	`file`: chemin et nom du fichier ou URL
`tm_xlab`	Ajouter un titre sur l'axe des X de la carte	`text`: nom de l'axe
`tm_ylab`	Ajouter un titre sur l'axe des Y de la carte	`text`: nom de l'axe
`tm_layout`	Spécifier des éléments de mise en page de la carte	`title`: titre de la carte
`tm_legend`	Paramétrer la légende de la carte	`position`: position de la légende avec les valeurs `left`, `center`, `right`, `bottom`, `top`
`tmap_options`	Paramétrer et conserver plusieurs options sur la carte	`unit`: unités de mesures ('imperial', 'km', 'm', 'mi', and 'ft')

1.5.3.1 Combinaison de plusieurs cartes

Tel que décrit dans le tableau 1.4, il existe deux fonctions pour combiner deux cartes : `tmap_arrange` et `tm_facets`.

Pour ceux et celles réalisant régulièrement des graphiques dans R avec `ggplot2`, `tmap_arrange` est très similaire à la fonction `ggarrange` du *package* `ggpubr` qui permet de fusionner plusieurs graphiques. Globalement, le principe est le suivant : vous réalisez deux cartes ou plus que vous combinez dans une même sortie avec `tmap_arrange`. Vous trouverez ci-dessous un exemple avec deux cartes (figure 1.30).

```
tmap_mode("plot")
## Carte 1
Carte1 =  tm_shape(SDR.Estrie)+
            tm_polygons(col="whitesmoke", border.col = "grey30", lwd = 1)+
            tm_bubbles(size = "SDRpop_2021",
                       border.col = "black",
                       col = "tomato1",
                       title.size = "Population",
                       scale = 3)+ # facteur multiplicateur pour la taille du cercle
```

```
          tm_shape(DR.Estrie)+ tm_borders(col="black", lwd = 2)
## Calcul de la densité de population
SDR.Estrie$HabKm2 <- as.numeric(SDR.Estrie$SDRpop_2021 / (st_area(SDR.Estrie) / 1000000))
## Carte 2
Carte2 =  tm_shape(SDR.Estrie)+
              tm_fill(col= "HabKm2",
                    palette = "Reds",
                    style = "quantile", n = 4,
                    title ="Hab./km2",
                    legend.format = list(text.separator = "à"))+
              tm_borders(col="black")+
          tm_shape(DR.Estrie)+ tm_borders(col="black", lwd = 2)
## Combinaison des deux cartes
tmap_arrange(Carte1, Carte2, ncol = 2, nrow = 1)
```

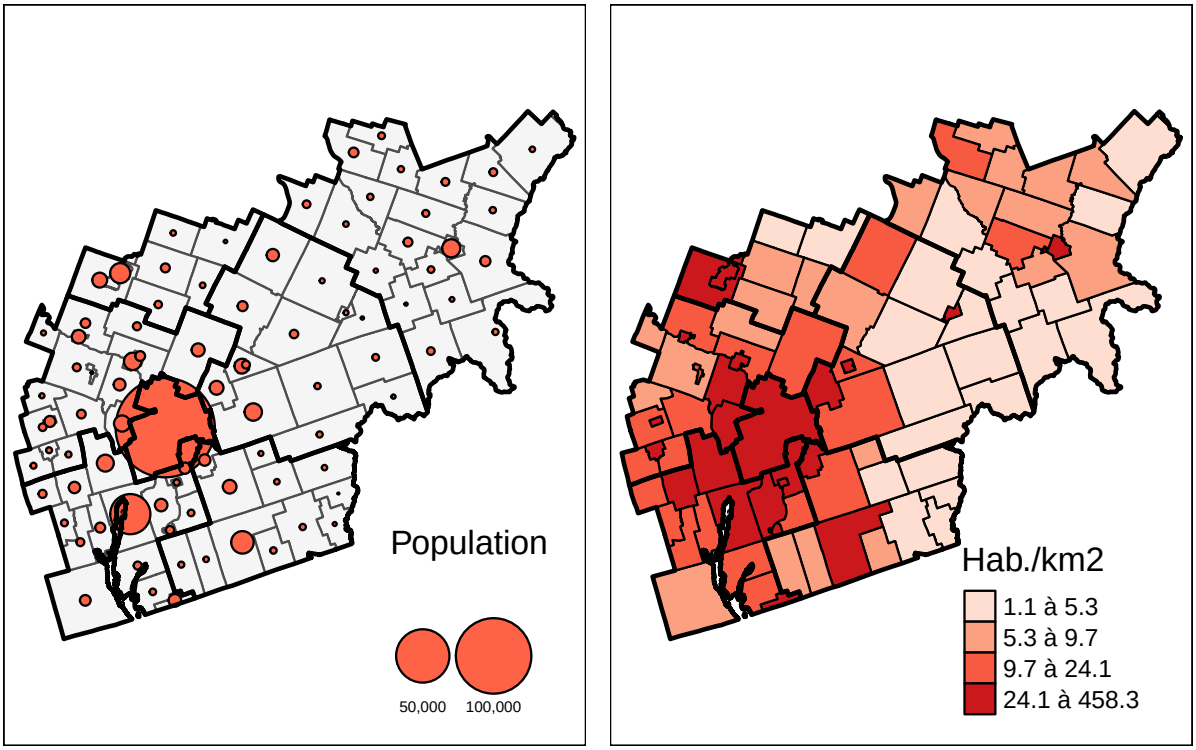


FIGURE 1.30 – Exemple de combinaisons de carte avec `tmap_arrange`

Quant à la fonction `tm_facets`, elle permet de créer plusieurs cartes avec l'argument `by`. Prenons un exemple concret : vous disposez d'une couche géographique des municipalités du Québec et vous souhaitez réaliser une carte pour chaque région administrative. L'argument `by = "Region"` vous permet alors d'avoir une vignette par région. Dans l'exemple ci-dessous, nous avons cartographié la même variable (densité de population) pour différentes zones de la région métropolitaine de Sherbrooke (figure 1.31).

```r
tmap_mode("plot")
## Création d'une variable zone basée sur les noms des municipalités
AD2021$Zone <- ifelse(AD2021$SDRNOM == "Sherbrooke", "A. Sherbrooke", "")
AD2021$Zone <- ifelse(AD2021$SDRNOM %in% c("Compton", "Waterville", "Hatley", "North Hatley"),
                      "B. Sud", AD2021$Zone)
AD2021$Zone <- ifelse(AD2021$SDRNOM %in% c("Orford", "Magog", "Saint-Denis-de-Brompton"),
                      "C. Est", AD2021$Zone)
AD2021$Zone <- ifelse(AD2021$SDRNOM %in% c("Ascot Corner", "Val-Joli", "Stoke"),
                      "C. Nord", AD2021$Zone)
## Création des cartes avec tm_facets
tmap_mode("plot")
tm_shape(AD2021)+
  tm_fill(col= "HabKm2",
          palette = "Reds",
          n = 5, # nombre de classes
          style = "quantile",
          title ="Hab./km2",
          legend.format = list(text.separator = "à"))+
  tm_borders(col="black", lwd = .5)+
  tm_facets(by = "Zone")
```

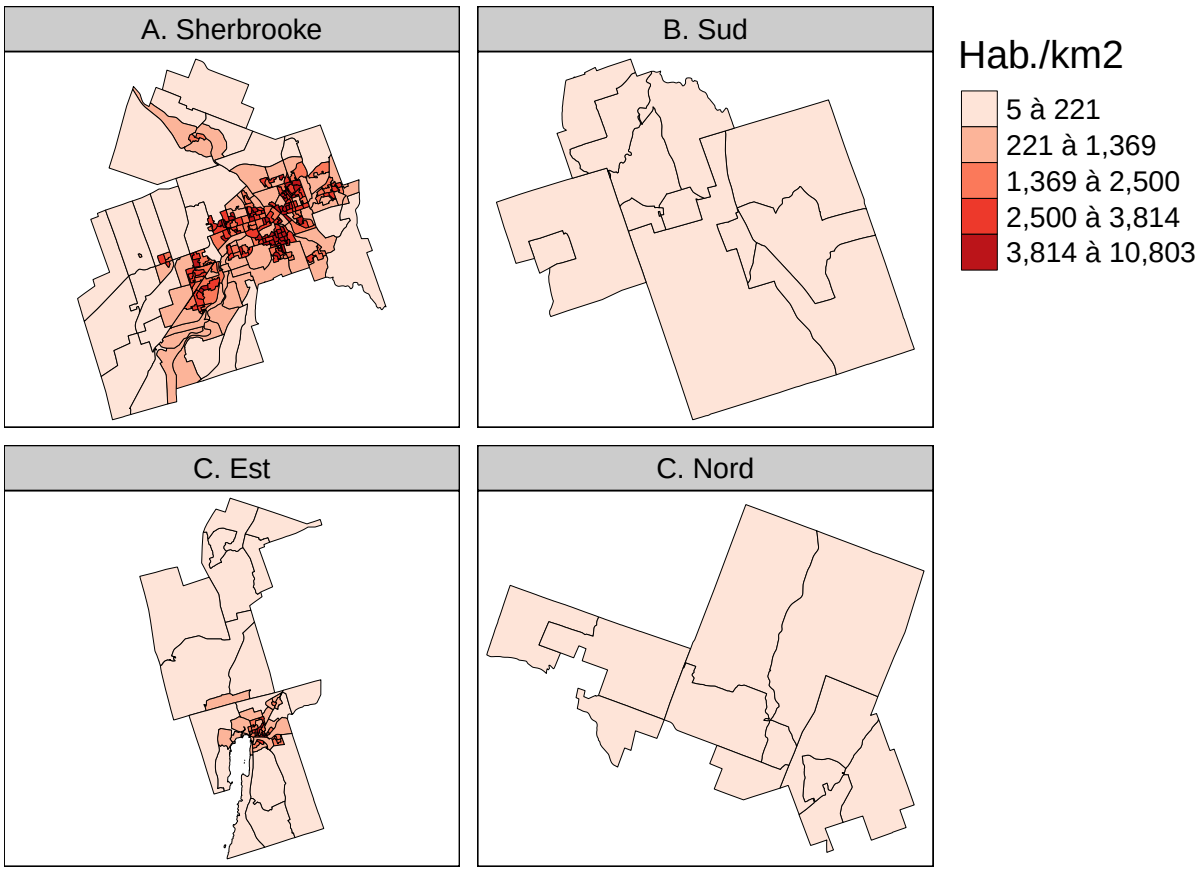


FIGURE 1.31 – Premier exemple de combinaison de cartes avec `tm_facets`

L'utilisation de `tm_facets` peut être également très utile pour comparer les distributions spatiales de points à différentes années (figure 1.32).

```
tmap_mode("plot")
## Importation des incidents
Incidents <- st_read("data/chap01/shp/IncidentsSecuritePublique.shp", quiet = TRUE)
## Création des cartes avec tm_facets
tmap_mode("plot")
tm_shape(Arrondissements) +
  tm_polygons(col="wheat", border.col = "black") +
tm_shape(Incidents) +
  tm_dots(shape=21, col="blue", size=.2) +
tm_facets(by = "ANNEE")
```

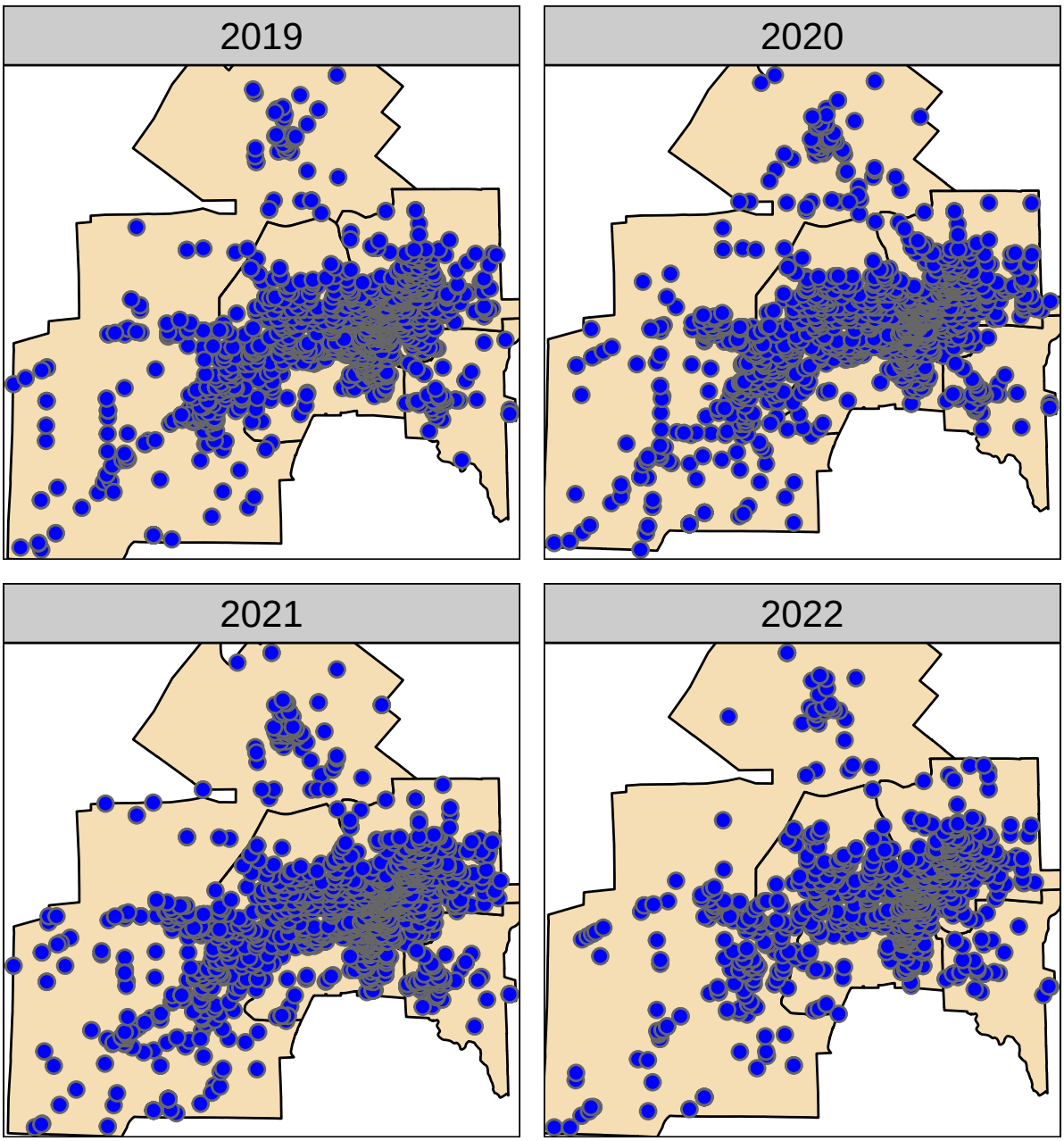


FIGURE 1.32 – Deuxième exemple de combinaisons de carte avec `tm_facets`

1.5.3.2 Mise en page d'une carte

Nous reprenons la figure 1.33 et l'habillons en ajoutant une échelle (`tm_scale_bar`), une flèche du Nord (`tm_compass`), la source et l'auteur (`tm_credits`) et un titre (`tm_layout`) (figure 1.33).

```
## Carte 1
tmap_mode("plot")
tm_shape(SDR.Estrie)+
```

```
            tm_fill(col= "HabKm2", palette = "Greens",
                    style = "quantile", n = 4,
                    title ="Hab./km2",
                    legend.format = list(text.separator = "à"))+
            tm_bubbles(size = "SDRpop_2021", border.col = "black", col = "tomato1", scale = 3,
                       title.size = "Population")+
            tm_borders(col="black")+
## Ajout de de la flèche du Nord
tm_compass(position = c("right", "bottom"),
           size = 2)+
## Ajout de l'échelle
tm_scale_bar(breaks  = c(0, 25, 50),
             position = c("right", "bottom"))+
## Ajout de la source
tm_credits("Source : recensement de 2021, Statistique Canada\nAuteur : Jéremy Lacartemplace.",
           position = c("right", "bottom"),
           size = 0.7,
           align = "right") +
## Légende
tm_legend(position = c("left", "top"),
          frame = FALSE, bg.color = "white")+
## Modification de la mise en page
tm_layout(main.title = "Municipalités de l'Estrie",
          legend.outside = TRUE,
          frame = FALSE)
```

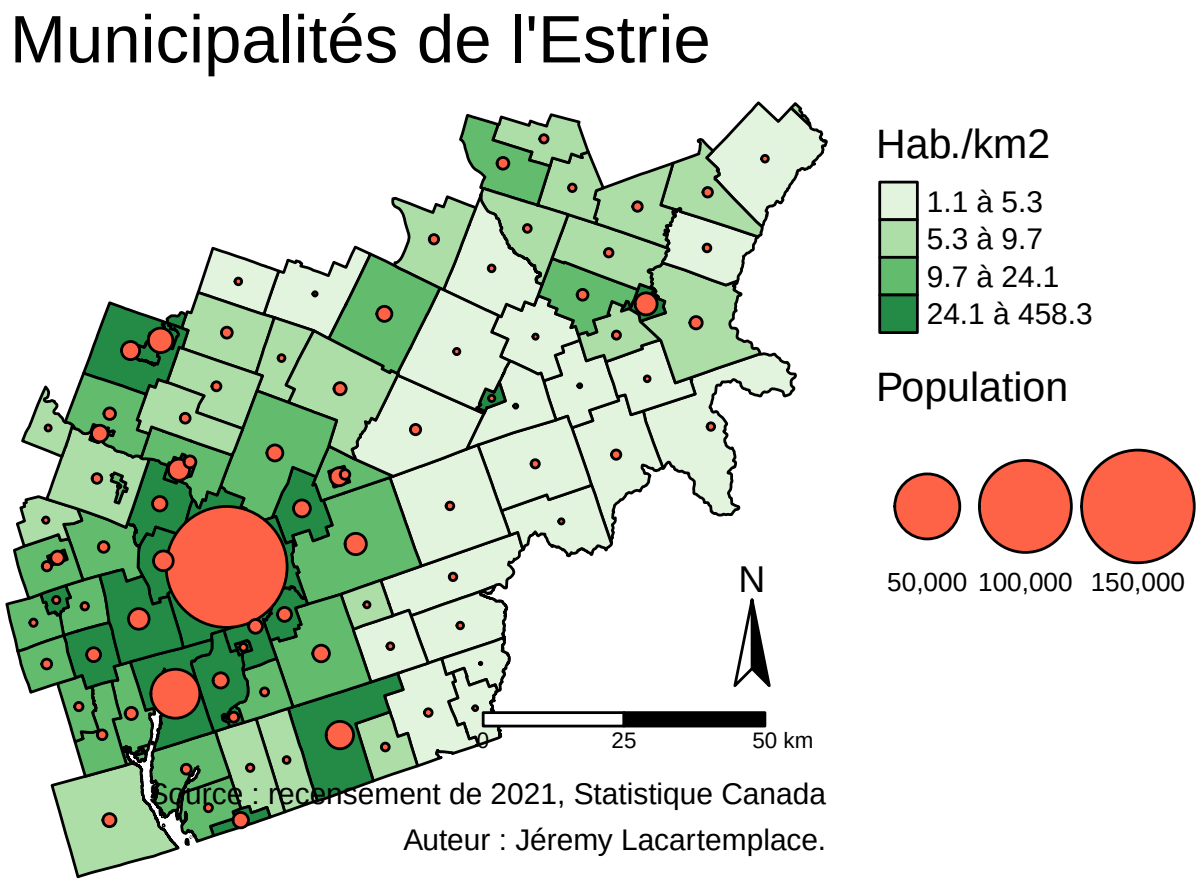


FIGURE 1.33 – Habillage d'une carte

Aller plus loin

Aller plus loin avec `tmap`?
Nous avons abordé uniquement les principales fonctions et arguments pour l'habillage d'une carte. Plusieurs exemples de très belles cartes créées avec `tmap` sont disponibles aux ressources suivantes :

- L'excellente vignette intitulée *`tmap`: get started!*
- *Visualizing Spatial Data in R with tmap*.
- *Making Maps with R*.
- Le chapitre *Making maps with R* du livre *Geocomputation with R*.

1.5.4 Exportation d'une carte

Une fois la carte finalisée, il est possible de l'exporter dans différents formats avec la fonction `tmap_save` :

- En mode image (png, jpg, bmp, tiff) pour l'insérer dans un logiciel de traitement de texte (Word ou OpenOffice Writer) ou dans un éditeur LaTeX (Overleaf par exemple).
- En mode vectoriel (PDF ou SVG) pour finaliser l'édition de la carte dans un logiciel de création graphique vectorielle (Illustrator par exemple).

— En HTML dans lequel la carte sera intégrée selon le mode de visualisation interactive, sous la forme d'un *widget* Leaflet.

```
## Transformation en long/lat
## Carte 1
tmap_mode("plot")
Carte1 <- tm_shape(SDR.Estrie)+
  tm_fill(col= "HabKm2", palette = "Greens", style = "quantile", n = 4, title ="Hab./km2")+
  tm_bubbles(size = "SDRpop_2021", border.col = "black", col = "tomato1", scale = 3,
                       title.size = "Population")+
  tm_borders(col="black")+
  tm_compass(position = c("right", "bottom"), size = 2)+
  tm_scale_bar(breaks  = c(0, 25, 50), position = c("right", "bottom"))+
  tm_credits("Source : recensement de 2021, Statistique Canada\nAuteur : Jéremy Lacartemplace.",
           position = c("right", "bottom"), size = 0.7, align = "right") +
  tm_legend(position = c("left", "top"), frame = FALSE, bg.color = "white")+
  tm_layout(main.title = "Municipalités de l'Estrie", legend.outside = TRUE, frame = FALSE)

## Exportation de la Carte1 au format png
tmap_save(Carte1, filename = "data/chap01/export/Carte1.png", dpi = 600)
## Exportation de la Carte1 au format PDF
tmap_save(Carte1, filename = "data/chap01/export/Carte1.pdf")
## Exportation de la Carte1 au format HTML
tmap_save(Carte1, filename = "data/chap01/export/Carte1.html")
```

1.6 Quiz de révision du chapitre

Questions

— **La classe sf est composée de trois éléments :**

  — simple feature geometry (sfg) : géométrie d'une observation
  — simple feature column (sfc) : liste toutes les géométries d'une couche
  — data.fame : données attributaires
  — raster : données images

  Relisez au besoin la section 1.1.1.4.

— **Laquelle de ces fonctions permet de changer la projection cartographique d'une couche géographique?**

  — st_crs(x)
  — st_transform(x, crs)
  — st_is_longlat(x)

  Relisez au besoin le début de la section 1.2.1.

— **Laquelle de ces fonctions n'est pas une fonction géométrique sur une couche?**

  — st_bbox(x)
  — st_union(x)

  - st_point_on_surface(x)
  - st_crop(x, y, xmin, ymin, xmax, ymax)
  - st_centroid(x)

  Relisez au besoin la section 1.2.2.
- **Comparativement à l'algorithme de Douglas et Peucker, que permet l'algorithme de Visvalingam lors de la simplification des contours?**

  - Il est plus rapide.
  - Il permet de conserver les frontières.

  Relisez au besoin la section 1.2.2.4.
- **Tous les points sont compris dans leur enveloppe convexe.**

  - Vrai
  - Faux

  Relisez au besoin la section 1.2.2.5.
- **Quelles sont les quatre fonctions de mesures géométriques et de récupération des coordonnées géographiques?**

  - st_area(x)
  - st_length(x)
  - st_distance(x,y)
  - st_coordinates(x)
  - st_union(x)

  Relisez au besoin la section 1.2.4.
- **Quelle est la différence entre les fonctions st_intersects(x, y) et st_intersection(x, y)?**

  - Elles génèrent le même résultat.
  - La première est une requête spatiale, la seconde renvoie l'intersection entre deux couches.
  - La première renvoie l'intersection entre deux couches, la seconde est une requête spatiale.

  Relisez le deuxième encadré à la section 1.2.6.
- **Laquelle des fonctions sf permet d'exporter des données vectorielles?**

  - st_read()
  - st_write()
  - writeRaster()

  Relisez au besoin la section 1.4.1.

Réponses

- La classe sf est composée de trois éléments :
  - simple feature geometry (sfg) : géométrie d'une observation
  - simple feature column (sfc) : liste toutes les géométries d'une couche
  - data.fame : données attributaires
- Laquelle de ces fonctions permet de changer la projection cartographique d'une couche géographique?
  - st_transform(x, crs)
- Laquelle de ces fonctions n'est pas une fonction géométrique sur une couche?
  - st_crop(x, y, xmin, ymin, xmax, ymax)

- Comparativement à l'algorithme de Douglas et Peucker, que permet l'algorithme de Visvalingam lors de la simplification des contours?
  - Il permet de conserver les frontières.
- Tous les points sont compris dans leur enveloppe convexe.
  - Vrai
- Quelles sont les quatre fonctions de mesures géométriques et de récupération des coordonnées géographiques?
  - st_area(x)
  - st_length(x)
  - st_distance(x,y)
  - st_coordinates(x)
- Quelle est la différence entre les fonctions st_intersects(x, y) et st_intersection(x, y)?
  - La première est une requête spatiale, la seconde renvoie l'intersection entre deux couches.
- Laquelle des fonctions sf permet d'exporter des données vectorielles?
  - st_write()

1.7 Exercices de révision

Exercice

Exercice 1. Découpage des rues de l'arrondissement des Nations de la ville de Sherbrooke
Complétez le code ci-dessous avec les étapes suivantes :

- Requête attributaire pour créer un objet `sf` avec uniquement l'arrondissement des Nations à partir de la couche `Arrondissements` et le champ `NOM` (voir la section 1.2.7.4).
- Découpez les rues (`Rues`) sur le nouvel objet `sf` (voir la section 1.2.3).

```
library(sf)
## Importation des deux couches
Arrond <- st_read("data/chap01/shp/Arrondissements.shp", quiet = TRUE)
Rues <- st_read("data/chap01/shp/Segments_de_rue.shp", quiet = TRUE)
## Requête attributaire : création d'un objet sf pour l'arrondissement des Nations
table(Arrond$NOM)
Arrond.DesNations <- subset(À compléter)
## Découper les rues avec le polygone de l'arrondissement des Nations
Rues.DesNations <- À compléter
```

Correction à la section 12.1.1.

Exercice

Exercice 2. Calcul d'un nouveau champ
Calculez un nouveau champ (`DistHVKM`) dans la couche des aires de diffusion (AD) (`AD.RMRSherb`) qui représente la distance en kilomètres entre l'hôtel de ville de Sherbrooke et les points des AD. Puis, cartographiez le champ `DistHVKM` en quatre classes selon la méthode de discrétisation par quantiles. Complétez le code ci-dessous avec les étapes suivantes :

- Ajoutez un champ pour la distance (`DistHVKM`) dans la couche `AD.RMRSherb` (voir la section 1.2.4).
- Cartographiez le champ `DistHVKM` en quatre classes selon la méthode des quantiles (voir la section 1.5.1.5).

```
library(sf)
library(tmap)
## Importation des deux couches
AD.RMRSherb <- st_read(dsn = "data/chap01/gpkg/Recen2021Sherbrooke.gpkg",
                       layer = "SherbAD", quiet = TRUE)
HotelVille <- data.frame(ID = 1, Nom = "Hôtel de ville",
                         lon = -71.89306, lat = 45.40417)
HotelVille <- st_as_sf(HotelVille, coords = c("lon","lat"), crs = 4326)
## Changement de projection avant de s'assurer que les deux couches ont la même
HotelVille <- st_transform(HotelVille, st_crs(AD.RMRSherb))
## Ajout d'un champ pour la distance en km à l'hôtel de ville pour les secteurs de recensement
AD.RMRSherb$DistHVKM <- À compléter
## Cartographie en quatre classes selon les quantiles
tmap_mode("plot")
tm_shape(À compléter)+
  tm_fill(À compléter)+
  tm_borders(col="black")
```

Correction à la section 12.1.2.

Exercice

Exercice 3. Importation d'une couche *shapefile*
Importez une couche *shapefile* pour les divisions de recensement et calculez la densité de population (nombre d'habitants au km^2). Complétez le code ci-dessous avec les étapes suivantes :

- Faites une jointure attributaire entre la couche `DR.Qc` et la table `DR.Data` (voir la section 1.2.7.2).
- Calculez le champ `HabKm2`, soit la division entre les champs `DRpop_2021` et `SUPTERRE` (voir la section 1.5.1.5).

```
library(sf)
## Importation de la couche des divisions de recensement du Québec
DR.Qc <- st_read(dsn = "data/chap01/gpkg/Recen2021Sherbrooke.gpkg",
                 layer = "DivisionsRecens2021", quiet = TRUE)
## Importation du fichier csv des divisions de recensement
DR.Data <- read.csv("data/chap01/tables/DRQC2021.csv")
## Jointure attributaire avec le champ IDUGD
DR.Qc <- A compléter
## Il y a déjà deux champs dans la table pour calculer la densité de population :
## SUPTERRE : superficie en km2
## DRpop_2021 : population en 2021
DR.Qc$HabKm2 <- A compléter
head(DR.Qc, n=2)
summary(DR.Qc$HabKm2)
```

Correction à la section 12.1.3.

Exercice

Exercice 4. Coordonnées géographiques
Vous recevez les coordonnées en degrés (WGS84, EPSG : 4326) : -71.91688, 45.37579. Créez un point pour cette localisation et calculez la distance la séparant du tronçon autoroutier le plus proche. Complétez le code ci-dessous avec les étapes suivantes :

- Faites une requête attributaire pour créer un objet `sf` avec uniquement les tronçons autoroutiers à partir de la couche `Rues` et le champ `TYPESEGMEN` (voir la section 1.2.7.4).
- Trouvez l'identifiant du tronçon le plus proche avec la fonction `st_nearest_feature` (voir la section 1.2.6).

```
library(sf)
## Importation du réseau de rues
Rues <- st_read("data/chap01/shp/Segments_de_rue.shp", quiet=TRUE)
unique(Rues$TYPESEGMEN)
## Sélection des tronçons autoroutiers
Autoroutes <- À compléter
## Création d'une couche sf pour le point avec les coordonnées
## en degrés (WGS84, EPSG : 4326) : -71.91688, 45.37579
Point1_sf <- À compléter
## Changement de projection avant de s'assurer que les deux couches ont la même
Point1_sf <- st_transform(Point1_sf, st_crs(Autoroutes))
## Trouver le tronçon autoroutier le plus proche avec la fonction st_nearest_feature
PlusProche <- À compléter
print(PlusProche)
Point1_sf$AutoroutePlusProche <- as.numeric(st_distance(Point1_sf,
                                                        Autoroutes[PlusProche,]))
cat("Distance à l'autoroute la plus proche :", Point1_sf$AutoroutePlusProche, "m.")
## Zone tampon
ZoneTampon <- st_buffer(Point1_sf, Point1_sf$AutoroutePlusProche)
## Cartographie
tmap_mode("view")
tm_shape(ZoneTampon)+
  tm_borders(col= "black")+
tm_shape(Autoroutes)+
  tm_lines(col="red")+
tm_shape(Point1_sf)+
  tm_dots(col= "blue", shape=21, size = .2)
```

Correction à la section 12.1.4.

Partie 2. Autocorrélation spatiale

2 Autocorrélation spatiale

Note

Première loi de la géographie proposée par Waldo Tobler
« Tout interagit avec tout, mais les objets proches ont plus de chance de le faire que les objets éloignés [*Everything is related to everything else, but near things are more related than distant things*] » (Tobler 1970).

Dans ce chapitre, nous mettons en œuvre dans R différentes méthodes qui permettent d'évaluer la dépendance spatiale d'une variable, soit les mesures d'autocorrélation spatiale globale et locale. Préalablement, nous voyons comment définir des matrices de pondération spatiale – selon la contiguïté et la proximité spatiale – qui sont utilisées dans les mesures d'autocorrélation spatiale, mais aussi dans les modèles spatiaux autorégressifs (chapitre 7).

Package

Liste des *packages* utilisés dans ce chapitre
- Pour importer et manipuler des fichiers géographiques :
  - `sf` pour importer et manipuler des données vectorielles.
  - `terra` pour importer et manipuler des données matricielles.
- Pour calculer des mesures d'autocorrélation spatiale :
  - `spdep` pour construire des matrices spatiales et calculer des mesures d'autocorrélation spatiale.
  - `rgeoda` pour calculer des mesures d'autocorrélation spatiale.
  - `geocmeans` pour calculer l'indicateur ELSA.
- Pour construire des cartes et des graphiques :
  - `tmap` pour construire des cartes thématiques.
  - `ggplot2` pour construire des graphiques.
  - `ggpubr` pour réaliser une figure combinant plusieurs graphiques construits avec `ggplot2`.
- Pour manipuler des données :
  - `dplyr` est un *package* facilitant la manipulation des données.

2.1 Notion d'autocorrélation spatiale

Comprendre la configuration spatiale d'un phénomène donné est une démarche fondamentale en analyse spatiale. Or, l'autocorrélation spatiale permet d'estimer la corrélation d'une variable par rapport à sa localisation dans l'espace, soit la dépendance spatiale. Autrement dit, elle permet de vérifier si les entités proches ou voisines ont tendance à être (dis)semblables en fonction d'un phénomène donné (soit une variable). Tel qu'illustré à la figure 2.1 (données fictives), on distingue trois formes d'autocorrélation spatiale :

— **(a) Autocorrélation spatiale positive** : lorsque les entités spatiales voisines ou proches se ressemblent davantage que celles non contiguës ou éloignées. Cela renvoie ainsi à la première loi de la géographie : « tout interagit avec tout, mais les objets proches ont plus de chance de le faire que les objets éloignés » (traduction libre) (Tobler 1970).
— **(b) Autocorrélation spatiale négative** : lorsque les entités spatiales voisines ou proches ont tendance à être dissemblables, comparativement à celles non contiguës ou éloignées.
— **(c) Absence d'autocorrélation spatiale** : lorsque les valeurs de la variable sont distribuées aléatoirement dans l'espace; autrement dit, lorsqu'il n'y a pas de relation entre le voisinage ou la proximité des entités spatiales et leur degré de ressemblance.

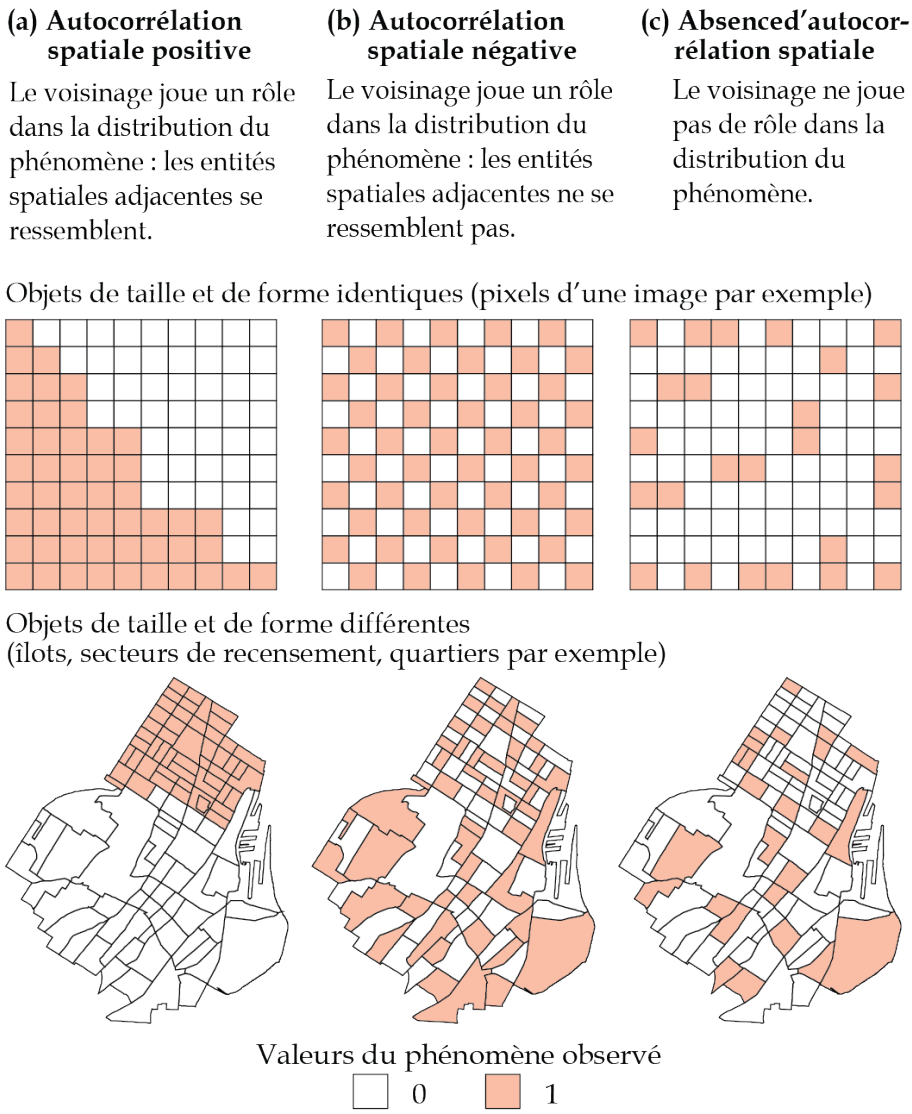


FIGURE 2.1 – Autocorrélation spatiale

Exercice

Analyse de la figure 2.2
Quelle est la variable pour laquelle le voisinage joue un rôle important dans sa distribution? L'autocorrélation pour cette variable est-elle positive ou négative? Pourquoi?

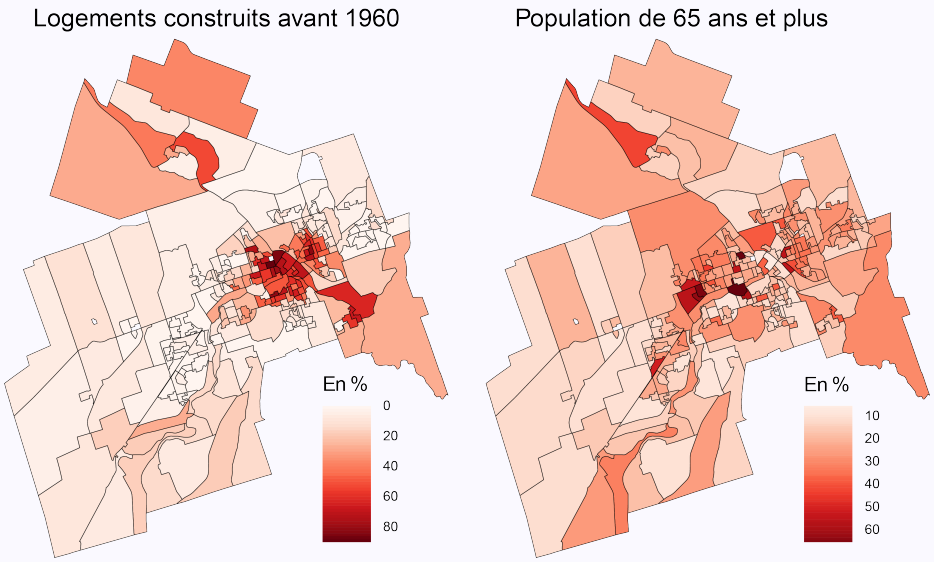


FIGURE 2.2 – Illustration de l'autocorrélation spatiale de deux variables pour les aires de diffusion de la ville de Sherbrooke

Réponse : L'autocorrélation spatiale semble bien plus forte pour le pourcentage des logements construits avant 1960. Les aires de diffusion (AD) contiguës ou proches dans la partie centrale de la ville ont clairement des pourcentages élevés (rouge foncé) tandis que celles voisines ou proches dans les périphéries présentent des pourcentages faibles. Cela traduit donc une forte autocorrélation spatiale positive. Par contre, la distribution spatiale du pourcentage de personnes de 65 ans et plus semble plus aléatoire, traduisant ainsi une faible autocorrélation spatiale (dépendance spatiale).

Vous avez compris que la simple cartographie d'une variable vous donne une indication de l'autocorrélation spatiale. Pour contre, pour « chiffrer » l'intensité de l'autocorrélation spatiale, il convient de : 1) choisir une matrice de pondération spatiale (selon le voisinage ou la distance) (section 2.2), 2) calculer une mesure d'autocorrélation spatiale à partir de cette matrice (comme l'indice de Moran) (section 2.3).

2.2 Matrices de pondération spatiale

Les mesures d'autocorrélation spatiale visent à vérifier si les entités spatiales contiguës ou proches ont tendance à être semblables (autocorrélation positive) ou dissemblables (autocorrélation négative) en fonction d'un phénomène donné (en fonction d'une variable). Il convient donc avant tout de définir la manière de mesurer la relation d'adjacence ou de proximité entre deux entités spatiales.

Il existe huit principales matrices de pondération spatiale regroupées en deux grandes catégories : celles de contiguïté (basées sur l'adjacence) et celles de proximité (basées sur la distance) (tableau 2.1). Lorsque la couche géographique est

composée de points, seules les matrices de proximité peuvent être utilisées.

Tableau 2.1 – Matrices de pondération spatiale selon la géométrie

Matrice	Points	Lignes	Polyg.	Raster
Matrices de contiguïté (basées sur l'adjacence)				
Partage d'un nœud (Queen)		X	X	X
Partage d'un segment (Rook)		X	X	X
Partage d'un nœud et ordre d'adjacence (Queen)		X	X	X
Partage d'un segment et ordre d'adjacence (Rook)		X	X	X
Matrices de proximité (basées sur la distance)				
Connectivité selon la distance	X	X	X	X
Inverse de la distance	X	X	X	X
Inverse de la distance au carré	X	X	X	X
Nombre de plus proches voisins	X	X	X	X

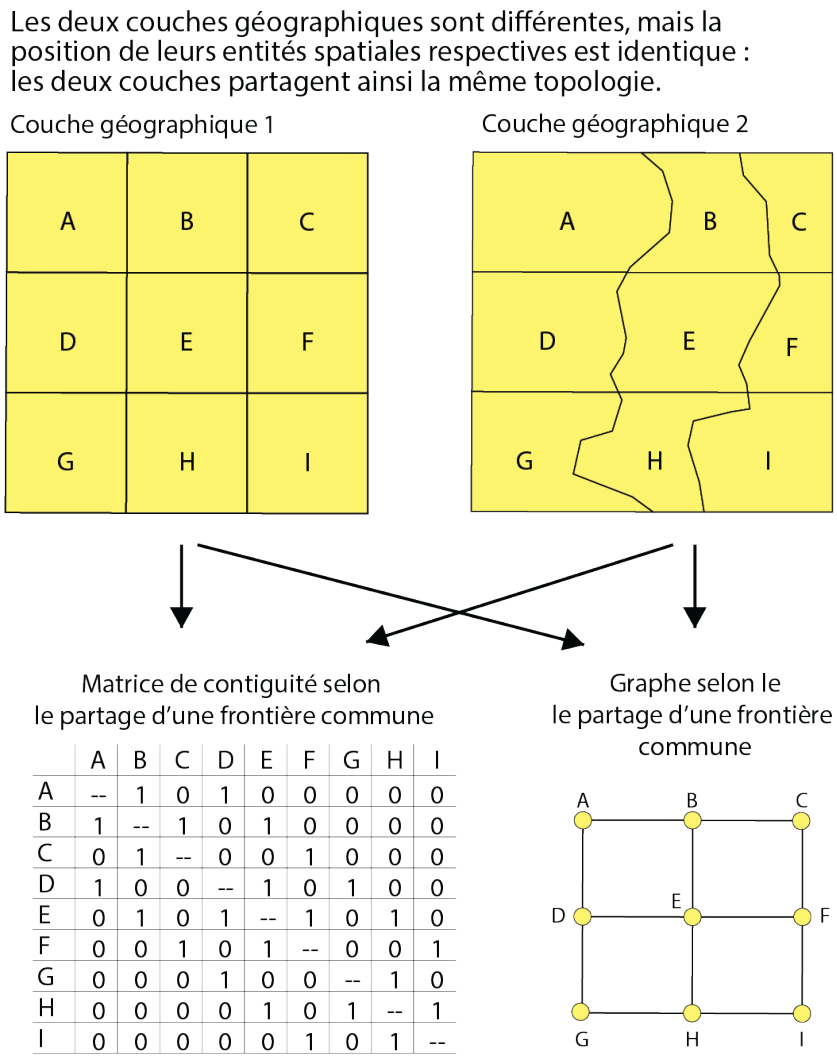
2.2.1 Matrices de contiguïté

La relation d'adjacence (de contiguïté) vise à déterminer si deux entités spatiales sont ou non voisines selon le partage soit d'un nœud, soit d'un segment (frontière commune). La contiguïté est liée à la notion de topologie qui prend en compte les relations de voisinage entre des entités spatiales, sans tenir compte de leurs tailles et de leurs formes géométriques. Elle peut être représentée à partir d'une **matrice de contiguïté** (avec une valeur de 1 quand deux entités sont voisines et de 0 pour une situation inverse) ou d'un **graphe** (formé de points représentant les entités spatiales et de lignes reliant les entités voisines) (figure 2.3).

Trois évaluations de la contiguïté sont représentées à la figure 2.4 :

- **Adjacence selon le partage d'un segment**, soit d'une frontière commune entre les polygones (A).
- **Adjacence selon le partage d'un nœud** (B).
- **Ordre d'adjacence selon le partage d'un segment** (C). L'ordre d'adjacence indique le nombre de frontières à traverser pour se rendre à l'entité spatiale contiguë, soit :
  - **Ordre 1** : une frontière à traverser pour se rendre dans l'entité spatiale adjacente.
  - **Ordre 2** : deux frontières à traverser pour atteindre les entités de la deuxième couronne.
  - **Ordre 3** : trois frontières à traverser pour atteindre les entités de la troisième couronne.
  - Etc.

Bien entendu, les ordres d'adjacence peuvent être également définis selon le partage d'un nœud commun.



	A	B	C	D	E	F	G	H	I
A	--	1	0	1	0	0	0	0	0
B	1	--	1	0	1	0	0	0	0
C	0	1	--	0	0	1	0	0	0
D	1	0	0	--	1	0	1	0	0
E	0	1	0	1	--	1	0	1	0
F	0	0	1	0	1	--	0	0	1
G	0	0	0	1	0	0	--	1	0
H	0	0	0	0	1	0	1	--	1
I	0	0	0	0	0	1	0	1	--

FIGURE 2.3 – Relation topologique entre des entités spatiales polygonales

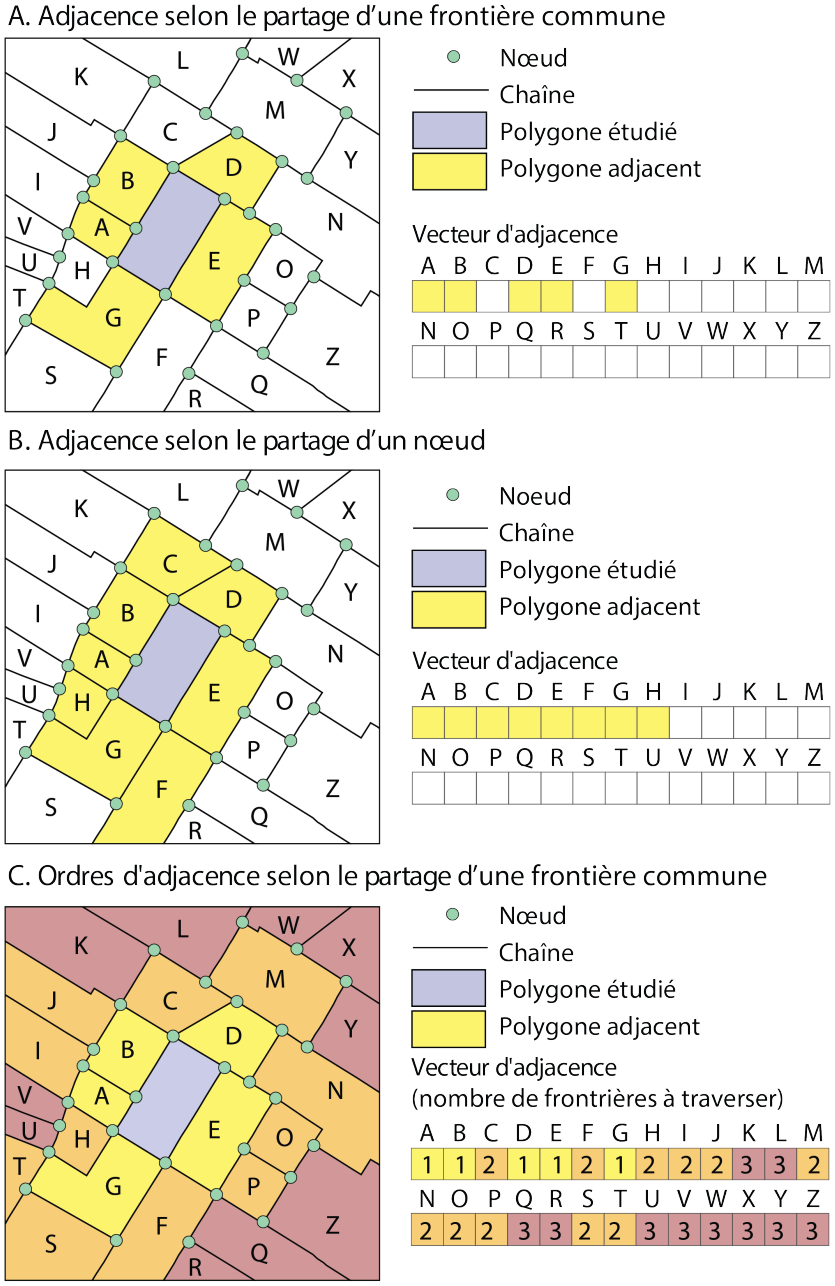


FIGURE 2.4 – Relations de voisinage et évaluation de la contiguïté

Astuce

Applicabilité des ordres d'adjacence
Les matrices d'adjacence sont souvent utilisées dans les analyses de diffusion spatiale. Prenons un exemple concret : imaginons que le polygone en gris à la figure 2.4 est un parc. Nous pourrions évaluer le prix moyen des maisons dans les îlots qui font face au parc (ordre 1), toutes choses étant égales par ailleurs (superficie du terrain, superficie habitable, nombre de pièces, etc.). Puis, nous pourrions comparer ce prix moyen à ceux calculés pour les ordres suivants. Il est probable que le prix au premier ordre soit significativement plus élevé qu'au deuxième ordre, voire au troisième ordre. Autre exemple, nous pourrions réaliser un exercice similaire pour des maisons dans des îlots adjacents à un tronçon autoroutier. La relation est probablement inverse : un prix moyen plus bas à l'ordre 1 comparativement aux ordres suivants.

Habituellement appelée W, la matrice de contiguïté est binaire selon le partage tant d'un nœud (*Queen* en anglais) (équation 2.1) que d'un segment commun (*Rook* en anglais) (équation 2.2).

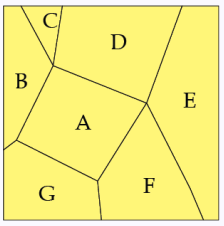
$$w_{ij} = \begin{cases} 1 & \text{si les entités spatiales } i \text{ et } j \text{ ont au moins un nœud commun; } i \neq j \\ 0 & \text{sinon} \end{cases} \tag{2.1}$$

$$w_{ij} = \begin{cases} 1 & \text{si les entités spatiales } i \text{ et } j \text{ partagent une frontière commune; } i \neq j \\ 0 & \text{sinon} \end{cases} \tag{2.2}$$

Exercice

Exercice 1. Compléter des matrices de contiguïté
Petit conseil pour la partie A : une matrice de contiguïté est symétrique c'est-à-dire que si le polygone A est voisin du polygone B, alors B est voisin de A! Par conséquent, pour gagner du temps, complétez une ligne et transposez-la en colonne.

A. Complétez les matrices de contiguité pour les entités spatiales ci-dessous.



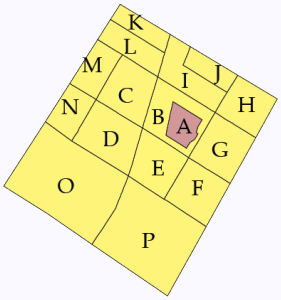
Matrice de contiguïté selon le partage d'un segment

	A	B	C	D	E	F	G
A	--						
B		--					
C			--				
D				--			
E					--		
F						--	
G							--

Matrice de contiguïté selon le partage d'un nœud

	A	B	C	D	E	F	G
A	--						
B		--					
C			--				
D				--			
E					--		
F						--	
G							--

B. Trouvez les polygones contigus aux différents ordres d'adjacence et selon le partage d'un segment pour le polygone A.



Polygones adjacents

Ordre 1 : .....

Ordre 2 : .....

Ordre 3 : .....

Ordre 4 : .....

FIGURE 2.5 – Exercice sur la contiguïté et les ordres d'adjacence

Correction à la section 12.2.1.

2.2.2 Matrices de proximité spatiale

2.2.2.1 Bref retour sur les différents types de distance

Pour calculer des mesures d'autocorrélation spatiale, nous pouvons aussi utiliser des matrices de pondération spatiale basées sur la proximité spatiale. Cette fois, nous ne cherchons pas à vérifier si les entités spatiales adjacentes se ressemblent, mais plutôt à vérifier si les entités spatiales proches les unes des autres se ressemblent. Pour ce faire, nous devons calculer les distances entre les entités spatiales.

Pour construire une matrice de pondération spatiale selon la proximité, nous pouvons utiliser plusieurs types de distance (Apparicio et al. 2017) : certaines sont cartésiennes, d'autres, dites réticulaires, sont calculées à partir d'un réseau de rues (figure 2.6).

Les distances cartésiennes – euclidienne et de Manhattan (équation 2.3 et équation 2.4) – sont facilement calculables à partir des coordonnées géographiques (*x*,*y*) dans un SIG ou dans n'importe quel logiciel tableur, de statistique ou de gestion de base de données, etc. Pour cela, la couche géographique doit être dans un système de projection plane. La distance euclidienne représente ainsi la distance à vol d'oiseau entre deux points, tandis que la distance de Manhattan est la somme des deux côtés formant l'angle droit d'un triangle rectangle (l'hypoténuse, le plus grand des côtés du triangle, étant la distance euclidienne) (figure 2.6, a). Si la projection de la couche est sphérique (longitude/latitude), il convient d'utiliser la formule de haversine (basée sur la trigonométrie sphérique) pour obtenir la distance à vol d'oiseau (équation 2.5).

Par contre, comme leurs noms l'indiquent, les distances réticulaires nécessitent un réseau de rues dans un SIG (notamment avec l'extension *Network Analyst* d'ArcGIS Pro) ou dans R (notamment avec le *package* R5R) pour calculer le chemin le plus rapide (chapitre 5).

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{2.3}$$

$$d_{ij} = |x_i - x_j| + |y_i - y_j| \tag{2.4}$$

$$d_{ij} = 2R \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\delta_i - \delta_j}{2}\right) + \cos\delta_i \cdot \cos\delta_j \cdot \sin^2\left(\frac{\phi_i - \phi_j}{2}\right)}\right) \tag{2.5}$$

avec R étant le rayon de la terre; δ_i et δ_j les coordonnées de longitude pour les points i et j; ϕ_i et ϕ_j les coordonnées de latitude pour les points i et j.

2.2.2.2 Matrice de distance binaire (de connectivité)

À partir d'une matrice de distance entre les entités spatiales d'une couche géographique, il est possible de créer une matrice de pondération binaire (équation 2.6). Ce type de matrice est habituellement appelée **matrice de connectivité**. Il convient alors de fixer un seuil de distance maximal. Par exemple, avec un seuil de 500 mètres, $w_{ij} = 1$ si la distance entre les entités spatiales i et j est inférieure ou égale à 500 mètres; sinon $w_{ij} = 0$. Notez que pour des lignes et des polygones, la distance est habituellement calculée à partir de leurs centroïdes.

$$w_{ij} = \begin{cases} 1 & \text{si } d_{ij} \leq \bar{d};\, i \neq j \\ 0 & \text{sinon} \end{cases} \tag{2.6}$$

avec d_{ij} étant la distance entre les entités spatiales i et j, et $\bar{d}$ étant un seuil de distance maximal fixé par la personne utilisatrice (par exemple, 500 mètres).

En guise d'exemple, à la figure 2.7, seuls les polygones jaunes seraient considérés comme voisins du polygone bleu avec un seuil de distance maximal fixé à 2,5 kilomètres (valeur de 1); les roses se verraient affecter la valeur de 0.

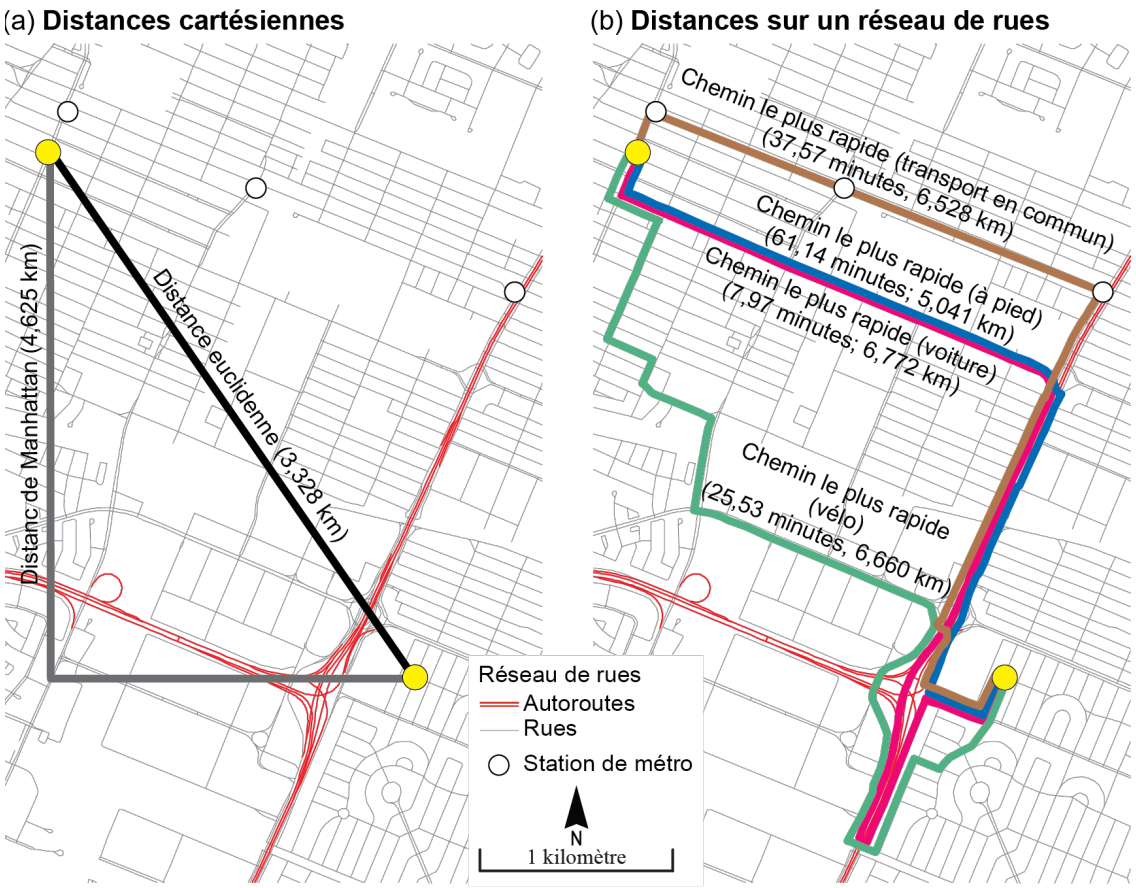


Figure adaptée de Apparicio et al. (2017).

FIGURE 2.6 – Les différents types de distance

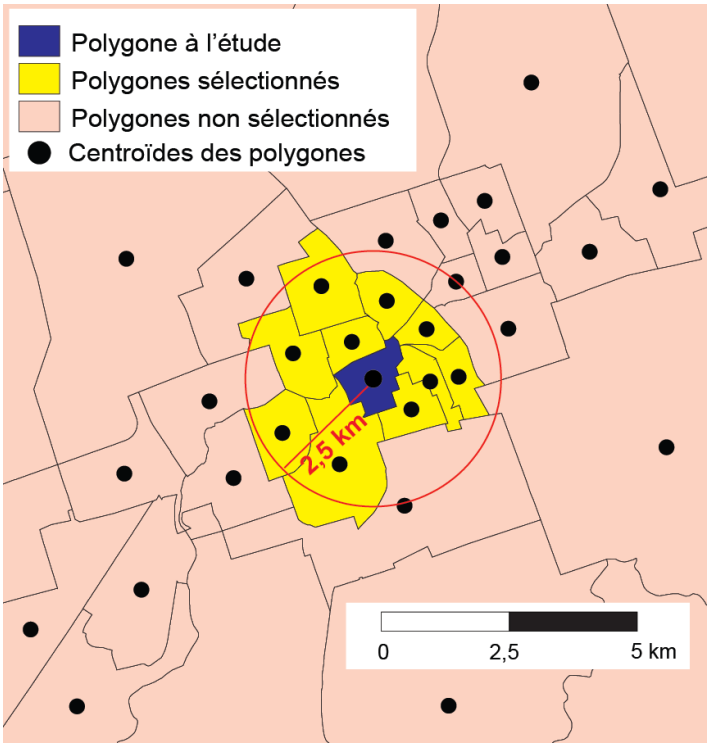


FIGURE 2.7 – Illustration de la connectivité basée sur la distance

2.2.2.3 Matrices basées sur la distance

À partir d'une matrice de distance entre les entités spatiales, les pondérations peuvent être calculées avec l'inverse de la distance ($1/d_{ij}$) ou l'inverse de la distance au carré ($1/d_{ij^2}$) (équation 2.7). Analysons le graphique à la figure 2.8.

- Premièrement, nous constatons que plus la distance est grande, plus la valeur de la pondération est faible et inversement. De la sorte, nous accordons un rôle plus important aux entités spatiales proches les unes des autres que celles éloignées.
- Deuxièmement, les pondérations chutent beaucoup plus rapidement avec l'inverse de la distance au carré qu'avec l'inverse de la distance. Autrement dit, le recours à une matrice de pondération calculée avec l'inverse de la distance au carré a comme effet d'accorder un poids plus important aux entités géographiques très proches.

$$w_{ij} = \begin{cases} \frac{1}{d_{ij}^{\gamma}} \\ 0 \qquad \text{si } i = j \end{cases} \tag{2.7}$$

avec $\gamma = 1$ pour une matrice de l'inverse de la distance et $\gamma = 2$ pour l'inverse de la distance au carré.

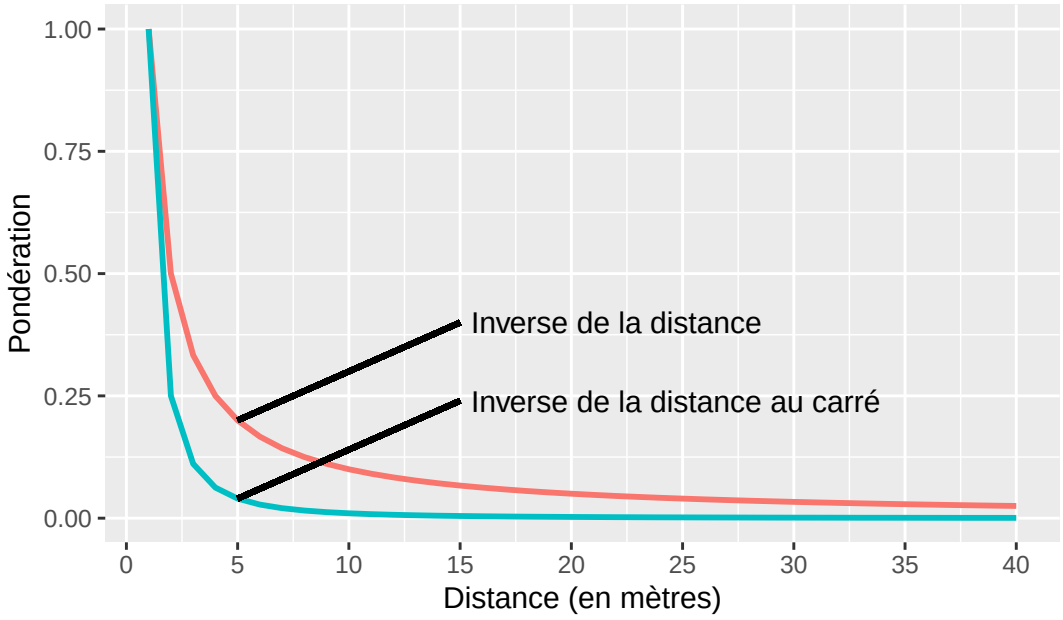


FIGURE 2.8 – Comparaison des matrices inverse de la distance et inverse de la distance au carré

Aller plus loin

Pondération avec l'exponentielle inverse

Dans un excellent livre intitulé *Économétrie spatiale appliquée des microdonnées*, Jean Dubé et Diego Legros (2014) proposent de transformer la matrice des distances avec l'inverse de l'exponentielle (ou l'exponentielle négative de la distance) (équation 2.8). Comparativement à l'inverse de la distance au carré, cette opération fait chuter encore plus rapidement les pondérations.

$$w_{ij} = \begin{cases} \frac{1}{e^{d_{ij}}} = e^{-d_{ij}} & \\ 0 & \text{si } i = j \end{cases} \tag{2.8}$$

Notez que l'équation 2.7 peut être légèrement modifiée en introduisant un seuil maximal de la distance au-delà duquel les pondérations sont mises à 0 (équation 2.9). Autrement dit, cela permet de ne pas tenir compte des entités spatiales distantes à plus d'un seuil fixé par l'analyste, ce qui est particulièrement intéressant lorsque vous analysez un phénomène dont la diffusion (ou propagation) cesse ou est très minime au-delà d'une certaine distance. Par exemple, pour la propagation du bruit routier, le seuil de 300 mètres est souvent utilisé. Par conséquent, une mesure d'autocorrélation spatiale sur des mesures de bruit routier devrait probablement recourir à un seuil maximal de 300 mètres. Autre exemple, la superficie du territoire vital diffère selon les espèces animales (cerf, caribou, ours et loup, par exemple). Par conséquent, une ou un biologiste calculant des mesures d'autocorrélation spatiale risque aussi de fixer un seuil maximal différent selon l'espèce étudiée.

$$w_{ij} = \begin{cases} \frac{1}{d_{ij}^{\gamma}} & \text{si } d_{ij} \leq \bar{d} \\ 0 & \text{si } d_{ij} > \bar{d} \\ 0 & \text{si } i = j \end{cases} \tag{2.9}$$

Astuce

Calculer le rayon maximal à partir d'une aire
Admettons que la superficie du territoire vital d'une espèce soit de 50 hectares, soit 0,5 km^2 ou 500 000 m^2. La formule bien connue pour calculer la superficie d'un cercle est $S = \pi r^2$ avec S et r étant respectivement la superficie et le rayon. Par conséquent, celle du rayon est $r = \sqrt{\frac{S}{\pi}}$. Pour trouver le rayon, vous devez taper `sqrt(500000 / pi)` dans la console de R et obtenir ainsi une distance de `398.9423` qui pourrait être arrondie à 400 mètres.

2.2.2.4 Matrices selon le critère des plus proches voisins

Une autre façon très utilisée pour définir une matrice de proximité à partir d'une matrice de distance consiste à retenir uniquement les *n* plus proches voisins. La matrice est aussi binaire avec les valeurs de 1 si les observations sont parmi les *n* plus proches de l'entité spatiale i et de 0 pour une situation inverse.

2.2.3 Standardisation des matrices de pondération spatiale en ligne

Il est recommandé de standardiser les matrices de pondération en ligne. La somme de la matrice de pondération sera alors égale au nombre d'entités spatiales de la couche géographique.

Attention

Quel est l'intérêt de la standardisation?
Nous verrons dans les sections suivantes que ces matrices sont utilisées pour évaluer le degré d'autocorrélation spatiale globale et locale. Or, il est fréquent de comparer les valeurs des mesures d'autocorrélation spatiale obtenues avec différentes matrices d'adjacence et de proximité (contiguïté selon le partage d'un nœud, d'une frontière commune; inverse de la distance, inverse de la distance au carré, etc.). Autrement dit, la standardisation des matrices de pondération spatiale permet de vérifier si le degré de (dis)ressemblance des entités spatiales en fonction d'une variable donnée est plus fort avec une matrice de contiguïté, d'inverse de la distance ou encore d'inverse de la distance au carré, etc.

Pour illustrer comment réaliser une standardisation, nous utilisons une couche géographique comprenant peu d'entités spatiales, soit celle des quatre arrondissements de la ville de Sherbrooke (figure 2.9).

Au tableau 2.2, différentes matrices de contiguïté et de distance ont été calculées, puis standardisées. Voici comment interpréter les différentes sections du tableau :

- **Contiguïté selon le partage d'une frontière commune.** La valeur de 1 signale que deux arrondissements sont voisins, sinon la valeur est à 0. Tel qu'indiqué aux équation 2.1 et équation 2.2, un arrondissement ne peut être voisin de lui-même (ex.: valeur de 0 pour la cellule `Bro.` et `Bro.`). L'arrondissement de Brompton–Rock Forest–Saint-Élie–Deauville (`Bro.`) a deux voisins, soit ceux des Nations et de Fleurimont (`Nat.` et `Fle.`), comme indiqué par la valeur 2 dans la colonne `total`. Par contre, les arrondissements des Nations et de Fleurimont sont voisins de tous les autres (valeur de 3 dans la colonne `total`).
- **Standardisation de la matrice de contiguïté.** Il suffit de diviser chaque valeur de la matrice de contiguïté par la somme de la ligne correspondante. De la sorte, la somme de chaque ligne est égale à 1 et la somme de l'ensemble des valeurs de la matrice est égale au nombre d'entités spatiales (ici 4).
- **Distance (km).** Nous avons calculé la distance euclidienne en kilomètres entre les centroïdes des arrondissements.

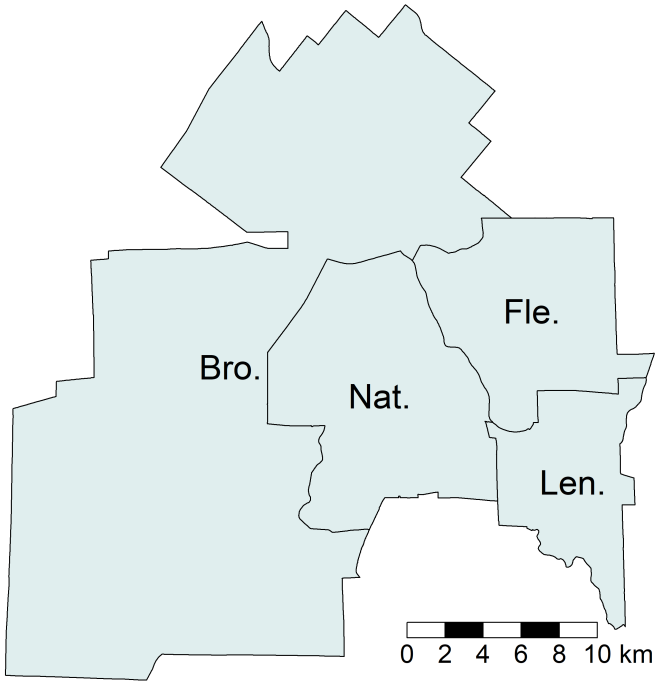


FIGURE 2.9 – Arrondissements de la ville de Sherbrooke

- **Inverse de la distance.** Les valeurs sont obtenues avec la formule $1/_{dij}$. Par exemple, entre `Bro.` et `Nat.`, nous avons $1/7,9930 = 0,1251$.
- **Inverse de la distance au carré.** Les valeurs sont obtenues avec la formule $1/_{dij^2}$. Par exemple, entre `Bro.` et `Nat.`, nous avons $1/7,9930^2 = 0,0160$.
- **Standardisation de l'inverse de la distance.** Comme précédemment, il suffit de diviser chaque valeur de la matrice par la somme de la ligne correspondante. Par exemple, pour `Bro.` et `Nat.`, nous avons $0,1251/0,3241 = 0,3860$. Remarquez que la somme des lignes est bien égale à 1.
- **Standardisation de l'inverse de la distance au carré.** Comme précédemment, il suffit de diviser chaque valeur de la matrice par la somme de la ligne correspondante. Par exemple, pour `Bro.` et `Nat.`, nous avons $0,0160/0,0360 = 0,4440$. Remarquez que la somme des lignes est bien égale à 1.

TABLEAU 2.2 – Standardisation de matrices de pondération spatiale

Arrondissement	Bro.	Nat.	Len.	Fle.	Somme (lignes)
Matrice de contiguïté selon le partage d'une frontière commune					
Bro.	0,0000	1,0000	0,0000	1,0000	2,0000
Nat.	1,0000	0,0000	1,0000	1,0000	3,0000
Len.	0,0000	1,0000	0,0000	1,0000	2,0000
Fle.	1,0000	1,0000	1,0000	0,0000	3,0000
Standardisation de la matrice de contiguïté					
Bro.	0,0000	0,5000	0,0000	0,5000	1,0000
Nat.	0,3330	0,0000	0,3330	0,3330	1,0000
Len.	0,0000	0,5000	0,0000	0,5000	1,0000

Fle.	0,3330	0,3330	0,3330	0,0000	1,0000
Distance (km)					
Bro.	0,0000	7,9930	18,9940	16,1140	
Nat.	7,9930	0,0000	11,1190	9,1650	
Len.	18,9940	11,1190	0,0000	9,2590	
Fle.	16,1140	9,1650	9,2590	0,0000	
Matrice selon l'inverse de la distance					
Bro.	0,0000	0,1251	0,0526	0,0621	0,2398
Nat.	0,1251	0,0000	0,0899	0,1091	0,3241
Len.	0,0526	0,0899	0,0000	0,1080	0,2505
Fle.	0,0621	0,1091	0,1080	0,0000	0,2792
Matrice selon l'inverse de la distance au carré					
Bro.	0,0000	0,0160	0,0030	0,0040	0,0230
Nat.	0,0160	0,0000	0,0080	0,0120	0,0360
Len.	0,0030	0,0080	0,0000	0,0120	0,0230
Fle.	0,0040	0,0120	0,0120	0,0000	0,0280
Standardisation de l'inverse de la distance					
Bro.	0,0000	0,5220	0,2190	0,2590	1,0000
Nat.	0,3860	0,0000	0,2770	0,3370	1,0000
Len.	0,2100	0,3590	0,0000	0,4310	1,0000
Fle.	0,2220	0,3910	0,3870	0,0000	1,0000
Standardisation de l'inverse de la distance au carré					
Bro.	0,0000	0,6960	0,1300	0,1740	1,0000
Nat.	0,4440	0,0000	0,2220	0,3330	1,0000
Len.	0,1300	0,3480	0,0000	0,5220	1,0000
Fle.	0,1430	0,4290	0,4290	0,0000	1,0000

2.2.4 Construction de matrices de pondération spatiale dans R

⚠ **Attention**

Construction des matrices dans R avec le *package*`spdep`.
Le package `spdep` dispose de différentes fonctions pour construire des matrices de contiguïté, de connectivité et de distance :

- `poly2nb` pour des matrices de contiguïté (section 2.2.4.1);
- `nblag` et `nblag_cumul` pour des matrices de contiguïté avec des ordres d'adjacence (section 2.2.4.2);
- `dnearneigh` pour des matrices de connectivité (section 2.2.4.3);
- `as.matrix(dist(coords))` et `mat2listw` pour des matrices de distance (section 2.2.4.4);
- `knn2nb` pour des matrices selon le critère des plus proches voisins (section 2.2.4.5).

2.2.4.1 Matrices de pondération spatiale selon la contiguïté

Pour créer des matrices de pondération spatiale selon la contiguïté décrites à la section 2.2.1, nous utilisons deux fonctions du *package* `spdep` :

- `poly2nb(Nom de l'objet sf, queen=TRUE)` crée une matrice de contiguïté sous la forme d'une classe `nb` (*A neighbours list with class nb*). Avec le paramètre `queen=TRUE`, la contiguïté est évaluée selon le partage d'un nœud; avec `queen=FALSE`, la contiguïté est évaluée selon le partage d'un segment (frontière). La matrice spatiale comprend une ligne par secteur de recensement avec les index des polygones adjacents. Par exemple, `Queen[[1]]` renvoie la liste des polygones voisins à la première entité spatiale, soit `2 14 15 16 23 32`, c'est-à-dire six voisins.
- `nb2listw(objet nb, zero.policy=TRUE, style = "W")` crée une matrice de pondération spatiale à partir de n'importe quelle matrice spatiale (de contiguïté ou de distance). Le `paramètre style = "W"`, qui est par défaut, permet de standardiser la matrice en ligne. Par exemple, `W.Queen$weights[[1]]` renvoie les valeurs des pondérations pour la première entité spatiale, soit `0.1666667 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667` (0,1666667 = 1 / 6 voisin). Pour obtenir une matrice non standardisée, vous devez écrire `style = "B"`, alors `W.Queen$weights[[1]]` renverra les valeurs de `1 1 1 1 1 1`.

```
library(sf)    # pour importer des couches géographiques
library(spdep) # pour construire les matrices de pondération
## Importation de la couche des secteurs de recensement de la ville de Sherbrooke
SR <- st_read(dsn = "data/chap02/Recen2021Sherbrooke.gpkg",
              layer = "DR_SherbSRDonnees2021", quiet=TRUE)
## Matrice selon le partage d'un nœud (Queen)
# Création de la matrice spatiale
Queen <- poly2nb(SR, queen=TRUE)
# Affichage de la première ligne de la matrice
Queen[[1]]
```

```
[1]  2 14 15 16 23 32
```

```
# Création de la matrice de pondération avec une standardisation en ligne
W.Queen <- nb2listw(Queen, zero.policy=TRUE, style = "W")
# Affichage de la première ligne des pondérations
W.Queen$weights[[1]]
```

```
[1] 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667
```

```
cat("La somme de la première ligne de la matrice de pondération est égale à",
    sum(W.Queen$weights[[1]]))
```

```
La somme de la première ligne de la matrice de pondération est égale à 1
```

```
## Matrice selon le partage d'un segment (Rook)
Rook <- poly2nb(SR, queen=FALSE)
W.Rook <- nb2listw(Rook, zero.policy=TRUE, style = "W")
##  Comparaison des deux matrices de contiguïté
# Résultat de la matrice de pondération (Queen)
summary(W.Queen)
```

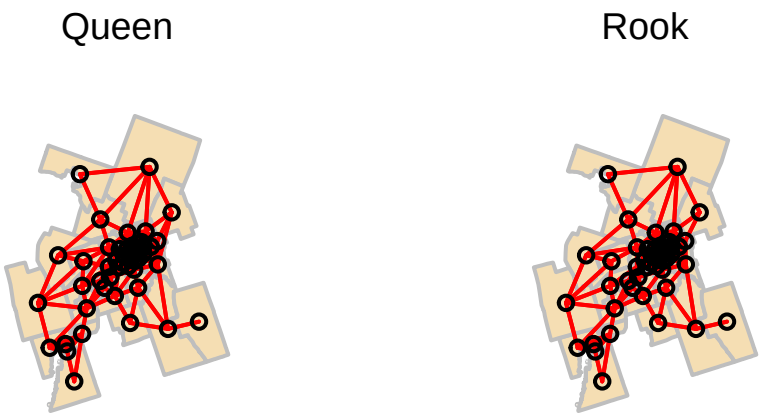
```
Characteristics of weights list object:
Neighbour list object:
Number of regions: 50
Number of nonzero links: 272
Percentage nonzero weights: 10.88
Average number of links: 5.44
Link number distribution:

 1  2  3  4  5  6  7  8 10
 1  2  2  9 13  9  8  5  1
1 least connected region:
41 with 1 link
1 most connected region:
29 with 10 links

Weights style: W
Weights constants summary:
   n   nn S0      S1       S2
W 50 2500 50 20.3056 205.5251
```

```
# Résultat de la matrice de pondération (Rook)
summary(W.Rook)
```

```
Characteristics of weights list object:
Neighbour list object:
Number of regions: 50
Number of nonzero links: 248
Percentage nonzero weights: 9.92
Average number of links: 4.96
Link number distribution:

 1  2  3  4  5  6  7  9
 1  2  4  9 17 11  5  1
1 least connected region:
41 with 1 link
1 most connected region:
29 with 9 links

Weights style: W
Weights constants summary:
   n   nn S0       S1       S2
W 50 2500 50 21.84674 205.0781
```

La syntaxe ci-dessous permet de visualiser et de comparer les graphes selon le partage d'un nœud (*Queen*) ou d'un segment commun (*Rook*).

```
par(mfrow=c(1,2)) # permet d'avoir quatre graphiques (2x2)
coords <- st_coordinates(st_centroid(SR))
## Graphe selon le partage d'un nœud
plot(st_geometry(SR), border="gray", lwd=2, col="wheat")
plot(Queen, coords, add=TRUE, col="red", lwd=2)
title(main="Queen", font.main= 1)
## Graphe selon le partage d'une frontière commune
plot(st_geometry(SR), border="gray", lwd=2, col="wheat")
plot(Rook, coords, add=TRUE, col="red", lwd=2)
title(main="Rook", font.main= 1)
```



2.2.4.2 Matrices de pondération spatiale selon la contiguïté et un ordre d'adjacence

Pour décrire la construction des matrices de contiguïté avec un ordre d'adjacence (décrites à la section 2.2.1), nous utilisons une couche géographique comprenant peu d'entités spatiales, soit celle des quatre arrondissements de la ville de Sherbrooke (figure 1). Le code ci-dessous permet d'obtenir les résultats suivants :

- `Rook <- poly2nb(Arrondissements, queen=FALSE)`: matrice d'ordre 1 selon le partage d'un segment.
- `str(Rook)`: pour chaque arrondissement, la liste des arrondissements adjacents d'ordre 1.
- `Rook.Ordre2 <- nblag(Rook, 2)`: création d'une matrice d'ordre 2 avec la fonction `nblag`.
- `str(Rook.Ordre2[[1]])`: liste des voisins d'ordre 1. Bien entendu, le résultat est identique à `str(Rook)`.
- `str(Rook.Ordre2[[2]])`: liste des voisins d'ordre 2.
- `Rook.Ordre2Cumule <- nblag_cumul(Rook.Ordre2)`: fusion des deux listes en une seule avec la fonction `nblag_cumul`.

```
Arrondissements <- st_read("data/chap02/Arrondissements.shp", quiet=TRUE)
## Matrice de contiguïté d'ordre 1 selon le partage d'un segment (Rook)
Rook <- poly2nb(Arrondissements, queen=FALSE)
str(Rook)
```

```
List of 4
 $ : int [1:2] 2 4
 $ : int [1:3] 1 3 4
```

```
 $ : int [1:2] 2 4
 $ : int [1:3] 1 2 3
 - attr(*, "class")= chr "nb"
 - attr(*, "region.id")= chr [1:4] "1" "2" "3" "4"
 - attr(*, "call")= language poly2nb(pl = Arrondissements, queen = FALSE)
 - attr(*, "type")= chr "rook"
 - attr(*, "snap")= num 0.01
 - attr(*, "sym")= logi TRUE
 - attr(*, "ncomp")=List of 2
  ..$ nc     : num 1
  ..$ comp.id: num [1:4] 1 1 1 1
```

```
## Matrice de contiguïté d'ordre 2 selon le partage d'un segment (Rook)
Rook.Ordre2 <- nblag(Rook, 2)
## Rook.Ordre2 comprend deux listes : l'une pour l'ordre 1 et l'autre pour l'autre 2.
str(Rook.Ordre2[[1]])
```

```
List of 4
 $ : int [1:2] 2 4
 $ : int [1:3] 1 3 4
 $ : int [1:2] 2 4
 $ : int [1:3] 1 2 3
 - attr(*, "class")= chr "nb"
 - attr(*, "region.id")= chr [1:4] "1" "2" "3" "4"
 - attr(*, "call")= language poly2nb(pl = Arrondissements, queen = FALSE)
 - attr(*, "type")= chr "rook"
 - attr(*, "snap")= num 0.01
 - attr(*, "sym")= logi TRUE
 - attr(*, "ncomp")=List of 2
  ..$ nc     : num 1
  ..$ comp.id: num [1:4] 1 1 1 1
```

```
str(Rook.Ordre2[[2]])
```

```
List of 4
 $ : int 3
 $ : int 0
 $ : int 1
 $ : int 0
 - attr(*, "class")= chr "nb"
 - attr(*, "region.id")= chr [1:4] "1" "2" "3" "4"
 - attr(*, "sym")= logi TRUE
 - attr(*, "ncomp")=List of 2
  ..$ nc     : num 3
  ..$ comp.id: num [1:4] 1 2 1 3
```

```
## La fonction nblag_cumul permet de combiner les deux ordres dans une seule liste
Rook.Ordre2Cumule <- nblag_cumul(Rook.Ordre2)
str(Rook.Ordre2Cumule)
```

```
List of 4
 $ : int [1:3] 2 3 4
 $ : int [1:3] 1 3 4
 $ : int [1:3] 1 2 4
 $ : int [1:3] 1 2 3
 - attr(*, "region.id")= chr [1:4] "1" "2" "3" "4"
 - attr(*, "call")= language nblag_cumul(nblags = Rook.Ordre2)
 - attr(*, "class")= chr "nb"
 - attr(*, "sym")= logi TRUE
```

```
## Création de la matrice de pondération spatiale standardisée
WRook.Ordre2Cumule <- nb2listw(Rook.Ordre2Cumule, zero.policy=TRUE, style = "W")
```

La figure 2.10 permet de constater qu'au second ordre, chacun des arrondissements est relié aux trois autres.

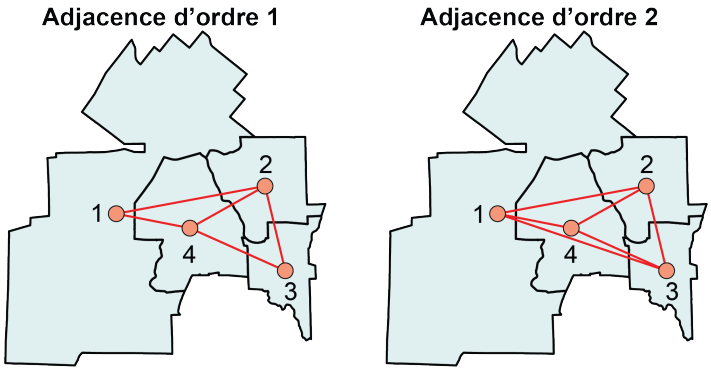


FIGURE 2.10 – Adjacence de premier et de second ordre

Reprenons la couche des secteurs de recensement de la ville de Sherbrooke pour construire des matrices d'adjacence d'ordre 1 à 3.

```
# Création des matrices d'ordre 1, 2 et 3
Queen1 <- poly2nb(SR, queen=TRUE)
Queen2 <- nblag_cumul(nblag(Queen1, 2))
Queen3 <- nblag_cumul(nblag(Queen1, 3))
# Création des matrices
W.Queen1 <- nb2listw(Queen, zero.policy=TRUE, style = "W")
W.Queen2 <- nb2listw(Queen2, zero.policy=TRUE, style = "W")
W.Queen3 <- nb2listw(Queen3, zero.policy=TRUE, style = "W")
```

2.2.4.3 Matrice de connectivité (matrice distance binaire)

La fonction `dnearneigh(sf points, d1=, d2=)` crée une matrice de connectivité (décrite à la section 2.2.2) à partir d'une couche de points. Les paramètres `d1` et `d2` permettent de spécifier le rayon de recherche (ex. : avec `d1 = 0` et `d2 = 2500`, le seuil maximal de distance est de 2500 mètres).

Si votre couche `sf` comprend des `lignes` ou des `polygones`, utilisez la fonction `st_centroid` ou `st_point_on_surface()` pour les convertir en points (section 1.2.2).

```
## Conversion des polygones en points avec st_centroid
SR.centroides <- st_centroid(SR)
## Matrice binaire avec un seuil de 2500 mètres
Connect2500m <- dnearneigh(SR.centroides, d1 = 0, d2 = 2500)
## Matrice de pondération spatiale standardisée en ligne
W.Connect2500m <- nb2listw(Connect2500m, zero.policy=TRUE, style = "W")
```

2.2.4.4 Matrices de pondération spatiale selon l'inverse de la distance et l'inverse de la distance au carré

Dans la section 2.2.3, nous avons présenté les matrices de l'inverse de la distance et de l'inverse de la distance au carré. Le code ci-dessous, qui permet de les créer, comprend les étapes suivantes :

- Récupération des coordonnées géographiques des entités spatiales.
- Création de la matrice de distance euclidienne $n \times n$ (n étant le nombre d'entités spatiales de la couche).
- Calcul des matrices d'inverse de la distance et d'inverse de la distance au carré.
- Standardisation de ces deux matrices et transformation dans des objets `listw` avec la fonction `mat2listw`.

```
## Coordonnées des centroïdes des entités spatiales
coords <- st_coordinates(SR.centroides)
## Création de la matrice de distance
distances <- as.matrix(dist(coords, method = "euclidean"))
# S'assurer que la diagonale de la matrice est à 0
diag(distances) <- 0
## Matrices inverse de la distance et inverse de la distance au carré
InvDistances <- ifelse(distances!=0, 1/distances, distances)
InvDistances2 <- ifelse(distances!=0, 1/distances^2, distances)
## Matrices de pondération spatiale standardisées en ligne
W_InvDistances  <- mat2listw(InvDistances, style="W")
W_InvDistances2 <- mat2listw(InvDistances2, style="W")
## Visualisation des valeurs des pondération pour la première entité spatiale
round(W_InvDistances$weights[[1]],4)
```

```
 [1] 0.0688 0.0505 0.0377 0.0330 0.0220 0.0191 0.0152 0.0116 0.0155 0.0220
[11] 0.0303 0.0382 0.0582 0.0677 0.0661 0.0366 0.0373 0.0316 0.0248 0.0123
[21] 0.0178 0.0241 0.0055 0.0084 0.0083 0.0084 0.0106 0.0232 0.0125 0.0231
[31] 0.0425 0.0192 0.0164 0.0049 0.0086 0.0071 0.0062 0.0061 0.0054 0.0049
[41] 0.0078 0.0050 0.0032 0.0043 0.0036 0.0030 0.0035 0.0042 0.0037
```

```
# La somme de la ligne est bien égale à 1
sum(W_InvDistances$weights[[1]])
```

```
[1] 1
```

Astuce

Intégration d'autres types de distance
À la section 2.2.2.1, nous avons vu que plusieurs types de distances peuvent être utilisés : cartésiennes (euclidienne et de Manhattan) et réticulaires (chemin le plus rapide à pied, à vélo, en automobile et en transport en commun). Pour construire une matrice de distance de Manhattan, vous devez changer la valeur du paramètre `method` de la fonction `dist` comme suit : `as.matrix(dist(coords, method = "manhattan"))`.
Pour intégrer une distance réticulaire, vous devez la calculer, soit dans R (chapitre 5), soit dans un logiciel SIG (ArcGIS Pro avec l'extension *Network Analyst* par exemple) et l'importer dans R. Le reste du code sera alors identique.

Nous avons vu qu'il est possible d'utiliser une matrice de distance en fixant une distance maximale au-delà de laquelle les pondérations sont mises à 0 (équation 2.9 à la section 2.2.2.1). Le code ci-dessous permet de créer des matrices de pondération standardisées avec l'inverse de la distance et l'inverse de la distance au carré avec des seuils de 2500 et de 5000 mètres.

```
## Coordonnées des centroïdes des entités spatiales
coords <- st_coordinates(SR.centroides)
## Création de la matrice de distance
distances <- as.matrix(dist(coords, method = "euclidean"))
## Création de différentes matrices avec différents seuils
InvDistances.2500m  <- ifelse(distances<=2500 & distances!=0, 1/distances, 0)
InvDistances.5000m  <- ifelse(distances<=5000 & distances!=0, 1/distances, 0)
InvDistances2.2500m <- ifelse(distances<=2500 & distances!=0, 1/distances^2, 0)
InvDistances2.5000m <- ifelse(distances<=5000 & distances!=0, 1/distances^2, 0)
## Matrices de pondération spatiale standardisées en ligne
W_InvDistances.2500 <- mat2listw(InvDistances.2500m, style="W", zero.policy = TRUE)
W_InvDistances.5000 <- mat2listw(InvDistances.5000m, style="W", zero.policy = TRUE)
W_InvDistances2.2500 <- mat2listw(InvDistances2.2500m, style="W", zero.policy = TRUE)
W_InvDistances2.5000 <- mat2listw(InvDistances2.5000m, style="W", zero.policy = TRUE)
```

Spécifier un seuil de distance peut toutefois être problématique. Par exemple, sur les 50 secteurs de recensement de la ville de Sherbrooke, 19 n'ont pas de voisins à 2500 mètres, indiqués par le résultat suivant :

```
## 19 regions with no links:

## 23 24 25 30 34 35 36 37 38 39 40 41 42 43 44 45 47 49 50
```

```
summary(W_InvDistances.2500, zero.policy=TRUE)
```

```
Characteristics of weights list object:
Neighbour list object:
Number of regions: 50
Number of nonzero links: 188
Percentage nonzero weights: 7.52
Average number of links: 3.76
19 regions with no links:
23, 24, 25, 30, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 47, 49,
50
21 disjoint connected subgraphs
Link number distribution:

 0  1  2  3  4  6  7  8  9 10 11 12 13
19  6  2  2  4  2  2  3  4  1  2  1  2
6 least connected regions:
21 26 31 33 46 48 with 1 link
2 most connected regions:
12 13 with 13 links

Weights style: W
Weights constants summary:
   n  nn S0       S1       S2
W 31 961 31 19.09079 128.6954
```

Même avec un seuil de 5000 mètres, il reste encore 11 SR sans voisins.

```
summary(W_InvDistances.5000, zero.policy=TRUE)
```

```
Characteristics of weights list object:
Neighbour list object:
Number of regions: 50
Number of nonzero links: 532
Percentage nonzero weights: 21.28
Average number of links: 10.64
11 regions with no links:
35, 36, 37, 39, 40, 41, 42, 43, 47, 49, 50
13 disjoint connected subgraphs
Link number distribution:

 0  1  2  3  5  6  7  8  9 10 11 16 18 19 20 21 22 23
11  2  2  4  2  1  1  2  1  1  1  2  2  4  2  4  7  1
2 least connected regions:
24 30 with 1 link
1 most connected region:
12 with 23 links
```

```
Weights style: W
Weights constants summary:
   n   nn S0       S1       S2
W 39 1521 39 12.11283 162.9801
```

> ⚠ **Attention**
>
> **Réduction de la taille des matrices de distance**
> Plusieurs logiciels (notamment ArcGIS Pro et GeoDa) réduisent par défaut la taille des matrices de distance de la façon suivante : 1) construction d'une matrice de distance uniquement pour l'entité la plus proche (la matrice résultante est donc de dimension $n \times 1$); 2) obtention de la distance maximale dans cette matrice, soit la distance la plus grande entre une entité spatiale et celle la plus proche; 3) construction de la matrice de distance finale avec comme seuil la distance maximale obtenue à l'étape précédente.
> Cette réduction procure deux avantages importants :
> - **Une diminution considérable des temps de calcul**, surtout pour les couches géographiques comprenant un nombre très élevé d'entités spatiales. Par exemple, avec une couche de 50 entités spatiales, la matrice des distances comprendra 2500 valeurs ($50 \times 50 = 2500$) tandis qu'avec 1000 entités spatiales, elle en comprendra un million ($1000 \times 1000 = 1\,000\,000$).
> - Comme décrit plus haut, il est préférable d'**éviter d'avoir une matrice de distance avec des entités spatiales sans voisins**, puisque cela a un impact négatif sur les mesures d'autocorrélation spatiale.

La syntaxe ci-dessous permet ainsi de construire des matrices de pondération (inverse de la distance et inverse de la distance au carré) à partir de la distance maximale et un SR et son voisin le plus proche.

```
## Coordonnées des centroïdes des entités spatiales
coords <- st_coordinates(SR.centroides)
## Trouver le plus proche voisin
k1 <- knn2nb(knearneigh(coords))
## Affichage des distances pour les 50 SR au le plus proche
round(unlist(nbdists(k1,coords)),0)
```

```
 [1]  1352   563   563   659   833  1275  1275  1299  2136  1553  1171   833
[13]   953   953  1024  1024   936   936  1149  1242  2378  1473  3863  4963
[25]  2841  1755  1755  2294  1507  4002  1843  1710  2486  2977 10953  6965
[37]  5331  4077  6717  6892  6892  6335  5639  3530  4202  1636  6662  1636
[49] 10745 10034
```

```
## Trouver la distance maximale
plusprochevoisin.max <- max(unlist(nbdists(k1,coords)))
cat("Distance maximale au plus proche voisin :", round(plusprochevoisin.max,0), "mètres")
```

```
Distance maximale au plus proche voisin : 10953 mètres
```

```
## Matrice de distance avec la valeur maximale
# les voisins les plus proches avec le seuil de distance maximal
Voisins.DistMax <- dnearneigh(coords, 0, plusprochevoisin.max)
# Distances avec le seuil maximum
distances <- nbdists(Voisins.DistMax, coords)
# Inverse de la distance
InvDistances <- lapply(distances, function(x) (1/x))
# Inverse de la distance au carré
InvDistances2 <- lapply(distances, function(x) (1/x^2))
## Matrices de pondération spatiale standardisées en ligne
W_InvDistances  <- nb2listw(Voisins.DistMax, glist = InvDistances, style = "W")
W_InvDistances2 <- nb2listw(Voisins.DistMax, glist = InvDistances2, style = "W")
```

2.2.4.5 Matrices de pondération spatiale selon le critère des plus proches voisins

La fonction `knearneigh` du *package* `spdep` crée des matrices de distance selon le critère des plus proches voisins (décrit à la section 2.2.2.4), dont le nombre est fixé avec le paramètre `k`.

```
## Coordonnées géographiques des centroïdes des polygones
coords <- st_coordinates(st_centroid(SR))
## Matrices des plus proches voisins de 2 à 5
k2 <- knn2nb(knearneigh(coords, k = 2))
k3 <- knn2nb(knearneigh(coords, k = 3))
k4 <- knn2nb(knearneigh(coords, k = 4))
k5 <- knn2nb(knearneigh(coords, k = 5))
## Matrices de pondération spatiale standardisées en ligne
W.k2 <-  nb2listw(k2, zero.policy=FALSE, style = "W")
W.k3 <-  nb2listw(k3, zero.policy=FALSE, style = "W")
W.k4 <-  nb2listw(k4, zero.policy=FALSE, style = "W")
W.k5 <-  nb2listw(k5, zero.policy=FALSE, style = "W")
```

La syntaxe ci-dessous permet de comparer les matrices des plus proches voisins de $k = 2$ à 5 (figure 2.11).

```
par(mfrow=c(2,2))
plot(st_geometry(SR), border="gray", lwd=2, col="wheat")
plot(k2, coords, add=TRUE, col="red", lwd=2)
title(main="k = 2")

plot(st_geometry(SR), border="gray", lwd=2, col="wheat")
plot(k3, coords, add=TRUE, col="red", lwd=2)
title(main="k = 3")

plot(st_geometry(SR), border="gray", lwd=2, col="wheat")
plot(k4, coords, add=TRUE, col="red", lwd=2)
title(main="k = 4")
```

```
plot(st_geometry(SR), border="gray", lwd=2, col="wheat")
plot(k5, coords, add=TRUE, col="red", lwd=2)
title(main="k = 5")
```

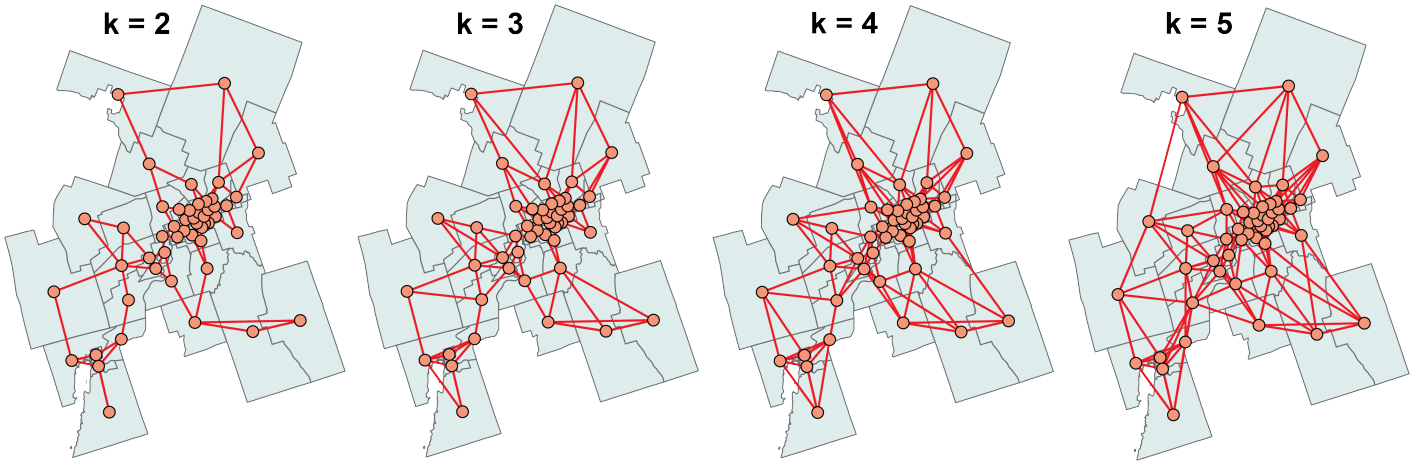


FIGURE 2.11 – Matrices selon le critère des plus proches voisins

2.3 Autocorrélation spatiale globale

Dans le cadre de cette section, nous présentons uniquement les mesures d'autocorrélation spatiale globale les plus utilisées, à savoir le I de Moran pour évaluer l'autocorrélation spatiale d'une variable continue (section 2.3.1), les statistiques de comptage de jointure (*Join Count Statistics*) pour une variable binaire ou catégorielle (section 2.3.2) et l'indice de Lee pour évaluer l'autocorrélation spatiale de deux variables continues (section 2.3.3).

2.3.1 Statistique du I de Moran

2.3.1.1 Formulation du I de Moran

Pour évaluer le degré d'autocorrélation spatiale d'une variable continue, les deux principales statistiques utilisées sont le I de Moran (1950) et le c de Geary (1954). Puisqu'elles renvoient une seule valeur pour la variable continue de la couche géographique étudiée, elles sont qualifiées de mesures globales de l'autocorrélation spatiale, par opposition aux mesures locales qui renvoient une valeur par entité spatiale (section 2.4).

Nous présentons ici uniquement le I de Moran pour deux raisons principales. Premièrement, étant basée sur la covariance, son interprétation est bien plus facile que celle du c de Geary (basé sur la variance des écarts), c'est-à-dire qu'elle est très similaire au bien connu coefficient de corrélation de Pearson (Apparicio et Gelb 2022). Deuxièmement, elle constitue la mesure la plus utilisée. Le I de Moran s'écrit :

$$I = \frac{n}{\Sigma_{i=1}^{n}\Sigma_{j=1}^{n} w_{ij}} \frac{\Sigma_{i=1}^{n}\Sigma_{j=1}^{n} w_{ij}(x_i - \bar{x})(x_j - \bar{x})}{\Sigma_{i=1}^{n}(x_i - \bar{x})^2} \text{ avec :} \tag{2.10}$$

- n, le nombre d'entités spatiales dans la couche géographique;
- w_{ij}, la valeur de la pondération spatiale entre les entités spatiales i et j;
- x_i et x_j, les valeurs de la variable continue pour les entités spatiales i et j;
- $\bar{x}$, la valeur moyenne de la variable X à l'étude.

Note

Standardisation de la matrice de pondération et *I* de Moran

Nous avons vu que si la matrice de pondération spatiale est standardisée en ligne (section 2.2.3), alors chaque ligne de la matrice vaut 1 et la somme de l'ensemble des valeurs de la matrice est égale au nombre d'entités spatiales (n). Or, dans l'équation 2.10), $\Sigma_{i=1}^{n}\Sigma_{j=1}^{n}w_{ij}$ représente la somme des pondérations de la matrice, soit n si elles sont standardisées en ligne. Puisque $\frac{n}{n} = 1$, alors l'équation du I de Moran est simplifiée comme suit :

$$I = \frac{\Sigma_{i=1}^{n}\Sigma_{j=1}^{n}w_{ij}(x_i - \bar{x})(x_j - \bar{x})}{\Sigma_{i=1}^{n}(x_i - \bar{x})^2} \tag{2.11}$$

Comme évoqué dans la section 2.2.3, cela démontre l'intérêt de la standardisation : la comparaison des valeurs du I de Moran obtenues avec différentes matrices de contiguïté afin de sélectionner (éventuellement) celle avec laquelle la dépendance spatiale est la plus forte.

2.3.1.2 Interprétation du I de Moran

Avec une matrice standardisée, la statistique du I de Moran varie de -1 à 1 et s'interprète de la façon suivante :

- quand $I > 0$, l'autocorrélation est positive, c'est-à-dire que les entités géographiques ont tendance à se ressembler d'autant plus qu'elles sont voisines ou proches;
- quand $I = 0$, l'autocorrélation est nulle, c'est-à-dire que la contiguïté ou la proximité spatiale des zones ne jouent aucun rôle;
- quand $I < 0$, l'autocorrélation est négative, c'est-à-dire que les entités géographiques ont tendance à être dissemblables d'autant plus qu'elles sont voisines ou proches.

Les limites de -1 et 1 sont les maximums théoriques du I de Moran. Dans la pratique, elles sont limitées par la matrice spatiale utilisée dans le calcul. En effet, selon la matrice spatiale, le maximum du I de Moran peut être inférieur à 1, et son minimum supérieur à -1. Le calcul de ces bornes propres à chaque matrice spatiale peut se faire en utilisant les maximums et minimums des valeurs propres de $\frac{W+W^T}{2}$, quand la matrice spatiale est standardisée.

À titre d'exemple, nous calculons ci-dessous les maximums et minimums possibles pour une matrice de contiguïté selon le partage d'un nœud (*Queen*) de nos secteurs de recensement.

```
# Matrice de contiguïté selon le partage d'un nœud (Queen)
Queen <- poly2nb(SR, queen = TRUE)
WQueen <- nb2listw(Queen, style = 'W')
QueenMat <- listw2mat(WQueen)
values <- range(eigen((QueenMat + t(QueenMat))/2)$values)
print(round(values,2))
```

```
[1] -0.74  1.02
```

Il apparaît ainsi que pour la matrice matrice de contiguïté selon le partage d'un nœud (*Queen*), quelle soit la variable analysée, la valeur de *I* de Moran ne pourra pas dépasser les limites -0,74 et 1,02.

2.3.1.3 Significativité du I de Moran

Comme pour le coefficient de corrélation calculé entre deux variables, il est possible de tester la significativité de la valeur du *I* de Moran obtenue. Sans que nous détaillions les calculs de significativité, notez qu'il existe trois manières de tester la significativité :

- selon l'hypothèse de la normalité;
- selon l'hypothèse de la randomisation;
- selon des permutations Monte-Carlo (habituellement avec 999 échantillons).

Aller plus loin

Comment calculer les trois tests de significativité du *I* de Moran?
Pour une description détaillée du calcul des trois tests, consultez l'ouvrage de Jean Dubé et Diego Legros (2014).

2.3.1.4 Mise en œuvre dans R

Objectif

Calcul du *I* de Moran dans R
Pour illustrer le calcul de *I* de Moran dans R, nous utilisons une couche des aires de diffusion (AD) de la ville de Sherbrooke. Les étapes suivantes sont réalisées :

1. Construire une panoplie de matrices de pondération spatiale selon la contiguïté, la connectivité, la proximité et le critère des plus proches voisins.
2. Comparer les valeurs de significativité (*p*) pour une variable continue (`HabKm2`).
3. Pour cette même variable, trouver avec quelle matrice la valeur du *I* de Moran est la plus forte.
4. Comparer les valeurs du *I* de Moran calculées sur plusieurs variables.

Étape 1. Construction des matrices de pondération spatiale

```
library(sf)
library(spdep)
## Importation de la couche des aires de diffusion de la ville de Sherbrooke
AD.DR <- st_read(dsn = "data/chap02/Recen2021Sherbrooke.gpkg",
              layer = "DR_SherbADDonnees2021", quiet=TRUE)

## Matrices de contiguïté
################################################
## Partage d'un nœud (Queen)
Queen <- poly2nb(AD.DR, queen=TRUE)
W.Queen <- nb2listw(Queen, zero.policy=TRUE, style = "W")
## Partage d'un segment (Rook)
```

```
Rook <- poly2nb(AD.DR, queen=FALSE)
W.Rook <- nb2listw(Rook, zero.policy=TRUE, style = "W")
## Partage d'un segment (Rook) et ordres d'adjacence de 2 à 5
Rook2 <- nblag_cumul(nblag(Rook, 2))
Rook3 <- nblag_cumul(nblag(Rook, 3))
Rook4 <- nblag_cumul(nblag(Rook, 4))
Rook5 <- nblag_cumul(nblag(Rook, 5))
W.Rook2 <- nb2listw(Rook2, zero.policy=TRUE, style = "W")
W.Rook3 <- nb2listw(Rook3, zero.policy=TRUE, style = "W")
W.Rook4 <- nb2listw(Rook4, zero.policy=TRUE, style = "W")
W.Rook5 <- nb2listw(Rook5, zero.policy=TRUE, style = "W")

## Matrice de connectivité
################################################
## Matrice binaire avec un seuil de 2500 mètres
Connect2500m <-   dnearneigh(st_centroid(AD.DR), d1 = 0, d2 = 2500)
W.Connect2500m <- nb2listw(Connect2500m, zero.policy=TRUE, style = "W")

## Matrices de proximité
################################################
## Coordonnées géographiques et matrice de distance
coords <- st_coordinates(st_centroid(AD.DR))
distances <- as.matrix(dist(coords, method = "euclidean"))
diag(distances) <- 0
## Matrices inverse de la distance et inverse de la distance au carré
InvDistances <- ifelse(distances!=0, 1/distances, distances)
InvDistances2 <- ifelse(distances!=0, 1/distances^2, distances)
## Matrices de pondération spatiale standardisées en ligne
W.InvDistances  <- mat2listw(InvDistances, style="W")
W.InvDistances2 <- mat2listw(InvDistances2, style="W")
## Création de différentes matrices avec différents seuils
InvDistances.2500m  <- ifelse(distances<=2500 & distances!=0, 1/distances, 0)
InvDistances.5000m  <- ifelse(distances<=5000 & distances!=0, 1/distances, 0)
InvDistances2.2500m <- ifelse(distances<=2500 & distances!=0, 1/distances^2, 0)
InvDistances2.5000m <- ifelse(distances<=5000 & distances!=0, 1/distances^2, 0)
W.InvDistances_2500 <- mat2listw(InvDistances.2500m, style="W", zero.policy = TRUE)
W.InvDistances_5000 <- mat2listw(InvDistances.5000m, style="W", zero.policy = TRUE)
W.InvDistances2_2500 <- mat2listw(InvDistances2.2500m, style="W", zero.policy = TRUE)
W.InvDistances2_5000 <- mat2listw(InvDistances2.5000m, style="W", zero.policy = TRUE)
## Matrice de distance réduite standardisée
k1 <- knn2nb(knearneigh(coords))
plusprochevoisin.max <- max(unlist(nbdists(k1,coords)))
Voisins.DistMax <- dnearneigh(coords, 0, plusprochevoisin.max)
distances <- nbdists(Voisins.DistMax, coords)
InvDistances <- lapply(distances, function(x) (1/x))
InvDistances2 <- lapply(distances, function(x) (1/x^2))
```

```
W_InvDistancesReduite  <- nb2listw(Voisins.DistMax, glist = InvDistances, style = "W")
W_InvDistances2Reduite <- nb2listw(Voisins.DistMax, glist = InvDistances2, style = "W")

## Matrice selon le critère des plus proches voisins
#########################################################
## Matrices des plus proches voisins de 2 à 5
k2 <- knn2nb(knearneigh(coords, k = 2))
k3 <- knn2nb(knearneigh(coords, k = 3))
k4 <- knn2nb(knearneigh(coords, k = 4))
k5 <- knn2nb(knearneigh(coords, k = 5))
## Matrices de pondération spatiale standardisées en ligne
W.k2 <-  nb2listw(k2, zero.policy=FALSE, style = "W")
W.k3 <-  nb2listw(k3, zero.policy=FALSE, style = "W")
W.k4 <-  nb2listw(k4, zero.policy=FALSE, style = "W")
W.k5 <-  nb2listw(k5, zero.policy=FALSE, style = "W")
```

Étape 2. Calcul du I de Moran et des trois tests de significativité

Calculons la statistique du I de Moran sur la variable continue cartographiée à la figure 2.12.

Les fonctions `moran.test` et `moran.mc` du *package* `spdep` permettent de calculer le I de Moran selon les trois façons de tester la significativité :

- selon l'hypothèse de la normalité avec le paramètre `randomisation = FALSE`
  - `moran.test(ObjetSf$Variable, listw=MatriceW, zero.policy=TRUE, randomisation = FALSE)`
- selon l'hypothèse de la randomisation avec le paramètre `randomisation = TRUE`
  - `moran.test(ObjetSf$Variable, listw=MatriceW, zero.policy=TRUE, randomisation = TRUE)`
- selon des permutations Monte-Carlo (ci-dessus avec 999 permutations)
  - `moran.mc(ObjetSf$Variable, listw=MatriceW, zero.policy=TRUE, nsim=999)`

Bien entendu, dans les sorties des trois méthodes, la valeur du I de Moran est la même, contrairement à la valeur de p qui peut varier.

```
moran.test(AD.DR$HabKm2,          # nom de l'objet sf et de la variable continue
           listw=W.Queen,         # nom de la matrice de pondération spatiale
           zero.policy=TRUE,
           randomisation = FALSE) # significativité selon l'hypothèse de la normalité
```

```
    Moran I test under normality

data:  AD.DR$HabKm2
weights: W.Queen

Moran I statistic standard deviate = 11.724, p-value < 2.2e-16
```

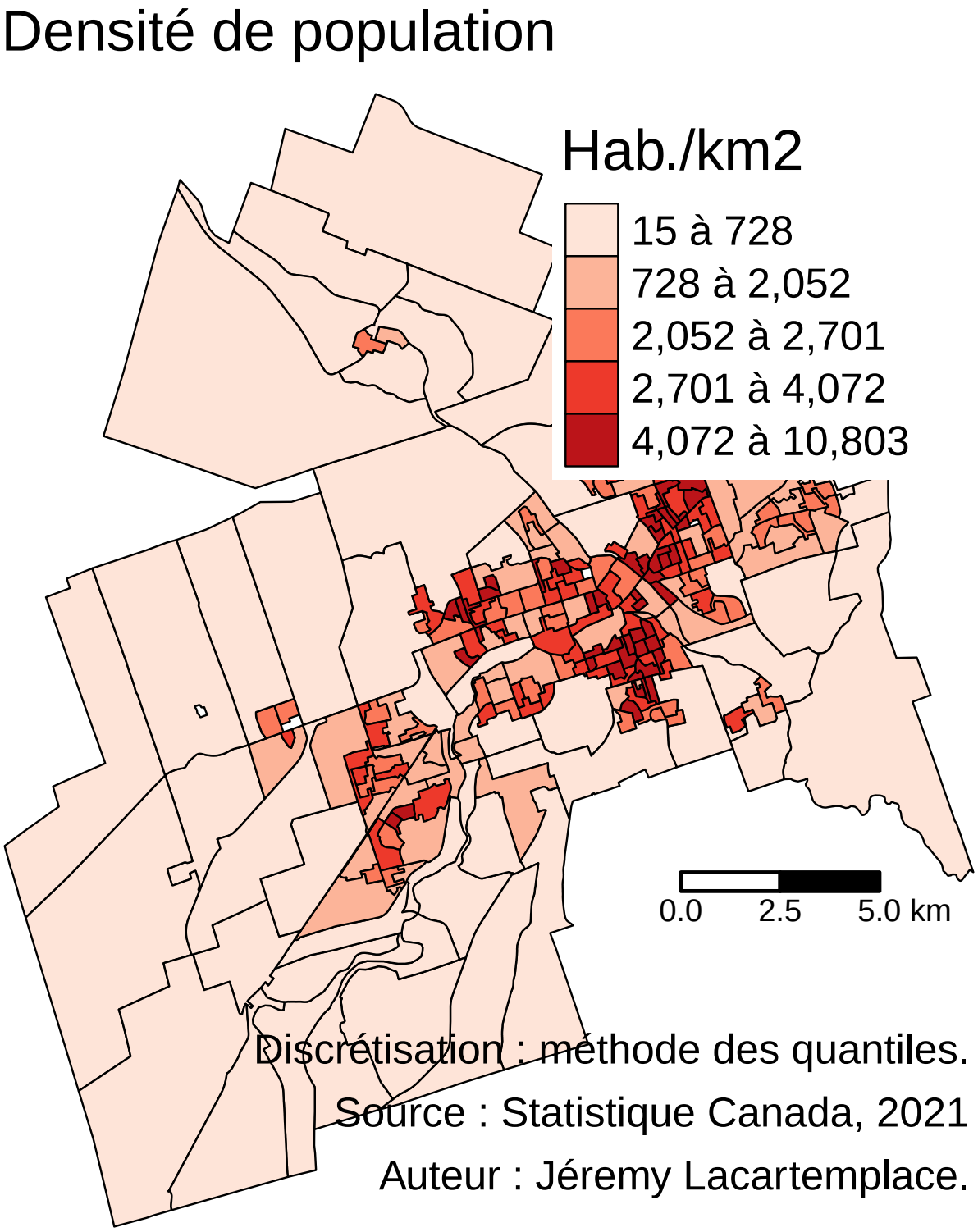


FIGURE 2.12 – Densité de population, aires de diffusion de la ville de Sherbrooke

```
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
      0.433714579      -0.004032258       0.001394035
```

```
moran.test(AD.DR$HabKm2,          # nom de l'objet sf et de la variable continue
           listw=W.Queen,         # nom de la matrice de pondération spatiale
           zero.policy=TRUE,
           randomisation = TRUE)  # significativité selon l'hypothèse de la randomisation
```

```
    Moran I test under randomisation

data:  AD.DR$HabKm2
weights: W.Queen

Moran I statistic standard deviate = 11.761, p-value < 2.2e-16
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
      0.433714579      -0.004032258       0.001385364
```

```
moran.mc(AD.DR$HabKm2,            # nom de l'objet sf et de la variable continue
         listw=W.Queen,           # nom de la matrice de pondération spatiale
         zero.policy=TRUE,
         nsim=999)                # 999 permutations
```

```
    Monte-Carlo simulation of Moran I

data:  AD.DR$HabKm2
weights: W.Queen
number of simulations + 1: 1000

statistic = 0.43371, observed rank = 1000, p-value = 0.001
alternative hypothesis: greater
```

Nous calculons la mesure du I de Moran sur la variable continue cartographiée à la figure 2.13.

La statistique du I de Moran (I = 0,43, p < 0,001) indique que la variable *densité de population* a une forte autocorrélation spatiale positive (figure 2.13), avec des valeurs fortes dans les aires de diffusion contiguës dans la partie centrale de la ville et des valeurs faibles dans les aires de diffusion contiguës dans les secteurs périphériques (figure 2.12).

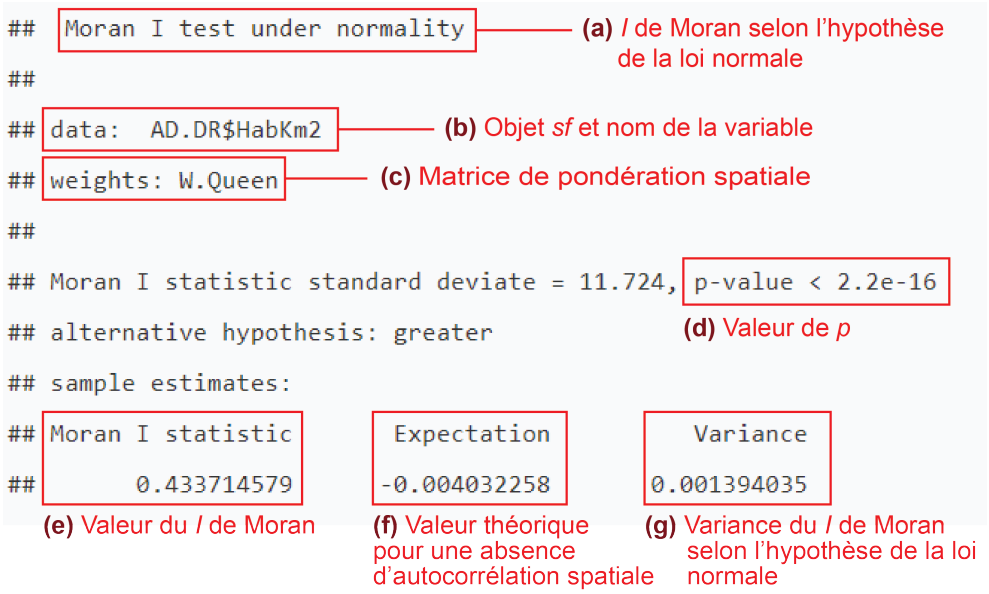


FIGURE 2.13 – Résultats du I de Moran selon l'hypothèse de la loi normale

Étape 3. Identification de la plus forte autocorrélation spatiale selon les différentes matrices

La syntaxe ci-dessous permet de calculer la statistique du I de Moran avec plusieurs matrices de pondération spatiale.

```
## Création d'un vecteur pour les noms des matrices
VecteurMatrices <- c("W.Queen", "W.Rook", "W.Rook2", "W.Rook3", "W.Rook4", "W.Rook5",
                     "W.Connect2500m",
                     "W.InvDistances", "W.InvDistances2",
                     "W_InvDistancesReduite", "W_InvDistances2Reduite",
                     "W.InvDistances_2500", "W.InvDistances_5000",
                     "W.InvDistances2_2500","W.InvDistances2_5000",
                     "W.k2", "W.k3", "W.k4", "W.k5")
## Création d'une liste pour toutes les matrices
ListeMatrices <- list(W.Queen, W.Rook, W.Rook2, W.Rook3, W.Rook4, W.Rook5,
                      W.Connect2500m,
                      W.InvDistances, W.InvDistances2,
                      W_InvDistancesReduite, W_InvDistances2Reduite,
                      W.InvDistances_2500, W.InvDistances2_2500,
                      W.InvDistances2_2500, W.InvDistances2_5000,
                      W.k2, W.k3, W.k4, W.k5)
## Vecteur pour le I de Moran et la valeur de p
MoranI <- c()
Pvalue <- c()
i<-0
## Boucle pour calculer le I de Moran avec la liste des matrices
for (e in ListeMatrices){
   i<-i+1
```

```
   Test <-moran.mc(AD.DR$HabKm2,
                   listw=e,
                   zero.policy=TRUE,
                   nsim=999)
   MoranI[i]<-Test$statistic
   Pvalue[i] <- Test$p.value
}
# Création d'un DataFrame avec les valeurs du I de Moran et de p
MoranData1 <- data.frame(Matrices=VecteurMatrices,
                         MoranIs=MoranI,
                         Pvalues=Pvalue)
print(MoranData1)
```

```
                Matrices    MoranIs Pvalues
1                 W.Queen 0.43371458   0.001
2                  W.Rook 0.44970946   0.001
3                 W.Rook2 0.32509097   0.001
4                 W.Rook3 0.21527754   0.001
5                 W.Rook4 0.12614476   0.001
6                 W.Rook5 0.07129756   0.001
7          W.Connect2500m 0.25583250   0.001
8          W.InvDistances 0.10632882   0.001
9         W.InvDistances2 0.27216034   0.001
10  W_InvDistancesReduite 0.28566705   0.001
11 W_InvDistances2Reduite 0.38836327   0.001
12    W.InvDistances_2500 0.32115230   0.001
13    W.InvDistances_5000 0.40350492   0.001
14   W.InvDistances2_2500 0.40350492   0.001
15   W.InvDistances2_5000 0.34988630   0.001
16                  W.k2 0.51070049   0.001
17                  W.k3 0.44458619   0.001
18                  W.k4 0.44800959   0.001
19                  W.k5 0.43874109   0.001
```

La lecture détaillée du tableau 2.3 permet d'avancer plusieurs constats intéressants :

- D'emblée, signalons que toutes les valeurs sont positives et significatives, témoignant d'une autocorrélation spatiale positive.
- Concernant les **matrices de contiguïté**, la dépendance spatiale est plus forte selon le partage d'un segment que d'un nœud (0,4497 contre 0,4337). Par conséquent, si nous devons choisir une matrice de contiguïté, il serait préférable d'utiliser celle définie selon le partage d'une chaîne (*Rook*).
- Sans surprise, plus nous ajoutons des **ordres d'adjacence**, plus la valeur de la statistique du I de Moran est faible, passant de 0,3251 à 0,0713 du deuxième au cinquième ordre.
- La valeur du I de Moran avec une **matrice de connectivité** avec 2500 mètres est de 0,2558. Elle est plus faible que celles de l'inverse de la distance et l'inverse de la distance au carré, avec le même seuil de 2500 mètres (0,3212 et 0,4035).

— Concernant les **matrices de proximité**, la méthode de l'inverse de la distance au carré, qui accorde un poids plus important aux entités spatiales très proches (comparativement à l'inverse de la distance), renvoie des valeurs toujours plus élevées, et ce, que la matrice soit complète ou réduite. Aussi, les matrices de distance réduites présentent toujours des valeurs plus fortes que celles complètes.
— Concernant les matrices **selon le critère des plus proches voisins**, l'autocorrélation spatiale diminue légèrement de $k = 2$ à $k = 5$. D'ailleurs, la valeur la plus forte est pour deux voisins ($I = 0{,}5107$). Toutefois, retenir uniquement deux voisins est discutable puisque les AD sont très majoritairement contiguës à plus de deux autres AD (sur les 249 AD, seuls 9 sont contiguës à deux autres AD selon le partage d'un segment). Pour le vérifier, tapez `summary(W.Rook)` et analysez le tableau sous la ligne `Link number distribution`.

Tableau 2.3 – Résultats du I de Moran selon les différentes matrices

Nom	Description	I de Moran	p (999 permutations)
Matrices de contiguïté			
W.Queen	Partage d'un nœud	0,4337	0,001
W.Rook	Partage d'un segment	0,4497	0,001
Matrices de contiguïté selon le partage d'un segment et ordre d'adjacence			
W.Rook2	Ordre 2	0,3251	0,001
W.Rook3	Ordre 3	0,2153	0,001
W.Rook4	Ordre 4	0,1261	0,001
W.Rook5	Ordre 5	0,0713	0,001
Matrices de connectivité			
W.Connect2500m	2500 mètres	0,2558	0,001
Matrices de distance (complètes)			
W.InvDistances	Inverse de la distance	0,1063	0,001
W.InvDistances2	Inverse de la distance au carré	0,2722	0,001
Matrices de distance (réduites)			
W_InvDistancesReduite	Inverse de la distance	0,2857	0,001
W_InvDistances2Reduite	Inverse de la distance au carré	0,3884	0,001
Matrices de distance avec un seuil maximal			
W.InvDistances_2500	Inverse de la distance (2500 mètres)	0,3212	0,001
W.InvDistances_5000	Inverse de la distance (5000 mètres)	0,4035	0,001
W.InvDistances2_2500	Inverse de la distance au carré (2500 mètres)	0,4035	0,001
W.InvDistances2_5000	Inverse de la distance au carré (5000 mètres)	0,3499	0,001
Matrices selon le critère des plus proches voisins			
W.k2	2 voisins	0,5107	0,001
W.k3	3 voisins	0,4446	0,001
W.k4	4 voisins	0,4480	0,001
W.k5	5 voisins	0,4387	0,001

Quelle est la matrice avec laquelle la dépendance spatiale de la variable est la plus forte?

Pour la trouver, nous construisons un graphique avec les valeurs du I de Moran triées par ordre décroissant. La valeur la plus forte est obtenue avec la matrice selon les deux plus proches voisins, suivie de la matrice *Rook*. Quoi qu'il en soit, il serait plus judicieux de privilégier la matrice de contiguïté selon le partage d'un segment comme expliqué plus haut.

```
library(ggplot2)
ggplot(data=MoranData1, aes(x=reorder(Matrices,MoranIs), y=MoranIs)) +
  geom_segment( aes(x=reorder(Matrices,MoranIs),
                    xend=reorder(Matrices,MoranIs),
                    y=0, yend=MoranIs)) +
  geom_point( size=4,fill="red",shape=21)+
  xlab("Matrice de pondération spatiale") +
  ylab("I de Moran")+
  coord_flip()
```

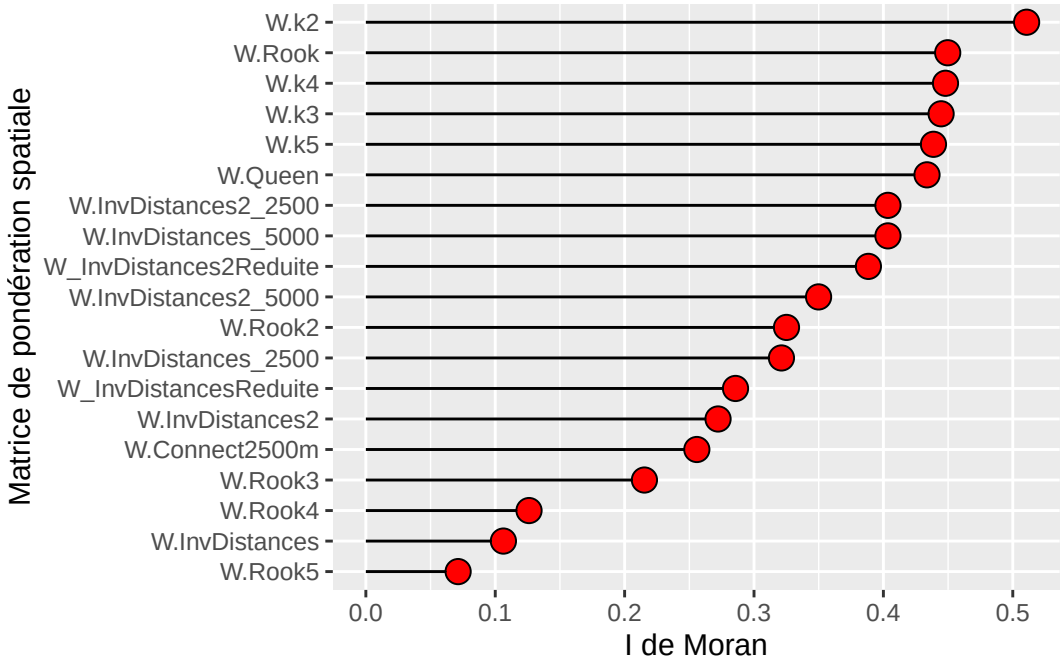


FIGURE 2.14 – Valeurs du I de Moran selon les différentes matrices de pondération spatiale

Étape 4. Comparaison des valeurs du I de Moran pour plusieurs variables avec la même matrice

La syntaxe ci-dessous permet de calculer la statistique du I de Moran pour plusieurs variables (figure 2.15) avec la même matrice de pondération spatiale (ici, matrice de contiguïté selon le partage d'un segment).

```
## Vecteur pour les variables à analyser
listeVars <- c("PctLog1960_Av", "RevMedMenage" , "PctProprieta", "PctPop0_14")

## Boucle pour calculer le I de Moran pour les différentes variables
MoranData2 <- t(sapply(listeVars, function(e){
   Test <- moran.mc(AD.DR[[e]],
                    listw=W.Rook,
                    zero.policy=TRUE,
                    nsim=999)
```

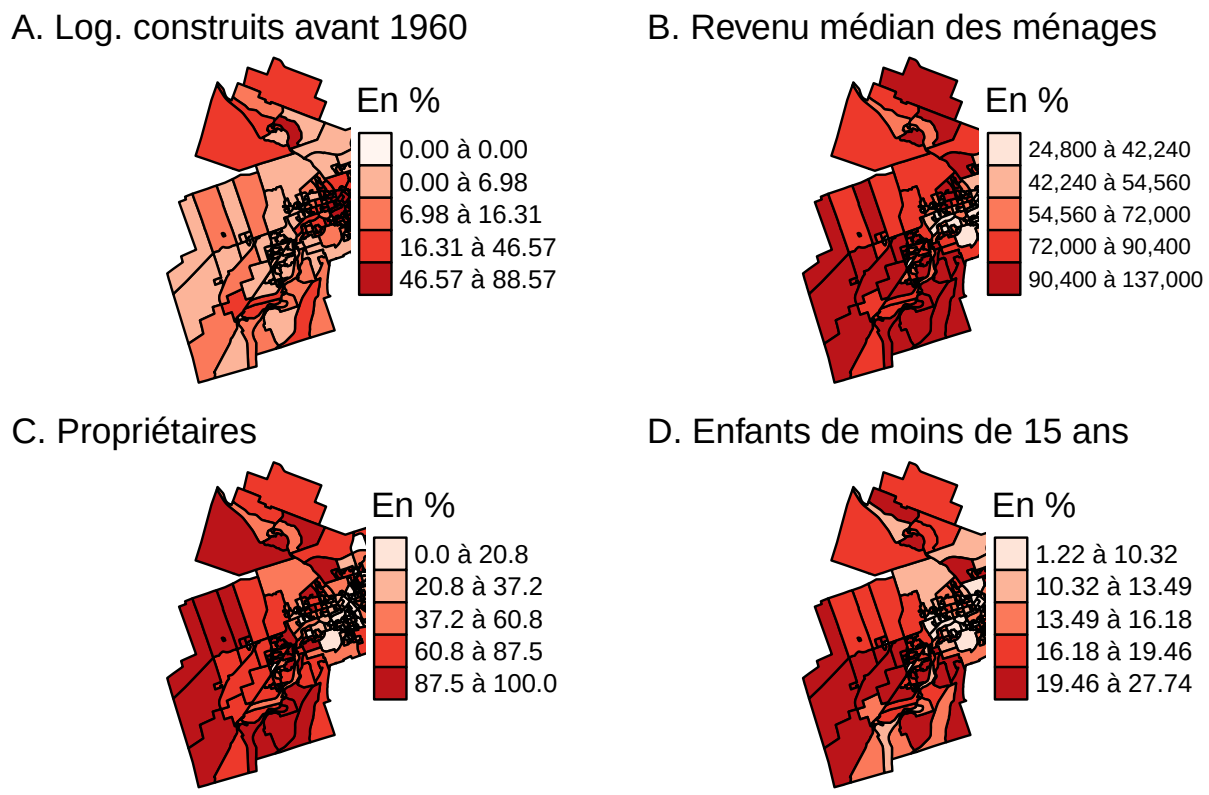


FIGURE 2.15 – Quatre variables sélectionnées pour les AD de la ville de Sherbrooke

```
   result <- c(round(Test$statistic,4),
               Test$p.value)
}))

MoranData2 <- data.frame(MoranData2)
names(MoranData2) <- c('MoranIs', 'Pvalues')
MoranData2$Variable <- listeVars
rownames(MoranData2) <- NULL

print(MoranData2)
```

```
  MoranIs Pvalues      Variable
1  0.7071   0.001 PctLog1960_Av
2  0.6168   0.001  RevMedMenage
3  0.6028   0.001  PctProprieta
4  0.4474   0.001    PctPop0_14
```

De nouveau, en quelques lignes de code, il est aisé de réaliser un graphique pour comparer les valeurs du I de Moran pour les différentes variables (figure 2.16).

```
library(ggplot2)
    ggplot(data=MoranData2, aes(x=reorder(Variable,MoranIs), y=MoranIs)) +
    geom_segment( aes(x=reorder(Variable,MoranIs), xend=reorder(Variable,MoranIs), y=0, yend=MoranIs)) +
    geom_point( size=4,fill="red",shape=21)+
    xlab("Variable continue") + ylab("I de Moran")+
    coord_flip()
```

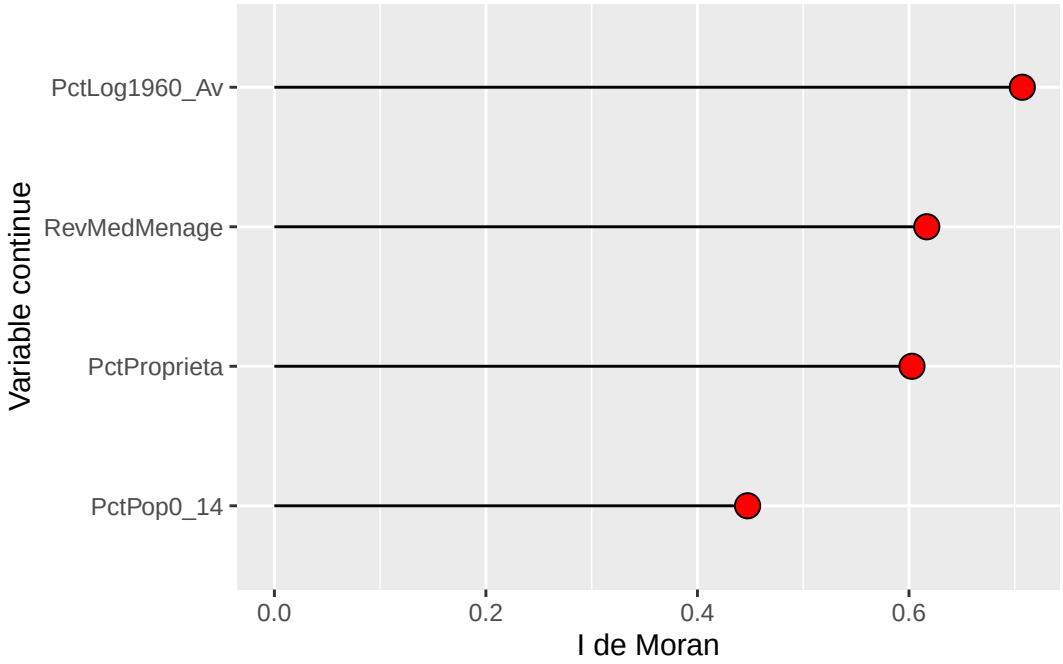


FIGURE 2.16 – Valeurs du I de Moran pour les quatre variables

2.3.2 Statistiques de comptage de jointure (Join Count Statistics)

Pour évaluer l'autocorrélation spatiale d'une variable qualitative dichotomique (binaire) ou polychotomique (catégorielle), il convient d'utiliser les ***Join Count Statistics***, qui peuvent être traduits par **statistiques de comptage de jointure**. Ces tests permettent de répondre à la question suivante : est-ce que le voisinage ou la proximité des entités spatiales augmente significativement les chances qu'elles partagent la même valeur (modalité) par rapport à ce que le hasard produirait (Cliff et Ord 1981)?

Objectif

Autocorrélation spatiale sur une variable qualitative
Ces statistiques permettent ainsi de vérifier si la distribution des modalités d'une variable binaire ou nominale est dispersée aléatoirement dans l'espace d'étude ou si elle tend à se regrouper spatialement. Voici quelques exemples applicatifs :

- **Variable binaire** (oui/non; absence/présence d'un phénomène) avec habituellement des valeurs de 0 ou 1. Dans une ville, les parcelles commerciales sont-elles distribuées aléatoirement ou tendent-elle à se regrouper?
- **Variable qualitative nominale**. À la suite d'une élection dans un pays donné, il s'agit de vérifier si les districts électoraux adjacents ont significativement été remportés par des personnes candidates du même parti. Autre exemple, la répartition d'espèces d'arbres sur un territoire donné est-elle aléatoire ou favorise-t-elle la proximité entre certaines espèces?

2.3.2.1 Formulation des statistiques de comptage de jointure

Application à une variable binaire

Pour décrire le fonctionnement de ces tests, nous utilisons deux situations fictives avec toutes deux 36 entités spatiales, dont 9 avec la valeur de 1 (noir, soit *B* par convention) et 27 avec la valeur de 0 (blanc, soit *W* par convention) (figure 2.17). La relation d'adjacence entre les entités spatiales est ici mesurée à partir d'une matrice de contiguïté selon le partage d'un segment qui est représentée avec un graphe à la figure 2.17 (b).

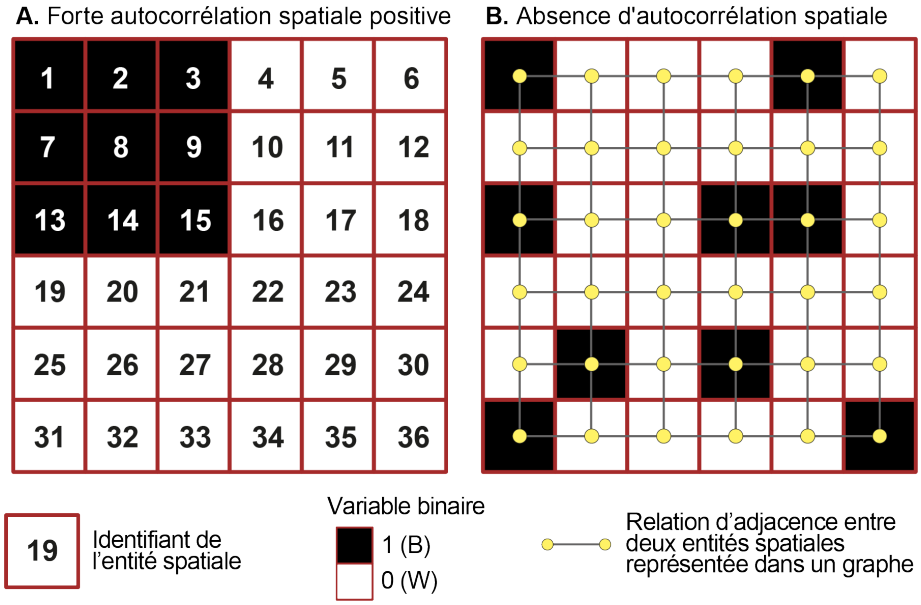


FIGURE 2.17 – Illustration de l'autocorrélation spatiale pour une variable binaire

Comptage des jointures

Le comptage des entités voisines partageant la même valeur (BB et WW, soit une autocorrélation spatiale positive) et inversement (WB, soit une autocorrélation spatiale négative) est réalisé comme suit :

- **Le nombre de voisins partageant la valeur de 1 (autocorrélation positive)** est obtenu avec l'équation 2.12. Lorsque deux entités ne sont pas adjacentes, alors $w_{ij} = 0$ et donc $w_{ij}x_ix_j = 0$. Par contre, lorsqu'elles sont voisines, trois cas de figure sont possibles :
  - **Toutes deux ont la valeur de 1**, alors $x_ix_j = 1 \times 1 = 1$.
  - Toutes deux ont la valeur de 0, alors $x_ix_j = 0 \times 0 = 0$.
  - Elles ont des valeurs différentes, alors $x_ix_j = 1 \times 0 = 0$.

$$O_{BB} = \frac{1}{2}\Sigma_{i=1}^{n}\Sigma_{j=1}^{n}w_{ij}x_ix_j \text{ avec :} \tag{2.12}$$

n étant le nombre d'entités spatiales dans la couche géographique; w_{ij} étant la valeur de la matrice de contiguïté non standardisée entre i et j (1 quand elles sont voisines, sinon 0), x_i et x_j étant les valeurs de la variable binaire (0 ou 1) pour les entités spatiales i et j.

- **Le nombre de voisins partageant la valeur de 0 (autocorrélation positive)** est obtenu avec l'équation 2.13 avec les cas suivants lorsque les deux entités sont voisines :
  - Toutes deux avec la valeur de 1, alors $(1 - x_i)(1 - x_j) = (1 - 1) \times (1 - 1) = 0 \times 0 = 0$.
  - **Toutes deux avec la valeur de 0**, alors $(1 - x_i)(1 - x_j) = (1 - 0) \times (1 - 0) = 1 \times 1 = 1$.
  - Avec des valeurs différentes, alors $(1 - x_i)(1 - x_j) = (1 - 1) \times (1 - 0) = 0 \times 1 = 0$.

$$O_{WW} = \frac{1}{2}\Sigma_{i=1}^{n}\Sigma_{j=1}^{n}w_{ij}(1 - x_i)(1 - x_j) \tag{2.13}$$

- **Le nombre de voisins ne partageant pas la même valeur (autocorrélation négative)** est obtenu avec l'équation 2.14 avec les cas suivants lorsque les deux entités sont voisines :
  - Toutes deux avec la valeur de 1, alors $(x_i - x_j)^2 = (1 - 1)^2 = 0$.
  - Toutes deux avec la valeur de 0, alors $(x_i - x_j)^2 = (0 - 0)^2 = 0$.
  - **Avec des valeurs différentes**, alors $(x_i - x_j)^2 = (1 - 0)^2 = 1$.

$$O_{BW} = \frac{1}{2}\Sigma_{i=1}^{n}\Sigma_{j=1}^{n}w_{ij}\left(x_i - x_j\right)^2 \tag{2.14}$$

Note

Note

Pour les trois équations ci-dessus, les sommes sont divisées par 2 puisque les mêmes résultats sont obtenus entre les paires (i, j) et (j, i). La somme des voisins partageant les mêmes valeurs (O_{BB} et O_{WW}) et ayant des valeurs différentes (O_{BW}) est égale à la somme de la matrice de contiguïté (S_0) divisée par deux (équation 2.15).

$$O_{BB} + O_{WW} + O_{BW} = \frac{1}{2}S_0 = \frac{1}{2}\Sigma_{i=1}^{n}\Sigma_{j=1}^{n}w_{ij} \tag{2.15}$$

Les résultats des comptages pour les situations A et B (figure 2.17) sont présentés au tableau 2.4. Ils démontrent clairement que les regroupements des valeurs de 0 et 1 sont bien plus importants pour la situation A (BB = 12 et WW = 42) que celle de B (BB = 1 et WW = 33). Reste à vérifier si ces résultats sont significatifs, c'est-à-dire s'ils diffèrent de ce que le hasard produirait.

Tableau 2.4 – Comptages des jointures

Situation	BB	WW	WB	Somme
A (forte autocorrélation spatiale)	12	42	6	60
B (absence autocorrélation spatiale)	1	33	26	60

Tests statistiques

Il existe deux approches d'inférence pour déterminer si des observations voisines tendent à produire certaines paires de catégories (BB, WW ou WB) plus souvent que le hasard : les tests reposant sur la loi binomiale et les tests par permutations. Notez que la seconde approche est habituellement privilégiée.

Tests selon la loi binomiale

Ces tests permettent d'obtenir les valeurs attendues des paires BB, WW et WB pour une absence d'autocorrélation spatiale, puis des valeurs de Z (équation 2.16) et de *p*.

$$Z_{WW} = \frac{O[WW] - E[WW]}{\sqrt{\text{Var}[WW]}} Z_{BB} = \frac{O[BB] - E[BB]}{\sqrt{\text{Var}[BB]}} Z_{WB} = \frac{O[WB] - E[WB]}{\sqrt{\text{Var}[WB]}} \text{ avec :} \tag{2.16}$$

$O[WW]$, $O[BB]$ et $O[WB]$ étant les nombres de paires observées; $E[WW]$, $E[BB]$ et $E[WB]$ étant les nombres de paires attendues et $\text{Var}[WW]$, $\text{Var}[BB]$ et $\text{Var}[WB]$ leurs variances selon la loi binomiale pour une distribution aléatoire. Pour une description détaillée des formules des valeurs attendues et des variances selon les deux types d'échantillons (indépendant et dépendant) et selon le type de matrice de pondération spatiale (standardisée ou non), consultez l'ouvrage de Wong et Lee (2005).

Aussi, le calcul des valeurs de Z repose sur deux cas de figure :

1. **Échantillon dépendant** (*non free Sampling*). Le nombre de valeurs possibles de chaque catégorie est défini en amont et ne peut pas changer. Prenons l'exemple suivant : nous souhaitons vérifier si deux restaurants voisins ont tendance à avoir une licence d'alcool (BB) ou non (WW). Si le nombre de licences d'alcool est réglementé, alors la probabilité d'en avoir une dépend du nombre de licences en circulation (échantillon dépendant).
2. **Échantillon indépendant** (*free Sampling*). Pour chaque entité spatiale, la probabilité d'avoir une catégorie est indépendante du nombre d'observations. En d'autres termes, il n'y a pas un nombre défini d'observations pour chaque catégorie (W ou B) qui sont réparties entre les entités spatiales (échantillon indépendant).

Bien distinguer ces deux configurations est essentiel, car elles affectent directement les valeurs des variances et par ricochet celles de Z et de *p* (voir les tableaux 2.5 et 2.4). Avec un test basé sur un échantillon indépendant pour la situation A, nous concluons que les valeurs de 1 et de 0 (BB et WW) ne sont pas distribuées aléatoirement puisque $p < 0{,}05$, ce qui traduit une autocorrélation spatiale. Par contre, pour la situation B, toutes les valeurs de *p* sont supérieures à 0,05, ce qui ne nous permet pas de rejeter l'hypothèse nulle d'une distribution aléatoire selon la loi binomiale.

Tableau 2.5 – Statistiques de comptage de jointures pour BB (tests basés sur la loi binomiale)

	O[BB]	E[BB]	Var[BB]	Z	p
A (échantillon dépendant)	42	33,43	3,73	4,439	0,000
A (échantillon indépendant)	42	33,75	45,98	1,217	0,112

Tableau 2.5 – Statistiques de comptage de jointures pour BB (tests basés sur la loi binomiale)

	O[BB]	E[BB]	Var[BB]	Z	p
B (échantillon dépendant)	33	33,43	3,73	-0,222	0,588
B (échantillon indépendant)	33	33,75	45,98	-0,111	0,544

Tableau 2.6 – Statistiques de comptage de jointures pour WW (tests basés sur la loi binomiale)

Situation et test	O[WW]	E[WW]	Var[WW]	Z	p
A (échantillon dépendant)	12	3,43	2,09	5,922	0,000
B (échantillon indépendant)	12	3,75	6,98	3,122	0,001
B (échantillon dépendant)	1	3,43	2,09	-1,678	0,953
B (échantillon indépendant)	1	3,75	6,98	-1,041	0,851

Tests par permutations

Cette seconde approche d'inférence est très simple et consiste à :

1. Mélanger les observations du jeu de données de nombreuses fois (habituellement 999).
2. Pour chaque permutation, compter les jointures BB, WW et WB.
3. Estimer la pseudo valeur de *p* à partir de l'équation 2.17. Cette valeur peut être interprétée comme la probabilité que le hasard génère plus souvent une paire (BB, WW ou WB) que ce qui est observé avec le jeu de données initial.

$$\text{Pseudo valeur de } p = (M+1)/(R+1) \text{ avec :} \tag{2.17}$$

M étant le nombre de fois que le comptage de jointures (BB ou WW par exemple) est égal ou supérieur à la valeur de référence (O_{BB} ou O_{WW} par exemple) et R étant le nombre de permutations (habituellement 999).

En guise d'exemple, nous réalisons 999 permutations des valeurs de la situation A qui comprend 12 paires de BB, 42 de WW et 6 de WB. Les comptages de BB et WW pour ces 999 permutations sont représentés à la figure 2.18. Nous constatons qu'elles sont toujours inférieures aux valeurs de référence (BB = 12 et WW = 42; lignes bleues). Par conséquent, la pseudo valeur de *p* est égale à $(0+1)/(999+1) = 0{,}001$. Cela signifie que nous ne pouvons pas rejeter l'hypothèse nulle stipulant que les distributions des paires BB et WW sont dues au hasard.

Effectuons la même démarche pour la situation B avec une seule paire de BB et 33 paires de WW (figure 2.19) :

- pour BB, il y a 984 permutations qui ont une valeur supérieure ou égale à 1. La pseudo valeur de *p* est égale à $(984+1)/(999+1) = 0{,}985$.
- pour WW, il y a 653 permutations qui ont une valeur supérieure ou égale à 1. La pseudo valeur de *p* est égale à $(653+1)/(999+1) = 0{,}653$.

Par conséquent, nous validons l'hypothèse nulle stipulant que les distributions des paires BB et WW sont dues au hasard pour la situation B.

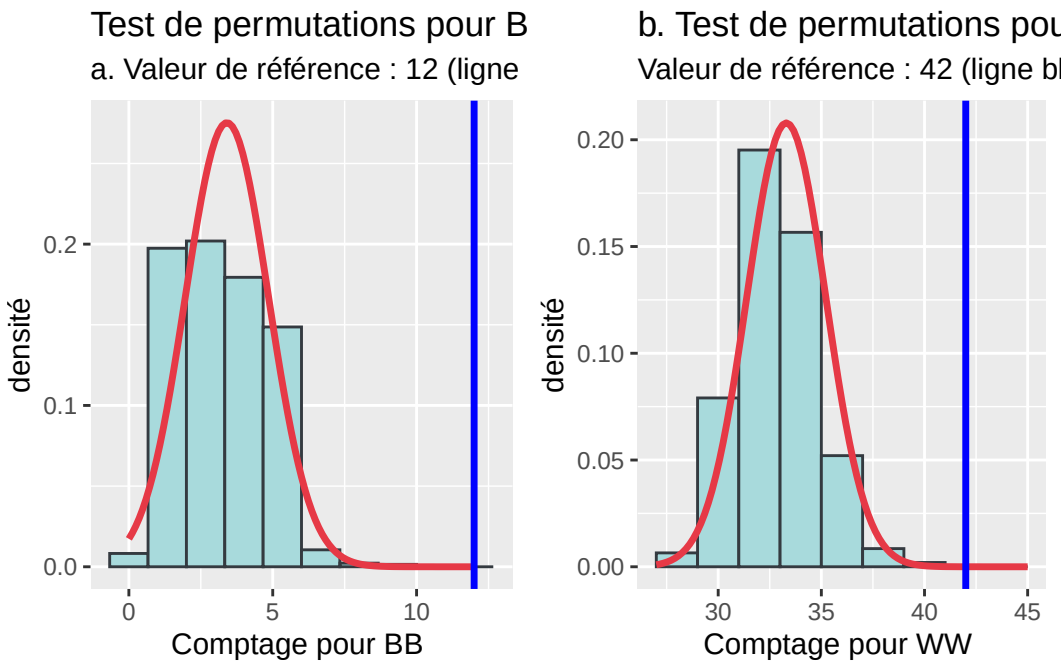


FIGURE 2.18 – Résultats des 999 permutations pour la situation A

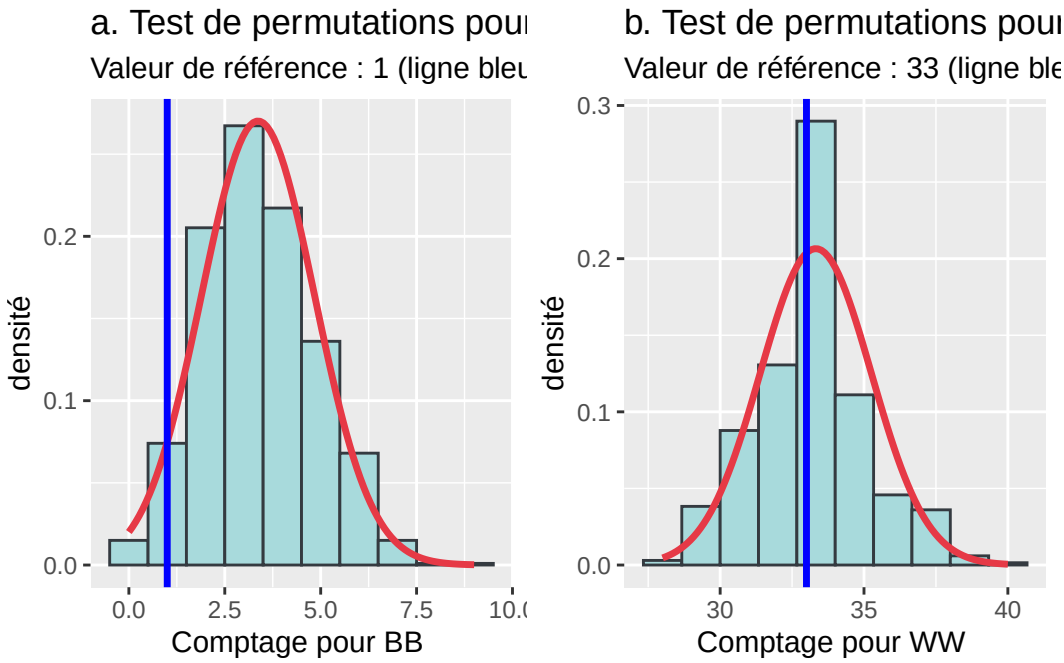


FIGURE 2.19 – Résultats des 999 permutations pour la situation B

Astuce

Comment réaliser une permutation aléatoire des valeurs d'un vecteur dans R?
Dans R, la fonction `sample(x)` permet de permuter les valeurs d'un vecteur.

```
## Valeurs initiales pour les 36 observations (cas A)
print(Carres$CasA)
```

```
 [1] 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
## Valeurs permutées
print(sample(Carres$CasA))
```

```
 [1] 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 0
```

Attention

Importance de la matrice de voisinage pour le *Join count Test*
Quelle que soit la méthode utilisée pour effectuer le *Join count Test*, le test est sensible à la définition de la matrice de voisinage, définie selon le partage d'un segment ou d'un nœud (*Queen* ou *Rook*). Elle affecte directement le nombre de paires observées et attendues et donc le niveau de significativité de l'autocorrélation spatiale.

2.3.2.2 Mise en œuvre dans R

Objectif

Calcul des statistiques de comptage de jointure dans R
Pour illustrer le calcul des statistiques de comptage de jointure (*Join Count Statistics*) dans R, nous utilisons une couche des aires de diffusion (AD) de la ville de Sherbrooke afin de répondre à la question suivante : les AD avec une majorité de locataires et celles avec une majorité de propriétaires sont-elles significativement voisines les unes des autres (autocorrélation spatiale positive) à Sherbrooke?

Dans le code ci-dessous, nous importons la couche géographique, créons une variable binaire indiquant si l'aire de diffusion a ou non une majorité de locataires et finalement, cartographions cette variable. Sans surprise, les AD avec une majorité de locataires sont concentrées dans la partie centrale de la ville (figure 2.20).

```
## Chargement des données des aires de diffusion de la ville de Sherbrooke
ADSherb <- st_read("data/chap02/Recen2021Sherbrooke.gpkg",
                   layer="DR_SherbADDonnees2021",
                   quiet=TRUE)
## Création de la variable binaire
ADSherb$maj_locataires <- ifelse(ADSherb$Locataire > ADSherb$Proprietaire, 1, 0)
ADSherb$maj_locataires <- factor(ADSherb$maj_locataires,
                                 levels =c(0,1),
                                 labels=c("Propriétaires", "Locataires"))
## Cartographie de la variable binaire
```

```
tmap_mode("plot")
tm_shape(ADSherb)+
  tm_borders(col="black", lwd=0.5)+
  tm_fill(col="maj_locataires",
          palette=c("#ededed", "#706f6f"),
          title = "Groupe majoritaire")+
  tm_layout(frame=FALSE)+
  tm_scale_bar(breaks=c(0,5))+
  tm_credits("Source : Statistique Canada, 2021.", align = "left")
```

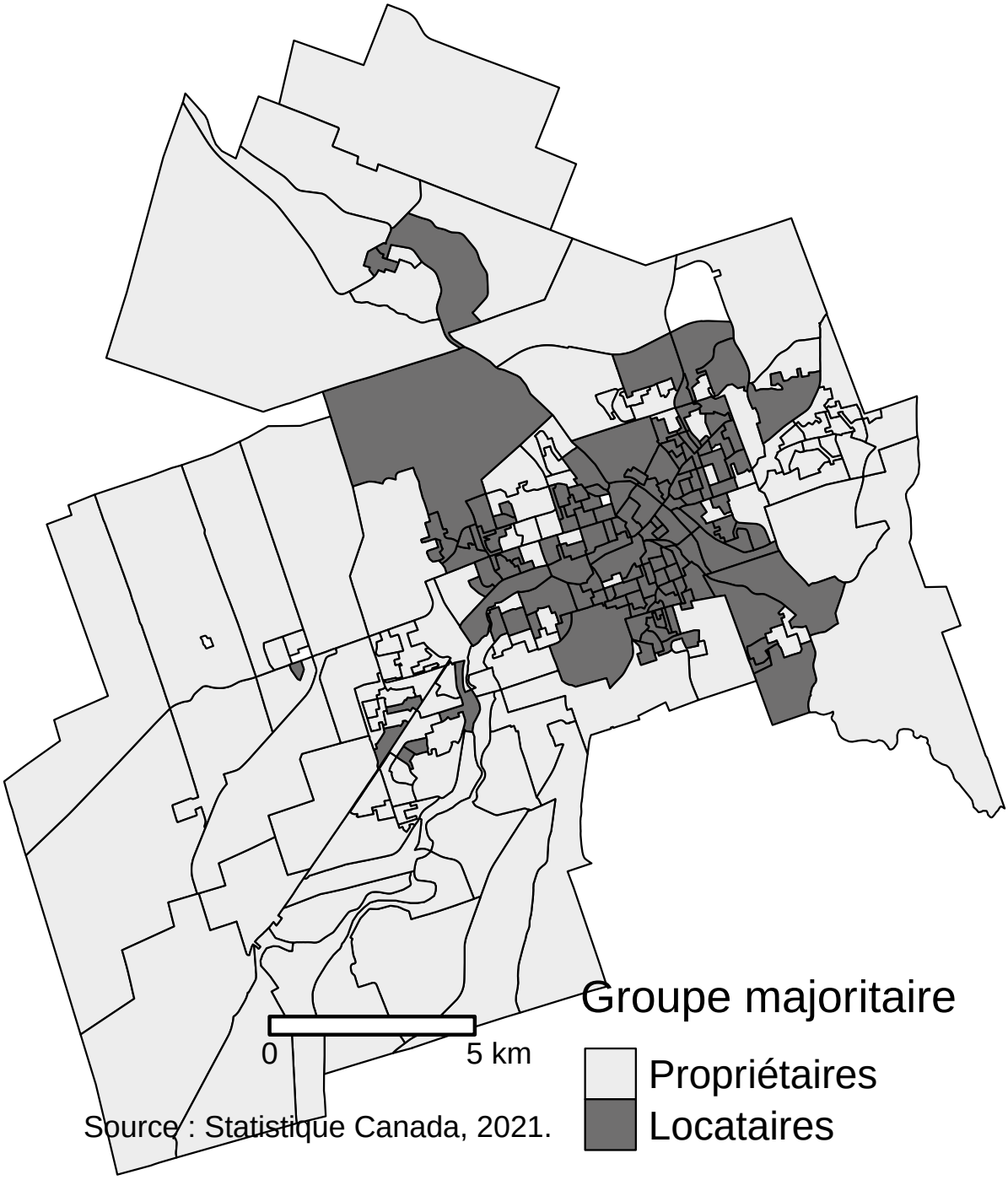


FIGURE 2.20 – Aires de diffusion avec une majorité de locataires ou de propriétaires, Sherbrooke, 2021

Pour calculer les statistiques de comptage de jointure, nous utilisons une matrice de contiguïté selon le partage d'un nœud (*Queen*). Notez que pour ces statistiques, il est préférable de ne pas standardiser la matrice spatiale (`style = "B"`), car nous souhaitons compter le nombre de paires formées par les deux catégories d'AD.

```
Queen <- poly2nb(ADSherb, queen = TRUE)
WQueen <- nb2listw(Queen, style = "B")
```

Pour réaliser les tests selon la méthode d'inférence basée sur la loi binomiale, nous utilisons la fonction `joincount.test` du *package* `spdep`. Le paramètre `sampling = "free"` permet de spécifier le calcul des variances selon un échantillon indépendant, puisque les nombres d'AD dans lesquelles les locataires ou les propriétaires sont majoritaires n'ont pas de limites fixes.

```
test1 <- joincount.test(ADSherb$maj_locataires, listw = WQueen, sampling = "free")
print(test1)
```

```
    Join count test under free sampling

data:  ADSherb$maj_locataires
weights: WQueen

Std. deviate for Propriétaires = 3.2832, p-value = 0.0005132
alternative hypothesis: greater
sample estimates:
Same colour statistic           Expectation              Variance
             244.0000              163.4227              602.3289


    Join count test under free sampling

data:  ADSherb$maj_locataires
weights: WQueen

Std. deviate for Locataires = 3.7367, p-value = 9.324e-05
alternative hypothesis: greater
sample estimates:
Same colour statistic           Expectation              Variance
             319.0000              214.8323              777.1427
```

Les résultats du test présentés à la figure 2.21 s'interprètent comme suit :

- **a. Valeurs observées** : 244 AD sont voisines et majoritairement occupées par des propriétaires contre 319 pour les locataires.
- **b. Valeurs attendues** : 163,4227 pour une majorité de propriétaires et 214,8323 pour une majorité de locataires.
- **c. variances** pour les deux groupes selon la loi binomiale avec un échantillon indépendant (602,3289 et 777,1427).
- **d. Valeurs Z**, soit $(244 - 163{,}4227)/\sqrt{602{,}3289} = 3{,}2832$ et $(319 - 214{,}8323)/\sqrt{777{,}1427} = 3{,}7367$.
- **e. Valeurs de *p*** sont inférieures à 0,005, signalant que les deux modalités de la variable binaire sont significativement autocorrélées positivement selon la matrice de contiguïté.

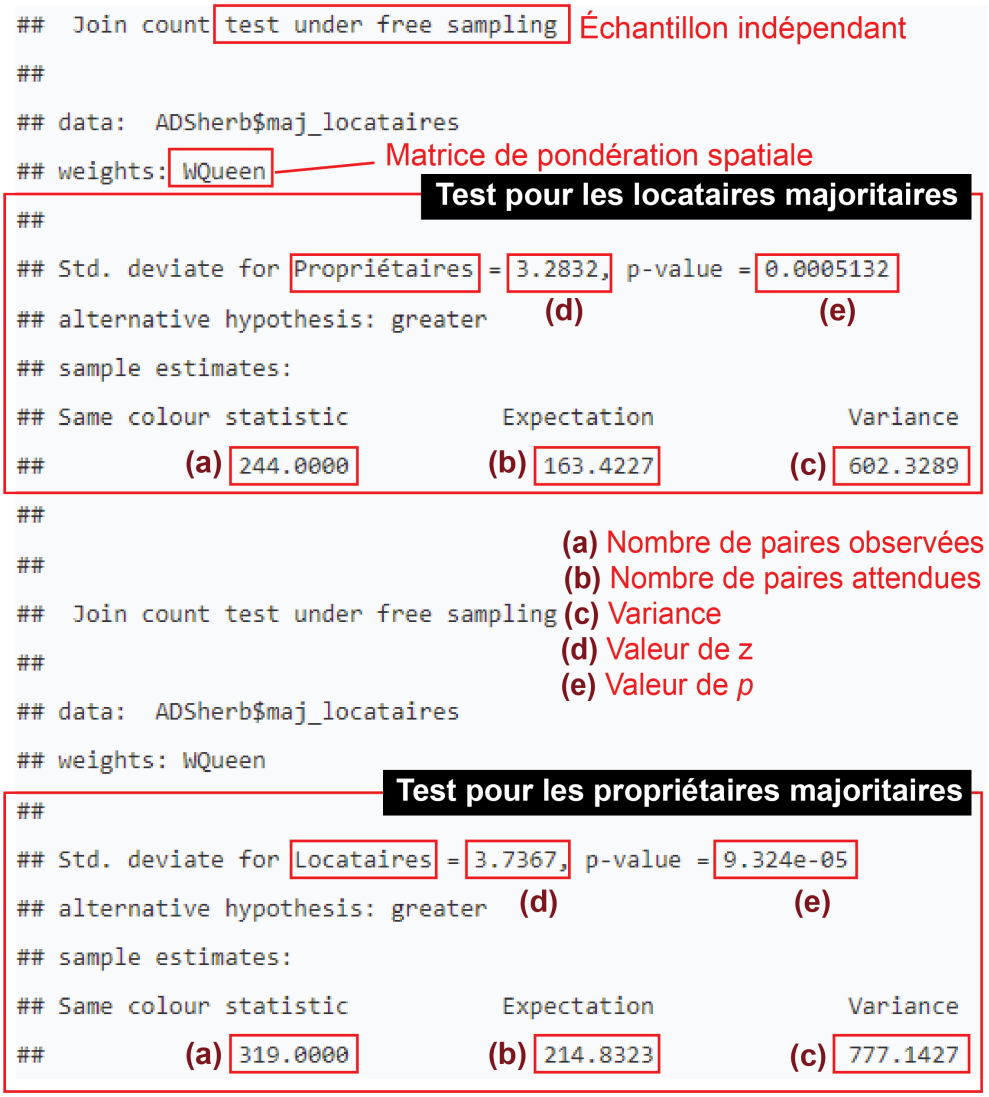


FIGURE 2.21 – Résultats des statistiques de comptage de jointure avec la fonction `joincount.test`

À des fins de comparaison, nous effectuons le calcul des statistiques de comptage de jointure avec l'approche d'inférence selon le test des permutations (999) avec la fonction `joincount.mc`.

```
test2 <- joincount.mc(ADSherb$maj_locataires, listw = WQueen, nsim = 999)
print(test2)
```

```
    Monte-Carlo simulation of join-count statistic

data:  ADSherb$maj_locataires
weights: WQueen
number of simulations + 1: 1000

Join-count statistic for Propriétaires = 244, rank of observed
statistic = 1000, p-value = 0.001
alternative hypothesis: greater
sample estimates:
    mean of simulation variance of simulation
              162.6106                112.1879


    Monte-Carlo simulation of join-count statistic

data:  ADSherb$maj_locataires
weights: WQueen
number of simulations + 1: 1000

Join-count statistic for Locataires = 319, rank of observed statistic =
1000, p-value = 0.001
alternative hypothesis: greater
sample estimates:
    mean of simulation variance of simulation
              214.3874                130.3638
```

L'approche d'inférence basée sur les permutations signale aussi que les deux distributions sont significativement autocorrélées spatialement (p = 0,001).

Aller plus loin

Fonction `joincount.multi`
Pour une variable qualitative comprenant plus de deux modalités (catégories), utilisez la fonction `joincount.multi` du *package* `spdep`. Cependant, celle-ci ne renvoie pas de valeurs de p, mais uniquement des valeurs de Z et des variances pour chaque modalité. Il est possible d'obtenir des valeurs de p en utilisant la fonction `pnorm` et surtout en ajustant ces valeurs de p pour contrôler l'effet de la multiplication des tests de significativité avec la fonction `p.adjust`.

2.3.3 Indice de Lee : autocorrélation spatiale dans un contexte bivarié

À la section 2.3.1, nous avons vu que le I de Moran permet d'évaluer le degré d'autocorrélation spatiale d'une variable continue, c'est-à-dire de vérifier si les entités spatiales proches ou voisines ont tendance à avoir des valeurs (dis)similaires pour une seule variable continue. Il est logique de vouloir étendre cette analyse à un contexte bivarié : est-ce que deux variables continues partagent ou non une relation marquée par de l'autocorrélation spatiale? En d'autres termes, l'association spatiale bivariée cherche à capturer la présence d'un patron spatial commun pour les deux variables (Lee 2001).

2.3.3.1 Formulation de l'indice de Lee

Il est important de distinguer la corrélation (non spatiale) entre deux variables et l'autocorrélation spatiale entre deux variables. La première mesure à quel point deux variables tendent à avoir une relation linéaire positive ou négative, alors que la seconde mesure la présence d'un patron spatial commun. Prenons l'exemple d'un jeu de données fictives avec 25 observations et deux variables continues X et Y (tableau 2.7).

Tableau 2.7 – Jeu de données fictives avec 25 observations et deux variables

X	Y
-2,67	-9,95
-1,22	-7,82
1,37	-3,14
-3,77	-11,73
1,75	3,13
-1,55	-0,95
-2,08	-11,88
1,18	9,57
1,14	5,48
0,55	4,03
0,17	-5,20
-2,71	-6,42
-0,48	2,14
0,80	-0,50
-2,39	-8,23
-0,61	-6,53
-1,25	-9,86
2,14	4,67
-0,37	3,30
3,19	11,36
1,46	7,17
-0,23	-6,66
-1,88	-8,17
-2,59	-9,09
-1,23	-5,42

Ces deux variables sont caractérisées par une forte corrélation linéaire positive avec un coefficient de corrélation de Pearson proche de 1 (figure 2.22).

Note

Retour sur la notion de corrélation entre deux variables continues.
Pour un rappel sur la notion de corrélation entre deux variables continues, notamment sur l'interprétation du coefficient de corrélation de Pearson, nous vous invitons à lire la section suivante (Apparicio et Gelb 2022).

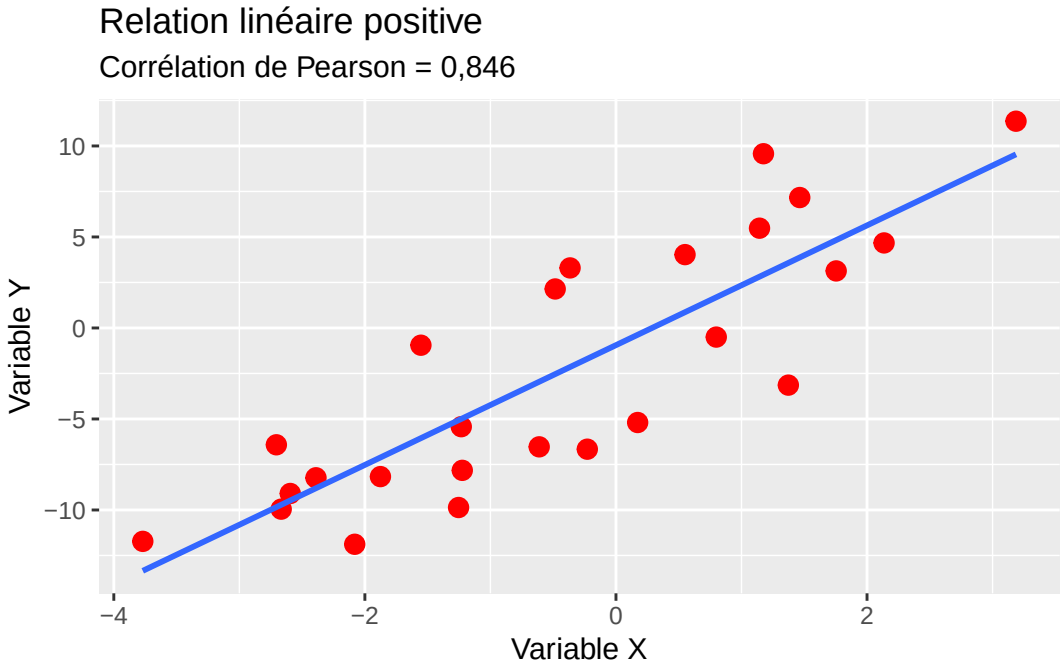


FIGURE 2.22 – Relation entre deux variables continues et coefficients de corrélation de Pearson

Cependant, cette information relative à la corrélation entre les deux variables ne nous permet pas d'appréhender leurs caractéristiques spatiales. En effet, ce tableau de données peut correspondre aux différents patrons spatiaux. En examinant les cartes **a** à **d** de la figure 2.23, nous constatons l'absence de tout patron spatial apparent. Par contre, à la figure 2.23 (e), nous observons une configuration spatiale particulière : à la fois, une forte autocorrélation spatiale positive univariée de chacune des deux variables (X et Y) et une forte autocorrélation spatiale positive bivariée puisque les deux patrons spatiaux de X et Y sont similaires.

Maintenant que nous avons fait la distinction entre la **corrélation bivariée** et **l'autocorrélation spatiale bivariée**, nous pouvons présenter un indicateur d'autocorrélation spatiale globale bivariée, soit l'indice de Lee (2001). Cet indicateur est construit à partir du produit de l'autocorrélation univariée de chacune des deux variables (numérateur) et de la corrélation de Pearson entre les versions spatialement décalées de ces variables (dénominateur). La formule de l'indicateur est la suivante :

$$L_{X,Y} = \frac{n}{\sum_i \left(\sum_j v_{ij}\right)^2} \cdot \frac{\sum_i \left[\left(\sum_j v_{ij}\left(x_j - \bar{x}\right)\right) \cdot \left(\sum_j v_{ij}\left(y_j - \bar{y}\right)\right)\right]}{\sqrt{\sum_i \left(x_i - \bar{x}\right)^2}\sqrt{\sum_i \left(y_i - \bar{y}\right)^2}} \tag{2.18}$$

Si la matrice de pondération spatiale est standardisée en ligne, l'indice de Lee est simplifié comme suit :

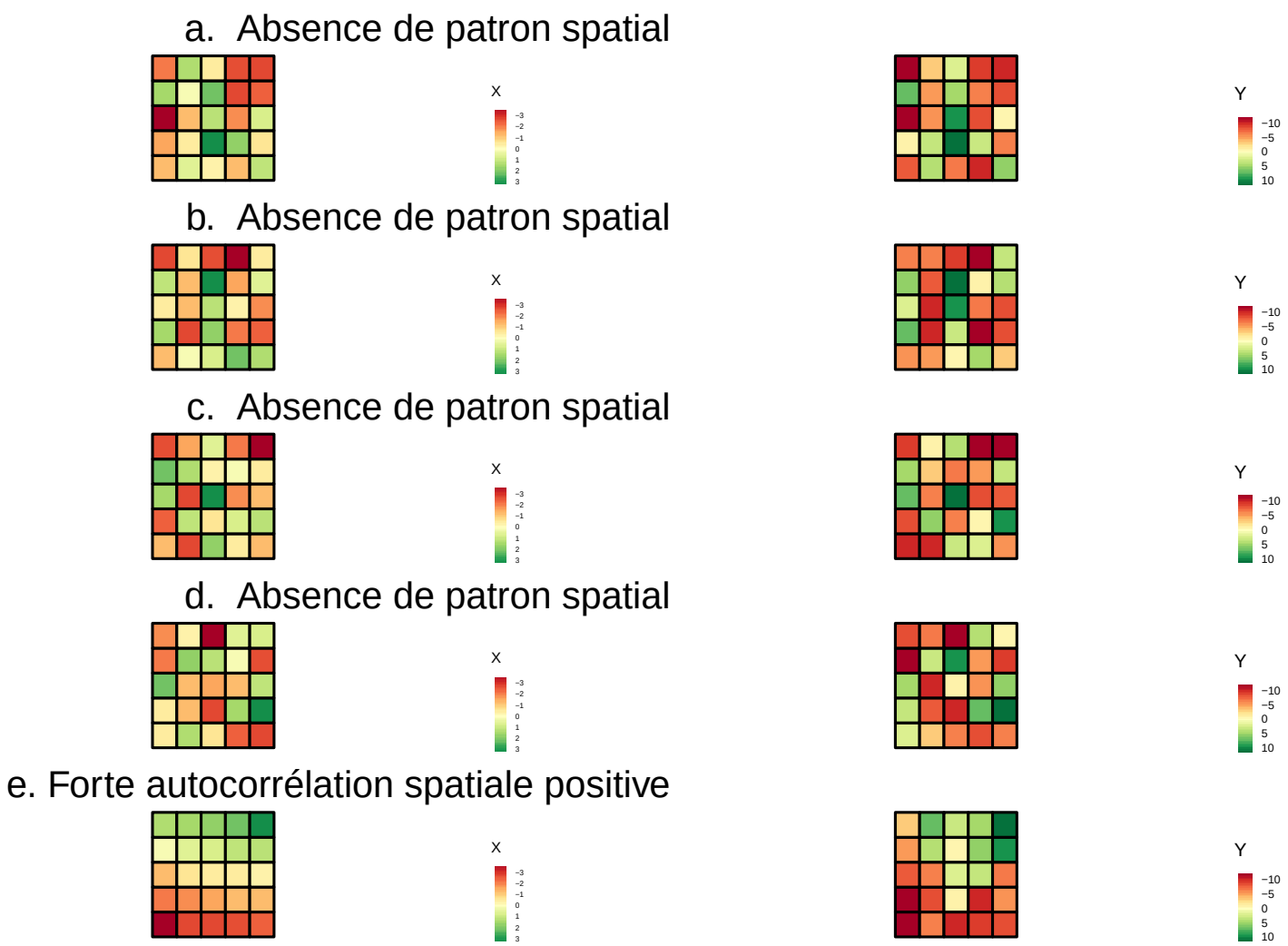


FIGURE 2.23 – Exemples de patrons spatiaux sans et avec autocorrélation spatiale dans un contexte bivarié

$$L_{X,Y} = \frac{\sum_i \left[\left(\sum_j v_{ij}\left(x_j - \bar{x}\right)\right) \cdot \left(\sum_j v_{ij}\left(y_j - \bar{y}\right)\right)\right]}{\sqrt{\sum_i \left(x_i - \bar{x}\right)^2}\sqrt{\sum_i \left(y_i - \bar{y}\right)^2}} \tag{2.19}$$

Cet indicateur varie entre -1 (autocorrélation spatiale bivariée négative) et +1 (autocorrélation bivariée positive).

Nous présentons trois cas à la figure 2.24 pour illustrer son interprétation. Prenons trois variables X, Y et X', avec X' étant l'inverse spatial de X. Pour simplifier l'exemple, ces trois variables ne peuvent avoir que trois valeurs (1, 2 ou 3).

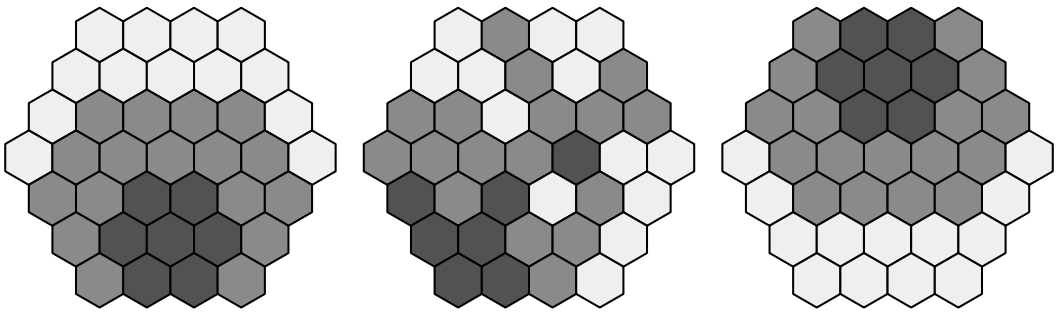


Figure 2.24 – Patrons spatiaux de X, Y et X'

Si nous calculons l'autocorrélation spatiale entre toutes les paires de variables avec une matrice de contiguïté standardisée, nous obtenons la matrice au tableau 2.8. Les résultats démontrent que les variables X et Y partagent un patron d'autocorrélation spatiale positive modérée ($0{,}327$), alors que les variables X et X' partagent un patron d'autocorrélation spatiale négative ($-0{,}512$). En d'autres termes, les endroits où X tend à avoir des regroupements d'observations avec des valeurs fortes, X' tend à avoir des regroupements d'observations avec des valeurs faibles.

Tableau 2.8 – Indice de Lee global

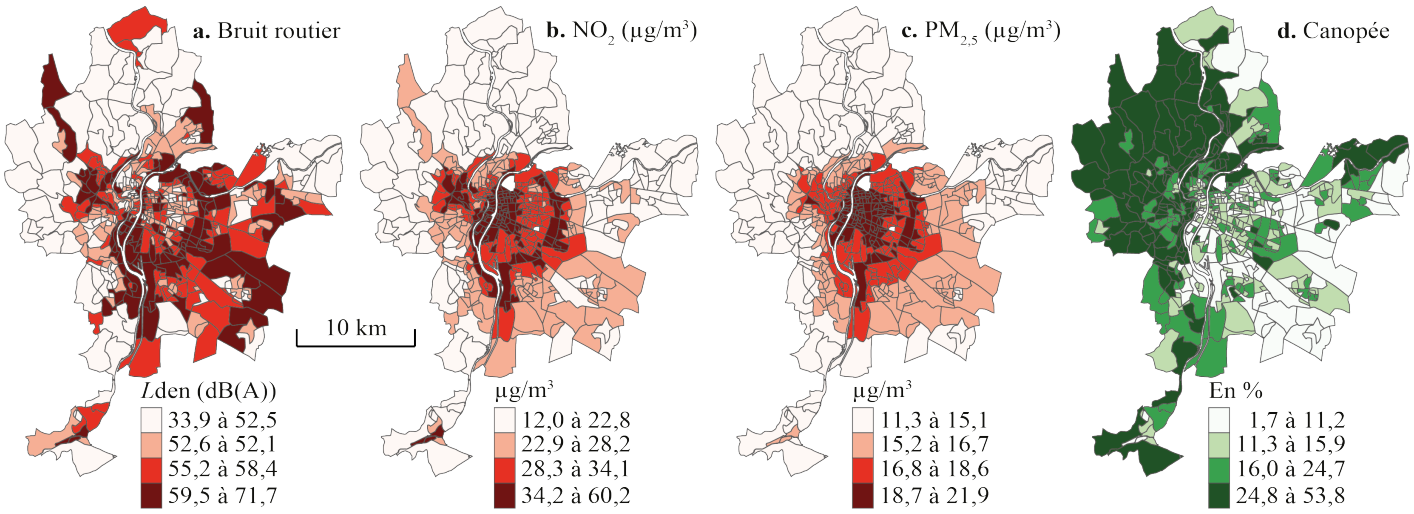
	X	Y	X'
X		0,327	-0,512
Y	0,327		-0,293
X'	-0,512	-0,293	

Astuce

Significativité du test de Lee
Au même titre que pour les indicateurs vus précédemment, il est possible de tester si l'indice de Lee obtenu est significativement différent de 0 avec des permutations Monte-Carlo.

2.3.3.2 Mise en œuvre dans R

Pour décrire le calcul de l'indice de Lee dans R, nous utilisons le jeu de données spatiales `LyonIris` du *package* `geocmeans` qui comprend quatre variables environnementales : le bruit routier (Lden dB(A)) (`Lden`), le dioxyde d'azote (ug/m^3) (`NO2`), les particules fines $PM_{2,5}$ (`PM25`) et la canopée (%) (`VegHautPrt`) pour les îlots regroupés pour l'information statistique (IRIS) de l'agglomération lyonnaise (figure 2.25).



Sources : **a.** Data Grand Lyon, 2010; **b et c.** Atmo Auvergne-Rhône-Alpes, 2015; **d.** Plateforme ouverte des données publiques françaises, 2015.
Conception et réalisation : Jérémy Gelb et Philippe Apparicio, 2021.

FIGURE 2.25 – Cartographie des variables du jeu de données LyonIris

Il est facile de calculer l'indice de Lee en utilisant les fonctions `lee.test(x, y, listw)`,`lee.mc(x, y, listw)` ou `lee(x, y, listw, n)` du package `spdep`. Par exemple, le code ci-dessous permet de le calculer pour les variables `Lden` et `NO2`.

```
library(geocmeans)
data("LyonIris")
## Matrice de pondération spatiale de contiguïté standardisée
nb <- poly2nb(LyonIris, queen = TRUE)
Wmat <- nb2listw(nb, style = 'W')
## Calcul de l'indice de Lee entre les deux variables
lee.test(LyonIris$Lden, LyonIris$NO2, listw = Wmat)
```

```
    Lee's L statistic randomisation

data:  LyonIris$Lden ,  LyonIris$NO2
weights: Wmat

Lee's L statistic standard deviate = 17.135, p-value < 2.2e-16
alternative hypothesis: greater
sample estimates:
```

```
Lee's L statistic       Expectation          Variance
     0.4072785543      0.1243825152      0.0002725625
```

```
lee.mc(LyonIris$Lden, LyonIris$NO2, listw = Wmat, nsim = 999)
```

```


    Monte-Carlo simulation of Lee's L

data:  LyonIris$Lden ,  LyonIris$NO2
weights: Wmat
number of simulations + 1: 1000

statistic = 0.40728, observed rank = 1000, p-value = 0.001
alternative hypothesis: greater
```

Si vous analysez plusieurs variables simultanément, il est pertinent de construire à la fois une matrice de corrélation et une matrice d'autocorrélation spatiale bivariée. Pour cette dernière, la diagonale peut être remplacée par le I de Moran de chaque variable.

```
## Liste des variables à analyser
vars <- c("Lden","NO2","PM25","VegHautPrt")
## Calcul de la matrice de corrélation de Pearson
corr_mat <- cor(st_drop_geometry(LyonIris)[vars])
## Calcul d'une matrice d'autocorrélation spatiale bivariée (indice de Lee)
bivar_autocor_mat <- sapply(vars, function(x1){
  row_values <- sapply(vars, function(x2){
    test <- lee(LyonIris[[x1]], LyonIris[[x2]], listw = Wmat, n = nrow(LyonIris))
    return(test$L)
  })
})
# Remplacement de la diagonale par les valeurs du I de Moran
moranIs <- sapply(vars, function(x){
  moran(LyonIris[[x]], Wmat, n = nrow(LyonIris), S0 = nrow(LyonIris))[[1]]
})
diag(bivar_autocor_mat) <- moranIs
```

Tableau 2.9 – Matrice de corrélation de Pearson

	Lden	NO2	PM25	VegHautPrt
Lden	1,000	0,623	0,489	-0,227
NO2	0,623	1,000	0,901	-0,283
PM25	0,489	0,901	1,000	-0,392
VegHautPrt	-0,227	-0,283	-0,392	1,000

Sans surprise, les différents polluants sont corrélés positivement entre eux et moyennement corrélés négativement avec le couvert végétal. La corrélation la plus forte s'observe entre le dioxyde d'azote et les particules fines (r = 0,901; tableau 2.9).

La lecture de la diagonale du tableau 2.10 permet de constater que les deux polluants atmosphériques sont les plus fortement spatialement autocorrélés (I de Moran de 0,82 et 0,94). Leur valeur de l'indice de Lee est aussi très forte (0,79), indiquant qu'ils tendent à varier de façon conjointe dans l'espace. En revanche, l'autocorrélation spatiale entre le bruit environnemental (Lden) et le dioxyde d'azote (indice de Lee de 0,41) est à peine plus forte que celle entre le Lden et les particules fines (indice de Lee de 0,38) alors que l'écart de la corrélation entre ces paires de variables est bien plus important (respectivement de 0,62 et 0,49). La corrélation non spatiale marquée entre le bruit et le dioxyde d'azote ne se traduit que par une autocorrélation spatiale bivariée modérée (0,41). La géographie de ces deux pollutions ne peut donc se résumer à un patron commun.

Tableau 2.10 – Matrice d'autocorrélation spatiale globale bivariée (indice de Lee)

	Lden	NO2	PM25	VegHautPrt
Lden	0,561	0,407	0,379	-0,231
NO2	0,407	0,819	0,789	-0,283
PM25	0,379	0,789	0,935	-0,412
VegHautPrt	-0,231	-0,283	-0,412	0,585

2.4 Autocorrélation spatiale locale

Nous avons vu que les statistiques d'autocorrélation spatiale globale comme le I de Moran (section 2.3.1), les statistiques de comptage de jointures (section 2.3.2) et l'indice de Lee (section 2.3.3) renvoient une valeur pour l'ensemble de l'espace d'étude. Une fois démontrée la présence d'autocorrélation spatiale globale (positive ou négative), il est pertinent de réaliser une analyse de l'autocorrélation spatiale locale afin de vérifier si chaque entité spatiale est significativement (dis)semblable de celles voisines ou proches. Comme l'indique l'adjectif *locale*, les mesures d'autocorrélation spatiale locale renvoient des valeurs pour chacune des entités spatiales.

2.4.1 Statistiques locales de Getis et Ord : repérer les points chauds et froids pour une variable continue

2.4.1.1 Formulation des statistiques locale de Getis et Ord

Les statistiques locales de Getis et Ord permettent d'évaluer la similarité d'une entité spatiale avec celles voisines ou proches en fonction **d'une variable continue** (Getis et Ord 1992; Ord et Getis 1995). Autrement dit, elles nous informent si les valeurs fortes et les valeurs faibles d'une variable continue se regroupent significativement dans l'espace, et ce, avec n'importe quel type de matrice de contiguïté, de proximité, de plus proches voisins, etc. Cartographier ces statistiques permet alors de vérifier simultanément l'existence d'agrégats spatiaux de valeurs fortes (**points chauds**) et d'agrégats spatiaux de valeurs faibles (**points froids**).

Il existe deux versions légèrement différentes de ces mesures locales (Getis et Ord 1992; Ord et Getis 1995) :

1. G_i tient compte uniquement des entités voisines ou proches à l'entité spatiale i. Ainsi, $\Sigma_{j=1}^{n} w_{ij}x_j$ représente la moyenne pondérée (par les poids de la matrice de pondération spatiale standardisée en ligne) de la variable continue X dans les entités voisines ou proches.
2. G_i^* tient compte à la fois des valeurs des entités voisines ou proches, mais aussi de celle de i.

Toutefois, nous cartographions rarement les valeurs de G_i et de G_i^*, mais plutôt celles de $Z(G_i)$ et de $Z(G_i^*)$ (équation 2.20 et équation 2.21) qui représentent la cote Z qui, lorsque positive, indique un agrégat de valeurs plus élevées que la moyenne, et qui, lorsque négative, indique un agrégat de valeurs plus faibles que la moyenne (Bivand et Wong 2018).

À première vue, ces deux formules peuvent sembler quelque peu complexes! Retenez simplement que le numérateur est la différence entre G_i ou G_i^* et la valeur attendue de G_i ou de G_i^* pour une distribution aléatoire (soit globalement la moyenne de la variable), tandis que le dénominateur représente l'écart-type de G_i ou de G_i^*.

$$Z(G_i) = \frac{\left[\Sigma_{j=1}^{n} w_{ij}x_j\right] - \left[(\Sigma_{j=1}^{n} w_{ij})\bar{x}_i\right]}{s_i\sqrt{\left[\left((n-1)\Sigma_{j=1}^{n} w_{ij}^2 - (\Sigma_{j=1}^{n} w_{ij})^2\right)\right]/(n-1)}} = \frac{G_i - \mathbb{E}(G_i)}{\sqrt{Var(G_i)}}, i \neq j \tag{2.20}$$

$$\text{avec } \bar{x}_i = \frac{\Sigma_{j=1}^{n} x_j}{n-1}, \text{ et } s_i = \sqrt{\frac{\Sigma_{j=1}^{n} x_j^2}{n-1} - \bar{x}^2}, i \neq j$$

$$Z(G_i^*) = \frac{\left[\Sigma_{j=1}^{n} w_{ij}x_j\right] - \left[(\Sigma_{j=1}^{n} w_{ij})\bar{x}^*\right]}{s^*\sqrt{\left[\left((n-1)\Sigma_{j=1}^{n} w_{ij}^2 - (\Sigma_{j=1}^{n} w_{ij})^2\right)\right]/(n-1)}} = \frac{G_i^* - \mathbb{E}(G_i^*)}{\sqrt{Var(G_i^*)}}, \text{ tous} \tag{2.21}$$

$$\text{avec } \bar{x}^* = \frac{\Sigma_{j=1}^{n} x_j}{n}, \text{ et } s^* = \sqrt{\frac{\Sigma_{j=1}^{n} x_j^2}{n} - \bar{x}^{*2}}, \text{ tous } j$$

avec :

- n, le nombre d'entités spatiales dans la couche géographique;
- w_{ij}, la valeur de la pondération spatiale entre les entités spatiales i et j;
- x_j, la valeur de variable continue X pour l'entité spatiale j;
- $\bar{x}^*$, la valeur moyenne de la variable pour toutes les observations;
- $\bar{x}_i$, la valeur moyenne de la variable pour toutes les observations sauf i.

Astuce

Deux manières de calculer $Z(G_i)$ et $Z(G_i^*)$ avec le *package* `spdep`

- la fonction `localG` vous renvoie les valeurs de cote Z (*Z-score* en anglais) des statistiques de Getis et Ord avec les formules décrites plus haut.
- la fonction `localG_perm` permet de les obtenir avec la méthode Monte-Carlo (avec habituellement 999 permutations).

Cartographie des cotes Z

À partir des cotes Z, utilisez les bornes des classes suivantes et la palette de couleurs `-RdBu` :

- Minimum à -3,29 : point froid significatif avec p < 0,001 (bleu foncé).
- -3,29 à -2,58 : point froid significatif avec p < 0,01 (bleu).
- -2,58 à -1,96 : point froid significatif avec p < 0,05 (bleu pâle).
- -1,96 à 1,96 : non significatif (gris).
- 1,96 à 2,58 : point chaud significatif avec p < 0,05 (rouge pâle).
- 2,58 à 3,29 : point chaud significatif avec p < 0,01 (rouge).
- 3,29 à Maximum : point chaud significatif avec *p <0 ,001 (rouge foncé).

Avec la palette `-RdBu`, les points froids et les points chauds sont respectivement bleus et rouges comme les pastilles de votre robinet! Bref, un beau travail de plomberie!

Pour un rappel sur la cote Z et les valeurs de p associées, consultez ce lien.

2.4.1.2 Mise en œuvre dans R

La syntaxe ci-dessous calcule les deux statistiques locales avec `localG` et `localG_perm` pour le revenu moyen des ménages pour les aires de diffusion de 2021 de la ville de Sherbrooke.

```
Rook1 <- poly2nb(AD.DR, queen=FALSE)
## Matrice de pondération
W.RookG    <- nb2listw(Rook1, zero.policy=TRUE)
W.RookStar <-nb2listw(include.self(Rook1), zero.policy=TRUE) # matrice incluant elle-même
## Calcul du Z(Gi) et du Z(Gi*)
localGetis     <- localG(AD.DR$RevMedMenage, W.RookG)
localGetisStar <- localG(AD.DR$RevMedMenage, W.RookStar)
## Calcul avec la méthode Monte-Carlo (999 permutations)
localGetis.MC999     <- localG_perm(AD.DR$RevMedMenage, W.RookG, nsim = 999)
localGetis.StarMC999 <- localG_perm(AD.DR$RevMedMenage, W.RookStar, nsim = 999)
## Sommaires statistiques
summary(localGetis)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-3.80885 -1.70432 -0.09828 -0.04120  1.41123  3.91835
```

```
summary(localGetisStar)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-4.04870 -1.88689 -0.13297 -0.03841  1.54641  4.10379
```

```
summary(localGetis.MC999)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-3.80885 -1.70432 -0.09828 -0.04120  1.41123  3.91835
```

```
summary(localGetis.StarMC999)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-4.04870 -1.88689 -0.13297 -0.03841  1.54641  4.10379
```

Cartographions ces valeurs et repérons les regroupements de valeurs significativement fortes et faibles avec les statistiques de Getis et Ord (figure 2.26 et figure 2.27). Les quatre cartes sont très semblables et permettent de repérer un regroupement de valeurs faibles dans le centre-ville et un autre de valeurs fortes dans l'est de la ville.

```
## Enregistrement des résultats dans deux nouveaux champs de la couche des AD
AD.DR$RevMed_localGetis <- localGetis
AD.DR$RevMed_localGetisStar <- localGetisStar
# Définition des intervalles et des noms des classes
classes.intervalles = c(-Inf, -3.29, -2.58, -1.96, 1.96, 2.58, 3.29, Inf)
classes.noms = c("Point froid (p = 0,001)",
                 "Point froid (p = 0,01)",
                 "Point froid (p = 0,05)",
                 "Non significatif",
                 "Point chaud (p = 0,05)",
                 "Point chaud (p = 0,01)",
                 "Point chaud (p = 0,001)")
# Création d'un champ avec les noms des classes
AD.DR$RevMed_localGetisP <- cut(AD.DR$RevMed_localGetis,
                                breaks = classes.intervalles,
                                labels = classes.noms)
AD.DR$RevMed_localGetisStarP <- cut(AD.DR$RevMed_localGetisStar,
                                  breaks = classes.intervalles,
                                  labels = classes.noms)
## Cartographie pour le Z(Gi)
Carte1 = tm_shape(AD.DR)+
            tm_polygons(col ="RevMed_localGetisP",
                        title="Z(Gi)", palette="-RdBu", lwd = 1)+
            tm_layout(frame =FALSE)
## Cartographie pour le Z(Gi)*
Carte2 = tm_shape(AD.DR)+
            tm_polygons(col ="RevMed_localGetisStarP",
                        title="Z(Gi*)",  palette="-RdBu", lwd = 1)+
            tm_layout(frame =FALSE)+
          tm_credits("Auteurs : Geoffroy et Jessie Chaux.",
```

```
          position = c("right", "bottom"), size = 0.7, align = "right")
## Composition avec les deux cartes
tmap_arrange(Carte1, Carte2, ncol = 2, nrow = 1)
```

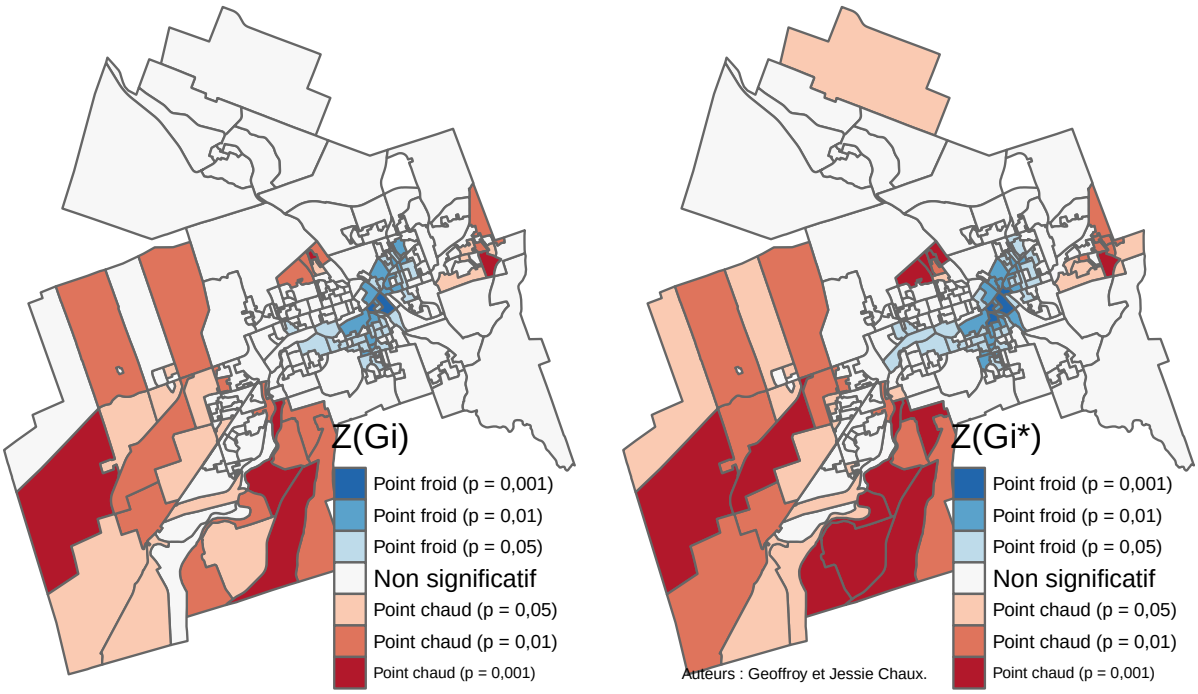


FIGURE 2.26 – Points chauds et froids du revenu médian des ménages selon les statistiques de Getis et Ord (méthode classique)

```
## Enregistrement les résultats dans deux champs de la couche des AD
AD.DR$RevMed_localGetis.MC999 <- localGetis.MC999
AD.DR$RevMed_localGetis.StarMC999 <- localGetis.StarMC999
# Définition des intervalles et des noms des classes
classes.intervalles = c(-Inf, -3.29, -2.58, -1.96, 1.96, 2.58, 3.29, Inf)
classes.noms = c("Point froid (p = 0,001)",
                 "Point froid (p = 0,01)",
                 "Point froid (p = 0,05)",
                 "Non significatif",
                 "Point chaud (p = 0,05)",
                 "Point chaud (p = 0,01)",
                 "Point chaud (p = 0,001)")
# Création d'un champ avec les noms des classes
AD.DR$RevMed_localGetis.MC999P <- cut(AD.DR$RevMed_localGetis.MC999,
                              breaks = classes.intervalles,
                              labels = classes.noms)
AD.DR$RevMed_localGetis.StarMC999P <- cut(AD.DR$RevMed_localGetis.StarMC999,
                                breaks = classes.intervalles,
                                labels = classes.noms)
```

```
## Cartographie pour le Z(Gi) Mont-Carlo
Carte3 = tm_shape(AD.DR)+
            tm_polygons(col ="RevMed_localGetis.MC999P",
                        title="Z(Gi)", palette="-RdBu", lwd = 1)+
            tm_layout(frame =FALSE)
## Cartographie pour le Z(Gi)* Mont-Carlo
Carte4 = tm_shape(AD.DR)+
            tm_polygons(col ="RevMed_localGetis.StarMC999P",
                        title="Z(Gi*)", palette="-RdBu", lwd = 1)+
            tm_layout(frame =FALSE)+
          tm_credits("Auteurs : Geoffroy et Jessie Chaux.",
           position = c("right", "bottom"), size = 0.7, align = "right")
## Composition avec les deux cartes
tmap_arrange(Carte3, Carte4, ncol = 2, nrow = 1)
```

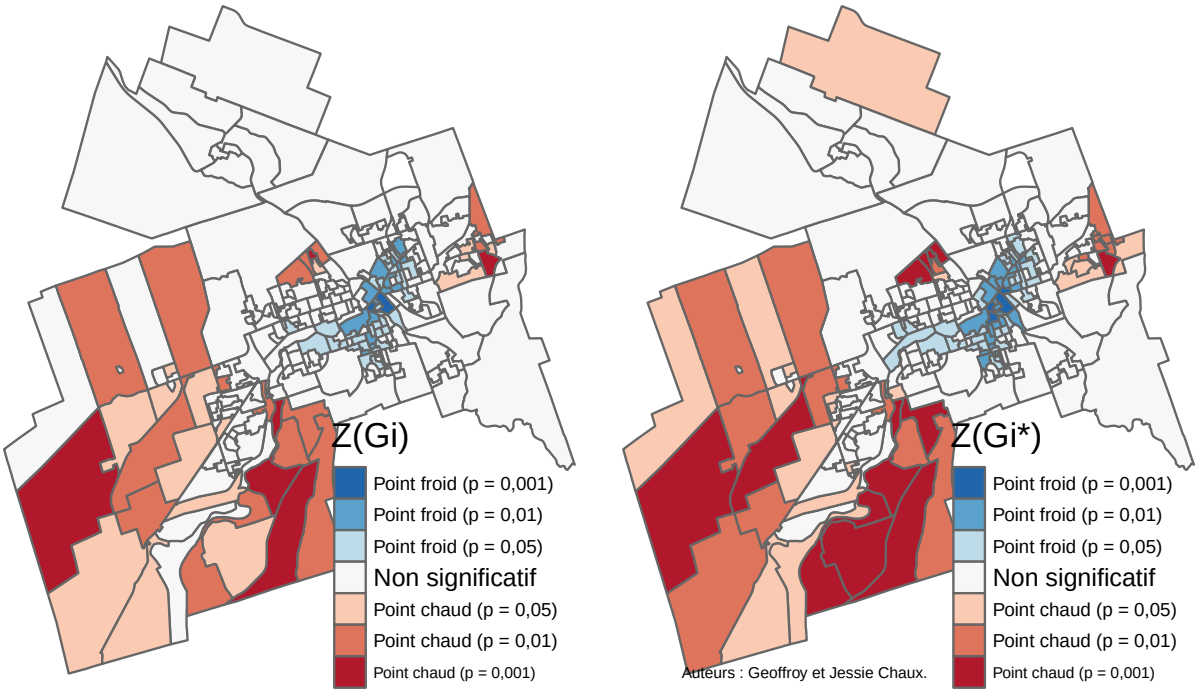


FIGURE 2.27 – Points chauds et froids du revenu médian des ménages selon les statistiques de Getis et Ord (999 permutations Monte-Carlo)

2.4.2 Version locale du I de Moran

2.4.2.1 Formulation de la version locale du I de Moran

Une version locale du I de Moran (I_i) a été proposée par Luc Anselin (1995). Elle permet de vérifier si une entité spatiale est voisine ou proche – dépendamment de la matrice spatiale utilisée – d'entités spatiales avec des valeurs semblables

(contexte d'autocorrélation spatiale locale positive) ou dissemblables (contexte d'autocorrélation spatiale locale négative). Le I de Moran local s'écrit :

$$I_i = \frac{(x_i - \bar{X})\Sigma_{j=1}^{n} w_{ij}(x_j - \bar{X})}{\frac{1}{n}\Sigma_{i=1}^{n}(x_i - \bar{X})^2} = \frac{z_i \Sigma_{j=1}^{n} z_j}{\frac{1}{n}\Sigma_{i=1}^{n} z_i^2}, i \neq j \tag{2.22}$$

avec :

- z_i étant la valeur de la variable continue centrée X pour l'entité spatiale i, c'est-à-dire simplement l'écart de i à la moyenne de X ($x_i - \bar{X}$).
- z_j étant la valeur de la variable centrée de X pour l'entité spatiale j.
- w_{ij} étant la valeur de la matrice de pondération spatiale standardisée en ligne entre i et j.
- n étant le nombre d'entités spatiales dans la couche géographique.

Comme pour la version globale, il est possible de tester la significativité du I de Moran local de manière classique (selon la loi normale) ou avec l'approche Monte-Carlo (avec habituellement 999 permutations).

Aller plus loin

Test de significativité du *I* de Moran local.
Pour comprendre les différentes variantes pour tester la significativité (Anselin 1995; Sokal, Oden et Thomson 1998), consultez Bivand et Wong (2018) ou l'ouvrage de Dubé et Legros (2014).

2.4.2.2 Mise en œuvre dans R

Le calcul du I de Moran local s'opère avec les fonctions `localmoran` et `localMoranI.mc` du *package* `spdep` (voir la syntaxe ci-dessous). Comme pour la version globale, les résultats du I de Moran local sont les mêmes pour les deux fonctions, seules les valeurs de p peuvent varier.

```
## Calcul du I de Moran local
localMoranI <- localmoran(AD.DR$RevMedMenage,  # variable
                          W.RookG)             # matrice de pondération spatiale
## Calcul du I de Moran local avec la méthode Monte-Carlo
localMoranI.mc <- localmoran_perm(AD.DR$RevMedMenage, # variable
                              W.RookG,                # matrice de pondération spatiale
                              nsim = 999)             # nombre de permutations
## Sommaires statistiques
summary(localMoranI)
```

```
       Ii                E.Ii                Var.Ii               Z.Ii
 Min.   :-0.90053   Min.   :-3.054e-02   Min.   :0.0000003   Min.   :-1.9962
 1st Qu.: 0.08313   1st Qu.:-5.368e-03   1st Qu.:0.0370546   1st Qu.: 0.4539
 Median : 0.49029   Median :-2.755e-03   Median :0.1339044   Median : 1.4404
 Mean   : 0.61678   Mean   :-4.032e-03   Mean   :0.2149974   Mean   : 1.2503
 3rd Qu.: 0.97602   3rd Qu.:-8.287e-04   3rd Qu.:0.2510284   3rd Qu.: 2.1938
 Max.   : 3.37625   Max.   :-8.000e-09   Max.   :1.8207164   Max.   : 3.9184
```

```
 Pr(z != E(Ii))
 Min.   :0.0000892
 1st Qu.:0.0282519
 Median :0.1268505
 Mean   :0.2467939
 3rd Qu.:0.3948293
 Max.   :0.9833048
```

```
summary(localMoranI.mc)
```

```
       Ii                E.Ii                Var.Ii               Z.Ii
 Min.   :-0.90053   Min.   :-0.060837   Min.   :0.0000003   Min.   :-1.8763
 1st Qu.: 0.08313   1st Qu.:-0.010606   1st Qu.:0.0378922   1st Qu.: 0.4225
 Median : 0.49029   Median :-0.002015   Median :0.1328532   Median : 1.3861
 Mean   : 0.61678   Mean   :-0.003963   Mean   :0.2176148   Mean   : 1.2407
 3rd Qu.: 0.97602   3rd Qu.: 0.002572   3rd Qu.:0.2590818   3rd Qu.: 2.1572
 Max.   : 3.37625   Max.   : 0.052074   Max.   :1.9288455   Max.   : 3.8750
 Pr(z != E(Ii))      Pr(z != E(Ii)) Sim Pr(folded) Sim      Skewness
 Min.   :0.0001066   Min.   :0.0020     Min.   :0.0010   Min.   :-0.462415
 1st Qu.:0.0309869   1st Qu.:0.0280     1st Qu.:0.0140   1st Qu.:-0.213973
 Median :0.1261421   Median :0.1320     Median :0.0660   Median :-0.081765
 Mean   :0.2500672   Mean   :0.2544     Mean   :0.1273   Mean   :-0.008389
 3rd Qu.:0.3937860   3rd Qu.:0.3960     3rd Qu.:0.1980   3rd Qu.: 0.208587
 Max.   :0.9777869   Max.   :0.9980     Max.   :0.4990   Max.   : 0.441534
    Kurtosis
 Min.   :-0.59003
 1st Qu.:-0.26012
 Median :-0.15362
 Mean   :-0.15894
 3rd Qu.:-0.05464
 Max.   : 0.34863
```

Avec la cartographie du I de Moran local, nous repérons localement l'autocorrélation spatiale positive (valeurs similaires, fortes ou faibles localement) et l'autocorrélation spatiale négative (valeurs dissemblables localement) (figure 2.28).

```
## Ajout de champs pour le I de Moran local
AD.DR$RevMed_IlocalMoran.MC999 <- localMoranI.mc[, 1]
AD.DR$RevMed_IlocalMoran.MC999p <- localMoranI.mc[, 5]
## Cartographie
tmap_mode("plot")
Carte1 = tm_shape(AD.DR)+
           tm_polygons(col ="RevMed_IlocalMoran.MC999", title="I Moran local",
                       style="cont", palette="-RdBu", lwd = 1)+
         tm_layout(frame = FALSE)
```

```
Carte2 = tm_shape(AD.DR)+
              tm_polygons(col= "RevMed_IlocalMoran.MC999p", title="valeur de p",
                    palette = c("darkgreen", "green", "lightgreen", "gray"),  lwd = 1,
                    breaks = c(0, 0.001, 0.01, 0.05, Inf),
                    legend.format = list(text.separator = "à"),
                    title ="En %")+
            tm_layout(frame = FALSE)
## Combinaison des deux cartes
tmap_arrange(Carte1, Carte2, ncol = 2, nrow = 1)
```

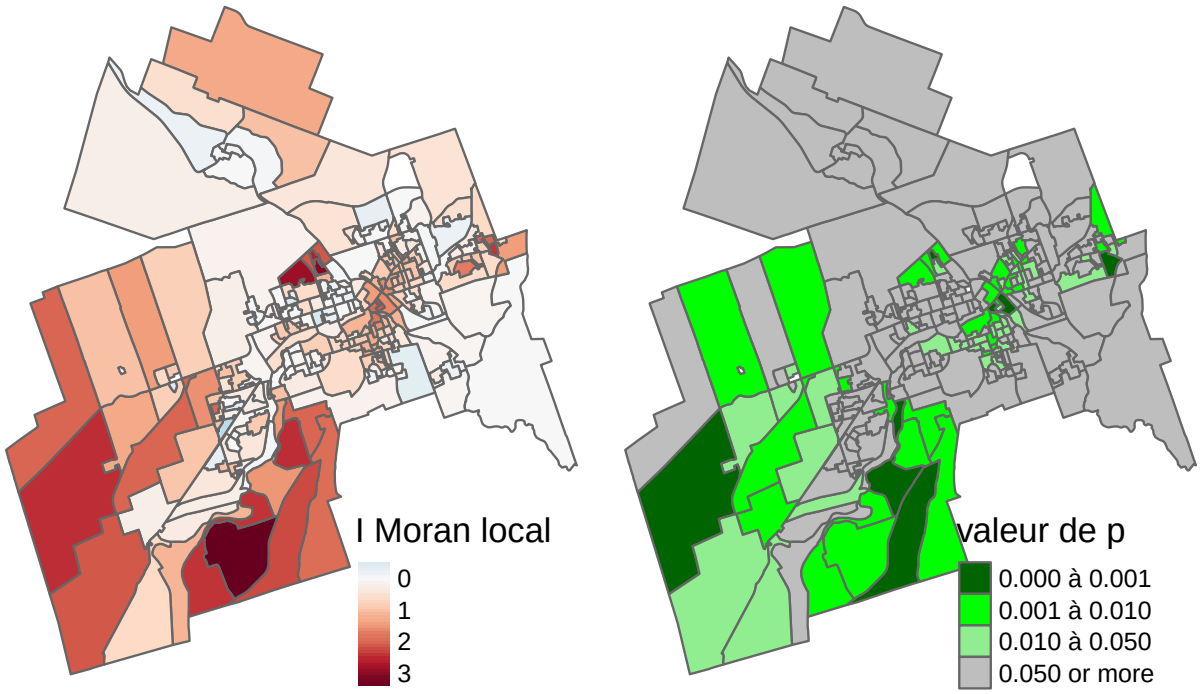


FIGURE 2.28 – Cartographie du I de Moran local et de la valeur de p associée

2.4.3 Typologie basée sur le diagramme de Moran

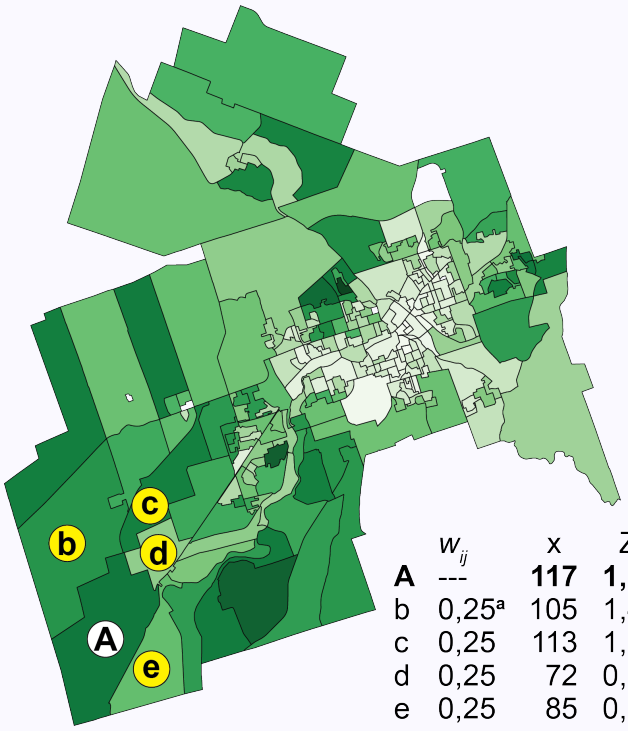
2.4.3.1 Formulation de la typologie basée sur le diagramme de Moran dans un contexte univarié

La typologie basée sur le diagramme de Moran a été proposée par Luc Anselin (1996). L'idée est fort simple, mais extrêmement efficace! Avant tout, la variable continue est centrée réduite (cote z, *z-score* en anglais). Pour un rappel sur la cote z, consultez la section suivante (Apparicio et Gelb 2022). La moyenne est ainsi égale à 0 et l'écart-type à 1. Puis, il s'agit de construire un nuage de points entre :

1. Les valeurs de la variable centrée réduite (Z) **sur l'axe des X** pour les entités spatiales de la couche géographique.
2. Les valeurs de la variable centrée réduite spatialement décalée obtenues avec les pondérations spatiales de la matrice W (voir l'encadré ci-dessous).

Note

Comment calculer une variable spatialement décalée avec une matrice de pondération spatiale?
À la figure 2.29, nous détaillons le calcul de la valeur d'une variable spatialement décalée pour l'entité spatiale **A** à partir d'une matrice de contiguïté (selon le partage d'un segment) standardisée en ligne. Notez que **A** est adjacente à quatre entités spatiales (b, c, d et e).



	w_{ij}	x	Zx	W.x	W.Zx
A	---	**117**	**1,96**[b]		
b	0,25[a]	105	1,49	26,25[c]	0,37[d]
c	0,25	113	1,81	28,25	0,45
d	0,25	72	0,20	18,00	0,05
e	0,25	85	0,71	21,25	0,18
				93,75	**1,05**

Moyenne = 66,96. Écart-type = 25,50.
[a] w_{ij} = poids standardisés en ligne de la matrice de contiguïté, soit 1/4 voisins.
[b] cote z = (117 − 66,96) / 25,50 = 1,96
[c] 105 x 0,25 = 26,25 [d] 1,49 x 0,25 = 0,37
$W.x_a$ = 93,75, soit la moyenne de X pour les entités spatiales adjacentes à A
$W.Zx_a$ = 1,05, soit la moyenne de la cote Z pour les entités spatiales adjacentes à A

FIGURE 2.29 – Illustration du calcul d'une variable spatialement décalée

Dans R, la syntaxe est fort simple pour créer une cote z et une variable spatiale décalée :

```
zx <- (x - mean(x))/sd(x)     # variable X centrée réduite (cote z)
wzx <- lag.listw(listW,zx)    # variable X centrée réduite spatialement décalée
```

Analysons les différents éléments du diagramme de Moran à la figure 2.30 :

— La droite de régression résume la relation linéaire entre la variable spatialement décalée ($W.ZX$) et l'originale (ZX). D'ailleurs, le coefficient de régression pour la variable ZX, soit la pente de la droite, équivaut au I de Moran!
— Les traits pointillés représentent les moyennes des deux variables, toutes deux égales à 0 puisqu'elles sont

centrées-réduites (cote z).

Pour analyser ce nuage, nous le décomposons en quatre quadrants :

1. Le quadrant **HH (*High-High*)** en haut à droite regroupe des entités spatiales avec des valeurs **fortes** (H) qui sont voisines ou proches d'autres entités spatiales avec aussi des valeurs **fortes** (H). Nous sommes donc en présence **d'autocorrélation spatiale locale positive avec des valeurs fortes (HH)**.
2. Le quadrant **LL (*Low-Low*)** en bas à gauche regroupe des entités spatiales avec des valeurs **faibles** (L) qui sont voisines ou proches d'autres entités spatiales avec aussi des valeurs **faibles** (L). Nous sommes donc en présence **d'autocorrélation spatiale locale positive avec des valeurs faibles (LL)**.
3. Le quadrant **HL (*High-Low*)** en bas à droite regroupe des entités spatiales avec des valeurs **fortes** (H) qui sont voisines ou proches d'autres entités spatiales avec des valeurs **faibles** (L). Nous sommes donc en présence **d'autocorrélation spatiale locale négative (HL)**. Avec humour, un collègue économiste, Jean Dubé, qualifie ce quadrant de **village gaulois** (Obélix, Astérix et leurs compagnons sont très forts et ils sont entourés de voisins romains plus faibles…).
4. Le quadrant **LH (*Low-High*)** en haut à gauche regroupe des entités spatiales avec des valeurs **faibles** (L) qui sont voisines ou proches d'autres entités spatiales avec des valeurs **fortes** (H). Nous sommes donc en présence **d'autocorrélation spatiale locale négative (LH)**.

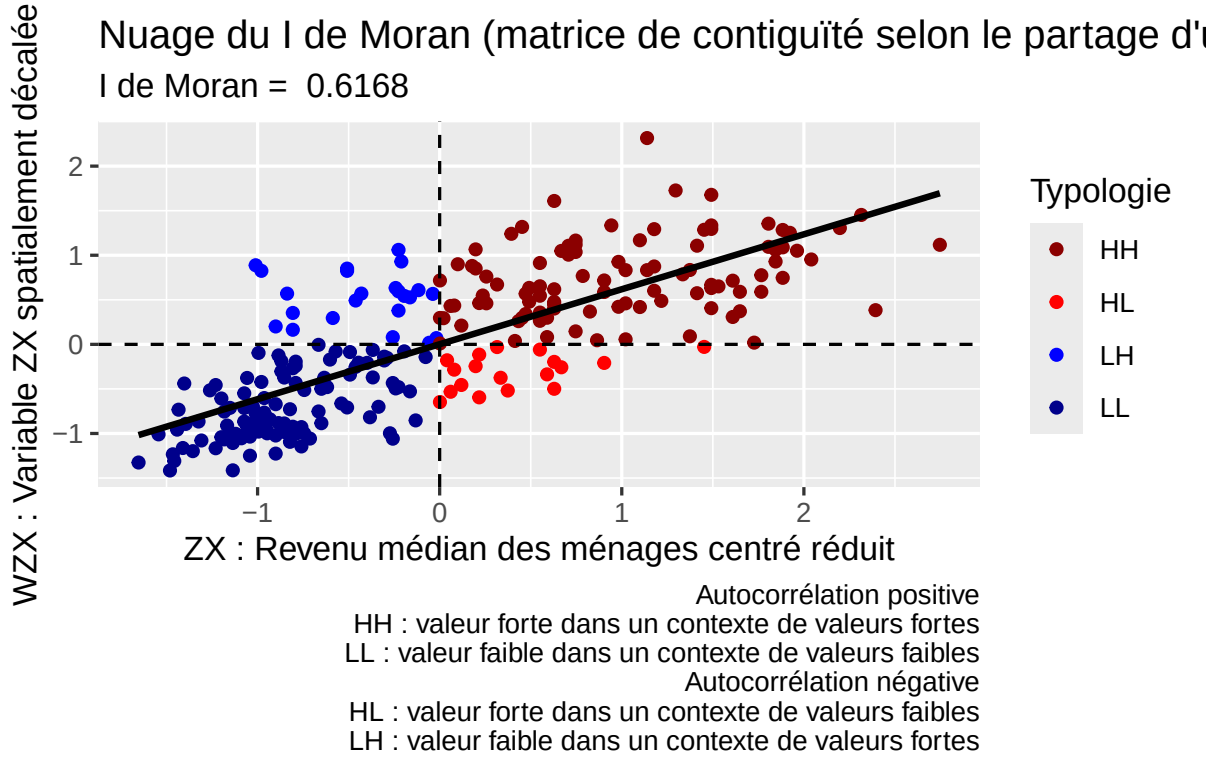
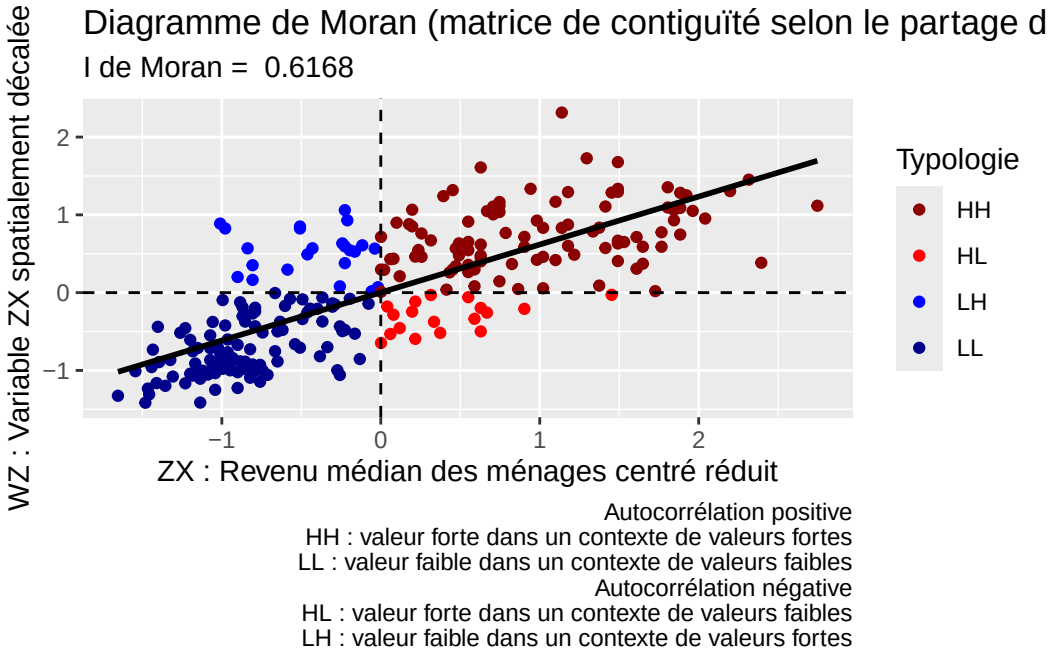


Figure 2.30 – Diagramme de Moran

2.4.3.2 Mise en œuvre dans R

Pour créer le diagramme de Moran, nous avons écrit la fonction `DiagMoranUnivarie`.

```
source("code_complementaire/DiagrammeMoran.R")
## Réalisation du diagramme de Moran avec la fonction DiagMoranUnivarie
DiagMoranUnivarie(
  x = AD.DR$RevMedMenage,
  listW = W.Rook,
  titre = "Diagramme de Moran (matrice de contiguïté selon le partage d'un segment)",
  titreAxeX = "ZX : Revenu médian des ménages centré réduit",
  titreAxeY = "WZ : Variable ZX spatialement décalée",
  AfficheAide=TRUE)
```



Reste à déterminer si l'autocorrélation spatiale locale pour ces quatre quadrants est significative. Rien de plus simple, il suffit d'utiliser la valeur de p du I de Moran local (section 2.4.2). Nous choisissons un seuil de signification (habituellement $p = 0{,}05$) et obtenons ainsi la typologie comprenant cinq catégories :

— Autocorrélation spatiale locale positive et significative ($p < 0{,}05$).

1. HH
2. LL

— Autocorrélation spatiale locale négative et significative ($p < 0{,}05$).

3. HL
4. LH

5. Autocorrélation spatiale locale non significative ($p > 0{,}05$).

Le code R ci-dessous permet d'obtenir la typologie de l'autocorrélation spatiale avec le I de Moran local pour le revenu médian des ménages avec une matrice de contiguïté selon le partage d'un segment (*Rook*). La figure 2.31 dénote surtout une autocorrélation spatiale positive importante (respectivement 33 et 48 aires de diffusion classées *HH* et *LL*), comparativement à l'autocorrélation spatiale négative qui ne comprend qu'une aire de diffusion (LH).

```
library(dplyr)
## Cote Z (variable centrée réduite)
zx <- (AD.DR$RevMedMenage - mean(AD.DR$RevMedMenage))/sd(AD.DR$RevMedMenage)
## variable X centrée réduite spatialement décalée avec une matrice Rook
wzx <- lag.listw(W.Rook, zx)
## I de Moran local (notez que vous pouvez aussi utiliser la fonction localmoran_perm)
localMoranI   <- localmoran(AD.DR$RevMedMenage, W.Rook)
#localMoranI.mc   <- localmoran_perm(AD.DR$RevMedMenage, W.Rook, n=999)
plocalMoranI  <- localMoranI[, 5]
## Choix d'un seuil de signification
signif = 0.05
## Construction de la typologie
Typologie <- attributes(localMoranI)$quadr$mean
Typologie <- case_when(
  plocalMoranI < signif ~ Typologie,
  TRUE ~ "Non sign."
)
## Enregistrement de la typologie dans un champ
AD.DR$TypoIMoran.RevMedian <- Typologie
AD.DR$TypoIMoran.RevMedian <- factor(AD.DR$TypoIMoran.RevMedian,
                                     levels = c("High-High", "High-Low", "Low-Low", "Low-High", "Non sign."),
                                     labels = c("HH", "HL", "LL", "LH", "Non sign."))
table(AD.DR$TypoIMoran.RevMedian, useNA = "always")
```

```
      HH        HL        LL        LH Non sign.      <NA>
      33         0        48         1       167         0
```

```
## Couleurs
Couleurs <- c("HH" = "#FF0000",
              "HL" = "#f4ada8",
              "LL" ="#0000FF",
              "LH" ="#a7adf9",
              "Non sign." = "#eeeeee")
## Cartographie
tmap_mode("plot")
tm_shape(AD.DR) +
  tm_polygons(col = "TypoIMoran.RevMedian",
              palette = Couleurs,
              title ="Autocorrélation spatiale locale")+
  tm_credits("HH : Positive dans un contexte de valeurs fortes (HH)
              LL : Négative dans un contexte de valeurs faibles (LL)
              HL : Positive dans un contexte de valeurs fortes (HL)
              LH : Négative dans un contexte de valeurs fortes (LH)
              Non sign. : Non significative",
```

```
            size = .5,
            position=c("right", "bottom"),
            align = "right")+
tm_layout(frame= FALSE,
          legend.outside = TRUE,
          legend.title.size = 1,
          legend.position = c("right", "center"))
```

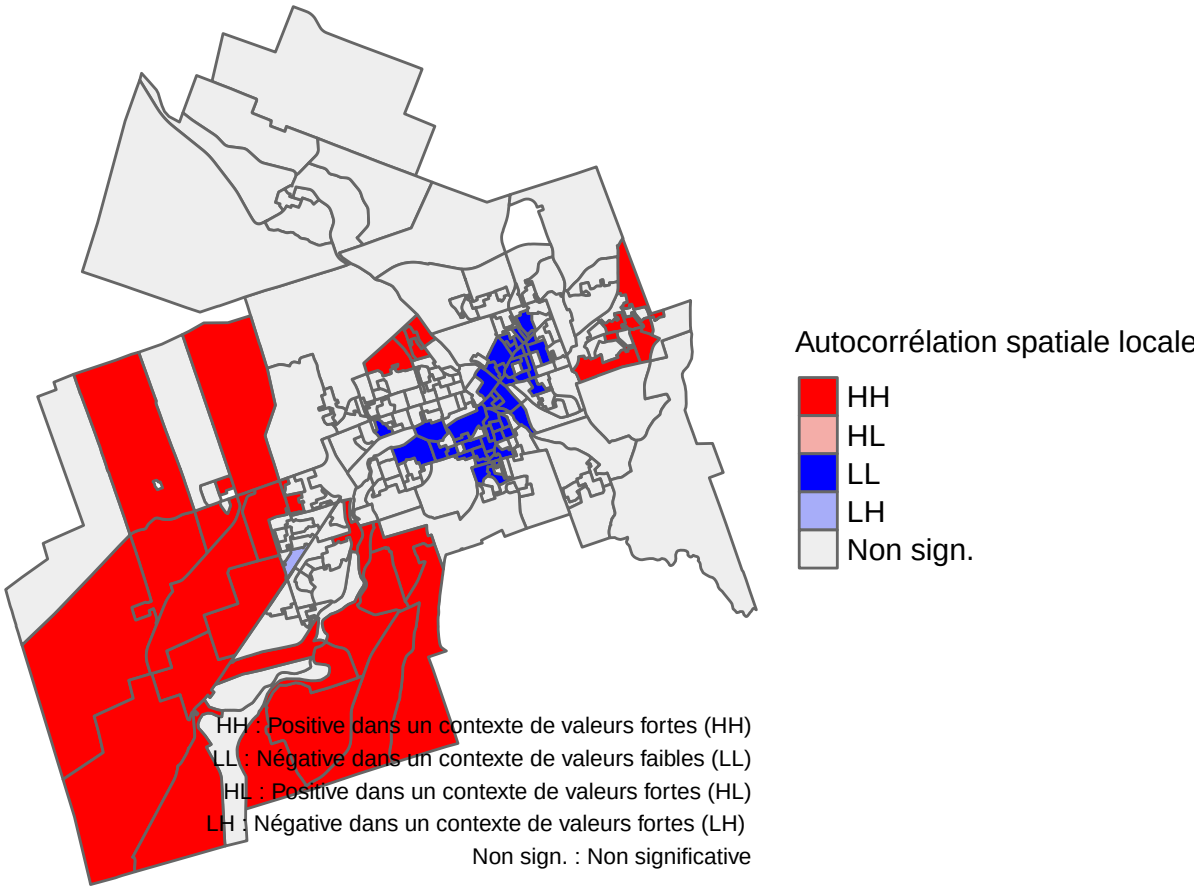


Figure 2.31 – Typologie de l'autocorrélation spatiale locale avec le I de Moran local

⚠ **Attention**

Ne pas confondre les statistiques locales de Getis et Ord et la typologie avec le I de Moran local.
Succinctement, ces deux familles de mesures renvoient deux typologies différentes :

- Le I de Moran local comprend cinq classes : HH, LL, HL, LH et non significatif.
- Les statistiques locales de Getis et Ord comprennent trois classes : points chauds (sensiblement l'équivalent de HH), points froids (sensiblement l'équivalent de LL) et non significatifs.

2.4.4 Version locale des statistiques de comptage de jointure (Join Count Statistics)

Une version locale des statistiques de comptage de jointure a été proposée par Luc Anselin et Xun Li (2019). Au même titre que la version locale du I de Moran, elle consiste à désagréger localement le calcul de l'indicateur global. Pour chaque observation, nous comptons ainsi le nombre de paires 0-0 (WW) ou 1-1 (BB) pour une variable binaire.

Pour déterminer si les associations 0-0 et 1-1 sont plus fréquentes que celles attendues du hasard, il est recommandé d'utiliser un test d'inférence par permutations. Ce test n'est actuellement pas implémenté dans `spdep`, mais il est disponible dans le *package* `rgeoda` avec la fonction `local_joincount`. Nous reprenons ici l'exemple des deux variables identifiant une majorité de locataires (`loc_maj`) ou de propriétaires (`proprio_maj`) pour les aires de diffusion (AD) de la ville de Sherbrooke.

```
library(rgeoda)
## Définition de la matrice de contiguïté avec rgeoda
wmat <- queen_weights(ADSherb)
## Création de deux variables binaires avec des valeurs de 0 et 1
## pour les locataires et propriétaires
ADSherb$loc_maj <- ifelse(ADSherb$Locataire > ADSherb$Proprietaire, 1, 0)
ADSherb$proprio_maj <- ifelse(ADSherb$Proprietaire > ADSherb$Locataire, 1, 0)
## Calcul de la version locale du join count test
testLoc <- local_joincount(wmat, ADSherb["loc_maj"], permutations = 999)
testPro <- local_joincount(wmat, ADSherb["proprio_maj"], permutations = 999)
```

Avec le code ci-dessous, nous cartographions les AD en fonction des seuils de significativité de la version locale du test. Sans surprise, les AD avec une autocorrélation spatiale significative sont les suivantes :

— celles au centre de la ville de Sherbrooke avec une majorité de locataires (figure 2.32, a);
— celles de l'ouest de la ville avec une majorité de propriétaires (figure 2.32, b).

```
library(dplyr)
## Sauvegarde des résultats
ADSherb$testLoc_pvals <- testLoc$p_vals
ADSherb$testPro_pvals <- testPro$p_vals
## Calcul d'une variable avec les seuils de significativité pour les locataires
ADSherb$clusterLoc <- case_when(
  ADSherb$testLoc_pvals < 0.05 & ADSherb$testLoc_pvals >= 0.01 ~ 1,
  ADSherb$testLoc_pvals < 0.01 ~ 2,
  TRUE ~ 0
)
ADSherb$clusterLoc <- factor(ADSherb$clusterLoc,
                         levels = 0:2,
                         labels = c("Non significatif (p > 0,05)",
                                    "p = 0,05","p = 0,01"))
## Calcul d'une variable avec les seuils de significativité pour les propriétaires
ADSherb$clusterPro <- case_when(
  ADSherb$testPro_pvals < 0.05 & ADSherb$testPro_pvals >= 0.01 ~ 1,
  ADSherb$testPro_pvals < 0.01 ~ 2,
```

```
  TRUE ~ 0
)
ADSherb$clusterPro <- factor(ADSherb$clusterPro,
                             levels = 0:2,
                             labels = c("Non significatif (p > 0,05)",
                                        "p = 0,05","p = 0,01"))

## Cartographie des résultats
Carte1 <-  tm_shape(ADSherb) +
  tm_borders(col="black", lwd=0.5)+
  tm_fill(col = "clusterLoc",
          palette = c("#f0f0f0", "#fc9272", "#de2d26"),
          title = "Seuil de significativité")+
  tm_layout(frame=FALSE,
            legend.outside = TRUE,
            legend.outside.position = c("bottom", "center"),
            title = "a. Locataires majoritaires",
            title.size = 1)
Carte2 <-  tm_shape(ADSherb) +
  tm_borders(col="black", lwd=0.5)+
  tm_fill(col = "clusterPro",
          palette = c("#f0f0f0", "#fc9272", "#de2d26"),
          title = "Seuil de significativité")+
  tm_layout(frame=FALSE,
            legend.outside = TRUE,
            legend.outside.position = c("bottom", "center"),
            title = "b. Propriétaires majoritaires",
            title.size = 1)
tmap_arrange(Carte1, Carte2)
```

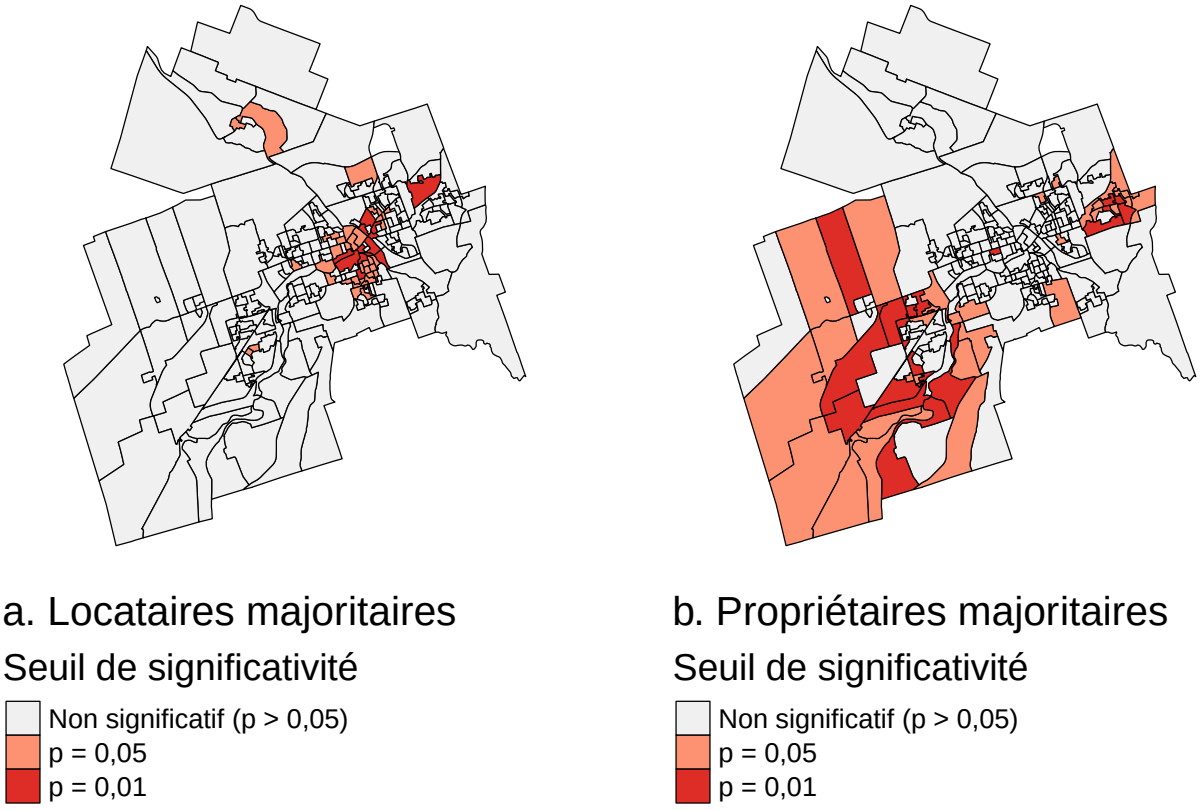


FIGURE 2.32 – Cartographie de la version locale des statistiques de comptage de jointure

2.4.5 Version locale de l'indice de Lee

2.4.5.1 Formulation de la version locale de l'indice de Lee

Dans le cas de l'autocorrélation spatiale bivariée, l'indice de Lee dispose aussi d'une version locale. Plus exactement, la version globale de l'indice correspond à l'agrégation des contributions locales de chaque observation. En cartographiant ces contributions locales, il est possible de se faire une idée de la relation spatiale entre les deux variables à l'étude. La version locale de l'indice est définie par l'équation 2.23.

$$L_i = \frac{n \cdot \left[\left(\sum_i w_{ij}\left(x_j - \bar{x}\right)\right)\left(\sum_i w_{ij}\left(y_j - \bar{y}\right)\right)\right]}{\sqrt{\sum_i \left(x_i - \bar{x}\right)^2}\sqrt{\sum_i \left(y_i - \bar{y}\right)^2}} = \frac{n \cdot \left(\tilde{x}_i - \bar{x}\right)\left(\tilde{y}_i - \bar{y}\right)}{\sqrt{\sum_i \left(x_i - \bar{x}\right)^2}\sqrt{\sum_i \left(y_i - \bar{y}\right)^2}} \tag{2.23}$$

Avec :

- n, le nombre d'observations;
- W_{ij}, la valeur de la pondération spatiale entre les entités spatiales i et j;
- $\bar{x}$, la moyenne de la variable x;
- $\bar{y}$, la moyenne de la variable y.

2.4.5.2 Mise en œuvre dans R

Reprenons notre exemple précédent sur les pollutions atmosphériques à Lyon. Nous avions observé que le dioxyde d'azote partageait un patron d'autocorrélation spatiale positive forte avec les particules fines, et un patron moins prononcé avec le bruit. Nous allons observer ces deux relations à l'échelle locale.

```
library(geocmeans)
data("LyonIris")
## Liste de variables à analyser
vars <- c("Lden","NO2","PM25","VegHautPrt")
## Matrice de pondération spatiale Queen standardisée
nb <- poly2nb(LyonIris, queen = TRUE)
Wmat <- nb2listw(nb, style = 'W')
## Calcul de la version locale de l'indice de Lee entre
# NO2 et PM25
Lee_1 <- lee(LyonIris$NO2, LyonIris$PM25, listw = Wmat, n = nrow(LyonIris))
# NO2 et DBA
Lee_2  <- lee(LyonIris$NO2, LyonIris$Lden, listw = Wmat, n = nrow(LyonIris))
```

Dans le *package* **spdep**, aucune fonction ne permet de calculer un test d'inférence pour les valeurs locales de Lee. Nous pouvons cependant assez facilement obtenir des pseudo valeurs de p en réalisant 999 permutations aléatoires et en comptant combien de fois les valeurs obtenues au hasard sont plus grandes ou plus petites que celles obtenues dans les données initiales.

```
## Nous créons une matrice à deux colonnes remplies de zéros qui
## servira à compter le nombre de fois où les valeurs des permutations
## sont au-dessus ou en dessous des valeurs réelles
p_vals_1 <- cbind(rep(0, nrow(LyonIris)),
                  rep(0, nrow(LyonIris))
                  )
p_vals_2 <- cbind(rep(0, nrow(LyonIris)),
                  rep(0, nrow(LyonIris))
                  )
N <- nrow(LyonIris)
for(i in 1:999){
  # Permutation aléatoire de l'ordre des observations
  perm_data <- LyonIris[sample(1:nrow(LyonIris)),]
  # Calcul de l'indice de Lee sur les données permutées
  Lee_1_perm <- lee(perm_data$NO2, perm_data$PM25, listw = Wmat, n = N)
  Lee_2_perm  <- lee(perm_data$NO2, perm_data$Lden, listw = Wmat, n = N)
  # Test pour vérifier si les valeurs obtenues sont plus grandes
  test1 <- Lee_1$localL < Lee_1_perm$localL
  test2 <- Lee_2$localL < Lee_2_perm$localL
  # Ajout du résultat aux matrices de valeurs de p
  p_vals_1[,1] <- p_vals_1[,1] + test1
  p_vals_2[,1] <- p_vals_2[,1] + test2
```

```
  # Test pour vérifier si les valeurs obtenues sont plus petites
  test1 <- Lee_1$localL > Lee_1_perm$localL
  test2 <- Lee_2$localL > Lee_2_perm$localL
  # Ajout du résultat aux matrices de valeurs de p
  p_vals_1[,2] <- p_vals_1[,2] + test1
  p_vals_2[,2] <- p_vals_2[,2] + test2
}

## Ajout des données originales
LyonIris$Lee1 <- Lee_1$localL
LyonIris$Lee2 <- Lee_2$localL
LyonIris$Lee1_p_higher <- p_vals_1[,1] / 999
LyonIris$Lee1_p_lower <- p_vals_1[,2] / 999
LyonIris$Lee2_p_higher <- p_vals_2[,1] / 999
LyonIris$Lee2_p_lower <- p_vals_2[,2] / 999

## Cartographie des résultats avec deux cartes dans lesquelles
## les valeurs non significatives au seuil 0,01 sont représentées en gris clair.
test_sign1 <- LyonIris$Lee1_p_higher < 0.01 | LyonIris$Lee1_p_lower < 0.01
map1 <- tm_shape(LyonIris)+tm_borders(col="black")+
  tm_shape(subset(LyonIris, test_sign1)) +
  tm_polygons(col = 'Lee1', n = 5, style = 'kmeans', midpoint = 0, palette = '-RdBu',
              legend.format = list(digits = 2, text.separator = "à"),
              title = 'Indice de Lee') +
  tm_shape(subset(LyonIris, !test_sign1)) +
  tm_polygons(col = 'lightgrey')+
  tm_layout(legend.outside = FALSE, frame = FALSE,
            title = "a. NO2 versus PM25", title.size = .75)+
  tm_credits(text="Gris : non significatif (p > 0,001).",
             position = c("RIGHT", "BOTTOM"))

test_sign2 <- LyonIris$Lee2_p_higher < 0.01 | LyonIris$Lee2_p_lower < 0.01
map2 <- tm_shape(LyonIris)+tm_borders(col="black")+
  tm_shape(subset(LyonIris, test_sign2)) +
  tm_polygons(col = 'Lee2', n = 5, style = 'kmeans', palette = '-RdBu',
              legend.format = list(digits = 2, text.separator = "à"),
              title = 'Indice de Lee') +
  tm_shape(subset(LyonIris, !test_sign2)) +
  tm_polygons(col = 'lightgrey') +
  tm_layout(legend.outside = FALSE, frame = FALSE,
            title = "b. NO2 versus Lden", title.size = .75)+
  tm_credits(text="Gris : non significatif (p > 0,001).",
             position = c("RIGHT", "BOTTOM"))

tmap_arrange(map1, map2, ncol = 2)
```

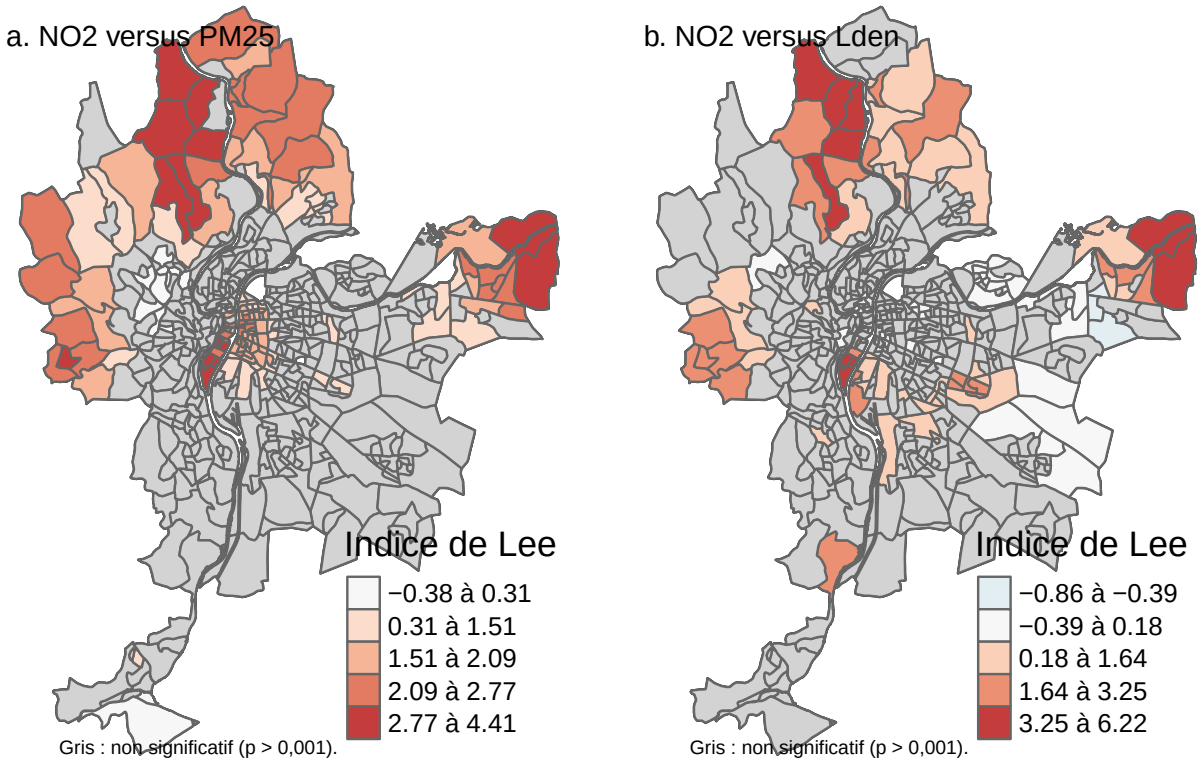


FIGURE 2.33 – Cartographie des valeurs de l'indice de Lee local

À la figure 2.33 (a), nous constatons une forte autocorrélation spatiale bivariée positive dans deux secteurs de l'agglomération :

— Au centre avec des entités spatiales ayant des valeurs fortes pour les deux variables.
— Dans les zones périphériques au nord, à l'est et à l'ouest avec des entités spatiales ayant des valeurs faibles pour les deux variables.

À la figure 2.33 (b), nous observons trois patrons d'autocorrélation spatiale bivariée :

— Autocorrélation spatiale positive pour des entités spatiales au centre de l'agglomération avec des valeurs fortes de bruit et de dioxyde d'azote.
— Autocorrélation spatiale positive pour des entités spatiales au nord de l'agglomération avec des valeurs faibles de bruit et de dioxyde d'azote.
— Autocorrélation spatiale négative à l'est de l'agglomération avec des valeurs fortes pour le bruit, mais faibles pour le dioxyde d'azote.

Aller plus loin

Autocorrélation spatiale multivariée
Il est possible d'analyser l'autocorrélation spatiale dans un contexte multivarié. Cependant, ces méthodes sont parfois difficiles à interpréter, car l'augmentation de la dimensionnalité des données analysées ajoute un haut niveau d'abstraction. Si ces méthodes vous intéressent, nous vous recommandons de jeter un œil à :

- L'indice de Geary multivarié local (Anselin 2019), qui peut être calculé facilement dans R (avec la fonction `local_multigeary` du *package* `rgeoda`) en faisant la moyenne des valeurs locales de l'indice de Geary univarié.
- Les méthodes factorielles intégrant l'espace, dont l'analyse en composantes principales spatiales (*spatial principal component analysis* en anglais, SPCA) (Jombart et al. 2008) qui permet d'identifier des facteurs maximisant le produit de la variance expliquée d'un ensemble de variables continues et l'autocorrélation spatiale de ces facteurs (mesurée avec le *I* de Moran). Dans R, les *packages* pertinents pour ces méthodes sont `adegenet` et `ade4`.

2.4.6 Typologie basée sur le diagramme de Moran dans un contexte bivarié

2.4.6.1 Formulation de la typologie basée sur le diagramme de Moran dans un contexte bivarié

À la section 2.4.3, nous avons décrit la construction du diagramme de Moran dans un contexte univarié avec la variable centrée réduite (**ZX**) sur l'axe des X et la variable centrée réduite spatialement décalée (**WZX**) sur l'axe des Y. Par la suite, ce diagramme est décomposé en quatre quadrants (HH, LL, HL, LH). Cette démarche peut être adaptée à un contexte bivarié avec deux variables continues, c'est-à-dire avec **ZX** et **WZY** pour construire le diagramme. Les quatre quadrants s'interprètent alors comme suit (figure 2.33) :

1. **HH (*High-High*)** regroupe des entités spatiales avec des valeurs fortes (H) pour la variable X qui sont voisines ou proches d'autres entités spatiales avec des valeurs fortes pour la variable Y (H). Nous sommes donc en présence **d'autocorrélation spatiale locale bivariée positive avec des valeurs fortes (HH)**.
2. **LL (*Low-Low*)** regroupe des entités spatiales avec des valeurs faibles (L) pour la variable X qui sont voisines ou proches d'autres entités spatiales avec des valeurs faibles pour la variable Y (L). Nous sommes donc en présence **d'autocorrélation spatiale locale bivariée positive avec des valeurs faibles (LL)**.
3. **HL (*High-Low*)** regroupe des entités spatiales avec des valeurs fortes (H) pour la variable X qui sont voisines ou proches d'autres entités spatiales avec des valeurs faibles pour la variable Y (L), soit de **l'autocorrélation spatiale locale bivariée négative (HL)**.
4. **LH (*Low-High*)** regroupe des entités spatiales avec des valeurs faibles (L) pour la variable X qui sont voisines ou proches d'autres entités spatiales avec des valeurs fortes (H) pour la variable Y, soit de **l'autocorrélation spatiale bivariée locale négative (LH)**.

Pour déterminer si l'autocorrélation spatiale locale (positive ou négative) pour les quatre est significative, nous utilisons la valeur de *p* de la version locale du *I* de Moran bivariée (habituellement p = 0,05).

Attention

Non-symétrie du *I* de Moran bivarié et du diagramme de Moran dans un contexte bivarié
À titre de rappel, le diagramme est construit avec **ZX** et **WZY**, tout comme l'indice de Moran bivarié. Par conséquent, les résultats entre **ZX** et **WZY** (figure 2.33, a) et **ZY** et **WZX** (figure 2.33, b) ne sont pas identiques!

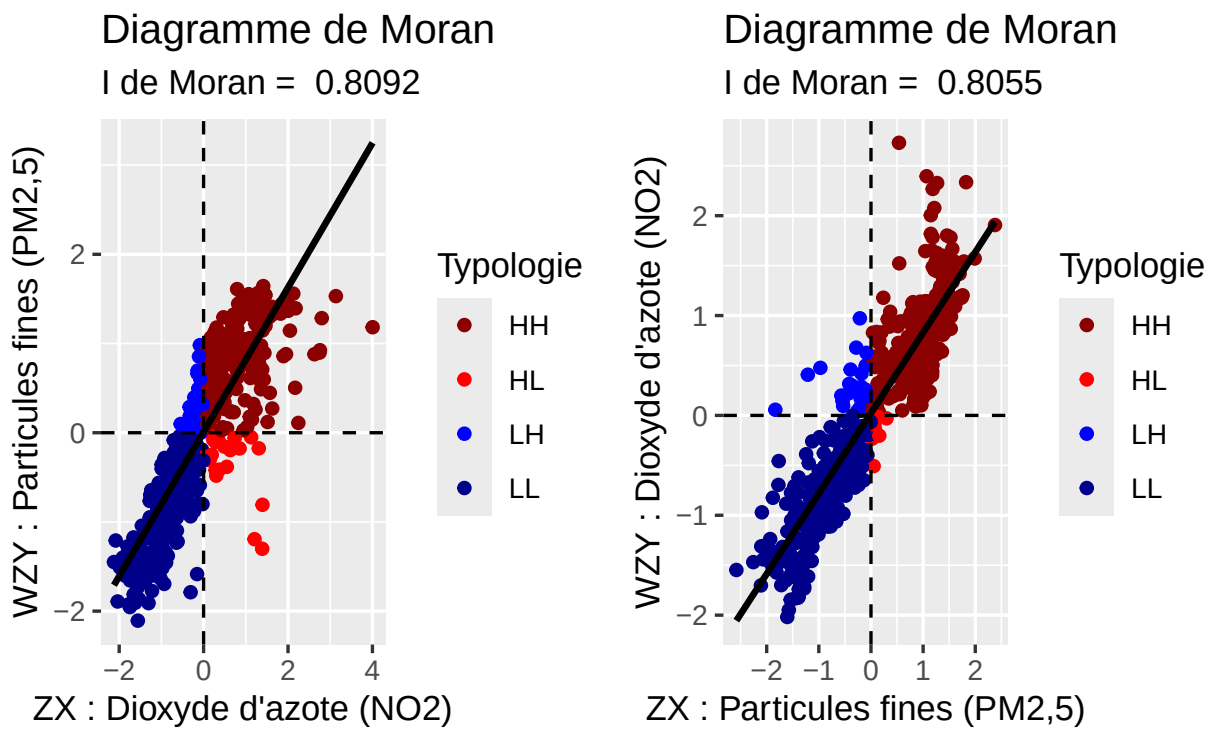


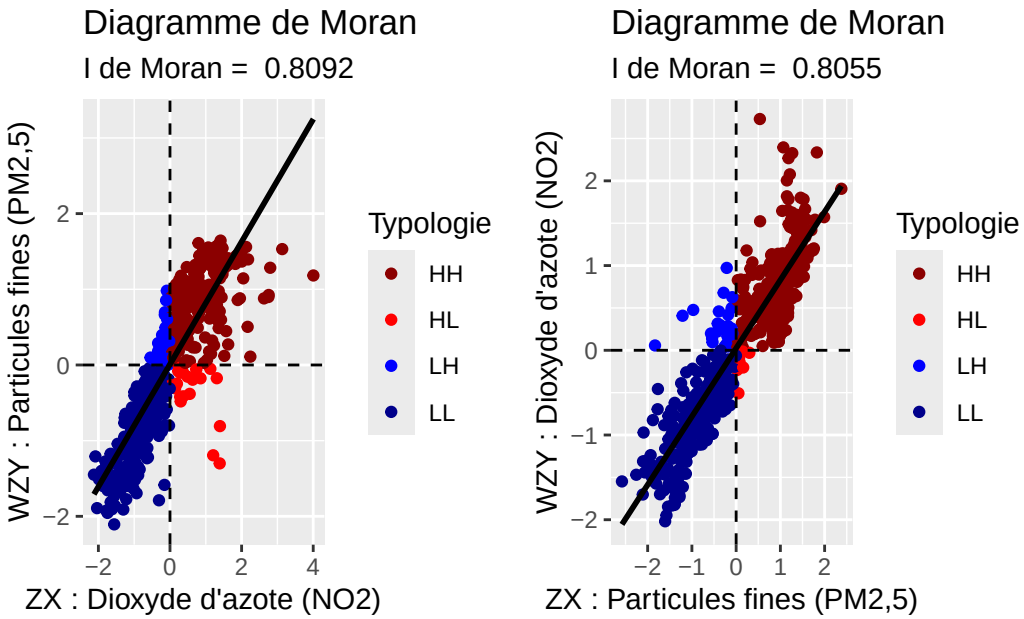
FIGURE 2.34 – Diagramme de Moran dans un contexte bivarié

2.4.6.2 Mise en œuvre dans R

Pour créer le diagramme de Moran dans un contexte bivarié, nous avons écrit la fonction `DiagMoranUnivarie`.

```
source("code_complementaire/DiagrammeMoran.R")
library(ggpubr)
## Matrice de contiguïté
Queen <- poly2nb(LyonIris, queen=FALSE)
W.Queen <- nb2listw(Queen, zero.policy=TRUE, style = "W")
## Réalisation du diagramme de Moran avec la fonction DiagMoranBivarie
Diag1 <- DiagMoranBivarie(x = LyonIris$NO2,
                 y = LyonIris$PM25,
                 listW = W.Queen,
                 titre = "Diagramme de Moran",
                 titreAxeX = "ZX : Dioxyde d'azote (NO2)",
                 titreAxeY = "WZY : Particules fines (PM2,5)",
                 AfficheAide=FALSE)
Diag2 <- DiagMoranBivarie(x = LyonIris$PM25,
                 y = LyonIris$NO2,
                 listW = W.Queen,
                 titre = "Diagramme de Moran",
                 titreAxeX = "ZX : Particules fines (PM2,5)",
                 titreAxeY = "WZY : Dioxyde d'azote (NO2)",
```

```
                AfficheAide=FALSE)
ggarrange(Diag1, Diag2)
```



Pour mettre en œuvre la typologie, vous pouvez utiliser soit la fonction `local_bimoran` du *package* `rgeoda`, soit la fonction `localmoran_bv` du *package* `spdep`. Cette dernière est mobilisée dans le code ci-dessous pour obtenir la typologie de l'autocorrélation spatiale bivariée entre :

— Le dioxyde d'azote (**ZX**) et les particules fines (**ZWY**) (figure 2.33, a).
— Les particules fines (**ZX**) et le dioxyde d'azote (**ZWY**) (figure 2.33, b).

```
library(spdep)
## Matrice de contiguïté
Queen <- poly2nb(LyonIris, queen=FALSE)
W.Queen <- nb2listw(Queen, zero.policy=TRUE, style = "W")
## Variables centrées réduites
z.no2 <- (LyonIris$NO2 - mean(LyonIris$NO2))/sd(LyonIris$NO2)
z.pm25 <- (LyonIris$PM25 - mean(LyonIris$PM25))/sd(LyonIris$PM25)
## Variables spatialement décalées
wz.no2 <- lag.listw(W.Queen, z.no2)
wz.pm25 <- lag.listw(W.Queen, z.pm25)
## I de Moran local bivarié
localMoranBivariee_NO2_PM25 <- localmoran_bv(LyonIris$NO2, LyonIris$PM25, W.Queen, nsim = 999)
localMoranBivariee_PM25_NO2 <- localmoran_bv(LyonIris$PM25, LyonIris$NO2, W.Queen, nsim = 999)
## Valeur de p du I de Moran bivariée
plocalMoranI_NO2_PM25  <- localMoranBivariee_NO2_PM25[, 7]
plocalMoranI_PM25_NO2  <- localMoranBivariee_PM25_NO2[, 7]
## Choix d'un seuil de signification
signif = 0.05
## Construction de la typologie NO2 versus PM25
```

```
TypoNO2_PM25 <- ifelse(z.no2 > 0 & wz.pm25 > 0, "1. HH", NA)
TypoNO2_PM25 <- ifelse(z.no2 < 0 & wz.pm25 < 0, "2. LL", TypoNO2_PM25)
TypoNO2_PM25 <- ifelse(z.no2 > 0 & wz.pm25 < 0, "3. HL", TypoNO2_PM25)
TypoNO2_PM25 <- ifelse(z.no2 < 0 & wz.pm25 > 0, "4. LH", TypoNO2_PM25)
TypoNO2_PM25 <- ifelse(plocalMoranI_NO2_PM25 > signif, "Non sign", TypoNO2_PM25)
## Construction de la typologie PM25 versus NO2
TypoPM25_NO2 <- ifelse(z.pm25 > 0 & wz.no2 > 0, "1. HH", NA)
TypoPM25_NO2 <- ifelse(z.pm25 < 0 & wz.no2 < 0, "2. LL", TypoPM25_NO2)
TypoPM25_NO2 <- ifelse(z.pm25 > 0 & wz.no2 < 0, "3. HL", TypoPM25_NO2)
TypoPM25_NO2 <- ifelse(z.pm25 < 0 & wz.no2 > 0, "4. LH", TypoPM25_NO2)
TypoPM25_NO2 <- ifelse(plocalMoranI_PM25_NO2 > signif, "Non sign", TypoPM25_NO2)
## Enregistrement des résultats dans la couche LyonIRS
LyonIris$TypoNO2_PM25 <- TypoNO2_PM25
LyonIris$TypoPM25_NO2 <- TypoPM25_NO2
## Couleurs
Couleurs <- c("1. HH" = "#FF0000",
              "2. LL" ="#0000FF",
              "3. HL" = "#f4ada8",
              "4. LH" ="#a7adf9",
              "Non sign" = "#eeeeee")
## Cartographie
tmap_mode("plot")
Carte1 <- tm_shape(LyonIris) +
            tm_polygons(col = "TypoNO2_PM25",
              palette = Couleurs,
              title ="Typologie")+
            tm_layout(frame = FALSE,
                      legend.outside = TRUE,
                      legend.outside.position = c("bottom", "center"),
                      title = "a. NO2 versus PM2,5",
                      title.size = .9)
Carte2 <- tm_shape(LyonIris) +
            tm_polygons(col = "TypoPM25_NO2",
              palette = Couleurs,
              title ="Typologie")+
            tm_layout(frame = FALSE,
                      legend.outside = TRUE,
                      legend.outside.position = c("bottom", "center"),
                      title = "b. PM2,5 versus NO2",
                      title.size = .9)
tmap_arrange(Carte1, Carte2)
```

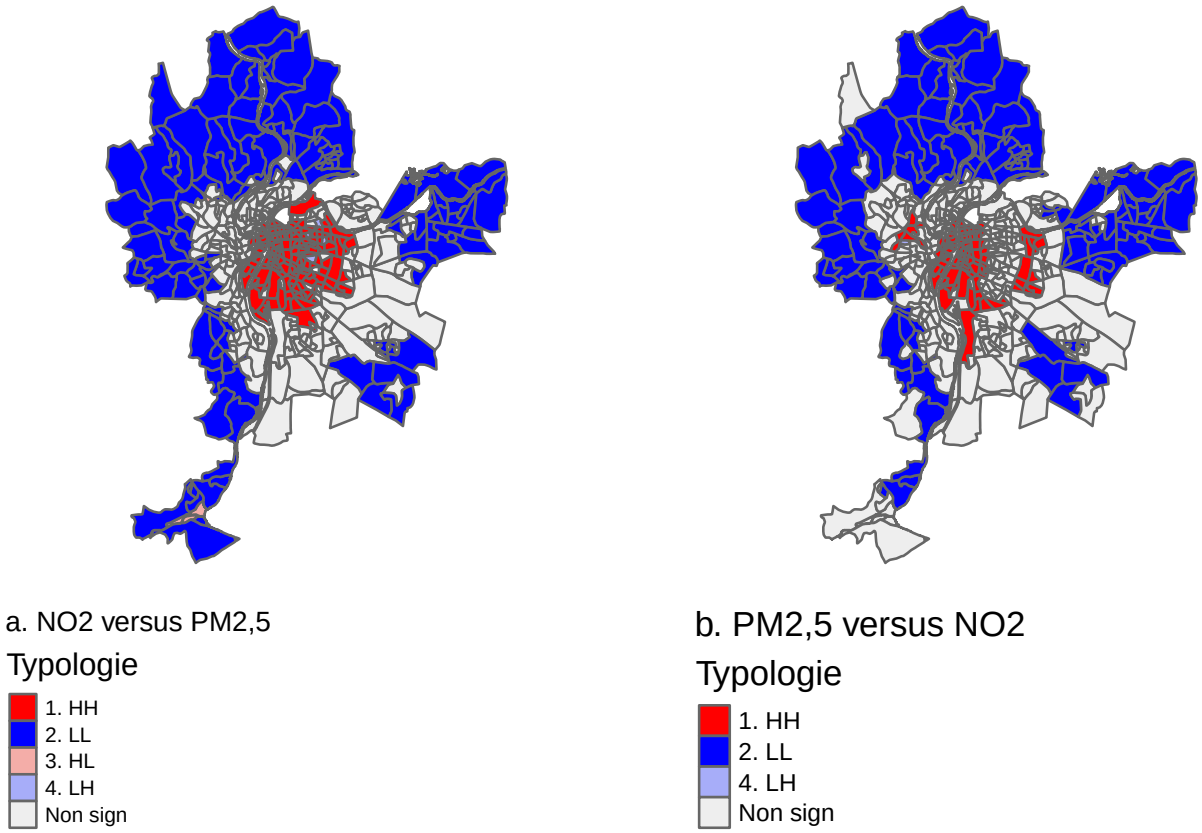


FIGURE 2.35 – Typologies basées sur le diagramme de Moran dans un contexte bivarié

Aller plus loin

Autocorrélation spatiale locale dans un contexte bivarié pour les variables binaires
À la section 2.4.4, nous avons abordé les statistiques locales de comptage de jointure. Tout comme la version locale de I de Moran, elles peuvent être adaptées à un contexte bivarié avec deux variables binaires. Pour ce faire, vous pouvez utiliser la fonction `local_bijoincount` du *package* `rgeoda`.

2.4.7 Autocorrélation locale pour une variable qualitative (catégorielle) : l'indicateur ELSA

2.4.7.1 Formulation de l'indicateur ELSA

Récemment, un indicateur basé sur la mesure de l'entropie locale a été proposé pour mesurer l'autocorrélation spatiale locale d'une variable qualitative comprenant plusieurs modalités (catégories) (Naimi et al. 2019). Cet indicateur ELSA (*Entropy-based local indicator of spatial association*), qui varie de 0 (autocorrélation spatiale parfaite) à 1 (forte hétérogénéité spatiale), est calculé comme suit :

$$
\begin{aligned}
E_i &= E_{ai} \times E_{ci} \\
E_{ai} &= \frac{\sum_j \omega_{ij} d_{ij}}{\max\{d\} \sum_j \omega_{ij}}, j \neq i \\
E_{ci} &= -\frac{\sum_{k=1}^{m_\omega} p_k \log_2 (p_k)}{\log_2 m_i}, j \neq i \\
m_i &= \begin{cases} m \text{ if } \sum_j \omega_{ij} > m \\ \sum_j \omega_{ij}, \text{ otherwise} \end{cases} \\
d_{ij} &= |c_i - c_j|
\end{aligned}
\tag{2.24}
$$

Il s'agit donc du produit de deux termes, soit E_{ai} et E_{ci}. Le premier mesure la dissimilarité entre l'observation *i* et ses voisins *j*. Le second est l'indice d'entropie de Shannon et quantifie la diversité des observations voisines de *i*.

L'indicateur ELSA a la particularité de tenir compte du degré de dissimilarité entre les modalités (catégories) de la variable qualitative à l'étude. Aussi, il permet de spécifier qu'une catégorie *A* est plus ressemblante à une catégorie *B* qu'à une catégorie *C*. Or, nous supposons habituellement que *A* est différent de *B* et de *C*, ce qui produit une matrice des distances sémantiques binaire, telle que présentée au tableau 2.11.

Tableau 2.11 – Matrice des distances sémantiques classique (binaire)

	A	B	C
A	0	1	1
B	1	0	1
C	1	1	0

Toutefois, si nous considérons que les catégories *A* et *B* sont plus proches entre elles que de la catégorie *C*, la matrice des distances sémantiques devrait être différente comme présentée au tableau 2.12.

Tableau 2.12 – Matrice des distances sémantiques modifiée

	A	B	C
A	0,0	0,5	1
B	0,5	0,0	1
C	1,0	1,0	0

Pour déterminer si l'autocorrélation spatiale est significative, Babak Naimi et ses collègues (2019) proposent une approche par simulations Monte-Carlo, très semblable aux tests par permutations vus pour les autres mesures d'autocorrélation spatiale.

2.4.7.2 Mise en œuvre dans R

Nous illustrons ici comment calculer l'indice *ELSA* sur des données matricielles (*raster*) avec une image dérivée d'un modèle numérique de terrain (élévation) et d'une image d'un indice de végétation par différence normalisée (NDVI)

(figure 2.36). Cette image est en accès libre sur le portail de données de la Communauté métropolitaine de Montréal. Pour accélérer les calculs, nous avons extrait une petite partie de l'image qui couvre une zone de la ville de Laval. Cette image classifie le territoire en quatre catégories :

1. Minéral bas (< 3 mètres et NDVI < 0,3) (route, stationnement, etc.).
2. Minéral haut (>= 3 mètres et NDVI < 0,3) (construction).
3. Végétal bas (< 3 mètres et NDVI >= 0,3) (culture, gazon, etc.).
4. Végétal haut (>= 3 mètres et NDVI >= 0,3) (canopée).

```
library(terra)
library(tmap)
library(geocmeans)
library(ggpubr)
## Chargement du raster
laval_data <- rast("data/chap02/65005_IndiceCanopee_2021_sub.tif")
tm_shape(laval_data) +
  tm_raster(palette = c("#fbd4b4", "#e36c0a", "#92d050", "#76923c"),
            style = 'cat',
            title = "Classe (catégorie)",
            labels = c("1. Minéral bas", "2. Minéral haut",
                       "3. Végétal bas", "4. Végétal haut"))+
  tm_layout(legend.outside = TRUE, frame = FALSE)
```

Figure 2.36 – Données matricielles sur une portion du territoire de Laval

Pour calculer l'indicateur ELSA, nous devons premièrement définir une matrice de voisinage. Pour ce faire, nous créons plusieurs fenêtres circulaires de taille différente (100, 200, 300 et 400 mètres; figure 2.37). Notez que la résolution spatiale de l'image utilisée est de 5 m x 5 m, ce qui est possible de vérifier avec la fonction `terra::res`. La figure 2.37 illustre la taille des différentes matrices spatiales circulaires que nous allons construire. Chaque pixel de la zone en noir sera considéré comme un voisin du pixel central pour lequel l'indicateur *ELSA* sera calculé.

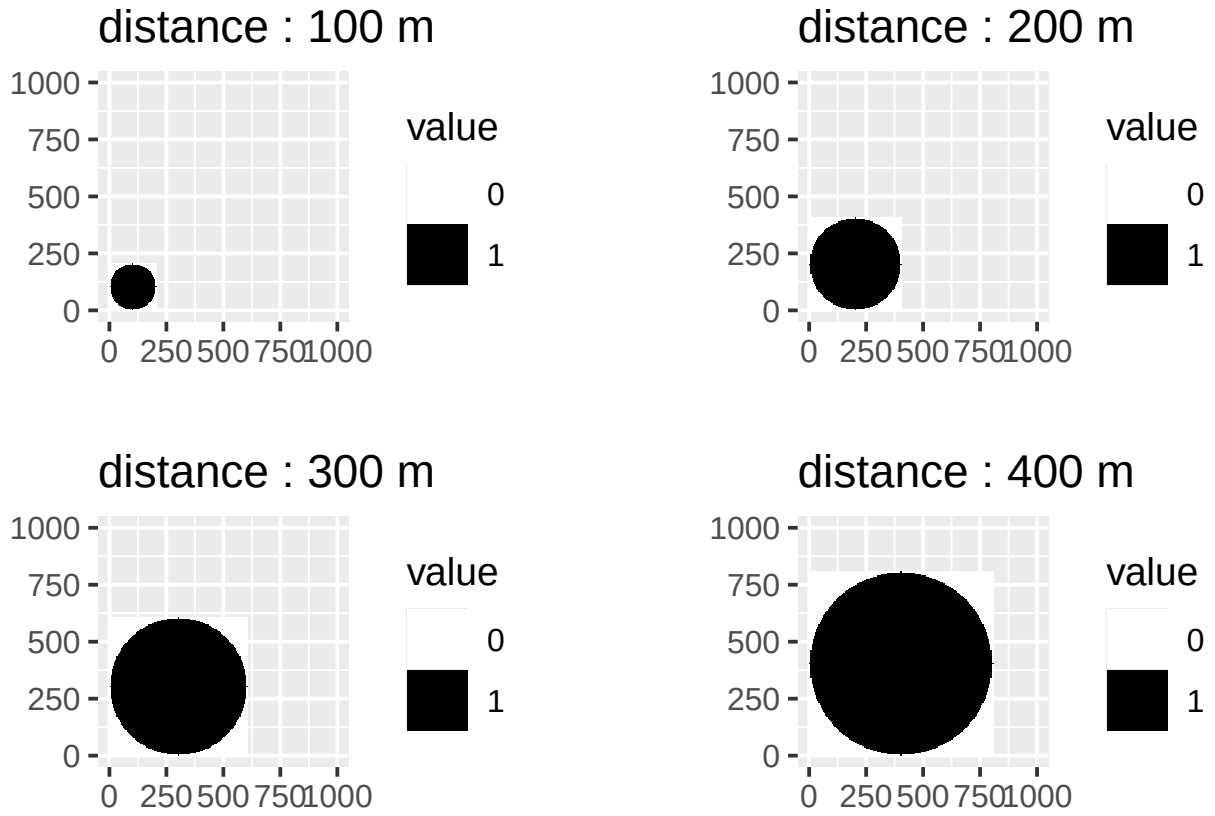


Figure 2.37 – Illustration des matrices spatiales circulaires sur une image

En plus de la matrice spatiale, nous devons définir une matrice de dissimilarité entre les modalités (catégories) de la variable qualitative. Nous proposons de considérer que les catégories **Végétal haut** et **Végétal bas** ont une distance de 0,5 entre elles, tout comme les catégories de surface **Minéral bas** et **Minéral haut**. Par contre, la distance est fixée à 1 entre les surfaces végétales et celles minérales (tableau 2.13).

Tableau 2.13 – Matrice de dissimilarité entre les catégories de l'image

	MB	MH	VB	VH
Minéral bas (MB)	0,0	0,5	1,0	1,0
Minéral haut (MH)	0,5	0,0	1,0	1,0
Végétal bas (VB)	1,0	1,0	0,0	0,5
Végétal haut (VH)	1,0	1,0	0,5	0,0

Finalement, nous calculons l'indicateur d'autocorrélation spatiale ELSA avec uniquement des rayons de 100 et de 400 mètres afin de limiter le temps de calcul. Pour ce faire, nous utilisons le *package* `geocmeans` et ses fonctions `circular_window` et `elsa_raster`.

```r
## Avec geocmeans, la première catégorie doit avoir la valeur 0
laval_data <- laval_data - 1
## Définition des fenêtres circulaires à 100 et 400 mètre
w100 <- circular_window(100,5)
w400 <- circular_window(400,5)
## Calcul de l'indicateur ELSA
elsa_100 <- elsa_raster(laval_data, w100, matrice_dissim)
elsa_400 <- elsa_raster(laval_data, w400, matrice_dissim)
## Cartographie des résultats
carte1 <- tm_shape(elsa_100) +
  tm_raster(palette = "RdYlGn", n = 6,
            legend.is.portrait = FALSE, title = "ELSA 100m") +
  tm_layout(legend.outside = TRUE, frame = FALSE,
            legend.outside.position = "bottom",
            legend.stack = "horizontal",
            legend.text.size = .55)
carte2 <- tm_shape(elsa_400) +
  tm_raster(palette = "RdYlGn", n = 6,
            legend.is.portrait = FALSE, title = "ELSA 400m") +
  tm_layout(legend.outside = TRUE, frame = FALSE,
            legend.outside.position = "bottom",
            legend.stack = "horizontal",
            legend.text.size = .55)
tmap_arrange(carte1, carte2, ncol = 2)
```

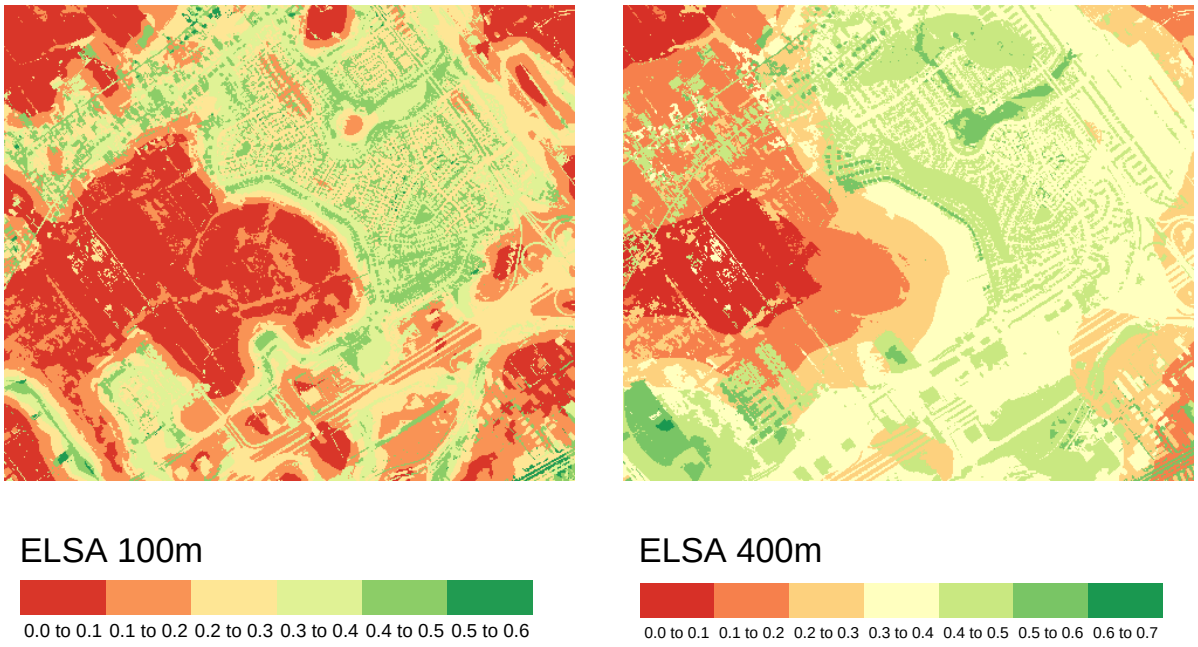


Figure 2.38 – Cartographie des valeurs ELSA obtenues

La figure 2.38 révèle que l'utilisation d'une matrice circulaire de 400 mètres lisse trop les résultats. Par conséquent, nous privilégions ceux obtenus avec une matrice circulaire de 100 mètres. Puis, avec le code ci-dessous, nous souhaitons identifier les bâtiments avec un environnement immédiat (défini à 100 mètres) avec une forte hétérogénéité spatiale, c'est-à-dire végétalisé.

```
# Nous isolons les bâtiments avec forte dissimilarité à proximité
# la valeur de l'image de sortie (buildings_green) sera 1
buildings_green <- (laval_data == 1) & (elsa_100 >= 0.4)
# Nous isolons les bâtiments avec faible dissimilarité à proximité
# la valeur de l'image de sortie (buildings_other) sera 2
buildings_other <- ((laval_data == 1) & (elsa_100 < 0.4)) * 2
# Nous combinons les deux images
raster_buildings <- buildings_green + buildings_other
tmap_mode("view")
tm_basemap(c("Esri.WorldImagery", "OpenStreetMap"))+
  tm_shape(raster_buildings) +
  tm_raster(palette = c(alpha("white",0),  "green", "grey"),
            title = "Bâtiments",
            labels = c("Transparent (0) : autre que bâtiments",
                       "Vert (1) : forte dissimilarité",
                       "Gris (2) : faible dissimilarité"),
            style = 'cat', zindex = 1)
```

Sur la carte ci-dessus, nous distinguons clairement deux groupes de bâtiments avec une forte hétérogénéité spatiale dans leur environnement immédiat (en vert) : ceux en bordure du quartier résidentiel et ceux situés le long des rangs Saint-Elzéar Est et Haut-Saint-François au nord-ouest. À l'inverse, les bâtiments au cœur du quartier résidentiel de banlieue affichent une faible hétérogénéité spatiale dans leur environnement immédiat (en gris).

Aller plus loin

Indicateur ELSA : données vectorielles et test d'inférence

Données vectorielles

Si les données sont vectorielles et non matricielles, il convient alors d'utiliser la fonction `elsa_vector` du *package* `geocmeans`.

Indicateur ELSA et test d'inférence

Dans les fonctions `elsa_raster` et `elsa_vector`, aucun test d'inférence n'est implémenté pour obtenir une valeur de p locale. Toutefois, il est assez facile dans R d'appliquer la méthode par simulation Monte-Carlo décrite dans l'article de Naimi *et al.* (2019). Appliquée à des données matricielles, la démarche est la suivante :

- Créer 999 images dont les valeurs des pixels sont tirées aléatoirement avec remplacement.
- Recalculer l'indicateur ELSA pour chaque image.
- Pour chaque pixel, compter le nombre de fois où la valeur observée de l'indicateur ELSA est supérieure ou égale à celles des 999 valeurs simulées.

La pseudo valeur de p pour chaque pixel i s'écrit alors :

$$\text{Pseudo valeur de } p_i = (M+1)/(R+1) \text{ avec :} \tag{2.25}$$

M étant le nombre de fois que la valeur observée est supérieure ou égale à la valeur simulée ($E_i \geq E^*_{ir}$) (Naimi et al. 2019, pp. 33-35).

Appliquons la démarche ci-dessous dans R pour obtenir une pseudo valeur pour l'indicateur ELSA obtenu sur l'image avec un rayon de 100 mètres. Notez que pour améliorer la vitesse de calcul, nous convertissons notre objet `terra::rast` en une matrice.

```
## Conversion des données matricielles en matrice
base_data <- terra::values(laval_data, format = "matrice")
## calcul de la proportion de chaque catégorie
N <- ncell(base_data)
probs <- table(base_data) / N
## Conversion de l'indicateur ELSA (à 100 mètres) en matrice
base_elsa <- terra::values(elsa_100, format = "matrice")
## Création d'une matrice qui contiendra les pseudo valeurs de p
p_matrix <- matrix(0, nrow = nrow(base_elsa), ncol = ncol(base_elsa))
## Lancement des 999 simulations
for(i in 1:999){
  # pour chaque simulation, nous tirons au hasard la valeur de chaque pixel
  sim_values <- sample(c(0,1,2,3), size = N, replace = TRUE, prob = probs[1:4])
  # Nous enregistrons les valeurs tirées au hasard dans une matrice
  dim(sim_values) <- dim(base_elsa)
  # Calcul de la valeur de ELSA simulée
  sim_elsa <- elsa_raster(sim_values, w100, matrice_dissim)
  # Comparaison des valeurs de l'indicateur ELSA (valeurs simulées versus observées)
  comp_matrix <- base_elsa >= sim_elsa
  # Ajout à notre matrice des valeurs de p
  p_matrix <- p_matrix + comp_matrix
}
```

```
## Conversion des comptages en pseudo valeur de p
p_values <- p_matrix / (999+1)
```

Dans la matrice obtenue, les pseudos valeurs de *p* représentent la probabilité que l'indice de *ELSA* obtenu sur les données réelles soit inférieur à celui obtenu si les données étaient réparties aléatoirement sur le territoire. Une pseudo valeur de *p* inférieure à 0,001 signifie donc qu'il n'y aurait que 0,1 % de chance que le hasard produise un patron d'autocorrélation spatiale plus fort que celui observé avec les données initiales.

```
raster_p_vals <- laval_data
values(raster_p_vals) <- p_values <= 0.001
tmap_mode("plot")

Carte1 <- tm_shape(raster_p_vals) +
  tm_raster(palette = c("grey", "red"),
            style = 'cat',
            title = "Significativité",
            labels = c("Non significatif (p > 0,001)",
                       "Significatif (p <= 0,001)")) +
  tm_layout(legend.outside = TRUE,
            frame = FALSE,
            legend.outside.position = c("bottom", "center"))

Carte2 <- tm_shape(elsa_100) +
  tm_raster(palette = "RdYlGn", n = 6,
            legend.format = list(text.separator = "à"),
            title = "ELSA 100 mètres") +
  tm_layout(legend.outside = TRUE,
            frame = FALSE,
            legend.outside.position =  c("bottom", "center"))

Carte3 <- tm_shape(laval_data) +
  tm_raster(palette = c("#fbd4b4", "#e36c0a", "#92d050", "#76923c"),
            style = 'cat',
            title = "Classe (catégorie)",
            labels = c("1. Minéral bas",
                       "2. Minéral haut",
                       "3. Végétal bas",
                       "4. Végétal haut"))+
  tm_layout(legend.outside = TRUE,
            frame = FALSE,
            legend.outside.position =  c("bottom", "center"))

tmap_arrange(Carte3, Carte2, Carte1, ncol=3)
```

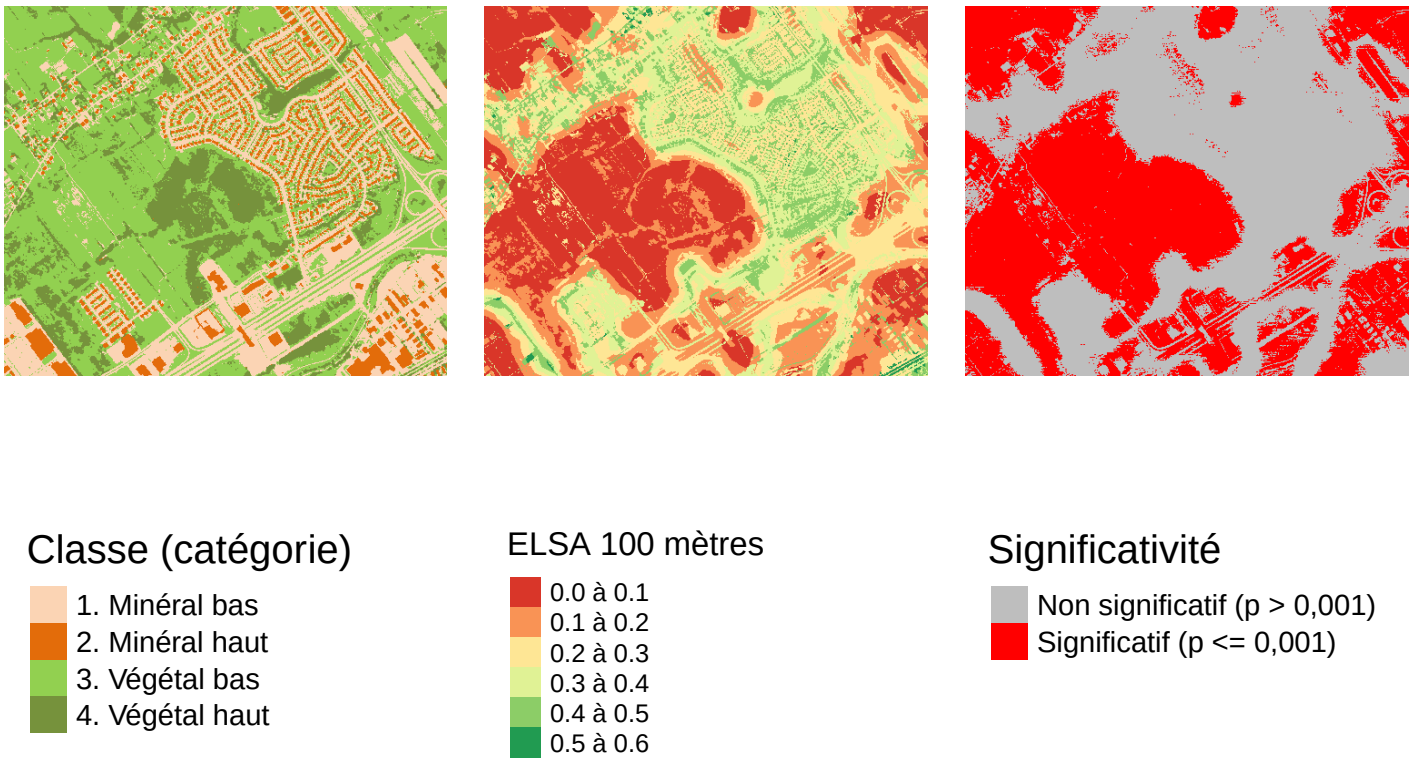


FIGURE 2.39 – Cartographie des valeurs de p de l'indicateur ELSA

La figure 2.39 montre en rouge les pixels situés dans des secteurs avec une forte autocorrélation spatiale, soit ceux principalement végétalisés ou principalement minéralisés.

2.5 Quiz de révision du chapitre

Questions

- **Parmi les matrices de pondération spatiale ci-dessous, lesquelles sont des matrices de contiguïté?**
  - Partage d'un nœud
  - Partage d'un segment
  - Partage d'un nœud et ordre d'adjacence
  - Partage d'un segment et ordre d'adjacence
  - Connectivité selon la distance

  Relisez au besoin la section 2.2.
- **En anglais, comment est appelée une matrice selon le partage d'un nœud?**
  - Rook
  - Queen

  Relisez au besoin le début de la section 2.2.
- **Comparativement à une matrice de l'inverse de la distance, une matrice de l'inverse de la distance au carré accorde un poids plus important aux entités proches.**

  - Vrai
  - Faux

  Relisez au besoin la section 2.2.2.3.
- **Quels sont les avantages de la standardisation en ligne des matrices de pondération spatiale?**
  - La somme de chaque ligne est égale à 1.
  - La somme de l'ensemble de la matrice est égale au nombre d'entités spatiales.
  - La standardisation permet de comparer la dépendance spatiale selon différentes matrices.
  - La standardisation augmente la vitesse des calculs.

  Relisez au besoin la section 2.2.3.
- **Quelle est la différence entre les deux mesures locales de Getis et Ord?**
  - Gi* tient compte à la fois des valeurs des entités voisines ou proches, mais aussi de celle de i.
  - Contrairement à Gi, Gi* a un z-score.

  Relisez au besoin la section 2.4.1.
- **Parmi les quatre catégories de la typologie basée sur le nuage de points du I de Moran, lesquelles renvoient à de l'autocorrélation spatiale locale positive?**
  - HH
  - LL
  - HL
  - LH

  Relisez au besoin la section 2.4.3.
- **Le village gaulois correspond à quelle catégorie?**
  - HH
  - LL
  - HL
  - LH

  Relisez au besoin la section 2.4.3.
- **Quelles sont les trois manières de tester la significativité des mesures d'autocorrélation globales?**
  - Avec l'hypothèse de la normalité
  - En relançant plusieurs fois les calculs
  - Avec l'hypothèse de la randomisation
  - Avec la méthode Monte-Carlo (habituellement avec 999 échantillons)

  Relisez au besoin la section 2.3.1.3.

Réponses

- Parmi les matrices de pondération spatiale ci-dessous, lesquelles sont des matrices de contiguïté?
  - Partage d'un nœud
  - Partage d'un segment
  - Partage d'un nœud et ordre d'adjacence
  - Partage d'un segment et ordre d'adjacence
- En anglais, comment est appelée une matrice selon le partage d'un nœud?
  - Queen

— Comparativement à une matrice de l'inverse de la distance, une matrice de l'inverse de la distance au carré accorde un poids plus important aux entités proches.
  — Vrai
— Quels sont les avantages de la standardisation en ligne des matrices de pondération spatiale?
  — La somme de chaque ligne est égale à 1.
  — La somme de l'ensemble de la matrice est égale au nombre d'entités spatiales.
  — La standardisation permet de comparer la dépendance spatiale selon différentes matrices.
— Quelle est la différence entre les deux mesures locales de Getis et Ord?
  — Gi* tient compte à la fois des valeurs des entités voisines ou proches, mais aussi de celle de i.
— Parmi les quatre catégories de la typologie basée sur le nuage de points du I de Moran, lesquelles renvoient à de l'autocorrélation spatiale locale positive?
  — HH
  — LL
— Le village gaulois correspond à quelle catégorie?
  — HL
— Quelles sont les trois manières de tester la significativité des mesures d'autocorrélation globales?
  — Avec l'hypothèse de la normalité
  — Avec l'hypothèse de la randomisation
  — Avec la méthode Monte-Carlo (habituellement avec 999 échantillons)

2.6 Exercices de révision

Exercice

Exercice 2. Construction de matrices de pondération spatiale
Construisez les matrices de pondération spatiale suivante pour la région métropolitaine de Québec :

- Matrice de pondération spatiale selon le partage d'un segment commun (voir la section 2.2.4.1).
- Matrice de pondération spatiale selon l'inverse de la distance au carré, à partir de la distance maximale et un SR et son voisin le plus proche (voir la section 2.2.4.4).
- Matrice de pondération spatiale selon le critère des plus proches voisins ($k = 2$) (voir la section 2.2.4.5).

Complétez le code ci-dessous.

```
library(sf)
library(spdep)
library(tmap)
## Importation de la couche des secteurs de recensement
SRQc <- st_read(dsn = "data/chap02/exercice/RMRQuebecSR2021.shp", quiet=TRUE)

## Matrice selon le partage d'un segment (Rook)
Rook <- À compléter
W.Rook <- À compléter

## Coordonnées des centroïdes des entités spatiales
coords <- st_coordinates(st_centroid(SRQc))

## Matrice de l'inverse de la distance réduite
# Trouver le plus proche voisin avec la fonction knn2nb
k1 <- À compléter
plusprochevoisin.max <- max(unlist(nbdists(k1,coords)))
# Voisins les plus proches avec le seuil de distance maximal
Voisins.DistMax <- À compléter
# Distances avec le seuil maximum
distances <- À compléter
# Inverse de la distance au carré
InvDistances2 <- À compléter
# Matrices de pondération spatiale standardisées en ligne
W_InvDistances2 <- À compléter

## Matrice des plus proches voisins avec k = 2
k2 <- À compléter
W.k2 <-  À compléter
```

Correction à la section 12.2.2.

Exercice

Exercice 3. Calcul du I de Moran global
Calculez le I de Moran global pour la variable `D1pct` (pourcentage du premier décile de revenu des familles économiques) de la couche `SRQc` avec les différentes matrices de pondération spatiale (voir la section 2.3.2.2). Complétez le code ci-dessous.

```
library(sf)
library(spdep)
library(tmap)
## Cartographie de la variable
tm_shape(SRQc)+
  tm_polygons(col="D1pct", title = "Premier décile de revenu (%)",
              style="quantile", n=5, palette="Greens")+
  tm_layout(frame = F)+tm_scale_bar(c(0,5,10))

## I de Moran avec la méthode Monte-Carlo avec 999 permutations
# utilisez la fonction moran.mc
# avec la matrice W.Rook
À compléter
# avec la matrice W_InvDistances2Reduite
À compléter
# avec la matrice W.k2
À compléter
```

Quelle matrice de pondération spatiale donne la dépendance spatiale la plus forte? Correction à la section 12.2.3.

Exercice

Exercice 3. Mesures d'autocorrélation spatiale locales
Calculez et cartographiez les mesures d'autocorrélation spatiale locale pour la variable `D1pct` de la couche `SRQc` avec la matrice spatiale `W.Rook` :

- Mesure G_i de Getis et Ord (voir la section 2.4.1).
- Typologie du nuage de points du I de Moran avec la fonction `localmoran` (voir la section 2.4.3).

Complétez le code ci-dessous.

```
#####################
## Calcul du Z(Gi)
#####################
SRQc$D1pct_localGetis <- localG(À compléter,
                                À compléter,
                                zero.policy=TRUE)
# Définition des intervalles et des noms des classes
classes.intervalles = À compléter
classes.noms = c("Point froid (p = 0,001)",
                 "Point froid (p = 0,01)",
                 "Point froid (p = 0,05)",
                 "Non significatif",
                 "Point chaud (p = 0,05)",
                 "Point chaud (p = 0,01)",
                 "Point chaud (p = 0,001)")
## Création d'un champ avec les noms des classes
SRQc$D1pct_localGetisP <- cut(SRQc$D1pct_localGetis,
                              breaks = classes.intervalles,
                              labels = classes.noms)
## Cartographie
À compléter

#####################
## Typologie LISA
#####################
## Cote Z (variable centrée réduite)
zx <- À compléter
## variable X centrée réduite spatialement décalée avec une matrice Rook
wzx <- lag.listw(À compléter)
## I de Moran local (notez que vous pouvez aussi utiliser la fonction localmoran_perm)
localMoranI  <- localmoran(À compléter)
plocalMoranI <- localMoranI[, 5]
## Choisir un seuil de signification
signif = 0.05
## Construction de la typologie
Typologie <- ifelse(zx > 0 & wzx > 0, "1. HH", NA)
Typologie <- ifelse(zx < 0 & wzx < 0, "2. LL", Typologie)
Typologie <- ifelse(zx > 0 & wzx < 0, "3. HL", Typologie)
Typologie <- ifelse(zx < 0 & wzx > 0, "4. LH", Typologie)
Typologie <- ifelse(plocalMoranI > signif, "Non sign", Typologie)  # Non significatif
## Enregistrement de la typologie dans un champ
SRQc$TypoIMoran.D1pct <- Typologie
## Couleurs
Couleurs <- c("red", "blue", "lightpink", "skyblue2", "lightgray")
names(Couleurs) <- c("1. HH","2. LL","3. HL","4. LH","Non sign")
## Cartographie
```

Partie 3. Méthodes de répartition ponctuelle et de détections d'agrégats spatiaux

3 Méthodes de répartition ponctuelle

Dans ce chapitre, nous abordons les méthodes de répartition ponctuelle qui permettent de décrire un semis de points dans un espace géographique donné. En géomatique appliquée, ces méthodes sont fréquemment employées dans le cadre d'études rattachées à des champs disciplinaires variés comme :

- En études urbaines, pour décrire la répartition de services et d'équipements collectifs à travers une ville afin de vérifier s'ils sont équitablement répartis ou à l'inverse, concentrés dans certaines parties de la ville.
- En biologie, pour décrire la répartition d'espèces fauniques ou végétales dans un territoire.
- En criminologie, pour analyser la répartition spatiale d'un ou de plusieurs types de crimes.
- En épidémiologie, pour comprendre l'évolution de la répartition spatiale des cas d'une maladie infectieuse.
- En transport, pour analyser la répartition d'accidents.
- Et même en sciences de l'activité physique, pour analyser la distribution spatiale de personnes pratiquant un sport sur un terrain de soccer, de rugby, de tennis, de baseball, etc.

Package

Liste des *packages* utilisés dans ce chapitre

- Pour importer et manipuler des fichiers géographiques :
  - `sf` pour importer et manipuler des données vectorielles.
- Pour construire des cartes et des graphiques :
  - `tmap` pour construire des cartes thématiques.
  - `ggplot2` est un *package* pour construire des graphiques.
- Pour les analyses de méthodes de répartition ponctuelle :
  - `spatstat` est sans aucun doute le meilleur *package*.
  - `sparr` pour calculer la STKDE, soit l'estimation de la densité par noyau spatio-temporel (*Space-Time Kernel density Estimatation*).

Objectif

Pour décrire la distribution d'un semis de points, nous voyons les méthodes suivantes :

- La fréquence et la densité des points dans l'espace d'étude.
- L'analyse centrographique :
  - Paramètres de tendance centrale (centre moyen et point central).
  - Dispersion du semis de points (distance standard) et ses différentes représentations graphiques (cercle et ellipse de distance standard).
- L'arrangement spatial du semis de points :
  - Méthode du plus proche voisin.
  - Méthode des quadrats.
- La cartographie de la densité :
  - Dans une maille irrégulière (des polygones de forme et de taille différentes).
  - Dans une maille régulière (estimation de la densité par noyau – *kernel density estimation, KDE*).

3.1 Fréquence et densité des points dans l'espace d'étude

La **fréquence** est tout simplement le nombre de points présents dans une région donnée (par exemple, le nombre d'hôpitaux, de stations de métro, d'arbres, etc.). La **densité** est le ratio entre la fréquence et la *superficie totale* de la région donnée ou la *population*.

Par exemple, le tableau 3.1 renvoie le nombre et la densité des stations de métro (pour 10 000 habitants) pour trois villes. Interprétez ces chiffres avec prudence, car ils varient en fonction de la taille du territoire retenu pour les trois villes.

Tableau 3.1 – Fréquence et densité des stations de métro dans trois villes

Ville	Population	Stations de métro	Densité (stations / 10 000 hab.)
New York	8 804 000	424	0,5
Paris	2 165 000	309	1,4
Île de Montréal	2 004 000	68	0,3

3.2 Analyse centrographique

L'analyse centrographique est une approche qui a été largement utilisée durant les décennies 1990 et 2000. Son utilisation est parfois critiquée pour deux raisons principales : 1) elle ne décrit que partiellement le semis de points; 2) aucun test d'inférence n'est calculé. Quoi qu'il en soit, elle permet d'explorer les données avant de se lancer dans des analyses plus avancées.

Note

Utilisation de l'analyse centrographique au Québec et dans le monde francophone
Marius Thériault (géographe et professeur émérite à l'Université Laval) a largement contribué à la popularité de l'analyse centrographique au Québec et ailleurs. Il est le créateur de *MapStat*, un module développé avec le langage MapBasic intégré dans le logiciel SIG MapInfo permettant de réaliser une analyse centrographique avant même qu'elle soit implémentée dans ArcGIS. En guise d'exemple, les études suivantes utilisent l'analyse centrographique calculée avec *MapStat* (López Castro, Thériault et Vandersmissen 2015; Barbonne, Villeneuve et Thériault 2007). Consultez-les au besoin.

3.2.1 Paramètres de tendance centrale d'un semis de points

Les deux principaux paramètres de tendance centrale d'un semis de points sont le centre moyen et le point central qui peuvent être ou non pondérés.

3.2.1.1 Centre moyen

Le centre moyen (cm) est le centre de gravité du semis de points et correspond aux valeurs des moyennes arithmétiques des coordonnées géographiques (équation 3.1).

$$(\bar{x}_{cm}, \bar{y}_{cm}) = \left(\frac{\Sigma_{i=1}^{n} x_i}{n}, \frac{\Sigma_{i=1}^{n} y_i}{n} \right) \text{ avec :} \tag{3.1}$$

- $(\bar{x}_{cm}, \bar{y}_{cm})$, les coordonnées géographiques du point moyen.
- n, le nombre de points dans la couche géographique.
- x_i et y_j, les coordonnées géographiques du point i.

Il est possible de calculer le centre moyen en pondérant chacun des points du semis avec la valeur d'une variable donnée (équation 3.2). Ainsi, l'importance accordée à chacun des points n'est pas la même. Par exemple, nous pourrions calculer le centre moyen pondéré (cmp) des cliniques médicales d'une ville en pondérant chaque clinique par le nombre de médecins, ou encore le point moyen des hôpitaux pondéré par le nombre de lits. Autre exemple, avec un jeu de données sur les arbres dans une érablière, nous pourrions utiliser une pondération basée sur le diamètre à la hauteur de la poitrine (DHP) afin d'accorder un poids plus important aux arbres de plus « grand volume ».

Note

Moyenne pondérée
Pour un rappel sur le calcul d'une moyenne pondérée, consultez la section intitulée *Statistiques descriptives pondérées* (Apparicio et Gelb 2022).

$$(\bar{x}_{cmp}, \bar{y}_{cmp}) = \left(\frac{\Sigma_{i=1}^{n} w_i x_i}{\Sigma_{i=1}^{n} w_i}, \frac{\Sigma_{i=1}^{n} w_i y_i}{\Sigma_{i=1}^{n} w_i} \right) \text{ avec :} \tag{3.2}$$

- n, x_i et y_j étant définis plus haut.
- $(\bar{x}_w, \bar{y}_w)$, les coordonnées géographiques du point moyen pondéré.
- w_i, la valeur de pondération associée au point i.

Le centre moyen et le centre moyen pondéré sont des mesures très utiles pour comparer la distribution de plusieurs semis de points (de différents services et équipements collectifs par exemple) ou encore pour décrire l'évolution dans le temps de la répartition d'un semis de points. L'analyse du déplacement du centre moyen (pondéré) à différentes dates nous informe ainsi de l'évolution du phénomène à l'étude et plus spécifiquement, de son orientation et de sa direction.

Astuce

Exemple d'utilisation temporelle du centre moyen pondéré
Un exemple classique d'utilisation du centre moyen pondéré sur plusieurs années est l'évolution du centre moyen pondéré de la population des États-Unis de 1790 à 2020. Il est calculé à partir des centroïdes des comtés américains et de la population comme variable de pondération extraite des recensements de l'*US Census Bureau*. Bien entendu, le centre moyen pondéré se déplace vers l'ouest.
Vous pouvez consulter cette carte ou visionner cette courte vidéo YouTube ludique.

Notez que les centres moyen et moyen pondéré peuvent aussi être calculés sur des données géographiques comprenant des valeurs d'élévation (x, y, z) :

$$(\bar{x}_{cm}, \bar{y}_{cm}, \bar{z}_{cm}) = \left(\frac{\Sigma_{i=1}^{n} x_i}{n}, \frac{\Sigma_{i=1}^{n} y_i}{n}, \frac{\Sigma_{i=1}^{n} z_i}{n} \right) \tag{3.3}$$

$$(\bar{x}_{cmp}, \bar{y}_{cmp}, \bar{z}_{cmp}) = \left(\frac{w_i \Sigma_{i=1}^{n} x_i}{\Sigma_{i=1}^{n} w_i}, \frac{w_i \Sigma_{i=1}^{n} y_i}{\Sigma_{i=1}^{n} w_i}, \frac{w_i \Sigma_{i=1}^{n} z_i}{\Sigma_{i=1}^{n} w_i} \right) \tag{3.4}$$

3.2.1.2 Point central

Le point central d'un semis de points est celui qui minimise la somme des distances le séparant de tous les autres points (équation 3.5). Tout comme le point moyen, il est aussi possible de calculer le point central pondéré (équation 3.6).

$$pc = \text{Min}\left(\Sigma_{i=1}^{n} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \right); \text{avec } i \neq j \tag{3.5}$$

$$pcp = \text{Min}\left(\Sigma_{i=1}^{n} w_i \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \right); \text{avec } i \neq j \tag{3.6}$$

Attention

Différence entre point central et centre moyen
Contrairement au centre moyen, le point central fait partie du semis de points initial.
Utilité du point central pondéré
Imaginez que des personnes étudiantes en géographie et en géomatique de toutes les universités du Québec souhaitent organiser une rencontre en présence. Calculer le point central, pondéré par le nombre de personnes étudiantes par établissement participant à la rencontre, permet d'identifier l'université la plus centrale. Idéalement, il faudrait obtenir ce point central pondéré avec les distances temps calculées avec un réseau routier.

3.2.2 Paramètres de dispersion d'un semis de points

Les deux principaux paramètres de dispersion d'un semis de points sont la distance standard et la distance standard pondérée. La dispersion d'un semis de points peut être représentée graphiquement avec une enveloppe convexe, un cercle de rayon de standard ou une ellipse (avec ou sans pondération).

3.2.2.1 Distance standard et distance standard pondérée

En statistique univariée, l'écart-type (équation 3.7) est une mesure de dispersion bien connue : plus la valeur de l'écart-type est élevée, plus la dispersion des valeurs de la variable autour de la moyenne (μ) est importante.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}} \tag{3.7}$$

Note

Écart-type
Pour un rappel sur l'écart-type, consultez la section intitulée *Paramètres de dispersion* (Apparicio et Gelb 2022).

Distance standard (pondérée ou non) des *X* et des *Y*

De manière analogue à l'écart-type, nous pouvons calculer la distance standard pour les coordonnées X et pour les coordonnées Y des points, soit l'écart moyen respectif au centre moyen (équation 3.8) ou au centre moyen pondéré (équation 3.9).

$$\sigma_x = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x}_{cm})^2}{n}} \text{ et } \sigma_y = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \bar{y}_{cm})^2}{n}} \tag{3.8}$$

$$\sigma_{xw} = \sqrt{\frac{\sum_{i=1}^{n} w_i(x_i - \bar{x}_{cmp})^2}{\sum_{i=1}^{n} w_i}} \text{ et } \sigma_{yw} = \sqrt{\frac{\sum_{i=1}^{n} w_i(y_i - \bar{y}_{cmp})^2}{\sum_{i=1}^{n} w_i}} \tag{3.9}$$

Distance standard (pondérée ou non)

Nous pouvons aussi calculer la distance standard (ds) sans pondération (équation 3.10) et avec pondération (équation 3.11). Plus elle est forte, plus les points sont dispersés autour du centre moyen ou du centre moyen pondéré. Inversement, une faible distance standard indique une concentration de points autour du centre moyen.

$$ds = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x}_{cm})^2}{n} + \frac{\sum_{i=1}^{n}(y_i - \bar{y}_{cm})^2}{n}} \tag{3.10}$$

$$ds_w = \sqrt{\frac{\sum_{i=1}^{n} w_i(x_i - \bar{x}_{cmp})^2}{\sum_{i=1}^{n} w_i} + \frac{\sum_{i=1}^{n} w_i(y_i - \bar{y}_{cmp})^2}{\sum_{i=1}^{n} w_i}} \tag{3.11}$$

De nouveau, ces mesures peuvent être adaptées pour des points avec une élévation (x, y, z) :

$$ds = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x}_{cm})^2}{n} + \frac{\sum_{i=1}^{n}(y_i - \bar{y}_{cm})^2}{n} + \frac{\sum_{i=1}^{n}(z_i - \bar{z}_{cm})^2}{n}} \tag{3.12}$$

$$ds_w = \sqrt{\frac{\sum_{i=1}^{n} w_i(x_i - \bar{x}_{cmp})^2}{\sum_{i=1}^{n} w_i} + \frac{\sum_{i=1}^{n} w_i(y_i - \bar{y}_{cmp})^2}{\sum_{i=1}^{n} w_i} + \frac{\sum_{i=1}^{n} w_i(z_i - \bar{z}_{cmp})^2}{\sum_{i=1}^{n} w_i}} \tag{3.13}$$

3.2.2.2 Représenter la dispersion : cercle de distance standard et ellipse

La dispersion d'un semis de points peut être représentée de quatre manières différentes :

1. Une enveloppe convexe des points décrite à la section 1.2.2.5).
2. Un rectangle centré sur le centre moyen avec les déviations des coordonnées X et Y pondérées ou non (équation 3.8 et équation 3.9 décrites précédemment). Dans le cas de données comprenant l'élévation, la forme géométrique est un parallélépipède rectangle.
3. Un cercle de rayon de distance standard pondérée ou non (équation 3.10 et équation 3.11) décrites précédemment). Dans le cas de données comprenant l'élévation, la forme géométrique est une sphère de rayon de distance standard.
4. Une ellipse de distance standard pondérée ou non. Dans le cas de données comprenant l'élévation, la forme géométrique est un ellipsoïde.

Prenons un jeu de données fictives pour décrire ces quatre représentations graphiques. Imaginons que nous avons observé le parc de la Laurentie à Sherbrooke pour comprendre son utilisation. Pour collecter des données sur la localisation des personnes utilisatrices, nous aurions pu utiliser un questionnaire dans QField ou ArcGIS Survey 123. La figure 3.1 illustre la localisation de dix personnes (points rouges).

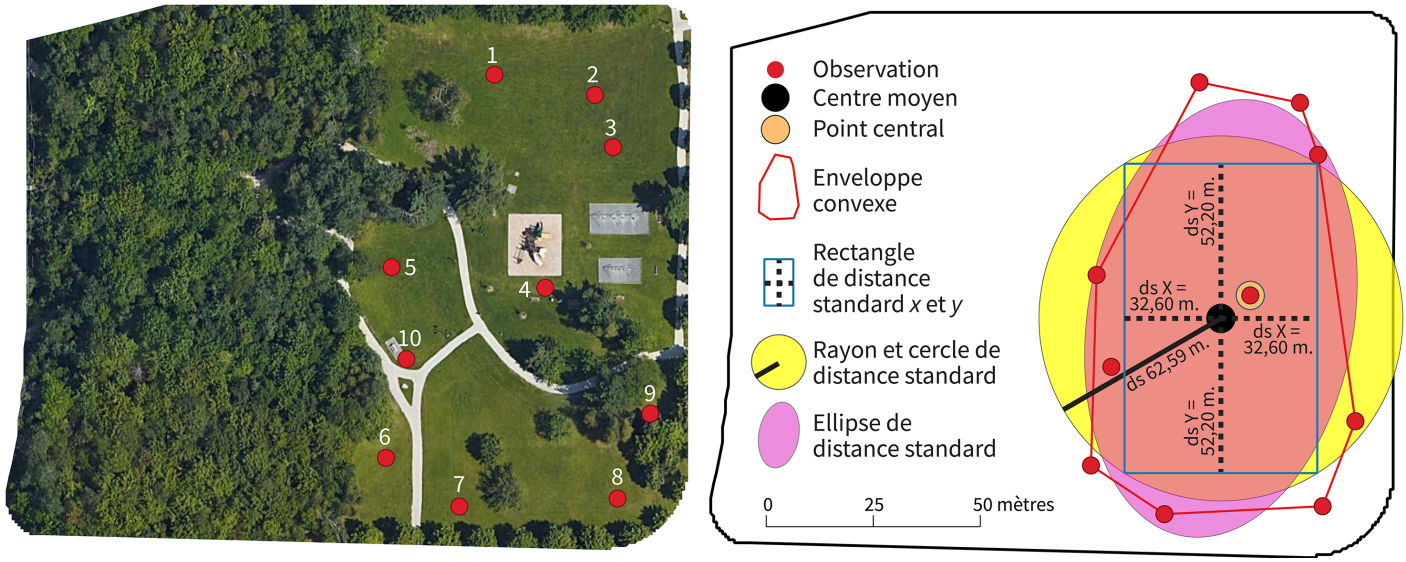


FIGURE 3.1 – Données fictives sur des personnes utilisatrices du parc de la Laurentie à Sherbrooke

À partir de ces dix points, nous obtenons :

— Les coordonnées du **centre moyen** qui sont égales à -8 007 869 et 5 685 921, soit simplement les moyennes des coordonnées X et Y des dix observations.
— L'**enveloppe convexe des points** qui contient tous les points.
— **Le rectangle construit avec les déviations standards des *X* et des *Y*** qui est centré sur le centre moyen. À partir de ce point, nous ajoutons à l'est et à l'ouest la valeur de la distance standard des X et au nord et au sud celle des Y. Les valeurs de ces distances standards sont égales à 32,60 et 52,20 mètres (voir les calculs au tableau 3.2).
— La **distance standard** est égale à 61,59 mètres (voir les calculs au tableau 3.2). À partir de cette distance, nous traçons le cercle ayant comme rayon la distance standard.
— L'**ellipse de déviation de distance standard** (figure 3.2).

Tableau 3.2 – Calcul des distances standards des X et des Y et de la distance standard

Point	x_i	y_i	$(x_i - \bar{x}_{cm})^2$	$(y_i - \bar{y}_{cm})^2$	$(x_i - \bar{x}_{cm})^2 + (y_i - \bar{y}_{cm})^2$
1	-8007877	5686000	52,80	6347,70	6400,50
2	-8007843	5685993	716,90	5298,00	6014,90
3	-8007836	5685976	1082,00	3046,30	4128,30
4	-8007859	5685928	98,90	60,30	159,20
5	-8007911	5685935	1770,20	214,60	1984,80
6	-8007913	5685871	1934,80	2461,00	4395,80
7	-8007888	5685855	365,70	4363,40	4729,10
8	-8007835	5685857	1185,00	4016,80	5201,80
9	-8007824	5685886	2071,70	1203,50	3275,20
10	-8007906	5685904	1376,50	266,70	1643,30
n	10				
Somme	-80078693	56859207	10654,50	27278,30	37932,80
Moyenne	-8007869	5685921	1065,45	2727,83	3793,28
Racine carrée			32,60	52,20	61,59

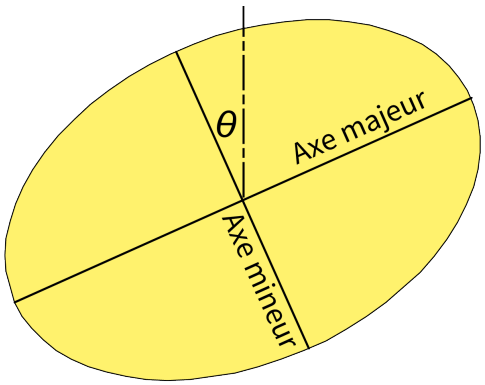


Figure 3.2 – Trois éléments composant une ellipse

⚠ **Attention**

Calcul des ellipses : des résultats qui varient d'un logiciel à l'autre…
Il existe plusieurs solutions pour tracer une ellipse. Pour une discussion détaillée de ces différentes solutions, lisez le court texte très intéressant de Martin Leroux.

- Ellipse de Yuill (Tveite 2020).
- Ellipse basée sur la covariance implémentée dans ArcGIS Pro.
- Ellipse de distance standard implémentée dans le logiciel CrimeStat.
- Ellipse avec la correction proposée par Wang et ses collègues (2015).

Notez qu'un plugin QGIS nommé *The QGIS Standard Deviational Ellipse Plugin* (Tveite 2020) intègre plusieurs de ces méthodes. Les résultats varient ainsi d'un logiciel à l'autre selon la méthode implémentée. Autrement dit, pour un même jeu de données ponctuelles, les ellipses obtenues avec ArcGIS Pro, ArcMap 9.3, ArcMap 10.x et QGIS seront différentes.

Quoi faire alors?
Quelle que soit la méthode utilisée, l'ellipse est toujours centrée sur le point moyen et a toujours le même angle de rotation. Par contre, la taille de l'ellipse (superficie) varie. Par conséquent, si vous souhaitez comparer des ellipses différentes, assurez-vous toujours qu'elles sont toutes obtenues dans le même logiciel et avec la même solution.

Aller plus loin

Calcul de l'ellipse selon la méthode implémentée dans CrimeStat
Ned Levine (2006; 2021), créateur du logiciel CrimeStat, propose les formulations suivantes pour le calcul d'une ellipse :

$$\theta = \frac{\arctan\left\{\left(\sum_{i=1}^{n} x_d^2 - \sum_{i=1}^{n} y_d^2\right) + \left[\left(\sum_{i=1}^{n} x_d^2 - \sum_{i=1}^{n} y_d^2\right)^2 + 4\left(\sum_{i=1}^{n} x_d y_d\right)^2\right]^{1/2}\right\}}{2\sum_{i=1}^{n} x_d y_d} \tag{3.14}$$

avec θ est la rotation de l'ellipse, $x_d = x_i - \bar{x}$ et $y_d = y_i - \bar{y}$.

$$\sigma_x = \sqrt{2 \times \frac{\sum_{i=1}^{n}\left((x_i - \bar{x})\cos\theta - (y_i - \bar{y})\sin\theta\right)^2}{n-2}} \tag{3.15}$$

$$\sigma_y = \sqrt{2 \times \frac{\sum_{i=1}^{n}\left((x_i - \bar{x})\sin\theta - (y_i - \bar{y})\cos\theta\right)^2}{n-2}} \tag{3.16}$$

$$l_x = 2\sigma_x \text{ et } l_y = 2\sigma_y \text{ et } S_e = \pi\sigma_x\sigma_y \tag{3.17}$$

avec l_x, l_y et S_e étant les longueurs de axes X et Y et la superficie de l'ellipse.

Prenons quatre situations fictives de répartition de dix personnes utilisatrices du parc de la Laurentie à Sherbrooke :

- **Situation A**. Les observations sont concentrées autour de l'aire de jeu.
- **Situation B**. Les observations sont dispersées dans la partie est du parc.
- **Situation C**. Les observations sont concentrées dans la partie nord du parc.
- **Situation D**. Les observations sont concentrées dans la partie nord du parc, excepté deux observations au sud.

Les cercles et les ellipses de distance standard (ds) centrés au centre moyen (cm) sont représentés à la (figure 3.4).

3.2.2.3 Comparaison de la dispersion de deux semis de points dans deux régions différentes

Pour comparer la dispersion de deux semis de points situés dans des régions de taille différente, il convient de supprimer les effets de taille des deux régions. Pour ce faire, nous divisons la **distance standard** ou la **distance standard pondérée** par la superficie de la région. Cette approche est donc très similaire au coefficient de variation en statistique univariée, soit le rapport entre l'écart-type et la moyenne.

Par exemple, si nous comparons les dispersions des personnes utilisatrices du parc de la Laurentie (0,078 ha) et parc du Mont-Bellevue (409 ha). Inévitablement, la valeur de la distance standard est plus forte pour le parc du Mont-Bellevue que celle du parc de la Laurentie. Il faut donc diviser chaque distance standard par la superficie associée.

3.2.2.4 Comparaison de la dispersion de deux semis de points dans la même région

Pour comparer la distribution spatiale de deux semis de points situés dans la même région, nous comparons leur cercle ou leur ellipse respective (par exemple, des points représentant des accidents l'été versus l'hiver ou encore deux espèces

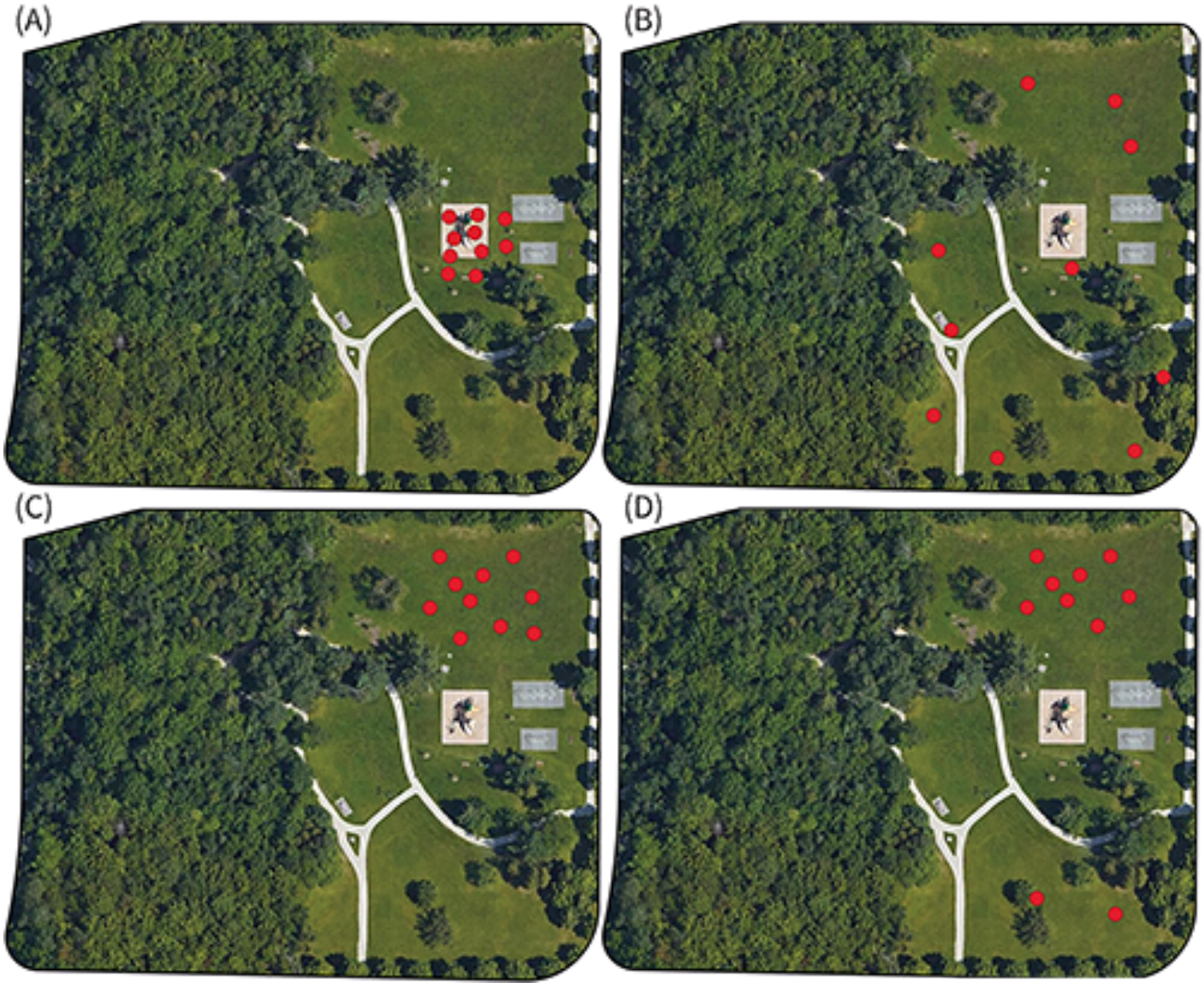


Figure 3.3 – Données fictives sur des personnes utilisatrices du parc de la Laurentie à Sherbrooke (quatre situations)

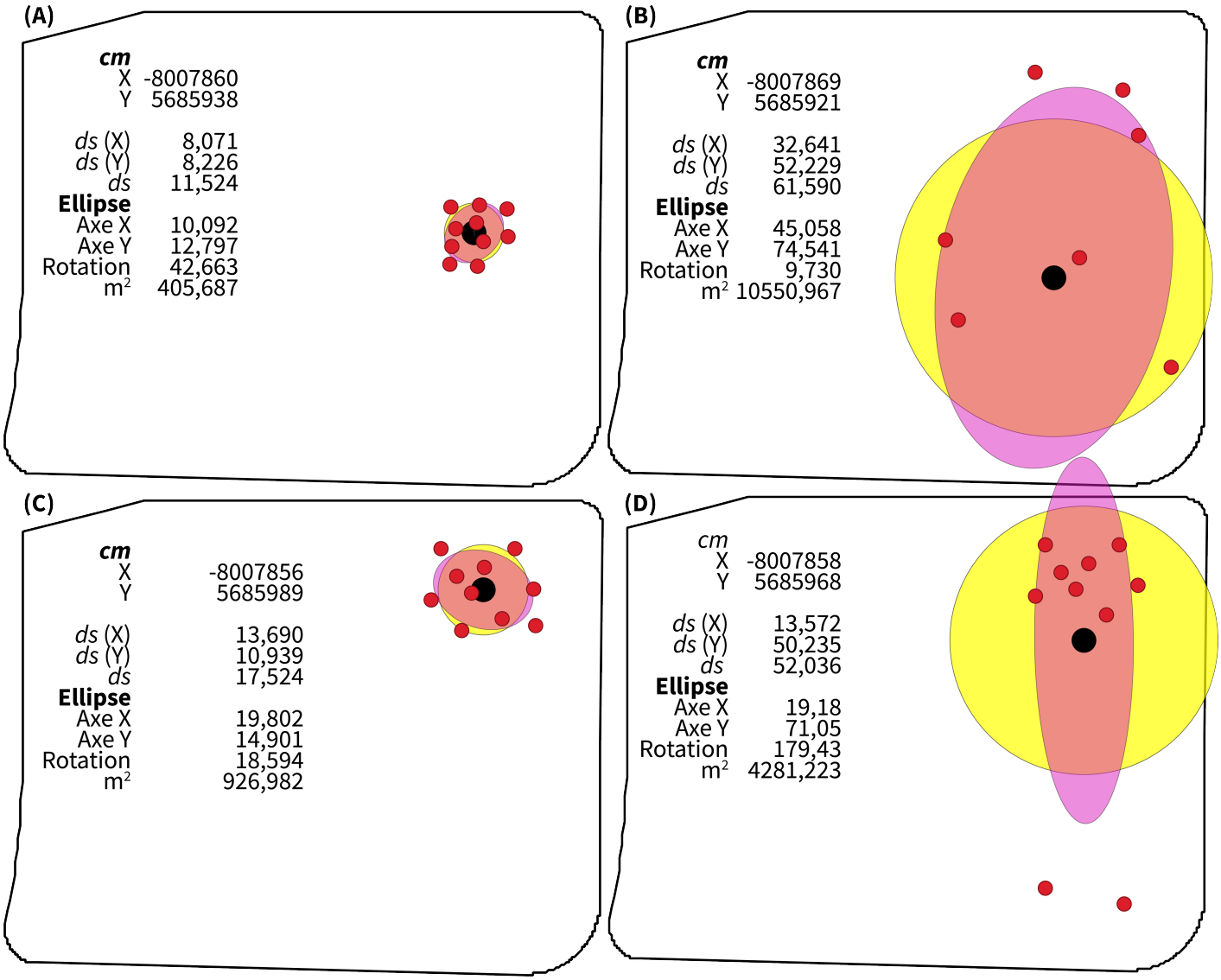


Figure 3.4 – Ellipse et cercle de distance standard pour les quatre situations

végétales sur le même territoire).

Une démarche similaire peut être appliquée à deux groupes de population rattachés à des entités polygonales : ils ont la même distribution spatiale si les deux ellipses de distance pondérée se juxtaposent significativement. David W.S. Wong (1999) propose d'ailleurs un indice dénommé S basé sur la comparaison des deux ellipses (équation 3.18). Le numérateur représente la surface d'intersection entre les deux ellipses tandis que le dénominateur représente leur surface d'union. L'indice varie de 0 à 1, soit respectivement d'une similitude parfaite à une dissemblance la plus grande entre les deux distributions spatiales :

- Si deux groupes de population ont des distributions spatiales identiques, les ellipses sont les mêmes et donc $E_i \cap E_j = E_i \cup E_j = 1$ et $S_{ij} = 1 - 1 = 0$.
- Si les deux groupes de population ont des distributions spatiales totalement différentes, les ellipses ne se touchent pas, alors $E_i \cap E_j = 0$ et la valeur de $S = 1 - 0 = 1$.

Bien évidemment, cet indice peut être étendu pour comparer les distributions de plus de deux groupes de population simultanément (équation 3.19).

$$S_{ij} = 1 - \frac{E_i \cap E_j}{E_i \cup E_j} \tag{3.18}$$

$$S = 1 - \frac{E_1 \cap E_2 \cap E_3 \cap \ldots \cap E_n}{E_1 \cup E_2 \cup E_3 \cup \ldots \cup E_n} \tag{3.19}$$

Voyons un exemple concret : calculons l'indice de Wong (1999) (équation 3.18) pour les ellipses de distances standards pondérées par les effectifs de propriétaires et de locataires par secteur de recensement pour la ville de Sherbrooke en 2021 (figure 3.5). D'emblée, nous constatons que les propriétaires ont une distribution plus dispersée que celle des locataires plus présents dans le centre de la ville.

Le code suivant permet d'obtenir l'indice de Wong (1999).

```
## Importation des deux ellipses
E1 <- st_read(dsn = "data/chap03/EllipseProprio.shp", quiet=TRUE)
E2 <- st_read(dsn = "data/chap03/EllipseLocataire.shp", quiet=TRUE)
## Intersection
Inter <- st_intersection(E1, E2)
## Union
Union <- st_union(E1, E2)
## Calcul de l'indice de Wong
Wong <- 1 - ( as.numeric(st_area(Inter)) / as.numeric(st_area(Union)))
print(paste0("Valeur de l'indice de Wong : ",round(Wong,3)))
```

```
[1] "Valeur de l'indice de Wong : 0.515"
```

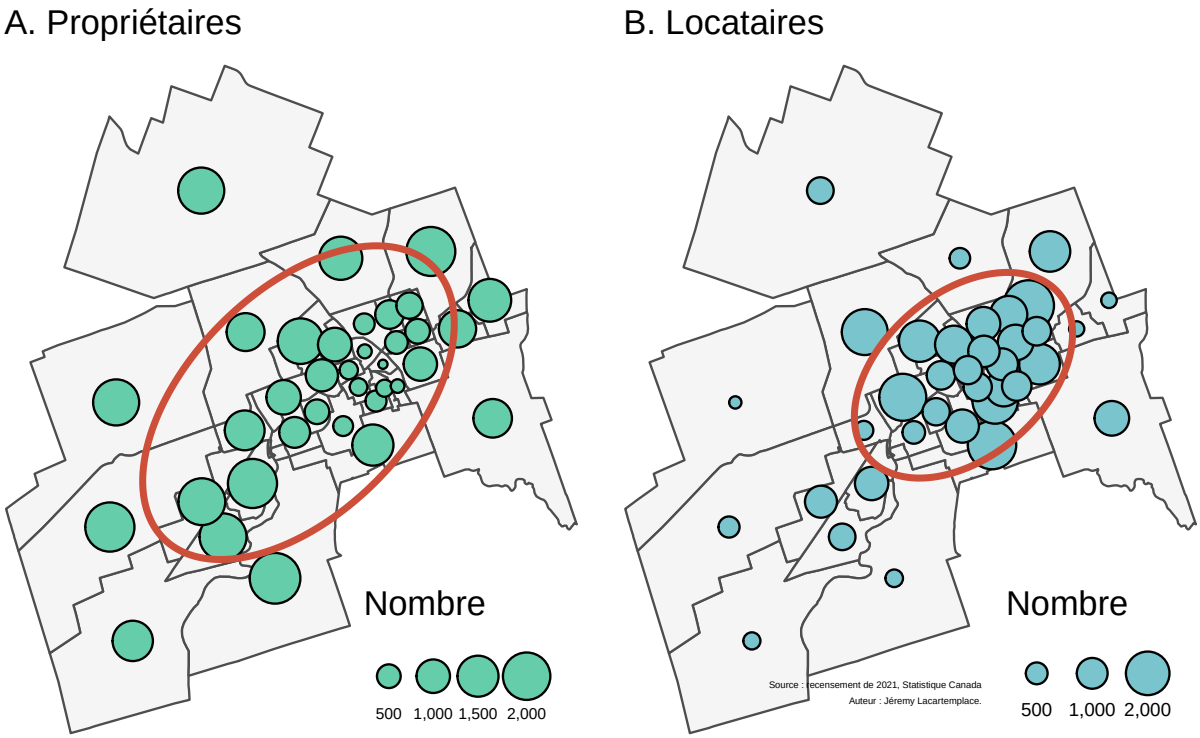


FIGURE 3.5 – Propriétaires et locataires dans la ville de Sherbrooke (avec ellipse de distance standard), 2021

3.2.3 Mise en œuvre de l'analyse centrographique dans R

3.2.3.1 Calcul de mesures non pondérées

Pour illustrer la mise en œuvre des différentes mesures de l'analyse centrographique dans R, nous utilisons un jeu de données ouvertes sur les incidents de sécurité publique de la ville de Sherbrooke. Dans le code ci-dessous, nous importons les données et constituons une couche `sf` par année (2019 à 2022) et extrayons les coordonnées géographiques.

```
library(sf)
library(tmap)
## Importation des données
Arrondissements <-  st_read(dsn = "data/chap03/Arrondissements.shp", quiet=TRUE)
Incidents <- st_read(dsn = "data/chap03/IncidentsSecuritePublique.shp", quiet=TRUE)
## Changement de projection
Arrondissements <- st_transform(Arrondissements, crs = 3798)
Incidents <- st_transform(Incidents, crs = 3798)
## Extaction des méfaits
Mefaits <- subset(Incidents, DESCRIPTIO == "Méfait")
# Méfaits par année
M2019 <- subset(Mefaits, ANNEE==2019)
M2020 <- subset(Mefaits, ANNEE==2020)
M2021 <- subset(Mefaits, ANNEE==2021)
M2022 <- subset(Mefaits, ANNEE==2022)
```

```
# Coordonnées géographiques
xy.2019 <- st_coordinates(M2019)
xy.2020 <- st_coordinates(M2020)
xy.2021 <- st_coordinates(M2021)
xy.2022 <- st_coordinates(M2022)
```

Les méfaits par année sont présentés à la figure 3.6.

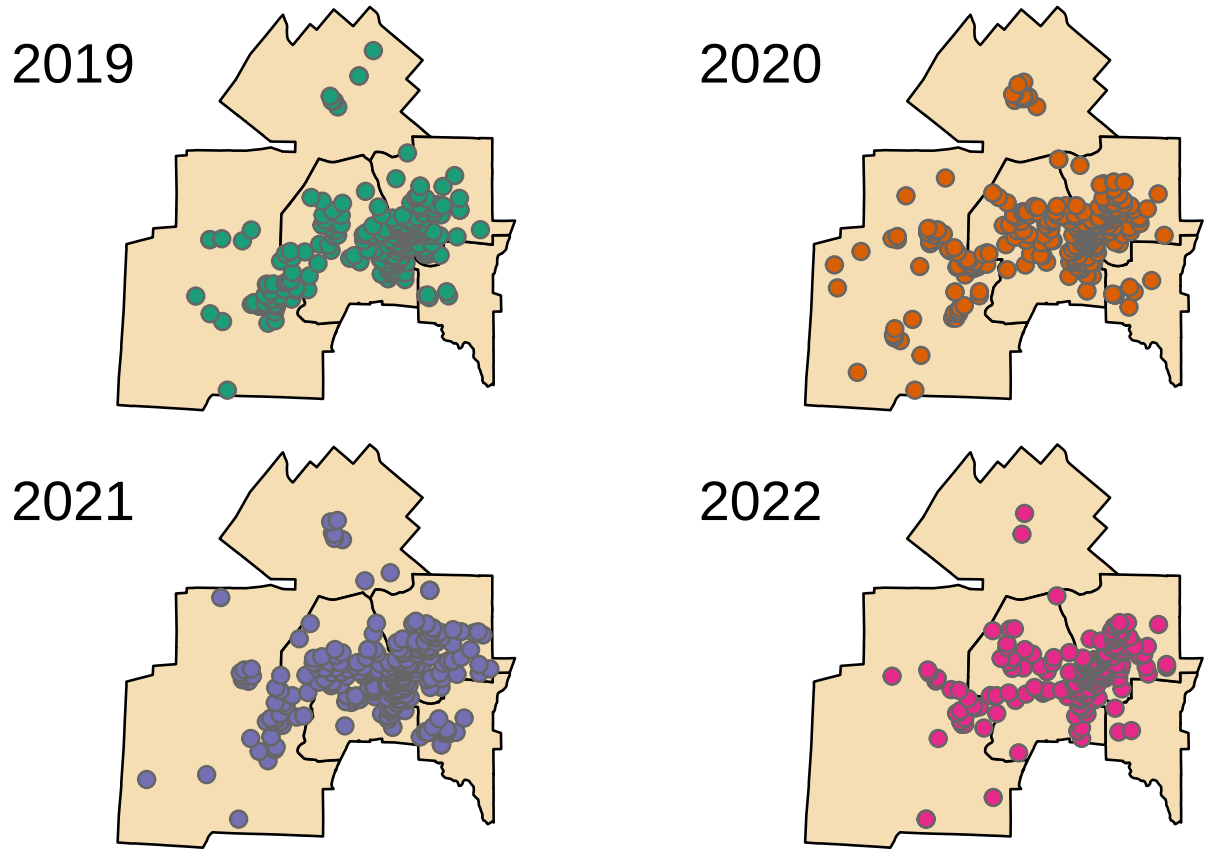


Figure 3.6 – Localisation des méfaits par année, ville de Sherbrooke, 2021

Centre moyen

Avec la fonction `mean`, nous pouvons calculer les valeurs moyennes sur les coordonnées X et Y, puis créer un objet `sf` avec les centres moyens.

```
## Récupération de la projection cartographique dans une variable
ProjCarto <- st_crs(Mefaits)
## Calcul du centre moyen pour une année (2019)
print(c(mean(xy.2019[,1]), mean(xy.2019[,2])))
```

```
[1] 649990.3 157059.3
```

```
## Calcul pour toutes les années
# vecteur pour les moyennes des X
X.moy <- c(mean(xy.2019[,1]), mean(xy.2020[,1]), mean(xy.2021[,1]), mean(xy.2022[,1]))
# Vecteur pour les moyennes des Y
Y.moy <- c(mean(xy.2019[,2]), mean(xy.2020[,2]), mean(xy.2021[,2]), mean(xy.2022[,2]))
# Enregistrement dans un objet sf
CentreMoyen <- data.frame(Annee = c("2019", "2020", "2021", "2022"),
                          X = X.moy,
                          Y = Y.moy,
                          CMx = X.moy,
                          CMy = Y.moy)
CentreMoyen <- st_as_sf(CentreMoyen, coords = c("X", "Y"), crs = ProjCarto)
# Affichage des résultats
print(CentreMoyen)
```

```
Simple feature collection with 4 features and 3 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 649696 ymin: 157059.3 xmax: 650663.7 ymax: 157544.8
Projected CRS: NAD83 / MTQ Lambert
  Annee      CMx      CMy                  geometry
1  2019 649990.3 157059.3 POINT (649990.3 157059.3)
2  2020 649696.0 157544.8   POINT (649696 157544.8)
3  2021 650663.7 157214.8 POINT (650663.7 157214.8)
4  2022 650442.5 157237.0   POINT (650442.5 157237)
```

Point central

Le code ci-dessous illustre comment identifier le point central, qui fait partie du jeu de données, pour l'année 2019.

```
## Calcul de la matrice de distances entre les points de l'année 2019
DistMatrice2019 <- as.matrix(dist(xy.2019, method = "euclidean", diag = TRUE, upper = TRUE))
## Somme de chaque ligne de la matrice, soit la somme des distances à tous les autres points
M2019$DistATous <- rowSums(DistMatrice2019)
## Sélection du point avec plus petite distance à tous les autres
PointCentral2019 <- subset(M2019, M2019$DistATous==min(M2019$DistATous))
```

Distance standard sur les coordonnées X et Y et distance standard combinée

Le code ci-dessous permet de calculer les trois distances standards.

```
## Calcul de la distance standard pour une année (2019)
# Distance standard sur les coordonnées X et Y
c(sqrt(mean((xy.2019[,1] - mean(xy.2019[,1]))^2)),
  sqrt(mean((xy.2019[,2] - mean(xy.2019[,2]))^2)))
```

```
[1] 3732.173 2593.207
```

```
# Distance standard
sqrt(mean((xy.2019[,1] - mean(xy.2019[,1]))**2 +
            (xy.2019[,2] - mean(xy.2019[,2]))**2))
```

```
[1] 4544.649
```

```
## Calcul pour toutes les années et enregistrement des centres moyens dans de nouveaux champs
CentreMoyen$DS.X <- c(sqrt(mean((xy.2019[,1] - mean(xy.2019[,1]))^2)),
                      sqrt(mean((xy.2020[,1] - mean(xy.2020[,1]))^2)),
                      sqrt(mean((xy.2021[,1] - mean(xy.2021[,1]))^2)),
                      sqrt(mean((xy.2022[,1] - mean(xy.2022[,1]))^2)))

CentreMoyen$DS.Y <- c(sqrt(mean((xy.2019[,2] - mean(xy.2019[,2]))^2)),
                       sqrt(mean((xy.2020[,2] - mean(xy.2020[,2]))^2)),
                       sqrt(mean((xy.2021[,2] - mean(xy.2021[,2]))^2)),
                       sqrt(mean((xy.2022[,2] - mean(xy.2022[,2]))^2)))

CentreMoyen$DS <- c(sqrt(mean((xy.2019[,1] - mean(xy.2019[,1]))**2 +
                              (xy.2019[,2] - mean(xy.2019[,2]))**2)),
                    sqrt(mean((xy.2020[,1] - mean(xy.2020[,1]))**2 +
                              (xy.2020[,2] - mean(xy.2020[,2]))**2)),+
                    sqrt(mean((xy.2021[,1] - mean(xy.2021[,1]))**2 +
                              (xy.2021[,2] - mean(xy.2021[,2]))**2)),+
                    sqrt(mean((xy.2022[,1] - mean(xy.2022[,1]))**2 +
                              (xy.2022[,2] - mean(xy.2022[,2]))**2)))
## Visualisation des résultats
head(CentreMoyen)
```

```
Simple feature collection with 4 features and 6 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 649696 ymin: 157059.3 xmax: 650663.7 ymax: 157544.8
Projected CRS: NAD83 / MTQ Lambert
  Annee      CMx      CMy                  geometry     DS.X     DS.Y       DS
1  2019 649990.3 157059.3 POINT (649990.3 157059.3) 3732.173 2593.207 4544.649
2  2020 649696.0 157544.8   POINT (649696 157544.8) 4134.913 2393.048 4777.466
3  2021 650663.7 157214.8 POINT (650663.7 157214.8) 3374.821 2294.171 4080.764
4  2022 650442.5 157237.0   POINT (650442.5 157237) 3601.208 2170.215 4204.585
```

Représentations graphiques de la dispersion

Une fois que la couche `sf` des centres moyens avec les trois champs pour la distance standard est créée, il suffit de tracer un rectangle et un cercle de distance standard.

```
## Enveloppe convexe
sf.Enveloppes <- st_sf(data.frame(Id=c("2019", "2020", "2021", "2022")),
                    geometry = c(st_convex_hull(st_union(M2019)),
                                 st_convex_hull(st_union(M2020)),
                                 st_convex_hull(st_union(M2021)),
                                 st_convex_hull(st_union(M2022))))
## Rectangle avec les distances standards sur les coordonnées X et Y
#' Fonction pour tracer le rectangle
#' @param MoyX coordonnées X du centre moyen.
#' @param MoyY coordonnées Y du centre moyen.
#' @param SDx distance standard sur les coordonnées X.
#' @param SDy distance standard sur les coordonnées Y.
#' @param crs projection cartographique.
CreationRec <- function(MoyX, MoyY, SDx, SDy, ProjCarto){
  pt1 = c(MoyX - SDx, MoyY - SDy)
  pt2 = c(MoyX - SDx, MoyY + SDy)
  pt3 = c(MoyX + SDx, MoyY + SDy)
  pt4 = c(MoyX + SDx, MoyY - SDy)
  Rectangle = st_polygon(list(rbind(pt1, pt2, pt3, pt4, pt1)))
  Rectangle = st_sfc(Rectangle)
  st_crs(Rectangle) = ProjCarto
  return(Rectangle)
}
MoyX <- CentreMoyen$CMx
MoyY <- CentreMoyen$CMy
SDx <- CentreMoyen$DS.X
SDy <- CentreMoyen$DS.Y
sf.Rectangles <- st_sf(data.frame(Id=c("2019", "2020", "2021", "2022")),
                    geometry = c(CreationRec(MoyX[1], MoyY[1], SDx[1], SDy[1], ProjCarto),
                                 CreationRec(MoyX[2], MoyY[2], SDx[2], SDy[2], ProjCarto),
                                 CreationRec(MoyX[3], MoyY[3], SDx[3], SDy[3], ProjCarto),
                                 CreationRec(MoyX[4], MoyY[4], SDx[4], SDy[4], ProjCarto)))
## Cercle de distance standard avec la fonction st_buffer
sf.CercleDS <- st_buffer(CentreMoyen, dist = CentreMoyen$DS)
```

Le calcul de l'ellipse est un peu plus complexe. Par conséquent, nous avons écrit deux fonctions :

- **`CreateEllipse`** qui construit une ellipse à partir d'une couche **`sf`** de points.
- **`CreateEllipse_gp`** qui construit des ellipses à partir d'une couche **`sf`** de points en fonction d'une colonne indiquant différents groupes de points (ici, les différentes années).

```
## Appel des deux fonctions dans le fichier ellipses.R
source("code_complementaire/ellipses.R")
## Création d'une ellipse pour une année
sf.Ellipse2019 <- CreateEllipse(M2019)
## Création de plusieurs ellipses regroupées selon les différentes années
```

```
sf.Ellipse <- CreateEllipse_gp(points = Mefaits, group = "ANNEE")
head(sf.Ellipse)
```

```
Simple feature collection with 4 features and 12 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 643833.8 ymin: 153377.3 xmax: 655559 ymax: 160937.9
Projected CRS: NAD83 / MTQ Lambert
       CMx      CMy   Sigmax   Sigmay       Lx        Ly     Aire    Theta
1 649990.3 157059.3 3055.157 5683.790 6110.315 11367.580 54553357 64.60944
2 649696.0 157544.8 3154.712 5994.647 6309.423 11989.294 59411858 75.81068
3 650663.7 157214.8 3128.891 4869.299 6257.783  9738.598 47863757 76.14675
4 650442.5 157237.0 2540.594 5406.184 5081.187 10812.368 43149511 68.52718
  ThetaCorr  Major  Minor                       geometry ANNEE
1  64.60944 SigmaY SigmaX POLYGON ((655125.1 159496.4...  2019
2  75.81068 SigmaY SigmaX POLYGON ((655507.7 159014.3...  2020
3  76.14675 SigmaY SigmaX POLYGON ((655391.4 158380.7...  2021
4  68.52718 SigmaY SigmaX POLYGON ((655473.5 159216, ...  2022
```

Les quatre représentations de la dispersion sont présentées à la figure 3.7 : 1) enveloppe convexe (gris), 2) cercle de distance standard (bleu), 3) rectangle de distance standard sur les X et Y (noir) et 4) ellipse de distance standard (rouge).

3.2.3.2 Calcul de mesures pondérées

Pour illustrer le calcul des mesures pondérées, nous utilisons des données sur les effectifs des premier et dernier déciles de revenu après impôt des familles économiques pour les secteurs de recensement de la ville de Sherbrooke en 2021 (figure 3.8).

Le code suivant permet d'importer et de structurer les données, puis de calculer les différentes mesures (centre moyen pondéré et distance standard pondérée).

```
## Importation et structuration des données
dfdecile <- read.csv("data/chap03/DataDecilesSR.csv", header = TRUE, sep = ",")
dfdecile$SRIDU <- substr(dfdecile$SRIDU, 1, 10)
SR <- st_read(dsn = "data/chap03/Recen2021Sherbrooke.gpkg",
              layer = "DR_SherbSRDonnees2021", quiet=TRUE)
SR <- merge(SR[,c("SRIDU")], dfdecile, by = "SRIDU")
ProjCarto <- st_crs(SR)
## Coordonnées géographiques des secteurs de recensement
xy <- st_coordinates(st_point_on_surface(SR))
## Pondérations pour les deux déciles
wd1  <- SR$D1
wd10 <- SR$D10
## Sommes des pondérations
swd1  <- sum(wd1)
```

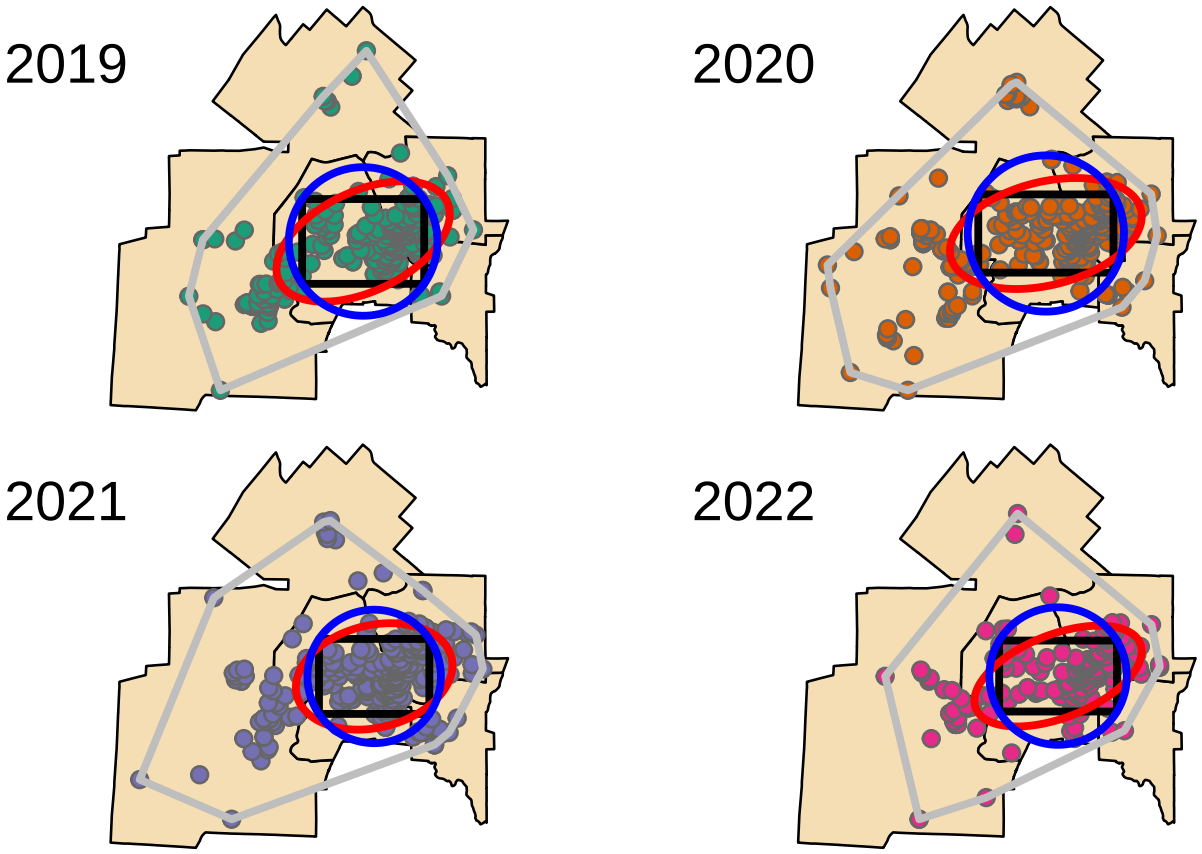


Figure 3.7 – Représentations de la dispersion des méfaits pour les quatre années

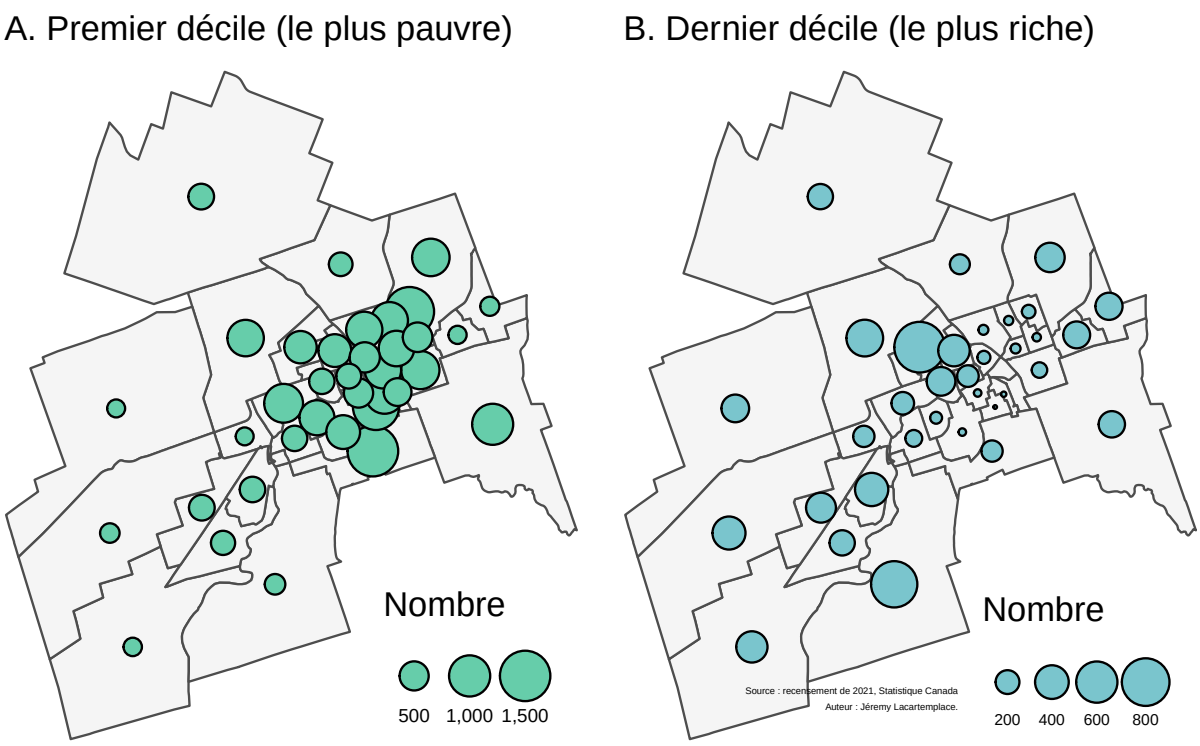


FIGURE 3.8 – Déciles extrêmes de revenu après impôt des familles économiques

```
swd10 <- sum(wd10)
## Calcul du centre pondéré
XmoyD1  <- sum(xy[,1]*wd1) / swd1
XmoyD10 <- sum(xy[,1]*wd10) / swd10
YmoyD1  <- sum(xy[,2]*wd1) / swd1
YmoyD10 <- sum(xy[,2]*wd10) / swd10
CentreMoyenPond <- data.frame(Decile = c("Premier", "Dernier"),
                          X = c(XmoyD1, XmoyD10),
                          Y = c(YmoyD1, YmoyD10),
                          CMwx = c(XmoyD1, XmoyD10),
                          CMwy = c(YmoyD1, YmoyD10))
CentreMoyenPond <- st_as_sf(CentreMoyenPond, coords = c("X", "Y"), crs = ProjCarto)
## Calcul de la distance standard pondérée
sdD1  <- sqrt((sum(wd1*(xy[,1]- XmoyD1)^2) / swd1) + (sum(wd1*(xy[,2]- YmoyD1)^2) / swd1))
sdD10 <- sqrt((sum(wd10*(xy[,1]- XmoyD10)^2) / swd1) + (sum(wd10*(xy[,2]- YmoyD10)^2) / swd10))
## Zones tampons avec la distance standard pondérée
CentreMoyenPond$SDw <- c(sdD1, sdD10)
sf.CercleDSW <- st_buffer(CentreMoyenPond, dist = CentreMoyenPond$SDw)

## Calcul de l'ellipse pondérée pour le premier décile
SR.points <- st_point_on_surface(SR)
ellipse.D1 <- CreateEllipse(SR.points, w = SR.points$D1)
ellipse.D10 <- CreateEllipse(SR.points, w = SR.points$D10)
```

```
# Carte 1 : premier décile
Carte1 = tm_shape(SR)+
              tm_polygons(col="whitesmoke", border.col = "grey30", lwd = 1)+
          tm_shape(CentreMoyenPond[1,])+
              tm_dots(size = .5, col="black")+
          tm_shape(sf.CercleDSW[1,])+
              tm_borders(col="blue", lwd = 2)+
          tm_shape(ellipse.D1)+
              tm_borders(col="red", lwd = 2)+
  tm_layout(main.title = "A. Premier décile (le plus pauvre)",
            main.title.size = .9, frame = FALSE)
# Carte 2 : dernier décile
Carte2 = tm_shape(SR)+
              tm_polygons(col="whitesmoke", border.col = "grey30", lwd = 1)+
          tm_shape(CentreMoyenPond[2,])+
              tm_dots(size = .5, col="black")+
          tm_shape(sf.CercleDSW[2,])+
              tm_borders(col="red", lwd = 2)+
          tm_shape(ellipse.D10)+
              tm_borders(col="blue", lwd = 2)+
 tm_layout(main.title = "B. Dernier décile (le plus riche)",
            main.title.size = .9, frame = FALSE)+
tm_credits("Source : recensement de 2021, Statistique Canada\nAuteur : Jéremy Lécartemplace.",
             position = c("right", "bottom"), size = 0.7, align = "right")
# Combinaison des deux cartes
tmap_arrange(Carte1, Carte2, ncol = 2, nrow = 1)
```

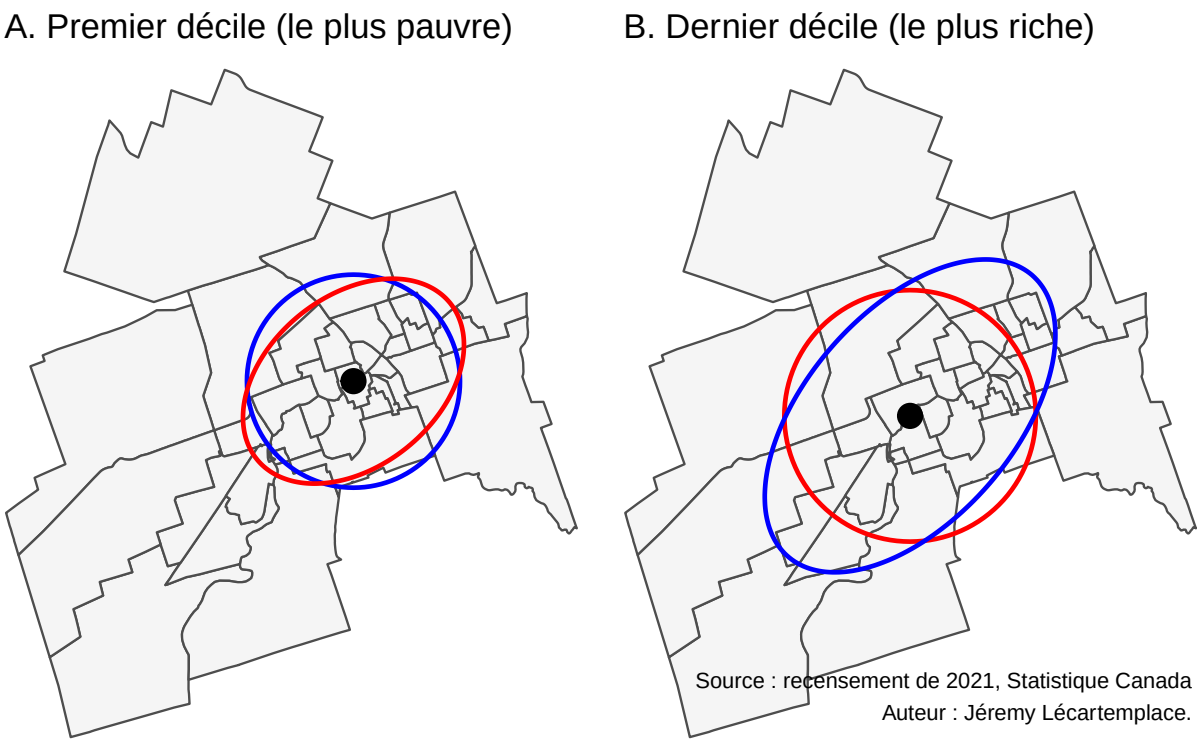


FIGURE 3.9 – Cercles de distance standard et ellipses pondérés

3.3 Forme d'un semis de points

Étudier la forme d'un semis de points, c'est vouloir décrire l'arrangement spatial et l'espacement des points dans une région donnée. Autrement dit, l'objectif est de répondre à la question suivante : comment se répartissent les points dans une région donnée? Nous distinguons habituellement trois types de distribution spatiale d'un semis de points (figure 3.10) :

1. **Distribution dispersée** quand les points du semis sont régulièrement espacés.
2. **Distribution aléatoire** quand la distribution des points n'est nullement guidée par des considérations géographiques. Autrement dit, chaque point du semis a la même probabilité d'être situé dans n'importe quelle partie de la zone d'étude.
3. **Distribution concentrée** quand il existe des regroupements de points dans une ou plusieurs parties de la région d'étude. Par exemple, les musées et les théâtres sont habituellement concentrés dans les parties centrales des métropoles.

⚠ **Attention**

Deux grandes familles pour décrire la forme d'un semis de points

1. Celles basées sur la distance (l'indice du plus proche voisin, fonctions K et L de Ripley) (section 3.3.1).
2. Celles basées sur la densité (méthode des quadrats avec différents tests statistiques) (section 3.3.2).

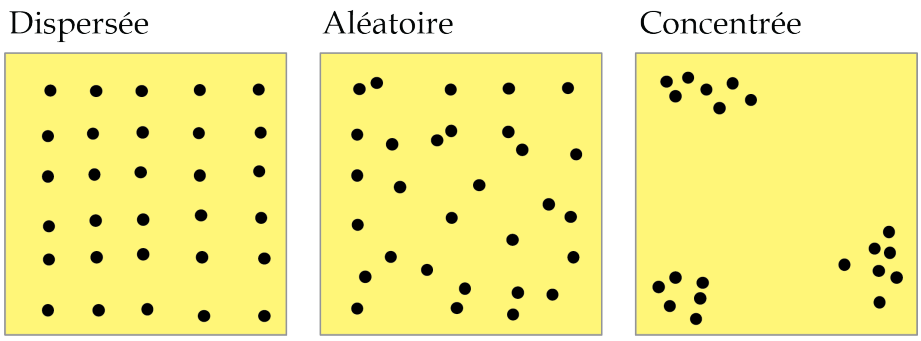


FIGURE 3.10 – Trois types de distribution spatiale d'un semis de points

3.3.1 Méthode du plus proche voisin

Le principe de base de cette méthode est fort simple et se décompose en quatre étapes :

1. Mesurer, pour chaque point du semis, la distance le séparant du point le plus proche, puis calculer la distance moyenne du point le plus proche (équation 3.20).
2. Calculer la moyenne attendue du point le plus proche pour une dispersion aléatoire (équation 3.22).
3. Calculer l'indice du plus proche voisin, soit le ratio entre la distance observée et la distance aléatoire (équation 3.21). L'indice R s'interprète alors comme suit :
   - Si R est égal à 1, la dispersion du semis de points est aléatoire.
   - Si R est inférieur à 1, la distribution du semis de points tend vers la concentration (avec une concentration absolue quand $R = 0$; tous les points ont les mêmes coordonnées géographiques).
   - Si R est supérieur à 1, la distribution du semis de points tend vers la dispersion.
4. Calculer les valeurs de Z et de p pour déterminer si la valeur de R obtenue est significative (équation 3.23).

$$R_o = \frac{\sum_{i=1}^{n} d_i}{n} \tag{3.20}$$

$$R_a = \frac{1}{2\sqrt{(n/S)}} \tag{3.21}$$

$$R = \frac{R_o}{R_a} \tag{3.22}$$

$$Z = \frac{R_o - R_a}{SE}, SE = \frac{0.26136}{\sqrt{(n^2/S)}} \text{ avec :} \tag{3.23}$$

- n, nombre de points.
- d_i, distance séparant le point i de son voisin le plus proche.
- S, superficie de l'espace d'étude.

Le code ci-dessous permet de mettre en œuvre la méthode du plus proche voisin pour les méfaits pour les quatre années.

```
library(spatstat)
library(ggplot2)
## Indice du plus proche voisin : R observé (équation 3.20)
# le paramètre k indique le nombre de plus proches voisins
Robs2019 <- mean(nndist(st_coordinates(M2019), k=1))
Robs2020 <- mean(nndist(st_coordinates(M2020), k=1))
Robs2021 <- mean(nndist(st_coordinates(M2021), k=1))
Robs2022 <- mean(nndist(st_coordinates(M2022), k=1))
## Indice du plus proche voisin : R attendu (distribution aléatoire) (équation 3.21)
# Attention, il faut spéficier S, la superficie de l'espace d'étude
Arrondissements <-  st_read(dsn = "data/chap03/Arrondissements.shp", quiet=TRUE)
Arrondissements <- st_transform(Arrondissements, crs = 3798)
S <- as.numeric(st_area(st_union(Arrondissements)))
# Nombre de points par année
N2019 <- nrow(M2019)
N2020 <- nrow(M2020)
N2021 <- nrow(M2021)
N2022 <- nrow(M2022)
# Calcul de Ra
Ra2019 <- 1 / (2 * sqrt(N2019 / S))
Ra2020 <- 1 / (2 * sqrt(N2020 / S))
Ra2021 <- 1 / (2 * sqrt(N2021 / S))
Ra2022 <- 1 / (2 * sqrt(N2022 / S))
## Calculons le R
# Création d'un DataFrame
IndicePPV <- data.frame(id = c("2019", "2020", "2021", "2022"),
                            points = c(N2019, N2020, N2021, N2022),
                            Superficie = c(S, S, S, S),
                            Robs = c(Robs2019, Robs2020, Robs2021, Robs2022),
                            Rattendu = c(Ra2019, Ra2020, Ra2021, Ra2022))
# Calcul du R (équation 3.22)
IndicePPV$R <- IndicePPV$Robs / IndicePPV$Rattendu
# Calcul du Z (équation 3.23)
IndicePPV$SE <- 0.26136 / sqrt(IndicePPV$points^2 / IndicePPV$Superficie)
IndicePPV$Z <- (IndicePPV$Robs - IndicePPV$Rattendu) / IndicePPV$SE
IndicePPV$P <- round(2*pnorm(q=abs(IndicePPV$Z), lower.tail=FALSE),3)
print(IndicePPV)
```

```
    id points Superficie     Robs Rattendu         R       SE         Z P
1 2019    251  366358191 251.5498 604.0684 0.4164260 19.93051 -17.68739 0
2 2020    383  366358191 183.9866 489.0166 0.3762380 13.06151 -23.35335 0
3 2021    344  366358191 227.1881 515.9929 0.4402931 14.54232 -19.85961 0
4 2022    220  366358191 251.3347 645.2256 0.3895299 22.73890 -17.32234 0
```

Interprétation des résultats

Analysons les différentes colonnes du tableau 3.3 :

- **points (n) :** il y a respectivement 251, 383, 344 et 220 méfaits pour les années 2019 à 2022.
- ***R* observé :** en moyenne, un méfait est distant de 252, 184, 227 et 251 mètres du méfait le plus proche pour les quatre années.
- ***R* attendu :** pour une distribution aléatoire, un méfait devrait être distant du méfait le plus proche de 604, 489, 516, et 645 mètres.
- **Indice du plus proche voisin :** toutes les valeurs sont inférieures à 1, indiquant des distributions spatiales concentrées. La concentration est la plus forte pour l'année 2020 (R = 0,376).
- **valeur de *p* :** toutes les valeurs sont égales à 0, signalant que les résultats sont significatifs.

Tableau 3.3 – Résultats de la méthode du plus proche voisin pour les méfaits par année

Année	points (n)	*R* observé	*R* attendu	Indice plus proche voisin	Erreur standard	Z	*p*
2019	251	252	604	0,416	19,931	-17,687	0
2020	383	184	489	0,376	13,062	-23,353	0
2021	344	227	516	0,440	14,542	-19,860	0
2022	220	251	645	0,390	22,739	-17,322	0

Notez qu'il est possible aussi de construire un graphique pour le *R* observé avec plusieurs voisins, tel que réalisé avec le code ci-dessous avec k = 1 à 50 (figure 3.11).

```
# k = 1 à 50
Robs2019N1_50 <- apply(nndist(st_coordinates(M2019), k=1:50), 2, FUN=mean)
Robs2020N1_50 <- apply(nndist(st_coordinates(M2020), k=1:50), 2, FUN=mean)
Robs2021N1_50 <- apply(nndist(st_coordinates(M2021), k=1:50), 2, FUN=mean)
Robs2022N1_50 <- apply(nndist(st_coordinates(M2022), k=1:50), 2, FUN=mean)
# Enregistrement dans des dataFrames
Robs2019N1_50 <- data.frame(An="2019", Voisins=1:length(Robs2019N1_50), Robs=Robs2019N1_50)
Robs2020N1_50 <- data.frame(An="2020", Voisins=1:length(Robs2020N1_50), Robs=Robs2020N1_50)
Robs2021N1_50 <- data.frame(An="2021", Voisins=1:length(Robs2021N1_50), Robs=Robs2021N1_50)
Robs2022N1_50 <- data.frame(An="2022", Voisins=1:length(Robs2022N1_50), Robs=Robs2022N1_50)
# Combinaison des dataFrames en un seul
RobsN1_50 <- rbind(Robs2019N1_50, Robs2020N1_50, Robs2021N1_50, Robs2022N1_50)
# Création du graphique
ggplot(RobsN1_50)+
 geom_point(aes(x = Voisins, y = Robs, color = An))+
 geom_line(aes(x = Voisins, y = Robs, color = An))+
labs(x = "Nombre de voisins",
     y = "Robs - Distance en mètres",
     color = "Année")
```

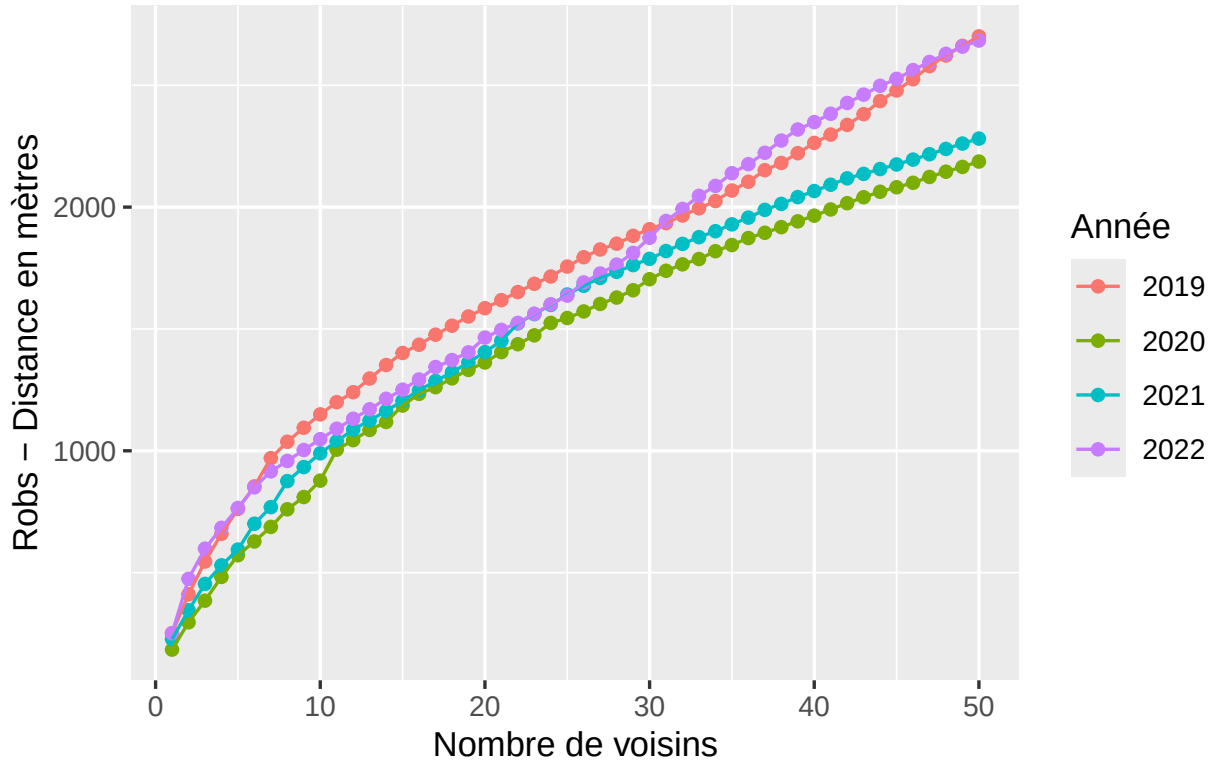


FIGURE 3.11 – Distance au plus proche voisin de 1 à 50

3.3.2 Méthode des quadrats

3.3.2.1 Principe de base

Le principe de base de la méthode des quadrats peut être décomposé en trois étapes :

1. Superposer à la région d'étude comprenant le semis de points un ensemble de **quadrats** (habituellement une grille régulière formée d'un ensemble de carrés).
2. Compter le nombre de points compris dans chacun des quadrats. De la sorte, certains quadrats ne comprennent aucun point tandis que d'autres en contiennent un, deux, trois, etc. Nous obtenons ainsi un tableau des fréquences.
3. Réaliser des tests statistiques à partir des fréquences observées et théoriques pour qualifier la distribution du semis de points (test de Kolmogorov-Smirnov, test du khi-deux ou méthode Monte-Carlo).

3.3.2.2 Forme, distribution et taille des quadrats

Il est possible de paramétrer les quadrats selon leur forme, leur distribution et leur taille. Habituellement, la forme retenue pour les quadrats est le carré, mais d'autres formes géométriques peuvent être utilisées comme l'hexagone et plus rarement, le cercle. La distribution des quadrats peut aussi être soit régulière, soit aléatoire (figure 3.12). Notez que dans le cas d'un cercle, le maillage ne peut être qu'irrégulier puisque certains points risqueraient de ne pas être contenus dans un cercle pour un maillage régulier.

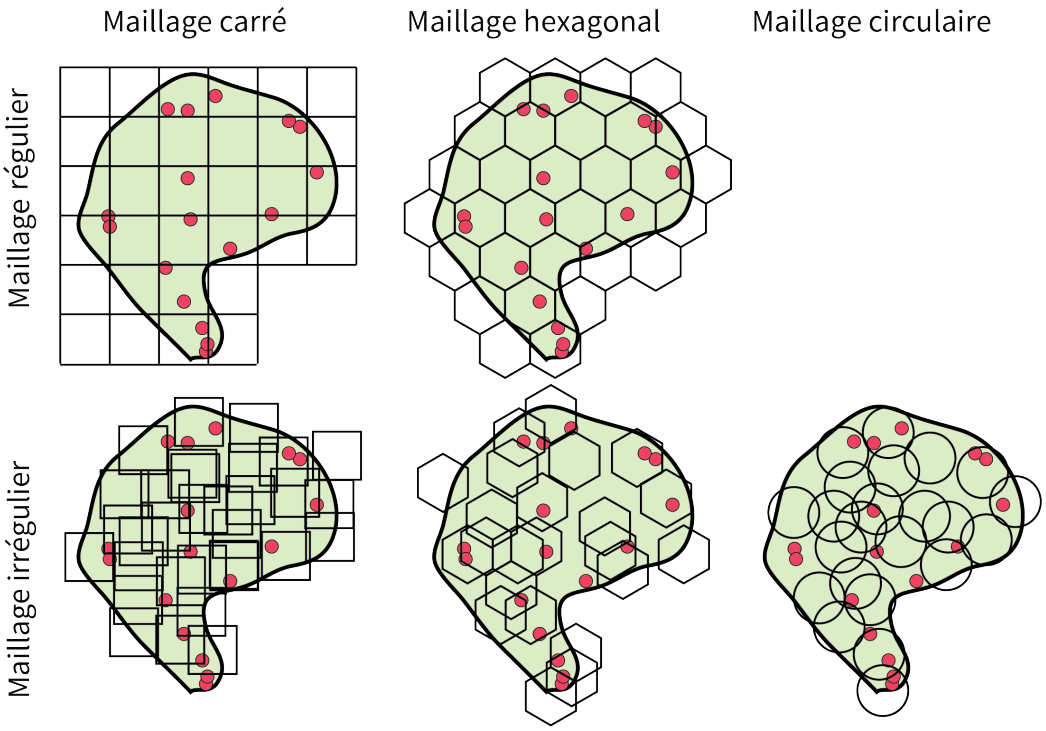


FIGURE 3.12 – Formes et distributions de quadrats

Bien entendu, les résultats varient selon la taille des quadrats. Par exemple, dans le cas d'une distribution spatiale concentrée d'un semis de points, diminuer la taille des quadrats risque d'augmenter la perception de la dispersion. Certains auteurs proposent alors une formule pour déterminer la superficie optimale du quadrat (Wong et Lee 2005; Mitchel 2005) :

$$S_q = \frac{2S}{n} \tag{3.24}$$

$$l_q = \sqrt{S_q} \text{ et } r_q = \sqrt{\frac{S_q}{\pi}} \text{ et } l_a = \sqrt{\frac{2S_q}{3\sqrt{3}}} \text{ avec :} \tag{3.25}$$

- S_q, superficie du quadrat.
- n, nombre de points dans le semis.
- l_q, longueur du côté si la forme du quadrat est un carré.
- r_q, longueur du rayon si la forme du quadrat est un cercle.
- l_a, longueur du côté d'un hexagone régulier.

3.3.2.3 Tests statistiques

Construction du tableau de fréquences observées et théoriques

Une fois les quadrats créés, nous devons compter le nombre de points compris dans chacun d'eux. Une distribution spatiale concentrée à l'extrême se traduit par la localisation de tous les points du semis d'un seul quadrat, tandis que pour une distribution dispersée maximale se traduit par le fait que tous les quadrats contiennent le même nombre de points. Par la suite, nous construisons un tableau de fréquences.

Prenons deux situations à la figure 3.13 :

1. **A. Une distribution dispersée**, puisque les points sont présents dans la plupart des quadrats.
2. **B. Une distribution concentrée**, puisque les points sont localisés dans quelques quadrats.

Notez que pour les deux situations, nous avons 42 points (n) et 36 quadrats (k), soit une moyenne de 1,167 point par quadrat ($\lambda = n/k = 42/36 = 1,167$). Détaillons les différentes colonnes du tableau de fréquences observées et théoriques :

- **Fréquences observées (f_o)** : pour la situation A, nous avons 16 quadrats qui ne comprennent aucun point, 4 quadrats avec 1 point, 10 quadrats avec 2 points et finalement 6 quadrats avec 3 points. À l'inverse, pour la situation B, 27 quadrats sur les 36 ne comprennent aucun point, suggérant ainsi une concentration plus forte!
- **Proportions observées :** simplement les fréquences observées divisées par le nombre total de quadrats (par exemple, $16/36 = 0,444$ pour A).
- **Proportions théoriques :** à partir de la loi de probabilité de Poisson (équation 3.26), il est possible de calculer les proportions théoriques que nous devrions avoir si les points étaient distribués aléatoirement. Pas de panique avec la lecture de la formule, nous verrons qu'il existe une fonction pour la calculer facilement dans R. Nous calculons aussi les **proportions théoriques cumulées**.
- **Fréquences théoriques :** les fréquences théoriques sont simplement les proportions théoriques multipliées par le nombre de quadrats (par exemple, 0,311 × 36 = 11,196).

$$p(x = k) = \frac{\lambda^k e^{-\lambda}}{x!} \text{ avec :} \tag{3.26}$$

λ (lambda) $= n/k$, soit le nombre moyen de points (n) par quadrat (k); x, le nombre de points dans le quadrat (0, 1, 2, etc.); $!x$, la factorielle d'un nombre (par exemple, $!3 = 1 \times 2 \times 3 = 6$); e, la constante de l'Euler, soit $exp(1) = 2,718282$.

À partir de ce tableau des fréquences observées et théoriques, nous pouvons calculer les tests de Kolmogorov-Smirnov et du khi-deux.

Test statistique de Kolmogorov-Smirnov

Ce test se décompose en six étapes :

1. Formuler l'hypothèse nulle stipulant que les fréquences observées et théoriques ne sont pas statistiquement différentes (H_0).
2. Choisir un seuil de signification pour valider ou réfuter l'hypothèse nulle (par exemple, $\alpha = 0,05$).
3. Calculer la différence absolue entre les proportions cumulées observées et théoriques.
4. Calculer la statistique D, soit la plus forte valeur des différences absolues entre les fréquences cumulées observées et théoriques (équation 3.27).
5. Calculer la valeur critique pour une distribution aléatoire avec un seuil de signification α (équation 3.28).
6. Comparer les valeurs de D et de $D_{\alpha=0.05}$:
   - Si $D = D_{\alpha = 0,05}$, la distribution est aléatoire.
   - Si $D < D_{\alpha = 0,05}$, la distribution est dispersée.

(A) Distribution dispersée

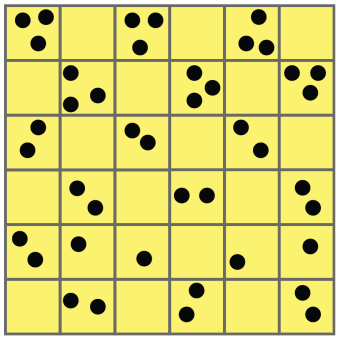


Tableau de fréquences observées et théoriques pour A

N points	Fréquences observées Fréq. (*fo*)	Proportion	Proportion cumulée	Fréquences théoriques Proportion théorique p(x)	Proportion théorique cumulée	Fréq. théor. (*ft*)	Différence absolue entre les proportions théoriques et observées	Test du khi-deux Khi² = (fo – ft) / ft
0	16	0,444	0,444	0,311	0,311	11,196	**0,133**	791,462
1	4	0,111	0,556	0,363	0,675	13,068	0,119	36,440
2	10	0,278	0,833	0,212	0,887	7,632	0,053	451,910
3	6	0,167	1,000	0,082	0,969	2,952	0,031	427,106
4	0	0,000	1,000	0,024	0,993	0,864	0,007	0,024
5	0	0,000	1,000	0,006	0,999	0,216	0,001	0,006

Nombre de points (n) = 42
Nombre de quadrats (k) = 36
($\lambda = n / k$) = 1,167

Test de Kolmogorov-Smirnov D = 0,133
Distibution aléatoire, $D_{\alpha=0,05}$ = 0,210
D < $D_{\alpha=0,05}$ alors distribution dispersée

1706,948

Test du khi-deux (χ^2)
Khi-deux : χ^2 = 1706,948
degrés de liberté : dl = k - 1 = 36 - 1 = 35
Khi-deux théorique avec α=0,001 et dl :
$\chi^2_{(0,001,\ 35)}$ = 66.6.
Le khi-deux est supérieur au khi-deux théorique, la distribution n'est pas aléatoire.

(B) Distribution concentrée

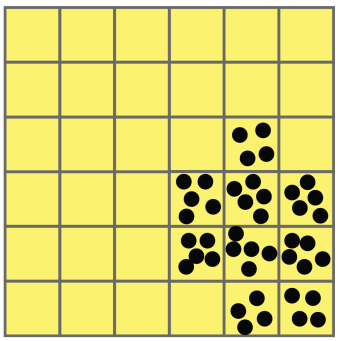


Tableau de fréquences observées et théoriques pour B

N points	Fréquences observées Fréq. (*fo*)	Proportion	Proportion cumulée	Fréquences théoriques Proportion théorique p(x)	Proportion théorique cumulée	Fréq. théor. (*ft*)	Différence absolue entre les proportions théoriques et observées	Test du khi-deux Khi² = (fo – ft) / ft
0	27	0,750	0,750	0,311	0,311	11,196	**0,439**	2290,362
1	0	0,000	0,750	0,363	0,675	13,068	0,075	0,363
2	0	0,000	0,750	0,212	0,887	7,632	0,137	0,212
3	0	0,000	0,750	0,082	0,969	2,952	0,219	0,082
4	3	0,083	0,833	0,024	0,993	0,864	0,160	369,024
5	6	0,167	1,000	0,006	0,999	0,216	0,001	5988,006

Nombre de points (n) = 42
Nombre de quadrats (k) = 36
($\lambda = n / k$) = 1,167

Test de Kolmogorov-Smirnov D = 0,439
Distibution aléatoire, $D_{\alpha=0,05}$ = 0,210
D > $D_{\alpha=0,05}$ alors distribution concentrée

8648,049

Test du khi-deux (χ^2)
Khi-deux : χ^2 = 1706,948
degrés de liberté : dl = k - 1 = 36 - 1 = 35
Khi-deux théorique avec α=0,001 et dl :
$\chi^2_{(0,001,\ 35)}$ = 66.6.
Le khi-deux est supérieur au khi-deux théorique, la distribution n'est pas aléatoire.

FIGURE 3.13 – Illustrations des tests statistiques sur les quadrats

— Si $D > D_{\alpha = 0,05}$, la distribution est concentrée. Plus la valeur de D est élevée, plus la distribution spatiale du semis de points est concentrée.

$$D = \max|poi_{cumul} - pti_{cumul}| \tag{3.27}$$

$$D_{\alpha=0,05} = \frac{1,36}{\sqrt{m}} \text{ avec} \tag{3.28}$$

m étant le nombre total de quadrats; poi_{cumul} et pti_{cumul}, les proportions cumulées observées et théoriques.

Appliquons cette démarche du test de Kolmogorov-Smirnov aux deux distributions de la figure 3.13 :

— $D_{\alpha=0,05} = \frac{1,36}{\sqrt{36}} = 0,210$
— pour la situation A, $D = 0,133$, donc $D < D_{\alpha=0,05}$, alors la distribution est significativement dispersée.
— pour la situation B, $D = 0,439$, donc $D > D_{\alpha=0,05}$, alors la distribution est significativement concentrée.

Test statistique du khi-deux

Ce test se décompose en quatre étapes :

1. Formuler l'hypothèse nulle stipulant que la distribution des fréquences observées dans les quadrats suit une distribution de Poisson (H_0).
2. Calculer le khi-deux (équation 3.29).
3. Comparer la valeur du khi-deux obtenue avec celle du khi-deux théorique ($\chi^2_{\alpha,dl}$) avec $k-1$ degrés de liberté (dl) et un seuil de signification α (0,05 par exemple).
4. Si $\chi^2 > \chi^2_{\alpha,dl}$, alors l'hypothèse nulle est rejetée.

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i} \text{ avec} \tag{3.29}$$

O_i et E_i étant respectivement les fréquences observée et attendue pour i (quadrat avec 0 point, 1, 2, etc.).

Pour les deux situations, le khi-deux calculé est supérieur au khi-deux théorique avec un seuil α de 0,001 et 35 degrés de liberté (paramètre `df` dans la fonction `qchisq` pour *degrees of freedom*). Par conséquent, les deux distributions ne sont pas aléatoires.

```
round(qchisq(p=0.95,  df=35, lower.tail = TRUE),3)
```

```
[1] 49.802
```

```
round(qchisq(p=0.99,  df=35, lower.tail = TRUE),3)
```

```
[1] 57.342
```

```r
round(qchisq(p=0.999,  df=35, lower.tail = TRUE),3)
```

```
[1] 66.619
```

> ⚠ **Attention**
>
> **Loi de Poisson : pas de panique!**
> Vous n'êtes pas familier avec la loi de probabilité de Poisson et le test du khi-deux, retenez simplement la démarche générale, nous utilisons des fonctions qui vont vous faciliter la vie dans R!

3.3.2.4 Mise en œuvre dans R

Le code suivant permet de déterminer la superficie optimale des quadrats en fonction du nombre de points (méfaits pour l'année 2020) et de la superficie de l'espace d'étude.

```r
library(spatstat)
## Taille des quadrats
# Nombre de points
npoints <- nrow(M2020)
# Superficie de l'espace d'étude
S <- as.numeric(st_area(st_union(Arrondissements)))
# Superficie du quadrat (équation 3.24)
Sq <- (2*S) / npoints
# Longueur du carré et du côté de l'hexagone régulier (équation 3.25)
lq <- sqrt(Sq)
la <- sqrt((2*Sq) / (3*sqrt(3)))
# Trouver la longueur du côté du carré dans lequel est compris l'hexagone
cellsizeHex <- 2 * sqrt(Sq/((3*sqrt(3)/2))) * sqrt(3)/2
cat("Nombre de points =", npoints,
    "\nSuperficie (éq. 3.24) =", Sq,
    "\nLongueur du côté du carré (éq. 3.25) =", lq,
    "\nLongueur du côté du l'hexagone (éq. 3.25) =", la,
    "\nLongueur du côté du carré dans lequel est compris l'hexagone =", cellsizeHex, "\n")
```

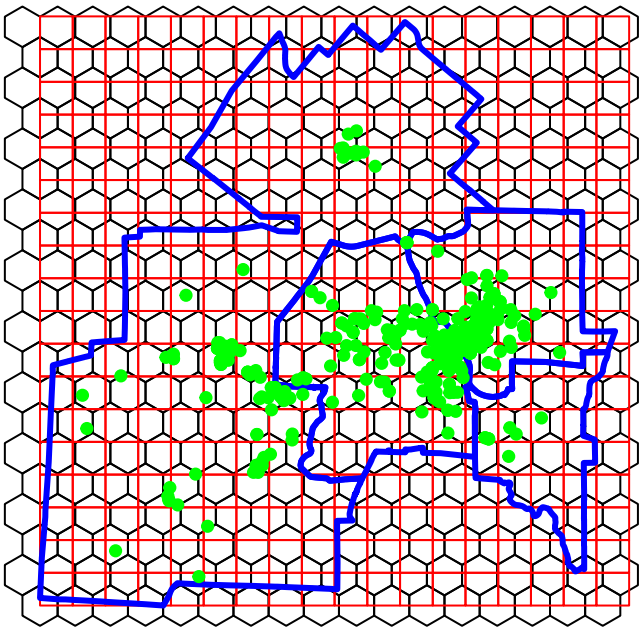
```
Nombre de points = 383
Superficie (éq. 3.24) = 1913098
Longueur du côté du carré (éq. 3.25) = 1383.148
Longueur du côté du l'hexagone (éq. 3.25) = 858.1093
Longueur du côté du carré dans lequel est compris l'hexagone = 1486.289
```

Nous pouvons ensuite créer deux couches avec des quadrats carrés et hexagonaux avec la fonction `st_make_grid` du *package* `sf`. Repérez le paramètre `square` dans la fonction `st_make_grid` : écrivez `square = TRUE` pour obtenir des carrés et `square = FALSE` pour des hexagones réguliers.

```
## Création des quadrats
#Géométrie pour l'espace d'étude
EspaceEtude <- st_geometry(st_union(Arrondissements))
# Création des carrés
Carres.sf <- st_make_grid(EspaceEtude,
                     lq,
                     crs = st_crs(Arrondissements),
                     what  = "polygons",
                     square = TRUE)
# Création des hexagones
Hexagones.sf <- st_make_grid(EspaceEtude,
                     cellsizeHex,
                     crs = st_crs(Arrondissements),
                     what  = "polygons",
                     square = FALSE)
Carres.sf <- st_sf(idCarre = 1:length(lengths(Carres.sf)), Carres.sf)
Hexagones.sf <- st_sf(idHex = 1:length(lengths(Hexagones.sf)), Hexagones.sf)
cat("Superficie (éq. 3.24) =", Sq,
    "\nVérifier la superficie des carrés et des hexagones",
    "\nSuperficie des carrés =", as.numeric(st_area(Carres.sf[1,])),
    "\nSuperficie des hexagones =", as.numeric(st_area(Hexagones.sf[1,])),
    "Les superficies sont bien égales!\n")
```

```
Superficie (éq. 3.24) = 1913098
Vérifier la superficie des carrés et des hexagones
Superficie des carrés = 1913098
Superficie des hexagones = 1913098 Les superficies sont bien égales!
```

```
## Visualisation
tmap_mode("plot")
 tm_shape(Hexagones.sf)+tm_borders(col="black")+
 tm_shape(Carres.sf)+tm_borders(col="red")+
 tm_shape(Arrondissements)+tm_borders(col="blue", lwd=3)+
   tm_shape(M2020)+tm_dots(col="green", size=0.15)+
   tm_layout(frame = FALSE)
```



La figure ci-dessus permet de constater que certains carrés et hexagones n'intersectent pas l'espace d'étude. Par conséquent, nous les supprimons puis calculons le nombre de points par carré et par hexagone.

```
## Suppression des carrés qui n'intersectent pas les quatre arrondissements
RequeteSpatiale <- st_intersects(Carres.sf,
                                 st_union(Arrondissements), sparse = FALSE)
Carres.sf$Intersect <- RequeteSpatiale[, 1]
Carres.sf <- Carres.sf[Carres.sf$Intersect == TRUE, ]
## Suppression des hexagones qui n'intersectent pas les quatre arrondissements
RequeteSpatiale <- st_intersects(Hexagones.sf,
                                 st_union(Arrondissements), sparse = FALSE)
Hexagones.sf$Intersect <- RequeteSpatiale[, 1]
Hexagones.sf <- Hexagones.sf[Hexagones.sf$Intersect == TRUE, ]
## Jointure spatiale : compter le nombre de méfaits de 2020 dans les carrés et les hexagones
Carres.sf$Mefaits2020    = lengths(st_intersects(Carres.sf, M2020))
Hexagones.sf$Mefaits2020 = lengths(st_intersects(Hexagones.sf, M2020))
## Tableau de fréquences
table(Carres.sf$Mefaits2020)
```

```
  0   1   2   3   4   5   6   7   8  10  11  13  21  38  41  64
172  25   6   3   8   1   3   4   3   2   3   1   1   1   1   1
```

```
table(Hexagones.sf$Mefaits2020)
```

```
  0   1   2   3   4   5   6   7   8   9  15  17  19  20  40  48
173  23   5   5   3   5   6   1   3   1   2   1   1   1   1   2
```

Visualisons les résultats à la figure 3.14. Il y a clairement une tendance à la concentration puisque de nombreux quadrats ne contiennent aucun point.

```
## Visualisation
tmap_mode("plot")
Carte1 <-
  tm_shape(subset(Carres.sf, Mefaits2020 == 0))+
     tm_polygons(col="gray90", border.col = "white", lwd = 1)+
   tm_shape(subset(Carres.sf, Mefaits2020!= 0))+
      tm_polygons(col="Mefaits2020", style="cont", title="Nombre",
                  border.col = "white", lwd = 1)+
   tm_shape(Arrondissements)+tm_borders(col="black", lwd=2)+
   tm_layout(frame = FALSE)
Carte2 <-
  tm_shape(subset(Hexagones.sf, Mefaits2020 == 0))+
     tm_polygons(col="gray90", border.col = "white", lwd = 1)+
   tm_shape(subset(Hexagones.sf, Mefaits2020!= 0))+
      tm_polygons(col="Mefaits2020", style="cont", title="Nombre",
                  border.col = "white", lwd = 1)+
   tm_shape(Arrondissements)+tm_borders(col="black", lwd=2)+
   tm_layout(frame = FALSE)
tmap_arrange(Carte1, Carte2)
```

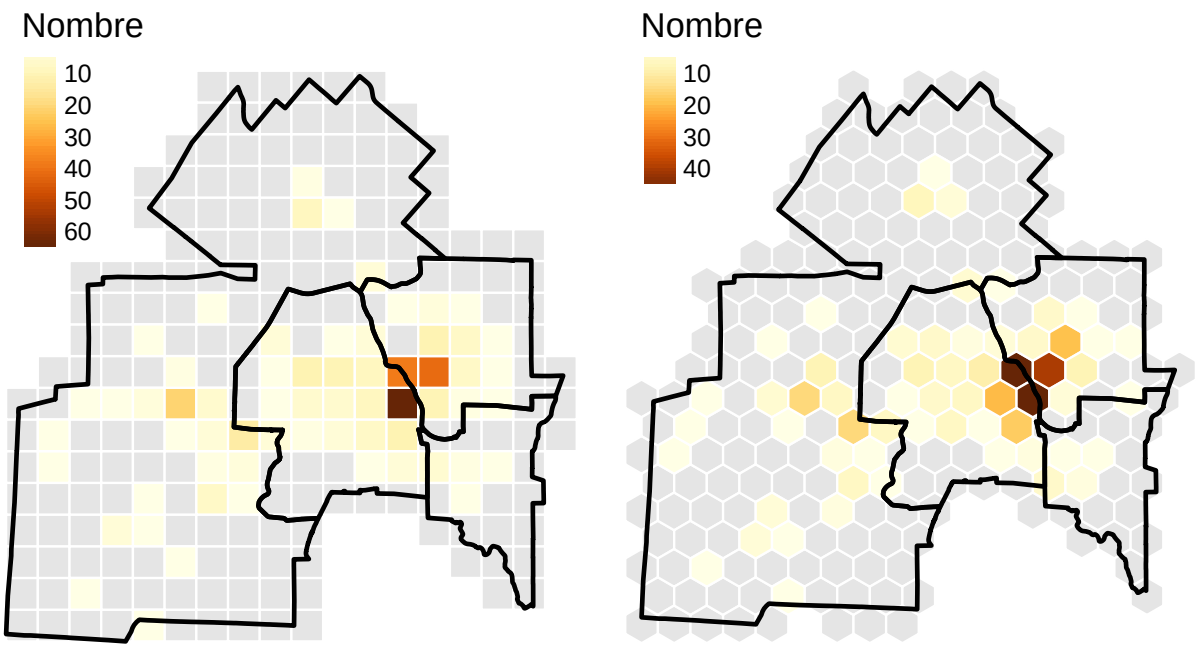


Figure 3.14 – Nombre de méfaits dans les deux géométries de quadrats

Nous pouvons maintenant construire le tableau de fréquences et mettre en œuvre les différents tests statistiques. Le code ci-dessous génère le tableau de fréquences et applique **le test de Kolmogorov-Smirnov**.

```
## Construction du tableau de fréquences
TabFreq <- as.data.frame(table(Carres.sf$Mefaits2020))
names(TabFreq) <- c("Npoints", "Fo")
TabFreq$Npoints <- as.numeric(as.character(TabFreq$Npoints))
# Calcul pour les fréquences observées (fo)
TabFreq$Fo.pro <- TabFreq$Fo / sum(TabFreq$Fo)
TabFreq$Fo.proCum <- cumsum(TabFreq$Fo.pro)
# Calcul pour les fréquences théoriques
npoints <- sum(TabFreq$Npoints*TabFreq$Fo)
nquadrats <- sum(TabFreq$Fo)
Lambda <- npoints / nquadrats
TabFreq$Ft.pro <- dpois(TabFreq$Npoints, lambda = Lambda)
TabFreq$Ft.proCum <- ppois(TabFreq$Npoints, lambda = Lambda, lower.tail = TRUE)
TabFreq$Ft <- TabFreq$Ft.pro * TabFreq$Npoints
# Différences absolues entre les fréquences observées et théoriques cumulées
TabFreq$Difffoft <- abs(TabFreq$Fo.proCum - TabFreq$Ft.proCum)
#calcul de D et Da
D <- max(TabFreq$Difffoft)
Da <- 1.36 / sqrt(nquadrats) # avec p à 0,05
## Diagnostic
if (D>Da){
  cat("D =",round(D,3)," et Da =", round(Da,3),
      "\nD > Da avec p =0,05, alors la distribution tend vers la concentration.")
}else{
  cat("D =",round(D,3)," et Da =", round(Da,3),
      "\nD < Da avec p =0,05, alors la distribution tend vers la dispersion.")
}
```

```
D = 0.536  et Da = 0.089
D > Da avec p =0,05, alors la distribution tend vers la concentration.
```

Le *package* **spatstat** permet de réaliser différents tests avec la fonction **quadrat.test**. Pour ce faire, il faut préalablement convertir les données dans les formats utilisés par ce *package* (**ppp**, **owin** et **tess**).

```
## Conversion des points au format ppp
M2020.ppp <- ppp(x = st_coordinates(M2020)[,1],
                 y = st_coordinates(M2020)[,2],
                 window = as.owin(Arrondissements),
                 check = T)
## Conversion des quadrats carrés ou hexagonaux en objet owin
quadrats <- as(st_geometry(Carres.sf), "Spatial")
FenetresC <- lapply(quadrats@polygons,function(x){
  coords <- x@Polygons[[1]]@coords
  coords<-coords[nrow(coords):1,]
  owin(poly = coords)})
```

```
quadrats <- as(st_geometry(Hexagones.sf), "Spatial")
FenetresH <- lapply(quadrats@polygons,function(x){
  coords <- x@Polygons[[1]]@coords
  coords<-coords[nrow(coords):1,]
  owin(poly = coords)})
## Ces fenêtres sont ensuite converties en un objet tess (tesselation)
TessalationC <- as.tess(FenetresC)
TessalationH <- as.tess(FenetresH)
```

Nous pouvons calculer différents tests avec la fonction `quadrat.test`. Dans un premier temps, nous vérifions si **la distribution est dispersée** avec la méthode Monte-Carlo et 999 permutations (`alternative ="regular"`). Que ce soit avec les quadrats carrés ou hexagonaux, la valeur de p est égale à 1. Cela signifie que nous sommes 100 % certains que la distribution n'est pas dispersée.

```
## Réalisation des différents tests du khi-deux
# test pour une distribution dispersée (Carrés)
cat("Test pour une distribution dispersée avec les quadrats carrés")
```

```
Test pour une distribution dispersée avec les quadrats carrés
```

```
quadrat.test(M2020.ppp, tess = TessalationC,
             method = "MonteCarlo", nsim = 999,
             alternative ="regular")  # dispersée
```

```
    Conditional Monte Carlo test of CSR using quadrat counts
    Test statistic: Pearson X2 statistic

data:  M2020.ppp
X2 = 5212.2, p-value = 1
alternative hypothesis: regular

Quadrats: 235 tiles (irregular windows)
```

```
cat("Test pour une distribution dispersée avec les quadrats hexagonaux")
```

```
Test pour une distribution dispersée avec les quadrats hexagonaux
```

```
quadrat.test(M2020.ppp, tess = TessalationH,
             method = "MonteCarlo", nsim = 999,
             alternative ="regular")  # dispersée
```

```
    Conditional Monte Carlo test of CSR using quadrat counts
    Test statistic: Pearson X2 statistic

data:  M2020.ppp
X2 = 4792.3, p-value = 1
alternative hypothesis: regular

Quadrats: 233 tiles (irregular windows)
```

Dans un second temps, nous vérifions si **la distribution est concentrée**, toujours avec la méthode Monte-Carlo et 999 permutations (`alternative ="clustered"`). Que ce soit avec les quadrats carrés ou hexagonaux, la valeur de p est égale à 0,001. Cela signifie qu'il y a moins de 1 % de chances que la distribution concentrée soit due au hasard.

```
# test pour une distribution concentrée
cat("Test pour une distribution concentrée avec les quadrats carrés")
```

```
Test pour une distribution concentrée avec les quadrats carrés
```

```
quadrat.test(M2020.ppp, tess = TessalationC,
             method = "MonteCarlo", nsim = 999,
             alternative ="clustered") # concentrée
```

```
    Conditional Monte Carlo test of CSR using quadrat counts
    Test statistic: Pearson X2 statistic

data:  M2020.ppp
X2 = 5212.2, p-value = 0.001
alternative hypothesis: clustered

Quadrats: 235 tiles (irregular windows)
```

```
cat("Test pour une distribution concentrée avec les quadrats hexagonaux")
```

```
Test pour une distribution concentrée avec les quadrats hexagonaux
```

```
quadrat.test(M2020.ppp, tess = TessalationH,
             method = "MonteCarlo", nsim = 999,
             alternative ="clustered") # concentrée
```

```
    Conditional Monte Carlo test of CSR using quadrat counts
```

```
    Test statistic: Pearson X2 statistic

data:  M2020.ppp
X2 = 4792.3, p-value = 0.001
alternative hypothesis: clustered

Quadrats: 233 tiles (irregular windows)
```

3.4 Cartographie de la densité des points

Pour repérer les concentrations d'un semis de points dans une région, il faut cartographier la densité des points dans une maille soit irrégulière, soit régulière.

3.4.1 Cartographie de la densité dans une maille irrégulière

Une maille irrégulière est habituellement un découpage administratif comme les municipalités régionales de comté (MRC) du Québec ou les arrondissements d'une ville.

À la figure 3.15, nous avons compté le nombre de méfaits par secteur de recensement pour l'année 2021, puis calculé la densité, soit le nombre de méfaits pour 1000 habitants. Les cercles proportionnels permettent ainsi de repérer les secteurs comprenant le plus de méfaits. Comme la population totale varie d'un SR à l'autre, il est préférable de cartographier la densité, soit le nombre de méfaits pour 1000 habitants. Notez que pour d'autres jeux de données, il serait plus judicieux d'utiliser la superficie comme dénominateur pour calculer la densité (par exemple, le nombre d'arbres au kilomètre carré ou à l'hectare).

En quelques lignes de code, il est très facile de calculer et de cartographier la densité dans un maillage irrégulier.

```
## Secteurs de recensement (SR) de la ville de Sherbrooke en 2021
SR <- st_read(dsn = "data/chap03/Recen2021Sherbrooke.gpkg",
              layer = "DR_SherbSRDonnees2021", quiet=TRUE)
## Sélection des méfaits pour l'année 2021
Incidents <- st_read(dsn = "data/chap03/IncidentsSecuritePublique.shp", quiet=TRUE)
M2021 <- subset(Incidents, DESCRIPTIO == "Méfait" & ANNEE==2021)
## Nous nous assurons que les deux couches ont la même projection cartographique
SR <- st_transform(SR, st_crs(M2021))
## Calcul du nombre d'incidents par SR
SR$Mefaits2021 <- lengths(st_intersects(SR, M2021))
## Calcul du nombre de méfaits pour 1000 habitants
SR$DensiteM21Hab <- SR$Mefaits2021 / (SR$SRpop_2021 / 1000)
## Cartographie
tm_shape(SR)+
  tm_polygons(col="Mefaits2021", style="pretty",
              title="Nombre pour 1000 habitants",
              legend.format = list(text.separator = "à"),
              border.col = "black", lwd = 1)+
```

```
tm_bubbles(size = "DensiteM21Hab", border.col = "black", alpha = .5,
                    col = "aquamarine3", title.size = "Nombre", scale = 1.5)+
tm_layout(frame = FALSE)+tm_scale_bar(text.size = .5, c(0, 5, 10))
```

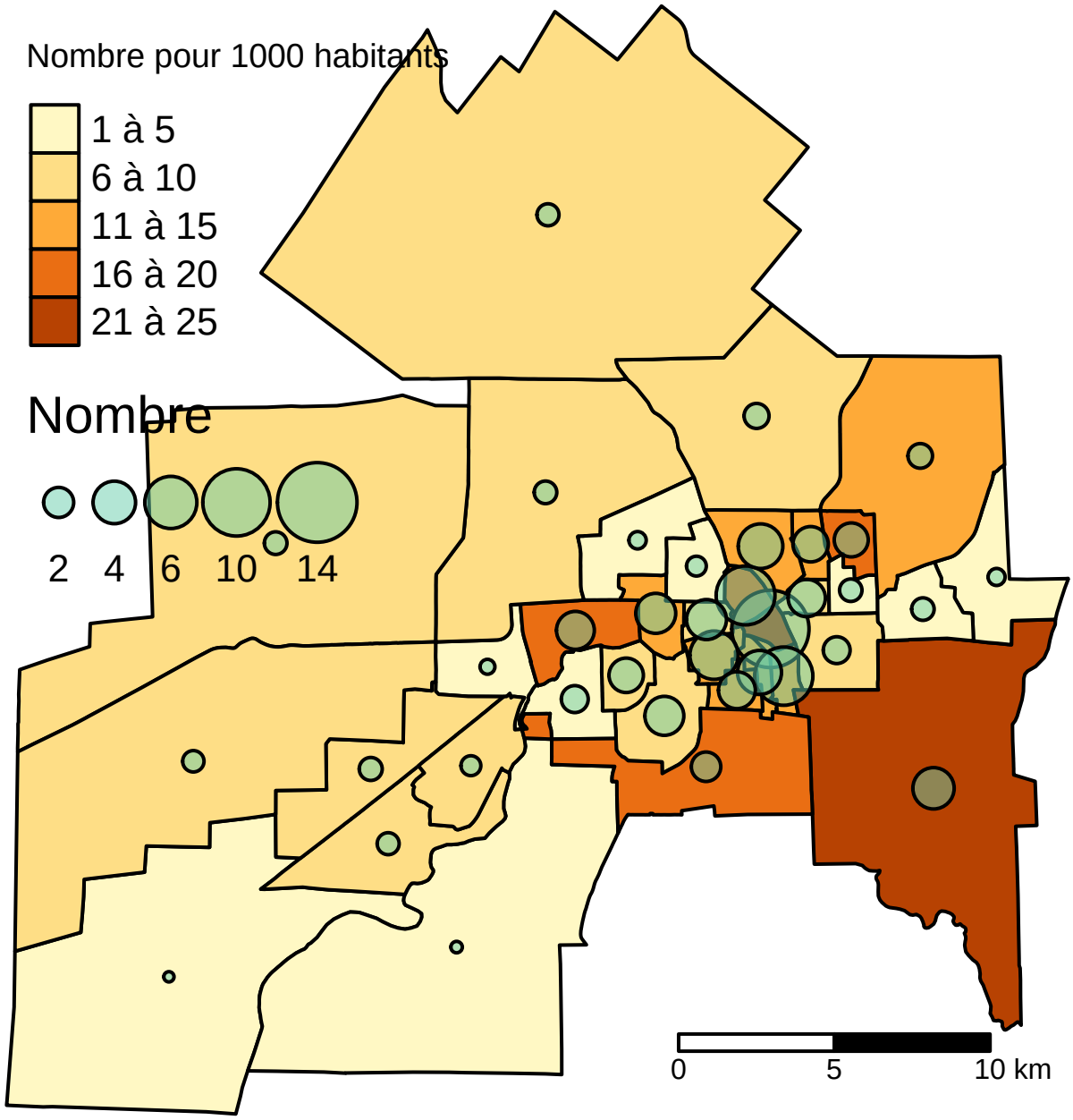


FIGURE 3.15 – Densité des méfaits par secteur de recensement, ville de Sherbrooke, 2021

3.4.2 Cartographie de la densité dans une maille régulière

3.4.2.1 Notion de processus spatial

Lorsque nous analysons des évènements ayant eu lieu dans un espace donné, nous pouvons considérer que ces évènements résultent d'un processus spatial que nous ne pouvons pas observer. Ce processus spatial est caractérisé par des points chauds (endroits où des évènements se produisent souvent) et des points froids (endroits où des évènements se produisent rarement). Si nous collectons suffisamment d'évènements (observations), alors leur densité devrait nous renseigner sur ce processus spatial. Puisqu'une image vaut mille mots, examinons un exemple d'un processus spatial à la figure 3.16.

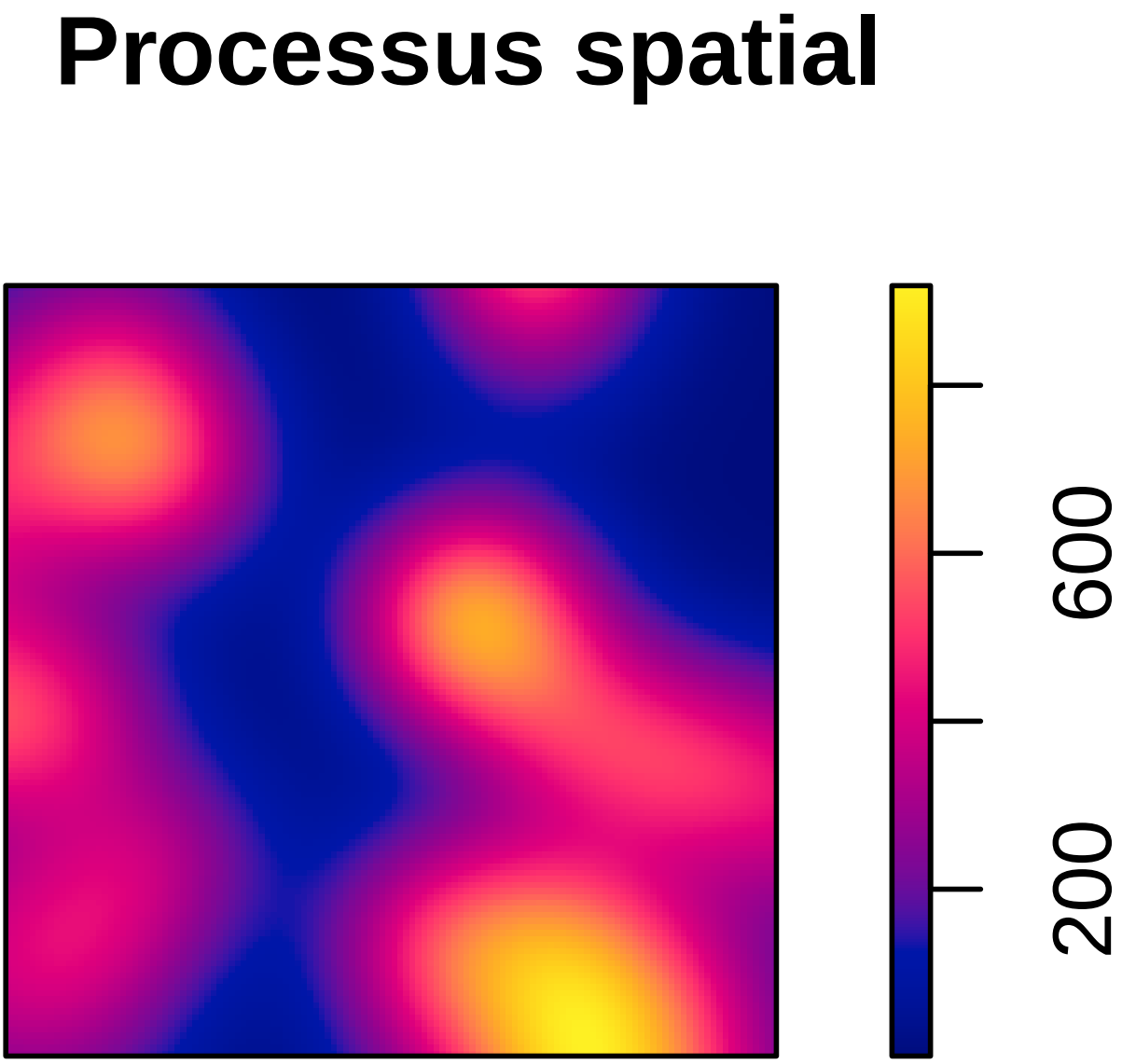


Figure 3.16 – Exemple d'un processus spatial

Imaginons que ce processus spatial représente le niveau de risque de se faire attaquer par un écureuil dans un parc urbain délimité ici par les limites de la figure. Dans la réalité, nous ne pouvons pas mesurer directement ce phénomène, mais nous disposons d'un registre des attaques d'écureuils qui ont été enregistrées et spatialisées (figure 3.17). Bien entendu,

ces données sont fictives! Dans un langage plus technique, il est dit que **les évènements sont des réalisations du processus spatial**.

```
pts <- rpoint(1500, dens1)
plot(pts, main = "Attaques d'écureuils enregistrées")
```

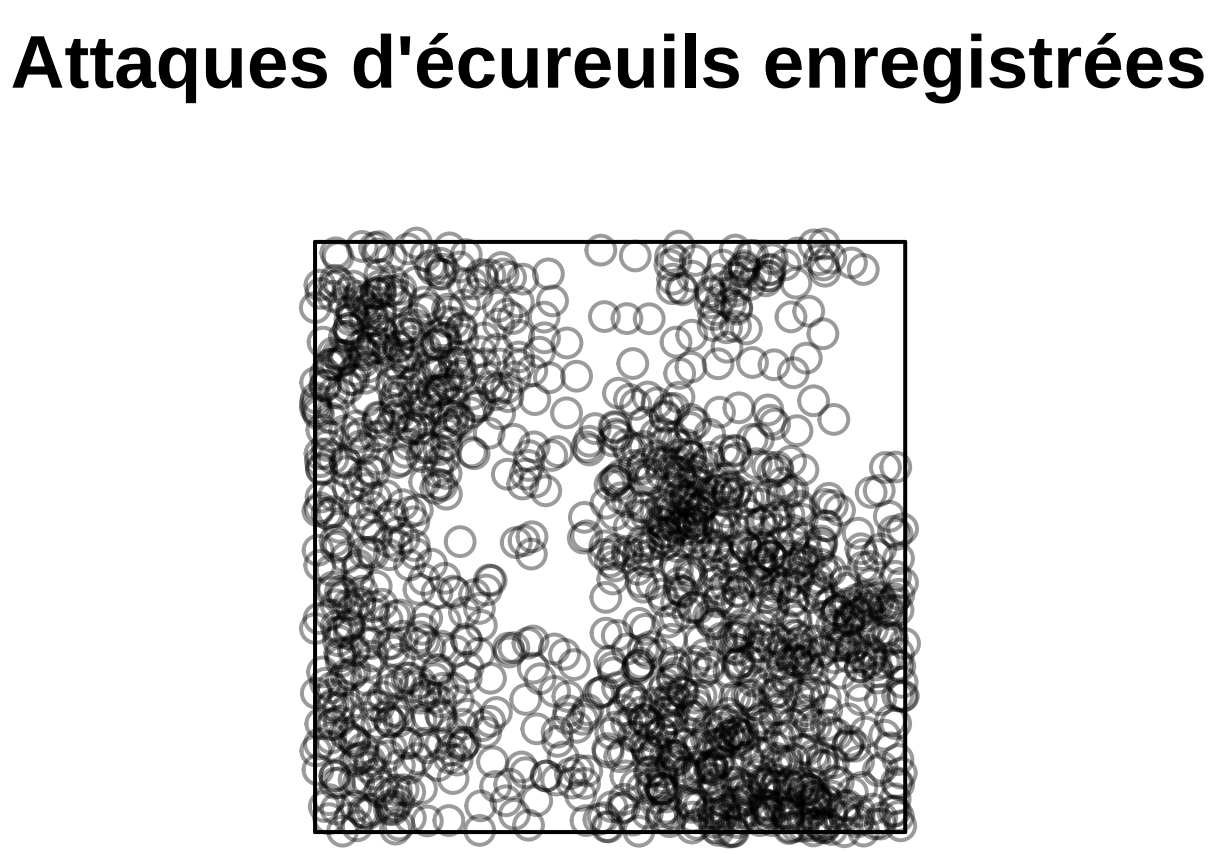


Figure 3.17 – Réalisation du processus spatial

Puisque nous disposons d'un grand nombre d'observations, nous pouvons reconstruire le processus spatial qui produit les évènements en appliquant la méthode d'analyse de densité dans une maille régulière (figure 3.18).

La qualité d'estimation du processus initial dépend ainsi directement du nombre d'évènements dont nous disposons (figure 3.18).

Une fois que nous sommes en mesure de recréer le processus spatial, nous pouvons en apprendre plus sur le phénomène et ses causes. Par exemple, est-ce que les secteurs du parc avec les plus grandes concentrations d'accidents sont aussi ceux dans lesquels sont localisés des jeux pour enfants?

3.4.2.2 Description de la méthode KDE

Le principe de base d'une cartographie de la densité dans une maille régulière – appelée aussi carte de chaleur ou estimation de la densité par noyau (*kernel density estimation* – KDE, en anglais) – est relativement simple et se décompose en trois étapes :

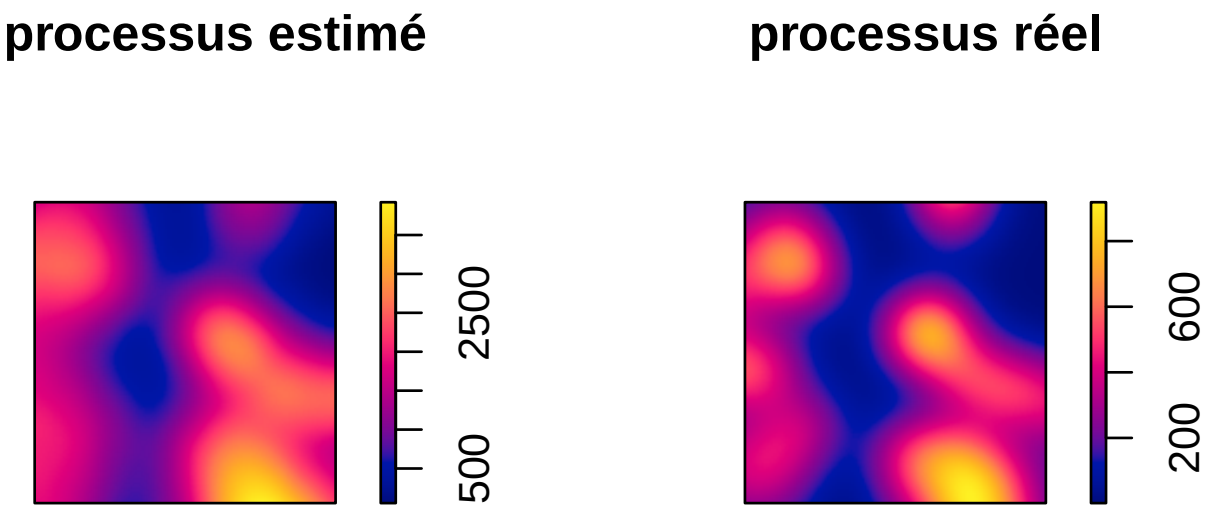


Figure 3.18 – Reconstruction du processus spatial

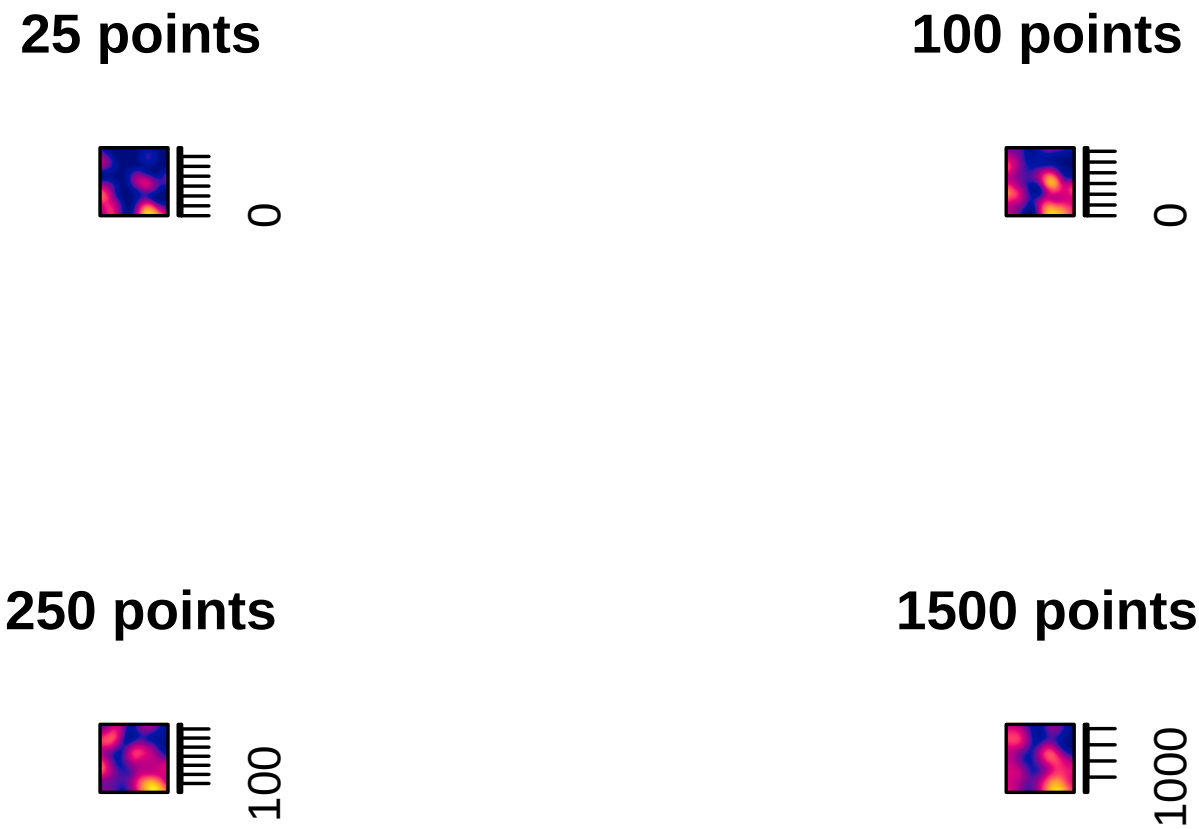


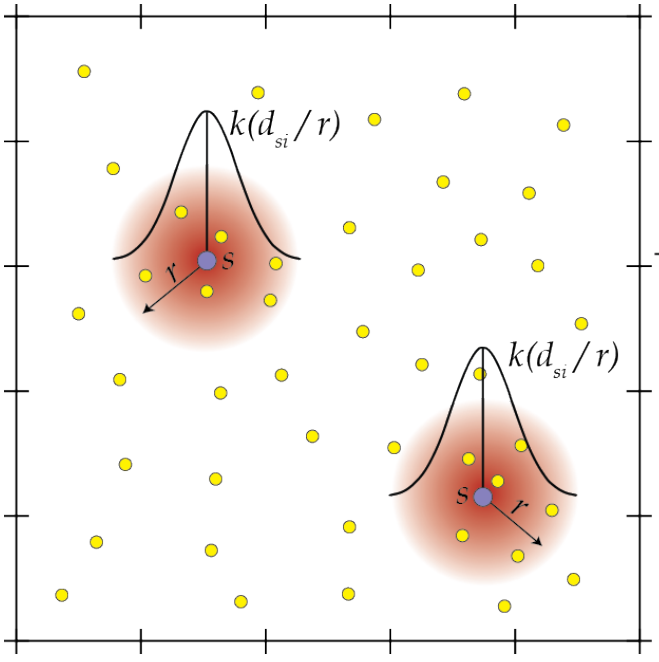
Figure 3.19 – Reconstruction du processus spatial

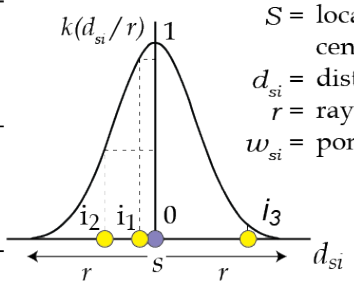
1. Juxtaposer une grille de cellules sur l'espace d'étude, soit les pixels d'une image de *n* mètres de côté (par exemple, 50 mètres).
2. Pour chaque pixel, juxtaposer une zone de recherche de *n* mètres de rayon (1000 mètres par exemple) et calculer le nombre de points présents dans cette zone de recherche.
3. Calculer la densité à partir d'une fonction simple (appelée habituellement fonction uniforme) ou d'une fonction de densité (*kernel*). Pour une fonction simple, nous divisons simplement le nombre de points présents dans la zone de recherche par la superficie de la zone, soit πr^2; cette approche est toutefois peu recommandée puisqu'elle accorde la même pondération à chaque point situé de la zone d'influence! En utilisant une fonction de densité (*kernel*) (équation 3.30), nous accordons une pondération à chacun des points compris dans la zone de recherche : plus le point est proche du centre de la cellule, plus son poids est important dans l'estimation de la densité.

$$\lambda(s) = \sum_{i=1}^{n} \frac{1}{\pi r^2} k\left(\frac{d_{si}}{r}\right) \text{ avec} \tag{3.30}$$

— $\lambda(s)$, estimation de la densité à la localisation s.
— r, rayon de la zone de recherche.
— d_{is}, distance entre la localisation s et le point i.
— k, fonction *kernel* définissant la pondération à accorder au ratio $\frac{d_{si}}{r}$ quand $0 < d_{si} \leq r$. Si $d_{si} > r$, alors $k(d_{si}/r) = 0$.

Il existe différentes fonctions *kernel* pour obtenir les pondérations des points. Telle qu'illustrée à la figure 3.20, l'idée générale est que plus la distance entre la localisation et le point augmente (d_{si}), plus la pondération $k(d_{si}/r)$ de l'équation équation 3.30 est faible.





S = localisation s sur laquelle est centré le noyau
d_{si} = distance entre les points i et j
r = rayon d'influence (1000 mètres)
w_{si} = pondération accordée au point i

En guise d'exemple, avec une valeur de r de 1000 mètres, les pondérations pour les points i sont reportées ci-dessous. Concrètement, plus la distance entre les points i et s augmente, plus la valeur de pondération est faible. Dès que distance est supérieure au rayon d'influence r, alors la pondération est égale à 0.

Point	d_{si} (m.)	w_{si}
i_1	0	1,000
i_2	110	0,988
i_3	500	0,750
i_4	974	0,051
i_5	1001	0,000

FIGURE 3.20 – Principe de la fonction *kernel* pour définir les pondérations des points voisins dans la zone d'influence

Astuce

Les différentes fonctions *kernel* d'estimation de la densité
Pour une description des différentes fonctions *kernel*, consultez le lien suivant. Notez que l'outil *Carte de chaleur (estimation par noyau)* de QGIS inclut plusieurs fonctions *kernel* alors que l'outil *Kernel Density* d'ArcGIS Pro utilise uniquement la fonction quadratique. Finalement, la fonction `density.ppp` du *package* `spatstat` intègre quatre fonctions.
Influence de la forme du noyau (fonction *kernel*)
Excepté la fonction uniforme, la plupart des auteurs s'entendent sur le fait que le choix de fonctions *kernel* a relativement peu d'influence sur le résultat final comparativement au choix du rayon d'influence.

Tableau 3.4 – Fonctions *kernel* disponibles selon différents logiciels

Fonction	QGis	ArcGIS Pro	CrimeStat	R (density.ppp)
Gaussienne			X	X
Triangulaire	X		X	
Quadratique	X	X	X	X
Uniforme	X		X	X
Cubique	X			
Epanechnikov	X			X
Exponentielle			X	

3.4.2.3 Mise en œuvre de la KDE dans R

Pour calculer la densité dans une maille régulière (KDE), vous pouvez utiliser deux *packages* :

1. `SpatialKDE`, un *package* proposé récemment qui reprend le code de l'outil *Carte de chaleur (estimation par noyau)* de QGIS (Caha 2023). Vous obtiendrez donc les mêmes fonctionnalités et résultats qu'avec QGIS. Pour un exemple d'utilisation, consultez cette vignette.
2. `spatstat`, soit le *package* le plus complet pour réaliser des analyses de répartition ponctuelle (Baddeley et Turner 2005; Baddeley, Rubak et Turner 2015). Par conséquent, nous privilégions son utilisation dans les exemples ci-dessous.

Afin d'expliquer comment réaliser une KDE dans R, nous utilisons les accidents survenus dans l'arrondissement des Nations de la ville de Sherbrooke. Notez que nous retenons uniquement cet arrondissement et non l'ensemble de la ville afin d'accélérer les temps de calculs.

```
library(sf)
library(spatstat)
library(tmap)
library(terra)
## Importation des données
Arrondissements <-  st_read(dsn = "data/chap03/Arrondissements.shp", quiet = TRUE)
Arrondissements <-  st_transform(Arrondissements, crs = 3798)
incidents <- st_read(dsn = "data/chap03/IncidentsSecuritePublique.shp", quiet = TRUE)
Incidents <- st_transform(Incidents, crs = 3798)
```

```
## Couche pour les accidents
Accidents <- subset(Incidents, Incidents$DESCRIPTIO %in%
                      c("Accident avec blessés", "Accident mortel"))
## Pour accélérer les calculs, nous retenons uniquement l'arrondissement des Nations
# Couche pour l'arrondissement des Nations
ArrDesNations <- subset(Arrondissements, NOM == "Arrondissement des Nations")
# Sélection des accidents localisés dans l'arrondissement des Nations
RequeteSpatiale <- st_intersects(Accidents, ArrDesNations, sparse = FALSE)
Accidents$Nations <- RequeteSpatiale[, 1]
AccNations <- subset(Accidents, Accidents$Nations == TRUE)
# Une couche par année
AccNations2019 <- subset(AccNations, AccNations$AN == 2019)
AccNations2020 <- subset(AccNations, AccNations$AN == 2020)
AccNations2021 <- subset(AccNations, AccNations$AN == 2021)
AccNations2022 <- subset(AccNations, AccNations$AN == 2022)
```

Nous convertissons les points dans le format `ppp` utilisé par le *package* `spatstat`.

```
## Conversion des données sf dans le format de spatstat
# la fonction as.owin est utilisée pour définir la fenêtre de travail
fenetre <- as.owin(ArrDesNations)
## Conversion des points au format ppp pour les différentes années
# 2019
AccN2019.ppp <- ppp(x = st_coordinates(AccNations2019)[,1],
                    y = st_coordinates(AccNations2019)[,2],
                    window = fenetre, check = T)
# 2020
AccN2020.ppp <- ppp(x = st_coordinates(AccNations2020)[,1],
                    y = st_coordinates(AccNations2020)[,2],
                    window = fenetre, check = T)
# 2021
AccN2021.ppp <- ppp(x = st_coordinates(AccNations2021)[,1],
                    y = st_coordinates(AccNations2021)[,2],
                    window = fenetre, check = T)
# 2022
AccN2022.ppp <- ppp(x = st_coordinates(AccNations2022)[,1],
                    y = st_coordinates(AccNations2022)[,2],
                    window = fenetre, check = T)
```

Puis, il faut définir la taille des pixels et le rayon d'influence (*bandwidth* en anglais) :

— Plus la taille des pixels est réduite, plus la taille de l'image résultante est importante et les calculs longs à réaliser.
— Le choix du rayon est aussi délicat. Avec un rayon trop petit, les résultats sont trop détaillés avec des valeurs très faibles. À l'inverse, un rayon trop grand a comme effet de lisser les résultats et de masquer des disparités locales.

Astuce

Ratio entre la taille du pixel et la taille du rayon d'influence
Assurez-vous que la taille du pixel soit bien inférieure (au moins dix fois plus petite) à celle du rayon d'influence afin d'obtenir des résultats précis. À notre connaissance, il n'existe pas de règle pour optimiser la valeur ratio entre la taille du pixel et la taille du rayon d'influence.

Plusieurs algorithmes permettant de sélectionner la valeur optimale du rayon d'influence sont implémentés dans le *package* `spatstat` avec les fonctions `bw.diggle`, `bw.ppl`, `bw.scott` et `bw.CvL`. À la lecture des résultats ci-dessous, la valeur du rayon proposée par l'algorithme de Diggle semble bien trop petite et celle du critère de Cronie et van Lieshout bien trop grande. Le rayon optimal est certainement compris entre 700 et 750 mètres. Par conséquent, nous retenons une taille de pixel de 50 mètres et la valeur du rayon optimisée par la fonction `bw.ppl`.

```
cat("\nDiggle :", bw.diggle(AccN2020.ppp),
    "\nMaximum de vraissemblance :", bw.ppl(AccN2020.ppp),
    "\nCritère de Scott 1 :", bw.scott(AccN2020.ppp)[1],
    "\nCritère  de Scott 2 :", bw.scott(AccN2020.ppp)[2],
    "\nCritère de Cronie et van Lieshout :", bw.CvL(AccN2020.ppp))
```

```
Diggle : 124.196
Maximum de vraissemblance : 494.5408
Critère de Scott 1 : 971.3489
Critère  de Scott 2 : 514.2688
Critère de Cronie et van Lieshout : 1644.489
```

```
## Taille des pixels et du rayon en mètres
pixel_m <- 50
RayonOpt <- bw.ppl(AccN2020.ppp)
```

Comparaison des fonctions kernel

Le paramètre `kernel` de la fonction `density.ppp` permet de sélectionner l'une des quatre fonctions *kernel* (`gaussian`, `quartic`, `epanechnikov` et `disc`). Pour la dernière, le même poids est attribué aux points compris dans le rayon d'influence; elle correspond ainsi à un *kernel* uniforme ou simple. Excepté ce dernier *kernel*, la figure 3.21 démontre que les résultats sont très semblables d'un *kernel* à l'autre.

```
## Calcul de la KDE avec différentes fonctions kernel
# kernel gaussien
kdeG <- density.ppp(AccN2020.ppp, sigma=RayonOpt, eps=pixel_m, kernel="gaussian")
# kernel quadratique
kdeQ <- density.ppp(AccN2020.ppp, sigma=RayonOpt, eps=pixel_m, kernel="quartic")
# kernel Epanechnikov
kdeE <- density.ppp(AccN2020.ppp, sigma=RayonOpt, eps=pixel_m, kernel="epanechnikov")
# kernel disc (uniforme ou simple)
# le même poids est accordé à chaque point dans le rayon
```

```r
kdeD <- density.ppp(AccN2020.ppp, sigma=RayonOpt, eps=pixel_m, kernel="disc")
## Conversion en raster
# étant donné que les valeurs sont exprimées en nombre de points par m2,
# nous les multiplions par 1 000 000 pour obtenir une densité au km2.
RkdeG <- terra::rast(kdeG)*1000000
RkdeQ <- terra::rast(kdeQ)*1000000
RkdeE <- terra::rast(kdeE)*1000000
RkdeD <- terra::rast(kdeD)*1000000
## Projection cartographique
crs(RkdeG) <- "epsg:3798"
crs(RkdeQ) <- "epsg:3798"
crs(RkdeE) <- "epsg:3798"
crs(RkdeD) <- "epsg:3798"
## Visualisation des résultats
tmap_mode("plot")
Carte1 <-
  tm_shape(RkdeG) + tm_raster(style = "cont", palette="Reds", title = "Gaussien")+
  tm_shape(AccNations2020) + tm_dots(col = "black", size = 0.01)+
  tm_shape(ArrDesNations) + tm_borders(col = "black", lwd = 2)+
  tm_layout(frame = FALSE)
Carte2 <-
  tm_shape(RkdeQ) + tm_raster(style = "cont", palette="Reds", title = "Quadratique")+
  tm_shape(AccNations2020) + tm_dots(col = "black", size = 0.01)+
  tm_shape(ArrDesNations) + tm_borders(col = "black", lwd = 2)+
  tm_layout(frame = FALSE)
Carte3 <-
  tm_shape(RkdeE) + tm_raster(style = "cont", palette="Reds", title = "Epanechnikov")+
  tm_shape(AccNations2020) + tm_dots(col = "black", size = 0.01)+
  tm_shape(ArrDesNations) + tm_borders(col = "black", lwd = 2)+
  tm_layout(frame = FALSE)
Carte4 <-
  tm_shape(RkdeD) + tm_raster(style = "cont", palette="Reds", title = "Uniforme")+
  tm_shape(AccNations2020) + tm_dots(col = "black", size = 0.01)+
  tm_shape(ArrDesNations) + tm_borders(col = "black", lwd = 2)+
  tm_scale_bar(breaks  = c(0, 1, 2))+tm_layout(frame = FALSE)
tmap_arrange(Carte1, Carte2, Carte3, Carte4,
             ncol = 2, nrow = 2)
```

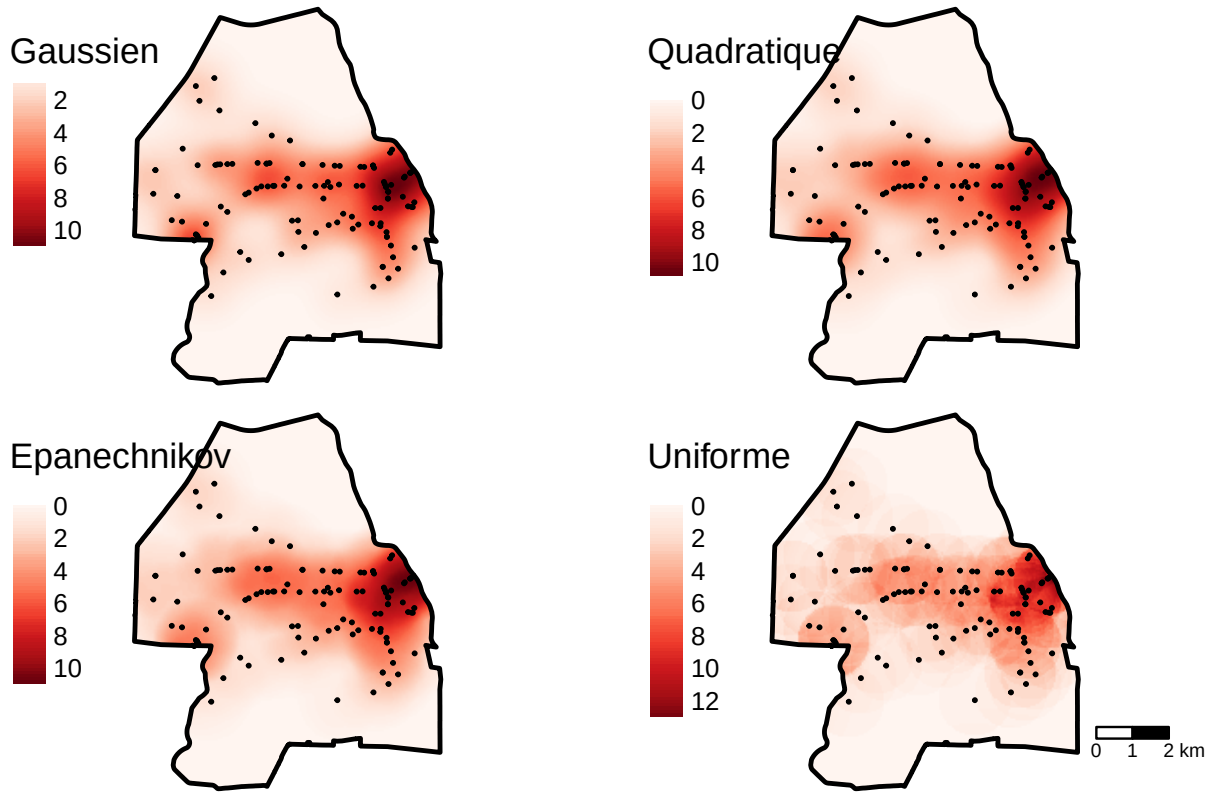


FIGURE 3.21 – Comparaison des résultats de la KDE pour différents kernels

Évaluation de l'impact du rayon d'influence

Le code R permet de réaliser des KDE avec des rayons de 250, 500, 750 et 1000 mètres. La figure 3.21 démontre bien que plus le rayon est grand, plus la carte est lissée.

```
kdeQ250  <- density.ppp(AccN2020.ppp, sigma=250,  eps=50, kernel="quartic")
kdeQ500  <- density.ppp(AccN2020.ppp, sigma=500,  eps=50, kernel="quartic")
kdeQ750  <- density.ppp(AccN2020.ppp, sigma=750,  eps=50, kernel="quartic")
kdeQ1000 <- density.ppp(AccN2020.ppp, sigma=1000, eps=50, kernel="quartic")
RkdeQ250  <- terra::rast(kdeQ250)*1000000
RkdeQ500  <- terra::rast(kdeQ500)*1000000
RkdeQ750  <- terra::rast(kdeQ750)*1000000
RkdeQ1000 <- terra::rast(kdeQ1000)*1000000
crs(RkdeQ250) <- "epsg:3798"
crs(RkdeQ500) <- "epsg:3798"
crs(RkdeQ750) <- "epsg:3798"
crs(RkdeQ1000) <- "epsg:3798"

## Visualisation des résultats
tmap_mode("plot")
Carte1 <-
  tm_shape(RkdeQ250) + tm_raster(style = "cont", palette="Reds", title = "r = 250 m")+
```

```
  tm_shape(AccNations2020) + tm_dots(col = "black", size = 0.01)+
  tm_shape(ArrDesNations) + tm_borders(col = "black", lwd = 2)+
  tm_layout(frame = FALSE)
Carte2 <-
  tm_shape(RkdeQ500) + tm_raster(style = "cont", palette="Reds", title = "r = 500 m")+
  tm_shape(AccNations2020) + tm_dots(col = "black", size = 0.01)+
  tm_shape(ArrDesNations) + tm_borders(col = "black", lwd = 2)+
  tm_layout(frame = FALSE)
Carte3 <-
  tm_shape(RkdeQ750) + tm_raster(style = "cont", palette="Reds", title = "r = 750 m")+
  tm_shape(AccNations2020) + tm_dots(col = "black", size = 0.01)+
  tm_shape(ArrDesNations) + tm_borders(col = "black", lwd = 3)+
  tm_layout(frame = FALSE)
Carte4 <-
  tm_shape(RkdeQ1000) + tm_raster(style = "cont", palette="Reds", title = "r = 1000 m")+
  tm_shape(AccNations2020) + tm_dots(col = "black", size = 0.01)+
  tm_shape(ArrDesNations) + tm_borders(col = "black", lwd = 2)+
  tm_scale_bar(breaks  = c(0, 1, 2))+
  tm_layout(frame = FALSE)
tmap_arrange(Carte1, Carte2, Carte3, Carte4,
             ncol = 2, nrow = 2)
```

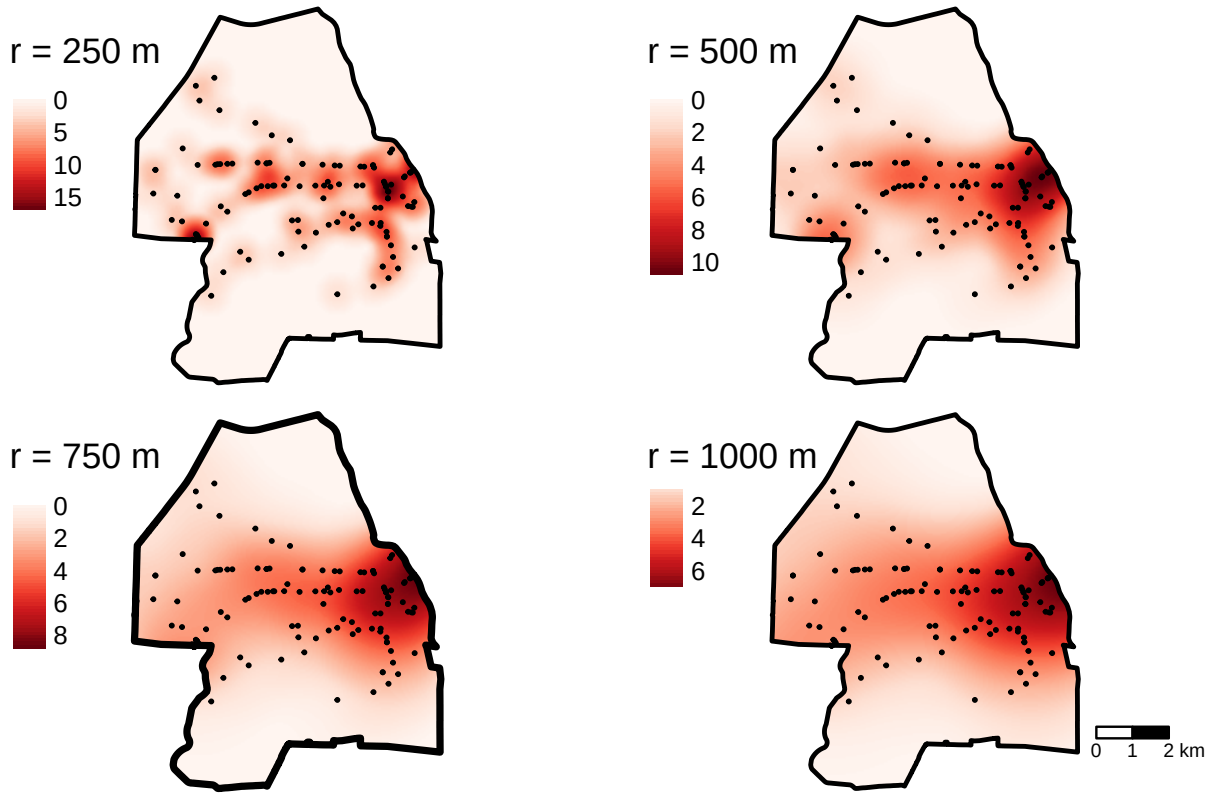


FIGURE 3.22 – Comparaison des résultats de la KDE selon le rayon d'influence

Comparaison entre différentes années

Bien entendu, pour un même jeu de données, il est possible de représenter la densité pour différentes années (figure 3.23).

```
kde2019 <- density.ppp(AccN2019.ppp, sigma=500, eps=50, kernel="quartic")
kde2020 <- density.ppp(AccN2020.ppp, sigma=500, eps=50, kernel="quartic")
kde2021 <- density.ppp(AccN2021.ppp, sigma=500, eps=50, kernel="quartic")
kde2022 <- density.ppp(AccN2022.ppp, sigma=500, eps=50, kernel="quartic")
Rkde2019 <- terra::rast(kde2019)*1000000
Rkde2020 <- terra::rast(kde2020)*1000000
Rkde2021 <- terra::rast(kde2020)*1000000
Rkde2022 <- terra::rast(kde2021)*1000000
crs(Rkde2019) <- "epsg:3798"
crs(Rkde2020) <- "epsg:3798"
crs(Rkde2021) <- "epsg:3798"
crs(Rkde2022) <- "epsg:3798"

Carte1 <-
  tm_shape(Rkde2019) + tm_raster(style = "cont", palette="Reds", title = "2019")+
  tm_shape(AccNations2019) + tm_dots(col = "black", size = 0.01)+
  tm_shape(ArrDesNations) + tm_borders(col = "black", lwd = 2)+
  tm_layout(frame = FALSE)
Carte2 <-
  tm_shape(Rkde2020) + tm_raster(style = "cont", palette="Reds", title = "2021")+
  tm_shape(AccNations2020) + tm_dots(col = "black", size = 0.01)+
  tm_shape(ArrDesNations) + tm_borders(col = "black", lwd = 2)+
  tm_layout(frame = FALSE)
Carte3 <-
  tm_shape(Rkde2021) + tm_raster(style = "cont", palette="Reds", title = "2022")+
  tm_shape(AccNations2021) + tm_dots(col = "black", size = 0.01)+
  tm_shape(ArrDesNations) + tm_borders(col = "black", lwd = 2)+
  tm_layout(frame = FALSE)
Carte4 <-
  tm_shape(Rkde2022) + tm_raster(style = "cont", palette="Reds", title = "2023")+
  tm_shape(AccNations2022) + tm_dots(col = "black", size = 0.01)+
  tm_shape(ArrDesNations) + tm_borders(col = "black", lwd = 2)+
  tm_scale_bar(breaks  = c(0, 1, 2))+tm_layout(frame = FALSE)
tmap_arrange(Carte1, Carte2, Carte3, Carte4,
             ncol = 2, nrow = 2)
```

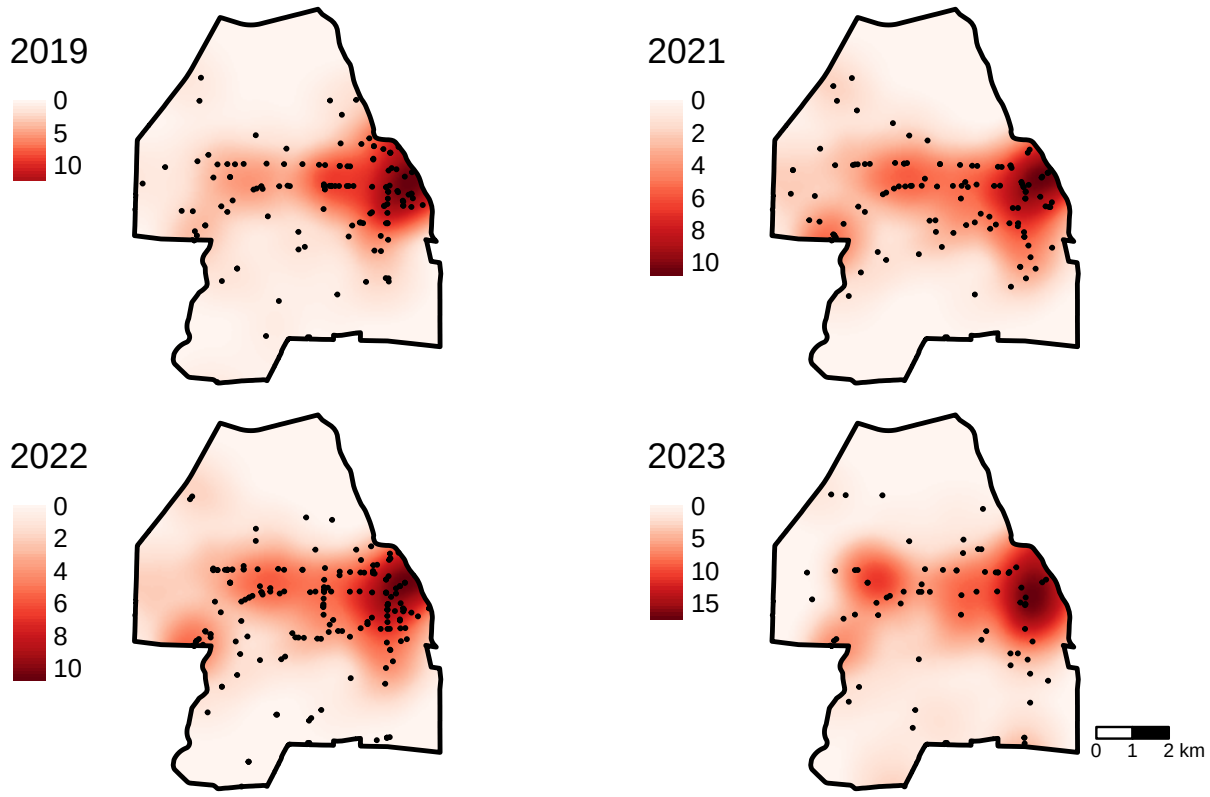


FIGURE 3.23 – Comparaison des résultats de la KDE pour différentes années

3.4.3 Densité spatio-temporelle dans une maille régulière

3.4.3.1 Description de la méthode STKDE

Dans les sections précédentes, nous avons vu que l'analyse de densité dans une maille régulière avec la méthode KDE consiste à répartir la masse des évènements étudiés dans un certain rayon d'influence (*bandwidth*) autour de chaque évènement selon une fonction décroissante (*kernel*).

Cette méthode peut être étendue assez facilement au contexte spatio-temporel. En effet, lorsque les évènements étudiés sont caractérisés par une date d'occurrence, il est possible de répartir leur masse à la fois dans l'espace et dans le temps. Plus exactement, il s'agit d'un kernel bivarié comprenant une *bandwidth* spatiale et une *bandwidth* temporelle. La densité locale est alors estimée avec le produit de la densité spatiale et de la densité temporelle de chaque évènement (équation 3.31).

$$\lambda(s) = \sum_{i=1}^{n} \frac{1}{\pi bw_s^2 \times bw_t} k\left(\frac{d_{si}}{bw_s}\right) k\left(\frac{t_{si}}{bw_t}\right) \text{ avec} \tag{3.31}$$

- bw_t et bw_s les rayons d'influence (*bandwidth*) temporel et spatial.
- t_{si} et d_{si} les distances temporelles et spatiales entre l'évènement i et un point dans l'espace et le temps s.
- k, fonction *kernel*.

La figure 3.24 illustre le principe de la STKDE (*Space-Time Kernel Density Estimatation*). Un évènement a lieu à un moment spécifique et en un lieu spécifique. Plus nous sommes proches de l'évènement dans le temps et dans l'espace, plus la densité est élevée. Hors du *bandwidth* dans le temps ou dans l'espace, la densité est nulle.

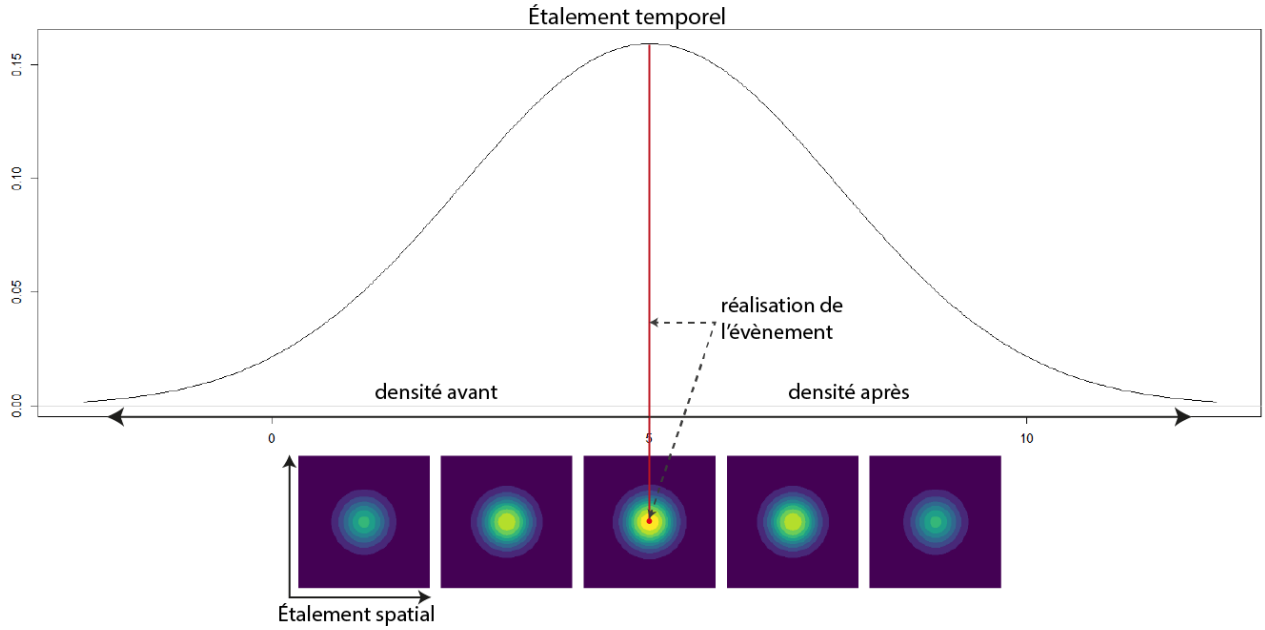


Figure 3.24 – Visualisation de la STKDE

Une représentation particulièrement efficace de la STKDE est une carte animée permettant de visualiser l'évolution spatiale de la densité dans le temps. À la figure 3.25, nous visualisons la densité spatio-temporelle produite par deux évènements ayant eu lieu à des moments et des localisations différentes.

En résumé, la seule différence avec la KDE et la STKDE est que cette dernière nécessite de déterminer une *bandwidth* temporelle (en plus de la *bandwidth* spatiale).

3.4.3.2 Mise en œuvre de la STKDE dans R

Pour mettre en œuvre la STKDE sur les données de collisions de la ville de Sherbrooke, nous utilisons le *package* `sparr` (Davies, Marshall et Hazelton 2018). À la figure 3.26, nous visualisons aisément les périodes durant lesquelles les accidents ont été les plus nombreux.

```
library(spatstat)
library(terra)
library(gifski)
library(sf)
library(tmap)
library(ggplot2)
library(sparr)
library(lubridate)
library(classInt)
```

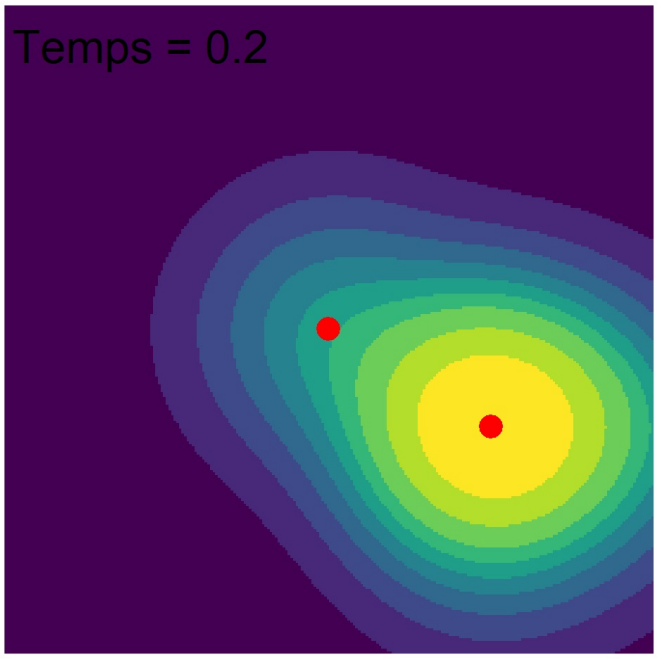


FIGURE 3.25 – Visualisation animée de la STKDE

```
library(viridis)

routes <- st_read('data/chap01/shp/Segments_de_rue.shp', quiet = TRUE)
collisions <- st_read('data/chap04/DataAccidentsSherb.shp', quiet = TRUE)
## Reprojection dans le même système
routes <- st_transform(routes, 2949)
collisions <- st_transform(collisions, 2949)
## Visualisation de la densité temporelle
collisions$dt <- as_date(collisions$DATEINCIDE)
collisions$dt_num <- as.numeric(collisions$dt - min(collisions$dt))
ggplot(collisions, aes(x=dt)) +
  geom_density(bw = 'sj', color = "blue", lwd = 1) +
  labs(y= "Densité", x = "Date")+
  scale_x_date(date_breaks = "6 months", date_labels =  "%b %Y")+
  theme_bw()
```

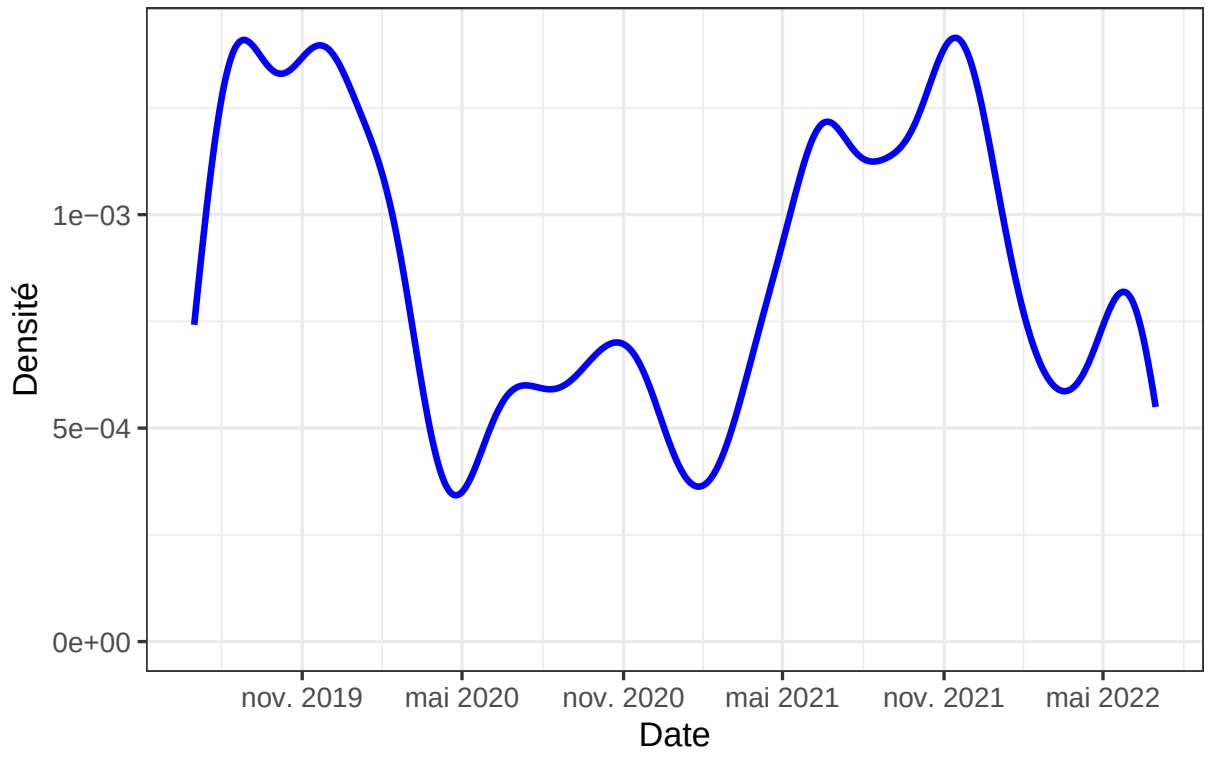


FIGURE 3.26 – Distribution temporelle de la densité des accidents

Pour calculer la STKDE, nous devons déterminer les valeurs des deux *bandwidths*, réalisé ici avec une approche dite de validation croisée par maximum de vraisemblance.

```
## Préparation des données de collisions
fenetre <- as.owin(st_buffer(collisions,500))
XY <- st_coordinates(collisions)
collisions.ppp <- ppp(x = XY[,1],
                      y = XY[,2],
                      window = fenetre, check = T)
## Estimation des deux meilleures bandwidths
scores_bw <- LIK.spattemp(collisions.ppp,
                          tt = collisions$dt_num,
                          tlim = c(0,1100),
                          parallelise = NA,verbose = FALSE)
print(scores_bw)
```

```
       h   lambda
736.8287 147.4328
```

Les valeurs optimales obtenues avec l'approche dite de validation croisée par maximum de vraisemblance sont de 736 mètres pour la *bandwidth* spatiale et 147 jours pour la *bandwidth* temporelle. Cette dernière est vraisemblablement trop grande. Par conséquent, nous utilisons des *bandwidths* de 750 mètres et 14 jours. Notez que les valeurs de densité

sont multipliées par 10 000, car leur étalement dans l'espace et le temps conduit à des valeurs trop petites pour le *package* `tmap`. Les résultats sont présentés sous la forme d'une animation à la figure 3.27.

```
# Calcul des valeurs de densités
dens_vals <- spattemp.density(collisions.ppp,
                              h = 750,
                              lambda = 14,
                              tt = collisions$dt_num,
                              tlim = c(0,1100),
                              sres = 500,
                              tres = 150, verbose = FALSE)

## Extraction des rasters à chaque période
all_rasts <- lapply(dens_vals$z, function(x){
  my_rast <- terra::rast(x) * 10000 # petit ajustement pour la carto
  vals <- terra::values(my_rast)
  vals <- ifelse(is.na(vals), 0, vals)
  terra::values(my_rast) <- vals
  terra::crs(my_rast) <- 'epsg:32188'
  return(my_rast)
})
## Extraction des valeurs pour créer une échelle commune de couleur
all_densities <- do.call(c, lapply(all_rasts, function(x){
  sample(terra::values(x),size = 100,replace = FALSE)
}))
color_breaks <- classIntervals(all_densities, n = 10, style = "kmeans")
## Préparation des dates
timestamps <- round(as.numeric(names(dens_vals$z)))
time_frames <- min(collisions$dt) + days(timestamps)
## Compilation des cartes
all_maps <- lapply(1:length(time_frames), function(i){
  tm_shape(all_rasts[[i]]) +
    tm_raster(breaks = color_breaks$brks, palette = viridis(10)) +
    tm_layout(legend.show = FALSE, frame = FALSE, title = time_frames[[i]])
})

# Création d'une animation pour produire la carte animée
tmap_animation(all_maps, filename = "images/Chap03/animated_STKDE_sherbrooke.gif",
               width = 500, height = 500, dpi = 150, delay = 25)
```

3.5 Quiz de révision du chapitre

Questions

— **Quelle est la différence entre le centre moyen et le point central?**

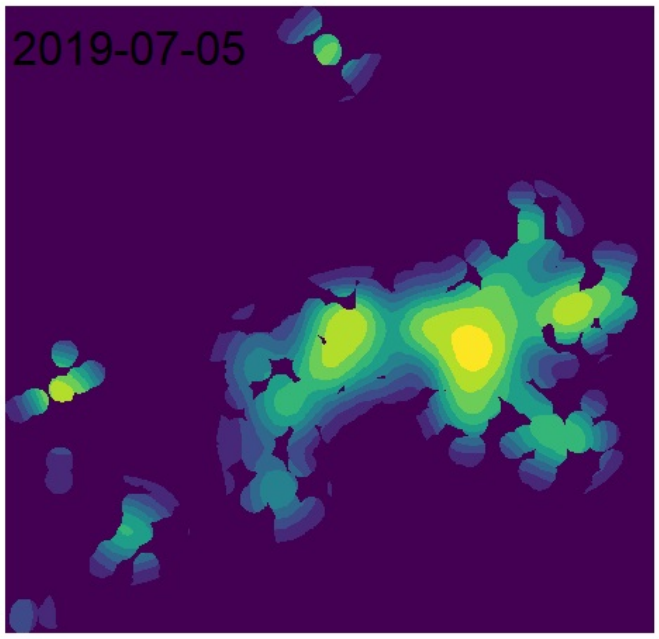
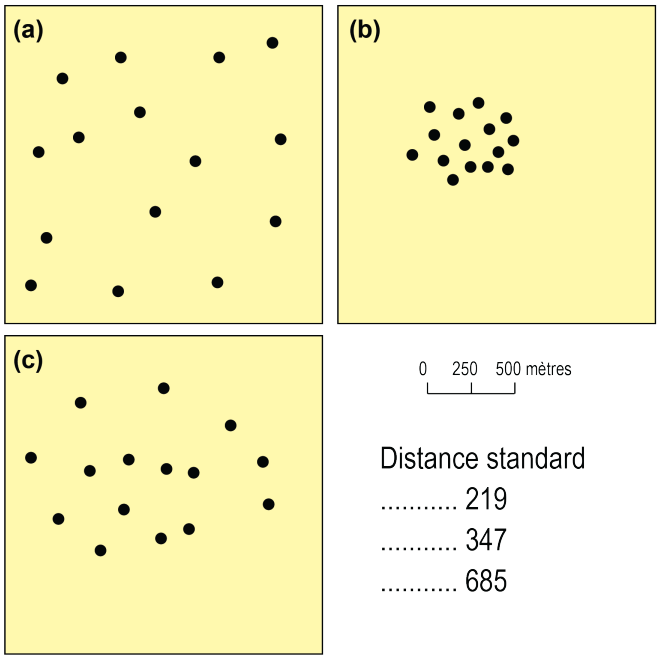


FIGURE 3.27 – Distribution spatio-temporelle de la densité des accidents

— Le centre moyen peut être pondéré contrairement au point central.
— Le point central est un point qui fait partie du semis de points initial.

Relisez au besoin la section 3.2.1.2.

— **Associer la distance standard à chacun des trois semis de points suivants :**



- A = 219, B = 347, C = 685
- A = 685, B = 219, C = 347
- A = 685, B = 347, C = 219

Relisez au besoin la section 3.2.2.

- **Quelles sont les quatre principales manières de représenter graphiquement la dispersion d'un semis de points?**

- Enveloppe convexe
- Centre moyen et point central
- Rectangle construit avec les déviations standards des X et des Y
- Distance standard
- Ellipse de déviation de distance standard

Relisez au besoin la section 3.2.3.1.

- **Quelles sont les trois principales distributions d'un semis de points?**

- Distribution dispersée
- Distribution aléatoire
- Distribution concentrée
- Distribution normale

Relisez au besoin la section 3.3.

- **À la lecture des résultats de la méthode du plus proche voisin, quelle année a la distribution spatiale la plus concentrée?**

Année	points (n)	*R* observé	*R* attendu	Indice plus proche voisin
2019	251	358	860	0,416
2020	383	262	696	0,376
2021	344	323	735	0,440
2022	220	358	919	0,389

- 2019
- 2020
- 2021
- 2022

Relisez au besoin la section 3.3.1.

- **En analyse des quadrats, quelles sont les trois principales formes utilisées?**

- Triangle
- Carré
- Hexagone
- Cercle

Relisez au besoin la section 3.3.2.1.

- **Quel est le paramètre influençant le plus les résultats de la méthode KDE?**

  - La fonction kernel utilisée
  - La taille du rayon d'influence

  Relisez au besoin la section 3.4.2.1.
- **Quelles sont les principales fonctions kernel?**

  - Gaussienne
  - Quadratique
  - Uniforme
  - Manhattan
  - Epanechnikov

  Relisez au besoin la section 3.4.2.1.

Réponses

- Quelle est la différence entre le centre moyen et le point central?
  - Le point central est un point qui fait partie du semis de points initial.
- Associer la distance standard à chacun des trois semis de points suivants :
  - A = 685, B = 219, C = 347
- Quelles sont les quatre principales manières de représenter graphiquement la dispersion d'un semis de points?
  - Enveloppe convexe
  - Rectangle construit avec les déviations standards des X et des Y
  - Distance standard
  - Ellipse de déviation de distance standard
- Quelles sont les trois principales distributions d'un semis de points?
  - Distribution dispersée
  - Distribution aléatoire
  - Distribution concentrée
- À la lecture des résultats de la méthode du plus proche voisin, quelle année a la distribution spatiale la plus concentrée?
  - 2020
- En analyse des quadrats, quelles sont les trois principales formes utilisées?
  - Carré
  - Hexagone
  - Cercle
- Quel est le paramètre influençant le plus les résultats de la méthode KDE?
  - La taille du rayon d'influence
- Quelles sont les principales fonctions kernel?
  - Gaussienne
  - Quadratique
  - Uniforme
  - Epanechnikov

3.6 Exercices de révision

Exercice

Exercice 1. Calcul du centre moyen et de la distance standard pour les accidents
Complétez le code ci-dessous.

```
library(sf)
library(tmap)
## Importation des données
Arrondissements <-  st_read(dsn = "data/chap03/Arrondissements.shp", quiet=TRUE)
Incidents <- st_read(dsn = "data/chap03/IncidentsSecuritePublique.shp", quiet=TRUE)
## Changement de projection
Arrondissements <- st_transform(Arrondissements, crs = 3798)
Incidents <- st_transform(Incidents, crs = 3798)
## Couche pour les accidents
Accidents <- subset(Incidents, Incidents$DESCRIPTIO %in%
                      c("Accident avec blessés", "Accident mortel"))
## Coordonnées et projection cartographique
xy <- À compléter
ProjCarto <- À compléter
## Centre moyen
CentreMoyen <- data.frame(À compléter)
CentreMoyen <- st_as_sf(CentreMoyen, coords = c("X", "Y"), crs = À compléter)
# Distance standard combinée
CentreMoyen$DS <- À compléter
CercleDS <- À compléter
head(CercleDS)
```

Correction à la section 12.3.1.

Exercice

Exercice 2. Calcul et cartographie de la densité des accidents dans un maillage irrégulier
Pour l'année 2021, complétez le code ci-dessous.

```
library(sf)
library(tmap)
## Importation des données
SR <- st_read(dsn = "data/chap03/Recen2021Sherbrooke.gpkg",
              layer = "DR_SherbSRDonnees2021", quiet=TRUE)
## Couche pour les accidents pour l'année 2021
Acc2021 <- subset(Incidents, Incidents$DESCRIPTIO %in%
                    c("Accident avec blessés", "Accident mortel")
                  & ANNEE==2021)
## Nous nous assurons que les deux couches ont la même projection cartographique
SR <- st_transform(SR, st_crs(Acc2021))
## Calcul du nombre d'incidents par SR
SR$Acc2021 <- À compléter
## Calcul du nombre de méfaits pour 1000 habitants
SR$DensiteMAcc2021Hab <- À compléter
## Cartographie
tm_shape(SR)+
  tm_polygons(col= À compléter, style="pretty",
              title="Nombre pour 1000 habitants",
              border.col = "black", lwd = 1)+
  tm_bubbles(size = À compléter, border.col = "black", alpha = .5,
             col = "aquamarine3", title.size = "Nombre", scale = 1.5)+
  tm_layout(frame = FALSE)+tm_scale_bar(text.size = .5, c(0, 5, 10))
```

Correction à la section 12.3.2.

Exercice

Exercice 3. Calcul et cartographie de la densité des accidents dans un maillage régulier
Pour l'année 2021, complétez le code ci-dessous.

```
library(sf)
library(spatstat)
library(tmap)
library(terra)

## Importation des données
Arrondissements <-  st_read(dsn = "data/chap03/Arrondissements.shp", quiet=TRUE)
Incidents <- st_read(dsn = "data/chap03/IncidentsSecuritePublique.shp", quiet=TRUE)
## Changement de projection
Arrondissements <- st_transform(Arrondissements, crs = 3798)
Incidents <- st_transform(Incidents, crs = 3798)
## Couche pour les méfaits pour l'année 2021
M2021 <- subset(Incidents, DESCRIPTIO == "Méfait" & ANNEE==2021)
## Pour accélérer les calculs, nous retenons uniquement l'arrondissement des Nations
# Couche pour l'arrondissement des Nations
ArrDesNations <- subset(Arrondissements, NOM == "Arrondissement des Nations")
# Sélection des accidents localisés dans l'arrondissement Des Nations
RequeteSpatiale <- st_intersects(M2021, ArrDesNations, sparse = FALSE)
M2021$Nations <- RequeteSpatiale[, 1]
M2021Nations <- subset(M2021, M2021$Nations == TRUE)

## Conversion des données sf dans le format de spatstat
# la fonction as.owin est utilisée pour définir la fenêtre de travail
fenetre <- à complétér
## Conversion des points au format ppp pour les différentes années
M2021.ppp <- à complétér

## Kernel quadratique avec un rayon de 500 mètres et une taille de pixel de 50 mètres
kdeQ <- density.ppp(M2021.ppp, sigma=500, eps=50, kernel="quartic")
## Conversion en raster
RkdeQ <- terra::rast(kdeQ)*1000000
## Projection cartographique
crs(RkdeQ) <- "epsg:3857"
## Visualisation des résultats
tmap_mode("plot")
  À compléter
```

Correction à la section 12.3.3.

4 Méthodes de détection d'agrégats spatiaux et spatio-temporels

Dans ce chapitre, nous abordons deux familles de méthodes de détection d'agrégats spatiaux et spatio-temporels qui s'appliquent à des géométries différentes : les méthodes de classification basées sur la densité des points (couche de points), principalement les algorithmes DBSCAN (Ester et al. 1996) et ST-DBSCAN (Birant et Kut 2007), et les méthodes de balayage de Kulldorff (1997) (couche polygonale).

Package

Liste des *packages* utilisés dans ce chapitre

- Pour importer et manipuler des fichiers géographiques :
  - `sf` pour importer et manipuler des données vectorielles.
  - `dplyr` pour manipuler les données.
- Pour construire des cartes et des graphiques :
  - `tmap` pour les cartes.
  - `ggplot2` pour construire des graphiques.
- `dbscan` pour l'algorithme DBSCAN.
- `SpatialEpi` pour les méthodes de balayage de Kurlldoff.

4.1 Agrégats d'entités spatiales ponctuelles

4.1.1 DBSCAN : agrégats spatiaux

Objectif

Pourquoi utiliser DBSCAN ?
Dans le chapitre précédent, portant sur les *méthodes de répartition ponctuelles*, nous avons abordé la méthode KDE permettant de cartographier la densité de points dans une maille régulière (section 3.4.2). La carte de chaleur obtenue avec la KDE représente les valeurs de densité (variable continue) pour les pixels couvrant le territoire à l'étude.
Avec l'algorithme DBSCAN (Ester et al. 1996), l'objectif est différent : il s'agit d'**identifier des agrégats spatiaux d'évènements ponctuels dans un territoire donné** (par exemple, des cas de maladies, d'accidents, d'espèces fauniques ou végétales, de crimes, etc.). Autrement dit, il s'agit d'identifier plusieurs régions du territoire à l'étude dans lesquelles la densité de points est forte.
Concrètement, si la méthode KDE renvoie une variable continue pour l'ensemble du territoire, l'algorithme DBSCAN renvoie une variable qualitative uniquement pour les points du jeu de données.

4.1.1.1 Fonctionnement de DBSCAN

DBSCAN (***D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise*) est un algorithme de classification non supervisée qui regroupe des observations en fonction de **leur densité** dans un espace à deux, trois ou n dimensions (Ester et al. 1996). Comme pour toute autre méthode de classification non supervisée, ces dimensions sont des variables. Par conséquent, en appliquant DBSCAN sur les coordonnées géographiques d'entités ponctuelles 2D (x, y) ou 3D (x, y, z), nous classifions les points du jeu de données.

Prenons un jeu de données fictives (figure 4.1, a). À l'œil nu, nous identifions clairement cinq régions distinctes avec une forte densité de points et des zones de faible densité; ces dernières étant représentées par les points noirs avec DBSCAN (figure 4.1, b).

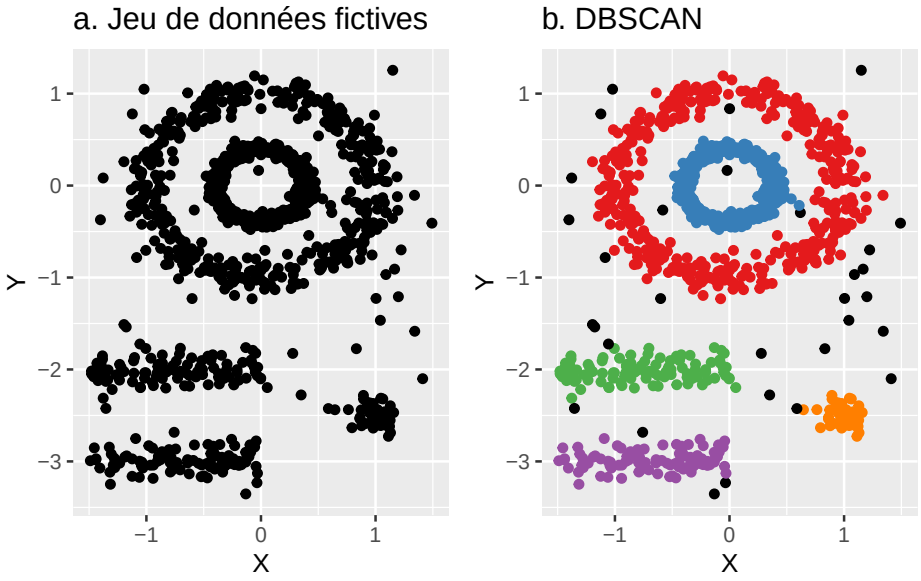


Figure 4.1 – Jeu de données fictives et classification DBSCAN avec cinq classes

L'intérêt majeur de l'algorithme DBSCAN est qu'il est basé sur la **densité des points** et non sur la **distance entre les points** comme les algorithmes classiques de classification non supervisée que sont les k-moyennes, k-médianes, k-médoïdes ou la classification ascendante hiérarchique. Tel qu'illustré à la figure 4.2, l'utilisation de la distance pour identifier cinq groupes de points renvoie des résultats peu convaincants. D'une part, tous les points appartiennent à une classe, sans séparer les régions de fortes et de faibles densités. D'autre part, les algorithmes classiques basés sur la distance ne parviennent pas à bien identifier les deux agrégats circulaires (bleu et rouge à la figure 4.1, b) et parfois linéaires (vert et mauve à la figure 4.1, b).

L'algorithme DBSCAN comprend deux paramètres qui doivent être définis par la personne utilisatrice :

- **Le rayon de recherche**, dénommé ϵ (epsilon), habituellement basé sur la distance euclidienne. Les distances de Manhattan ou réticulaires peuvent aussi être utilisées.
- **Le nombre minimum de points**, dénommé $MinPts$, requis pour qu'un point, incluant lui-même, soit considéré comme un point central et appartienne à un agrégat, un regroupement (*cluster* en anglais).

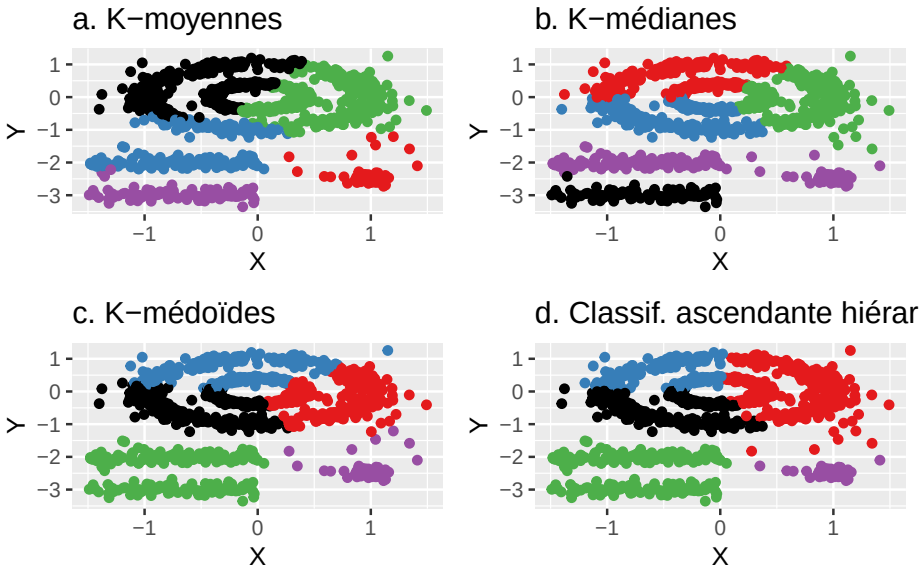


FIGURE 4.2 – Classification avec d'autres algorithmes basés sur la distance

Note

Avantage de DBSCAN : nul besoin de spécifier le nombre d'agrégats (*clusters*)!
Comparativement à d'autres méthodes de classification non supervisées comme les k-moyennes, k-médianes et k-médoïdes, DBSCAN ne requiert pas de spécifier le nombre de classes à identifier dans le jeu de données. Autrement dit, appliqué à des géométries ponctuelles, l'algorithme DBSCAN détecte autant d'agrégats spatiaux que nécessaire en fonction des valeurs des deux paramètres (ϵ et $MinPts$).

À la figure 4.3, nous appliquons l'algorithme DBSCAN à un semis de points avec un rayon de recherche de 500 mètres ($\epsilon = 500$) et un nombre minimum de cinq points ($MinPts = 5$). Dans un premier temps, l'algorithme distingue trois types de points :

1. Des **points centraux** (*core points* en anglais) qui ont au moins cinq points (incluant eux-mêmes) dans un rayon de 500 mètres (points rouges).
2. Des **points frontières** (*border points*) qui ont moins de cinq points (incluant eux-mêmes) dans un rayon de 500 mètres, mais qui sont inclus dans la zone tampon de 500 mètres d'un point central (points bleus).
3. Des **points aberrants** (*noise points*) qui ont moins de cinq points (incluant eux-mêmes) dans un rayon de 500 mètres et qui ne sont pas inclus dans la zone tampon d'un point central (points noirs).

Par la suite, les étapes de l'algorithme sont les suivantes :

— **Étape 1.** Formation du premier agrégat

  — Nous tirons au hasard un point central et l'assignons au premier agrégat (groupe ou *cluster*).
  — Puis, les points compris dans la zone tampon du premier point central sont ajoutés à ce premier agrégat.
  — De façon itérative, nous étendons l'agrégat avec les points centraux ou frontières qui sont compris dans les zones tampons des points ajoutés précédemment.

— **Étape 2.** Formation d'autres agrégats

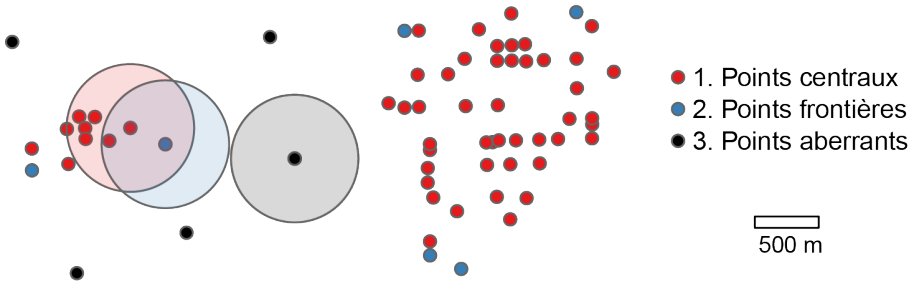


FIGURE 4.3 – Trois types de points identifiés par l'algorithme DBSCAN

— Lorsque le premier agrégat est complété, nous tirons au hasard un autre point central n'appartenant pas au premier agrégat.
— Nous appliquons la même démarche qu'à l'étape 1 pour étendre et compléter cet autre agrégat.
— Les deux sous-étapes ci-dessus sont répétées jusqu'à ce que tous les points centraux et frontières soient assignés à un agrégat.

Nous obtenons ainsi k agrégats (valeurs de 1 à k) tandis que les points aberrants sont affectés à la même classe (valeur de 0 habituellement). Appliqué au semis de points, DBSCAN a détecté deux agrégats et quatre points aberrants (figure 4.4).

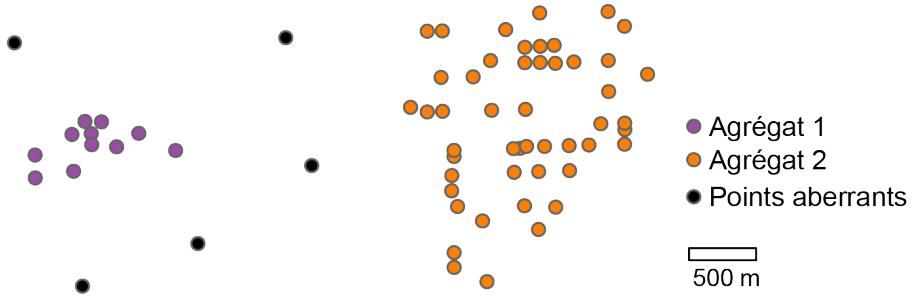


FIGURE 4.4 – Résultats de l'algorithme DBSCAN

4.1.1.2 Sensibilité et optimisation des paramètres de DBSCAN

Les résultats de l'algorithme de DBSCAN varient en fonction de ses deux paramètres, soit le rayon de recherche (ϵ) et le nombre minimum de points ($MinPts$).

Concernant le paramètre ϵ, plus sa valeur est réduite, plus le nombre de points identifiés comme aberrants est important. Inversement, plus elle est grande, plus le nombre d'agrégats diminue. En guise d'illustration, faisons varier la valeur du rayon en maintenant à cinq le nombre minimum de points :

— Avec un rayon de 250 mètres, cinq agrégats sont identifiés tandis que 29 points sont considérés comme du bruit (figure 4.5, a).
— Avec un rayon de 500 mètres, la solution est plus optimale avec deux agrégats et cinq points aberrants (figure 4.5, b).
— Avec un rayon de 1000 mètres, deux agrégats sont aussi identifiés. Par contre, il ne reste plus qu'un point aberrant. Par conséquent, quatre points qui, à l'œil nu, sont très éloignés d'un agrégat y sont pourtant affectés (figure 4.5, c).
— Avec un rayon de 1500 mètres, tous les points sont affectés à un et un seul agrégat (figure 4.5, d).

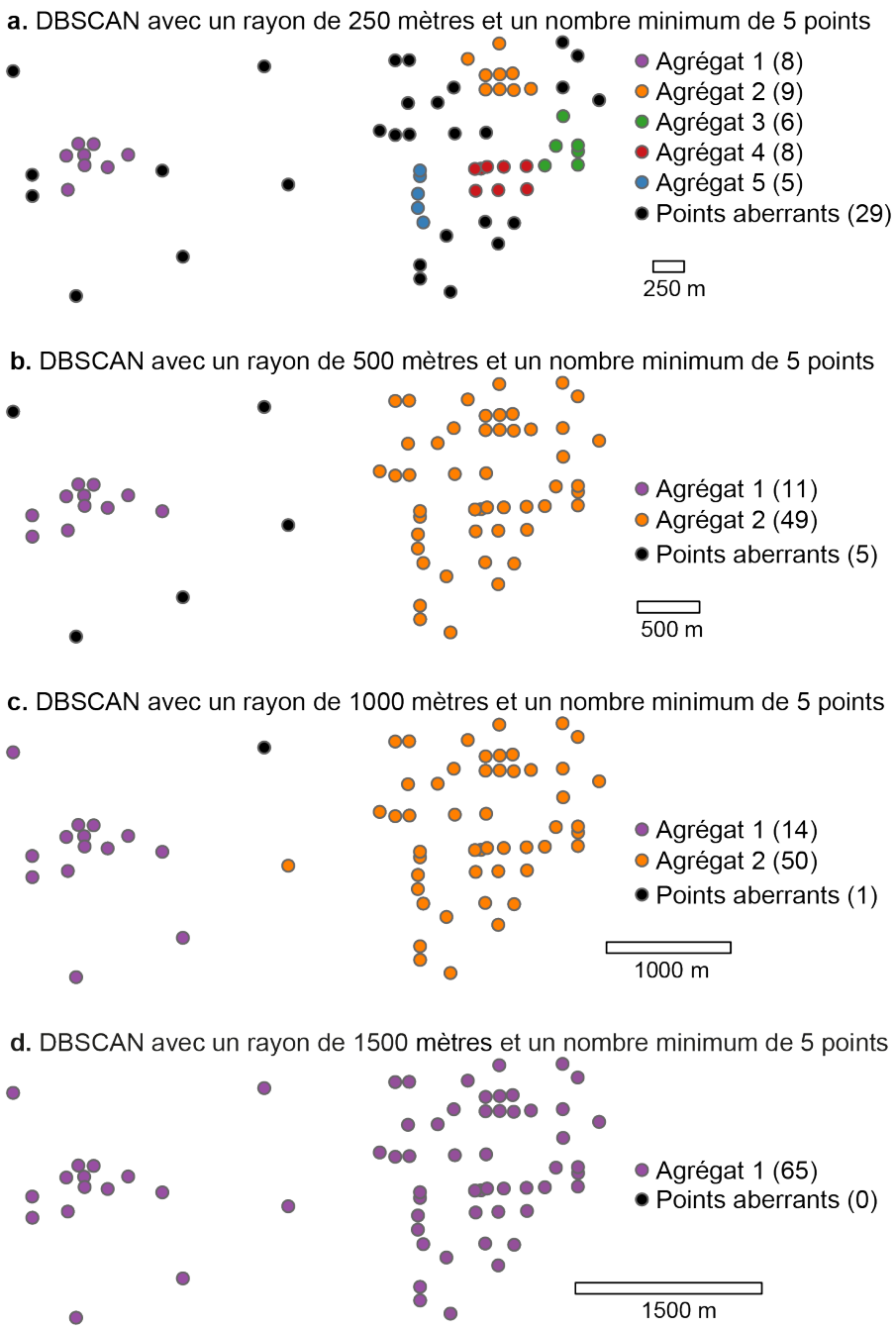


FIGURE 4.5 – Variations de résultats de l'algorithme DBSCAN selon la taille du rayon

Concernant le paramètre $MinPts$, plusieurs stratégies ont été proposées pour fixer sa valeur :

- $MinPts \geq dim(D) + 1$, c'est-à-dire que sa valeur doit être minimalement égale au nombre de dimensions (variables) plus un du jeu de données.
- $MinPts = dim(D) \times 2$, c'est-à-dire que le nombre de points devrait être égal à deux fois le nombre de dimensions du tableau (Sander et al. 1998).
- $MinPts = 4$ quand le jeu de données ne comprend que deux dimensions (Ester et al. 1996), soit un critère qui s'applique à des géométries ponctuelles 2D.

Après avoir fixé le nombre minimal de points, nous pouvons optimiser la valeur du rayon de recherche de la façon suivante :

- Pour chacun des points, calculer la distance au $k^{\text{ième}}$ point le plus proche.
- Trier les valeurs obtenues pour construire un graphique en courbe.
- Dans ce graphique, utiliser le *critère du coude* pour repérer la ou les valeurs signalant un décrochement dans la courbe. À la lecture de la figure 4.6, les valeurs d'epsilon (ϵ) à retenir pourraient être 300, 350, 425 et 450 mètres.

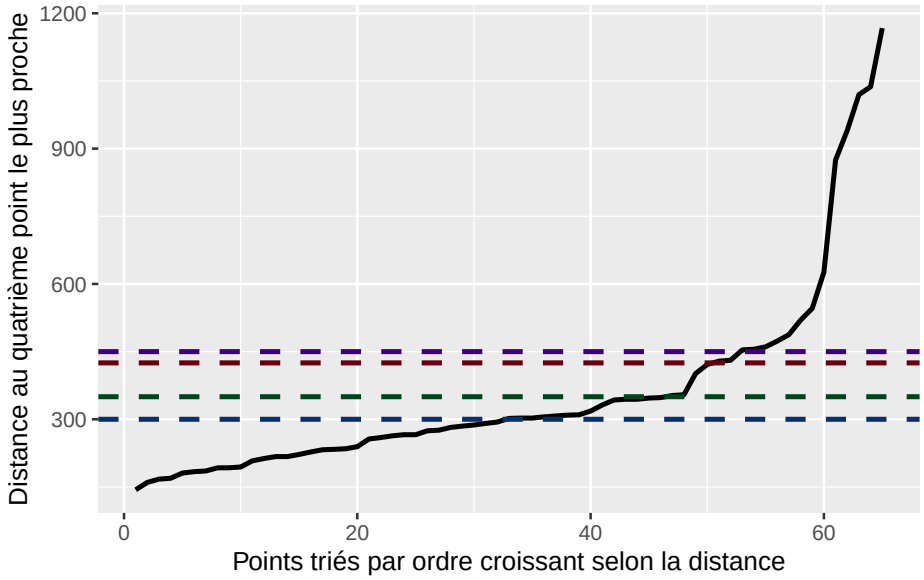


Figure 4.6 – Optimisation de la valeur d'epsilon

Si vous repérez plusieurs seuils de distance dans le graphique des distances au $k^{\text{ième}}$ plus proche voisin, réalisez et comparez les résultats des DBSCAN avec ces valeurs d'epsilon. À la figure 4.7, nous constatons que les résultats avec des seuils de 425 et 450 mètres sont identiques et semblent optimaux. Par contre, la solution avec un rayon de 350 mètres identifie deux points aberrants qui pourraient être intégrés au deuxième agrégat tandis que celle avec un rayon de 300 mètres identifie un agrégat supplémentaire, mais classifie de nombreux points comme aberrants.

⚠ **Attention**

Quel résultat choisir parmi les quatre solutions?
Comme pour toute analyse de classification, votre choix peut être objectif et reposer uniquement sur des indicateurs statistiques (ici, le graphique des distances au *k* plus proche voisin). Il devrait aussi s'appuyer sur vos connaissances du terrain. Par exemple, l'identification d'un troisième agrégat avec une valeur d'epsilon fixée à 300 mètres pourrait refléter selon vous une réalité terrain particulièrement intéressante qui motiverait fortement le choix de cette solution.

🚀 **Aller plus loin**

Autres algorithmes de classification non supervisée basée sur la densité
Bien que DBSCAN soit l'algorithme le plus utilisé, d'autres algorithmes basés sur la densité peuvent être utilisés pour détecter des agrégats spatiaux de points, notamment :

- HDBSCAN (***H**ierarchical **D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise*) (Campello, Moulavi et Sander 2013). Brièvement, cette version modifiée de DBSCAN permet d'obtenir une hiérarchie de partitions, comme dans une classification ascendante hiérarchique.
- OPTICS (***O**rdering **P**oints **T**o **I**dentify the **C**lustering **S**tructure*) (Campello, Moulavi et Sander 2013). Avec OPTICS, la distance de voisinage (ϵ) n'a pas besoin d'être spécifiée. Succinctement, pour chaque point du jeu de données, il utilise la distance au k ($MinPts$) plus proche voisin.

Application à des évènements localisés sur un réseau de rues
Lorsque les évènements sont localisés sur un réseau de rues (des accidents par exemple), il convient d'utiliser une autre métrique que la distance euclidienne pour le rayon de recherche (ϵ), soit la distance du chemin le plus court à travers le réseau de rues, ce que nous verrons au chapitre suivant (section 6.4). Geoff Boeing a aussi proposé un un code Python basé sur la librairie OSMnx.

4.1.2 ST-DBSCAN : agrégats spatio-temporels

Derya Birant et Alp Kut (2007) ont proposé une modification de l'algorithme de DBSCAN afin qu'il puisse s'appliquer à des données spatio-temporelles (x, y, d) avec d étant la date de l'évènement. Dénommé ST-DBSCAN, l'algorithme comprend toujours les deux paramètres de DBSCAN ($MinPts$ et ϵ), auxquels s'ajoute un autre paramètre ϵ pour le temps (défini en heure, jour, semaine, mois ou année). Autrement dit, deux paramètres de distance sont utilisés : ϵ_1 pour la proximité spatiale (comme avec DBSCAN) et ϵ_2 pour la proximité temporelle (Birant et Kut 2007). De la sorte, deux points sont considérés comme voisins si la distance spatiale et la distance temporelle sont toutes deux inférieures aux seuils fixés.

⚠ **Attention**

Fenêtre temporelle des points formant un agrégat
Attention, les points formant un agrégat peuvent avoir une fenêtre temporelle bien plus grande que le seuil ϵ_2 fixé. Par exemple, fixons les valeurs de ϵ_1 à 500 mètres et de ϵ_2 à 7 jours. Si le point A (d = 2023-01-15) est distant de 400 mètres du point B (d = 2023-01-20), les deux points sont considérés comme voisins. Par contre, si le point B est distant du point C (d = 2023-01-25) de moins de 500 mètres, il peut être aussi agrégé à l'agrégat puisque l'écart temporel entre B et C est de 5 jours.
Habituellement, plus la valeur de ϵ_2 est faible, plus le nombre de points considérés comme aberrants est important.

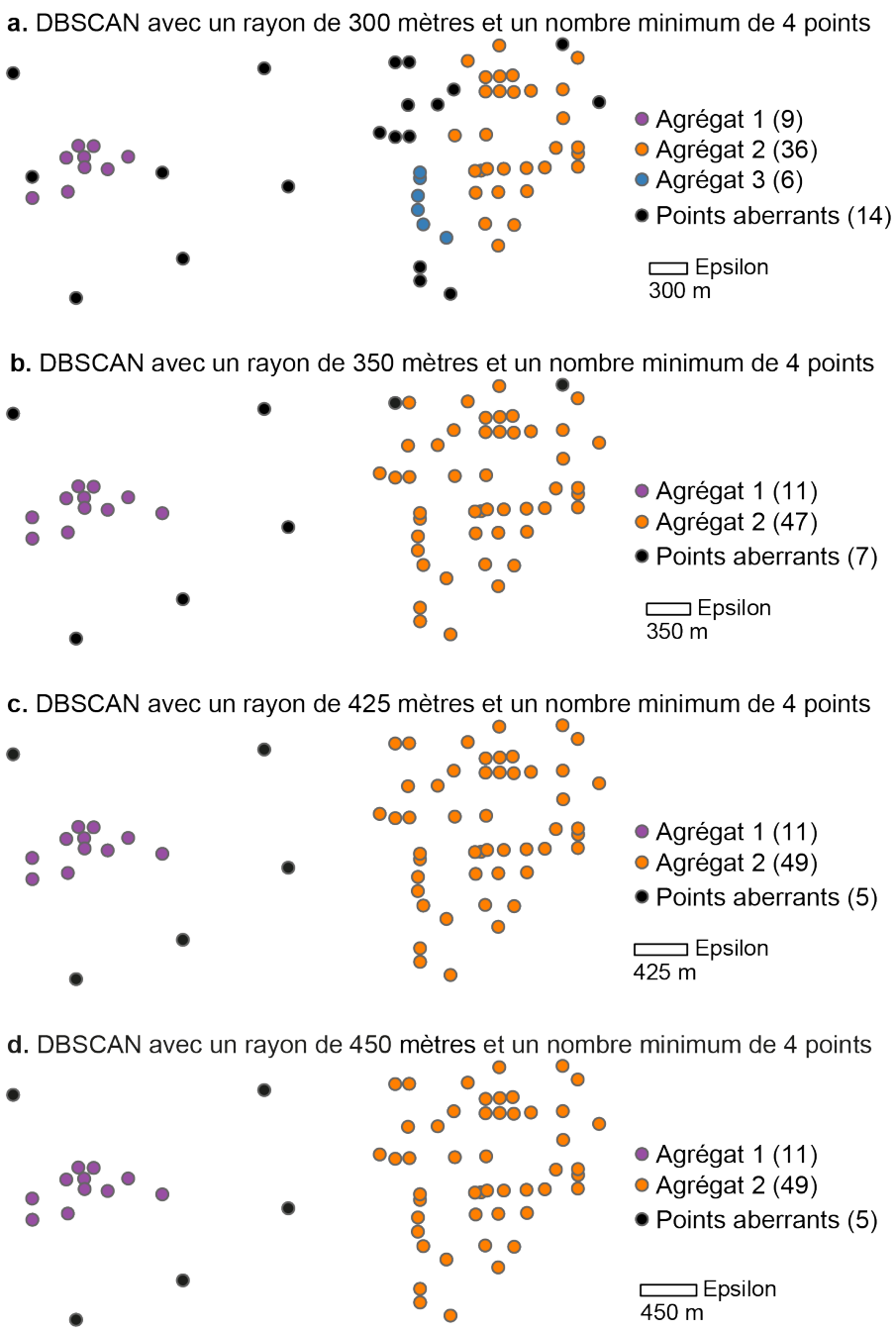


FIGURE 4.7 – Comparaison de solutions DBSCAN avec différentes valeurs d'epsilon

4.1.3 Mise en œuvre dans R

4.1.3.1 DBSCAN

Nous utilisons le *package* `dbscan` (Hahsler et Piekenbrock 2022; Hahsler, Piekenbrock et Doran 2019) dans lequel sont implémentés plusieurs algorithmes, dont DBSCAN, mais aussi OPTICS et HDBSCAN. La fonction `dbscan(x, eps, minPts, weights = NULL)` comprend plusieurs paramètres :

- `x`: une matrice, un *DataFrame*, un objet `dist` ou un objet `frNN`.
- `eps`: le rayon de recherche epsilon (ϵ).
- `minPts`: le nombre de points minimum requis pour que chaque point soit considéré comme un point central.
- `weights`: un vecteur numérique optionnel pour pondérer les points.

Pour illustrer le fonctionnement de la méthode DBSCAN, nous avons extrait les accidents d'un jeu de données sur les incidents de sécurité publique survenus sur le territoire de la Ville de Sherbrooke de juillet 2019 à juin 2022 (figure 4.8).

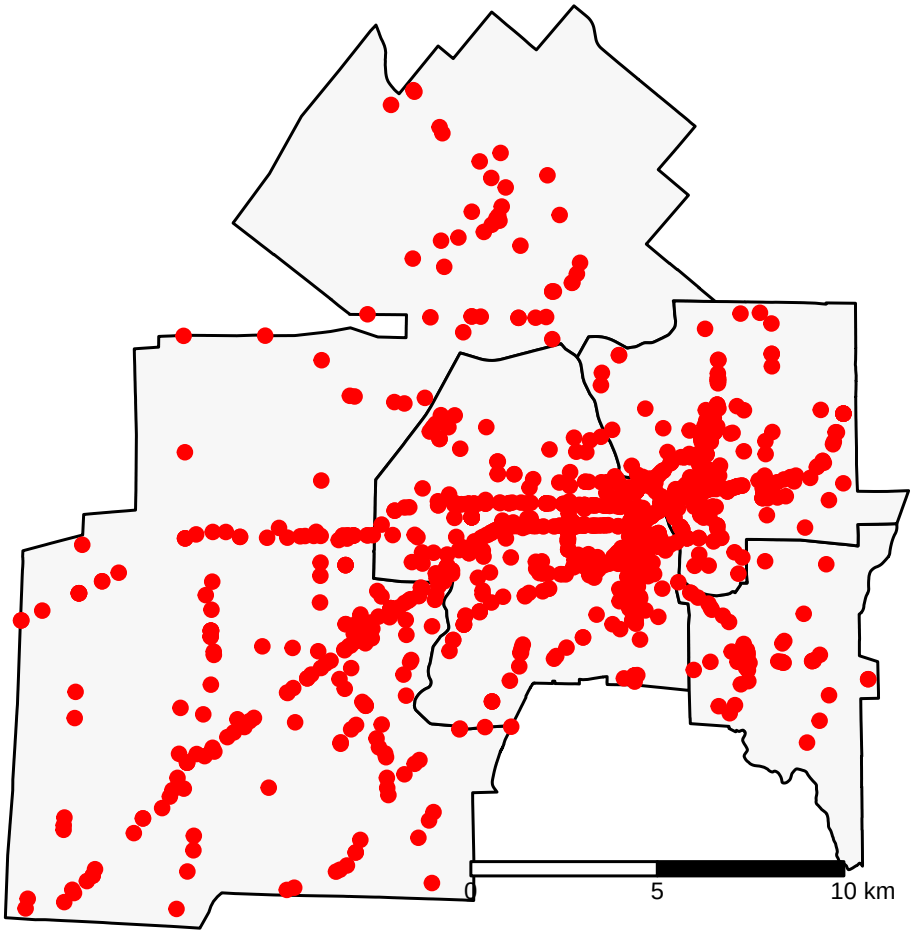


FIGURE 4.8 – Accidents survenus entre juillet 2019 et juin 2022, Ville de Sherbrooke

Dans le code ci-dessous, nous réalisons trois étapes préalables au calcul de DBSCAN :

- Importation des accidents.
- Récupération des coordonnées (x, y) des accidents et stockage dans une matrice.

— Construction du graphique à partir de la distance au quatrième point le plus proche.

Nous n'observons pas de décrochement particulier dans la courbe de la figure 4.9. Par conséquent, nous pourrions tout aussi bien retenir une distance euclidienne de 250, 500, 1000 ou 1500 mètres pour epsilon.

```
library(sf)
library(tmap)
library(dbscan)
library(ggplot2)
## Importation des accidents
Accidents.sf <- st_read(dsn = "data/chap04/DataAccidentsSherb.shp", quiet=TRUE)
## Coordonnées géographiques
xy <- st_coordinates(Accidents.sf)
## Graphique pour la distance au quatrième voisin le plus proche
DistKplusproche <- kNNdist(xy, k = 4)
DistKplusproche <- as.data.frame(sort(DistKplusproche, decreasing = FALSE))
names(DistKplusproche) <- "distance"
ggplot(data = DistKplusproche)+
  geom_path(aes(x = 1:nrow(DistKplusproche), y = distance), size=1)+
  labs(x = "Points triés par ordre croissant selon la distance",
       y = "Distance au quatrième point le plus proche")+
  geom_hline(yintercept=250, color = "#08306b", linetype="dashed", size=1)+
  geom_hline(yintercept=500, color = "#00441b", linetype="dashed", size=1)+
  geom_hline(yintercept=1000, color = "#67000d", linetype="dashed", size=1)+
  geom_hline(yintercept=1500, color = "#3f007d", linetype="dashed", size=1)
```

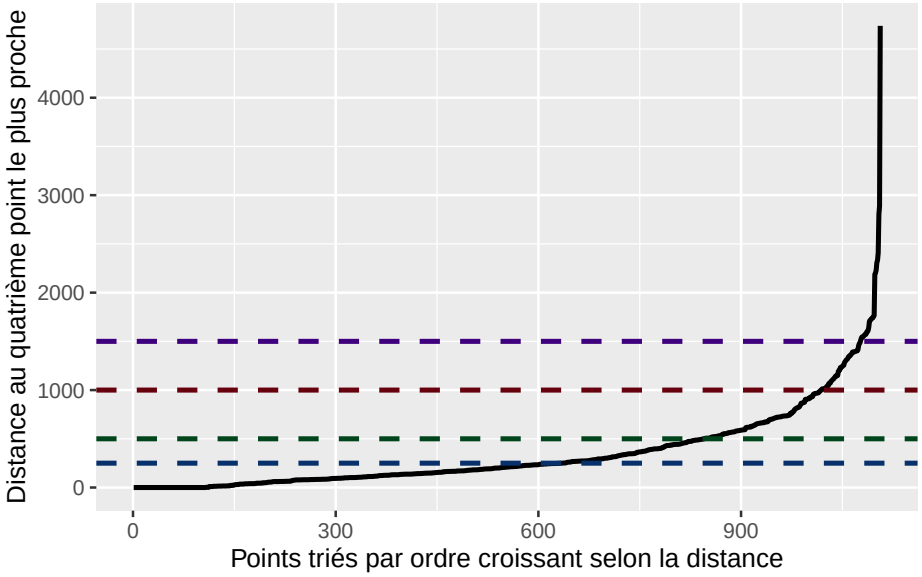


Figure 4.9 – Optimisation de la valeur d'epsilon pour les accidents

Appliquons la méthode DBSCAN avec un minimum de quatre points et les quatre valeurs de distance euclidienne.

```
set.seed(123456789)
## DBSCAN avec les quatre distances
dbscan250  <- dbscan(xy, eps = 250, minPts = 4)
dbscan500  <- dbscan(xy, eps = 500, minPts = 4)
dbscan1000 <- dbscan(xy, eps = 1000, minPts = 4)
dbscan1500 <- dbscan(xy, eps = 1500, minPts = 4)
## Affichage des résultats
dbscan250
```

```
DBSCAN clustering for 1106 objects.
Parameters: eps = 250, minPts = 4
Using euclidean distances and borderpoints = TRUE
The clustering contains 45 cluster(s) and 353 noise points.

  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
353   4   6   7   5  15   5   4  22   7   5   5  19 295   4  18   5   4   7   8
 20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39
  4   6  49   4  11   4   4  41   5   4   8  31  25  10   6  23   4   5  18  15
 40  41  42  43  44  45
  6   4   6   4   7   4

Available fields: cluster, eps, minPts, metric, borderPoints
```

```
dbscan500
```

```
DBSCAN clustering for 1106 objects.
Parameters: eps = 500, minPts = 4
Using euclidean distances and borderpoints = TRUE
The clustering contains 33 cluster(s) and 143 noise points.

  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
143   7   6  14   4   5   5   4   6 734  18   8   6   6   5   9  16   9   5   4
 20  21  22  23  24  25  26  27  28  29  30  31  32  33
 23   5   3   4   6   5   4   9   6   9   4   4   4   6

Available fields: cluster, eps, minPts, metric, borderPoints
```

```
dbscan1000
```

```
DBSCAN clustering for 1106 objects.
Parameters: eps = 1000, minPts = 4
Using euclidean distances and borderpoints = TRUE
The clustering contains 10 cluster(s) and 42 noise points.

  0   1   2   3   4   5   6   7   8   9  10
```

```
 42   8   6  37 962   8   4   6   5  12  16

Available fields: cluster, eps, minPts, metric, borderPoints
```

```
dbscan1500
```

```
DBSCAN clustering for 1106 objects.
Parameters: eps = 1500, minPts = 4
Using euclidean distances and borderpoints = TRUE
The clustering contains 3 cluster(s) and 7 noise points.

   0    1    2    3
   7 1047   12   40

Available fields: cluster, eps, minPts, metric, borderPoints
```

Pour les 1106 accidents du jeu de données, les résultats des quatre DBSCAN ci-dessus sont les suivants :

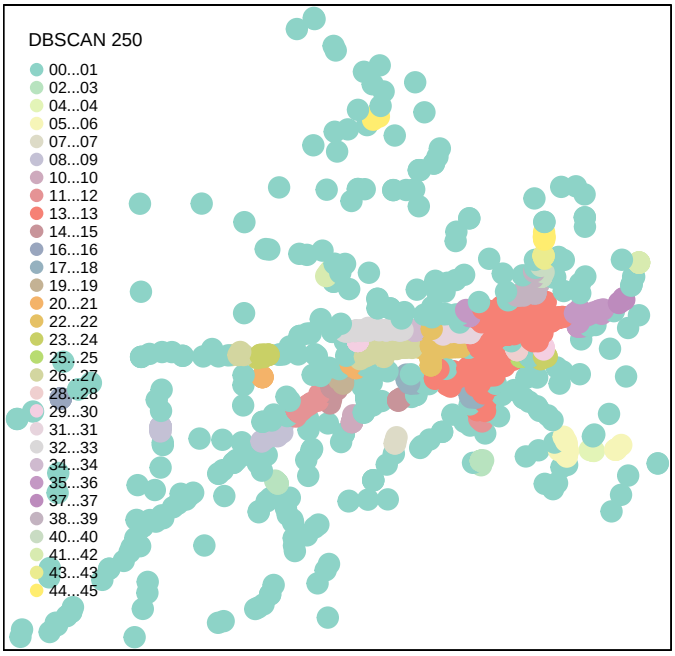
- Avec $\epsilon = 250$, 45 agrégats et 353 points aberrants (bruit).
- Avec $\epsilon = 500$, 33 agrégats et 143 points aberrants.
- Avec $\epsilon = 1000$, 10 agrégats et 42 points aberrants.
- Avec $\epsilon = 1500$, 3 agrégats et 7 points aberrants.

Pour les n points du jeu de données, l'appartenance à un agrégat est enregistrée dans un vecteur numérique avec des valeurs de 0 à k agrégats (`ResultatDbscan$cluster`). Notez que la valeur de 0 est attribuée aux points aberrants. Avec ce vecteur, nous enregistrons les résultats dans un nouveau champ de la couche de points `sf`.

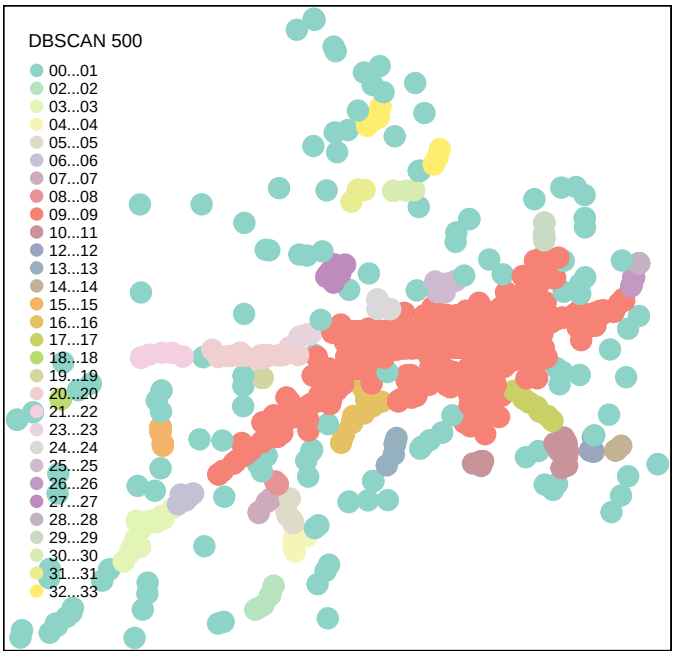
```
## Enregistrement des résultats de DBSCAN dans la couche de points sf
Accidents.sf$Dbscan250  <- as.character(dbscan250$cluster)
Accidents.sf$Dbscan500  <- as.character(dbscan500$cluster)
Accidents.sf$Dbscan1000 <- as.character(dbscan1000$cluster)
Accidents.sf$Dbscan1500 <- as.character(dbscan1500$cluster)

Accidents.sf$Dbscan250 <- ifelse(nchar(Accidents.sf$Dbscan250) == 1,
                                 paste0("0", Accidents.sf$Dbscan250),
                                 Accidents.sf$Dbscan250)
Accidents.sf$Dbscan500 <- ifelse(nchar(Accidents.sf$Dbscan500) == 1,
                                 paste0("0", Accidents.sf$Dbscan500),
                                 Accidents.sf$Dbscan500)
Accidents.sf$Dbscan1000 <- ifelse(nchar(Accidents.sf$Dbscan1000) == 1,
                                 paste0("0", Accidents.sf$Dbscan1000),
                                 Accidents.sf$Dbscan1000)
Accidents.sf$Dbscan1500 <- ifelse(nchar(Accidents.sf$Dbscan1500) == 1,
                                 paste0("0", Accidents.sf$Dbscan1500),
                                 Accidents.sf$Dbscan1500)
```

Nous cartographions finalement les résultats pour les quatre solutions.

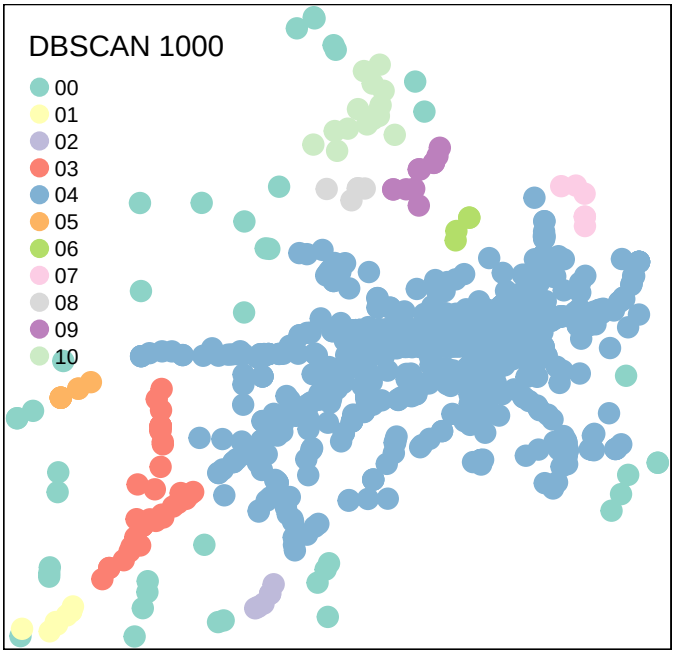
```
tmap_mode("plot")
tm_shape(Accidents.sf)+tm_dots(col="Dbscan250", title = "DBSCAN 250", size = .5)
```



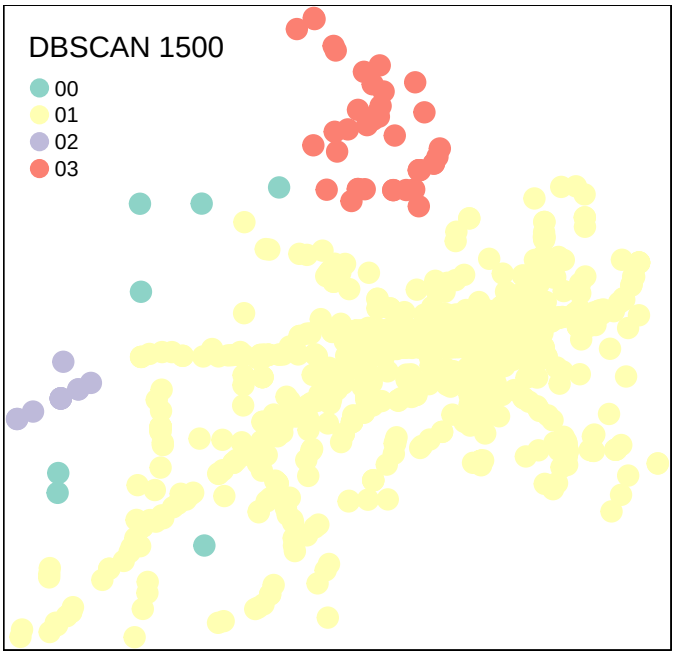

```
tm_shape(Accidents.sf)+tm_dots(col="Dbscan500", title = "DBSCAN 500", size = .5)
```




```
tm_shape(Accidents.sf)+tm_dots(col="Dbscan1000", title = "DBSCAN 1000", size = .5)
```




```
tm_shape(Accidents.sf)+tm_dots(col="Dbscan1500", title = "DBSCAN 1500", size = .5)
```



4.1.3.2 ST-DBSCAN

Pour l'algorithme ST-DBSCAN, nous utilisons une fonction R proposée par Colin Kerouanton.

```
source("code_complementaire/stdbscan.R")
```

Calculons ST-DBSCAN avec une distance spatiale de 1000 mètres et une distance temporelle de 21 jours. Nous obtenons 26 agrégats et 584 points identifiés comme aberrants.

```
## Importation des accidents
Accidents.sf <- st_read(dsn = "data/chap04/DataAccidentsSherb.shp", quiet=TRUE)
## Coordonnées géographiques
xy <- st_coordinates(Accidents.sf)
Accidents.sf$x <- xy[,1]
Accidents.sf$y <- xy[,2]
## Vérifions que le champ DATEINCIDE est bien au format date
str(Accidents.sf$DATEINCIDE)
```

```
 Date[1:1106], format: "2021-12-11" "2022-05-16" "2021-08-12" "2019-08-02" "2020-03-02" ...
```

```
## Calcul de st-dbscan avec une distance de 1000 mètres et 21 jours
Resultats.stdbscan <- stdbscan(x = Accidents.sf$x,
                               y = Accidents.sf$y,
                               time = Accidents.sf$DATEINCIDE,
                               eps1 = 1000,
                               eps2 = 21,
                               minpts = 4)
## Enregistrement des résultats de ST-DBSCAN dans la couche de points sf
Accidents.sf$stdbscan.1000_21 <- as.character(Resultats.stdbscan$cluster)
Accidents.sf$stdbscan.1000_21 <- ifelse(nchar(Accidents.sf$stdbscan.1000_21) == 1,
                                        paste0("0", Accidents.sf$stdbscan.1000_21),
                                        Accidents.sf$stdbscan.1000_21)
## Nombre de points par agrégat
table(Accidents.sf$stdbscan.1000_21)
```

```
 00  01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19
584   4   6   5   4   4   7   7   4 178 156  13  17   4  22  32   6   7   8   8
 20  21  22  23  24  25  26
  5   6   4   3   5   4   3
```

Pour faciliter l'analyse des résultats de ST-DBSCAN, nous conseillons de :

1. Construire un tableau récapitulatif pour les agrégats avec le nombre de points, les dates de début et de fin et l'intervalle temporel.
2. Construire un graphique avec les agrégats (axe des y) et la dimension temporelle (axe des x).
3. Cartographier les résultats.

Le code ci-dessous génère le tableau récapitulatif. Nous constatons ainsi que les agrégats 9 et 10 incluent respectivement 178 et 156 points avec des intervalles temporels importants (respectivement 251 et 319 jours).

```
library(dplyr)
## Sélection des points appartenant à un agrégat
Agregats <- subset(Accidents.sf,
                   Accidents.sf$stdbscan.1000_21 != "00")
## Conversion de la date au format POSIXct
Agregats$dtPOSIXct <- as.POSIXct(Agregats$DATEINCIDE, format = "%Y/%m/%d")
## Tableau récapitulatif
library("dplyr")
Tableau.stdbscan <-
    st_drop_geometry(Agregats) %>%
      group_by(stdbscan.1000_21) %>%
      summarize(points = n(),
                date.min = min(DATEINCIDE),
                date.max = max(DATEINCIDE),
                intervalle.jours = as.numeric(max(DATEINCIDE)-min(DATEINCIDE)))
## Affichage du tableau
print(Tableau.stdbscan, n = nrow(Tableau.stdbscan))
```

```
# A tibble: 26 x 5
   stdbscan.1000_21 points date.min   date.max   intervalle.jours
   <chr>             <int> <date>     <date>                <dbl>
 1 01                    4 2019-08-08 2019-09-04               27
 2 02                    6 2021-12-15 2022-01-25               41
 3 03                    5 2019-07-21 2019-08-30               40
 4 04                    4 2020-11-10 2020-12-12               32
 5 05                    4 2022-01-08 2022-02-13               36
 6 06                    7 2021-06-09 2021-07-02               23
 7 07                    7 2020-06-23 2020-08-07               45
 8 08                    4 2021-09-30 2021-10-27               27
 9 09                  178 2019-07-02 2020-03-09              251
10 10                  156 2021-03-13 2022-01-26              319
11 11                   13 2021-07-24 2021-09-11               49
12 12                   17 2021-10-21 2022-01-12               83
13 13                    4 2021-06-16 2021-07-07               21
14 14                   22 2022-04-11 2022-06-27               77
15 15                   32 2020-09-11 2020-12-18               98
16 16                    6 2020-01-17 2020-02-08               22
17 17                    7 2022-05-07 2022-05-30               23
18 18                    8 2021-04-01 2021-05-27               56
19 19                    8 2020-07-15 2020-09-11               58
20 20                    5 2019-07-05 2019-07-31               26
21 21                    6 2022-01-15 2022-03-02               46
22 22                    4 2020-06-17 2020-06-30               13
```

```
23 23                  3 2022-03-12 2022-03-18              6
24 24                  5 2021-06-29 2021-07-21             22
25 25                  4 2021-09-20 2021-10-25             35
26 26                  3 2021-04-13 2021-04-26             13
```

La figure 4.10 présente les points et l'étendue temporelle de chaque agrégat.

```
## Construction du graphique
ggplot(Agregats) +
  geom_point(aes(x = dtPOSIXct,
                 y = stdbscan.1000_21,
                 color = stdbscan.1000_21),
             show.legend = FALSE) +
  scale_x_datetime(date_labels = "%Y/%m")+
  labs(x= "Temps",
       y= "Identifiant de l'agrégat",
       title = "ST-DBSCAN avec Esp1 = 1000, Esp2 = 21 et MinPts = 4")
```

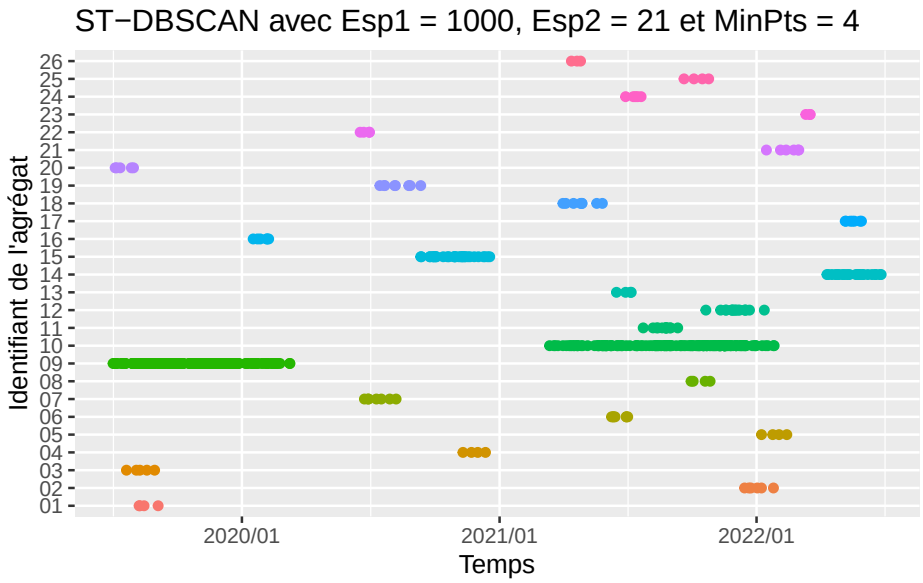


FIGURE 4.10 – Intervalles temporels des agrégats ST-DBSCAN

La cartographie des agrégats est présentée à la figure 4.11 avec en noir les points aberrants.

```
## Création de deux couches : l'une pour les agrégats, l'autre pour les points aberrants
stdbcan.Agregats <- subset(Accidents.sf, Accidents.sf$stdbscan.1000_21 != "00")
stdbcan.Bruit    <- subset(Accidents.sf, Accidents.sf$stdbscan.1000_21 == "00")
## Cartographie
tmap_mode("plot")
tm_shape(Arrondiss)+tm_fill(col="#f7f7f7")+tm_borders(col="black")+
tm_shape(stdbcan.Bruit)+
```

```
  tm_dots(shape = 21, col="black", size=.2)+
  tm_shape(stdbcan.Agregats)+
  tm_dots(shape = 21, col="stdbscan.1000_21", size=.2, title = "Agrégat")+
tm_layout(frame = FALSE, legend.position = c("center", "bottom"),
          legend.text.size = .85, legend.outside = TRUE)
```

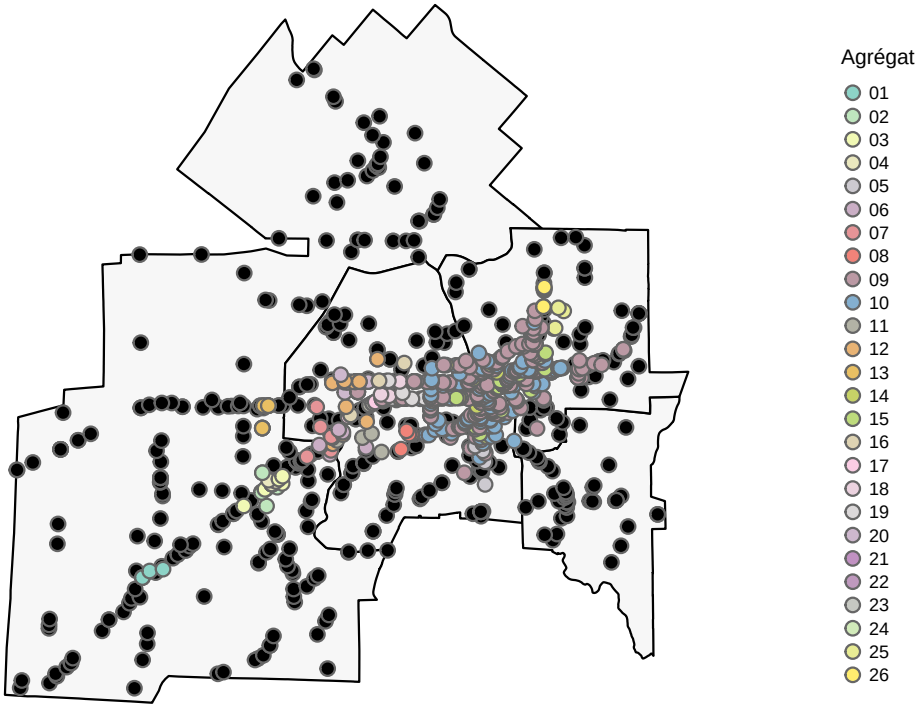


Figure 4.11 – Agrégats identifiés avec ST-DBSCAN

4.2 Méthodes de balayage de Kulldorff

4.2.1 Objectifs de la méthode, types d'analyses, de modèles et d'agrégats

4.2.2 Principes de base de la méthode

4.2.2.1 Type de balayage (cercle ou ellipse)

4.2.2.2 Variable de contrôle

4.2.3 Mise en œuvre dans R

4.2.3.1 Agrégats temporels, spatiaux et spatio-temporels

4.2.3.2 Introduction de variables de contrôle

4.2.3.3 Exploration d'autres types de modèles

4.3 Quiz de révision du chapitre

Questions

- **L'algorithme DBSCAN est basé sur :**
  - La distance entre les points
  - La densité des points

  Relisez au besoin la section 4.1.1.1.
- **Avec l'algorithme DBSCAN, vous devez spécifier le nombre de groupes (agrégats).**
  - Vrai
  - Faux

  Relisez au besoin la section 4.1.1.1.
- **Quels sont les trois types de points identifiés par DBSCAN?**
  - Points centraux
  - Points médians
  - Points frontières
  - Points aberrants

  Relisez au besoin l'encadré à la section 4.1.1.1.
- **Quels sont les deux paramètres de l'algorithme DBSCAN?**
  - Le nombre de points minimum pour identifier les points centraux (MinPts)
  - Le rayon de recherche (epsilon)
  - La distance standard (DS)

  Relisez au besoin la section 4.1.1.1.
- **Plus la valeur d'epsilon est grande, plus le nombre d'agrégats diminue.**

  - Vrai
  - Faux

  Relisez au besoin la section 4.1.1.2.
- **Quels sont les trois paramètres de l'algorithme ST-DBSCAN?**
  - Le nombre de points minimum pour identifier les points centraux (MinPts)
  - Le rayon de recherche pour la proximité spatiale (eps1)
  - Le rayon de recherche pour la proximité temporelle (eps2)
  - La région d'étude

  Relisez au besoin la section 4.1.2.

Réponses

- L'algorithme DBSCAN est basé sur :
  - La densité des points
- Avec l'algorithme DBSCAN, vous devez spécifier le nombre de groupes (agrégats).
  - Faux
- Quels sont les trois types de points identifiés par DBSCAN?
  - Points centraux
  - Points frontières
  - Points aberrants
- Quels sont les deux paramètres de l'algorithme DBSCAN?
  - Le nombre de points minimum pour identifier les points centraux (MinPts)
  - Le rayon de recherche (epsilon)
- Plus la valeur d'epsilon est grande, plus le nombre d'agrégats diminue.
  - Vrai
- Quels sont les trois paramètres de l'algorithme ST-DBSCAN?
  - Le nombre de points minimum pour identifier les points centraux (MinPts)
  - Le rayon de recherche pour la proximité spatiale (eps1)
  - Le rayon de recherche pour la proximité temporelle (eps2)

4.4 Exercices de révision

Exercice

Exercice 1. Application de l'algorithme DBSCAN
L'objectif est d'appliquer cet algorithme sur des accidents impliquant des personnes à vélo sur l'île de Montréal (voir la section 4.1.3.1). Ces données ouvertes sur les collisions routières et leur documentation sont disponibles au lien suivant.

```
library(sf)
library(tmap)
library(dbscan)
library(ggplot2)
## Importation des données
Collissions <- st_read(dsn = "data/chap04/collisions.gpkg",
                       layer = "CollisionsRoutieres",
                       quiet = TRUE)
## Collisions impliquant au moins une personne à vélo en 2020 et 2021
Coll.Velo <- subset(Collissions,
                    Collissions$NB_VICTIMES_VELO > 0 &
                      Collissions$AN %in% c(2020, 2021))
## Coordonnées géographiques
xy <- st_coordinates(Coll.Velo)
## Graphique pour la distance au quatrième voisin le plus proche
DistKplusproche <- kNNdist(xy, k = 4)
DistKplusproche <- as.data.frame(sort(DistKplusproche, decreasing = FALSE))
names(DistKplusproche) <- "distance"
ggplot(à compléter)+
  geom_path(à compléter)+
  labs(à compléter)+
  geom_hline(yintercept=250, color = "#08306b", linetype="dashed", size=1)+
  geom_hline(yintercept=500, color = "#00441b", linetype="dashed", size=1)+
  geom_hline(yintercept=1000, color = "#67000d", linetype="dashed", size=1)
## DBSCAN avec les quatre distances
set.seed(123456789)
dbscan250  <- à compléter
dbscan500  <- à compléter
dbscan1000 <- à compléter
## Affichage des résultats
dbscan250
dbscan500
dbscan1000
## Enregistrement dans la couche de points sf Coll.Velo
Coll.Velo$dbscan250 <- à compléter
Coll.Velo$dbscan500 <- à compléter
Coll.Velo$dbscan1000 <- à compléter

Coll.Velo$dbscan250 <- ifelse(nchar(Coll.Velo$dbscan250) == 1,
                              paste0("0", Coll.Velo$dbscan250),
                              Coll.Velo$dbscan250)
Coll.Velo$dbscan500 <- ifelse(nchar(Coll.Velo$dbscan500) == 1,
                               paste0("0", Coll.Velo$dbscan500),
                               Coll.Velo$dbscan500)
Coll.Velo$dbscan1000 <- ifelse(nchar(Coll.Velo$dbscan1000) == 1,
```

Exercice

Exercice 2. Application de l'algorithme ST-DBSCAN
Avec le même jeu de données, réaliser un ST-DBSCAN avec les paramètres suivants : distance spatiale de 500 mètres, distance temporelle de 30 jours et quatre points minimum (voir la section 4.1.3.2).

```
library(sf)
library(tmap)
library(dbscan)
library(ggplot2)
## Importation des données
Collissions <- st_read(dsn = "data/chap04/collisions.gpkg",
                       layer = "CollisionsRoutieres",
                       quiet = TRUE)
## Collisions impliquant au moins une personne à vélo en 2020 et 2021
Coll.Velo <- subset(Collissions,
                    Collissions$NB_VICTIMES_VELO > 0 &
                      Collissions$AN %in% c(2020, 2021))
## Coordonnées géographiques
xy <- st_coordinates(Coll.Velo)
Coll.Velo$x <- xy[,1]
Coll.Velo$y <- xy[,2]
## Conversion du champ DT_ACCDN au format Date
Coll.Velo$DT_ACCDN <- as.Date(Coll.Velo$DT_ACCDN)
## ST-DBSCAN avec eps1 = 500, esp2 = 30 et minpts = 4
Resultats.stdbscan <- stdbscan(À compléter)
## Enregistrement des résultats ST-DBSCAN dans la couche de points sf
Coll.Velo$stdbscan <- as.character(Resultats.stdbscan$cluster)
Coll.Velo$stdbscan <- ifelse(nchar(Coll.Velo$stdbscan) == 1,
                             paste0("0", Coll.Velo$stdbscan),
                             Coll.Velo$stdbscan)
## Nombre de points par agrégat avec la fonction table
table(Coll.Velo$stdbscan)
## Sélection des points appartenant à un agrégat avec la fonction subset
Agregats <- subset(Coll.Velo, stdbscan != "00")
## Conversion de la date au format POSIXct
Agregats$dtPOSIXct <- as.POSIXct(Agregats$DT_ACCDN, format = "%Y/%m/%d")
## Tableau récapitulatif
library("dplyr")
Tableau.stdbscan <- À compléter
## Affichage du tableau
print(Tableau.stdbscan, n = nrow(Tableau.stdbscan))
## Construction du graphique
À compléter
## Création d'une couche pour les agrégats
stdbcan.Agregats <- subset(Coll.Velo, stdbscan != "00")
## Cartographie
À compléter
```

Correction à la section 12.4.2.

Partie 4. Analyses de données avec des réseaux de transport

5 Mesures d'accessibilité spatiale selon différents modes de transport

Dans ce chapitre, nous voyons comment construire un réseau multimode (voiture, marche, vélo, transport en commun) pour calculer différentes mesures d'accessibilité dans R. Aussi, nous discutons de la notion d'accessibilité à un service, notamment des cinq dimensions identifiées par Penchansky et Thomas (1981), de l'accessibilité réelle versus l'accessibilité potentielle et de l'accessibilité spatiale versus l'accessibilité aspatiale (Luo et Wang 2003).

Package

Liste des *packages* utilisés dans ce chapitre

- Pour importer et manipuler des fichiers géographiques :
  - `sf` pour importer et manipuler des données vectorielles.
  - `terra` pour importer et manipuler des données matricielles.
- Pour construire des cartes et des graphiques :
  - `tmap` est certainement le meilleur *package* pour la cartographie.
  - `ggplot2` pour construire des graphiques.
- Pour construire un réseau :
  - `osmextract` pour extraire des fichiers OpenStreetMap.
  - `gtfstools` pour valider la structure d'un fichier GTFS.
  - `R5R` pour calculer des trajets et des matrices origines-destinations selon différents modes de transport.
- Pour manipuler les données :
  - `dplyr` pour calculer des moyennes pondérées.

5.1 Notions relatives à l'analyse de réseau

5.1.1 Définition d'un réseau

Un réseau est un ensemble de lignes connectées par des nœuds – voies ferrées, voies routières, canalisations d'eau ou de gaz, fleuves et affluents drainant une région, etc. – reliant un territoire (figure 5.1). Pour un réseau routier, l'information sémantique rattachée tant aux lignes (sens de circulation, vitesse autorisée, rues piétonnières, pistes cyclables, etc.) qu'aux nœuds (types d'intersection, autorisation de virage à gauche, etc.) est utilisée pour modéliser un réseau.

5.1.2 Principaux problèmes résolus en analyse de réseau

L'analyse de réseau permet de résoudre trois principaux problèmes (figure 5.2) :

- Trouver le trajet le plus court ou le plus rapide entre deux points, basé sur l'algorithme de Dijkstra (1959).

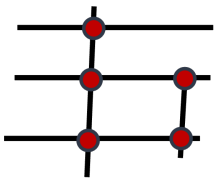


FIGURE 5.1 – Réseau : un ensemble de lignes connectées par des nœuds

— Trouver la route optimale comprenant plusieurs arrêts (problème du voyageur de commerce).
— Définir des zones de desserte autour d'une origine, basé aussi sur l'algorithme de Dijkstra.

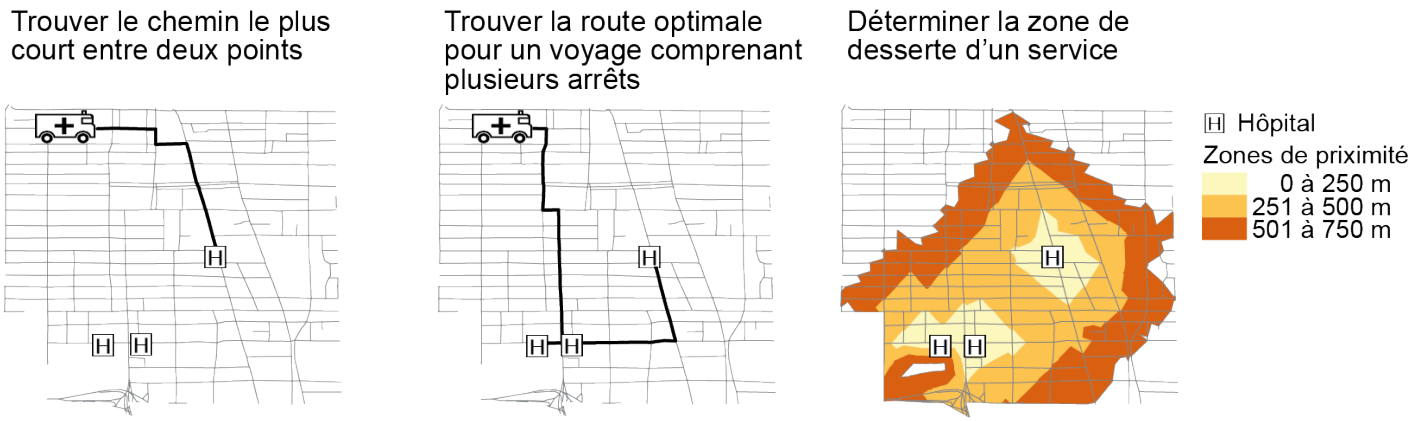


FIGURE 5.2 – Trois principaux problèmes résolus en analyse de réseau

À cela s'ajoutent quatre autres problèmes :

— Trouver les k services les plus proches à partir d'une origine (figure 5.3, a).
— Construire une matrice de distance origines-destinations (figure 5.3, b), dont l'intérêt principal est de permettre par la suite le calcul de n'importe quelle mesure d'accessibilité.
— Résoudre le problème de tournées de véhicules dont l'objectif est d'optimiser les tournées d'une flotte de véhicules en fonction des ressources disponibles, de la localisation des clients, des lieux de dépôt de marchandises, etc.
— Réaliser un modèle localisation-affectation dont l'objectif est d'optimiser la localisation d'un ou plusieurs nouveaux équipements en fonction de la demande, et ce, en minimisant la distance agrégée entre les points d'offre et de demande. Par exemple, une région ayant 15 hôpitaux desservant deux millions d'habitants souhaiterait ajouter trois autres établissements. En fonction de l'offre existante (hôpitaux pondérés par le nombre de lits), de la distribution spatiale de la population et de la localisation des sites potentiels des trois hôpitaux, il s'agit de choisir ceux qui minimisent la distance entre les points d'offre et de demande.

Ces problèmes peuvent être résolus selon différents modes de transport, à savoir le chemin le plus rapide en véhicule motorisé, à pied, en vélo et en transport en commun (figure 5.4).

5.1.3 Analyse de réseau et entités polygonales

Vous avez compris que les problèmes présentés plus haut sont réalisés à partir d'entités spatiales ponctuelles. Pour estimer le trajet le plus court ou le plus rapide entre un point et un polygone (un parc urbain par exemple), il faut préalablement le convertir en point. Plusieurs solutions sont envisageables (Apparicio et Séguin 2006) :

(a) Trouver les *k* services les plus proches **(b)** Construire une matrice de distance origines-destinations

FIGURE 5.3 – Autres problèmes résolus en analyse de réseau

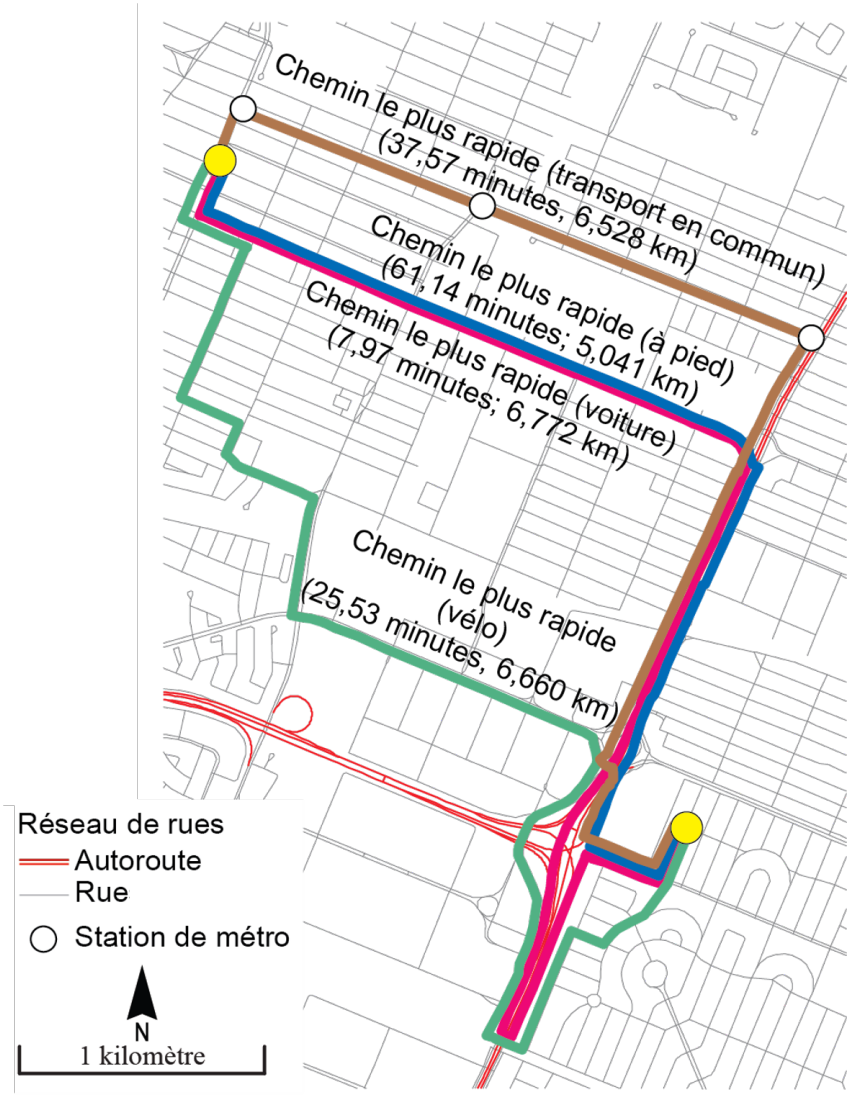


FIGURE 5.4 – Chemin le plus rapide selon différents modes de transport

- Calculer le trajet entre le point et le centroïde du parc (figure 5.5, a). Le centroïde est alors rattaché au tronçon de rue le plus proche. Cette solution est peu précise : plus la superficie du polygone est grande, plus l'imprécision augmente.
- Si le parc a plusieurs entrées, il suffit de les positionner le long du périmètre et de calculer les trajets à partir de ces points.
- Si le parc n'a pas d'entrée, il convient de positionner des points le long du périmètre, espacés d'une distance prédéterminée (figure 5.5, b). Bien qu'elle soit longue à calculer, cette solution est bien plus précise. Par exemple, avec des points espacés de 20 mètres le long du périmètre du parc, l'erreur maximale est de 10 mètres.

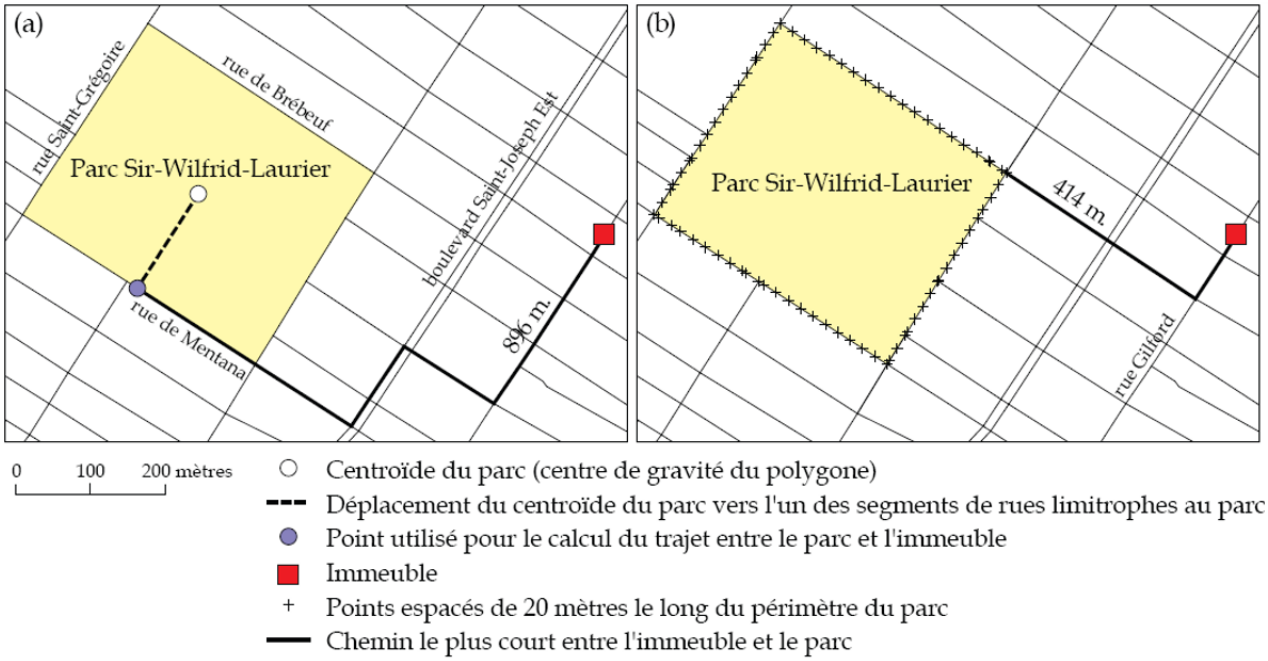


FIGURE 5.5 – Méthode pour déterminer le trajet le plus court entre une entité ponctuelle et une entité polygonale

Aller plus loin

Modélisation d'un réseau dans un logiciel SIG (systèmes d'information géographique)
Il est aussi possible de construire un réseau dans un logiciel SIG (QGIS et ArcGIS Pro par exemple). Pour une description détaillée de la construction d'un réseau selon différents modes de transport (voiture, marche, vélo et transport en commun) dans un SIG, consultez l'article d'Apparicio *et al.* (2017).

5.2 Construction d'un réseau avec R5R

Pour construire un réseau pour différents modes de transport dans R, nous utilisons le *package* `R5R` (Pereira et al. 2021). Il existe d'autres *packages*, notamment `opentripplanner` (Morgan et al. 2019) qui a été largement utilisé ces dernières années. Étant plus rapide, `R5R` s'impose actuellement comme la solution la plus efficace pour calculer des trajets à travers un réseau de rues selon différents modes de transport.

Note

Documentation du *package* `R5R`
Nous vous conseillons vivement de lire la documentation de `R5R` sur le site de CRAN, notamment les nombreuses vignettes présentant des exemples d'analyses avec du code R très bien documenté.

5.2.1 Extraction des données spatiales pour R5R

Pour construire un réseau multimode, `R5R` a besoin de trois fichiers qui doivent être localisés dans le même dossier (figure 5.6) :

1. un fichier *pbf* (*Protocolbuffer Binary Format*) pour les données d'OpenStreetMap.
2. un ou plusieurs fichiers *GTFS* (*General Transit Feed Specification*) pour les flux relatifs aux transports en commun.
3. un fichier *GeoTIFF* d'élévation.

Notez que ce dernier fichier est optionnel. Toutefois, pour calculer des trajets à pied ou à vélo, il est préférable de tenir compte de la pente, surtout dans une ville comme Sherbrooke!

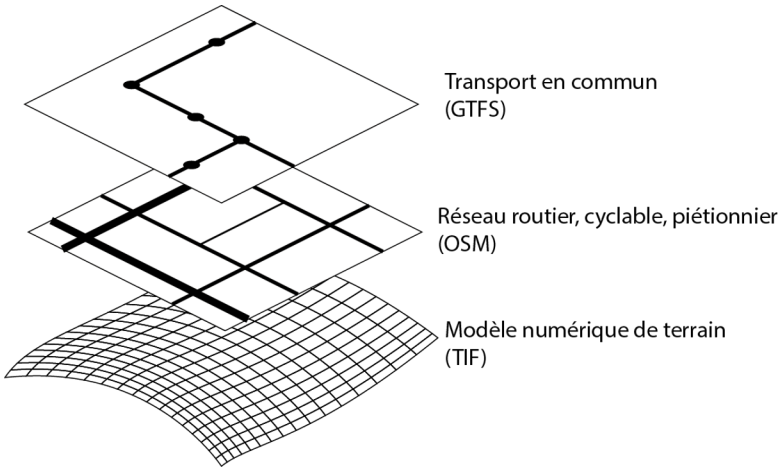


FIGURE 5.6 – Trois types de données nécessaires pour modéliser un réseau dans R5R

5.2.1.1 Extraction d'un fichier OpenStreetMap

Pour récupérer un fichier OpenStreetMap (OSM, format *pbf*), nous utilisons deux fonctions du *package* `osmextract` (Gilardi et Lovelace 2022) :

— `oe_match` pour repérer le fichier OSM pour la région de l'Estrie.
— `oe_download` pour télécharger le fichier OSM pour la région de l'Estrie.

```
library(osmextract)
## Identification du fichier OSM (format pbf) pour l'Estrie
Estrie = oe_match(place="Estrie", provider = "openstreetmap_fr")
## Téléchargement du fichier OSM (format pbf) pour l'Estrie
```

```
oe_download(
  file_url = Estrie$url,
  file_size = Estrie$file_size,
  provider = "openstreetmap_fr",
  download_directory = "data/chap05/__TempOSM")
```

Le fichier OSM téléchargé plus haut couvre la région de l'Estrie. À notre connaissance, il n'existe pas de solution pour découper un fichier *pbf* dans R. Par conséquent, nous récupérons les coordonnées minimales et maximales de l'enveloppe d'une zone tampon de 5000 mètres autour de la couche de la ville de Sherbrooke. Puis, pour découper le fichier *pbf*, nous utilisons l'outil **osmconvert64-0.8.8p.exe**. Le code ci-dessous renvoie les quatre coordonnées de l'enveloppe.

```
library(sf)
library(tmap)
## Importation du polygone pour la ville de Sherbrooke avec sf
Sherbrooke <- st_read(dsn = "data/chap05/AutresDonnees/Sherbrooke.shp", quiet=TRUE)
## Création d'une zone tampon de 5 km
zone <- st_buffer(Sherbrooke, dist = 5000)
## Changement de la projection en 4326
zone <- st_transform(zone, 4326)
## Création de l'enveloppe autour de la couche
enveloppe = st_bbox(zone)
## Visualisation des coordonnées minimales et maximales
cat(paste0("Minimum longitude : ", round(enveloppe[1],4),
           "\n Minimum latitude : ", round(enveloppe[2],4),
           "\n Maximum longitude : ", round(enveloppe[3],4),
           "\n Maximum latitude : ", round(enveloppe[4],4)
    ))
```

```
Minimum longitude : -72.1537
 Minimum latitude : 45.2683
 Maximum longitude : -71.7576
 Maximum latitude : 45.5554
```

L'application **Osmconvert** est disponible pour Windows et Linux. Pour découper un fichier avec les coordonnées latitude/longitude minimales et maximales avec **Osmconvert**, nous écrivons une commande système et l'exécutons avec la fonction `system`. Pour utiliser le code ci-dessous, vous devez avoir préalablement téléchargé **Osmconvert** et connaitre son emplacement.

```
## Préparation des chemins
path_to_osm_convert <- paste0(getwd(),'/data/chap05/__TempOSM/osmconvert64-0.8.8p.exe')
path_to_big_osm <- paste0(getwd(),'/data/chap05/__TempOSM/openstreetmap_fr_estrie-latest.osm.pbf')
path_to_small_osm <- paste0(getwd(),'/data/chap05/__TempOSM/openstreetmap_fr_estrie-latest.osm_01.pbf')
## Écriture de la commande
commande <- paste0('"',path_to_osm_convert, '" "',
                   path_to_big_osm, '" -b=',paste0(enveloppe[1:4], collapse = ','),
                   ' -o="',path_to_small_osm,'"')
```

Une fois que la commande est écrite, nous l'exécutons.

```
system(commande)
```

Notez qu'il est possible d'obtenir le même résultat avec l'application **Osmosis**. Il s'agit d'un outil rédigé en **Java** et qui dispose de fonctionnalités similaires à celles de **Osmconvert**. Puisqu'il est rédigé en **Java**, il peut être utilisé sur davantage de plateformes qu'**Osmconvert**.

5.2.1.2 Construction d'un fichier GeoTIFF pour l'élévation

Pour créer un fichier *GeoTIFF* pour l'élévation, nous utilisons les modèles numériques de terrain (MNT) du ministère des Ressources naturelles et des Forêts.

```
library(terra)
## Couche polygonale sf pour la ville de Sherbrooke
# Shapefile pour les régions du Québec au 1/20000
# https://www.donneesquebec.ca/recherche/dataset/decoupages-administratifs/resource/b368d470-71d6-40a2-8457-e441
Sherbrooke <- st_read(dsn = "data/chap05/AutresDonnees/Sherbrooke.shp", quiet=TRUE)
Sherbrooke <- st_buffer(Sherbrooke, dist = 5000)
## Feuillets pour les MNT au 1/20000
## Importation du shapefile des feuillets
## (https://www.donneesquebec.ca/recherche/dataset/modeles-numeriques-d-altitude-a-l-echelle-de-1-20-000/resourc
Feuillets <- st_read(dsn = "data/chap05/AutresDonnees/Index_MNT20k.shp", quiet=TRUE)
### Nous nous assurons que les deux couches ont la même projection, soit EPSG 4326
Sherbrooke <- st_transform(Sherbrooke, st_crs(Feuillets))
### Sélection des feuillets qui intersectent le polygone de l'Estrie
RequeteSpatiale <- st_intersects(Feuillets, Sherbrooke, sparse = FALSE)
Feuillets$Intersect <-  RequeteSpatiale[, 1]
FeuilletsSherbrooke <- subset(Feuillets,Intersect == TRUE)

## Téléchargement des GRIDS
### Création d'un dossier temporaire pour les MNT
dir.create(paste0("data/chap05/AutresDonnees/MNT"))
grids <- FeuilletsSherbrooke$GRID
i = 0
for (e in grids) {
  i = i+1
  # Téléchargement
  Fichier <- substr(e, 88, nchar(e))
  Chemin <-  "data/chap05/AutresDonnees/MNT"
  CheminFichier <- paste0(Chemin, "/", Fichier)
  download.file(e, destfile = CheminFichier)
  # Décompression du fichier zip
  unzip(CheminFichier, exdir = Chemin)
  # suppression du zip
```

```
  unlink(CheminFichier)
}

## Importation des GRIDS avec le package Terra
Fichier1 <- substr(grids, 88, 98)
Fichier2 <- paste0(substr(tolower(Fichier1), 1, 7),
                   substr(Fichier1, 9, 11))
NomsFichiers <- paste0("data/chap05/AutresDonnees/MNT/",
                       Fichier1, "/",
                       Fichier2)
rlist <- list()
for(e in NomsFichiers) {
 print(e)
 rasterGrid <- terra::rast(e)
 rlist[[length(rlist)+1]] <- rasterGrid
}
## Création d'une mosaique avec les GRIDS
rsrc <- terra::sprc(rlist)
MosaicSherbrooke <- mosaic(rsrc)
MosaicSherbrooke

# Pour réduire la taille du fichier d'élévation, nous arrondissons les valeurs au mètre
MosaicSherbrooke <- round(MosaicSherbrooke)

## Exporter en GeoTIFF
terra::writeRaster(MosaicSherbrooke,
                   "data/chap05/_DataReseau/Elevation.tif",
                   filetype = "GTiff",
                   datatype = 'INT2U',
                   overwrite = TRUE)

## Suppression des GRIDS
dossier <- "data/chap05/AutresDonnees/MNT/"
f <- list.files(dossier, include.dirs = FALSE, full.names = TRUE, recursive = TRUE)
file.remove(f)
d <- list.dirs(dossier)
unlink(d, recursive = TRUE)
```

Le fichier d'élévation ainsi construit est présenté à la figure 5.7.

```
MosaicSherbrooke <- terra::rast("data/chap05/_DataReseau/Elevation.tif")
terra::plot(MosaicSherbrooke)
```

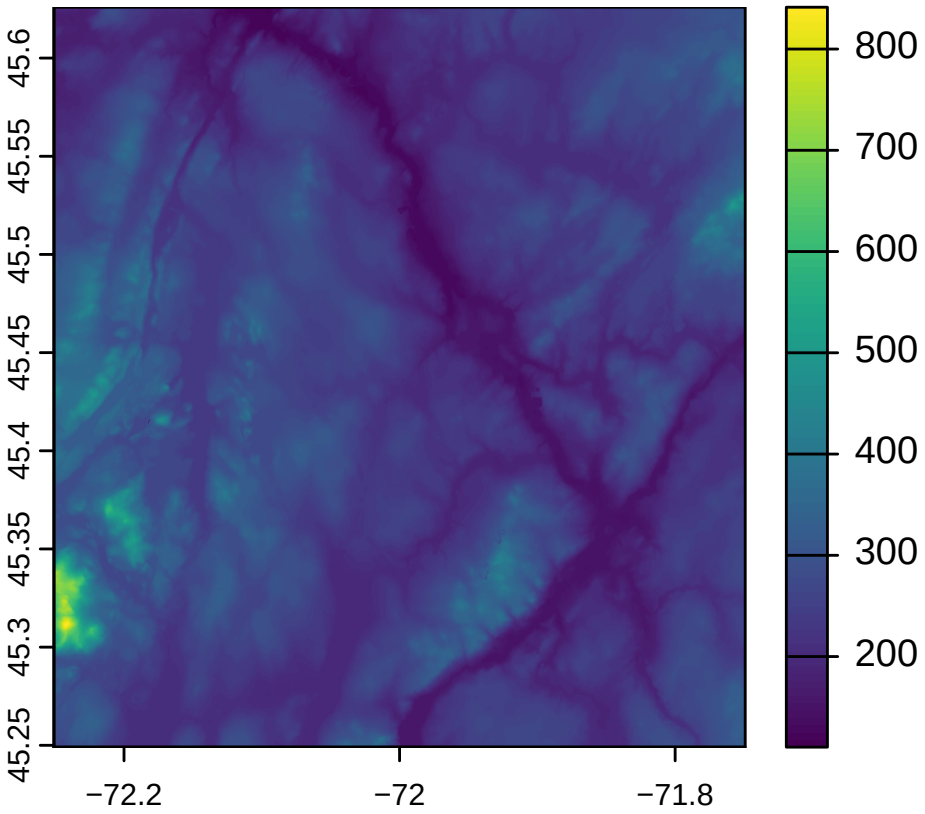


FIGURE 5.7 – Modèle numérique d'élévation au 1/20000 pour la région de Sherbrooke

Aller plus loin

***Package* `elevatr` pour extraire des images d'élévation pour une région donnée**
Plus haut, nous avons utilisé des modèles numériques de terrain (MNT) du ministère des Ressources naturelles et des Forêts pour construire un fichier *GeoTIFF* pour l'élévation pour la ville de Sherbrooke (Québec). Pour d'autres régions du monde, vous pouvez aussi utiliser le *package* `elevatr` (Hollister et al. 2023) pour extraire une image d'élévation à partir de l'API OpenTopography.
Pour plus d'information, consultez cette vignette,

5.2.1.3 Extraction et validation d'un fichier GTFS

Le fichier GTFS pour la Société de Transport de Sherbrooke est disponible à l'adresse suivante : https://gtfs.sts.qc.ca:8443/gtfs/client/GTFS_clients.zip. Pour le télécharger, nous utilisons la fonction `download.file`.

```
url <- "https://gtfs.sts.qc.ca:8443/gtfs/client/GTFS_clients.zip"
destfile <- "data/chap05/_DataReseau/GTFS_clients.zip"
download.file(url, destfile)
```

Pour s'assurer du bon fonctionnement d'un GTFS dans `r5r`, il faut préalablement valider la structure du fichier. Dans un premier temps, vous pouvez valider la structure générale de votre GTFS en utilisant l'outil en ligne gtfs-validator

proposé par l'organisation *MobilityData*. Dans un second temps, il convient de s'assurer que les types de lignes utilisés font partie des types standards définis par Google, soit avec l'un des numéros suivants : 0, 1, 2, 3, 4, 5, 6, 7, 11, 12. Si votre GTFS contient des lignes de transport en commun provenant par exemple des types étendus, `r5r` renverra une erreur.

Nous vous proposons ci-dessous une fonction qui remplace les types problématiques dans un GTFS. Le plus simple est de les remplacer par un type *bus* (3), ce qui n'affectera pas les trajets estimés.

```
library(gtfstools)
## Fonction
clean_gtfs_types <- function(gtfs_file, replace_by = 3){
  # Lecture du GTFS
  seed <- gtfstools::read_gtfs(gtfs_file)
  print(unique(seed$routes$route_type))
  # Vérification des types de routes
  seed$routes$route_type <- ifelse(
    seed$routes$route_type %in% c(0:7,11,12),
    seed$routes$route_type,
    replace_by
  )
  # Réécriture du fichier GTFS avec la modification
  gtfstools::write_gtfs(seed, gtfs_file)
}
## Appel de la fonction
destfile <- "data/chap05/_DataReseau/GTFS_clients.zip"
clean_gtfs_types(destfile)
```

```
[1] 3
```

Pour ce fichier, le seul type de ligne utilisé est le numéro 3, soit des lignes de bus.

5.2.2 Construction du réseau avec R5R

⚠ **Attention**

`R5R` et JDK Java et allocation de la mémoire
Le *package* `R5R` utilise la version 21 de la JDK de Java (*Java Development Kit*). Vous devez préalablement la télécharger et l'installer sur votre ordinateur. Les deux lignes de code ci-dessous permettent de vérifier si elle est bien installée sur votre ordinateur.

```
## Vérification que la JDK version 21 est bien installée
rJava::.jinit()
rJava::.jcall("java.lang.System", "S", "getProperty", "java.version")
```

```
[1] "21.0.3"
```

Il est fortement conseillé d'augmenter la mémoire allouée au processus JAVA, surtout si vous souhaitez calculer des matrices origines-destinations de grande taille. Par exemple, la commande `options(java.parameters = "-Xmx2G")` permet d'augmenter la mémoire disponible pour JAVA à deux gigaoctets. Si votre ordinateur ne manque pas de mémoire vive (16, 32 ou 64 gigaoctets), n'hésitez pas à augmenter ce paramètre (par exemple, `options(java.parameters = "-Xmx8G")`).
Si vous avez plusieurs version de la jdk présentent sur votre ordinateur, vous pouvez aussi indiquer **en tout début de script, avant de charger le moindre package** laquelle utiliser avec l'option suivante:

```
Sys.setenv(JAVA_HOME='C:/Program Files/Java/jdk-21')
```

Notez que vous devez bien sûr adapter le chemin vers votre propre installation de la jdk-21.

Nous utilisons la fonction `setup_r5` pour construire un réseau multimode à partir des trois fichiers :

- Le fichier OSM (`openstreetmap_fr_sherbrooke.pbf`).
- Le fichier GTFS pour la Société de Transport de Sherbrooke (`GTFS_clients.zip`).
- Le fichier d'élévation pour la région de Sherbrooke (`Elevation.tif`).

Notez les paramètres suivants pour la fonction `setup_r5` :

- `data_path` pour définir le dossier dans lequel sont présents les trois fichiers.
- `elevation = "TOBLER"` pour utiliser la fonction d'impédance de Tobler pour la marche et le vélo qui prend en compte la pente.
- `overwrite = FALSE` pour ne pas écraser le réseau s'il est déjà construit. La construction d'un réseau peut être très longue dépendamment de la taille des trois fichiers (OSM, GTFS, élévation). Par conséquent, n'oubliez pas de spécifier cette option si votre réseau a déjà été construit.

```
library(r5r)
## Allocation de la mémoire pour Java
options(java.parameters = "-Xmx2G")
## Construction du réseau R5R
r5r_core <- setup_r5(data_path = "data/chap05/_DataReseau/",
                     elevation = "TOBLER",
                     verbose = FALSE, overwrite = FALSE)
```

La fonction `setup_r5` a créé deux nouveaux fichiers (*network.dat* et *network_settings.json*) qui sont utilisés par le *package* `r5r` pour les analyses de réseau.

```
## Liste des fichiers dans le dossier
list.files("data/chap05/_DataReseau/")
```

```
[1] "Elevation.tif"
[2] "GTFS_clients.zip"
[3] "network.dat"
[4] "network_settings.json"
[5] "openstreetmap_fr_estrie-latest.osm_01.pbf"
[6] "openstreetmap_fr_estrie-latest.osm_01.pbf.mapdb"
[7] "openstreetmap_fr_estrie-latest.osm_01.pbf.mapdb.p"
```

5.2.3 Calcul d'itinéraires avec R5R selon le mode de transport

Pour calculer un trajet, nous utilisons la fonction `detailed_itineraries` dont les paramètres sont décrits dans l'encadré ci-dessous.

Note

Paramètres de la fonction `detailed_itineraries`

- **`r5r_core`**: le réseau créé avec la fonction `setup_r5()` décrite plus haut.
- **`origins`**: un point `sf` projeté en WGS84 ou un `data.frame` comprenant les colonnes `id`, `lon` et `lat`.
- **`destinations`**: un point `sf` projeté en WGS84 ou un `data.frame` comprenant les colonnes `id`, `lon` et `lat`.
- **`mode`**: un vecteur de type caractères définissant les modes de transport dont les principaux sont :
  - `"WALK"` pour la marche.
  - `"BICYCLE"` pour le vélo.
  - `"CAR"` pour l'automobile.
  - `"c("WALK",TRANSIT")` pour la marche et le transport en commun.
- **`departure_datetime`**: un objet `POSIXct` définissant la date et l'heure de départ à utiliser si vous souhaitez calculer un trajet en transport en commun.
- **`max_walk_time=inf`**: un nombre entier définissant le temps de marche maximal en minutes pour chaque segment du trajet. La valeur par défaut est sans limite (`Inf`).
- **`max_bike_time = Inf`**: un nombre entier définissant le temps maximal à vélo en minutes.
- **`max_car_time = Inf`**: un nombre entier définissant le temps maximal en automobile en minutes.
- **`max_trip_duration = 120`**: un nombre entier définissant le temps maximal du trajet qui est fixé à 120 minutes par défaut. Par conséquent, tout trajet d'une durée supérieure à ce seuil ne sera pas calculé.
- **`walk_speed = 3.6`**: une valeur numérique définissant la vitesse moyenne de marche qui est fixée par défaut à 3,6 km/h. Ce seuil est très conservateur et pourrait être augmenté à 4,5 km/h.
- **`bike_speed = 12`**: une valeur numérique définissant la vitesse moyenne à vélo qui est fixée par défaut à 12 km/h. Ce seuil est aussi très conservateur et pourrait être augmenté à 15 ou 16 km/h.
- **`max_lts = 2`**: un nombre entier de 1 à 4 qui indique le niveau de stress lié à la circulation que les cyclistes sont prêts à tolérer. Une valeur de 1 signifie que les cyclistes n'emprunteront que les rues les plus calmes, tandis qu'une valeur de 4 indique que les cyclistes peuvent emprunter n'importe quelle route.
  - `max_lts = 1`: tolérable pour les enfants.
  - `max_lts = 2`: tolérable pour la population adulte en général.
  - `max_lts = 3`: tolérable pour les cyclistes enthousiastes et confiants.
  - `max_lts = 4`: tolérable uniquement pour les cyclistes intrépides.
- **`drop_geometry = FALSE`**: si ce paramètre est fixé à TRUE, la géométrie du trajet ne sera pas incluse.

5.2.3.1 Calcul d'itinéraires selon les modes de transport actif

Dans un premier temps, nous calculons des trajets à vélo entre les deux localisations suivantes :

- Le point `Pt.W` situé à l'intersection de la rue Wellington et de la Côte de l'Acadie (45,38947; -71,88393).
- Le point `Pt.D` situé à l'intersection des rues Darche et Dorval (45,38353; -71,89169).

```
## Création d'une couche sf avec les deux points
Pts <- data.frame(id = c("Rue Wellington S.", "Rue Darche"),
                          lat = c( 45.38947,  45.38353),
                          lon = c(-71.88393, -71.89169)
                          )
Pts <- st_as_sf(Pts, coords = c("lon","lat"), crs = 4326)
```

```
Pt.W <- Pts[1,]
Pt.D <- Pts[2,]
```

Avec la fonction `detailed_itineraries`, les durées sont estimées à respectivement six et huit minutes (figure 5.8). Bien que le chemin emprunté soit le même, cet écart s'explique par la Côte de l'Acadie, soit l'un des tronçons de rue les plus pentus de la ville de Sherbrooke.

```
## Trajets en vélo
velo.1 <- detailed_itineraries(r5r_core = r5r_core,
                               origins = Pt.W,
                               destinations = Pt.D,
                               mode = "BICYCLE", # Vélo
                               bike_speed = 12,
                               shortest_path = FALSE,
                               drop_geometry = FALSE)
velo.2 <- detailed_itineraries(r5r_core = r5r_core,
                               origins = Pt.D,
                               destinations = Pt.W,
                               mode = "BICYCLE", # Vélo
                               bike_speed = 12,
                               shortest_path = FALSE,
                               drop_geometry = FALSE)
velo.1
```

```
Simple feature collection with 1 feature and 16 fields
Geometry type: LINESTRING
Dimension:     XY
Bounding box:  xmin: -71.89191 ymin: 45.3835 xmax: -71.88387 ymax: 45.38955
Geodetic CRS:  WGS 84
            from_id from_lat  from_lon      to_id   to_lat    to_lon option
1 Rue Wellington S. 45.38947 -71.88393 Rue Darche 45.38353 -71.89169      1
  departure_time total_duration total_distance segment    mode segment_duration
1       20:48:39            8.1           1215       1 BICYCLE              8.1
  wait distance route                       geometry
1    0     1215       LINESTRING (-71.88396 45.38...
```

```
velo.2
```

```
Simple feature collection with 1 feature and 16 fields
Geometry type: LINESTRING
Dimension:     XY
Bounding box:  xmin: -71.89191 ymin: 45.3835 xmax: -71.88387 ymax: 45.38955
Geodetic CRS:  WGS 84
     from_id from_lat  from_lon             to_id   to_lat    to_lon option
1 Rue Darche 45.38353 -71.89169 Rue Wellington S. 45.38947 -71.88393      1
```

```
  departure_time total_duration total_distance segment    mode segment_duration
1       20:48:40            6.5           1215       1 BICYCLE              6.5
  wait distance route                        geometry
1    0     1215       LINESTRING (-71.89191 45.38...
```

Pour visualiser les trajets, nous utilisons le *package* `tmap` en mode `view`. Notez qu'un clic sur le trajet fait apparaître une fenêtre surgissante.

```
library(tmap)
# Cartographie des trajets avec tmap
tmap_mode("view")
Carte1 <- tm_shape(velo.1)+
  tm_lines(col="black", lwd = 2,
           popup.vars = c("mode", "from_id", "to_id", "segment_duration", "distance"))+
  tm_shape(Pt.W)+tm_dots(col="green", size = .15)+
  tm_shape(Pt.D)+tm_dots(col="red", size = .15)
Carte2 <- tm_shape(velo.2)+
  tm_lines(col="black", lwd = 2,
           popup.vars = c("mode", "from_id", "to_id", "segment_duration", "distance"))+
  tm_shape(Pt.D)+tm_dots(col="green", size = .15)+
  tm_shape(Pt.W)+tm_dots(col="red", size = .15)
tmap_arrange(Carte1, Carte2, ncol = 2)
```

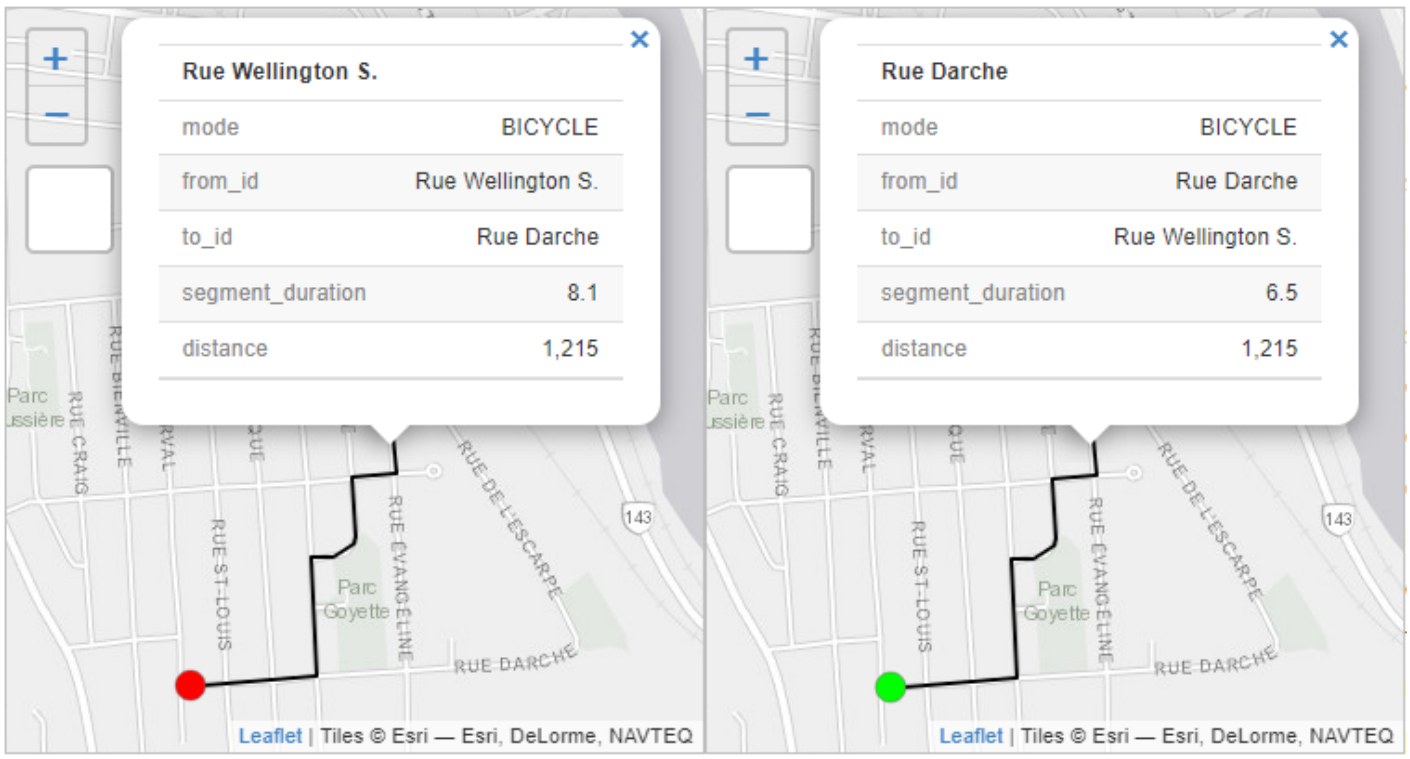


Figure 5.8 – Trajets à vélo entre les deux destinations

Dans un second temps, nous calculons les trajets à pied avec les deux mêmes localisations qui sont estimées à 16 et 21 minutes (figure 5.9).

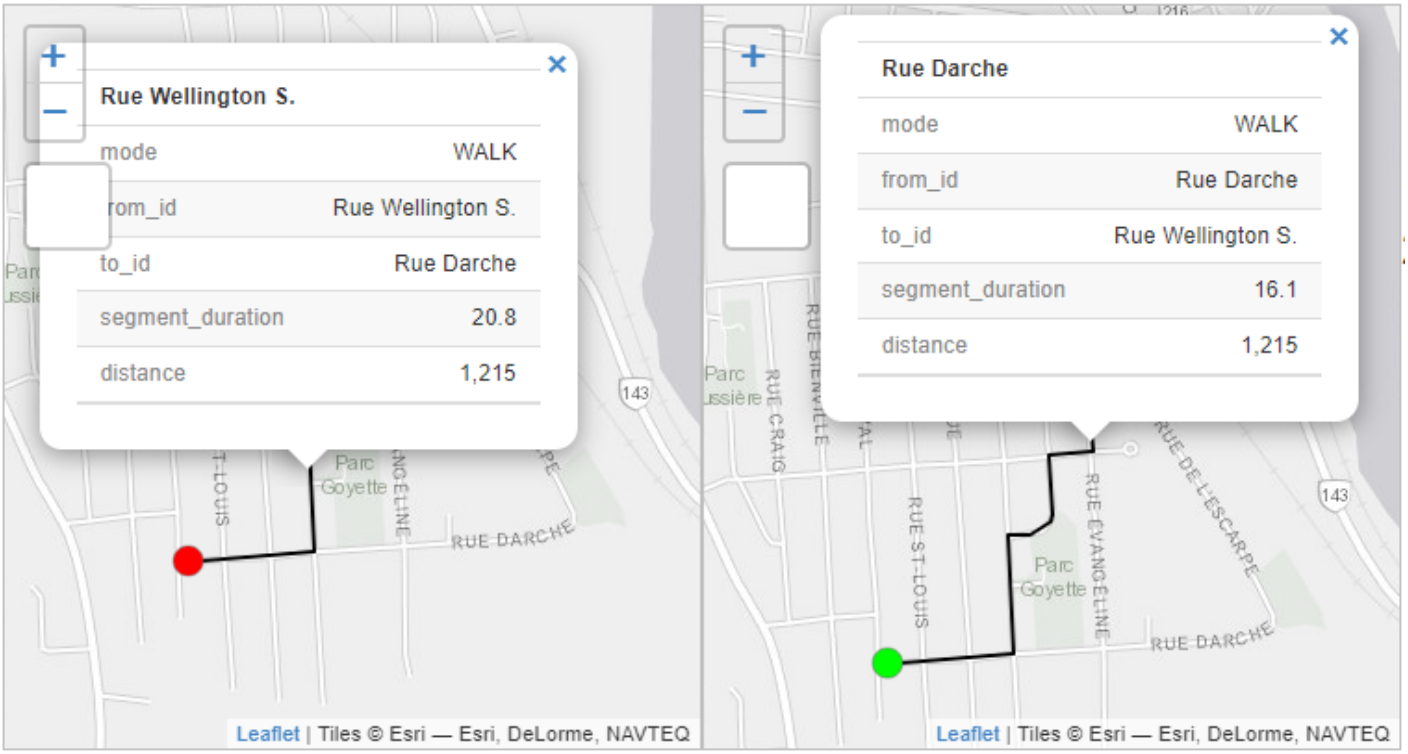


FIGURE 5.9 – Trajets à pied entre les deux destinations

5.2.3.2 Calcul d'itinéraires en automobile

Nous calculons ici l'itinéraire entre le campus principal de l'Université de Sherbrooke et une localisation (45,4220308; -71,881828). Nous fixons alors `mode = "CAR"` pour la fonction `detailed_itineraries`. Les trajets sont respectivement estimés à 21,7 et 18,8 minutes à destination et au départ du campus principal.

```
UDeS <- data.frame(id = "Campus principal", lon = -71.929526, lat = 45.378017)
UDeS <- st_as_sf(UDeS, coords = c("lon","lat"), crs = 4326)
Point <- data.frame(id = "Départ", lon = -71.881828, lat = 45.4220308)
Point <- st_as_sf(Point, coords = c("lon","lat"), crs = 4326)

Auto.Aller <- detailed_itineraries(r5r_core = r5r_core,
                                   origins = Point,
                                   destinations = UDeS,
                                   mode = "CAR", # Automobile
                                   shortest_path = FALSE,
                                   drop_geometry = FALSE)
Auto.Retour <- detailed_itineraries(r5r_core = r5r_core,
                                  origins = UDeS,
```

```
                                destinations = Point,
                                mode = "CAR", # Automobile
                                shortest_path = FALSE,
                                drop_geometry = FALSE)
Auto.Aller
```

```
Simple feature collection with 1 feature and 16 fields
Geometry type: LINESTRING
Dimension:     XY
Bounding box:  xmin: -71.93373 ymin: 45.37722 xmax: -71.88129 ymax: 45.42365
Geodetic CRS:  WGS 84
  from_id from_lat  from_lon            to_id   to_lat    to_lon option
1  Départ 45.42203 -71.88183 Campus principal 45.37802 -71.92953      1
  departure_time total_duration total_distance segment mode segment_duration
1       20:48:40           21.7          11348       1  CAR             21.7
  wait distance route                        geometry
1    0    11348       LINESTRING (-71.88129 45.42...
```

```
Auto.Retour
```

```
Simple feature collection with 1 feature and 16 fields
Geometry type: LINESTRING
Dimension:     XY
Bounding box:  xmin: -71.93312 ymin: 45.37759 xmax: -71.87607 ymax: 45.42234
Geodetic CRS:  WGS 84
           from_id from_lat  from_lon  to_id   to_lat    to_lon option
1 Campus principal 45.37802 -71.92953 Départ 45.42203 -71.88183      1
  departure_time total_duration total_distance segment mode segment_duration
1       20:48:40           18.9           9685       1  CAR             18.9
  wait distance route                        geometry
1    0     9685       LINESTRING (-71.92952 45.37...
```

Bien entendu, les deux trajets sont différents en raison des sens de circulation (figure 5.10).

```
# Cartographie des trajets avec tmap
Carte1 <- tm_shape(Auto.Aller)+
  tm_lines(col="black", lwd = 2,
           popup.vars = c("mode", "from_id", "to_id", "segment_duration", "distance"))+
  tm_shape(Point)+tm_dots(col="green", size = .15)+
  tm_shape(UDeS)+tm_dots(col="red", size = .15)
Carte2 <- tm_shape(Auto.Retour)+
  tm_lines(col="black", lwd = 2,
           popup.vars = c("mode", "from_id", "to_id", "segment_duration", "distance"))+
  tm_shape(Point)+tm_dots(col="red", size = .15)+
  tm_shape(UDeS)+tm_dots(col="green", size = .15)
```

```
# Figure avec les deux cartes
tmap_arrange(Carte1, Carte2)
```

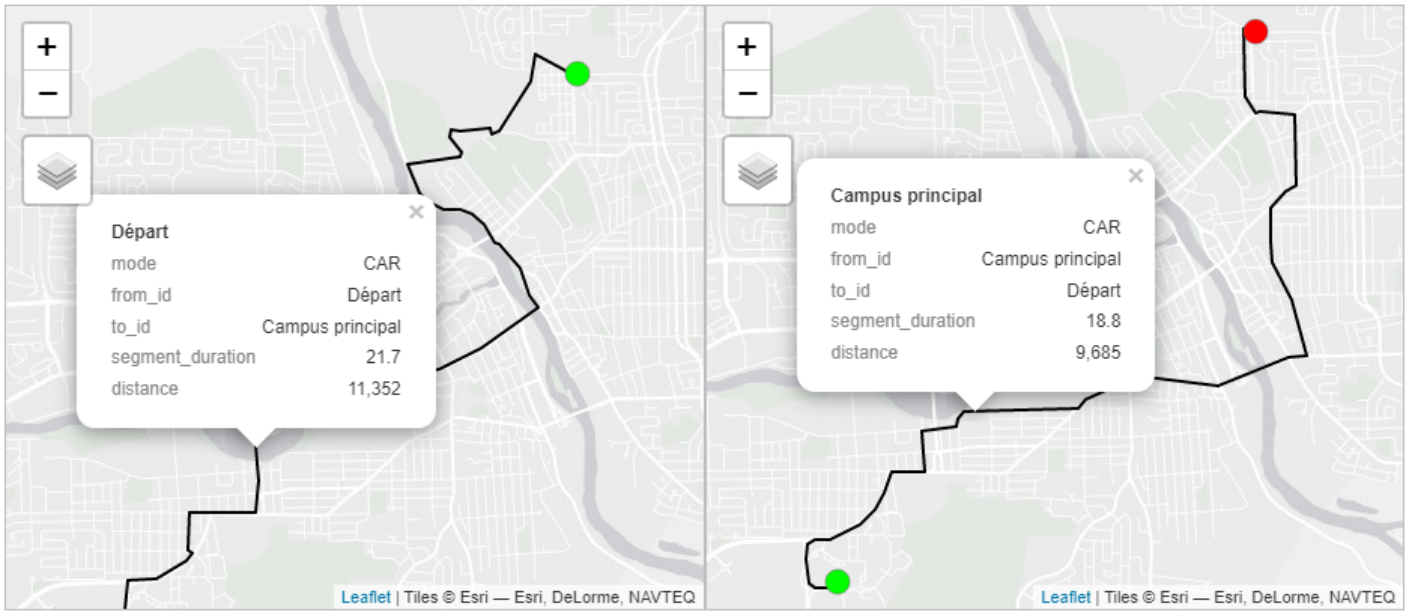


FIGURE 5.10 – Trajets en voiture entre les deux destinations

5.2.3.3 Calcul d'itinéraires en transport en commun

Pour le calcul d'itinéraires en transport en commun, nous devons fixer une heure de départ et un temps de marche maximal pour chaque segment du trajet réalisé à pied, soit :

- du domicile à l'arrêt de bus le plus proche;
- entre deux arrêts de bus de lignes différentes;
- du dernier arrêt de bus à la destination finale.

Dans le code ci-dessous, nous fixons les heures de départ et d'arrivée à 8 h et à 18 h pour le 24 avril 2025 et un temps de marche maximal de 20 minutes.

```
### Définition de la journée et de l'heure de départ
dateheure.matin <- as.POSIXct("24-04-2025 08:00:00",
                              format = "%d-%m-%Y %H:%M:%S")

dateheure.soir  <- as.POSIXct("24-04-2025 18:00:00",
                              format = "%d-%m-%Y %H:%M:%S")
### Définition du temps de marche maximal
minutes_marches_max <- 20
```

Toujours avec la fonction `detailed_itineraries`, nous modifions les paramètres comme suit :

- `mode = c("WALK", "TRANSIT")` pour le transport en commun.
- `walk_speed = 4.5` pour une vitesse moyenne à la marche de 4,5 km/h.
- `departure_datetime = dateheure.matin` pour un départ le 24 avril 2025 à 8 h.
- `departure_datetime = dateheure.soir` pour un départ le 24 avril 2025 à 18 h.
- `max_walk_time = minutes_marches_max` pour un temps maximal de marche de 20 minutes.

```
TC.Aller <- detailed_itineraries(r5r_core = r5r_core,
                                 origins = Point,
                                 destinations = UDeS,
                                 mode = c("WALK", "TRANSIT"),
                                 max_walk_time = minutes_marches_max,
                                 walk_speed = 4.5,
                                 departure_datetime = dateheure.matin,
                                 shortest_path = FALSE,
                                 drop_geometry = FALSE)

TC.Retour <- detailed_itineraries(r5r_core = r5r_core,
                                  origins = UDeS,
                                  destinations = Point,
                                  mode = c("WALK", "TRANSIT"),
                                  max_walk_time = minutes_marches_max,
                                  walk_speed = 4.5,
                                  departure_datetime = dateheure.soir,
                                  shortest_path = FALSE,
                                  drop_geometry = FALSE)
```

Pour l'option 1, la durée du trajet à 8 h vers l'Université de Sherbrooke est estimée à 52,8 minutes avec trois segments :

- Un premier segment de 8,6 minutes et 661 mètres à pied.
- Un second de 41,0 minutes et 10,862 km en autobus (ligne 8).
- Un troisième de 1,8 minutes et 127 mètres à pied.

```
#TC.Aller
subset(TC.Aller, option == 1)
```

```
Simple feature collection with 3 features and 16 fields
Geometry type: LINESTRING
Dimension:     XY
Bounding box:  xmin: -71.92952 ymin: 45.37756 xmax: -71.88129 ymax: 45.42204
Geodetic CRS:  WGS 84
  from_id from_lat  from_lon            to_id   to_lat    to_lon option
1  Départ 45.42203 -71.88183 Campus principal 45.37802 -71.92953      1
2  Départ 45.42203 -71.88183 Campus principal 45.37802 -71.92953      1
3  Départ 45.42203 -71.88183 Campus principal 45.37802 -71.92953      1
  departure_time total_duration total_distance segment mode segment_duration
1       08:05:58           43.5           8595       1 WALK             15.0
```

```
2       08:05:58           43.5           8595       2  BUS             25.4
3       08:05:58           43.5           8595       3 WALK              1.5
  wait distance route                       geometry
1  0.0     1246       LINESTRING (-71.88129 45.42...
2  1.6     7244     9 LINESTRING (-71.88479 45.41...
3  0.0      105       LINESTRING (-71.92938 45.37...
```

Pour l'option 1, la durée du trajet à 18 h au départ de l'Université de Sherbrooke est estimée à 51,8 minutes avec trois segments :

- Un premier segment de 2,0 minutes et 150 mètres à pied.
- Un second de 33,0 minutes et 11,207 km en autobus (ligne 8).
- Un troisième de 9,5 minutes et 750 mètres à pied.

```
TC.Retour
```

```
Simple feature collection with 10 features and 16 fields
Geometry type: LINESTRING
Dimension:     XY
Bounding box:  xmin: -71.9331 ymin: 45.3776 xmax: -71.86948 ymax: 45.42237
Geodetic CRS:  WGS 84
            from_id from_lat  from_lon  to_id   to_lat    to_lon option
1  Campus principal 45.37802 -71.92953 Départ 45.42203 -71.88183      1
2  Campus principal 45.37802 -71.92953 Départ 45.42203 -71.88183      1
3  Campus principal 45.37802 -71.92953 Départ 45.42203 -71.88183      1
4  Campus principal 45.37802 -71.92953 Départ 45.42203 -71.88183      2
5  Campus principal 45.37802 -71.92953 Départ 45.42203 -71.88183      2
6  Campus principal 45.37802 -71.92953 Départ 45.42203 -71.88183      2
7  Campus principal 45.37802 -71.92953 Départ 45.42203 -71.88183      2
8  Campus principal 45.37802 -71.92953 Départ 45.42203 -71.88183      2
9  Campus principal 45.37802 -71.92953 Départ 45.42203 -71.88183      2
10 Campus principal 45.37802 -71.92953 Départ 45.42203 -71.88183      2
   departure_time total_duration total_distance segment mode segment_duration
1        18:09:44           51.8          12219       1 WALK              4.3
2        18:09:44           51.8          12219       2  BUS             33.0
3        18:09:44           51.8          12219       3 WALK              9.6
4        18:03:26           30.2           9254       1 WALK              1.0
5        18:03:26           30.2           9254       2  BUS             10.0
6        18:03:26           30.2           9254       3 WALK              1.4
7        18:03:26           30.2           9254       4  BUS              8.0
8        18:03:26           30.2           9254       5 WALK              1.3
9        18:03:26           30.2           9254       6  BUS              3.0
10       18:03:26           30.2           9254       7 WALK              0.5
   wait distance route                       geometry
1   0.0      307       LINESTRING (-71.92953 45.37...
2   5.0    11207     8 LINESTRING (-71.9295 45.378...
```

```
3   0.0      705       LINESTRING (-71.87282 45.42...
4   0.0       81       LINESTRING (-71.92953 45.37...
5   1.6     4222    16 LINESTRING (-71.92961 45.37...
6   0.0      145       LINESTRING (-71.91845 45.40...
7   1.6     3315     4 LINESTRING (-71.91785 45.40...
8   0.0       97       LINESTRING (-71.88477 45.41...
9   1.8     1357     5 LINESTRING (-71.88371 45.41...
10  0.0       37       LINESTRING (-71.88222 45.42...
```

Les deux trajets en transport en commun sont représentés à la figure 5.11.

```
tmap_mode("view")
Carte1 <- tm_shape(subset(TC.Aller, option == 1))+
            tm_lines(col="mode", lwd = 3,
                     popup.vars = c("mode", "from_id", "to_id",
                                    "segment_duration", "distance",
                                    "total_duration", "total_distance"))+
  tm_shape(Point)+tm_dots(col="green", size = .15)+
  tm_shape(UDeS)+tm_dots(col="red", size = .15)+
  tm_view(view.legend.position = c("left", "top"))

Carte2 <- tm_shape(subset(TC.Retour, option == 1))+
              tm_lines(col="mode", lwd = 3,
                       popup.vars = c("mode", "from_id", "to_id",
                                      "segment_duration", "distance",
                                      "total_duration", "total_distance"))+
  tm_shape(Point)+tm_dots(col="red", size = .15)+
  tm_shape(UDeS)+tm_dots(col="green", size = .15)+
  tm_view(view.legend.position = c("left", "top"))

tmap_arrange(Carte1, Carte2, ncol = 2)
```

5.2.4 Délimitation d'isochrones avec R5R selon le mode de transport

Le fonction `isochrone` du *package* `R5R` permet de délimiter des zones de desserte selon la distance-temps et différents modes de transport. Ses paramètres sont d'ailleurs les mêmes que ceux de la fonction `detailed_itineraries`, à l'exception de :

- Il n'y a pas de paramètre `destinations` puisque l'isochrone est uniquement délimité à partir de points d'origines (`origins`).
- Le paramètre `cutoffs = c(0, 15, 30)` permet de définir différentes isochrones en minutes.
- Le paramètre `sample_size` est utilisé pour tracer les isochrones. Variant de 0,2 à 1, sa valeur par défaut de 0,8 signifie que 80 % des nœuds du réseau de transport sont utilisés pour tracer l'isochrone. Plus cette valeur est proche de 1, plus l'isochrone est précise, mais plus sa vitesse de calcul est longue.

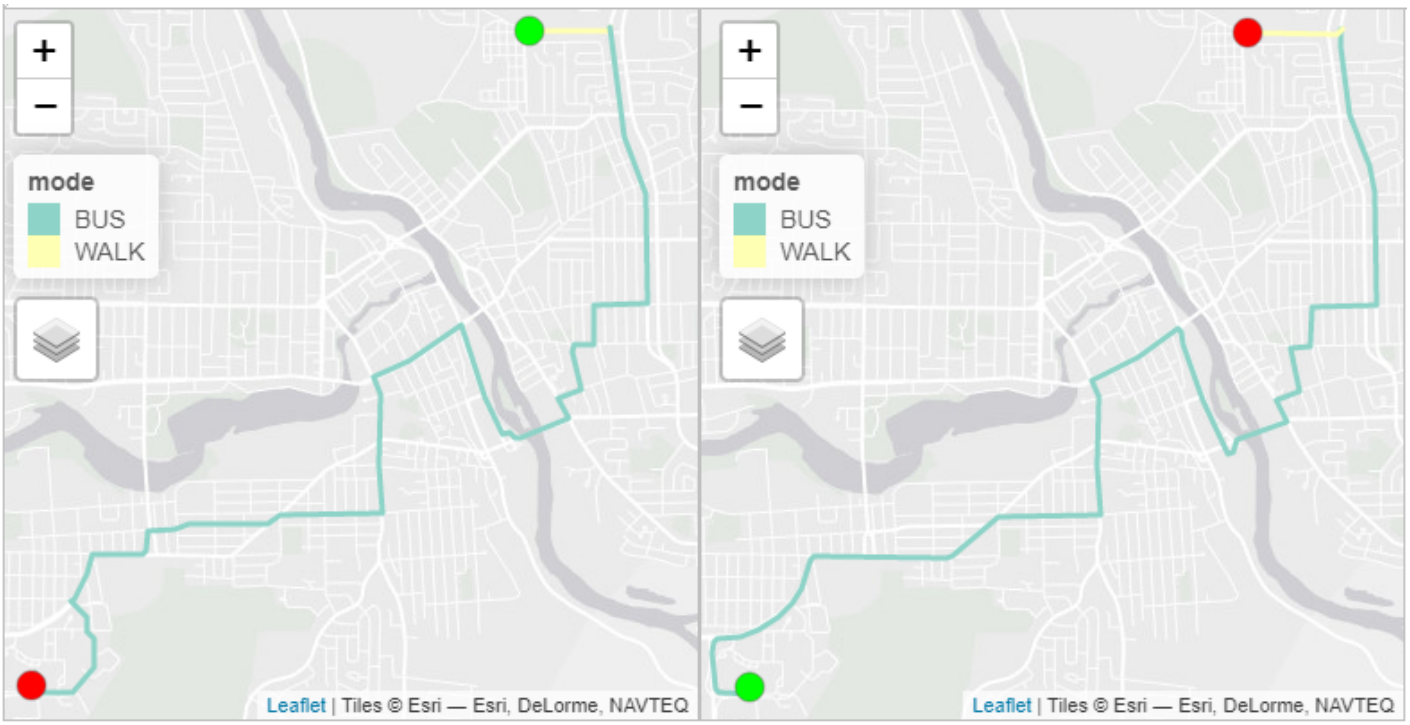


FIGURE 5.11 – Trajets en transport en commun entre les deux destinations

Dans l'exemple ci-dessous, nous calculons plusieurs isochrones à partir du campus principal de l'Université de Sherbrooke en fonction du mode transport (figure 5.12) :

1. Trois isochrones de 10, 20 et 30 minutes à pied.
2. Trois isochrones de 10, 20 et 30 minutes en vélo.
3. Trois isochrones de 5, 10 et 20 minutes en automobile.
4. Deux isochrone de 20 et 40 minutes en transport en commun le 24 avril 2025 à 18 h avec un durée maximale de marche de 15 minutes.

```
## Point pour l'université de Sherbrooke
UDeS <- data.frame(id = "Campus principal", lon = -71.929526, lat = 45.378017)
UDeS <- st_as_sf(UDeS, coords = c("lon","lat"), crs = 4326)
tmap_mode("view")
tmap_options(check.and.fix = TRUE)
## Trois isochrones à pied de 10, 20 et 30 minutes
Isochrome.marche <- isochrone(r5r_core,
                      origins = UDeS,
                      mode = "WALK",
                      cutoffs = c(10, 20, 30),
                      sample_size = .8,
                      time_window = 120,
                      progress = FALSE)
```

```
Carte.Marche <- tm_shape(Isochrome.marche)+
                    tm_fill(col="isochrone",
                            alpha = .4,
                            breaks = c(0, 10, 20, 30),
                            title ="Marche",
                            legend.format = list(text.separator = "à"))+
                    tm_shape(UDeS)+tm_dots(col="darkred", size = .25)

## Trois isochrones à vélo de 10, 20 et 30 minutes
Isochrome.velo <- isochrone(r5r_core,
                      origins = UDeS,
                      mode = "BICYCLE",
                      cutoffs = c(10, 20, 30),
                      sample_size = .8,
                      time_window = 120,
                      progress = FALSE)

Carto.Velo <- tm_shape(Isochrome.velo)+
                    tm_fill(col="isochrone",
                            alpha = .4,
                            breaks = c(0, 10, 20, 30),
                            title ="Vélo",
                            legend.format = list(text.separator = "à"))+
                    tm_shape(UDeS)+tm_dots(col="darkred", size = .25)

## Trois isochrones en auto de 5, 10 et 20 minutes
Isochrome.auto <- isochrone(r5r_core,
                      origins = UDeS,
                      mode = "CAR",
                      cutoffs = c(5, 10, 20),
                      sample_size = 1,
                      time_window = 120,
                      progress = FALSE)

Carto.Auto <- tm_shape(Isochrome.auto)+
                    tm_fill(col="isochrone",
                            alpha = .4,
                            breaks = c(0, 5, 10, 20),
                            title ="Automobile",
                            legend.format = list(text.separator = "à"))+
                    tm_shape(UDeS)+tm_dots(col="darkred", size = .25)

## Deux isochrones en transport en commun de 20, 40 et 60 minutes
dateheure.soir  <- as.POSIXct("24-04-2025 08:00:00",
                              format = "%d-%m-%Y %H:%M:%S")
Isochrome.tc <- isochrone(r5r_core,
```

```
                      origins = UDeS,
                      mode = c("WALK", "TRANSIT"),
                      max_walk_time = 15,
                      departure_datetime = dateheure.soir,
                      cutoffs = c(20, 40),
                      sample_size = 1,
                      time_window = 120,
                      progress = FALSE)

Carto.TC <- tm_shape(Isochrome.tc)+
                    tm_fill(col="isochrone",
                            alpha = .4,
                            breaks = c(0, 20, 40),
                            title ="Transport en commun",
                            legend.format = list(text.separator = "à"))+
                    tm_shape(UDeS)+tm_dots(col="darkred", size = .25)
## Figure avec les quatre cartes
tmap_arrange(Carte.Marche, Carto.Velo,
             Carto.Auto, Carto.TC,
             ncol = 2, nrow = 2)
```

5.2.5 Calcul de matrices OD selon différents modes de transport

Pour calculer des matrices origines-destinations selon différents modes de transport, nous utilisons la fonction `travel_time_matrix` dont les paramètres sont quasi les mêmes que la `detailed_itineraries` (section 5.2.3). Dans le code ci-dessous, nous importons 284 adresses tirées aléatoirement et les supermarchés présents sur le territoire de la ville de Sherbrooke.

```
## Importation des couches
Adresses <- st_read(dsn = "data/Chap05/AutresDonnees/Commerces.gpkg",
                        layer = "AdressesAleatoires", quiet = TRUE)
supermarches <- st_read(dsn = "data/Chap05/AutresDonnees/Commerces.gpkg",
                        layer = "supermarche", quiet = TRUE)
## Nombre de distances à calculer
nO = nrow(Adresses)
nD = nrow(supermarches)
NOD = nO * nD
cat("Origines (O) :", nO, "adresses",
    "\n Destinations (D) :", nD, "supermarchés",
    "\n Distances OD à calculer = ", NOD)
```

```
Origines (O) : 184 adresses
 Destinations (D) : 27 supermarchés
 Distances OD à calculer =  4968
```

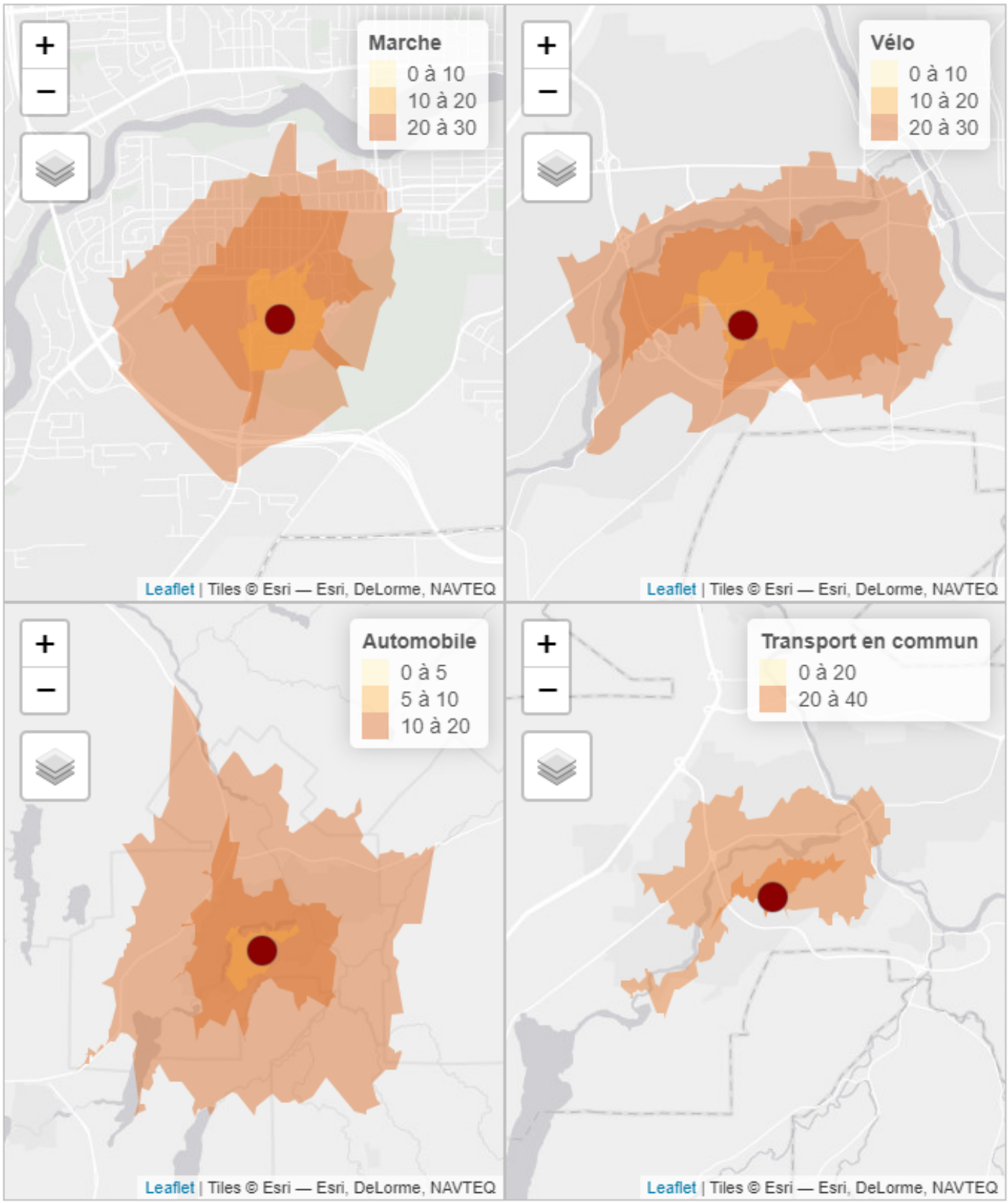


FIGURE 5.12 – Isochrones selon les quatre modes de transport

```r
## Origines et destinations
Origines <- Adresses
Origines$id <- as.character(Adresses$id)
Origines$lat <- st_coordinates(Adresses)[,2]
Origines$lon <- st_coordinates(Adresses)[,1]
Destinations <- supermarches
Destinations$id <- supermarches$osm_id
Destinations$lat <- st_coordinates(supermarches)[,2]
Destinations$lon <- st_coordinates(supermarches)[,1]
names(Destinations)[1] <- "id"
```

Par la suite, nous calculons les différentes matrices OD :

- **`matriceOD.Auto`** avec `mode = "CAR"`.
- **`matriceOD.Marche`** avec `mode = "WALK"`, `walk_speed = 4.5` et `max_trip_duration = 60`. La durée du trajet est limitée à 60 minutes avec une vitesse moyenne de 4,5 km/h.
- **`matriceOD.Velo`** avec `mode = "BICYCLE"`, `bike_speed = 15` et `max_trip_duration = 60`. La durée du trajet est limitée à 60 minutes avec une vitesse moyenne de 15 km/h.
- **`matriceOD.TC`** avec `mode = "c("WALK", "TRANSIT")"`, `walk_speed = 4.5`, `max_walk_time = 30`, `max_trip_duration = 120` et `departure_datetime = dateheure.soir`. La durée du trajet est limitée à 60 minutes avec une vitesse moyenne de marche de 4,5 km/h et une durée maximale de 30 minutes pour chaque segment à la marche. L'heure de départ a été fixée comme suit : `dateheure.soir  <- as.POSIXct("04-05-2023 18:00:00", format = "%d-%m-%Y %H:%M:%S")`.

```r
## Matrice OD en voiture
t1 <-Sys.time()
matriceOD.Auto <- travel_time_matrix(r5r_core = r5r_core,
                                     origins = Origines,
                                     destinations = Destinations,
                                     mode = "CAR")
t2 <-Sys.time()
duree.auto = as.numeric(difftime(t2, t1), units = "secs")
## Matrice OD à la marche
t1 <-Sys.time()
matriceOD.Marche <- travel_time_matrix(r5r_core = r5r_core,
                                       origins = Origines,
                                       destinations = Destinations,
                                       mode = "WALK",
                                       walk_speed = 4.5,  # valeur par défaut 3.6
                                       max_trip_duration = 60, # 1 heure de marche maximum
                                       max_walk_time = Inf)
t2 <-Sys.time()
duree.marche = as.numeric(difftime(t2, t1), units = "secs")
## Matrice OD en vélo
t1 <-Sys.time()
matriceOD.Velo <- travel_time_matrix(r5r_core = r5r_core,
```

```
                                    origins = Origines,
                                    destinations = Destinations,
                                    mode = "BICYCLE",
                                    bike_speed = 15,
                                    max_trip_duration = 60,
                                    max_bike_time = Inf)
t2 <-Sys.time()
duree.velo = as.numeric(difftime(t2, t1), units = "secs")
## Matrice OD en transport en commun
dateheure.soir  <- as.POSIXct("24-04-2025 08:00:00",
                              format = "%d-%m-%Y %H:%M:%S")
t1 <-Sys.time()
matriceOD.TC <- travel_time_matrix(r5r_core = r5r_core,
                                   origins = Origines,
                                   destinations = Destinations,
                                   mode = c("WALK", "TRANSIT"),
                                   walk_speed = 4.5,
                                   max_walk_time = 30,
                                   max_trip_duration = 120,
                                   departure_datetime = dateheure.soir)
t2 <-Sys.time()
duree.tc = as.numeric(difftime(t2, t1), units = "secs")
```

Les temps de calcul des différentes matrices sont reportés ci-dessous.

```
cat("Temps de calcul des matrices :",
     "\n Voiture : ", round(duree.auto,2), "secondes",
     "\n Marche : ", round(duree.marche,2), "secondes",
     "\n Vélo : ", round(duree.velo,2), "secondes",
     "\n Transport en commun : ", round(duree.tc,2), "secondes")
```

```
Temps de calcul des matrices :
 Voiture :  8.82 secondes
 Marche :  0.32 secondes
 Vélo :  2.93 secondes
 Transport en commun :  0.41 secondes
```

Une fois les matrices obtenues, nous les enregistrons dans un fichier `Rdata`.

```
save(matriceOD.Auto, matriceOD.Marche,
     matriceOD.Velo, matriceOD.TC,
     file="data/chap05/Resultats/MatricesOD.Rdata")
```

À partir de ces matrices, nous extrayons la valeur minimale pour chacune des adresses pour les quatre modes de transport. Puis, nous opérons une jointure attributaire avec la couche des adresses aléatoires avec la fonction `merge`.

```
## Création d'un vecteur pour la distance au supermarché le plus proche
SupermarchePlusProche.Auto <- aggregate(travel_time_p50 ~ from_id, matriceOD.Auto, min)
SupermarchePlusProche.Pied <- aggregate(travel_time_p50 ~ from_id, matriceOD.Marche, min)
SupermarchePlusProche.Velo <- aggregate(travel_time_p50 ~ from_id, matriceOD.Velo, min)
SupermarchePlusProche.tc   <- aggregate(travel_time_p50 ~ from_id, matriceOD.TC, min)
## Changement des noms des champs
names(SupermarchePlusProche.Auto) <- c("id", "SupPlusProcheAuto")
names(SupermarchePlusProche.Pied) <- c("id", "SupPlusProchePied")
names(SupermarchePlusProche.Velo) <- c("id", "SupPlusProcheVelo")
names(SupermarchePlusProche.tc)   <- c("id", "SupPlusProcheTC")
## Jointure avec la couche des adresses
Adresses <- merge(Adresses, SupermarchePlusProche.Auto, by ="id", all.x=TRUE)
Adresses <- merge(Adresses, SupermarchePlusProche.Pied, by ="id", all.x=TRUE)
Adresses <- merge(Adresses, SupermarchePlusProche.Velo, by ="id", all.x=TRUE)
Adresses <- merge(Adresses, SupermarchePlusProche.tc, by ="id", all.x=TRUE)
```

Finalement, nous utilisons le *package* `tmap` pour cartographier les résultats et réaliser une figure avec la fonction `tmap_arrange`.

```
## Importation des arrondissements de la ville de Sherbrooke
arrondissements <- st_read(dsn = "data/Chap05/AutresDonnees/Arrondissements.shp",
                           quiet=TRUE)
## Construction des cartes
tmap_mode("plot")
max.auto <- max(Adresses$SupPlusProcheAuto,na.rm=TRUE)
Carte1 <- tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_shape(Adresses)+
            tm_dots(col="SupPlusProcheAuto",
                    border.lwd = 1,
                    style = "fixed",
                    breaks = c(0,5,10,max.auto),
                    palette="YlOrRd",
                    size = .2,
                    legend.format = list(text.separator = "à"),
                    title="Voiture")+
          tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_layout(legend.format = list(text.separator = "à"),
                    frame = FALSE, legend.outside = TRUE)

max.pied <- max(Adresses$SupPlusProchePied,na.rm=TRUE)
Carte2 <- tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_shape(Adresses)+
            tm_dots(col="SupPlusProchePied",
                    border.lwd = 1,
                    style = "fixed",
                    breaks = c(0,5,10,15,20,30,45, max.pied),
```

```
                    palette="YlOrRd",
                    size = .2,
                    title="Marche",
                    textNA = "Plus de 60")+
          tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_layout(legend.format = list(text.separator = "à"),
                    frame = FALSE, legend.outside = TRUE)

max.velo <- max(Adresses$SupPlusProcheVelo,na.rm=TRUE)
Carte3 <- tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_shape(Adresses)+
            tm_dots(col="SupPlusProcheVelo",
                    border.lwd = 1,
                    style = "fixed",
                    breaks = c(0,5,10,20,30,45,max.velo),
                    palette="YlOrRd",
                    size = .2,
                    title="Vélo",
                    textNA = "Plus de 60")+
          tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_layout(legend.format = list(text.separator = "à"),
                    frame = FALSE, legend.outside = TRUE)

max.tc <- max(Adresses$SupPlusProcheTC,na.rm=TRUE)
Carte4 <- tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_shape(Adresses)+
            tm_dots(col="SupPlusProcheTC",
                    border.lwd = 1,
                    style = "fixed",
                    breaks = c(0,10,15,20,25,max.tc),
                    palette="YlOrRd",
                    size = .2,
                    title="Bus",
                    textNA = "Plus de 60")+
          tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_layout(legend.format = list(text.separator = "à"),
                    frame = FALSE, legend.outside = TRUE)

## Figure avec les quatre cartes
tmap_arrange(Carte1, Carte2, Carte3, Carte4)
```

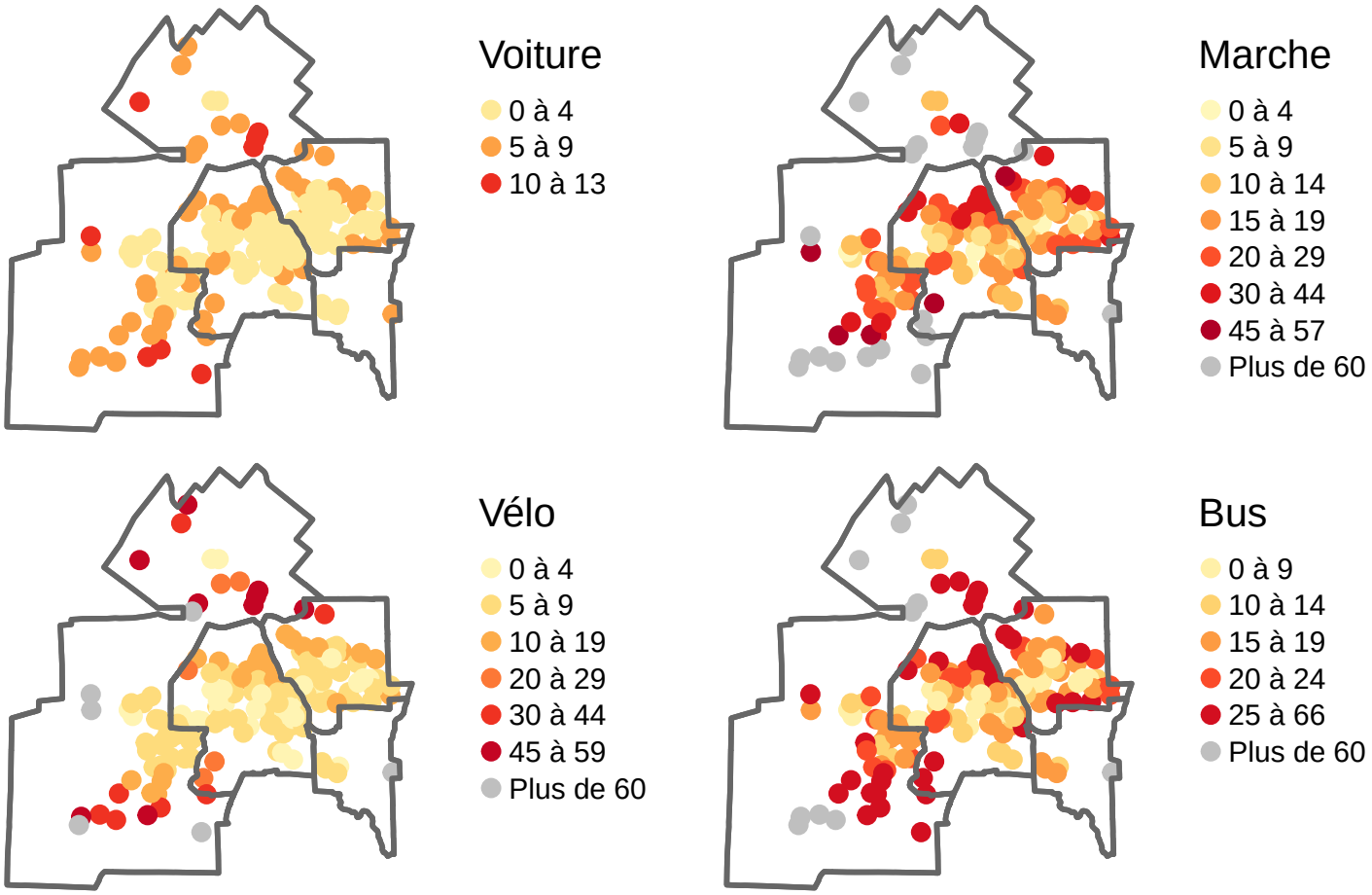


FIGURE 5.13 – Supermarché le plus proche en minutes selon le mode de transport

5.3 Mesures d'accessibilité

5.3.1 Notion d'accessibilité

Dans un article fondateur intitulé *The concept of access: definition and relationship to consumer satisfaction*, Roy Penchansky et William Thomas (1981) ont identifié cinq dimensions fondamentales au concept d'accessibilité aux services de santé :

1. **L'accessibilité spatiale** (*accessibility*) renvoie à la proximité géographique du service par rapport à la population.
2. **La disponibilité** (*availability*) renvoie à la quantité et aux types de services offerts selon les besoins des individus.
3. **L'organisation** (*accommodation*) renvoie au fonctionnement du service (horaires, délais d'attente, prises de rendez-vous, etc.).
4. **L'accessibilité financière** (*affordability*) renvoie aux coûts du service qui peuvent constituer une barrière financière pour les personnes défavorisées.
5. **L'accessibilité socioculturelle** (*acceptability*) renvoie à l'acceptation et à l'adaptation des services aux différences sociales, culturelles ou linguistiques des personnes.

Note

Les cinq dimensions de l'accessibilité et le type de service analysé
L'importance accordée à chacune des cinq dimensions identifiées par Roy Penchansky et William Thomas (1981) varie en fonction du type de service à l'étude. Prenons l'exemple des parcs urbains :

1. L'**accessibilité spatiale**, soit la proximité géographique est sans aucun doute une dimension très importante.
2. La **disponibilité** (*availability*) renvoie à différents équipements (aires de jeu pour enfants, terrains de sports, etc.) présents dans le parc.
3. La dimension de l'**organisation** (*accommodation*) risque d'être moins importante puisque les heures d'ouverture et les modalités de réservation de certains types de terrain de sport (comme un terrain de tennis) ne varient habituellement pas d'un parc à l'autre au sein d'une même ville.
4. La dimension de l'**accessibilité financière** (*affordability*) risque aussi d'être moins importante puisque l'accès au parc et à ses équipements est gratuit, à l'exception de certains terrains de sport très spécialisés.
5. L'**accessibilité socioculturelle** (*acceptability*) peut être une dimension très importante et renvoie à l'acceptation des différences sociales, générationnelles et ethnoculturelles des personnes utilisatrices des parcs.

Plus récemment, la notion d'accessibilité à un service a été définie selon deux dimensions, soit **réelle (ou révélée)** ou **potentielle** et **spatiale** ou **aspatiale** (Guagliardo 2004; Luo et Wang 2003; Khan 1992) :

1. L'accessibilité réelle renvoie à l'utilisation effective des services tandis que l'accessibilité potentielle renvoie à leur utilisation probable.
2. L'accessibilité spatiale renvoie à l'importance de la séparation spatiale entre l'offre et la demande de services en tant que barrière ou facilitateur, et l'accessibilité aspatiale (dimensions financière, socioculturelle, organisationnelle) se concentre sur les barrières ou facilitateurs non géographiques (Luo et Wang 2003; Ngui et Apparicio 2011).

Par conséquent, la notion d'accessibilité aux services de santé englobe quatre catégories principales : **l'accessibilité spatiale réelle**, **l'accessibilité aspatiale réelle**, **l'accessibilité spatiale potentielle** et **l'accessibilité aspatiale potentielle** (Khan 1992). Par exemple, si nous questionnons un groupe de personnes sur la localisation des parcs qu'elles fréquentent habituellement, nous pourrions dresser un portrait sur leur accessibilité spatiale réelle aux parcs. Par contre, si nous calculons le nombre d'hectares de parcs présents dans un rayon de 20 minutes de marche autour de leur domicile, nous pourrions évaluer leur accessibilité spatiale potentielle aux parcs.

5.3.2 Accessibilité spatiale potentielle

Dans le cadre de cette section, nous abordons l'accessibilité spatiale potentielle qui suppose de paramétrer quatre éléments : l'unité spatiale de référence, la méthode d'agrégation, la ou les mesures d'accessibilité et le type de distance (Apparicio et al. 2017).

5.3.2.1 Unité spatiale de référence

L'entité spatiale de référence correspond aux entités spatiales pour lesquelles l'accessibilité sera évaluée et cartographiée qui pourrait être :

— Les centroïdes des bâtiments résidentiels d'une ville donnée.
— Des entités polygonales représentant des zones résidentielles comme des aires de diffusion (comprenant de 400 à 700 habitants) ou des secteurs de recensement (de 2500 à 8000 habitants).

L'aire de diffusion et surtout le secteur de recensement sont souvent choisis puisqu'une panoplie de variables socioéconomiques, sociodémographiques et relatives au logement sont rattachées à ces entités spatiales pour les différents recensements de Statistique Canada. La sélection de ces entités spatiales (aire de diffusion ou secteur de recensement) permet alors d'évaluer les relations entre les mesures d'accessibilité et les variables socioéconomiques ou sociodémographiques. Néanmoins, cela nécessite de recourir à méthodes d'agrégation afin de limiter les erreurs dans la mesure de l'accessibilité spatiale potentielle (Apparicio et al. 2017; Hewko, Smoyer-Tomic et Hodgson 2002).

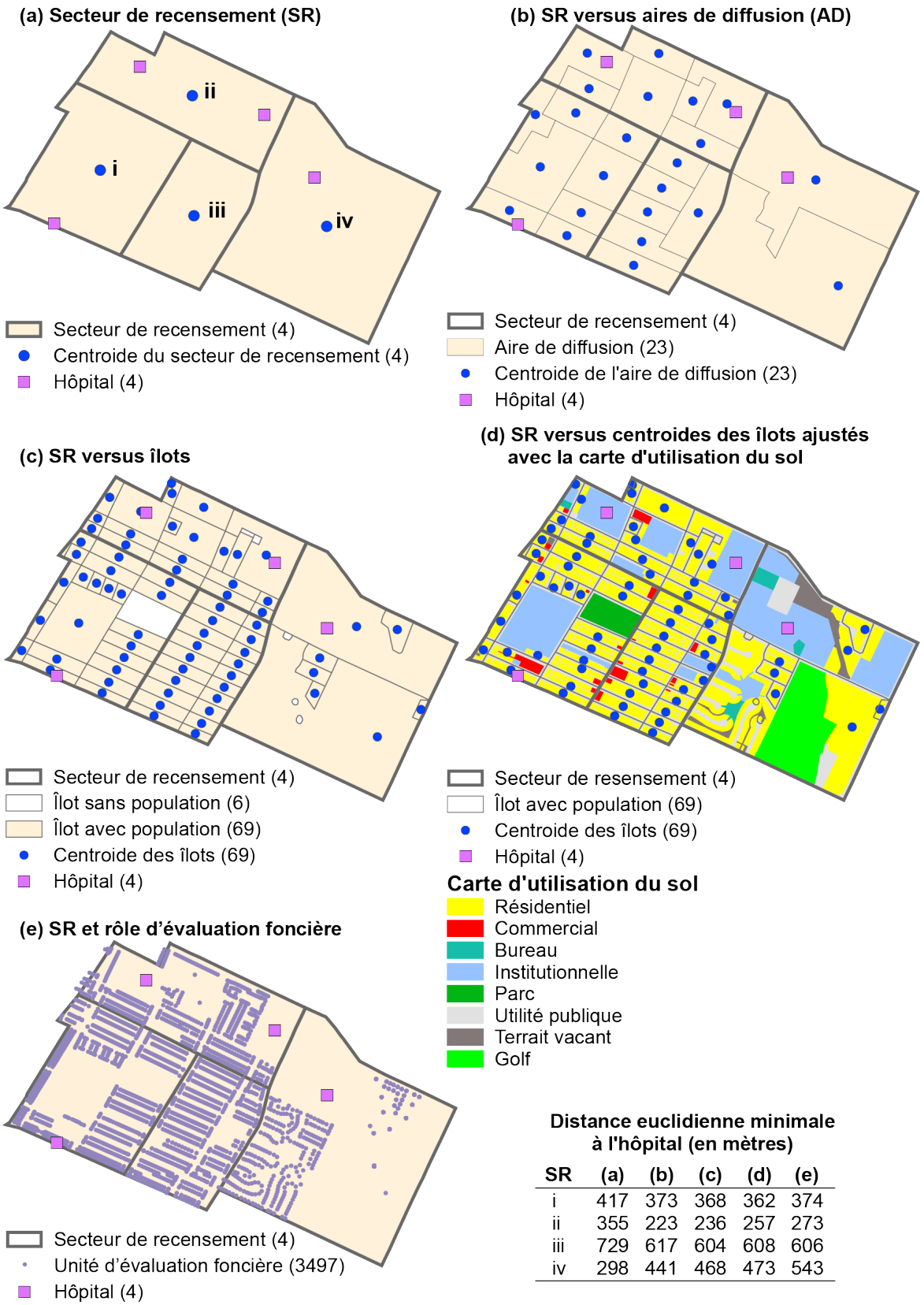
5.3.2.2 Méthodes d'agrégation

Dans un article méthodologique sur la comparaison des approches pour évaluer l'accessibilité spatiale potentielle, Apparicio *et al.* (2017) ont répertorié quatre principales méthodes d'agrégation pour évaluer une mesure d'accessibilité pour les secteurs de recensement. Ces approches, de la moins à la plus précise, sont les suivantes :

— La première approche consiste à calculer la distance entre le centroïde du secteur de recensement et le service (figure 5.14, a). Plus la taille du secteur de recensement est grande, plus l'erreur d'agrégation (l'imprécision de la mesure d'accessibilité) risque d'être importante puisqu'elle ne tient pas compte de la distribution spatiale de la population à l'intérieur du secteur de recensement. Autrement dit, cette approche revient à supposer que toute la population réside en son centroïde.
— La seconde approche consiste à calculer la distance entre les services et les centroïdes d'entités spatiales entièrement incluses dans les secteurs de recensement, puis à calculer la moyenne de ces distances pondérée par la population totale de chaque entité spatiale. Cette approche est réalisée avec les aires de diffusion et les îlots inclus dans les secteurs de recensement (figure 5.14, b et c). Bien entendu, les résultats sont plus précis avec les îlots de diffusion que les aires de diffusion puisqu'ils sont de taille plus réduite.
— La troisième approche consiste à ajuster la localisation des centroïdes des îlots en ne retenant que la partie résidentielle avec une carte d'occupation du sol (figure 5.14, d).
— Finalement, la quatrième approche consiste à utiliser le rôle d'évaluation foncière. Nous calculons alors les distances entre chaque unité d'évaluation foncière et les services, puis la moyenne pondérée de ces distances par le nombre de logements. Cette approche est sans aucun doute la plus précise, mais elle est bien plus chronophage. En effet, à la figure 5.14, nous avons respectivement 4 secteurs de recensement (a), 23 aires de diffusion (b), 69 îlots (c et d) et 3497 unités d'évaluation foncière (e).

5.3.2.3 Mesures d'accessibilité

Différentes mesures renvoyant à différentes conceptualisations de l'accessibilité peuvent être utilisées pour évaluer l'accessibilité spatiale potentielle; les principales sont reportées au tableau 5.1. Pour une description détaillée de ces mesures et de leurs formules, consultez l'article d'Apparicio *et al.* (2017).



Distance euclidienne minimale à l'hôpital (en mètres)

SR	(a)	(b)	(c)	(d)	(e)
i	417	373	368	362	374
ii	355	223	236	257	273
iii	729	617	604	608	606
iv	298	441	468	473	543

Note : Les chiffres entre parenthèses indiquent le nombre d'entités spatiales. Figure adaptée d'Apparicio *et al.* (2017).

FIGURE 5.14 – Méthodes d'agrégation et erreurs potentielles

Tableau 5.1 – Conceptualisation et mesures de l'accessibilité spatiale potentielle aux services

Tableau 5.1 – Conceptualisation et mesures de l'accessibilité spatiale potentielle aux services

Conceptualisation	Mesures d'accessibilité
Proximité immédiate	Distance entre l'origine et le service le plus proche
Offre de services dans l'environnement immédiat	Nombre de services présents à moins de *n* mètres ou minutes
Coût moyen pour atteindre tous les services	Distance moyenne entre une origine et tous les services
Coût moyen pour atteindre toutes les *n* destinations	Distance moyenne entre une origine et *n* services
Accessibilité en fonction de l'offre et la demande	Modèles gravitaires et méthodes *two-step floating catchment area* (2SFCA)

Source : Apparicio *et al.* (2017).

Pour poser un diagnostic d'accessibilité spatiale potentielle à un service pour un territoire donné, plusieurs chercheuses et chercheurs recommandent d'utiliser plusieurs mesures d'accessibilité.

Par exemple, dans une étude sur les déserts alimentaires à Montréal, Apparicio *et al.* (2007) utilisent trois mesures d'accessibilité : la distance au supermarché le plus proche (proximité immédiate), le nombre de supermarchés dans un rayon de 1000 mètres (offre dans l'environnement immédiat) et la distance moyenne aux trois supermarchés d'enseignes différentes (diversité en termes d'offre et de prix) à travers le réseau de rues.

Dans une autre étude portant sur l'accessibilité spatiale potentielle aux parcs urbains à Montréal, Jepson *et al.* (2022) utilisent deux mesures d'accessibilité : la distance au parc le plus proche (proximité immédiate) et la mesure E2FCA (congestion potentielle en fonction de l'offre et la demande) calculées pour les aires de diffusion de la Communauté métropolitaine de Montréal (figure 5.15). Concernant la proximité immédiate, le niveau d'accessibilité est bien élevé sur l'île de Montréal et inversement, plus faible à Laval et dans la Rive-Nord et la Rive-Sud (figure 5.15, a). En effet, la quasi-totalité des aires de diffusion de l'île de Montréal a un parc à moins de 200 mètres de marche. Concernant la congestion potentielle des parcs, le portrait est tout autre : le niveau de congestion potentielle est faible dans les zones suburbaines (Laval et les deux Rives) tandis qu'il est élevé dans les quartiers centraux de l'île de Montréal (figure 5.15, b). Autrement dit, les habitants des quartiers centraux de la ville de Montréal vivent plus près d'un parc, mais ce dernier est potentiellement plus congestionné. Or, une surutilisation des parcs peut entraîner une dégradation accélérée des équipements (aires de jeu, terrains de sports, etc.), voire décourager certaines personnes à visiter un parc durant les périodes plus achalandées.

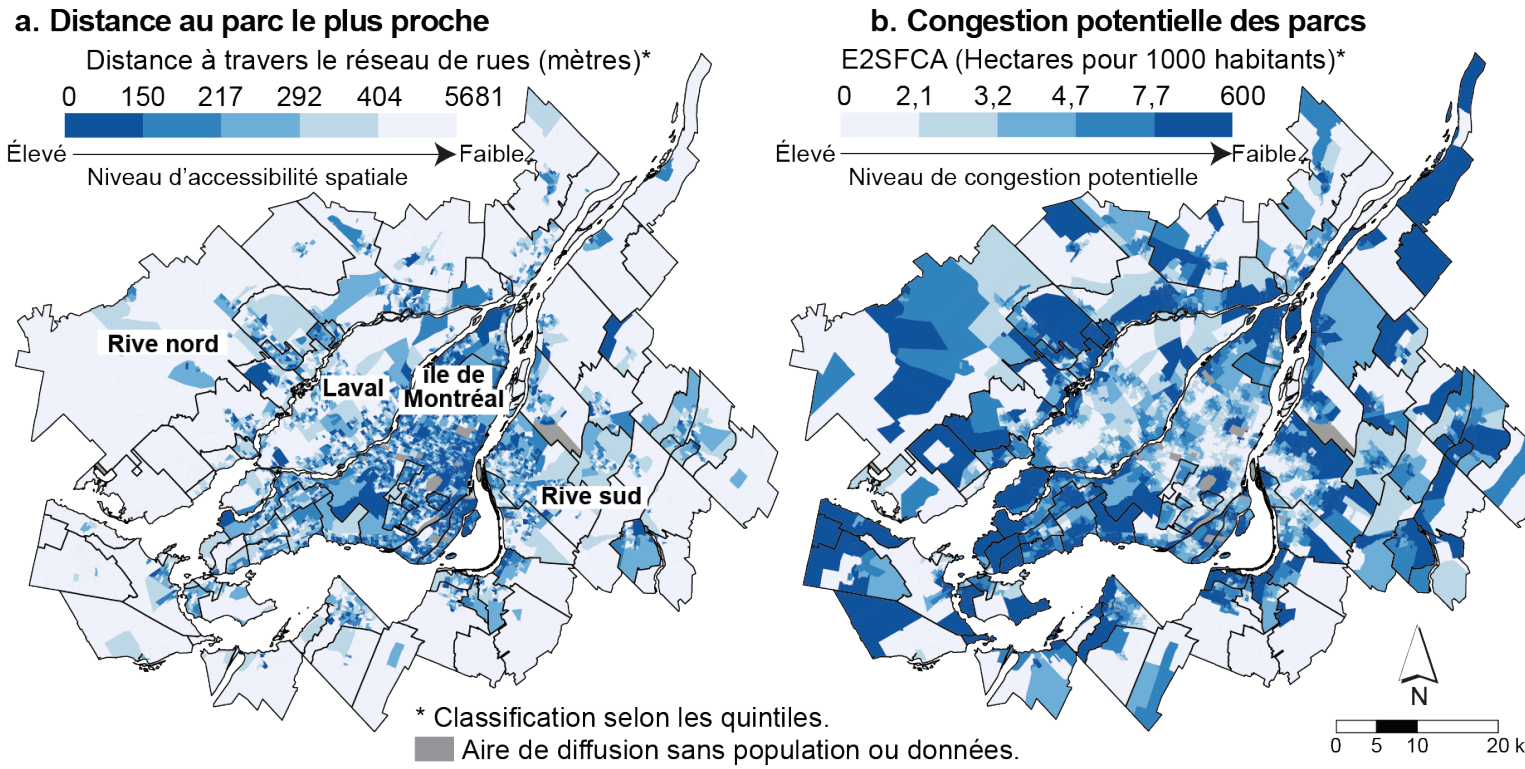


Figure adaptée de Jepson, Apparicio et Pham (2022).

FIGURE 5.15 – Deux mesures d'accessibilité spatiale potentielle aux parcs, aires de diffusion de la Communauté métropolitaine de Montréal, 2016

Aller plus loin

Le 2SFCA et ses fonctions de pondération

Les mesures appartenant à la famille des *Two Step Floating Catchement Area* (TSFCA) permettent d'évaluer l'accessibilité à des services en tenant comptant à la fois de l'offre (taille du service, par exemple le nombre de lits dans un hôpital) et de la demande (population résidant à la proximité des services).

Tel le nom l'indique, le calcul des mesures du TSFCA comprend deux étapes. Si nous reprenons l'exemple des parcs, la première étape consiste à calculer un ratio R_j pour chaque parc, indiquant le nombre d'hectares de parcs pour 1000 personnes résidant dans un rayon de *n* mètres ou minutes du parc :

$$R_j = \frac{S_j}{\sum_{j \in \{d_{ij} \leq d_0\}} P_i \times W(d_{ij})} \tag{5.1}$$

avec :

- j, un parc.
- S_j la capacité du parc j (ici, la superficie en hectares).
- i, une entité géographique (ici, une aire de diffusion).
- d_{ij}, la distance entre l'entité géographique i et le parc j.
- d_0, le seuil de distance maximale (par exemple un kilomètre ou trente minutes).
- P_i, la population totale résidant dans i. Cette population est habituellement exprimée en milliers d'habitants, soit $P_i/1000$.
- W, une fonction de pondération permettant de contrôler le fait qu'un parc plus éloigné est moins attractif et contribue moins à la valeur finale de la mesure d'accessibilité.

En résumé, R_j est le ratio entre la superficie du parc et la population ayant accès au parc dans un rayon *n* mètres ou minutes (d_0).

La deuxième étape consiste à calculer la somme pour chaque aire de diffusion i de la disponibilité des parcs à proximité.

$$A_i = \sum_{j=1}^{m} R_j \times W(d_{ji}) \tag{5.2}$$

Nous obtenons au final A_i, soit pour chaque aire de diffusion le nombre total d'hectares de parcs disponible pour 1000 habitants.

Notez que la fonction W joue un rôle très important dans la formulation du 2SFCA. Elle peut prendre plusieurs formes, la plus simple étant une fonction binaire donnant un poids de 1 aux parcs situés en-dessous d'une certaine distance. Par exemple, la fonction R ci-dessous donne un poids de 1 à des parcs situés à moins de 500 mètres et 0 au-delà.

```
w_binaire <- function(d){ifelse(d <= 500, 1, 0)}
```

Le problème de cette formulation est qu'elle implique que l'accessibilité au parc est la même que l'on habite à 25 ou 500 mètres du parc. Il a donc été proposé d'améliorer la méthode en ajoutant une fonction de pondération par palier qui accorde un poids différent selon des seuils de distances prédéfinis. Les fonctions R ci-dessous font ainsi varier le poids dans trois catégories de distance selon une pondération appelée *slow decay* et *fast decay* (Luo et Qi 2009) et des paliers de 150, 300 et 500 mètres.

```
w_slow_decay <- function(d){
  w <- case_when(
    d <= 150 ~ 1,
    d > 150 & d <= 300 ~ 0.68,
    d > 300 & d <= 500 ~ 0.22,
    d > 500 ~ 0
  )
  return(w)
}
```

5.3.2.4 Types de distance

Tel que décrit à la section 5.1.2, les mesures d'accessibilité peuvent être calculées en fonction de trajets les plus rapides selon différents modes de transport, soit en automobile, à pied, en vélo et en transport en commun (figure 5.4).

5.4 Mesures d'accessibilité spatiale potentielle dans R

5.4.1 Accessibilité spatiale potentielle aux supermarchés

Dans ce premier exemple applicatif dans R, nous élaborons un diagnostic de l'accessibilité spatiale potentielle aux supermarchés dans la ville de Sherbrooke avec les quatre paramètres suivants :

1. **Unité spatiale de référence** : aires de diffusion (AD) de 2021 de la ville de Sherbrooke.
2. **Méthode d'agrégation** : calcul des moyennes pondérées par le nombre de logements des immeubles du rôle d'évaluation foncière compris dans les AD.
3. **Trois mesures d'accessibilité** : le supermarché le plus proche (en minutes), le nombre de supermarchés à 30 minutes et moins, la distance moyenne aux trois supermarchés les plus proches.
4. **Type de distance** : chemin le plus rapide à la marche.

Étape 1. Importation des trois couches géographiques.

```
## Unités d'évaluation foncière
Role <- st_read(dsn = "data/chap05/AutresDonnees/Role2022Sherb.gdb",
                layer = "rol_unite_Sherbrooke", quiet = TRUE)
## Aires de diffusion
AD <- st_read(dsn = "data/chap05/AutresDonnees/Commerces.gpkg",
                        layer = "AD_Sherbrooke", quiet=TRUE)
## Supermarchés
Supermarches <- st_read(dsn = "data/chap05/AutresDonnees/Commerces.gpkg",
                        layer = "supermarche", quiet=TRUE)
## Changement de projection
Supermarches <- st_transform(Supermarches, crs = 4326)
Role <- st_transform(Role, crs = 4326)
AD <- st_transform(AD, crs = 4326)
```

Étape 2. Réalisation d'une jointure spatiale pour attribuer à chaque unité d'évaluation l'identifiant de l'aire de diffusion (champ `ADIDU`) dans laquelle elle est comprise.

```
## Jointure spatiale entre le Role et les AD
Role <- st_join(Role, AD["ADIDU"], join = st_intersects)
Role <- subset(Role, is.na(ADIDU)==FALSE)
## Nombre de distances à calculer
nO = nrow(Role)
nD = nrow(Supermarches)
```

```
NOD = nO * nD
cat("Origines (O) :", nO, "îlots",
    "\n Destinations (D) :", nD, "supermarchés",
    "\n Distances OD = ", NOD)
```

```
Origines (O) : 46534 îlots
 Destinations (D) : 27 supermarchés
 Distances OD =  1256418
```

Étape 3. Création des points d'origine et de destination.

> ⚠ **Attention**
>
> **Rattacher les points aux tronçons de rue**
> Puisque nous utilisons le trajet le plus court à pied, les points d'origine et de destination doivent être rattachés à des tronçons de rues qui ne sont pas des autoroutes ou tout autre tronçon interdit à la marche. Pour ce faire, le *package* `R5R` dispose d'une fonction très intéressante :
> `find_snap(r5r_core, points, mode = "WALK")`.
> Sans le recours à cette fonction, un point d'origine ou de destination risque d'être rattaché à un tronçon autoroutier, faisant en sorte que le trajet ne pourra être calculé à la marche.

```
## Origines
Origines <- Role
Origines$lat <- st_coordinates(Origines)[,2]
Origines$lon <- st_coordinates(Origines)[,1]
Origines$id <- Origines$mat18
Origines <- find_snap(r5r_core, Origines, mode = "WALK")
Origines$lat <- Origines$snap_lat
Origines$lon <- Origines$snap_lon
Origines <- Origines[, c("point_id", "lat", "lon", "distance")]
names(Origines) <- c("id", "lat", "lon", "distance")
## Destinations
Destinations <- Supermarches
Destinations$lat <- st_coordinates(Destinations)[,2]
Destinations$lon <- st_coordinates(Destinations)[,1]
names(Destinations)[1] <- "id"
Destinations <- find_snap(r5r_core, Destinations, mode = "WALK")
Destinations$lat <- Destinations$snap_lat
Destinations$lon <- Destinations$snap_lon
Destinations <- Destinations[, c("point_id", "lat", "lon", "distance")]
names(Destinations) <- c("id", "lat", "lon", "distance")
```

Étape 4. Construction de la matrice origines-destinations avec la fonction `travel_time_matrix` et sauvegarde dans un fichier `Rdata`.

```
## Matrice OD à la marche
t1 <-Sys.time()
matriceOD.Marche <- travel_time_matrix(r5r_core = r5r_core,
                                       origins = Origines,
                                       destinations = Destinations,
                                       mode = "WALK",
                                       walk_speed = 4.5,  # valeur par défaut 3.6
                                       max_trip_duration = 240,
                                       max_walk_time = Inf)
t2 <-Sys.time()
duree.marche = as.numeric(difftime(t2, t1), units = "mins")
cat("Temps de calcul :", round(duree.marche,2), "minutes")
## Enregistrement des résultats dans un fichier Rdata
save(duree.marche, matriceOD.Marche,
     file="data/chap05/Resultats/MatricesMarcheRoleSupermarche.Rdata")
```

Étape 5. Calcul des trois mesures d'accessibilité pour les unités d'évaluation.

```
## Chargement du fichier Rdata
load("data/chap05/Resultats/MatricesMarcheRoleSupermarche.Rdata")
cat("Temps de calcul pour la matrice :", round(duree.marche,2), "minutes")
```

```
Temps de calcul pour la matrice : 4.53 minutes
```

```
## Supermarché le plus proche
Supermarche.PlusProche <- matriceOD.Marche %>%
  group_by(from_id) %>%
  summarise(
    plus_proche = min(travel_time_p50))

## Nombre de supermarchés à moins de 30 minutes
Supermarche.N30mins <- matriceOD.Marche %>%
  filter(travel_time_p50 <= 30) %>%
  group_by(from_id) %>%
  summarise(
    nb_30_min = n())

# Distance moyenne aux trois supermarchés les plus proches
Supermarche.3 <- matriceOD.Marche %>%
  group_by(from_id) %>%
  mutate(dist_rank = order(order(travel_time_p50, decreasing=FALSE))) %>%
  filter(dist_rank <= 3) %>%
  summarise(nb = n(),
            sum_dist = sum(travel_time_p50))
```

```
# Notez ici que nous pouvons avoir des origines pour lesquelles
# n'avons pas trois supermarchés à moins de 240 minutes.
# Pour ces quelques cas, nous ajoutons des temps de 240 minutes pour les
# origines avec des supermarchés manquants.
Supermarche.3$sum_dist <- Supermarche.3$sum_dist + 240 * (3 - Supermarche.3$nb)
Supermarche.3$mean_n3 <- Supermarche.3$sum_dist / 3
Supermarche.3$nb <- NULL
Supermarche.3$sum_dist <- NULL

# Nous pouvons à présent fusionner nos différent dataframes avec les
# indicateurs d'accessibilité
Role <- Role %>%
  left_join(Supermarche.PlusProche, by = c('mat18' = 'from_id')) %>%
  left_join(Supermarche.N30mins, by = c('mat18' = 'from_id')) %>%
  left_join(Supermarche.3, by = c('mat18' = 'from_id')) %>%
  rename(
      SupPlusProcheMin = 'plus_proche',
      SupN30min = 'nb_30_min',
      Moy3Sup = 'mean_n3'
  )

# Certaines observations n'ont pas de supermarchés à 30 minutes
# Nous mettons alors les valeurs à 0
Role$SupN30min[is.na(Role$SupN30min)] <- 0
```

Étape 6. Calcul des moyennes pondérées par le nombre de logements (champ `Logements`) pour les aires de diffusion avec le *package* `dplyr`.

```
library(dplyr)
## Création d'un DataFrame temporaire sans la géométrie
Role.Temp <- st_drop_geometry(Role)
## Moyennes pondérées pour les trois indicateurs d'accessibilité
MesureAcc <- Role.Temp %>%
                group_by(ADIDU) %>%
                summarize(
                  SupPlusProcheMin = weighted.mean(SupPlusProcheMin, Logements),
                  SupN30min = weighted.mean(SupN30min, Logements),
                  Moy3Sup = weighted.mean(Moy3Sup, Logements)
                )

## Fusion avec la couche des AD
AD <- left_join(AD, MesureAcc, by="ADIDU")
```

Étape 7. Cartographie des résultats avec le *package* `tmap`.

Tout d'abord, nous analysons les statistiques univariées pour repérer les valeurs minimales et maximales pour les trois mesures d'accessibilité avec la fonction `summary`.

```
TroisMesures <- c("SupPlusProcheMin","SupN30min","Moy3Sup")
summary(AD[, TroisMesures])
```

```
 SupPlusProcheMin    SupN30min        Moy3Sup                  geom
 Min.   :  4.528   Min.   :0.000   Min.   :  6.149   MULTIPOLYGON :254
 1st Qu.: 10.995   1st Qu.:1.000   1st Qu.: 17.416   epsg:4326    :  0
 Median : 16.244   Median :2.536   Median : 23.555   +proj=long...:  0
 Mean   : 21.903   Mean   :2.774   Mean   : 32.007
 3rd Qu.: 24.413   3rd Qu.:4.518   3rd Qu.: 33.837
 Max.   :130.615   Max.   :7.333   Max.   :139.209
 NA's   :2                         NA's   :2
```

Une fois les valeurs maximales et minimales analysées, réalisons les cartes (figure 5.16).

```
## Importation des arrondissements de la ville de Sherbrooke
arrondissements <- st_read(dsn = "data/Chap05/AutresDonnees/Arrondissements.shp",
                           quiet=TRUE)
## Construction des cartes
tmap_mode("plot")
# Carte pour les supermarchés
Carte0 <- tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_shape(Supermarches)+
              tm_dots(col="red", size=0.25)+
          tm_layout(frame = FALSE, legend.outside = TRUE,
                    legend.format = list(text.separator = "à"),
                    title = "Supermarché",
                    title.size = 1)
# Carte pour le supermarché le plus proche
max.acc1 <- max(AD$SupPlusProcheMin,na.rm=TRUE)
Carte1 <- tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_shape(AD)+
            tm_fill(col="SupPlusProcheMin",
                    border.lwd = 1,
                    style = "fixed",
                    breaks = c(0,10,20,30,40,60,max.acc1),
                    palette="-YlOrRd",
                    size = .2,
                              legend.format = list(text.separator = "à"),
                    textNA = "Sans données",
                              title="Plus proche")+
          tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_layout(frame = FALSE, legend.outside = TRUE)
# Carte pour le nombre de supermarchés à 30 minutes ou moins
max.acc2 <- max(AD$SupN30min,na.rm=TRUE)
Carte2 <- tm_shape(arrondissements)+tm_borders(lwd = 2)+
```

```
          tm_shape(AD)+
            tm_fill(col="SupN30min",
                    border.lwd = 1,
                    style = "fixed",
                    breaks = c(0,1,2,3,4,5,max.acc2),
                    palette="YlOrRd",
                    size = .2,
                              legend.format = list(text.separator = "à"),
                            textNA = "Sans données",
                    title="Sup. à 30 min")+
          tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_layout(frame = FALSE, legend.outside = TRUE)
# Carte pour la distance moyenne aux trois supermarchés les plus proches
max.acc3 <- max(AD$Moy3Sup,na.rm=TRUE)
Carte3 <- tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_shape(AD)+
            tm_fill(col="Moy3Sup",
                    border.lwd = 1,
                    style = "fixed",
                    breaks = c(5,10,20,30,40,60,max.acc3),
                    palette="-YlOrRd",
                    size = .2,
                            legend.format = list(text.separator = "à"),
                            textNA = "Sans données",
                    title="Moy. 3 plus proches")+
          tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_layout(frame = FALSE, legend.outside = TRUE)
## Figure avec les quatre cartes
tmap_arrange(Carte0, Carte1, Carte2, Carte3)
```

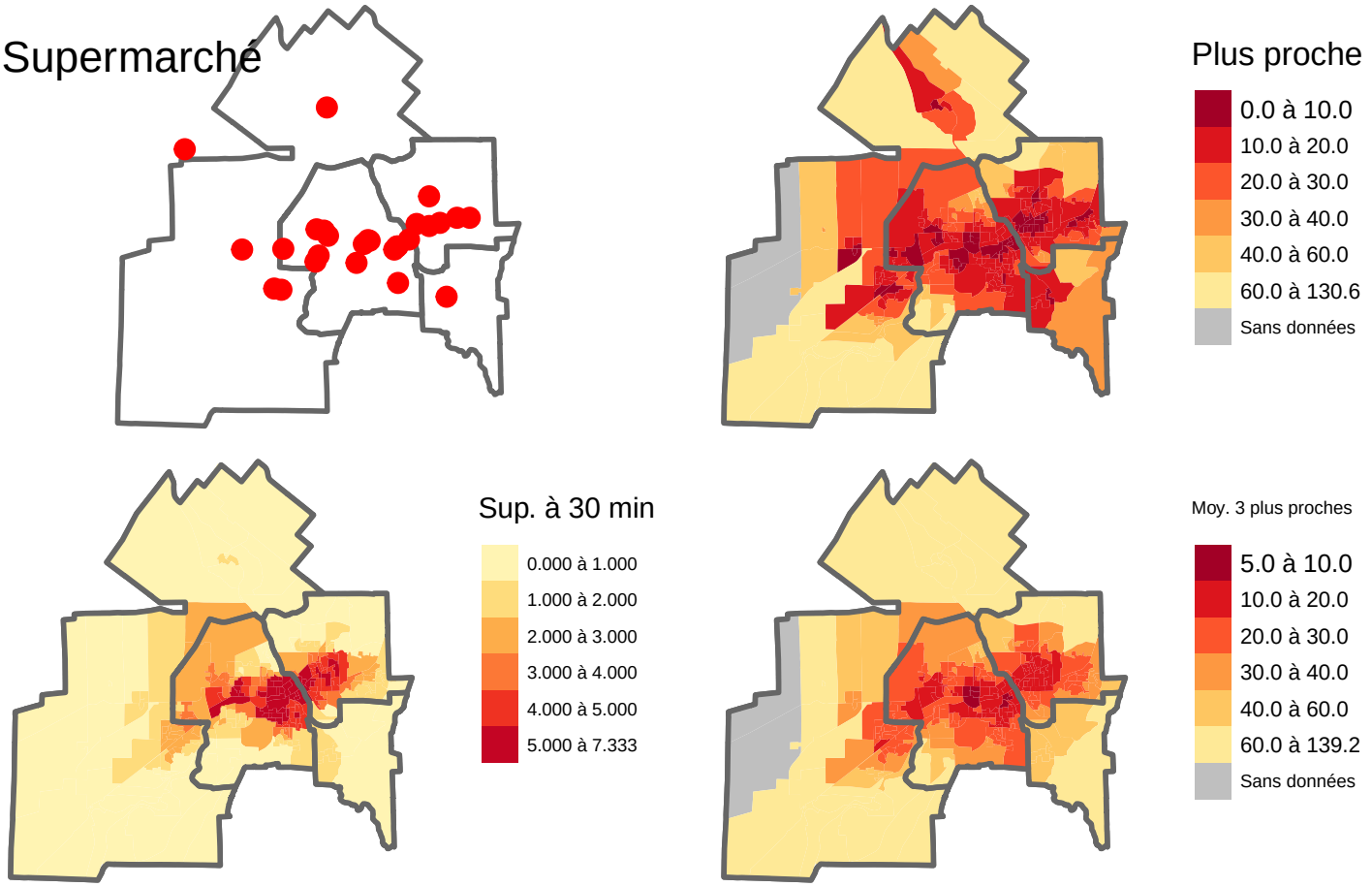


FIGURE 5.16 – Accessibilité spatiale potentielle à pied aux supermarchés (en minutes), aires de diffusion de la ville de Sherbrooke, 2021

5.4.2 Accessibilité spatiale potentielle aux patinoires extérieures

Dans ce second exemple applicatif dans R, nous élaborons un diagnostic de l'accessibilité spatiale potentielle aux patinoires extérieures dans la ville de Sherbrooke avec les quatre paramètres suivants :

1. **Unité spatiale de référence** : aires de diffusion (AD) de 2021 de la ville de Sherbrooke.
2. **Méthode d'agrégation** : calcul des moyennes pondérées par la population totale des îlots compris dans les AD.
3. **Deux mesures d'accessibilité** : patinoire la plus proche (en minutes); E2SFCA (*Enhanced two-step floating catchment area*), soit le nombre de patinoires pour 1000 habitants dans un rayon de 30 minutes de marche.
4. **Type de distance** : chemin le plus rapide à la marche.

Étape 1. Importation des trois couches géographiques.

```
## Unités d'évaluation foncière
Patinoires <- st_read(dsn = "data/chap05/AutresDonnees/Patinoires.shp",
                      quiet = TRUE)
## Aires de diffusion
```

```
AD <- st_read(dsn = "data/chap05/AutresDonnees/Commerces.gpkg",
              layer = "AD_Sherbrooke", quiet=TRUE)
## Ilots de recensements
Ilots <- st_read(dsn = "data/chap05/AutresDonnees/Commerces.gpkg",
                 layer = "Ilots", quiet=TRUE)
## Changement de projection
Patinoires <- st_transform(Patinoires, crs = 4326)
AD <- st_transform(AD, crs = 4326)
Ilots <- st_transform(Ilots, crs = 4326)
```

Étape 2. Réalisation d'une jointure spatiale pour attribuer à chaque îlot l'identifiant de l'aire de diffusion (champ `ADIDU`) dans laquelle il est compris.

```
## Jointure spatiale entre le Rôle et les AD
Ilots <- st_join(st_centroid(Ilots), AD["ADIDU"], join = st_intersects)
Ilots <- Ilots[,c("id","pop2021","ADIDU")]
```

Étape 3. Création des points d'origine et de destination.

```
## Origines
Origines <- Ilots
Origines$lat <- st_coordinates(Origines)[,2]
Origines$lon <- st_coordinates(Origines)[,1]
Origines <- find_snap(r5r_core, Origines, mode = "WALK")
Origines$lat <- Origines$snap_lat
Origines$lon <- Origines$snap_lon
Origines <- Origines[, c("point_id", "lat", "lon", "distance")]
names(Origines) <- c("id", "lat", "lon", "distance")
## Destinations
Destinations <- Patinoires
Destinations$lat <- st_coordinates(Destinations)[,2]
Destinations$lon <- st_coordinates(Destinations)[,1]
Destinations$id <- as.character(1:nrow(Destinations))
Destinations <- find_snap(r5r_core, Destinations, mode = "WALK")
Destinations$lat <- Destinations$snap_lat
Destinations$lon <- Destinations$snap_lon
Destinations <- Destinations[, c("point_id", "lat", "lon", "distance")]
names(Destinations) <- c("id", "lat", "lon", "distance")
```

Étape 4. Construction de la matrice origines-destinations avec la fonction `travel_time_matrix` et sauvegarde dans un fichier `Rdata`.

```
## Matrice OD à la marche
t1 <-Sys.time()
matriceODPatinoire.Marche <- travel_time_matrix(r5r_core = r5r_core,
```

```
                                                origins = Origines,
                                                destinations = Destinations,
                                                mode = "WALK",
                                                walk_speed = 4.5,
                                                max_trip_duration = 240,
                                                max_walk_time = Inf)
t2 <-Sys.time()
duree.marche = as.numeric(difftime(t2, t1), units = "mins")
cat("Temps de calcul :", round(duree.marche,2), "minutes")
## Enregistrement des résultats dans un fichier Rdata
save(duree.marche, matriceODPatinoire.Marche,
     file="data/chap05/Resultats/matriceODPatinoire.Rdata")
```

Étape 5. Calcul des deux mesures d'accessibilité pour les îlots.

```
## Chargement du fichier Rdata
load("data/chap05/Resultats/matriceODPatinoire.Rdata")
cat("Temps de calcul pour la matrice :", round(duree.marche,2), "minutes")
```

```
Temps de calcul pour la matrice : 0.37 minutes
```

Le code ci-dessous permet de calculer la distance à la patinoire la plus proche pour les aires de diffusion.

```
## Patinoire la plus proche
Patinoire.PlusProche <- matriceODPatinoire.Marche %>%
  group_by(from_id) %>%
  summarise(PatinoirePlusProche = min(travel_time_p50))
## Fusion avec la couche des îlots
Ilots <- left_join(Ilots, Patinoire.PlusProche, by=c("id" = 'from_id'))
```

Puis, nous calculons la version de la méthode du E2SFCA avec une fonction de gradient continue (McGrail et Humphreys 2009).

```
source("code_complementaire/E2SFCA.R")
## Ajout des champs de population dans la matrice
TempIlots <- st_drop_geometry(Ilots)
matriceODPatinoire <- merge(matriceODPatinoire.Marche,
                  TempIlots[,c("id","pop2021")],
                  by.x = "from_id", by.y="id")
matriceODPatinoire$Wd <- 1
names(matriceODPatinoire) <- c("from_id", "to_id", "Marche", "Wo", "Wd")
head(matriceODPatinoire, n=2)
```

```
Key: <from_id>
       from_id  to_id Marche    Wo    Wd
        <char> <char>  <int> <num> <num>
1: 24430011001      1     90   180     1
2: 24430011001      2    159   180     1
```

```
# Utilisation de la pondération de gradient continu du E2SFCA
Wfun <- function(x){
  w <- ifelse(x < 5, 1,((30 - x) / (30 - 5))**1.5)
  w[x  > 30] <- 0
  return(w)
}
# Calcul du résultat en milliers d'habitants
matriceODPatinoire$Wo <- matriceODPatinoire$Wo / 1000
MesureE2SFCA <- GTSFCA(dist_mat = matriceODPatinoire,
                       Wfun = Wfun,
                       IDorigine = "from_id",
                       IDdestination = "to_id",
                       CoutDistance = "Marche",
                       Wo = "Wo",
                       Wd = "Wd",
                       ChampSortie = "E2SFCA_G")

MesureE2SFCA$E2SFCA_G[is.na(MesureE2SFCA$E2SFCA_G)] <- 0
Ilots <- merge(Ilots, MesureE2SFCA, by.x ="id", by="from_id", all.x = TRUE)
Ilots$E2SFCA_G[is.na(Ilots$E2SFCA_G)] <- 0
```

Étape 6. Calcul des moyennes pondérées par la population des îlots (champ `pop2021`) pour les aires de diffusion avec le *package* `dplyr`.

```
## Création d'un DataFrame temporaire sans la géométrie
Ilots.Temp <- st_drop_geometry(Ilots)
## Moyenne pondérées pour la patinoire la plus proche
MesureAcc1 <- Ilots.Temp %>%
                group_by(ADIDU) %>%
                summarize(PatinoirePlusProche = weighted.mean(PatinoirePlusProche, pop2021))
## Moyenne non pondérée pour le E2SFCA, car la population est déjà prise en compte
MesureAcc2 <- aggregate(E2SFCA_G ~ ADIDU, Ilots.Temp, FUN = mean)
## Fusion avec la couche des îlots
AD <- merge(AD, MesureAcc1, by="ADIDU")
AD <- merge(AD, MesureAcc2, by="ADIDU")
```

Étape 7. Cartographie des résultats avec le *package* `tmap`.

Tout d'abord, nous analysons les statistiques univariées pour repérer les valeurs minimales et maximales pour les trois mesures d'accessibilité.

```
DeuxMesures <- c("PatinoirePlusProche","E2SFCA_G")
summary(AD[, DeuxMesures])
```

```
 PatinoirePlusProche    E2SFCA_G                  geom
 Min.   : 1.00       Min.   :0.00000   MULTIPOLYGON :254
 1st Qu.: 9.28       1st Qu.:0.09836   epsg:4326    :  0
 Median :14.50       Median :0.19188   +proj=long...:  0
 Mean   :17.29       Mean   :0.22520
 3rd Qu.:19.83       3rd Qu.:0.30166
 Max.   :85.58       Max.   :1.26495
 NA's   :2
```

Une fois avoir pris connaissance des valeurs maximales et minimales, nous pouvons réaliser les cartes (figure 5.17).

```
## Importation des arrondissements de la ville de Sherbrooke
arrondissements <- st_read(dsn = "data/Chap05/AutresDonnees/Arrondissements.shp",
                           quiet=TRUE)
## Construction des cartes
tmap_mode("plot")
# Carte pour les patinoires
Carte0p <- tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_shape(Patinoires)+
              tm_dots(col="red", size=0.25)+
          tm_layout(frame = FALSE, legend.outside = TRUE,
                    title = "Patinoire extérieure",
                    title.size = 1)
# Carte pour la patinoire la plus proche
max.acc1 <- max(AD$PatinoirePlusProche,na.rm=TRUE)
Carte1p <- tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_shape(AD)+
            tm_fill(col="PatinoirePlusProche",
                    border.lwd = 1,
                    style = "fixed",
                    breaks = c(0,10,20,30,40,60,max.acc1),
                    palette="-YlOrRd",
                    size = .2,
                    legend.format = list(text.separator = "à"),
                    textNA = "Sans données",
                    title="Plus proche (min)")+
          tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_layout(frame = FALSE, legend.outside = TRUE)
# Carte pour le 2ESFCA
AD2 <- subset(AD, E2SFCA_G !=0)
min.acc2 <- min(AD2$E2SFCA_G,na.rm=TRUE)
max.acc2 <- max(AD2$E2SFCA_G,na.rm=TRUE)
```

```
Carte2p <- tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_shape(AD)+tm_fill(col="gray")+
          tm_shape(AD2)+
            tm_fill(col="E2SFCA_G",
                    border.lwd = 1,
                    style = "fixed",
                    breaks = c(min.acc2,0.25,0.50,0.75,1,max.acc2),
                    palette="YlOrRd",
                    size = .2,
                    legend.format = list(text.separator = "à"),
                    title="Patinoire pour 1000 hab.")+
          tm_shape(arrondissements)+tm_borders(lwd = 2)+
          tm_layout(frame = FALSE, legend.outside = TRUE)
## Figure avec les trois cartes
tmap_arrange(Carte0p, Carte1p, Carte2p,
             nrow=2, ncol=2)
```

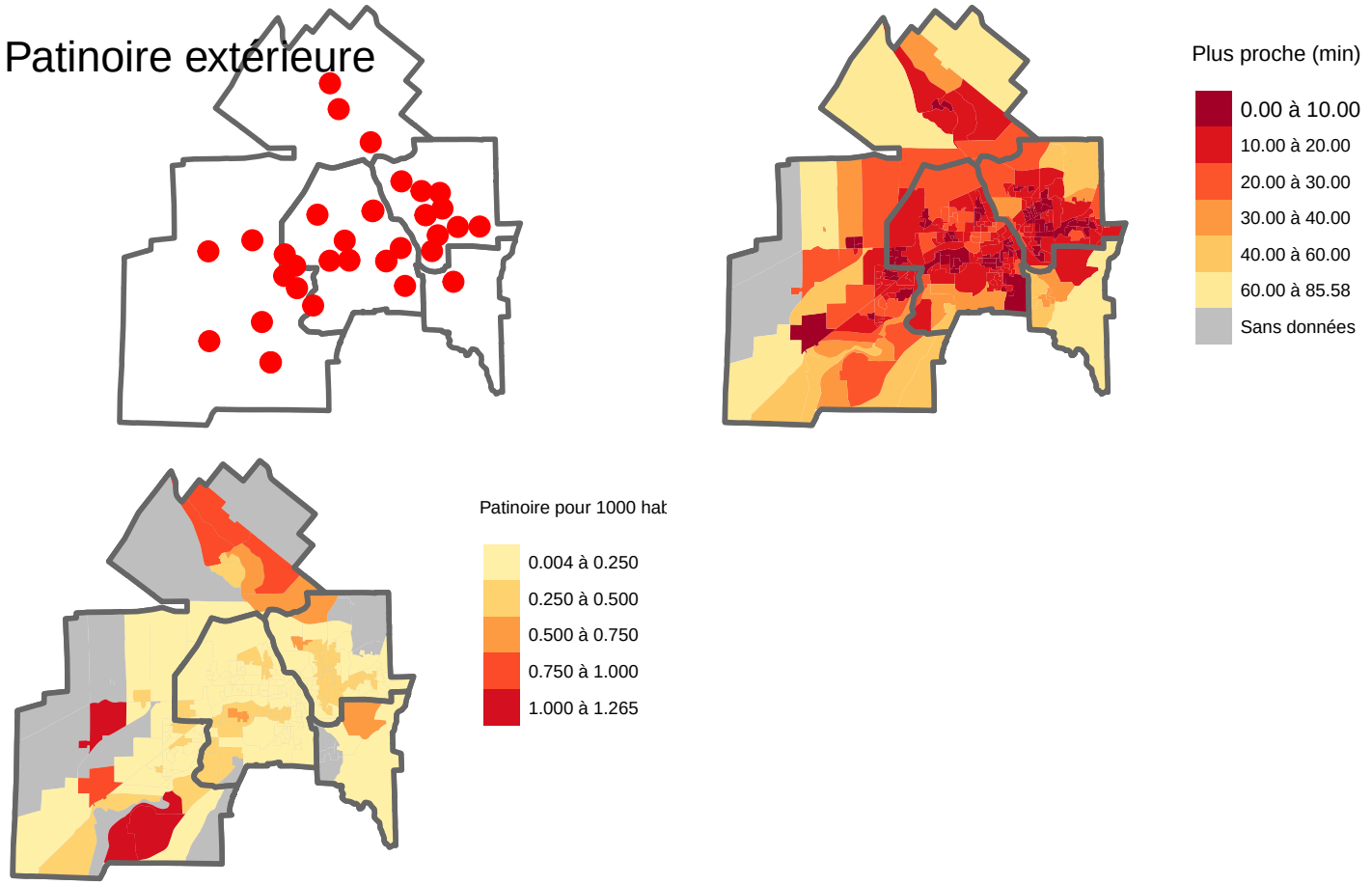


FIGURE 5.17 – Accessibilité spatiale potentielle à pied aux patinoires extérieures, aires de diffusion de la ville de Sherbrooke, 2021

⚠ **Attention**

Libération de la mémoire allouée à JAVA
Une fois les calculs avec `R5R` terminés, il convient de détruire l'objet `r5r_core` et d'arrêter le processus JAVA avec les deux lignes de code ci-dessous.

```
r5r::stop_r5(r5r_core)
rJava::.jgc(R.gc = TRUE)
```

5.5 Quiz de révision du chapitre

Questions

- **Quels sont les trois principaux problèmes résolus en analyse de réseau?**
  - Trouver le trajet le plus court ou le plus rapide entre deux points (algorithme de Dijkstra)
  - Trouver la route optimale comprenant plusieurs arrêts (problème du voyageur de commerce)
  - Définir des zones de desserte autour d'une origine (algorithme de Dijkstra)
  - Mesurer la dépendance spatiale

  Relisez au besoin la section 5.1.2.
- **Quels sont les quatre autres problèmes résolus en analyse de réseau?**
  - Trouver les k services les plus proches à partir d'une origine
  - Construire une matrice de distance origines-destinations
  - Répartir aléatoirement des points dans un territoire donné
  - Résoudre le problème de tournées de véhicules
  - Réaliser un modèle localisation-affectation

  Relisez au besoin la section 5.1.2.
- **Quels sont les trois fichiers nécessaires pour construire un réseau multimode avec R5R?**
  - Un fichier pbf pour les données d'OpenStreetMap
  - Un ou plusieurs fichiers GTFS
  - Un fichier pour les bâtiments
  - Un fichier GeoTIFF d'élévation

  Relisez au besoin le début de la section 5.1.2.
- **Quelle fonction de R5R permet de construire un réseau multimode?**
  - detailed_itineraries()
  - setup_r5()
  - travel_time_matrix()

  Relisez au besoin la section 5.2.2.
- **Quelle fonction de R5R permet de construire un itinéraire?**
  - detailed_itineraries()
  - setup_r5()
  - travel_time_matrix()

Relisez au besoin la section 5.2.3.

- **Quelle fonction de R5R permet de construire une matrice origine-destination?**
  - detailed_itineraries()
  - setup_r5()
  - travel_time_matrix()

Relisez au besoin la section 5.2.5.

- **Pour évaluer l'accessibilité selon la conceptualisation de la proximité immédiate, quelle mesure d'accessibilité utilisez-vous?**
  - Distance entre l'origine et le service le plus proche
  - Nombre de services présents à moins de n mètres ou minutes
  - Distance moyenne entre une origine et n services
  - Modèles gravitaires et méthodes two-step floating catchment area (2SFCA)

Relisez au besoin la section 5.3.2.3.

- **Pour évaluer l'accessibilité selon la conceptualisation de l'offre et la demande, quelle mesure d'accessibilité utilisez-vous?**
  - Distance entre l'origine et le service le plus proche
  - Nombre de services présents à moins de n mètres ou minutes
  - Distance moyenne entre une origine et n services
  - Modèles gravitaires et méthodes two-step floating catchment area (2SFCA)

Relisez au besoin la section 5.3.2.3.

- **Pour évaluer l'accessibilité selon l'offre de services dans l'environnement immédiat, quelle mesure d'accessibilité utilisez-vous?**
  - Distance entre l'origine et le service le plus proche
  - Nombre de services présents à moins de n mètres ou minutes
  - Distance moyenne entre une origine et n services
  - Modèles gravitaires et méthodes two-step floating catchment area (2SFCA)

Relisez au besoin la section 5.3.2.3.

Réponses

- Quels sont les trois principaux problèmes résolus en analyse de réseau?
  - Trouver le trajet le plus court ou le plus rapide entre deux points (algorithme de Dijkstra)
  - Trouver la route optimale comprenant plusieurs arrêts (problème du voyageur de commerce)
  - Définir des zones de desserte autour d'une origine (algorithme de Dijkstra)
- Quels sont les quatre autres problèmes résolus en analyse de réseau?
  - Trouver les k services les plus proches à partir d'une origine
  - Construire une matrice de distance origines-destinations
  - Résoudre le problème de tournées de véhicules
  - Réaliser un modèle localisation-affectation
- Quels sont les trois fichiers nécessaires pour construire un réseau multimode avec R5R?
  - Un fichier pbf pour les données d'OpenStreetMap
  - Un ou plusieurs fichiers GTFS
  - Un fichier GeoTIFF d'élévation
- Quelle fonction de R5R permet de construire un réseau multimode?

  - setup_r5()
- Quelle fonction de R5R permet de construire un itinéraire?
  - detailed_itineraries()
- Quelle fonction de R5R permet de construire une matrice origine-destination?
  - travel_time_matrix()
- Pour évaluer l'accessibilité selon la conceptualisation de la proximité immédiate, quelle mesure d'accessibilité utilisez-vous?
  - Distance entre l'origine et le service le plus proche
- Pour évaluer l'accessibilité selon la conceptualisation de l'offre et la demande, quelle mesure d'accessibilité utilisez-vous?
  - Modèles gravitaires et méthodes two-step floating catchment area (2SFCA)
- Pour évaluer l'accessibilité selon l'offre de services dans l'environnement immédiat, quelle mesure d'accessibilité utilisez-vous?
  - Nombre de services présents à moins de n mètres ou minutes

5.6 Exercices de révision

Exercice

Exercice 1. Calcul de trajets selon différents modes de transport
Complétez le code ci-dessous pour réaliser les étapes suivantes :

1. Construisez un réseau R5R pour la région de Laval avec un fichier OMS (pbf), un fichier d'élévation et un fichier GTFS.
2. Créez deux points : l'un pour la station de métro Morency (-73.7199, 45.5585) l'autre pour une adresse fictive (-73.7183, 45.5861).
3. Calculez les trajets en automobile, à vélo, à pied, et en transport en commun de l'adresse vers la station de métro et l'inverse avec (10 points) :
   — Une vitesse de 15 km/h pour le vélo.
   — Une vitesse de 4,5 km/h pour la marche.
   — Un trajet aller le 12-02-2024 à 8h de l'adresse vers la station de métro.
   — Un trajet retour le 12-02-2024 à 18h de la station de métro vers l'adresse.
4. Réalisez deux figures :
   — Une figure avec quatre cartes des trajets aller (marche, vélo, auto, transport en commun).
   — Une figure avec quatre cartes des trajets retour (marche, vélo, auto, transport en commun).
5. Arrêtez java.

```
library(sf)
library(tmap)
library(r5r)

setwd("data/chap05/Laval")
rJava::.jinit()
options(java.parameters = "-Xmx2G")

# 1. Construction du réseau
dossierdata <- paste0(getwd(),"/_DataReseau")
list.files(dossierdata)
r5r_core <- setup_r5(à compléter)

# 2. Création de deux points
Pts <- data.frame(id = c("Station Morency", "Adresse 1"),
                  lon = c(à compléter),
                  lat = c(à compléter))
Pts <- st_as_sf(Pts, coords = c("lon","lat"), crs = 4326)
StationMorency <- Pts[1,]
Adresse1 <- Pts[2,]

## 2.1. Trajets en automobile
Auto.1 <- detailed_itineraries(r5r_core = r5r_core,
                               origins = Adresse1,
                               destinations = StationMorency,
                               mode = "CAR",
                               shortest_path = FALSE,
                               drop_geometry = FALSE)
Auto.2 <- detailed_itineraries(r5r_core = r5r_core,
                              origins = StationMorency,
                              destinations = Adresse1,
                              mode = "CAR",
```

Exercice

Exercice 2. Calcul d'isochrones
Complétez le code ci-dessous pour réaliser les étapes suivantes :

1. Calculez des isochrones à pied de 5, 10 et 15 minutes.
2. Calculez des isochrones à vélo de 5, 10 et 15 minutes.
3. Cartographiez les résultats.

```
library(sf)
library(tmap)
library(r5r)

## Construction du réseau
setwd("data/chap05/Laval")
rJava::.jinit()
options(java.parameters = "-Xmx2G")
dossierdata <- paste0(getwd(),"/_DataReseau")
list.files(dossierdata)
r5r_core <- setup_r5(data_path = dossierdata,
                     elevation = "TOBLER",
                     verbose = FALSE, overwrite = FALSE)

## Point pour la Station Morency
StationMorency <- data.frame(id = "Station Morency",
                             lon = -73.7199,
                             lat = 45.5585,  45.5861)
StationMorency <- st_as_sf(StationMorency,
                           coords = c("lon","lat"), crs = 4326)

# 1. Calcul d'isochrones à pied de 5, 10 et 15 minutes
Iso.Marche <- isochrone(à compléter)
# 1.2. Isochrone à vélo de 5, 10 et 15 minutes
Iso.Velo <- isochrone(à compléter)

# 3. Cartographie les résultats
tmap_mode("view")
tmap_options(check.and.fix = TRUE)
Carte.Marche <- tm_shape(à compléter)+
                    tm_fill(à compléter)+
                tm_shape(StationMorency)+tm_dots(col="darkred", size = .25)

Carte.Velo <- tm_shape(à compléter)+
                    tm_fill(à compléter)+
                tm_shape(StationMorency)+tm_dots(col="darkred", size = .25)

tmap_arrange(Carte.Marche, Carte.Velo, ncol = 2)

# 4. Arrêt de java
r5r::stop_r5(r5r_core)
rJava::.jgc(R.gc = TRUE)
```


Correction à la section 12.5.2.

6 Analyses d'évènements localisés sur un réseau

Dans le chapitre 3, nous avons décrit des méthodes de répartition ponctuelle qui s'appliquent à des espaces classiques en deux dimensions, homogènes et sans limites dans toutes les directions. Par contre, lorsque les évènements (points) ne peuvent se produire que le long des lignes d'un réseau, les hypothèses de base des méthodes de répartition ponctuelle ne tiennent plus et produisent alors des résultats biaisés. Par conséquent, dans ce chapitre, nous abordons des méthodes d'analyse de répartition ponctuelle spécifiques à des évènements localisés sur un réseau.

Package

Liste des *packages* utilisés dans ce chapitre

- Pour importer et manipuler des fichiers géographiques :
  - `sf` pour importer et manipuler des données vectorielles.
  - `lubridate` pour manipuler des champs de format date.
- Pour construire des cartes et des graphiques :
  - `tmap` pour construire des cartes thématiques.
  - `ggplot2` pour construire des graphiques.
  - `classInt` pour définir des intervalles sur une variable continue.
  - `viridis` pour des palettes de couleurs.
- Pour les analyses de méthodes de répartition ponctuelle sur un réseau :
  - `spNetwork` dédié aux analyses spatiales sur un réseau.
  - `future` pour accélérer les calculs de `spNetwork`.
  - `spdep` pour calculer des indices d'autocorrélation spatiale.
  - `dbscan` pour l'algorithme DBSCAN.

6.1 Pourquoi recourir à un réseau pour des méthodes d'analyse de répartition ponctuelle?

Les méthodes classiques d'analyse de répartition ponctuelle postulent que l'espace analysé est le plus souvent en deux dimensions, homogène et sans limite dans toutes les directions. Toutefois, des phénomènes se produisent dans des espaces pour lesquels ces hypothèses sont totalement invalidées, menant à l'obtention de résultats biaisés, voire incohérents. C'est notamment le cas d'évènements localisés le long des segments d'un réseau qui sont par exemple :

- Des accidents de la route se produisent nécessairement le long des axes routiers.
- Des fuites d'eau se produisent le long des canalisations d'une ville.
- Des interruptions de service de bus se produisent le long de lignes de transport collectif.
- Certaines espèces d'oiseaux nichent systématiquement le long de cours d'eau.

Contrairement à un espace géographique classique en deux dimensions, il est possible de considérer qu'un réseau géographique (c'est-à-dire un réseau dont les nœuds et les lignes ont des coordonnées spatiales) est un espace en 1,5

dimension puisqu'il n'est possible de se déplacer que le long des lignes et de ne changer de direction qu'au niveau d'un nœud (Steenberghen, Aerts et Thomas 2010). Cette distinction pose trois problèmes principaux si nous utilisons les méthodes de répartition ponctuelle classiques.

Premièrement, la distance euclidienne (à vol d'oiseau) tend systématiquement à sous-estimer la distance réelle entre deux points. En effet, la longueur d'un trajet sur un réseau est toujours plus grande ou égale à la distance euclidienne. La figure 6.1 illustre ce premier problème avec un cas simple comprenant trois points sur un réseau.

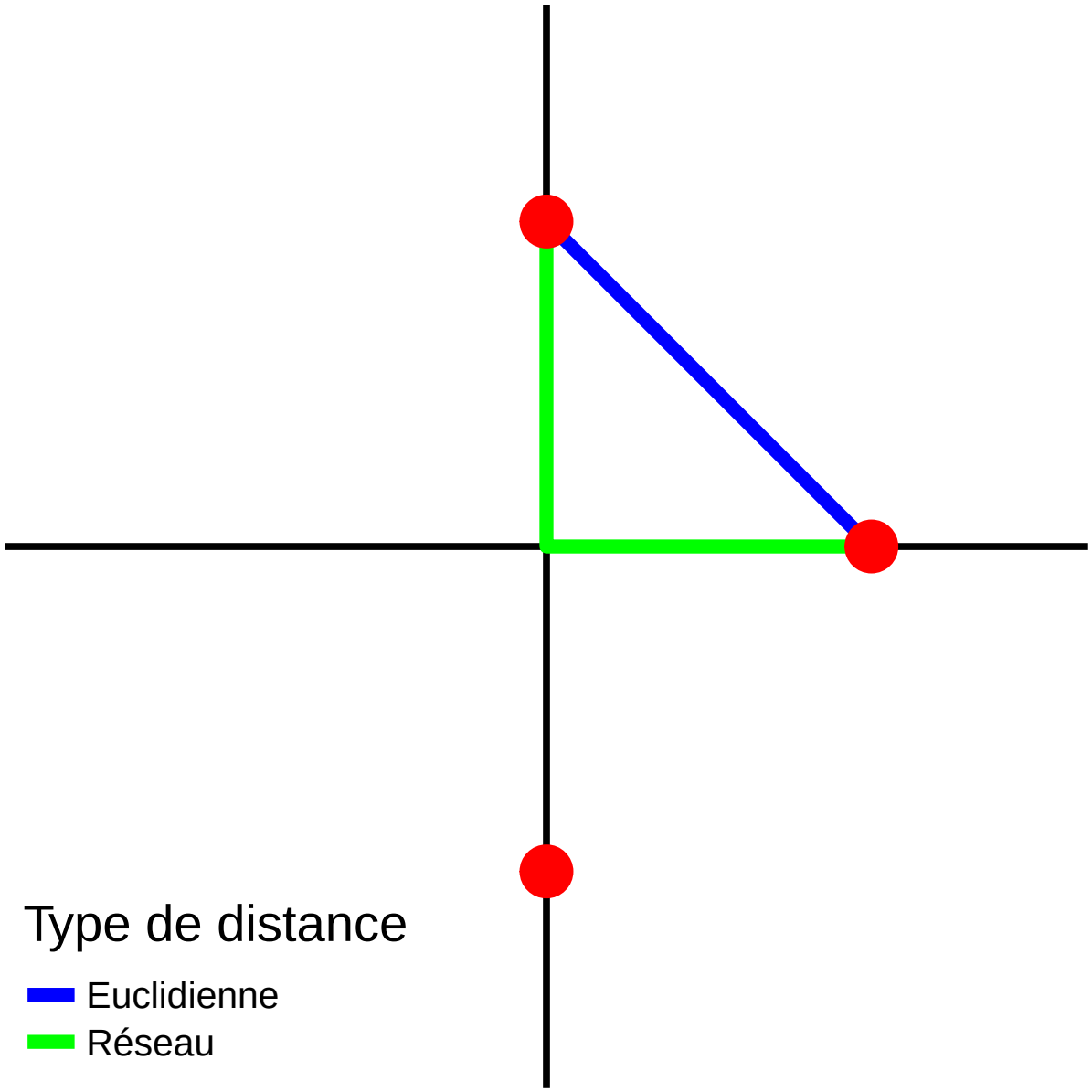


FIGURE 6.1 – Problèmes générés par la non-prise en compte du réseau : sous-estimation des distances

Deuxièmement, la non-prise en compte des lignes du réseau peut mener à analyser des secteurs dans lesquels les évènements ne peuvent pas se produire. La figure 6.2 illustre un cas où nous tenterions de produire un nouveau jeu de points distribués aléatoirement (points bleus), mais en respectant la densité initiale des points réels (points rouges). Ce type de méthode est notamment utilisé pour déterminer si un ensemble de points est plus ou moins concentré

comparativement à ce que le hasard produirait. Comparer des points sur le réseau à des points hors du réseau conduit à surévaluer la concentration des points sur le réseau, car il y a plus d'espace en dehors du réseau que sur le réseau.

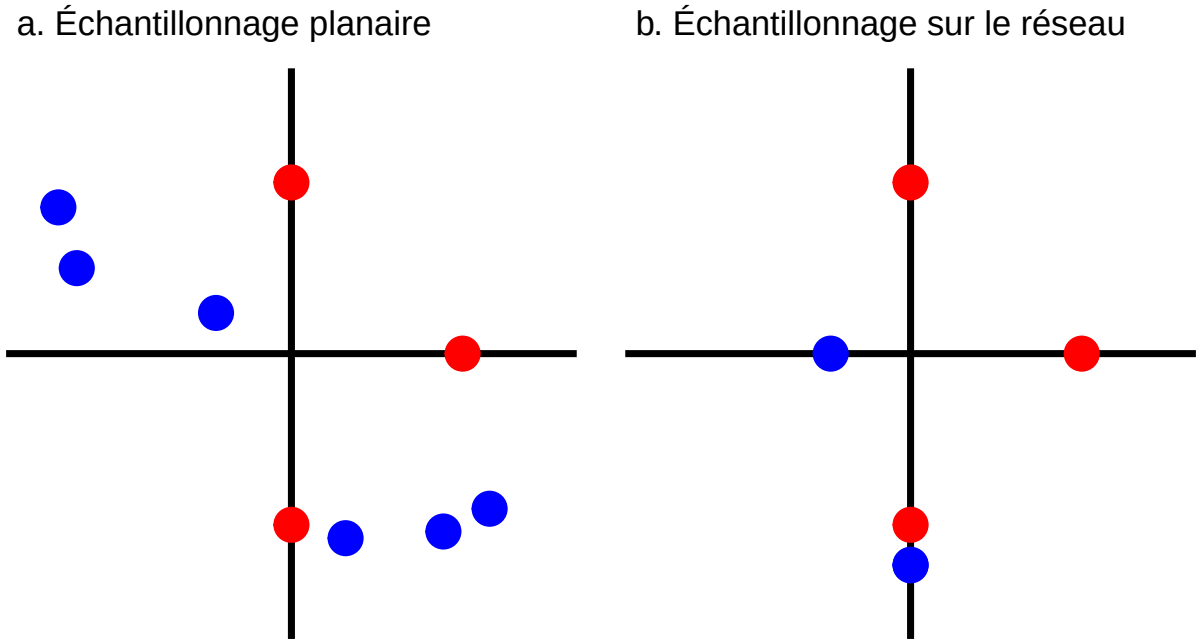


FIGURE 6.2 – Problèmes générés par la non-prise en compte du réseau : surestimation de la concentration des points

Troisièmement, la masse des évènements ne se propage pas dans un espace en 2D comme dans un espace en 1,5D (figure 6.3). Dans un réseau, la masse des évènements doit être divisée aux intersections. Si cette division n'est pas prise en compte, alors la masse des évènements est dupliquée aux intersections. Cette problématique se pose particulièrement aux méthodes d'estimation de densité par noyau que nous décrirons plus tard dans ce chapitre.

De nombreuses méthodes d'analyse de répartition ponctuelle ont été adaptées pour pouvoir être appliquées avec des réseaux géographiques (Okabe et Sugihara 2012). Dans ce chapitre, nous abordons les méthodes de densité par noyau sur un réseau (*network kernel density estimation*, NKDE) et la création de matrices de pondération spatiale avec des distances sur un réseau pour calculer des mesures d'autocorrélation spatiale.

6.2 Cartographie de la densité d'évènements sur un réseau

À la section 3.4.2, nous avons décrit les méthodes d'analyse de densité dans une maille régulière : carte de chaleur ou estimation de la densité par noyau (*kernel density estimation* – KDE). Pour un rappel, une KDE peut être utilisée pour tenter de reconstruire un processus spatial produisant des évènements. Le processus spatial en lui-même est impossible à mesurer, mais nous tentons de le reconstruire et de l'approximer en nous basant sur les évènements observés qui sont des réalisations du processus sous-jacent. Sur un réseau, la logique est exactement la même : un processus spatial qui est invisible conduit à la réalisation d'évènements le long des lignes d'un réseau géographique.

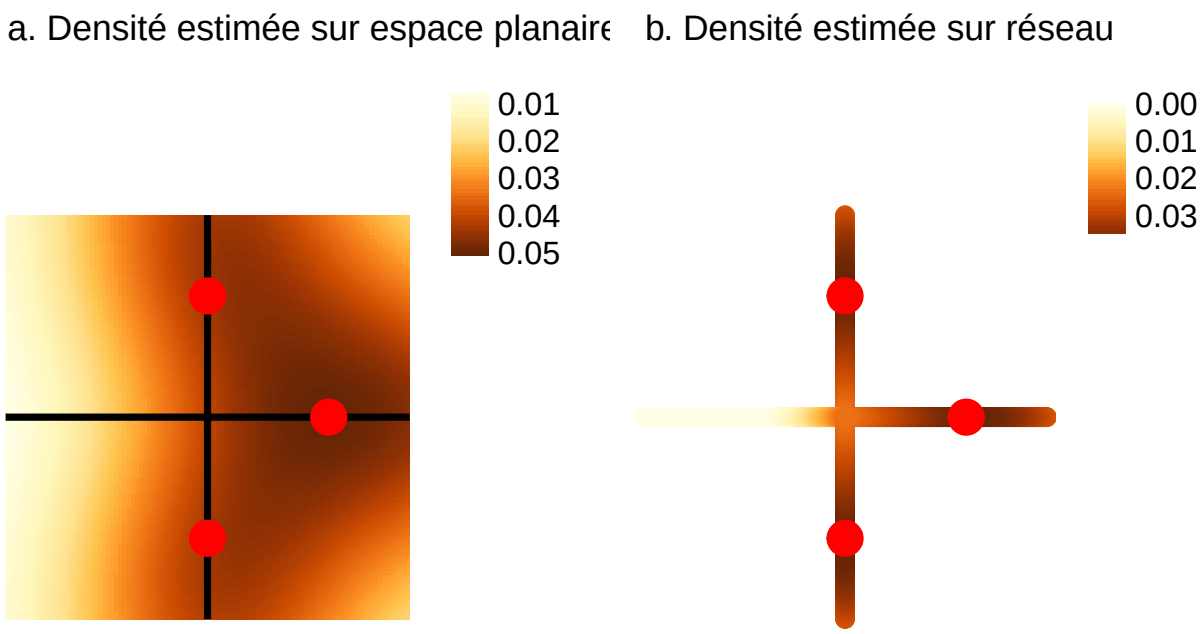


FIGURE 6.3 – Problèmes générés par la non-prise en compte du réseau : la masse des évènements

6.2.1 Estimation de la densité des points sur un réseau

L'estimation de la densité sur un réseau (*Network Kernel Density Estimation* – NKDE) utilise une approche similaire à la KDE (section 3.4.2). L'idée générale est de répartir la masse des évènements le long des lignes du réseau autour de chaque évènement et d'additionner ensuite ces densités pour obtenir une estimation locale de l'intensité du processus spatial générant ces évènements. Comparativement à la KDE, les spécificités de la NKDE sont les suivantes :

- Les intensités sont calculées non pas sur des pixels, mais sur leurs équivalents appelés **lixels**. Un lixel correspond à simplement une portion de segment de ligne du réseau d'une longueur déterminée (50 mètres par exemple).
- Les distances sont calculées sur le réseau et non à vol d'oiseau.

Comme pour la KDE, il faut déterminer la valeur du rayon d'influence (*bandwidth*) et choisir une fonction *kernel*.

6.2.1.1 Trois formes de NKDE

La NKDE peut prendre trois formes différentes traitant différemment la répartition de la masse aux intersections dans un réseau.

NKDE géographique (Geo-NKDE). Il s'agit du cas le plus simple, car aucun traitement particulier n'est réalisé aux intersections du réseau et la densité d'un lixel est simplement basée sur la distance entre le centre de ce lixel et un évènement. Cette méthode est intuitive et peu coûteuse en temps de calcul, mais elle produit des résultats biaisés. En effet, si la masse d'un évènement se propage de façon continue dans toutes les directions à une intersection, alors la masse est multipliée par le nombre de directions possibles. Par exemple, à la figure 6.4, qui comprend une intersection avec trois segments connectés, la masse totale est égale à 150 % et non à 100 %. Ainsi, dans un réseau avec de nombreuses intersections, l'intensité est systématiquement surestimée avec cette méthode.

Un aperçu en 3D de la répartition de la masse est disponible à la figure ci-dessous avec un seul évènement.

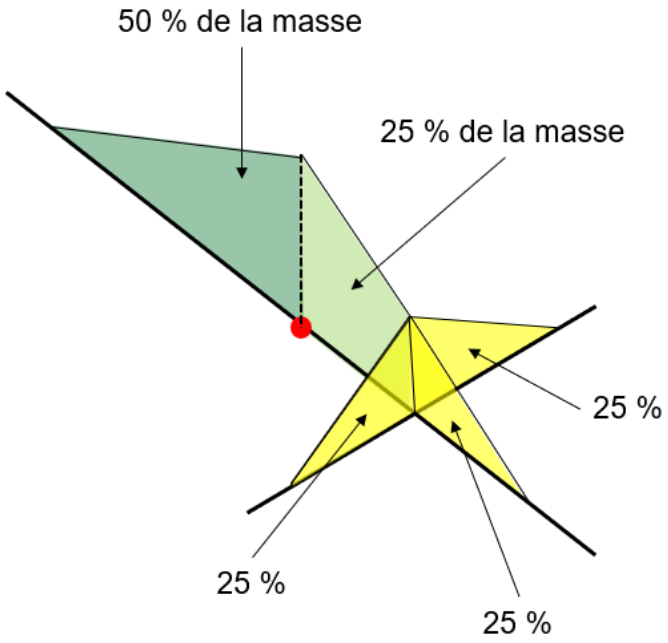


FIGURE 6.4 – Répartition de la masse avec une NKDE géographique

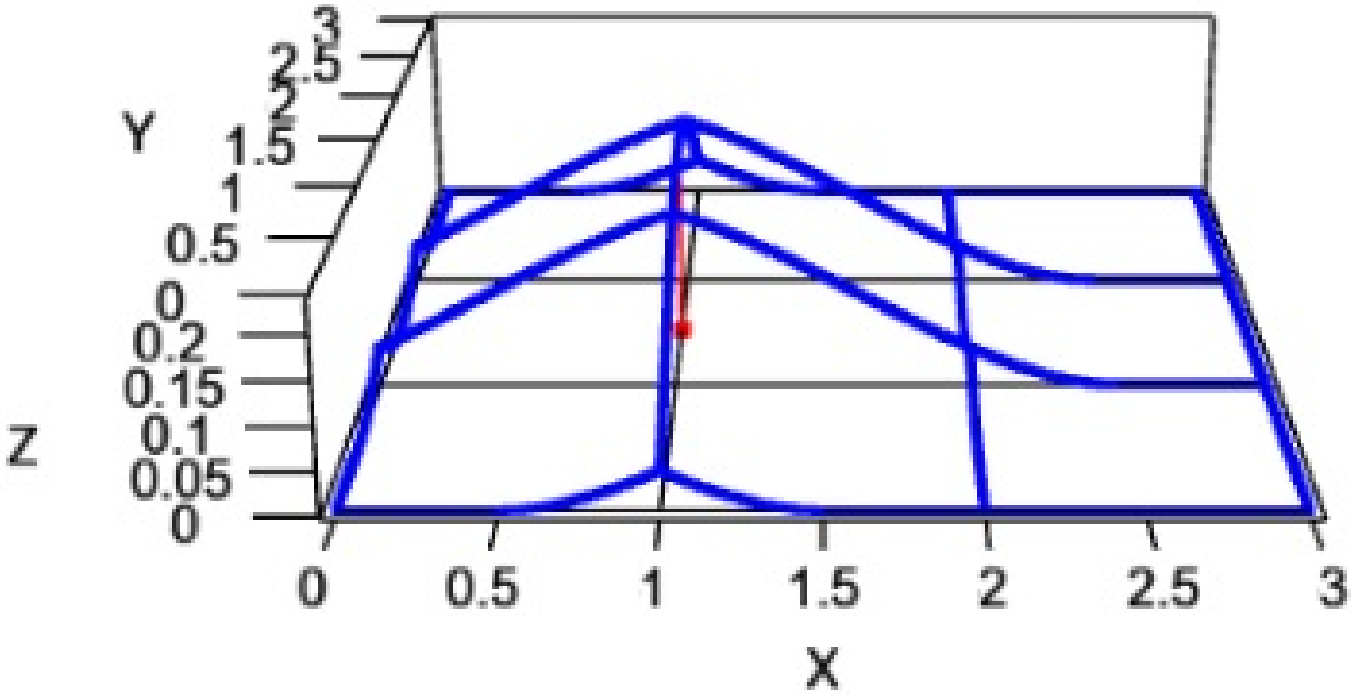


FIGURE 6.5 – Visualisation du Geo-NKDE

La formule pour calculer la Geo-NKDE est :

$$\hat{\lambda}_h(u) = \frac{1}{h} \sum_{i=1}^{N} k\left(\frac{\text{dist}_{\text{net}}\ (u, e_i)}{h}\right) \text{ avec :} \tag{6.1}$$

- $\hat{\lambda}_h(u)$, l'estimation de l'intensité au point u avec la *bandwidth* h.
- N, le nombre d'évènements.
- k, une fonction *kernel*.
- $\text{dist}_{\text{net}}\ (u, e_i)$, la distance réseau entre la localisation u et l'évènement e_i.

NKDE discontinue (ESD-NKDE). Cette seconde méthode impose que la masse des évènements soit divisée aux intersections par le nombre de directions possible. En procédant ainsi, il est possible d'éviter le biais de la Geo-NKDE, mais cela conduit à une estimation discontinue de la NKDE. En effet, l'intensité d'un évènement chute fortement au détour d'une intersection ce qui est contre-intuitif en géographie bien que l'intensité produite soit non biaisée (voir la figure ci-dessous). Comme pour la précédente, le second avantage de cette méthode est qu'elle est peu coûteuse en temps de calcul.

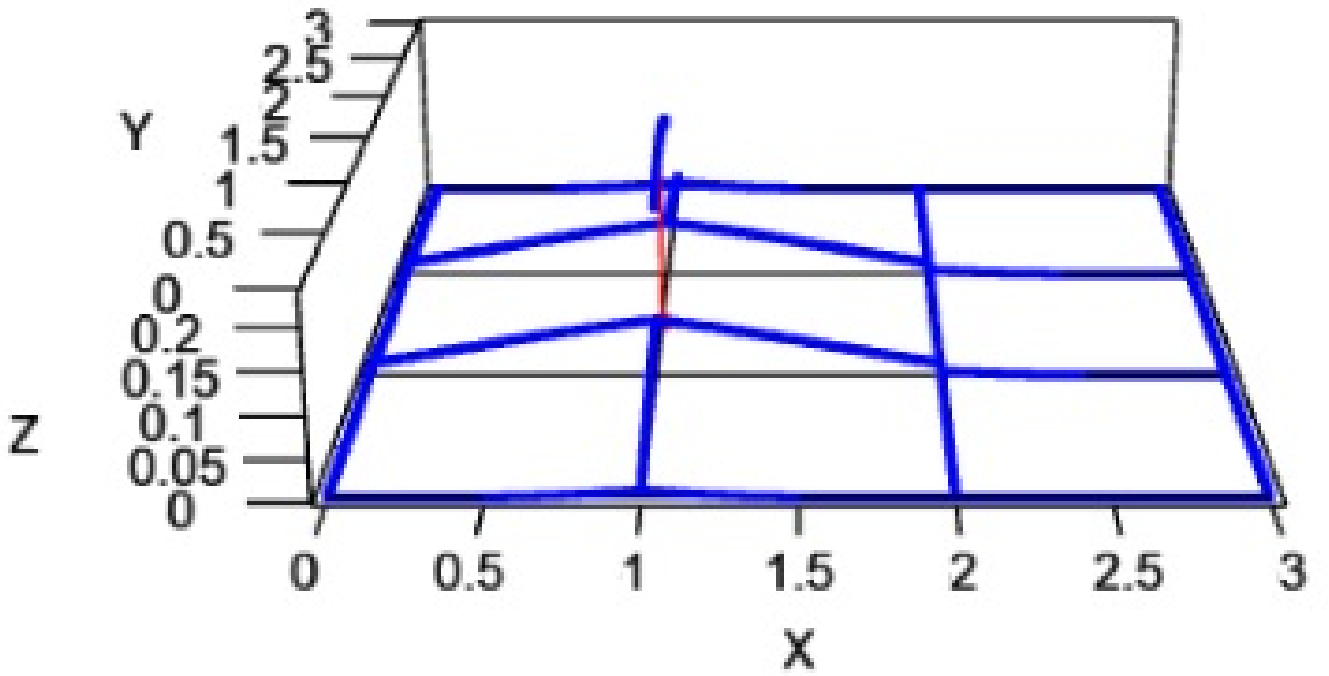


Figure 6.6 – Visualisation du Geo-NKDE

La formule pour calculer la ESD-NKDE est la suivante :

$$\hat{\lambda}_h\left(u, e_i\right) = k\left(d_{ist_{net}}\left(u, e_i\right)\right) \prod_{j=1}^{J} \left(\frac{1}{(n_{ij} - 1)}\right) \text{ avec :} \tag{6.2}$$

– $\prod_{j=1}^{J}\left(\frac{1}{(n_{ij}-1)}\right)$ est le terme qui permet de contrôler la réduction de la masse due aux J intersections rencontrées entre u et e_i et ayant un nombre d'embranchements n_{ij}.

NKDE continue (ESC-NKDE). La troisième méthode implique également de diviser la masse des évènements aux intersections, mais aussi de corriger rétroactivement la masse précédant l'intersection pour forcer l'estimation à être continue. Cette estimation est donc non biaisée et ne produit pas de discontinuité (voir la figure ci-dessous). Cependant, la correction rétroactive nécessite un temps de calcul nettement plus long que les deux précédentes méthodes.

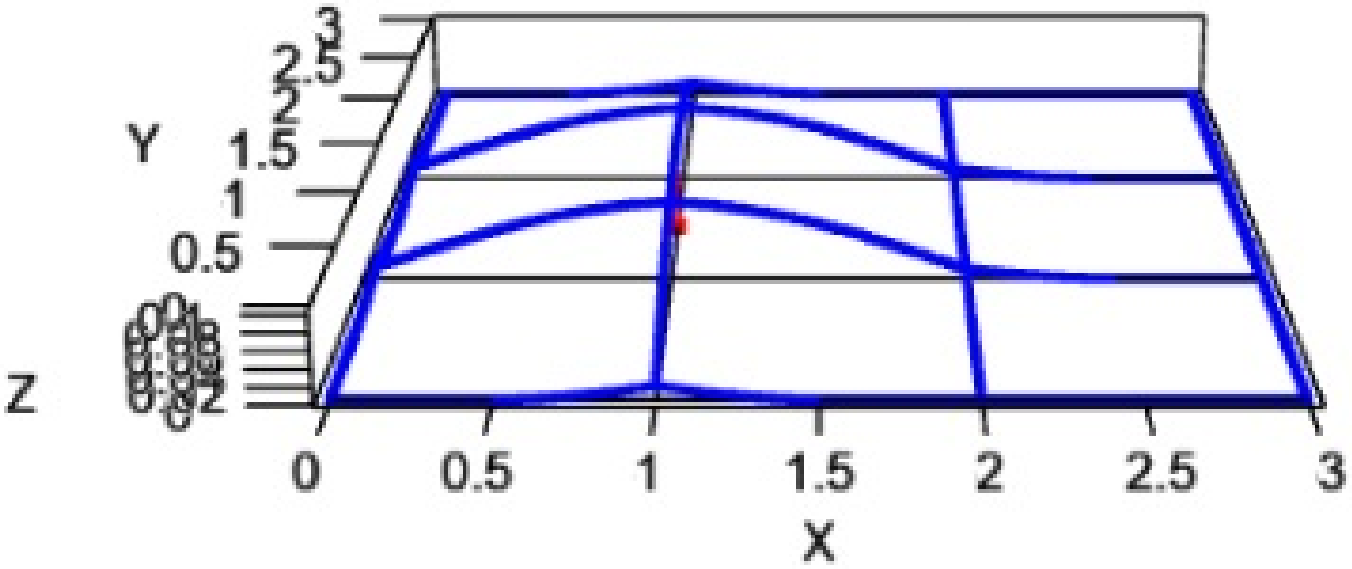


Figure 6.7 – Visualisation du Geo-NKDE

Du fait de sa nature récursive, il est difficile de présenter la ESC-NKDE avec une équation. Cependant, l'algorithme complet est décrit par Atsuyuki Okabe et Sugihara Kokichi (2012).

La figure 6.8 permet de comparer la répartition de la masse des évènements aux intersections entre les NKDE, tandis que le tableau 6.1 résume les avantages et inconvénients des trois types de NKDE.

Tableau 6.1 – Comparaison des trois NKDE

NKDE	Avantage	Désavantage
NKDE géographique (Geo-NKDE)	Intuitive et facile à calculer	La somme de la masse totale est inexacte.
NKDE discontinue (ESD-NKDE)	Respect de la masse totale et facile à calculer	L'espace discontinu est contre-intuitif
NKDE continue (ESC-NKDE)	Respect de la masse totale et intuitive	Couteux en termes de temps de calcul

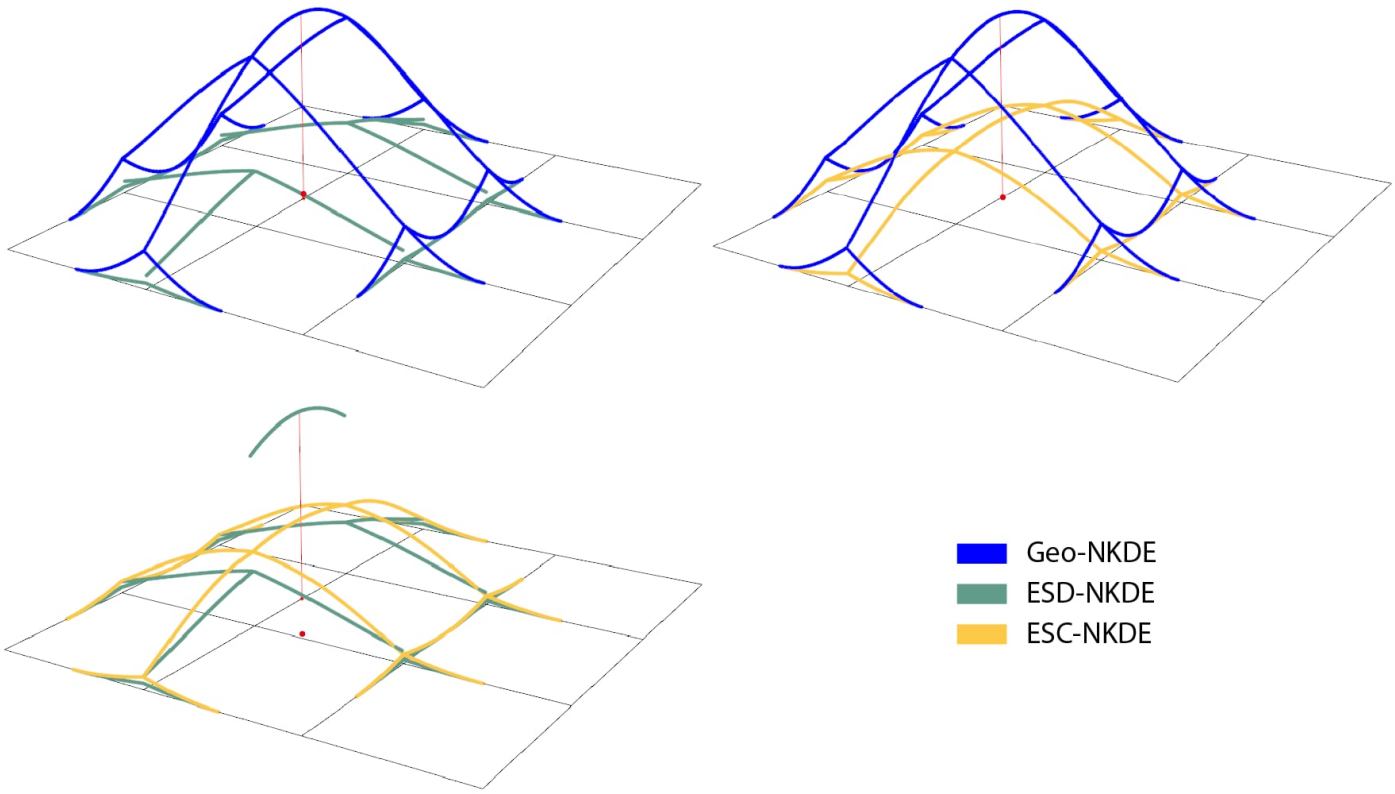


FIGURE 6.8 – Comparaison des trois types de NDKDE : géographique (Geo-NKDE), discontinue (ESD-NKDE) et continue (ESC-NKDE)

Aller plus loin

***Package* `spNetwork`**
Pour une lecture plus détaillée sur les trois NKDE, nous vous recommandons de lire l'article décrivant le *package* `spNetwork` (Jeremy Gelb 2021).

6.2.1.2 Correction de Diggle

Très souvent, les données collectées pour un phénomène analysé sont limitées à une zone géographique (territoire d'étude) et les évènements se produisant en dehors de cette zone ne sont pas enregistrés. Ce biais de collecte entraîne une sous-estimation systématique de l'intensité estimée par les méthodes KDE et NKDE aux frontières de la zone d'étude. Pour limiter cette sous-estimation, il est préférable de collecter directement les données dans un périmètre plus large que la zone étudiée. Cependant, lorsque les données ont déjà été collectées, il est possible d'appliquer la correction de Diggle (1985) (équation 6.3).

$$\begin{aligned} \lambda^D(u) &= \frac{1}{bw} \sum_{i=1}^{n} w_i \cdot \frac{1}{e\left(e_i\right)} K\left(\text{dist}\left(u, e_i\right)\right) \\ e(u) &= \int_W^v K(\text{dist}(u, v)) \text{ avec :} \end{aligned} \tag{6.3}$$

— $e(u)$ étant la masse de l'évènement u localisé dans la zone d'étude W.

Concrètement, cette correction propose d'augmenter la masse des évènements localisés à proximité de la frontière de la zone d'étude. Ces évènements voient leur pondération multipliée par l'inverse de la proportion de leur masse comprise à l'intérieur de la zone d'étude. Ainsi, un point dont 100 % de la masse se trouve dans la zone d'étude garde le même poids (1/1 = 1); un point avec 75 % de sa masse dans la zone d'étude a son poids multiplié par 4/3 (1/0,75 = 4/3) et un point avec 50 % de sa masse dans la zone d'étude a son poids multiplié par deux (1/0,5 = 2).

6.2.1.3 Bandwidths adaptatives

Jusqu'ici, nous avons présenté les méthodes d'estimation de la densité par *kernel* avec des *bandwidths* globales. Il existe une catégorie de *kernels* appelés adaptatifs qui utilisent, comme leur nom l'indique, des *bandwidths* s'adaptant localement.

L'idée générale est que chaque évènement peut avoir sa propre *bandwidth* locale. Cette modification se justifie du point de vue théorique. Pour un rappel, nous considérons que les évènements ont eu lieu à un certain endroit du fait d'un patron spatial d'arrière-plan. Nous formulons donc l'hypothèse qu'un évènement aurait pu se produire dans un certain rayon (*bandwidth*) autour de son emplacement réel selon une probabilité décroissante avec la distance (fonction *kernel*). Dans les secteurs où se situent de nombreux points, notre incertitude sur la localisation d'un point est moins grande, nous pouvons donc utiliser des *bandwidths* plus petites. Cependant, dans les secteurs avec très peu de points, le processus spatial est beaucoup plus diffus et l'évènement aurait pu se produire dans un rayon plus large, incitant à utiliser des *bandwidths* plus grandes.

Deux approches sont le plus souvent utilisées pour créer des *bandwidths* locales : la méthode d'Abramson (1982) et la méthode des *k* plus proches voisins (Orava 2011).

Bandwidths adaptatives par la méthode d'Abramson

La méthode d'Abramson (1982) propose d'utiliser des *bandwidths* locales qui sont inversement proportionnelles à la racine carrée de l'intensité locale du processus spatial étudié. Cependant, puisque nous ne disposons pas d'une mesure de ce processus, nous devons en fournir une approximation à priori. Cette approximation est obtenue en sélectionnant une première *bandwidth* globale (appelée pilote) et l'estimation de la densité est effectuée. Une fois que la densité à priori est calculée à la localisation exacte de chaque évènement, il est ensuite possible de calculer une *bandwidth* locale pour chaque évènement avec l'équation 6.4 :

$$h(e_i) = h_0 \times \frac{1}{\sqrt{\tilde{f}h_0(e_i)}} \times \frac{1}{\gamma_f}\gamma_f = \exp(\frac{\sum_i log(\frac{1}{\sqrt{\tilde{f}h_0(e_i)}})}{n}) \text{ avec :} \tag{6.4}$$

- $h(e_i)$ est la *bandwidth* locale pour l'évènement e_i.
- h_0 est la *bandwidth* pilote globale.
- $\tilde{f}h_0(e_i)$ est l'estimation locale de la densité à priori pour l'évènement e_i avec h_0.

L'objectif est bien évidemment de sélectionner la *bandwidth* pilote h_0.

Bandwidths adaptatives par la méthode des k plus proches voisins

Cette méthode est certainement plus facile à expliquer. Elle consiste à calculer, pour chaque évènement, la distance qui le sépare de son plus proche voisin de rang *k* (*k* étant un entier plus grand que 0). Dans des secteurs avec peu d'évènements, la distance au plus proche voisin de rang *k* sera plus grande. L'enjeu pour cette méthode est donc de déterminer *k*. La figure 6.9 illustre l'impact de ces méthodes sur les *bandwidths* locales en prenant le jeu de données fourni dans le *package* `spNetwork`, portant sur les accidents à vélo dans les quartiers centraux de Montréal.

6.2.1.4 Sélection d'une bandwidth

Comme signalé dans la section 3.4.2.1, le choix de la *bandwidth* est crucial dans l'application d'une estimation de densité par *kernel*. Dans le cas de la NKDE, le nombre de méthodes est plus limité que pour la KDE, mais il est toujours possible d'utiliser l'approche par validation croisée des probabilités (*likelihood cross validation*). Plus exactement, cette méthode choisit une *bandwidth* de façon à minimiser l'impact qu'aurait le fait de retirer un évènement du jeu de donnée (*leave one out cross validation*). En effet, puisque chaque point est une réalisation d'un processus spatial, le fait de retirer un point des données ne devrait affecter que marginalement l'estimation locale de l'intensité du processus spatial. Il est possible de minimiser ce score en sélectionnant la bonne *bandwidth*. Il est possible de calculer ce score pour une *bandwidth* donnée avec l'équation 6.5 :

$$\mathrm{LCV}(bw) = \sum_i \log \hat{\lambda}_{-i}(x_i) \text{ avec :} \tag{6.5}$$

- bw est la *bandwidth* à évaluer.
- $\hat{\lambda}_{-i}$ est l'intensité estimée à la localisation de l'évènement *i* sans la présence de l'évènement *i*.

Notez qu'il s'agit d'une simplification de l'équation qui comporte normalement un second terme qui tend à être une constante et peut donc être retiré pour alléger les calculs (Loader 2006). Cette méthode peut aussi être utilisée pour sélectionner la *bandwidth* pilote ou *k* lorsque nous utilisons une *bandwidth* adaptative.

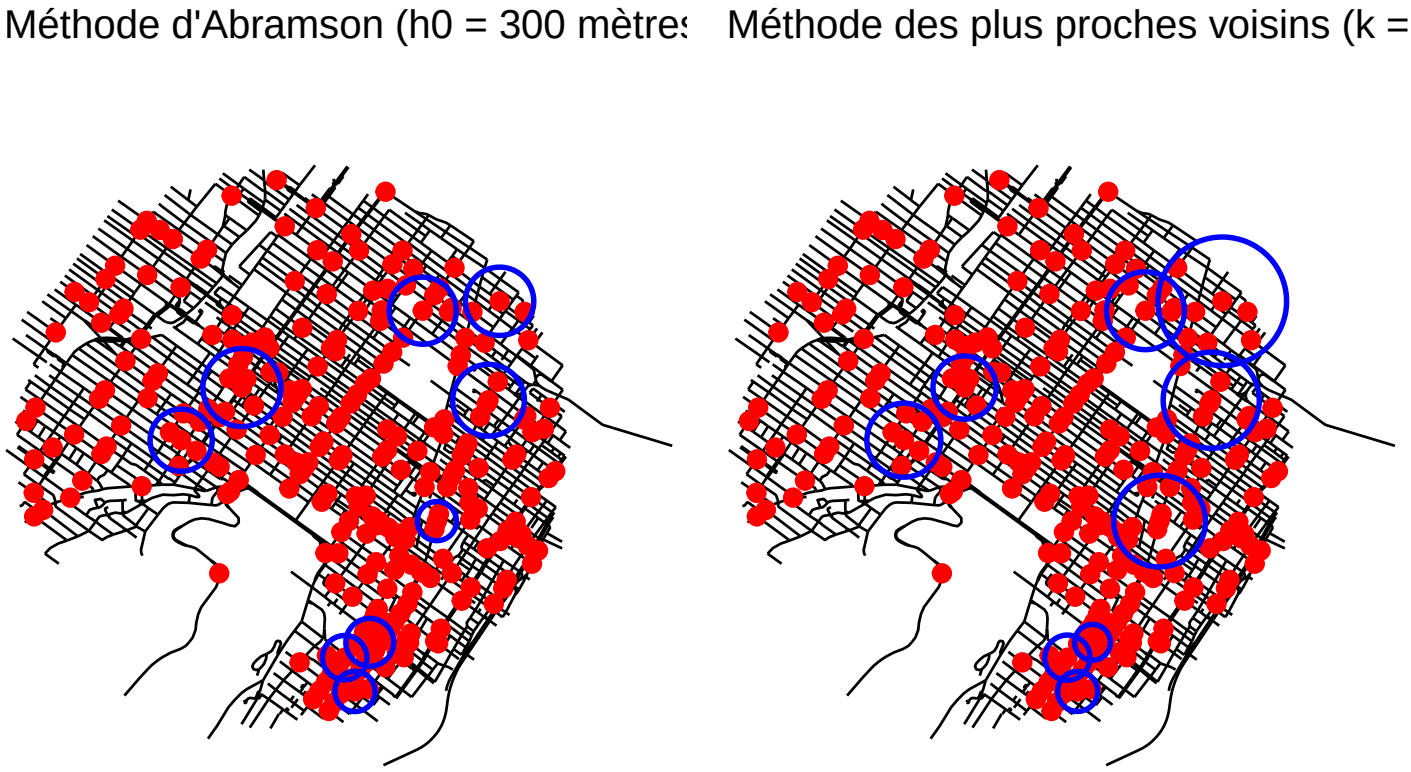


FIGURE 6.9 – Comparaison des deux principales méthodes de création de *bandwidths* locales

6.2.2 Mise en œuvre dans R

Nous analysons ici les collisions ayant eu lieu sur le réseau routier de la ville de Sherbrooke. Nous commençons par appliquer une NKDE avec *bandwidth* fixe et nous la comparons avec deux NKDE utilisant des *bandwidths* adaptatives. Nous utilisons principalement le *package* `spNetwork` (Jeremy Gelb 2021).

La première étape consiste à charger les données des accidents et le réseau routier. La figure 6.10 permet de visualiser la répartition spatiale des accidents.

```
library(sf)
library(tmap)
library(ggplot2)
library(spNetwork)
library(future) # package utilisé pour accélérer les calculs dans spNetwork
future::plan(future::multisession(workers = 5))
## Importation des couches géographiques
routes <- st_read('data/chap01/shp/Segments_de_rue.shp', quiet = TRUE)
collisions <- st_read('data/chap04/DataAccidentsSherb.shp', quiet = TRUE)
## Application de la même projection
routes <- st_transform(routes, 2949)
routes <- sf::st_cast(routes, 'LINESTRING')
collisions <- st_transform(collisions, 2949)
## Cartographie des données des collisions et du réseau routier
```

```
tm_shape(routes) +
  tm_lines('grey20') +
  tm_shape(collisions) +
  tm_dots('red', size = 0.05)+
tm_layout(frame = FALSE)
```

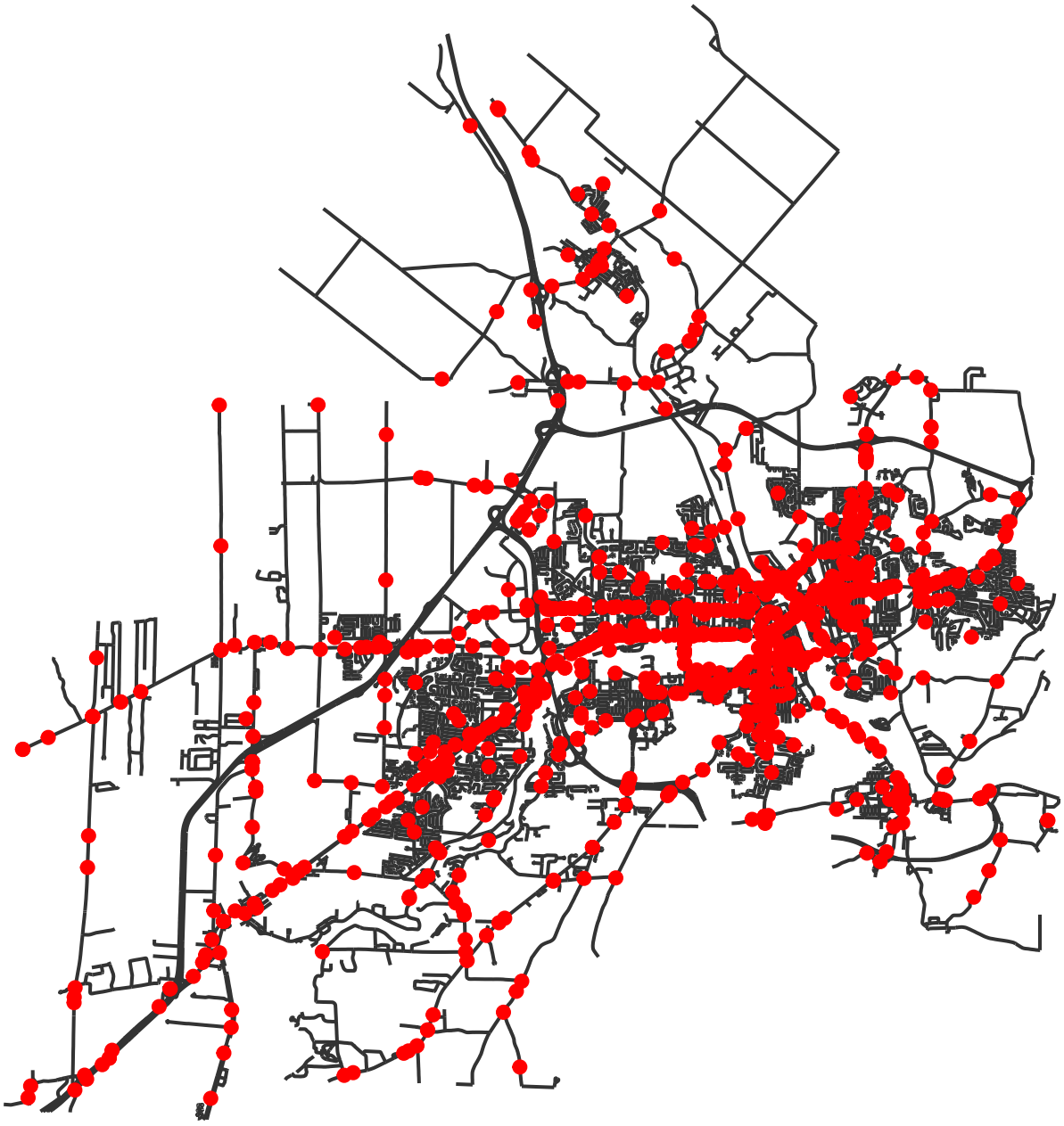


Figure 6.10 – Accidents sur le réseau routier de la ville de Sherbrooke

Pour l'analyse, nous utilisons la fonction *kernel* quadratique et la NKDE continue (ESC-NKDE). Puis, nous choisissons une *bandwidth* avec l'approche par validation croisée des probabilités. Notez que pour réduire le temps de calcul, la NKDE

discontinue (ESD-NKDE) est utilisée dans la phase de sélection de la *bandwidth*, car il est bien plus rapide à calculer.

```
eval_bandwidth <- bw_cv_likelihood_calc.mc(
  bws = seq(100, 1200, 50),
  lines = routes,
  events = collisions,
  w = rep(1, nrow(collisions)), # le poids de chaque évènement est 1
  kernel_name = 'quartic',
  method = 'discontinuous',
  adaptive = FALSE,
  max_depth = 10,
  digits = 1,
  tol = 0.1,
  agg = 5, # les accidents dans un rayon de 5 mètres seront agrégés
  grid_shape = c(5,5),
  verbose = TRUE)
```

À la figure 6.11, nous constatons qu'au-delà de 900 mètres, le gain obtenu en augmentant la valeur de la *bandwidth* est marginal. Par conséquent, nous retenons cette valeur de *bandwidth* pour la première estimation de l'intensité des accidents.

```
## Graphique pour les bandwidths
ggplot(eval_bandwidth) +
  geom_path(aes(x = bw, y = cv_scores)) +
  geom_point(aes(x = bw, y = cv_scores), color = 'red')+
  labs(x = "Valeur de la bandwidth", y = "Valeur du CV")
```

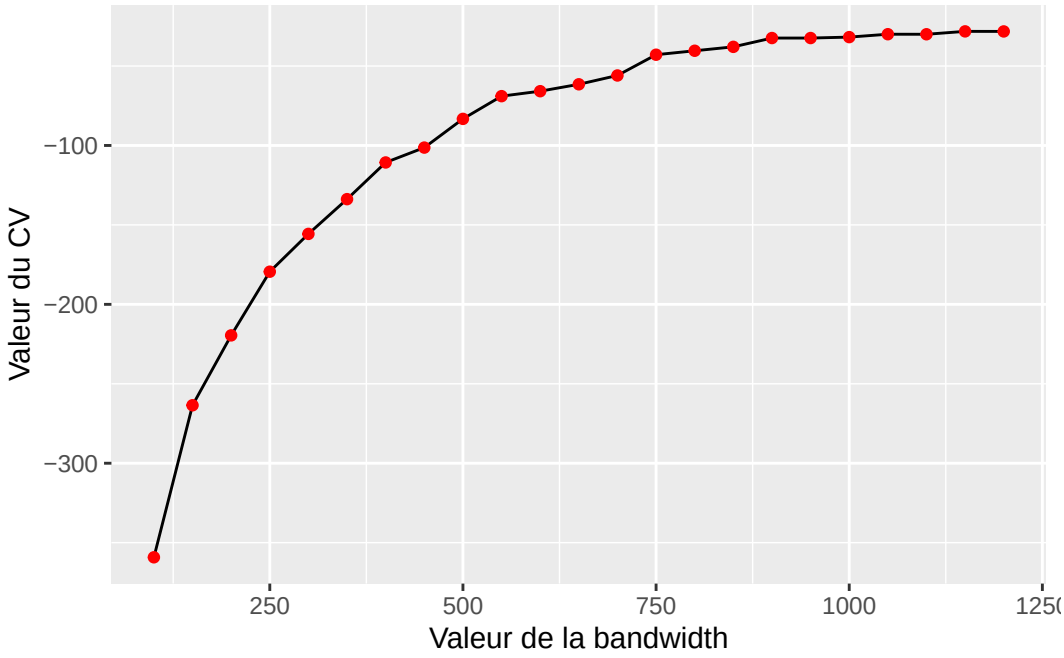


Figure 6.11 – Scores des *bandwidths* globales

```
## Création des lixels d'une longueur de 100 mètres
lixels <- lixelize_lines(routes, 100, mindist = 50)
## Centroïdes des lixels
lixels_centers <- spNetwork::lines_center(lixels)
```

```
## Calcul de la NKDE
future::plan(future::multisession(workers=2))
intensity <- nkde.mc(lines = routes,
                     events = collisions,
                     w = rep(1, nrow(collisions)),
                     samples = lixels_centers,
                     kernel_name = 'quartic',
                     bw = 900,
                     adaptive = FALSE,
                     method = 'continuous',
                     max_depth = 8,
                     digits = 1,
                     tol = 0.1,
                     agg = 5,
                     verbose = FALSE,
                     grid_shape = c(5,5))
if (!inherits(future::plan(), "sequential")) future::plan(future::sequential)
```

Une fois que les valeurs de densité sont obtenues, nous pouvons les cartographier à l'échelle des lixels (figure 6.12).

```
lixels$density <- intensity * 1000
tm_shape(lixels) +
  tm_lines("density", lwd = 1.5, n = 7, style = "fisher",
           legend.format = list(text.separator = "à"))+
  tm_layout(frame=FALSE)
```

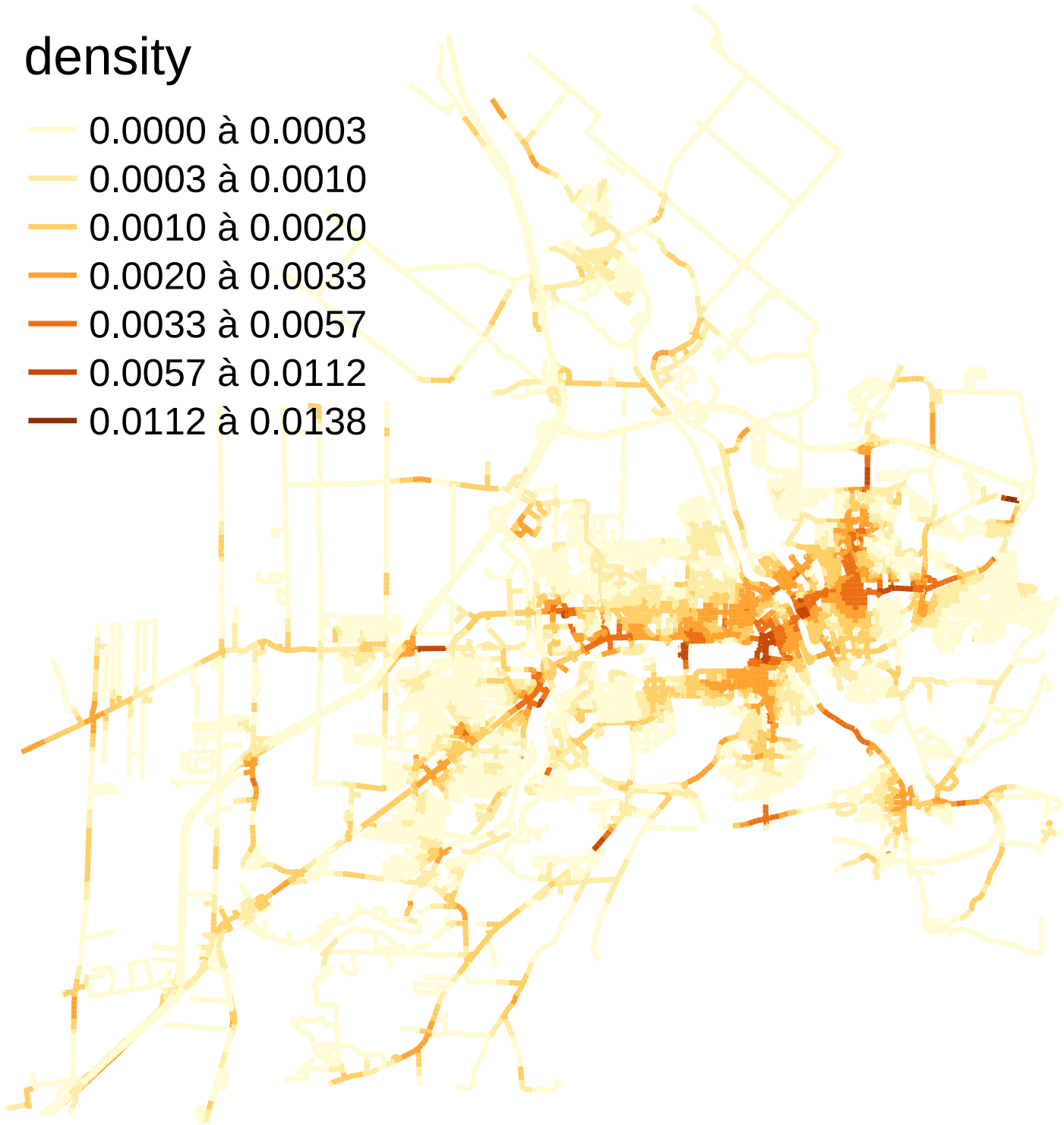


FIGURE 6.12 – Densité des accidents sur le réseau routier de Sherbrooke

Nous pouvons à présent utiliser une *bandwidth* adaptative. Pour cela, nous devons réévaluer les différentes *bandwidths* globales avec l'approche par validation croisée des probabilités.

```
future::plan(future::multisession(workers=2))
eval_bandwidth_adapt <- bw_cv_likelihood_calc.mc(
  bws = seq(100, 1200, 50),
  lines = routes,
  events = collisions,
  w = rep(1, nrow(collisions)), # le poids de chaque évènement sera 1
  kernel_name = 'quartic',
  method = 'discontinuous',
  adaptive = TRUE,
  trim_bws = seq(100, 1200, 50) * 2,
  max_depth = 10,
  digits = 2,
  tol = 0.1,
  agg = 5, # tous les accidents dans un rayon de 5m seront agrégés
  grid_shape = c(5,5),
  verbose = TRUE
)
if (!inherits(future::plan(), "sequential")) future::plan(future::sequential)
```

```
ggplot() +
  geom_path(data = eval_bandwidth,
            mapping =  aes(x = bw, y = cv_scores)) +
  geom_point(data = eval_bandwidth,
             mapping =  aes(x = bw, y = cv_scores), color = 'red') +
  geom_path(data = eval_bandwidth_adapt,
            mapping =  aes(x = bw, y = cv_scores)) +
  geom_point(data = eval_bandwidth_adapt,
             mapping =  aes(x = bw, y = cv_scores), color = 'blue')+
  labs(x = "Valeur de la bandwidth", y = "Valeur du CV")
```

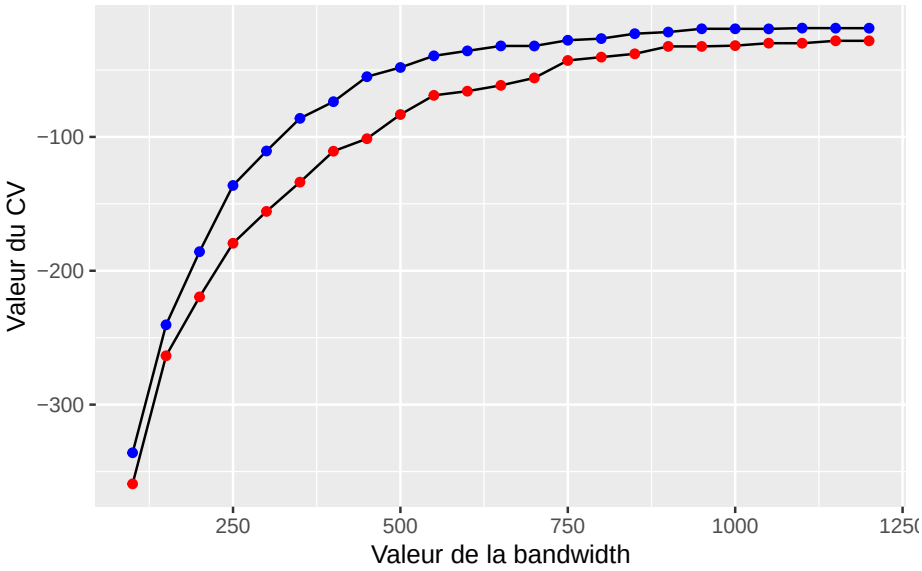


FIGURE 6.13 – Scores des bandwidths adaptatives

La figure 6.13 indique que les scores obtenus par les *bandwidths* adaptatives sont systématiquement supérieurs à ceux obtenus par les *bandwidths* fixes. Nous gardons une *bandwidth* pilote de 900 mètres pour recalculer notre ESC-NKDE avec une *bandwidth* adaptative.

```
intensity_adpt <- nkde.mc(lines = routes,
                  events = collisions,
                  w = rep(1, nrow(collisions)),
                  samples = lixels_centers,
                  kernel_name = 'quartic',
                  bw = 900,
                  adaptive = TRUE,
                  trim_bw = 1800,
                  method = 'continuous',
                  max_depth = 8,
                  digits = 1,
                  tol = 0.1,
                  agg = 5,
                  verbose = TRUE,
                  grid_shape = c(5,5))
```

```
lixels$density_adpt <- intensity_adpt$k * 1000
tm_shape(lixels) +
  tm_lines("density_adpt", lwd = 1.5, n = 7, style = "fisher",
           legend.format = list(text.separator = "à"))+
  tm_layout(frame=FALSE)
```

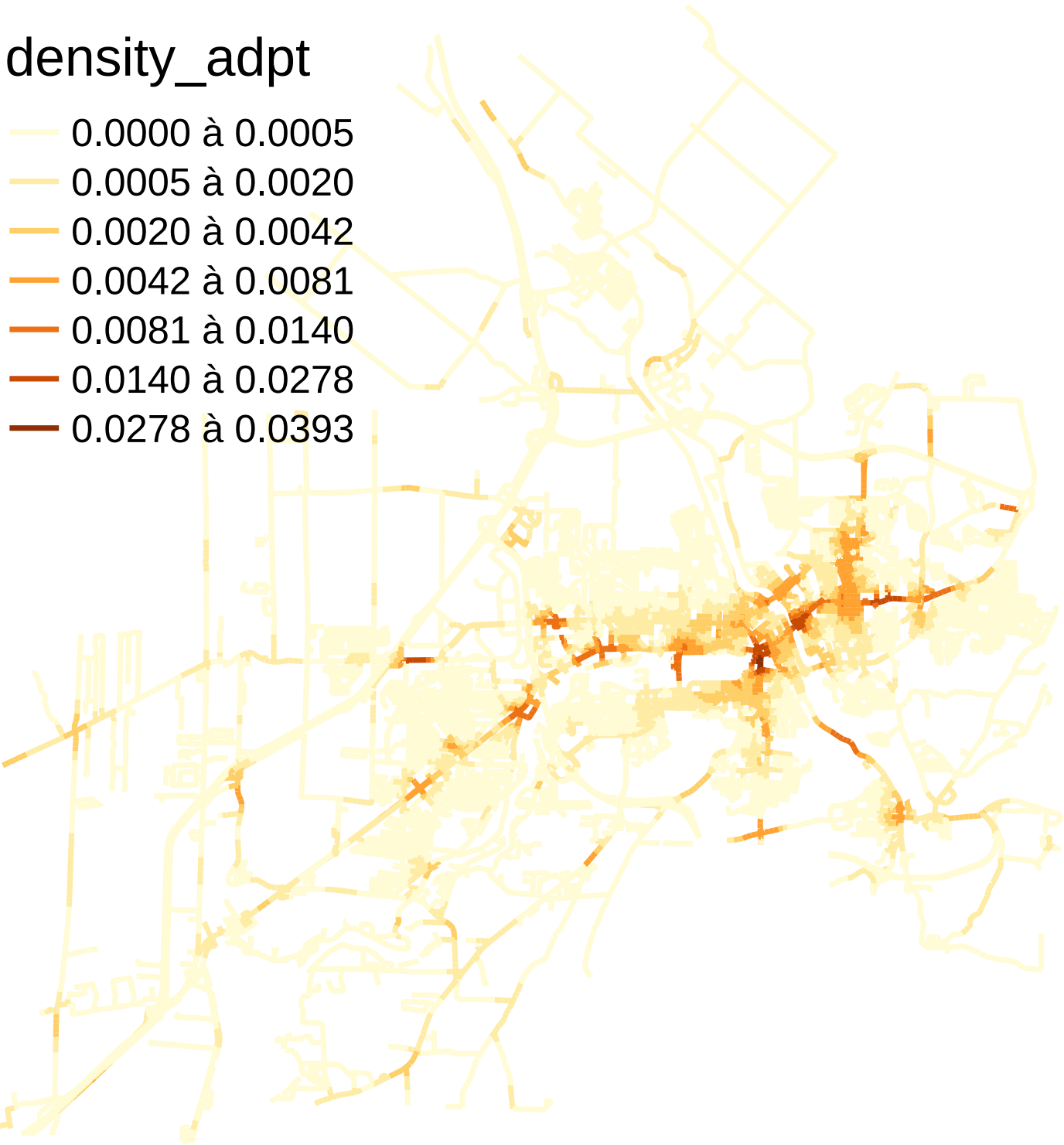


FIGURE 6.14 – Densité des accidents sur le réseau routier de Sherbrooke avec une *bandwidth* adaptative

Comparativement aux résultats obtenus avec les *bandwidths* fixes, nous constatons que le lissage est beaucoup plus faible et que les points chauds sont plus faciles à identifier. Le *package* `spNetwork` permet aussi d'utiliser la méthode des k plus

proches voisins comme *bandwidth* locale.

```
knn_net <- spNetwork::network_knn(collisions,
                                  lines = routes,
                                  k = 30,
                                  maxdistance = 3000,
                                  grid_shape = c(1,1),
                                  verbose = FALSE)
dist_mat <- knn_net$distances
# Nous limitons les bandwidths avec des bornes de 100 à 3000 m
dist_mat <- ifelse(dist_mat > 3000, 3000, dist_mat)
dist_mat <- ifelse(dist_mat < 100, 100, dist_mat)
```

```
eval_bandwidth_knearest <- bw_cv_likelihood_calc(
  bws = NULL,
  mat_bws = dist_mat,
  lines = routes,
  events = collisions,
  w = rep(1, nrow(collisions)), # le poids de chaque évènement sera 1
  kernel_name = 'quartic',
  method = 'discontinuous',
  adaptive = TRUE,
  trim_bws = NULL,
  max_depth = 10,
  digits = 1,
  tol = 0.1,
  agg = 5, # tous les accidents dans un rayon de 5 mètres seront agrégés
  grid_shape = c(5,5),
  verbose = TRUE
)
```

```
eval_bandwidth_knearest$knn <- 1:30
ggplot() +
  geom_path(data = eval_bandwidth_knearest,
            mapping =  aes(x = knn, y = cv_scores)) +
  geom_point(data = eval_bandwidth_knearest,
             mapping =  aes(x = knn, y = cv_scores), color = 'red')+
  labs(x = "Nombre de plus proches voisins (k)", y = "Valeur du CV")
```

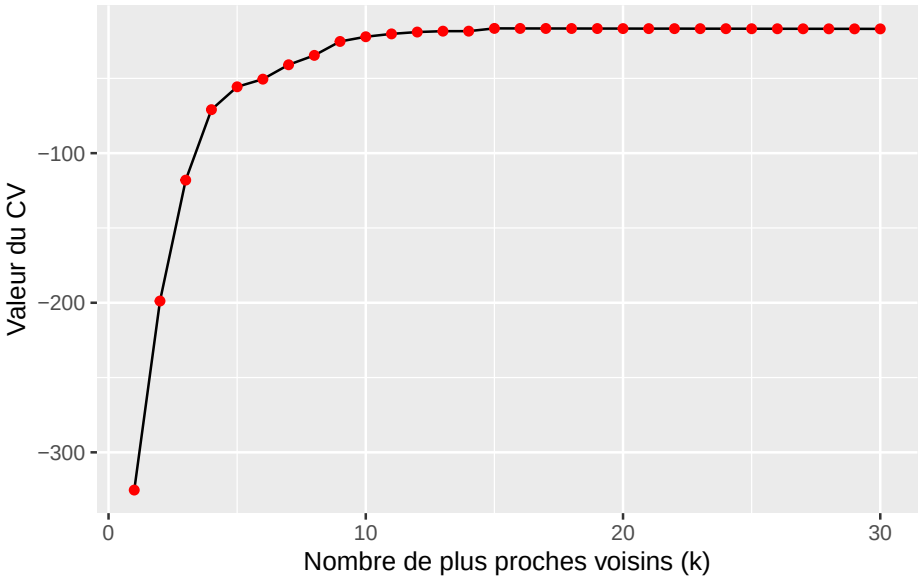


FIGURE 6.15 – Scores des bandwidths adaptatives par k plus proches voisins

La figure 6.15 indique que le meilleur score est obtenu pour une *bandwidth* allant jusqu'au 16^{e} voisin.

```
intensity_adpt_knn <- nkde.mc(lines = routes,
                  events = collisions,
                  w = rep(1, nrow(collisions)),
                  samples = lixels_centers,
                  kernel_name = 'quartic',
                  bw = dist_mat[,16],
                  trim_bw = 1800,
                  method = 'continuous',
                  max_depth = 8,
                  digits = 1,
                  tol = 0.1,
                  agg = 5,
                  verbose = TRUE,
                  grid_shape = c(5,5))
```

```
lixels$density_adpt_knn <- intensity_adpt_knn * 1000
tm_shape(lixels) +
  tm_lines("density_adpt_knn", lwd = 1.5, n = 7, style = "fisher",
           legend.format = list(text.separator = "à"))+
  tm_layout(frame=FALSE)
```

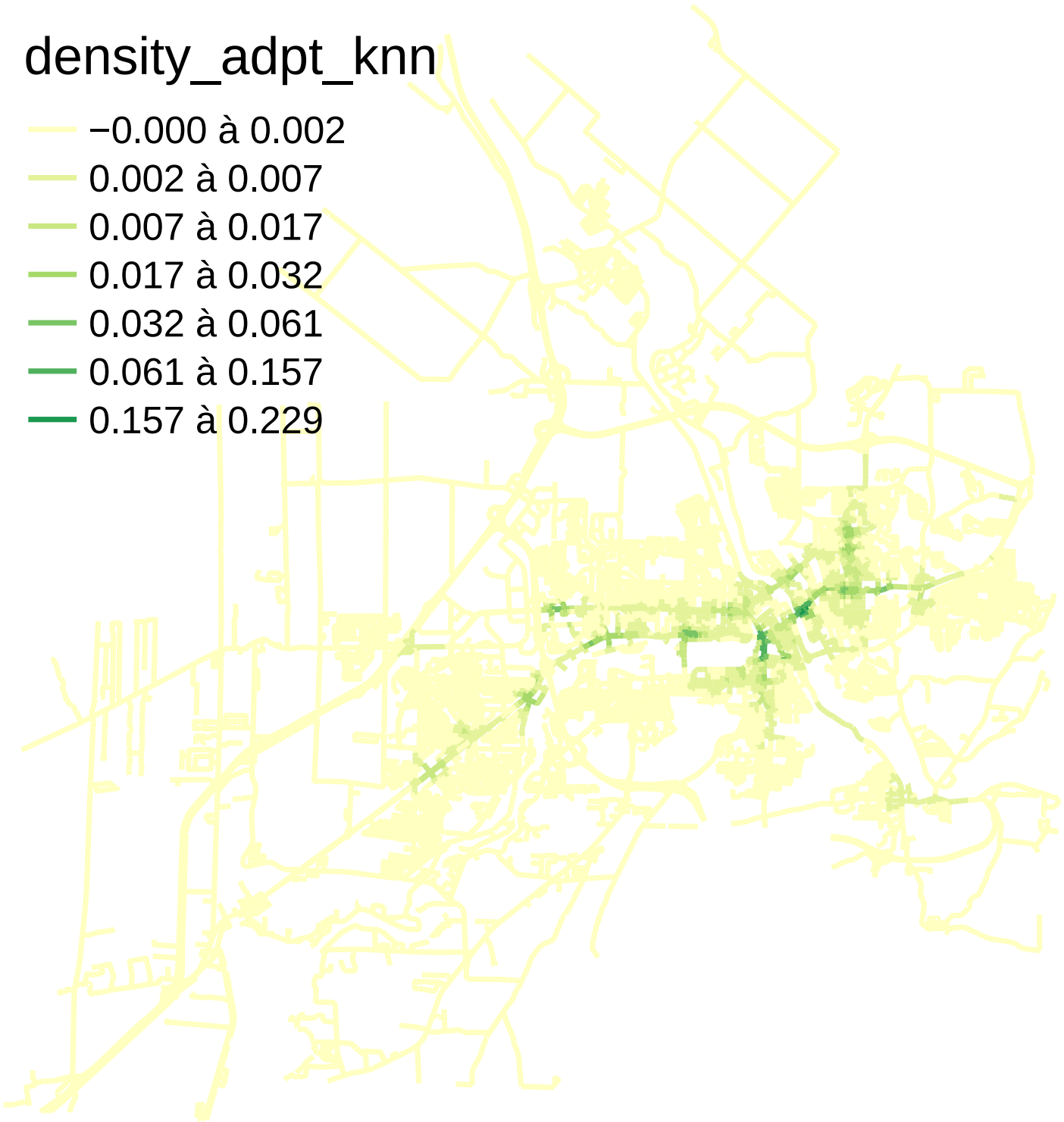


FIGURE 6.16 – Densité des accidents sur le réseau routier de Sherbrooke avec une *bandwidth* adaptative pour 16 plus proches voisins

6.2.3 Estimation de la densité spatio-temporelle sur un réseau

Comme nous avons pu le voir dans la section 3.4.3, il est courant d'analyser des données d'évènements disposant à la fois d'une localisation dans l'espace et dans le temps. Il est alors possible de calculer une densité spatio-temporelle, soit de lisser les évènements à la fois dans la dimension spatiale et dans la dimension temporelle. Plus exactement, la densité d'un évènement i en un point p et un instant t correspond au produit de la densité spatiale et de la densité temporelle de i.

Cette extension est aussi valide dans le cas de l'analyse d'évènement sur réseau.

$$\hat{\lambda}_{h_n h_t}(u_{nt}) = \frac{1}{h_n h_t} \sum_{i=1}^{N} k_{net}\left(\frac{\text{dist}_{net}(u_{nt}, e_i)}{h_n}\right) \sum_{i=1}^{N} k_{\text{time}}\left(\frac{\text{dist}_{\text{time}}(u_{nt}, e_i)}{h_t}\right) \text{ avec :}$$

- h_t et h_n les *bandwidths* temporelle et réseau.
- u_{nt} un évènement localisé au point n du réseau et à l'instant t dans le temps.

La figure 6.17 illustre le calcul de la TNKDE, soit l'estimation de la densité spatio-temporelle sur un réseau (*Temporal Network Kernel Density Estimate*).

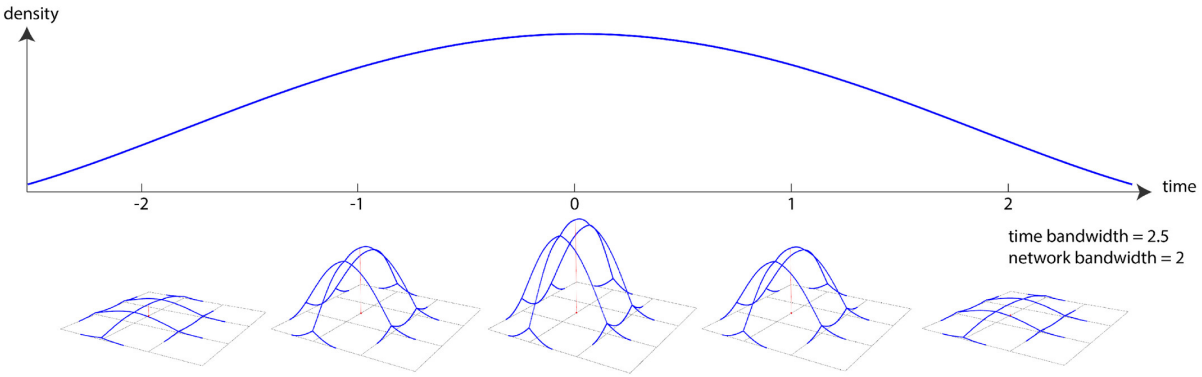


FIGURE 6.17 – La TNKDE comme produit des densités spatiale et temporelle

Comme pour le NKDE, il est possible de :

- Utiliser des *bandwidths* variant localement dans l'espace et le temps.
- Comparer des *bandwidths* par des méthodes de validation croisée.
- Appliquer des correctifs aux frontières spatio-temporelles de la zone d'étude.

6.2.3.1 Application dans R

Nous reprenons simplement l'exemple de la section sur la NKDE et voir comment l'étendre au contexte spatio-temporel. Pour cela, nous utiliserons principalement le *package* `spNetwork`.

```
library(sf)
library(spNetwork)
library(lubridate)
library(metR)
library(future) # pour accélérer les calculs de spNetwork
future::plan(future::multisession(workers = 5))

routes <- st_read('data/chap01/shp/Segments_de_rue.shp', quiet = TRUE)
collisions <- st_read('data/chap04/DataAccidentsSherb.shp', quiet = TRUE)
## Préparation de la colonne avec les dates
collisions$dt <- as_date(collisions$DATEINCIDE)
collisions$dt_num <- as.numeric(collisions$dt - min(collisions$dt))
## Reprojection dans le même système
routes <- st_transform(routes, 32187)
collisions <- st_transform(collisions, 32187)
routes <- sf::st_cast(routes, 'LINESTRING')
routes$length <- st_length(routes)
## Préparation des routes et des lixels
routes <- sf::st_cast(routes, 'LINESTRING')
```

Nous commençons par compléter le réseau routier. En effet, certaines sections sont isolées et forment des enclaves inaccessibles. Nous avons ignoré cette problématique jusqu'ici, mais nous verrons comment retirer les petites enclaves déconnectées de la partie principale du réseau. Pour cela, nous commençons par créer un objet de type `graph` à partir des routes avec le *package* `spNetwork`.

```
library(igraph)
library(dbscan)
library(tmap)
routes <- sf::st_cast(routes, 'LINESTRING')
routes$length <- st_length(routes)
graph <- spNetwork::build_graph(routes, digits = 2,line_weight = "length")
parts <- components(graph$graph)
graph$spvertices$part <- as.character(parts$membership)
tm_shape(graph$spvertices) +
  tm_dots("part", size = 0.1)
```

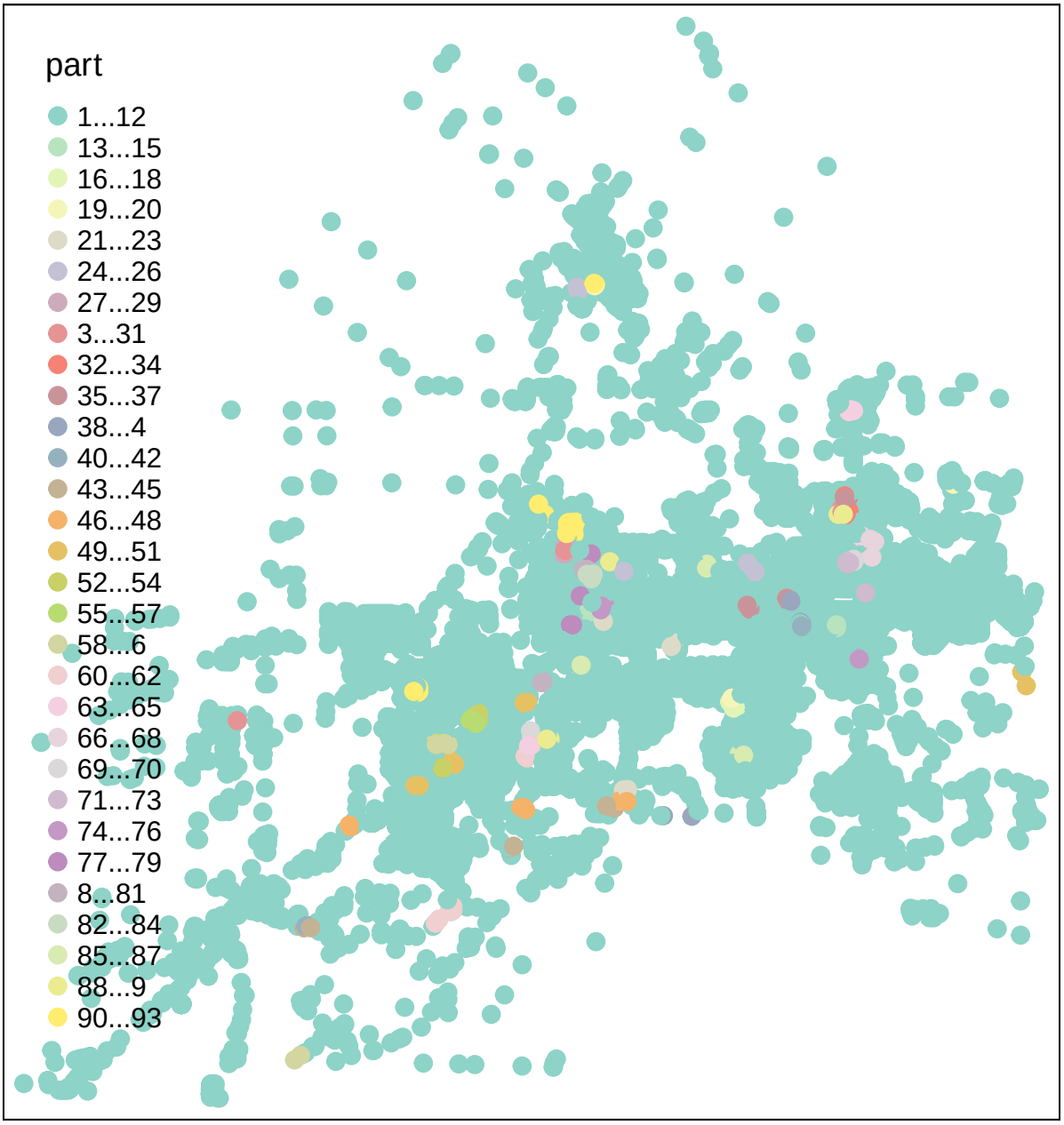


FIGURE 6.18 – Visualisation des composantes du réseau routier

À la figure 6.18, nous pouvons identifier la partie principale du réseau et les segments déconnectés (éléments avec des valeurs supérieures à 1). Puis, nous soustrayons ces éléments du réseau pour alléger le graphe et éviter d'associer des collisions avec des parties inaccessibles du réseau routier.

```
main_component <- subset(graph$spvertices, graph$spvertices$part == "1")
main_network <- subset(graph$spedges,
                  (graph$spedges$start_oid %in% main_component$id) |
                  (graph$spedges$end_oid %in% main_component$id)
```

```
                    )
main_network <- subset(main_network, as.numeric(st_length(main_network)) > 0)
```

Maintenant que nous avons nettoyé notre réseau, nous calculons les scores pour les *bandwidths*.

```
lixels_main <- lixelize_lines(main_network, 100, mindist = 50)
lixels_main_centers <- spNetwork::lines_center(lixels_main)
# Calcul des scores pour les bandwidths
cv_scores_tnkde <- bw_tnkde_cv_likelihood_calc(
  bws_net = seq(700, 1500, 100),
  bws_time =  seq(10, 40, 5),
  lines = main_network,
  events = collisions,
  time_field = "dt_num",
  w = rep(1, nrow(collisions)),
  kernel_name = "quartic",
  method = "continuous",
  max_depth = 10,
  digits = 2,
  tol = 0.1,
  agg = 10,
  grid_shape = c(5,5),
  verbose = TRUE)
```

```
# Création d'un graphique pour visualiser les résultats
library(ggplot2)
df2 <- reshape2::melt(cv_scores_tnkde)
ggplot(df2) +
  geom_tile(aes(x = Var1, y = Var2, fill = value)) +
  geom_contour(aes(x = Var1, y = Var2, z = value),
               breaks = c(-400,-300, -250, -200, -180, -150),
               color = 'white', linetype = 'dashed')+
  scale_fill_viridis_c() +
  labs(x = "Bandwidth spatiale (mètres)",
       y = "Bandwidth temporelle (jours)",
       fill = "cv score") +
  coord_fixed(ratio=30)
```

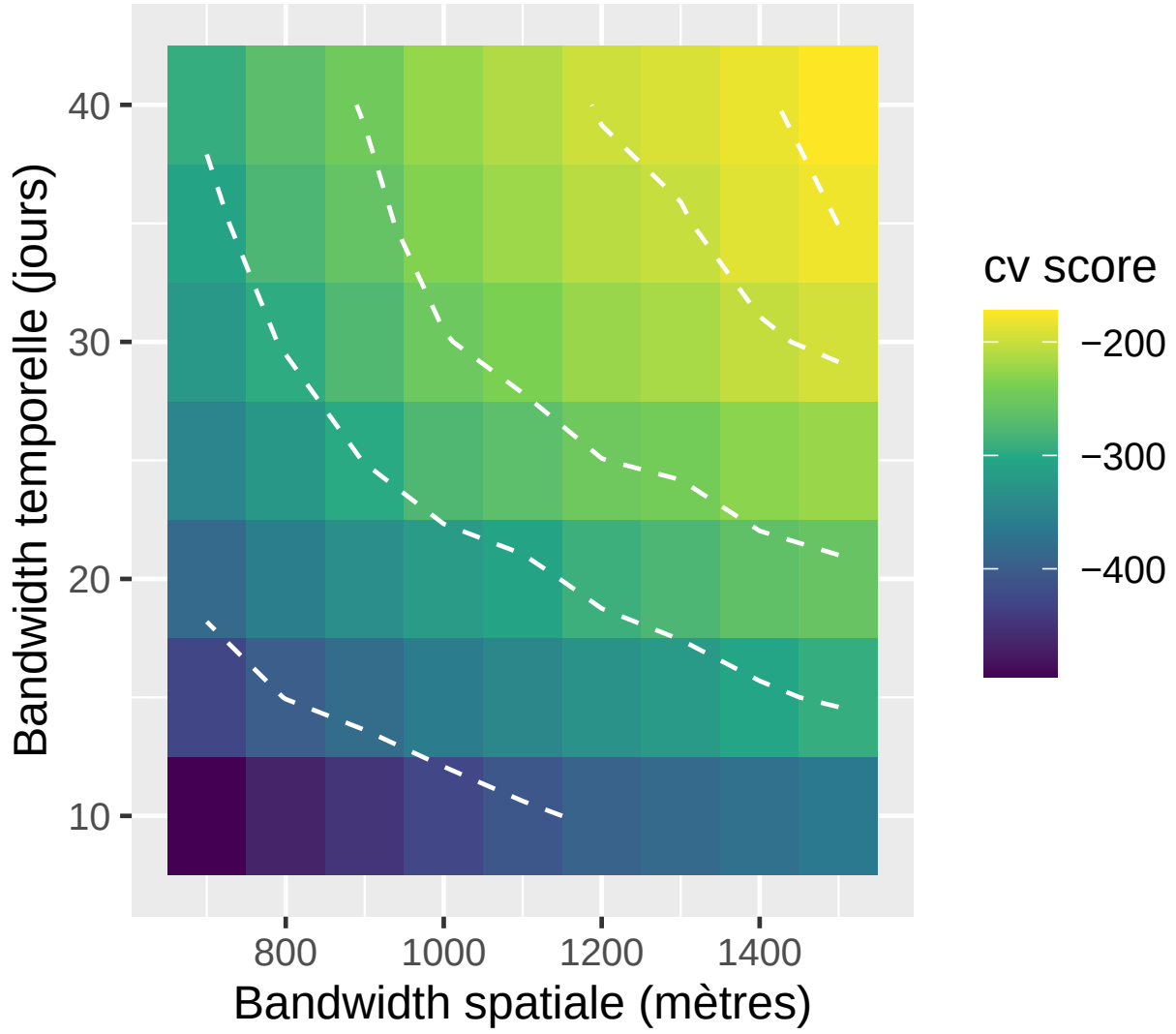


FIGURE 6.19 – Scores obtenus pour différentes combinaisons de *bandwidths* spatiales et temporelles

La figure 6.19 indique clairement que des *bandwidths* plus larges produisent de meilleurs résultats. Pour éviter d'avoir des résultats trop lissées, nous choisisons dans un premier temps la paire de *bandwidths* 30 jours et 1500 m. Puisque le temps de calcul peut être assez long compte tenu de la longueur des *bandwidths* (supérieure à 1 km), nous continuons donc à utiliser une NKDE discontinue.

```
# Choix de la résolution temporelle (dix jours ici)
sample_time <- seq(0, max(collisions$dt_num), 10)
# Calcul des densités
tnkde_densities <- tnkde.mc(lines = main_network,
                  events = collisions,
                  time_field = "dt_num",
                  w = rep(1, nrow(collisions)),
                  samples_loc = lixels_main_centers,
                  samples_time = sample_time,
```

```
                  kernel_name = "quartic",
                  bw_net = 1500, bw_time = 30,
                  adaptive = TRUE,
                  trim_bw_net = 1800,
                  trim_bw_time = 60,
                  method = "continuous",
                  div = "bw", max_depth = 10,
                  digits = 2, tol = 0.01,
                  adaptive_separate = FALSE,
                  agg = 10, grid_shape = c(5,5),
                  verbose  = TRUE)
```

On peut à présent représenter notre TNKDE avec une carte animée!

```
library(classInt)
library(viridis)
all_times <- min(collisions$dt) + days(sample_time)
tnkde_densities$k <- tnkde_densities$k*10000
tnkde_densities$k <- ifelse(tnkde_densities$k < 0, 0, tnkde_densities$k)
color_breaks <- classIntervals(c(tnkde_densities$k), n = 10, style = "kmeans")
all_maps <- lapply(1:length(all_times), function(i){
  dens <- tnkde_densities$k[,i]
  dt <- all_times[[i]]
  lixels_main$dens <- dens
  lixels2 <- lixels_main[order(-1*lixels_main$dens),]
  map <- tm_shape(lixels2) +
    tm_lines("dens", breaks = color_breaks$brks,
             palette = mako(10,direction = -1), lwd = 2) +
    tm_layout(frame = FALSE, legend.show=FALSE,
              main.title = as.character(all_times[[i]]))
  return(map)
})
# Création d'une animation pour produire la carte animée
tmap_animation(all_maps, filename = "images/Chap06/animated_TNKDE_sherbrooke.gif",
               width = 1000, height = 1000, dpi = 150, delay = 50)
```

6.3 Mesure d'autocorrélation spatiale sur un réseau

Dans le chapitre 3, nous avons présenté plusieurs mesures d'autocorrélation spatiale globales et locales. À titre de rappel, ces mesures utilisent une matrice W indiquant les relations spatiales (voisinage, distance, interaction) entre les observations. Lorsque nous analysons des observations sur un réseau, l'utilisation de matrices spatiales basées sur les distances réseaux plutôt qu'euclidiennes permet de représenter plus fidèlement l'organisation spatiale des observations. Voici quelques exemples de matrices de pondération spatiale construites à partir de distances calculées sur un réseau :

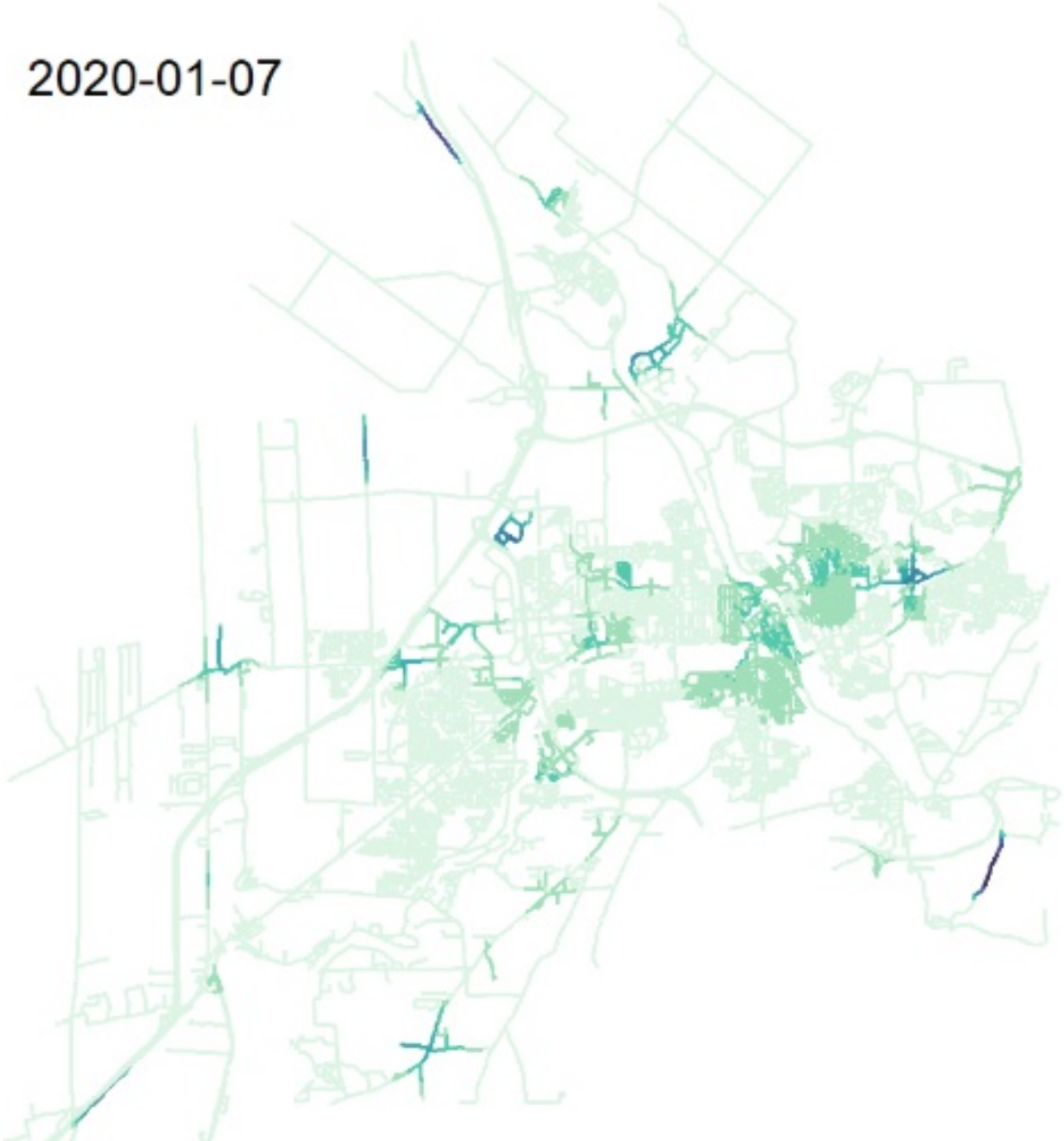


Figure 6.20 – Densité spatio-temporelle des collisions routières à Sherbrooke

— Matrice de connectivité selon la distance (section 2.2.2.2) : deux observations sont considérées comme voisines si la longueur du plus court chemin qui les sépare est inférieure au seuil de distance maximal fixé.
— Matrice des k plus proches voisins (section 2.2.2.4) : une observation a uniquement pour voisins les k autres observations les plus proches.
— Matrices basées sur la distance (section 2.2.2.3) : chaque observation est voisine de toutes les autres observations et le poids accordé à une paire d'observations dans la matrice est obtenu en appliquant une fonction décroissante (inverse de la distance, inverse de la distance au carrée, inverse de l'exponentielle, etc.) à la longueur du plus court chemin entre les deux observations. Si la fonction renvoie une pondération de 0, alors les deux observations sont trop éloignées pour être considérées comme voisines.

Ces matrices peuvent ensuite être standardisées en ligne tel que décrit dans la section 2.2.3.

Un bon exemple d'utilisation de ce type de matrice est l'évaluation de l'autocorrélation spatiale de l'intensité estimée par la méthode NKDE vu dans la section 6.1. En effet, la NKDE est une méthode essentiellement descriptive. Par conséquent, appliquer une mesure classique d'autocorrélation spatiale locale, comme les statistiques locales de Getis et Ord (section 2.4.1) ou la typologie basée sur le diagramme de Moran dans un contexte univarié (section 2.4.3.1), permet d'identifier les points chauds et froids de densité d'événemements statistiquement significatifs. Cette approche a notamment été proposée par Ikuho Yamada et Jean-Claude Thill (2007) sous le nom de *ILINCS* (*I Local Indicators of Network-Constrained Clusters*), mais ces auteurs utilisaient à l'époque une GEO-NKDE. Nous reprenons donc l'exemple précédent avec l'ESC-NKDE estimée pour les collisions routières à Sherbrooke et calculons le I de Moran global et une mesure d'autocorrélation spatiale locale sur les lixels, soit la typologie basée sur le diagramme de Moran.

6.3.1 Mise en œuvre dans R

La première étape consiste à créer une matrice de pondération spatiale entre les lixels sur le réseau routier de Sherbrooke. Nous testons plusieurs matrices pour trouver celle qui permet d'obtenir la plus haute valeur pour le I de Moran global.

Pour commencer, nous calculons simplement les distances réseau entre les lixels à l'aide de `spNetwork`. Notez que la fonction `network_listw` et sa version multicœur `network_listw.mc` renvoient un objet de type `listw` typique du *package* `spdep`. Il est aussi possible de l'utiliser pour simplement obtenir des distances (avec les paramètres `dist_func = 'identity'` et `matrice_type = 'I'`) pour ensuite leur appliquer des fonctions spécifiques. Nous utilisons ici cette approche pour éviter d'avoir à calculer plusieurs fois les chemins plus courts entre les lixels. Pour réduire le temps de calcul de la matrice, nous fixons la distance maximale à 2500 mètres avec (paramètre `maxdistance = 2500` de la fonction `spNetwork::network_listw.mc`).

```
routes <- st_read('data/chap01/shp/Segments_de_rue.shp', quiet = TRUE)
routes <- st_transform(routes, 2949)
routes <- sf::st_cast(routes, 'LINESTRING')
future::plan(future::multisession(workers = 5))
net_distances <- spNetwork::network_listw.mc(
  origins = lixels_centers,
  lines = routes,
  maxdistance = 2500,
  mindist = 1,
  dist_func = 'identity',
  matrice_type = 'I',
```

```
  grid_shape = c(5, 5)
)
```

Premièrement, nous construisons plusieurs matrices de connectivité selon la distance standardisées en ligne, captant les voisins à moins de 250, 300, 500, 700 et 900 mètres (rappelons que les lixels ont une longueur de 100 m et que nous partons du centre des lixels pour le calcul des distances).

```
# une liste qui permettra de stocker toutes les matrices
all_matrices <- list()
bin_dists <- c(250, 300, 500, 700, 900)
for(d in bin_dists){
  new_weights <- lapply(net_distances$weights, function(x){
    return( (x <= d) / sum(x <= d))
  })
  net_distances_temp <- net_distances
  net_distances_temp$weights <- new_weights
  all_matrices[[paste0("dist_mat_",d)]] <- net_distances_temp
}
```

Deuxièmement, nous considérons un ensemble de matrices binaires standardisées en ligne captant les k plus proches voisins (de 3 à 10); il faut cependant que les distances entre les observations restent inférieures à 2500 mètres.

```
for(k in 3:10){
  new_weights <- lapply(net_distances$weights, function(x){
    x_r <- rank(x, ties.method = "min")
    return( (x_r <= k) / sum(x_r <= k))
  })
  net_distances_temp <- net_distances
  net_distances_temp$weights <- new_weights
  all_matrices[[paste0("k_mat_",k)]] <- net_distances_temp
}
```

Troisièmement, nous construisons des matrices avec l'inverse de la distance, l'inverse de la distance au carrée et un *kernel* quadratique avec des seuils maximaux de la distance fixés à 250, 300, 500, 700 et 900 mètres.

```
# Inverse de la distance
inv_weights <- lapply(net_distances$weights, function(x){
  inv <- (1/x)
  return(inv / sum(inv))
})
net_distances_temp <- net_distances
net_distances_temp$weights <- inv_weights
all_matrices[["inv_mat"]] <- net_distances_temp

inv2_weights <- lapply(net_distances$weights, function(x){
```

```
  inv <- (1/x**2)
  return(inv / sum(inv))
})

# Inverse de la distance au carré
net_distances_temp <- net_distances
net_distances_temp$weights <- inv2_weights
all_matrices[["inv2_mat"]] <- net_distances_temp

bin_dists <- c(250, 300, 500, 700, 900)

for(d in bin_dists){
  new_weights <- lapply(net_distances$weights, function(x){
    if(is.null(x) == FALSE){
      w <- spNetwork::quartic_kernel(x,d)
      return( (w) / sum(w))
    }
    else{
      return(NULL)
    }
  })
net_distances_temp <- net_distances
net_distances_temp$weights <- new_weights
all_matrices[[paste0("dist_quartic_",d)]] <- net_distances_temp
}
```

Maintenant que nous disposons de toutes ces matrices, nous pouvons calculer le I de Moran global pour chaque matrice.

```
library(spdep)
## Calcul du I de Moran pour les différentes matrices
moran_vals <- sapply(all_matrices, function(W){
  # Petite conversion vers le type de spdep
  attr(W,'class') <- c("listw", "nb")
  W$style <- "W"
  W$neighbours <- W$nb_list
  W$nb_list <- NULL
  val <- moran(lixels$density_adpt_knn, listw = W,
        n = nrow(lixels),
        S0 = nrow(lixels),
        zero.policy = TRUE)
  return(val$I)
})
## Enregistrement dans un dataframe
df_moran <- data.frame(
  Matrices = names(all_matrices),
  MoranIs = moran_vals
```

```
)
## Réalisation d'un graphique
ggplot(data=df_moran, aes(x=reorder(Matrices,MoranIs), y=MoranIs)) +
  geom_segment( aes(x=reorder(Matrices,MoranIs),
                    xend=reorder(Matrices,MoranIs),
                    y=0, yend=MoranIs)) +
  geom_point( size=4,fill="red",shape=21)+
  xlab("Matrice de pondération spatiale sur le réseau") +
  ylab("I de Moran")+
  coord_flip()
```

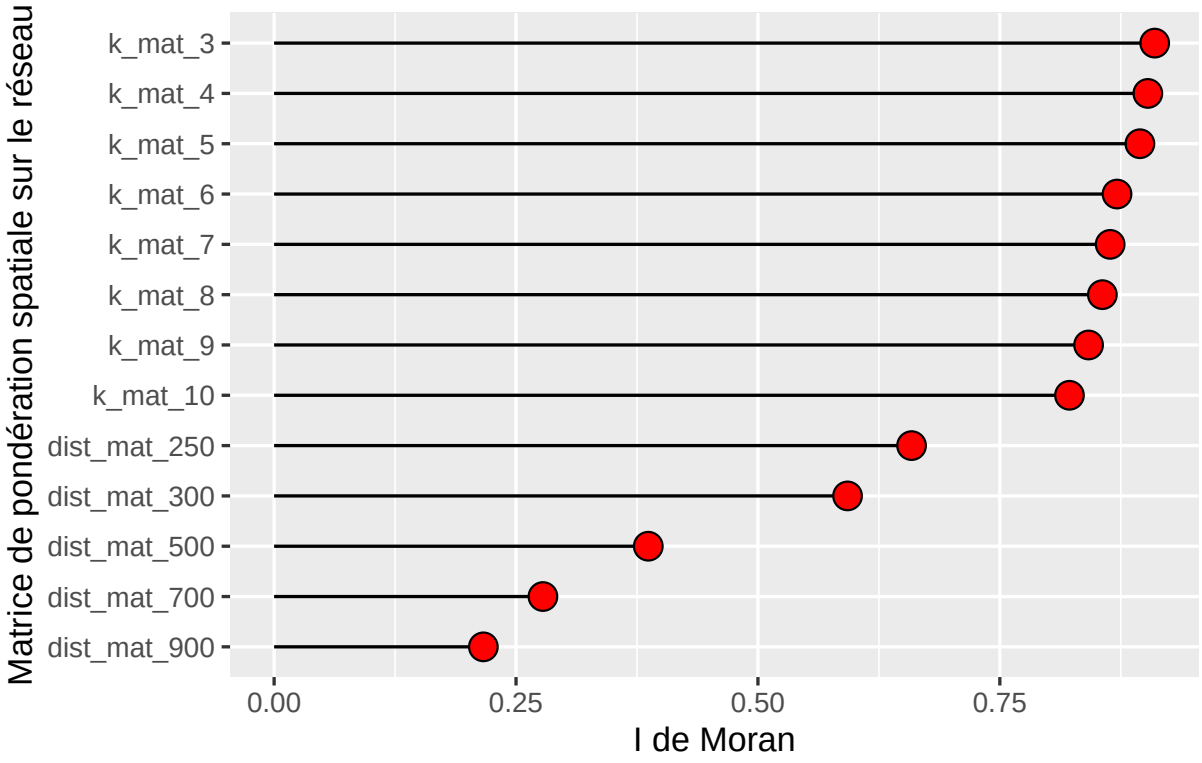


FIGURE 6.21 – *I* de Moran obtenu pour différentes matrices de pondération spatiale sur réseau

À la figure 6.21, nous constatons que la matrice utilisant les trois plus proches voisins obtient la valeur du *I* de Moran la plus élevé. Nous la conservons pour calculer la mesure d'autocorrélation spatiale locale basée sur la typologie basée sur le diagramme de Moran (figure 6.22).

```
W <- all_matrices$k_mat_3
attr(W,'class') <- c("listw", "nb")
W$style <- "W"
W$neighbours <- W$nb_list
W$nb_list <- NULL
local_I <- localmoran(lixels$density_adpt, listw = W, zero.policy = TRUE)
lixels$loc_classes <- attributes(local_I)$quadr$mean
```

```
lixels$loc_p <- local_I[,5]
lixels$loc_classes2 <- case_when(
  lixels$loc_p < 0.01 ~ lixels$loc_classes,
  TRUE ~ 'non sign.'
)
Couleurs <- c("High-High" = "#FF0000",
              "Low-Low" ="#0000FF",
              "High-Low" = "#f4ada8",
               "Low-High" ="#a7adf9",
              'non sign.' = "#eeeeee")
tm_shape(lixels) +
  tm_lines('loc_classes2', palette = Couleurs,
           lwd = 1.2, title.col = "Typologie")
```

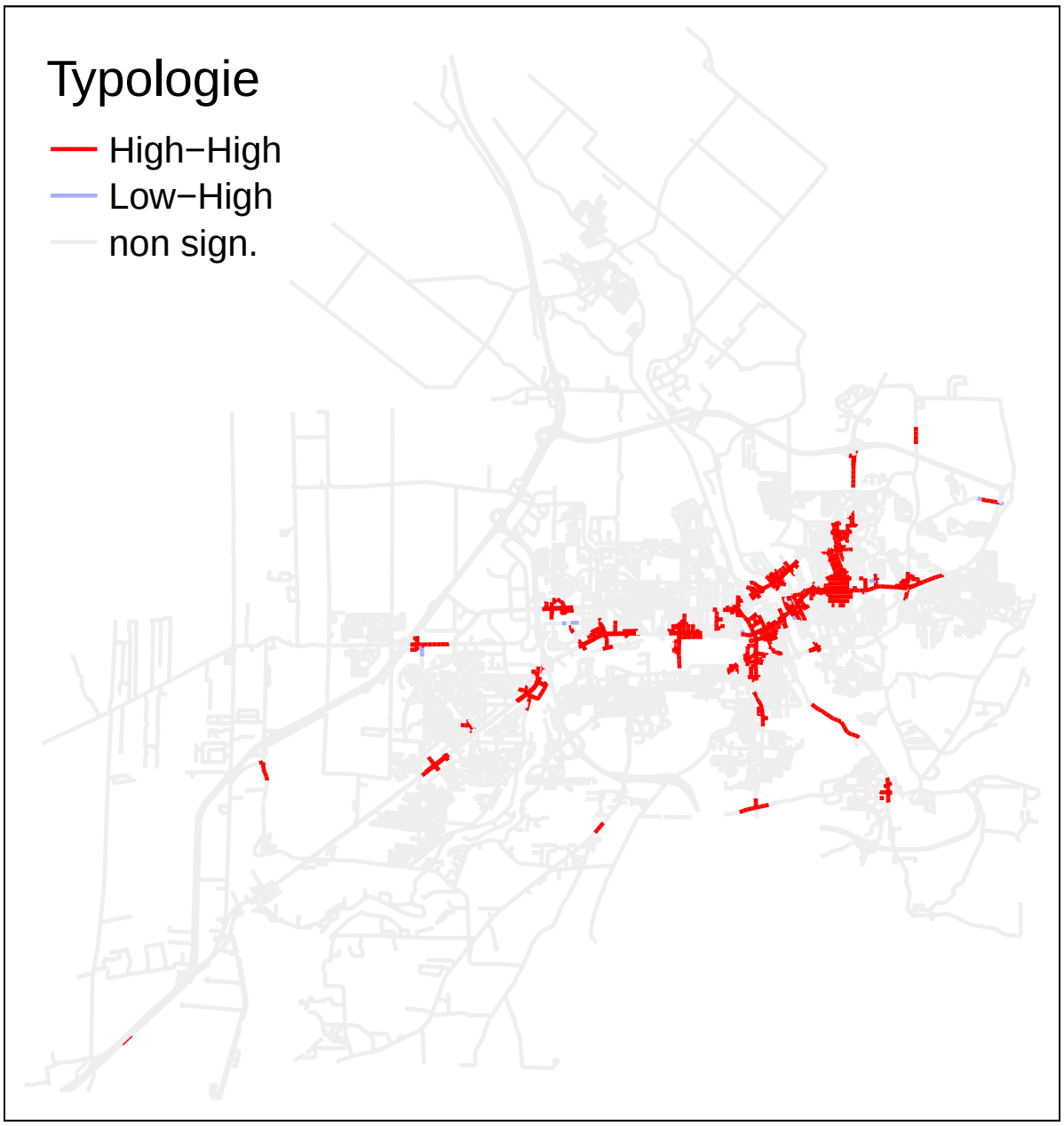


FIGURE 6.22 – Typologie basée sur le diagramme de Moran

Aller plus loin

Mesures d'autocorrélation spatiale globale et locale sur un réseau
Au chapitre 2, nous avons vu une panoplie de mesures d'autocorrélation spatiale globale (section 2.3) et locale (section 2.4) calculées avec différentes matrices de pondération spatiale (section 2.2).
Or, n'importe quelle mesure d'autocorrélation spatiale peut être calculée à partir d'une matrice de pondération spatiale construite à partir des distances réseau. Retenez que le choix de la matrice reste un enjeu important puisqu'elle affecte significativement les résultats comme illustré à la figure 6.21.

6.4 DBSCAN sur un réseau

Dans la section 4.1.1, Nous avons vu l'algorithme DBSCAN détectant des agrégats de points dans l'espace. Cet algorithme de classification non supervisée basée sur la densité des points identifie des agrégats de points à partir de deux paramètres : un rayon de recherche (ϵ, epsilon) et un nombre minimum de points ($MinPts$). Il est assez facile d'adapter cette méthode afin que le rayon de recherche ne soit pas basée sur la distance euclidienne, mais plutôt sur la distance réseau. Cela est particulièrement pertinent lors les évènements sont localisés sur un réseau, comme des accidents routiers.

6.4.1 Mise en œuvre dans R

Pour illustrer la version réseau du DBSCAN, nous reprenons l'exemple de la section 4.1.3 basé sur un jeu de données sur les incidents de sécurité publique survenus sur le territoire de la ville de Sherbrooke de juillet 2019 à juin 2022.

Tout d'abord, nous importons les couches géographiques des accidents et du réseau routier (figure 6.23).

```
library(sf)
library(tmap)
library(spNetwork)
library(dbscan)
library(ggplot2)
library(spdep)
## Importation des accidents
Accidents.sf <- st_read(dsn = "data/chap04/DataAccidentsSherb.shp", quiet=TRUE)
## Importation des routes (réseau)
routes <- st_read('data/chap01/shp/Segments_de_rue.shp', quiet=TRUE)
# reprojection dans le même système
routes <- st_transform(routes, 32187)
Accidents.sf <- st_transform(Accidents.sf, 32187)
routes <- sf::st_cast(routes, 'LINESTRING')
tm_shape(routes) +
  tm_lines('black') +
  tm_shape(Accidents.sf) +
  tm_dots('red', size = 0.2)
```

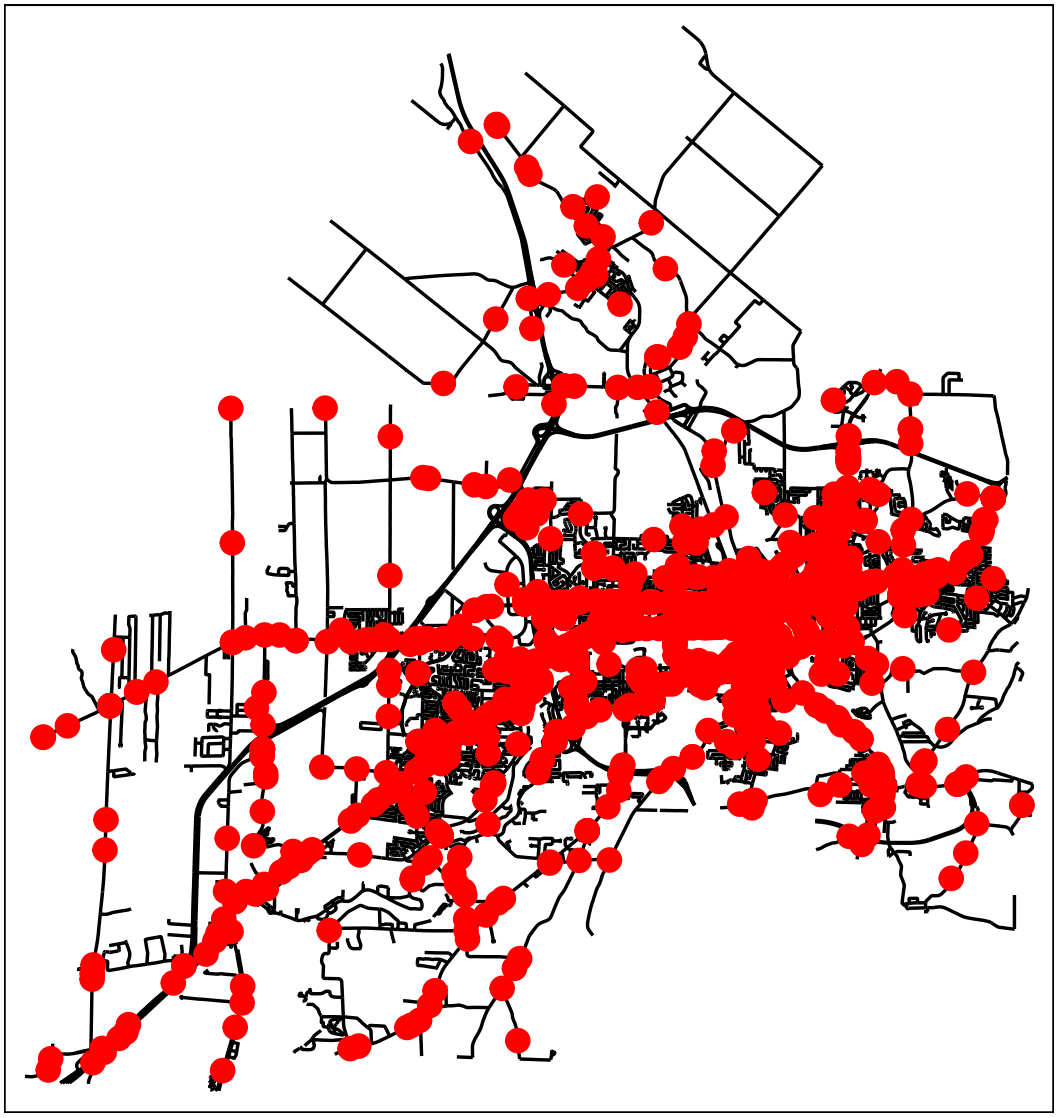


FIGURE 6.23 – Accidents sur le réseau de la ville de Sherbrooke

Puis, nous recherchons la valeur optimale d'epsilon (ϵ) avec un nombre minimal de quatre points ($MinPts$) pour former un agrégat. À la lecture de la figure 6.24, nous constatons que le coude se situe certainement entre 750 et 1250 mètres.

⚠ **Attention**

Matrice de pondération spatiales des distances réseau avec `spNetwork` et `spdep`
Dans le code ci-dessous, les distances réseau sont obtenues avec le *package* `spNetwork`, sous forme d'un objet de type `listw`. Elles sont ensuite converties en matrice avec la fonction `listw2mat` du *package* `spdep`, et enfin en objet de type `distance` avec la fonction `as.dist`. Si le nombre de points à analyser est très grand, la matrice de distances pourrait excéder la mémoire vive disponible dans votre ordinateur.

```
## Calcul des distances pour les quatre plus proches voisins
knn_dists <- network_knn(origins = Accidents.sf,
```

```
                        lines = routes,
                        k = 4,
                        maxdistance = 5000,
                        grid_shape = c(1,1),
                        verbose = FALSE)
## Graphique pour la distance au quatrième voisin le plus proche
dists4 <- knn_dists$distances[,4]
dists4 <- dists4[order(dists4)]
DistKplusproche <- data.frame(
  distance = dists4,
  id = 1:length(dists4)
)
ggplot(data = DistKplusproche)+
  geom_path(aes(x = id, y = distance), size=1)+
  labs(x = "Points triés par ordre croissant selon la distance",
       y = "Distance au quatrième point le plus proche")+
  geom_hline(yintercept=250, color = "#08306b", linetype="dashed", size=1)+
  geom_hline(yintercept=500, color = "#00441b", linetype="dashed", size=1)+
  geom_hline(yintercept=1000, color = "#67000d", linetype="dashed", size=1)+
  geom_hline(yintercept=1500, color = "#3f007d", linetype="dashed", size=1)
```

Figure 6.24 – Optimisation de la valeur d'epsilon pour les accidents sur réseau

La comparaison des figures 6.25, 6.26, 6.27 et 6.28 montre clairement qu'au-delà de 1000 mètres, les résultats sont assez peu intéressants, car presque tous les points sont agrégés dans le même groupe.

```
## Calcul des distances réseau entre chaque évènement
network_dists <- network_listw(origins = Accidents.sf,
                               lines = routes,
```

```
                                maxdistance = 5000,
                                mindist = 1,
                                dist_func = "identity",
                                matrice_type = "I",
                                grid_shape = c(1,1))

network_dists$neighbours <- network_dists$nb_list
network_dists_matrix <- listw2mat(network_dists)

# 0 signifie que la distance entre les deux points
# est plus grande que le paramètre maxdistance
network_dists_matrix <- ifelse(network_dists_matrix == 0,
                               5000, network_dists_matrix)
network_dists_matrix <- as.dist(network_dists_matrix)

# Algorithme dbscan avec les différentes valeurs d'epsilon
result250 <- dbscan(network_dists_matrix, eps = 250, minPts = 4)
result500 <- dbscan(network_dists_matrix, eps = 500, minPts = 4)
result1000 <- dbscan(network_dists_matrix, eps = 1000, minPts = 4)
result1500 <- dbscan(network_dists_matrix, eps = 1500, minPts = 4)
Accidents.sf$gp_250 <- as.character(result250$cluster)
Accidents.sf$gp_500 <- as.character(result500$cluster)
Accidents.sf$gp_1000 <- as.character(result1000$cluster)
Accidents.sf$gp_1500 <- as.character(result1500$cluster)

# Cartographie
tmap_options(max.categories = 50)
tm_shape(Accidents.sf)+tm_dots(col="gp_250", title = "DBSCAN 250", size = .5)
```

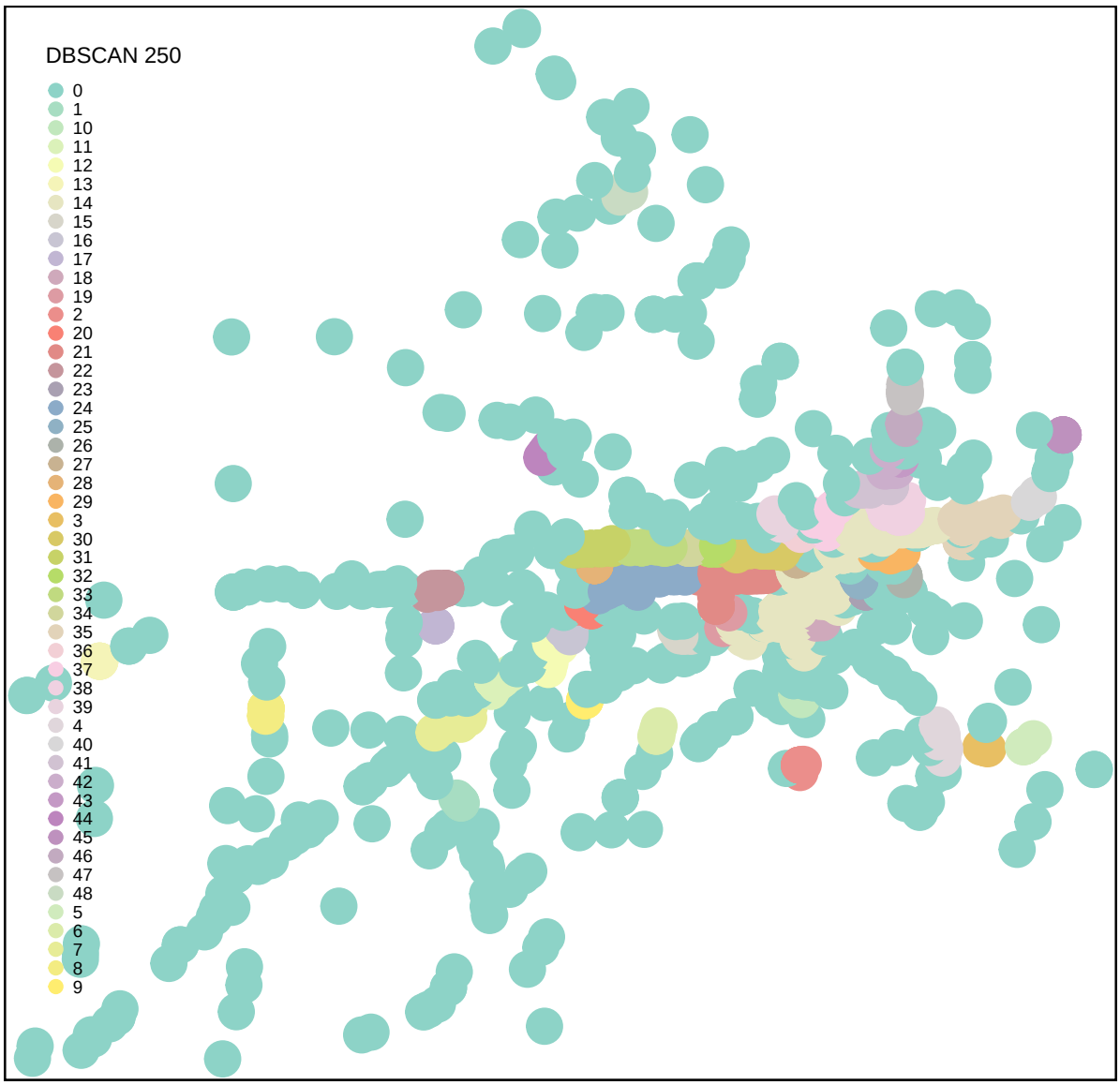


FIGURE 6.25 – Résultats obtenus pour le dbscan avec un valeur d'epsilon de 250 mètres

```
tm_shape(Accidents.sf)+tm_dots(col="gp_500", title = "DBSCAN 500", size = .5)
```

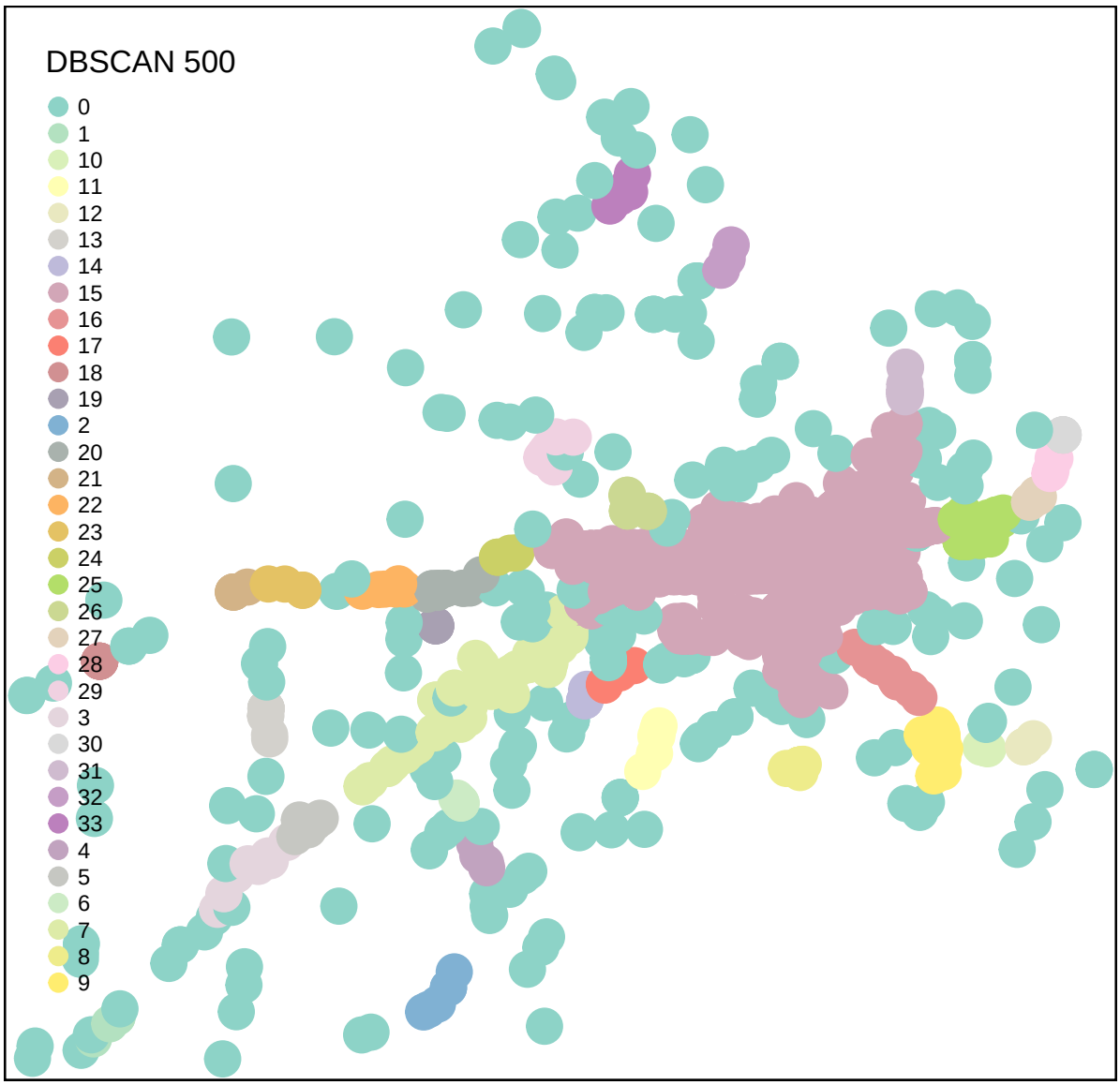


FIGURE 6.26 – Résultats obtenus pour le dbscan avec un valeur d'epsilon de 500 mètres

```
tm_shape(Accidents.sf)+tm_dots(col="gp_1000", title = "DBSCAN 1000", size = .5)
```

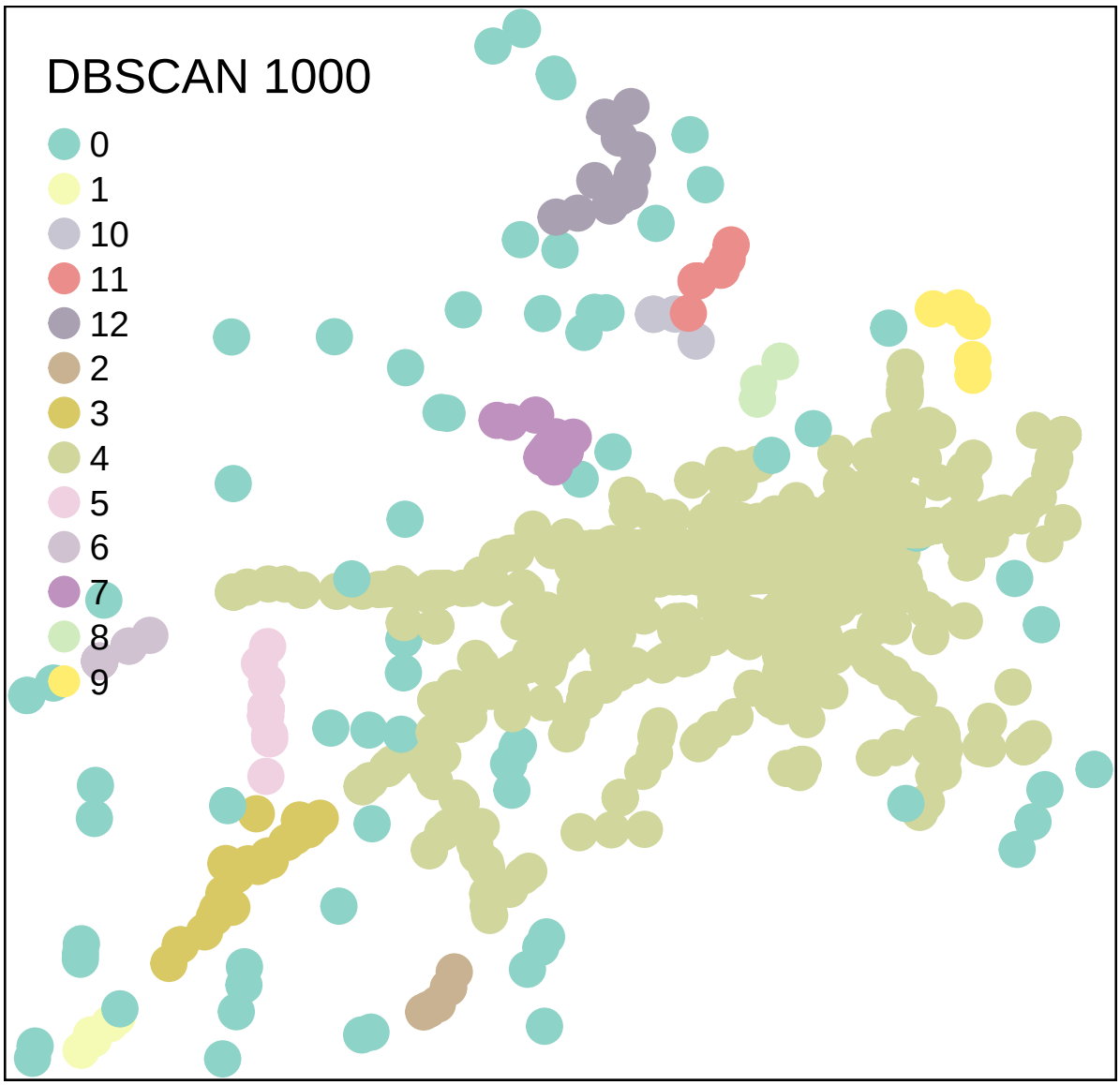


FIGURE 6.27 – Résultats obtenus pour le dbscan avec un valeur d'epsilon de 1000 mètres

```
tm_shape(Accidents.sf)+tm_dots(col="gp_1500", title = "DBSCAN 1500", size = .5)
```

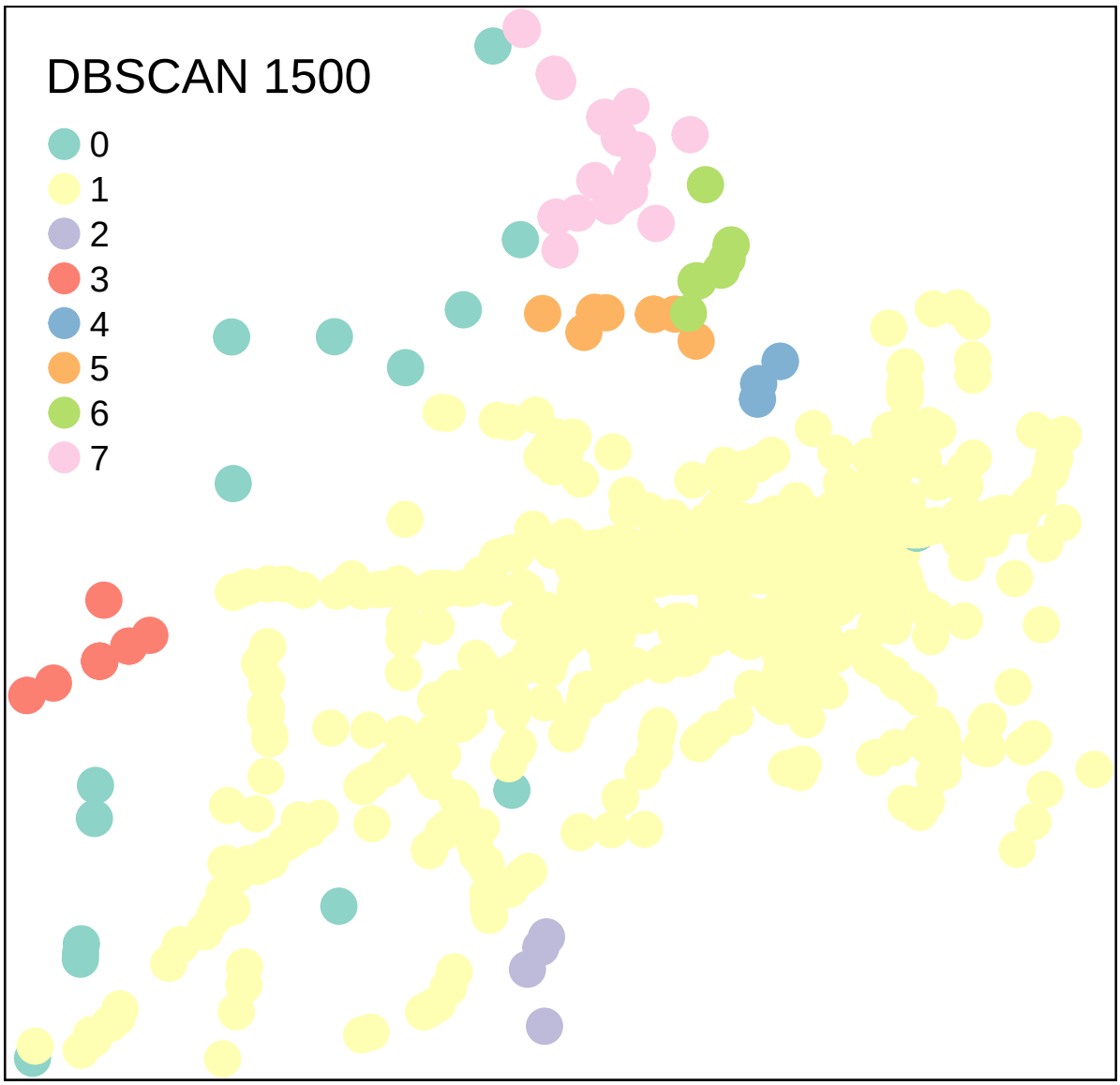


FIGURE 6.28 – Résultats obtenus pour le dbscan avec un valeur d'epsilon de 1500 mètres

6.5 Quiz de révision du chapitre

Questions

- **Quels problèmes posent l'utilisation des méthodes classiques d'analyse spatiale pour des données se produisant sur un réseau?**
  - La distance euclidienne tend à sous-estimer les distances réelles.
  - La distance euclidienne tend à surestimer les distances réelles.
  - Le réseau est un espace comportant davantage de dimensions qu'un espace planaire.
  - Le réseau est un espace ne respectant pas les hypothèses d'homogénéité et d'isotropie de l'espace planaire.
  - Les méthodes classiques analysent des portions de l'espace en dehors du réseau.

Relisez au besoin la section 6.1.

— **Pour adapter la méthode classique du Kernel Density Estimate, il est nécessaire de :**

  — Utiliser une distance réseau plutôt qu'euclidienne.
  — N'estimer les densités que le long des lixels représentant des fragments réguliers du réseau.
  — Adapter la fonction de Kernel pour tenir compte de la division de la masse aux intersections.
  — Appliquer une pénalité aux portions du réseau avec peu d'évènements.

Relisez au besoin le début de la section 6.2.

— **Pour une NKDE, il est uniquement possible de calculer des densités selon une bandwidth fixe.**

  — Vrai
  — Faux

Relisez au besoin la section 6.2.1.3.

— **Contrairement à la KDE classique, la NKDE est moins sensible au choix de la bandwidth.**

  — Vrai
  — Faux

Relisez au besoin la section 6.2.1.4.

— **L'extension spatio-temporelle du NKDE (TNKDE) consiste à :**

  — Calculer une somme des densités spatiales et temporelles des évènements le long du réseau.
  — Calculer le produit des densités spatiales et temporelles des évènements le long du réseau.
  — Calculer l'intégrale des densités spatiales et temporelles des évènements le long du réseau.

Relisez au besoin la section 6.2.3.

— **Pour une TNKDE, il est uniquement possible de calculer des densités selon une bandwidth fixe.**

  — Vrai
  — Faux

Relisez au besoin la section 6.2.1.

— **Les méthodes KDE, NKDE, et TNKDE peuvent être vues comme :**

  — Des méthodes descriptives visant à résumer un grand ensemble d'évènements en une carte de chaleur plus facilement interprétable.
  — Des méthodes inférentielles permettant de déterminer statistiquement la présence de points chauds ou de points froids.
  — Des algorithmes de type boite noire réduisant la dimensionnalité des données.

Relisez au besoin la section 6.2.

— **L'utilisation d'une distance réseau plutôt qu'euclidienne pour un algorithme DBSCAN risque de :**

  — Augmenter la taille des groupes détectés par l'algorithme, car la recherche de voisin sera plus facile sur le réseau
  — Réduire la taille des groupes formés par l'algorithme, car la distance euclidienne tend à sous-estimer les distances réelles sur le réseau

Relisez au besoin la section 6.4.

Réponses

— Quels problèmes posent l'utilisation des méthodes classiques d'analyse spatiale pour des données se produisant

sur un réseau?
  — La distance euclidienne tend à sous-estimer les distances réelles.
  — Le réseau est un espace ne respectant pas les hypothèses d'homogénéité et d'isotropie de l'espace planaire.
  — Les méthodes classiques analysent des portions de l'espace en dehors du réseau.
— Pour adapter la méthode classique du Kernel Density Estimate, il est nécessaire de :
  — Utiliser une distance réseau plutôt qu'euclidienne.
  — N'estimer les densités que le long des lixels représentant des fragments réguliers du réseau.
  — Adapter la fonction de Kernel pour tenir compte de la division de la masse aux intersections.
— Pour une NKDE, il est uniquement possible de calculer des densités selon une bandwidth fixe.
  — Faux
— Contrairement à la KDE classique, la NKDE est moins sensible au choix de la bandwidth.
  — Faux
— L'extension spatio-temporelle du NKDE (TNKDE) consiste à :
  — Calculer le produit des densités spatiales et temporelles des évènements le long du réseau.
— Pour une TNKDE, il est uniquement possible de calculer des densités selon une bandwidth fixe.
  — Faux
— Les méthodes KDE, NKDE, et TNKDE peuvent être vues comme :
  — Des méthodes descriptives visant à résumer un grand ensemble d'évènements en une carte de chaleur plus facilement interprétable.
— L'utilisation d'une distance réseau plutôt qu'euclidienne pour un algorithme DBSCAN risque de :
  — Réduire la taille des groupes formés par l'algorithme, car la distance euclidienne tend à sous-estimer les distances réelles sur le réseau

6.6 Exercices de révision

Exercice

Exercice 1. Réalisation d'un graphique pour trouver la valeur de la *bandwidth* optimale
Utilisez la fonction *kernel* quadratique et la NKDE continue (ESC-NKDE), contruisez un graphique pour choisir une *bandwidth* avec l'approche par validation croisée des probabilités. Complétez le code ci-dessous.

```
library(sf)
library(spNetwork)
library(future)

future::plan(future::multisession(workers = 5))
# Importation des données sur les collisions cycles et le réseau de rues
Collisions <- st_read(dsn = "data/chap06/Mtl/DonneesMTL.gpkg", layer="CollisionsAvecCyclistes", quiet=TRUE)
ReseauRues <- st_read(dsn = "data/chap06/Mtl/DonneesMTL.gpkg", layer="Rues", quiet=TRUE)
ReseauRues$LineID <- 1:nrow(ReseauRues)
LongueurKm <- sum(as.numeric(st_length(ReseauRues)))/1000
Collisions <- st_transform(Collisions, st_crs(ReseauRues))
cat("Informations sur les couches",
    "\n  Collisions avec cylistes :", nrow(Collisions),
    "\n  Réseau :", round(LongueurKm,3), "km")
# Cartographie
tmap_mode("view")
tm_shape(ReseauRues) + tm_lines("black") +
  tm_shape(Collisions) + tm_dots("blue", size = 0.025)+
tm_scale_bar(c(0,1,2), position = 'left')+
  tm_layout(frame = FALSE)
## Évaluation des bandwidths de 100 à 1200 avec un saut de 50
eval_bandwidth <- bw_cv_likelihood_calc.mc(à compléter)
## Graphique pour les bandwidths
à compléter
```

Correction à la section 12.6.1.

Exercice

Exercice 2. Réalisation d'une NKDE continue.
Complétez le code ci-dessous pour réaliser une NKDE continue avec un fonction *kernel* quadratique et une valeur de *bandwidth* de 500 mètres.

```
library(sf)
library(spNetwork)
library(future)
## Création des lixels d'une longueur de 100 mètres
lixels <- lixelize_lines(ReseauRues, 100, mindist = 50)
lixels_centers <- spNetwork::lines_center(lixels)
## Calcul de la NKDE continue
intensity <- nkde.mc(À compléter)
lixels$density <- intensity * 1000
## Cartographie
À compléter
```

Correction à la section 12.6.2.

Partie 5. Régressions spatiales et classifications spatiales

7 Introduction aux modèles de régression spatiale

Depuis une trentaine d'années, économètres, épidémiologistes et géographes développent et utilisent abondamment des méthodes de régression intégrant l'espace : modèles économétriques spatiaux, modèles géographiquement pondérés, analyses multiniveaux, etc. L'objectif de ce chapitre est de donner un aperçu de ces méthodes. Nous y décrivons principalement les différents modèles économétriques spatiaux, les modèles généralisés additifs avec une *spline* bivariée sur les coordonnées géographiques et les modèles géographiquement pondérés. Afin d'explorer un plus large éventail de méthodes de régression spatiale, nous vous recommandons la lecture d'un autre ouvrage de la Série *Un grand Bol d'R* intitulé *Méthodes de régression spatiale : un grand bol d'R* (Apparicio et Gelb 2025).

Package

Liste des *packages* utilisés dans ce chapitre

- Pour importer et manipuler des fichiers géographiques :
  - `sf` pour importer et manipuler des données vectorielles.
  - `raster` et `terra` pour manipuler des données matricielles.
- Pour construire des cartes et des graphiques :
  - `tmap` est certainement le meilleur *package* pour la cartographie.
  - `ggplot2` pour construire des graphiques.
- Pour construire des modèles spatiaux :
  - `spdep` pour construire des matrices de pondération spatiales et calculer le I de Moran.
  - `spatialreg` pour construire des modèles économétriques spatiaux.
  - `mgcv` pour construire des modèles généralisés additifs avec une *spline* sur les coordonnées géographiques.
  - `spgwr` pour construire des régressions géographiquement pondérées.

Pour décrire les différents modèles, nous proposons d'utiliser le jeu de données spatiales `LyonIris` du *package* `geocmeans`. Ce jeu de données spatiales pour l'agglomération lyonnaise (France) comprend dix variables, dont quatre environnementales (EN) et six socioéconomiques (SE), pour les îlots regroupés pour l'information statistique (IRIS) de l'agglomération lyonnaise (tableau 7.1 et figure 7.1).

Tableau 7.1 – Statistiques descriptives du jeu de données LyonIris

	Nom	Intitulé	Type	Moy.	E.-T.	Min.	Max.
Lden	Lden	Bruit routier (Lden dB(A))	EN	55,6	4,9	33,9	71,7
NO2	NO2	Dioxyde d'azote (ug/m^3)	EN	28,7	7,9	12,0	60,2
PM25	PM25	Particules fines ($PM_{2,5}$)	EN	16,8	2,1	11,3	21,9
VegHautPrt	VegHautPrt	Canopée (%)	EN	18,7	10,1	1,7	53,8
Pct0_14	Pct0_14	Moins de 15 ans (%)	SE	18,5	5,7	0,0	54,0
Pct_65	Pct_65	65 ans et plus (%)	SE	16,2	5,9	0,0	45,1
Pct_Img	Pct_Img	Immigrants (%)	SE	14,5	9,1	0,0	59,8
TxChom1564	TxChom1564	Taux de chômage	SE	14,8	8,1	0,0	98,8

Tableau 7.1 – Statistiques descriptives du jeu de données LyonIris

	Nom	Intitulé	Type	Moy.	E.-T.	Min.	Max.
Pct_brevet	Pct_brevet	Personnes à faible scolarité (%)	SE	23,5	12,6	0,0	100,0
NivVieMed	NivVieMed	Médiane du niveau de vie (milliers d'euros)	SE	21,8	4,9	11,3	38,7

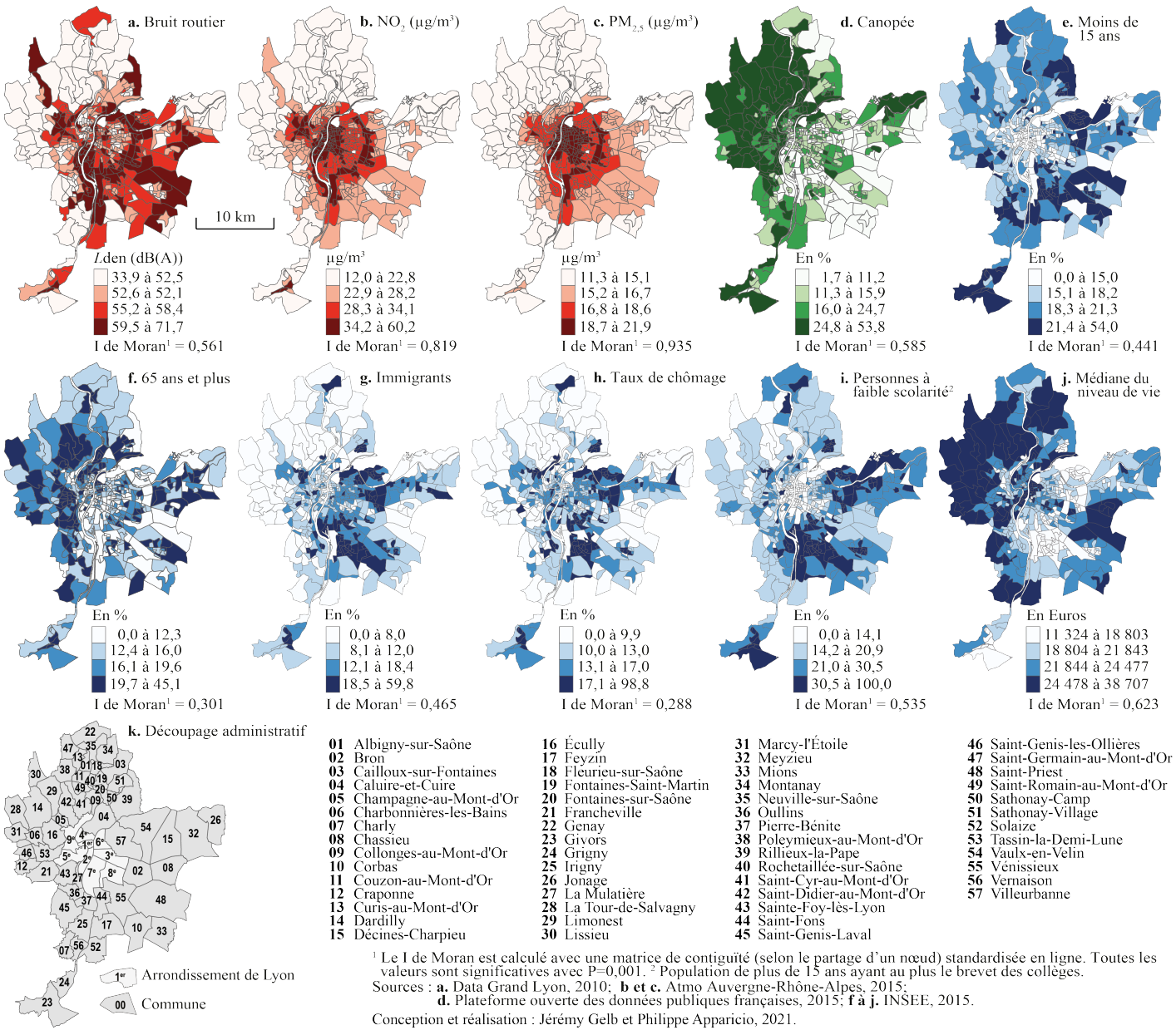


Figure 7.1 – Cartographie des variables du jeu de données LyonIris

7.1 Modèles économétriques spatiaux

⚠ **Attention**

Régression linéaire multiple et modèles économétriques spatiaux
Dans cette section, nous décrivons uniquement les modèles économétriques spatiaux dont la variable dépendante est continue. Sommairement, ces modèles sont des extensions de la régression linéaire multiple dans laquelle est intégrée l'autocorrélation spatiale. Avant de lire cette section, il faut donc bien maîtriser la régression linéaire multiple. Si ce n'est pas le cas, nous vous invitons vivement à lire le chapitre suivant (Apparicio et Gelb 2022). Ces deux dernières décennies, plusieurs ouvrages traitant des modèles économétriques spatiaux ont été publiés, surtout en anglais (LeSage et Pace 2008; Anselin et Rey 2014; Bivand et al. 2008). Ils méritent grandement d'être consultés, tout comme l'excellent livre en français de Jean Dubé et Diègo Legros (2014).

Pourquoi recourir à des modèles économétriques spatiaux?

Dans un modèle, les résidus (ϵ) sont la différence entre les valeurs observées (y_i) et les valeurs prédites par le modèle ($\widehat{y_i}$). Une des hypothèses de la régression linéaire multiple est que les observations doivent être indépendantes les unes des autres (*indépendance du terme d'erreur*). Le non-respect de cette hypothèse produit des résultats biaisés, notamment pour les coefficients de régression.

Lorsque les observations sont des entités spatiales (polygones, points par exemple), si les résidus du modèle sont autocorrélés spatialement, il y a un problème de dépendance spatiale du modèle. Autrement dit, les observations ne sont pas spatialement indépendantes les unes des autres. Pour vérifier la dépendance spatiale d'un modèle, il suffit de calculer le *I* de Moran sur les résidus du modèle, comme décrit au chapitre 2 (section 2.3).

Autrement dit, un modèle de régression construit avec des données spatiales ne devrait pas avoir des résidus spatialement autocorrélés. Or, les modèles économétriques spatiaux permettent justement d'intégrer l'autocorrélation spatiale de différentes manières afin de s'assurer que l'hypothèse de l'indépendance du terme d'erreur est respectée.

7.1.1 Bref retour sur la régression linéaire multiple

À titre de rappel, la régression linéaire multiple permet de prédire et d'expliquer une variable dépendante (Y) en fonction de plusieurs variables indépendantes (X). L'équation de régression s'écrit alors :

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \ldots + \beta_k x_{ki} + \epsilon_i \tag{7.1}$$

avec :

- y_i, la valeur de la variable dépendante Y pour l'observation *i*.
- β_0, la constante, soit la valeur prédite pour Y quand toutes les variables indépendantes sont égales à 0.
- k le nombre de variables indépendantes.
- β_1 à β_k, les coefficients de régression pour les variables indépendantes de 1 à *k* (X_1 à X_k).
- ϵ_i, le résidu pour l'observation de *i*, soit la partie de la valeur de y_i qui n'est pas expliquée par le modèle de régression.

Il existe plusieurs écritures simplifiées de cette équation. Dans le cadre de ce chapitre, nous utilisons la forme matricielle suivante :

$$y = X\beta + \epsilon \tag{7.2}$$

avec :

- y, un vecteur de dimension $n \times 1$ pour la variable dépendante, soit une colonne avec *n* observations.
- X, une matrice de dimension $n \times (k+1)$ pour les *k* variables indépendantes, incluant une autre colonne (avec la valeur de 1 pour les *n* observations) pour la constante, d'où $k+1$.
- β, un vecteur de dimension $k+1$, soit les coefficients de régression pour les *k* variables et la constante.
- ϵ, un vecteur de dimension $n \times 1$ pour les résidus.

Construction du modèle MCO dans R

Avec la fonction `lm()`, il est facile de construire un modèle de régression linéaire multiple basé sur la méthode des moindres carrés ordinaires (MCO). Dans le code ci-dessous, la formule de l'équation du modèle est donc `NO2 ~ Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed`. Notez que la variable dépendante et les variables indépendantes sont séparées avec un tilde (`~`). Quant à la fonction `summary(NomDuModèle)`, elle affiche les résultats du modèle.

```
# Appel des différents packages utilisés dans le chapitre
library(spdep)
## Construction du modèle
Modele.MCO <- lm(NO2 ~ Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed,
                 data = LyonIris)
## Résultats du modèle
summary(Modele.MCO)
```

```
Call:
lm(formula = NO2 ~ Pct0_14 + Pct_65 + Pct_Img + Pct_brevet +
    NivVieMed, data = LyonIris)

Residuals:
    Min      1Q  Median      3Q     Max
-27.733  -4.457  -0.499   3.507  29.160

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 49.43296    2.99550  16.502  < 2e-16 ***
Pct0_14     -0.53352    0.06305  -8.461 2.94e-16 ***
Pct_65      -0.15047    0.05627  -2.674  0.00774 **
Pct_Img      0.28287    0.05113   5.532 5.12e-08 ***
Pct_brevet  -0.24004    0.03721  -6.451 2.63e-10 ***
NivVieMed   -0.31625    0.10180  -3.107  0.00200 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 6.685 on 500 degrees of freedom
Multiple R-squared:  0.2832,    Adjusted R-squared:  0.276
F-statistic:  39.5 on 5 and 500 DF,  p-value: < 2.2e-16
```

Dépendance spatiale du modèle MCO?

Pour vérifier si ce modèle linéaire multiple a un problème de dépendance spatiale, nous calculons le I de Moran sur ses résidus avec la fonction `lm.morantest`, puis nous les cartographions.

```
## Matrice de contiguïté selon le partage d'un segment (Rook)
Rook <- poly2nb(LyonIris, queen=FALSE)
W.Rook <- nb2listw(Rook, zero.policy=TRUE, style = "W")
# Autocorrélation spatiale globale des résidus
lm.morantest(Modele.MCO, W.Rook, alternative="two.sided")
```

```
    Global Moran I for regression residuals

data:
model: lm(formula = NO2 ~ Pct0_14 + Pct_65 + Pct_Img + Pct_brevet +
NivVieMed, data = LyonIris)
weights: W.Rook

Moran I statistic standard deviate = 21.266, p-value < 2.2e-16
alternative hypothesis: two.sided
sample estimates:
Observed Moran I      Expectation         Variance
     0.587312061     -0.005375800      0.000776745
```

Avec une valeur du I de Moran de 0,587 (p < 0,001), les résidus sont fortement autocorrélés spatialement, traduisant ainsi un problème de dépendance spatiale du modèle MCO et la nécessité de recourir à des modèles économétriques spatiaux. La cartographie des résidus à la figure 7.2 corrobore ce résultat.

```
library(tmap)
## Ajout de la colonne dans LyonIris avec les valeurs des résidus
LyonIris$ModeleMCO.Residus <- residuals(Modele.MCO)
## Cartographie
tmap_mode("plot")
tm_shape(LyonIris)+
  tm_fill(col="ModeleMCO.Residus", n = 5, style = "quantile",
          legend.format = list(text.separator = "à"),
          palette = "-RdBu", title = "Résidus") +
  tm_layout(frame=FALSE) +
  tm_scale_bar(breaks = c(0,5))
```

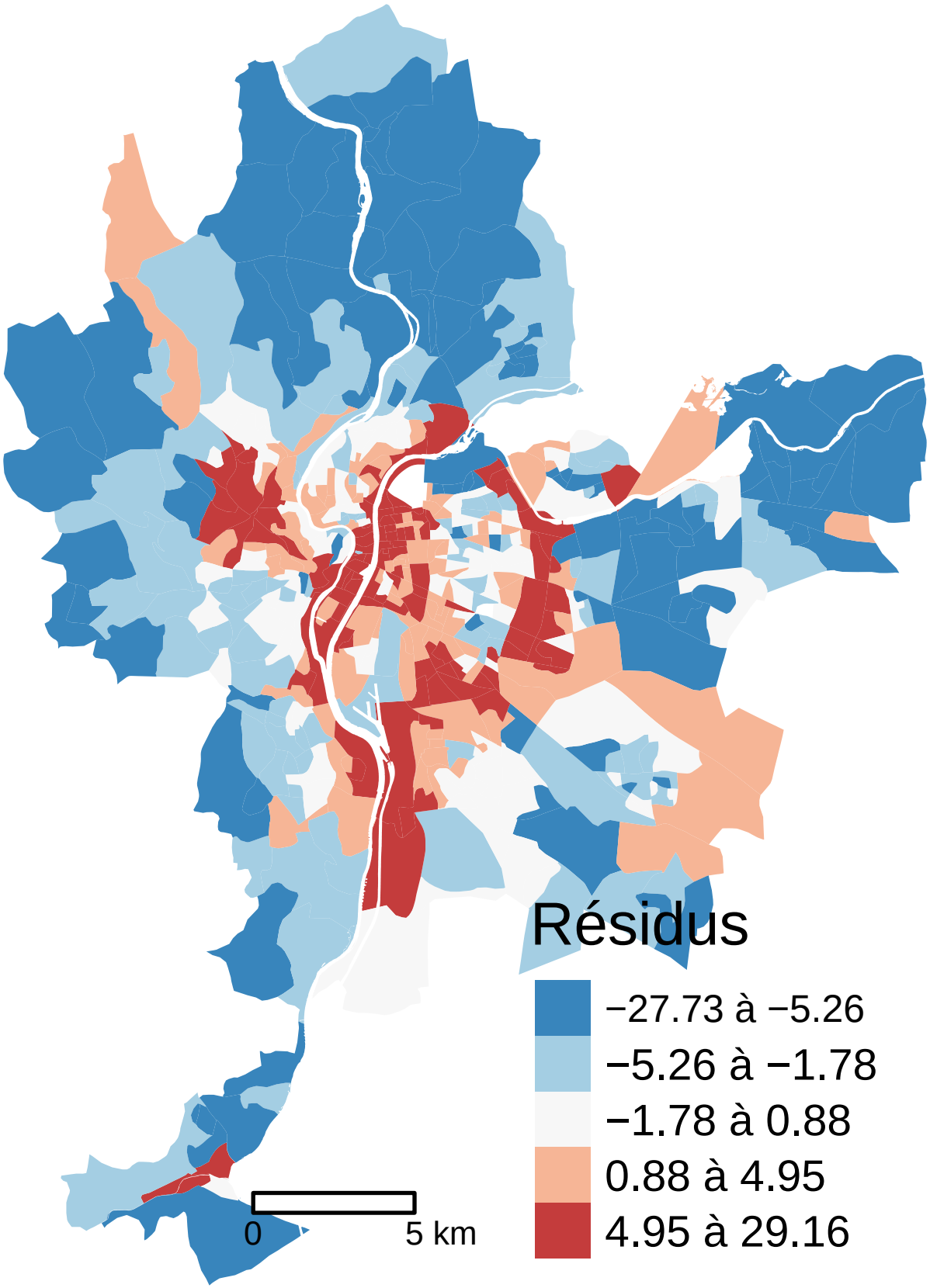


FIGURE 7.2 – Cartographie des résidus du modèle de régression multiple

7.1.2 Les différents modèles spatiaux autorégressifs

Selon Jean Dubé et Diègo Legros, « cinq raisons peuvent motiver le choix d'un modèle autorégressif : la présence d'externalités, les effets d'entraînement, l'omission de variables importantes, la présence d'hétérogénéité spatiale des comportements, les effets mixtes » (2014, 120). Les effets mixtes peuvent être la combinaison d'externalités avec des effets d'entraînement ou encore d'externalités avec l'omission d'une ou de plusieurs variables importantes spatialement structurées.

7.1.2.1 Modèle SLX : autocorrélation spatiale sur les variables indépendantes

Dans un modèle SLX, l'autocorrélation spatiale est intégrée au niveau des variables indépendantes. Autrement dit, les variables indépendantes spatialement décalées (WX) sont introduites aussi dans le modèle. Par conséquent, la valeur de chaque unité spatiale du modèle est ainsi expliquée à la fois par ses propres caractéristiques et celles dans le voisinage ou à proximité en fonction de la matrice de pondération spatiale (W).

Aller plus loin

Rappel sur les variables spatialement décalées
Dans le chapitre 2 sur l'autocorrélation spatiale, nous avons vu comment calculer une variable spatialement décalée avec une matrice de pondération spatiale (figure 2.29). À titre de rappel, lorsque cette dernière est standardisée en ligne, elle correspond à la valeur moyenne dans le voisinage.

L'idée est alors d'introduire des **externalités** puisque les caractéristiques des entités spatiales proches ou voisines peuvent avoir un effet sur la variable dépendante (Dubé et Legros 2014). L'équation du modèle SLX, qui est estimée selon la méthode des moindres carrés ordinaires (comme la régression linéaire multiple), s'écrit alors :

$$y = X\beta + WX\theta + \epsilon \tag{7.3}$$

avec :

- y, la variable dépendante.
- X, les variables indépendantes.
- β, les coefficients des variables indépendantes.
- W, la matrice de pondération spatiale.
- WX, les variables indépendantes spatiales décalées.
- θ, les coefficients des variables indépendantes spatiales décalées.
- ϵ, les résidus.

Construction du modèle SLX dans R

Le modèle SLX est construit avec la fonction `lmSLX` du *package* `spatialreg` (Bivand, Millo et Piras 2021). Remarquez, dans le code ci-dessous, le paramètre `listw=W.Rook` qui est utilisé pour spécifier la matrice de pondération spatiale.

```
library(spatialreg)
## Construction du modèle
Modele.SLX <- lmSLX(NO2 ~ Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed,
                    listw=W.Rook,     # matrice de pondération spatiale
```

```
                  data = LyonIris)  # dataframe
## Résultats du modèle
summary(Modele.SLX)
```

```
Call:
lm(formula = formula(paste("y ~ ", paste(colnames(x)[-1], collapse = "+"))),
    data = as.data.frame(x), weights = weights)

Coefficients:
               Estimate    Std. Error  t value     Pr(>|t|)
(Intercept)     5.068e+01   4.188e+00   1.210e+01   1.040e-29
Pct0_14        -2.040e-01   6.268e-02  -3.255e+00   1.211e-03
Pct_65         -3.771e-02   5.361e-02  -7.033e-01   4.822e-01
Pct_Img         1.041e-01   4.849e-02   2.146e+00   3.235e-02
Pct_brevet     -7.363e-02   3.550e-02  -2.074e+00   3.857e-02
NivVieMed      -1.844e-01   1.106e-01  -1.667e+00   9.617e-02
lag.Pct0_14    -7.759e-01   1.030e-01  -7.537e+00   2.315e-13
lag.Pct_65     -6.454e-02   9.115e-02  -7.081e-01   4.792e-01
lag.Pct_Img     6.465e-01   8.593e-02   7.524e+00   2.526e-13
lag.Pct_brevet -3.013e-01   6.157e-02  -4.893e+00   1.344e-06
lag.NivVieMed  -1.805e-02   1.750e-01  -1.031e-01   9.179e-01
```

⚠ **Attention**

Effets directs, indirects et totaux

La formulation d'un modèle SLX implique deux types d'effets pour les variables indépendantes (X) :

- les **effets directs**, soit ceux des caractéristiques des entités spatiales. Ils correspondent aux coefficients β des variables indépendantes (X). Autrement dit, pour une observation i, à chaque augmentation d'une unité d'une caractéristique X, la valeur de y_i va varier (augmenter ou diminuer) en fonction du coefficient β.
- les **effets indirects**, soit ceux des caractéristiques des entités spatiales voisines ou proches définies selon la matrice de pondération spatiale. Ils correspondent aux coefficients θ des variables indépendantes spatialement décalées (WX). Autrement dit, les valeurs de WX des entités spatiales proches ou voisines j de i vont aussi être amenées à varier, impactant alors les valeurs y_j selon les coefficients θ.

Prenons l'exemple d'un modèle visant à prédire le prix de vente des maisons dans une ville en fonction de leurs caractéristiques des maisons, dont la superficie du jardin. Il est probable que plus la superficie du jardin de la maison i augmente, plus le prix de vente augmente également (**effet direct**, coefficient β). Cette augmentation de la taille du jardin aura aussi un impact sur le prix des maisons voisines puisque leur prix est dépendant de la taille des jardins des maisons voisines. Ainsi, chaque maison j, voisine de i verra son prix augmenter à cause de l'augmentation de la taille du jardin de la maison i (**effet indirect**).

Pour capturer l'impact total sur le prix des maisons d'une augmentation de la superficie du jardin de la maison i, il suffit de sommer son effet direct (augmentation du prix de la maison i) et son effet indirect (augmentation du prix des maisons j) pour obtenir son effet total.

Le code suivant permet de calculer ces effets directs et indirects.

```
## Effets directs, indirects et totaux (uniquement les coefficients)
impacts(Modele.SLX)
```

```
Impact measures (SlX, glht):
               Direct    Indirect      Total
Pct0_14    -0.20403803 -0.77590830 -0.9799463
Pct_65     -0.03770918 -0.06453809 -0.1022473
Pct_Img     0.10406359  0.64653923  0.7506028
Pct_brevet -0.07363272 -0.30128171 -0.3749144
NivVieMed  -0.18440960 -0.01804718 -0.2024568
```

```
## Effets directs, indirects et totaux (coefficients, valeurs de z et de p)
summary(impacts(Modele.SLX))
```

```
Impact measures (SlX, glht, n-k):
               Direct    Indirect      Total
Pct0_14    -0.20403803 -0.77590830 -0.9799463
Pct_65     -0.03770918 -0.06453809 -0.1022473
Pct_Img     0.10406359  0.64653923  0.7506028
Pct_brevet -0.07363272 -0.30128171 -0.3749144
NivVieMed  -0.18440960 -0.01804718 -0.2024568
========================================================
Standard errors:
              Direct   Indirect      Total
Pct0_14    0.06268202 0.10295210 0.10045332
Pct_65     0.05361420 0.09114695 0.08556272
Pct_Img    0.04849085 0.08593145 0.08060028
Pct_brevet 0.03549819 0.06157121 0.05975821
NivVieMed  0.11063207 0.17499339 0.14911021
========================================================
Z-values:
```

Dépendance spatiale du modèle SLX?

Ce modèle a-t-il corrigé le problème de dépendance spatiale du modèle de régression linéaire classique? Avec une valeur du I de Moran de 0,605 (p < 0,001), les résidus sont toujours fortement autocorrélés spatialement (figure 7.3).

```
lm.morantest(Modele.SLX, W.Rook, alternative="two.sided")
```

```
    Global Moran I for regression residuals

data:
model: lm(formula = formula(paste("y ~ ", paste(colnames(x)[-1],
collapse = "+"))), data = as.data.frame(x), weights = weights)
weights: W.Rook

Moran I statistic standard deviate = 21.951, p-value < 2.2e-16
alternative hypothesis: two.sided
sample estimates:
Observed Moran I      Expectation         Variance
    0.6046602748    -0.0072844321     0.0007771643
```

```
LyonIris$SLX.Residus <- residuals(Modele.SLX)
tm_shape(LyonIris)+
  tm_fill(col="SLX.Residus", n = 5, style = "quantile",
          legend.format = list(text.separator = "à"),
          palette = "-RdBu", title = "Résidus") +
  tm_layout(frame=FALSE) +
  tm_scale_bar(breaks = c(0,5))
```

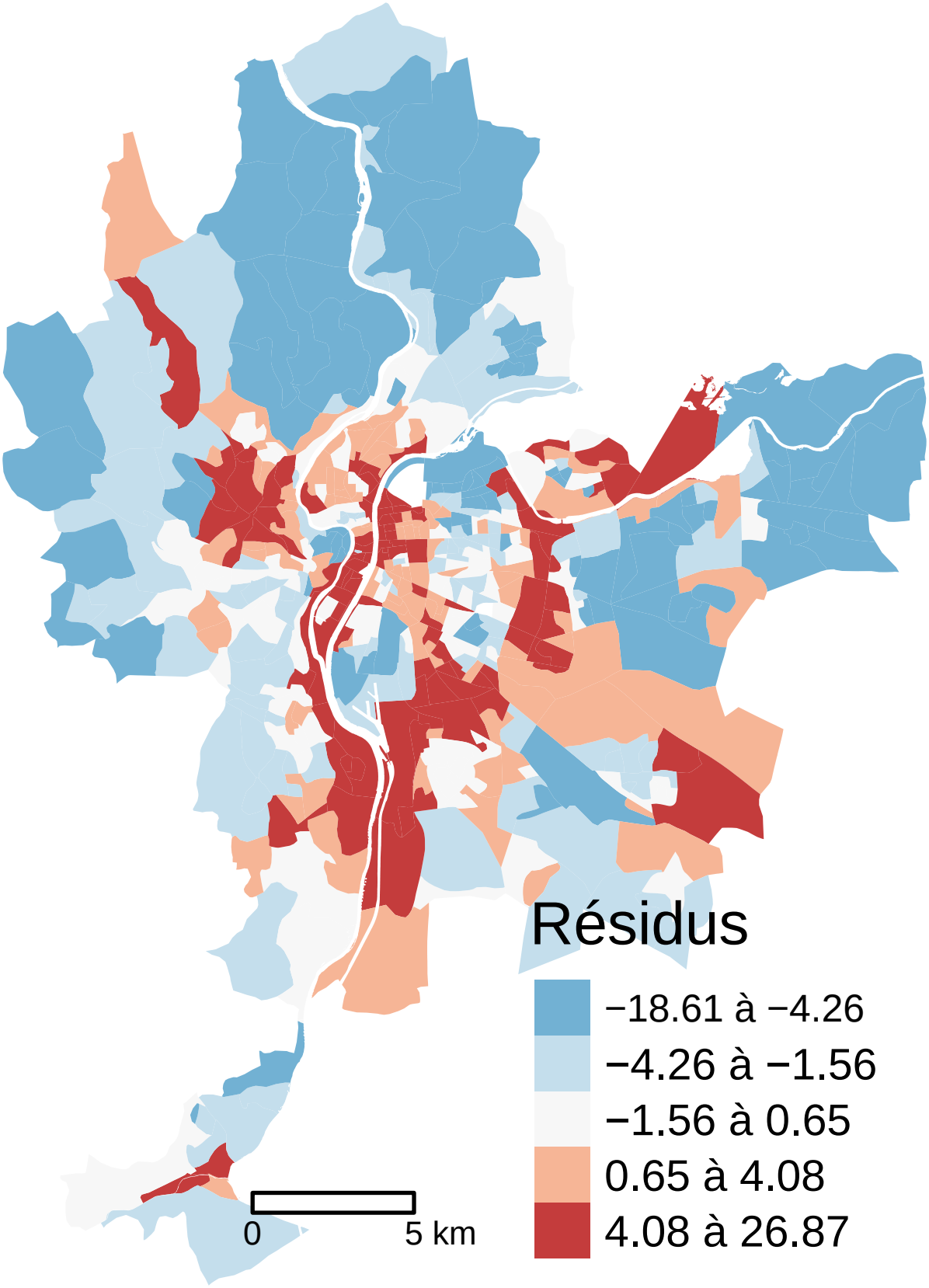


Figure 7.3 – Cartographie des résidus du modèle SLX

7.1.2.2 Modèle SAR : autocorrélation spatiale sur la variable dépendante

Dans le modèle SAR (aussi appelé SAR-LAG), l'autocorrélation spatiale est intégrée au niveau de la variable dépendante (Wy), qui est ainsi spatialement décalée. L'idée générale est que la valeur de la variable dépendante pour une observation (y_i) peut être influencée par les valeurs de y des observations voisines et proches. L'exemple le plus classique est le prix de vente des maisons : il est influencé à la fois par les caractéristiques intrinsèques de la maison (X, par exemple, la superficie habitable, le nombre de chambres à coucher, de salles de bains, etc.) et par le prix de vente des maisons voisines (Wy). Jean Dubé et Diègo Legros (2014) qualifient ce phénomène « **d'effets d'entraînement ou d'effets de débordement** (*spillover effects*) » (2014, 123). L'équation du modèle SAR s'écrit alors :

$$y = Wy\rho + X\beta + \epsilon \tag{7.4}$$

avec :

- y, la variable dépendante.
- W, la matrice de pondération spatiale.
- Wy, la variable dépendante spatialement décalée.
- ρ (prononcez *rho*), le coefficient de la variable dépendante spatialement décalée. Il varie de -1 à 1.
- X, les variables indépendantes.
- β, les coefficients des variables indépendantes.
- ϵ, les résidus.

Construction du modèle SAR dans R

Le modèle SAR est construit avec la fonction `lagsarlm` du *package* `spatialreg`.

```
## Construction du modèle
Modele.SAR <- lagsarlm(NO2 ~ Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed,
                           listw=W.Rook,    # matrice de pondération spatiale
                           data = LyonIris, # dataframe
                           type = 'lag')    # Modèle lag par défaut
## Résultats du modèle
summary(Modele.SAR, Nagelkerke=TRUE)
```

```
Call:lagsarlm(formula = NO2 ~ Pct0_14 + Pct_65 + Pct_Img + Pct_brevet +
    NivVieMed, data = LyonIris, listw = W.Rook, type = "lag")

Residuals:
      Min        1Q    Median        3Q       Max
-12.86859  -1.88111  -0.49760   0.94464  18.21351

Type: lag
Coefficients: (asymptotic standard errors)
             Estimate Std. Error z value  Pr(>|z|)
(Intercept)  7.838906   1.646232  4.7617 1.919e-06
Pct0_14     -0.098708   0.030554 -3.2306  0.001235
```

```
Pct_65      -0.034543   0.026957 -1.2814  0.200044
Pct_Img      0.030241   0.024491  1.2348  0.216917
Pct_brevet  -0.019234   0.017855 -1.0772  0.281384
NivVieMed   -0.098413   0.048985 -2.0090  0.044534

Rho: 0.87939, LR test value: 620.31, p-value: < 2.22e-16
Asymptotic standard error: 0.01942
    z-value: 45.283, p-value: < 2.22e-16
Wald statistic: 2050.5, p-value: < 2.22e-16

Log likelihood: -1366.157 for lag model
ML residual variance (sigma squared): 10.181, (sigma: 3.1908)
Nagelkerke pseudo-R-squared: 0.78962
Number of observations: 506
Number of parameters estimated: 8
AIC: 2748.3, (AIC for lm: 3366.6)
LM test for residual autocorrelation
test value: 0.6198, p-value: 0.43112
```

Dans les résultats ci-dessus, la valeur de *rho* est de 0,88 (LR = 620, $p < 0{,}001$), traduisant un très fort effet d'entraînement. Autrement dit, lorsqu'en moyenne la concentration de dioxyde d'azote augmente dans les IRIS voisines (Wy), elle augmente aussi fortement chaque IRIS (y).

⚠ **Attention**

Effets directs, indirects et totaux
Tout comme le modèle SLX vu précédemment, la formulation du modèle SAR-LAG implique des effets particuliers. Reprenons l'exemple d'un modèle prédisant le prix de vente des maisons avec cette fois-ci un modèle de type SAR-LAG :

1. L'augmentation de la superficie du jardin de la maison i va faire augmenter le prix de la maison i (y_i).
2. Cette augmentation de prix de la maison i aura un impact sur les voisins de i, soit les maisons j, car leur prix dépend du prix de la maison i au travers du terme $Wy\rho$ du modèle. Par exemple, si ρ vaut 0,8, alors 80% de l'augmentation du prix de i va se répercuter sur le prix des maisons j.
3. De même, les voisines des maisons j, les maisons k vont aussi être impactées par le changement de prix des maisons j et ainsi de suite de voisins en voisins.
4. Au final, la maison i verra son prix augmenter encore plus, car le prix de ses voisines aura augmenté par effet de rétroaction.

Notez que puisque $-1 < \rho < 1$, l'effet de déversement décroit à mesure qu'il se transmet de voisins en voisins jusqu'à disparaitre, un peu à l'image d'une onde se propageant sur une surface d'eau.
Ce processus de propagation est appelé l'effet d'entraînement ou de débordement (*spillover*) en économétrie.
L'effet original de l'augmentation de la taille du jardin sur la maison i, combiné à l'augmentation par rétroaction, est appelé l'effet direct. L'effet cumulé de l'augmentation de la taille du jardin sur toutes les autres maisons ($\neq i$) est appelé l'effet indirect. La somme des effets indirects et des directs est appelée effets totaux.
À nouveau, il est possible d'utiliser la fonction `impacts` pour calculer ces effets directs et indirects.

```
## Effets directs, indirects et totaux (uniquement les coefficients)
impacts(Modele.SAR, listw = W.Rook, R = 999)
```

```
Impact measures (lag, exact):
               Direct   Indirect      Total
Pct0_14    -0.13878038 -0.6796248 -0.8184052
Pct_65     -0.04856624 -0.2378349 -0.2864012
Pct_Img     0.04251743  0.2082131  0.2507306
Pct_brevet -0.02704205 -0.1324283 -0.1594703
NivVieMed  -0.13836534 -0.6775923 -0.8159576
```

```
## Effets directs, indirects et totaux (coefficients, valeurs de z et de p)
summary(impacts(Modele.SAR, listw = W.Rook, R = 999), zstats = TRUE, short = TRUE)
```

```
Impact measures (lag, exact):
               Direct   Indirect      Total
Pct0_14    -0.13878038 -0.6796248 -0.8184052
Pct_65     -0.04856624 -0.2378349 -0.2864012
Pct_Img     0.04251743  0.2082131  0.2507306
Pct_brevet -0.02704205 -0.1324283 -0.1594703
NivVieMed  -0.13836534 -0.6775923 -0.8159576
========================================================
Simulation results ( variance matrix):
========================================================
Simulated standard errors
              Direct  Indirect     Total
Pct0_14    0.04311854 0.2417281 0.2801439
Pct_65     0.03799207 0.1942553 0.2311891
Pct_Img    0.03429199 0.1761186 0.2095703
Pct_brevet 0.02447359 0.1248117 0.1487223
NivVieMed  0.06692148 0.3563214 0.4195402
```

Dépendance spatiale du modèle SAR?

```
## Autocorrélation spatiale des résidus
test <- moran.mc(resid(Modele.SAR), W.Rook, nsim=999)
print(test)
```

```
    Monte-Carlo simulation of Moran I

data:  resid(Modele.SAR)
weights: W.Rook
number of simulations + 1: 1000

statistic = -0.014281, observed rank = 340, p-value = 0.66
alternative hypothesis: greater
```

```
## Cartographie des résidus
LyonIris$SAR.Residus <- resid(Modele.SAR)
tm_shape(LyonIris)+
  tm_fill(col="SAR.Residus", n = 5, style = "quantile",
          legend.format = list(text.separator = "à"),
          palette = "-RdBu", title = "Résidus") +
  tm_layout(frame=FALSE) +
  tm_scale_bar(breaks = c(0,5))
```

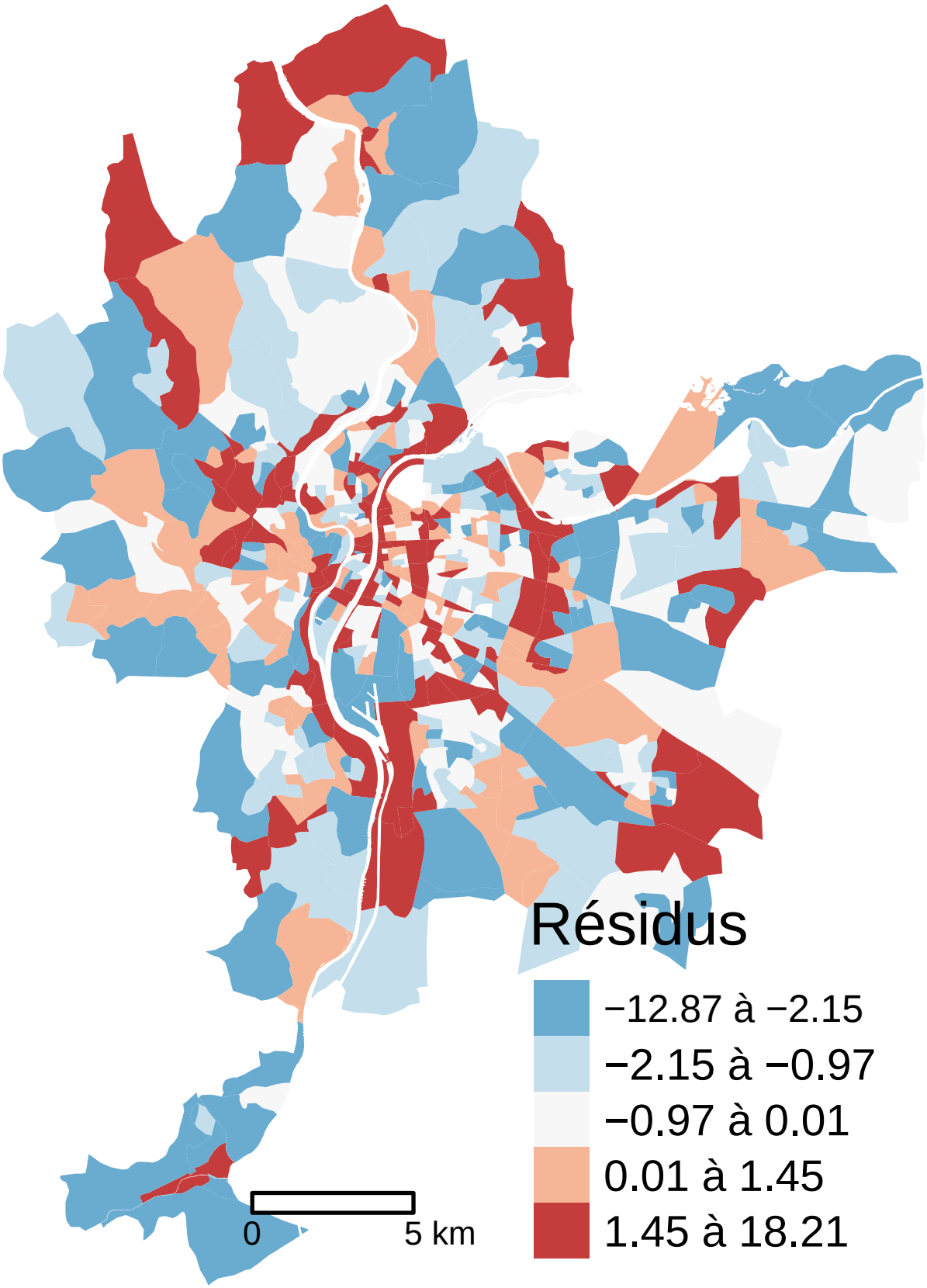


Figure 7.4 – Cartographie des résidus du modèle SAR

Ce modèle a-t-il corrigé le problème de dépendance spatiale du modèle de régression linéaire classique? Avec une valeur du I de Moran de -0.014 (p = 0.66), les résidus ne sont plus spatialement autocorrélés (figure 7.4).

7.1.2.3 Modèle SEM : autocorrélation spatiale sur le terme d'erreur

Dans le modèle SEM (*Spatial Error Model*, appelé aussi *SAR-ERROR*), l'intégration de l'autocorrélation spatiale est réalisée sur le terme d'erreur, ce qui pourrait se justifier par l'omission d'une variable dépendante spatialement structurée (Dubé et Legros 2014, 126). L'équation du modèle SEM s'écrit :

$$y = X\beta + u, u = \lambda W u + \epsilon \tag{7.5}$$

avec :

- y, la variable dépendante.
- W, la matrice de pondération spatiale.
- λ (prononcez *lambda*), le coefficient sur le terme d'erreur spatialement décalé. Il varie de -1 à 1.
- X, les variables indépendantes.
- β, les coefficients des variables indépendantes.
- ϵ, les résidus.

Construction du modèle SAR dans R

Le modèle SEM est construit avec la fonction `errorsarlm` du *package* `spatialreg`.

```
## Construction du modèle
Modele.SEM <- errorsarlm(NO2 ~ Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed,
                         listw=W.Rook,    # matrice de pondération spatiale
                         data = LyonIris) # dataframe
## Résultats du modèle
summary(Modele.SEM, Nagelkerke=TRUE)
```

```
Call:errorsarlm(formula = NO2 ~ Pct0_14 + Pct_65 + Pct_Img + Pct_brevet +
    NivVieMed, data = LyonIris, listw = W.Rook)

Residuals:
      Min        1Q    Median        3Q       Max
-12.86150  -1.83161  -0.44106   0.91029  17.94924

Type: error
Coefficients: (asymptotic standard errors)
             Estimate Std. Error z value Pr(>|z|)
(Intercept) 30.544576   2.358173 12.9526  < 2e-16
Pct0_14     -0.035019   0.033393 -1.0487  0.29431
Pct_65      -0.026039   0.028970 -0.8988  0.36874
Pct_Img     -0.016770   0.026176 -0.6407  0.52175
```

```
Pct_brevet   0.023708   0.019074  1.2430  0.21388
NivVieMed   -0.146309   0.060273 -2.4274  0.01521

Lambda: 0.91138, LR test value: 613.15, p-value: < 2.22e-16
Asymptotic standard error: 0.01651
    z-value: 55.201, p-value: < 2.22e-16
Wald statistic: 3047.2, p-value: < 2.22e-16

Log likelihood: -1369.737 for error model
ML residual variance (sigma squared): 9.9971, (sigma: 3.1618)
Nagelkerke pseudo-R-squared: 0.78662
Number of observations: 506
Number of parameters estimated: 8
AIC: 2755.5, (AIC for lm: 3366.6)
```

Dans les résultats ci-dessus, la valeur de *lambda* est de 0,91 (LR = 613, p < 0,001), traduisant une très forte autocorrélation spatiale sur le terme d'erreur.

Dépendance spatiale du modèle SEM?

```
## Autocorrélation spatiale des résidus
test <- moran.mc(resid(Modele.SEM), W.Rook, nsim=999)
print(test)
```

```
    Monte-Carlo simulation of Moran I

data:  resid(Modele.SEM)
weights: W.Rook
number of simulations + 1: 1000

statistic = -0.011827, observed rank = 379, p-value = 0.621
alternative hypothesis: greater
```

Ce modèle a-t-il corrigé le problème de dépendance spatiale du modèle de régression linéaire classique? Avec une valeur du I de Moran de -0.012 (p = 0.621), les résidus ne sont plus spatialement autocorrélés.

7.1.2.4 Modèle SDM : autocorrélation spatiale sur la variable dépendante et les variables indépendantes

Le modèle SDM (*Spatial Durbin Model*) est un modèle mixte qui intègre à la fois l'autocorrélation spatiale sur la variable dépendante (Wy, **effets d'entraînement ou de débordement**) et sur les variables indépendantes (WX, **externalités**). Il s'écrit alors :

$$y = Wy\rho + X\beta + WX\theta + \epsilon \tag{7.6}$$

avec :

- y, la variable dépendante.
- W, la matrice de pondération spatiale.
- Wy, la variable dépendante spatialement décalée.
- ρ, le coefficient de la variable dépendante spatialement décalée.
- X, les variables indépendantes.
- β, les coefficients des variables indépendantes.
- WX, les variables indépendantes spatiales décalées.
- θ, les coefficients des variables indépendantes spatiales décalées.
- ϵ, les résidus.

Construction du modèle SDM dans R

Le modèle SDM est construit avec la fonction `lagsarlm` du *package* `spatialreg`. Notez que le paramètre `type = "mixed"` spécifie l'utilisation d'un modèle mixte.

```
## Construction du modèle
Modele.DurbinSpatial <- lagsarlm(NO2 ~ Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed,
                        listw = W.Rook,    # matrice de pondération spatiale
                        data = LyonIris,   # dataframe
                        type = "mixed")
## Résultats du modèles
summary(Modele.DurbinSpatial, Nagelkerke=TRUE)
```

```
Call:lagsarlm(formula = NO2 ~ Pct0_14 + Pct_65 + Pct_Img + Pct_brevet +
    NivVieMed, data = LyonIris, listw = W.Rook, type = "mixed")

Residuals:
      Min        1Q    Median        3Q       Max
-12.60922  -1.77753  -0.43909   0.99252  18.15526

Type: mixed
Coefficients: (asymptotic standard errors)
                Estimate Std. Error z value Pr(>|z|)
(Intercept)    8.1130457  2.5671301  3.1604 0.001576
Pct0_14       -0.0574046  0.0344908 -1.6643 0.096043
Pct_65        -0.0238715  0.0293647 -0.8129 0.416256
Pct_Img        0.0048364  0.0266560  0.1814 0.856025
Pct_brevet     0.0112746  0.0195259  0.5774 0.563656
NivVieMed     -0.1463876  0.0605853 -2.4162 0.015682
lag.Pct0_14   -0.1242574  0.0581170 -2.1381 0.032512
lag.Pct_65     0.0255480  0.0499646  0.5113 0.609125
lag.Pct_Img    0.1559952  0.0482138  3.2355 0.001214
lag.Pct_brevet -0.0883930  0.0342496 -2.5809 0.009856
lag.NivVieMed  0.1032469  0.0960201  1.0753 0.282257

Rho: 0.84127, LR test value: 492.38, p-value: < 2.22e-16
```

```
Asymptotic standard error: 0.023363
    z-value: 36.009, p-value: < 2.22e-16
Wald statistic: 1296.7, p-value: < 2.22e-16

Log likelihood: -1353.106 for mixed model
ML residual variance (sigma squared): 9.9845, (sigma: 3.1598)
Nagelkerke pseudo-R-squared: 0.8002
Number of observations: 506
Number of parameters estimated: 13
AIC: 2732.2, (AIC for lm: 3222.6)
LM test for residual autocorrelation
test value: 0.0748, p-value: 0.78447
```

Effets directs, indirects et totaux

```
# Effets directs, indirects et totaux (uniquement les coefficients)
impacts(Modele.DurbinSpatial, listw = W.Rook, R = 999)
```

```
Impact measures (mixed, exact):
                Direct    Indirect       Total
Pct0_14    -0.12369039 -1.02079497 -1.14448536
Pct_65     -0.02177191  0.03233406  0.01056215
Pct_Img     0.06632543  0.94692639  1.01325182
Pct_brevet -0.01903815 -0.46681402 -0.48585217
NivVieMed  -0.15403413 -0.11775603 -0.27179016
```

```
# Effets directs, indirects et totaux (coefficients, valeurs de z et de p)
summary(impacts(Modele.DurbinSpatial, listw = W.Rook, R = 999), zstats = TRUE, short = TRUE)
```

```
Impact measures (mixed, exact):
                Direct    Indirect       Total
Pct0_14    -0.12369039 -1.02079497 -1.14448536
Pct_65     -0.02177191  0.03233406  0.01056215
Pct_Img     0.06632543  0.94692639  1.01325182
Pct_brevet -0.01903815 -0.46681402 -0.48585217
NivVieMed  -0.15403413 -0.11775603 -0.27179016
========================================================
Simulation results ( variance matrix):
========================================================
Simulated standard errors
               Direct  Indirect     Total
Pct0_14    0.04372816 0.3232734 0.3525711
Pct_65     0.03497663 0.2840611 0.3059622
Pct_Img    0.03316650 0.2739069 0.2943299
Pct_brevet 0.02602762 0.1992782 0.2165489
```

```
NivVieMed  0.06996510 0.4978543 0.5365154

Simulated z-values:
               Direct   Indirect       Total
Pct0_14    -2.8331109 -3.1721374 -3.25992233
Pct_65     -0.6252506  0.1387484  0.05733996
Pct_Img     2.0552290  3.5303304  3.51696123
Pct_brevet -0.7664894 -2.3574956 -2.26160078
NivVieMed  -2.1721692 -0.1951222 -0.46432678

Simulated p-values:
           Direct    Indirect   Total
Pct0_14    0.0046097 0.00151321 0.00111443
Pct_65     0.5318066 0.88964900 0.95427439
Pct_Img    0.0398569 0.00041504 0.00043652
Pct_brevet 0.4433851 0.01839867 0.02372208
NivVieMed  0.0298429 0.84529725 0.64241364
```

Notez que les effets ci-dessus sont différents de ceux que nous avions obtenus pour le simple modèle SAR-LAG car les effets des variables X laguées entrainent également des effets de déversement qui viennent se combiner aux effets de déversement du terme Wy.

Dépendance spatiale du modèle SDM?

```
test <- moran.mc(resid(Modele.DurbinSpatial), W.Rook, nsim=999)
print(test)
```

```
    Monte-Carlo simulation of Moran I

data:  resid(Modele.DurbinSpatial)
weights: W.Rook
number of simulations + 1: 1000

statistic = -0.0046127, observed rank = 476, p-value = 0.524
alternative hypothesis: greater
```

Ce modèle a-t-il corrigé le problème de dépendance spatiale du modèle de régression linéaire classique? Avec une valeur du I de Moran de -0.005 (p = 0.524), les résidus ne sont plus spatialement autocorrélés.

7.1.2.5 Modèle SDEM : autocorrélation spatiale sur les variables indépendantes et sur le terme d'erreur

Le modèle SDEM (*Spatial Durbin Error Model* en anglais) est un autre modèle mixte qui intègre à la fois l'autocorrélation spatiale sur les valeurs indépendantes (WX, **externalités**) et sur le terme d'erreur ($u = \lambda W u + \epsilon$). Il s'écrit alors :

$$y = X\beta + WX\theta + u, u = \lambda Wu + \epsilon \tag{7.7}$$

avec :

- y, la variable dépendante.
- W, la matrice de pondération spatiale.
- X, les variables indépendantes.
- β, les coefficients des variables indépendantes.
- WX, les variables dépendantes spatiales décalées.
- θ, les coefficients des variables indépendantes spatiales décalées.
- λ (prononcez *lambda*), le coefficient sur le terme d'erreur spatialement décalé.
- ϵ, les résidus.

Construction du modèle SDEM dans R

Le modèle SDEM est construit avec la fonction `errorsarlm` du *package* `spatialreg`. Notez que le paramètre `etype = "mixed"` spécifie l'utilisation d'un modèle mixte.

```
## Construction du modèle
Modele.DurbinErreur <- errorsarlm(NO2 ~ Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed,
                                  listw=W.Rook,    # matrice de pondération spatiale
                                  data = LyonIris, # dataframe
                                  etype = 'emixed')
## Résultats du modèle
summary(Modele.DurbinErreur, Nagelkerke=TRUE)
```

```
Call:errorsarlm(formula = NO2 ~ Pct0_14 + Pct_65 + Pct_Img + Pct_brevet +
    NivVieMed, data = LyonIris, listw = W.Rook, etype = "emixed")

Residuals:
      Min        1Q    Median        3Q       Max
-12.99324  -1.82407  -0.45644   1.06084  18.21108

Type: error
Coefficients: (asymptotic standard errors)
               Estimate Std. Error z value  Pr(>|z|)
(Intercept)    37.061010   6.501018  5.7008 1.192e-08
Pct0_14        -0.081998   0.041699 -1.9664   0.04925
Pct_65         -0.026329   0.034714 -0.7585   0.44817
Pct_Img         0.004656   0.031028  0.1501   0.88072
Pct_brevet      0.009785   0.023884  0.4097   0.68203
NivVieMed      -0.167855   0.068005 -2.4683   0.01358
lag.Pct0_14    -0.176747   0.102345 -1.7270   0.08417
lag.Pct_65      0.010533   0.089183  0.1181   0.90599
lag.Pct_Img     0.092785   0.079704  1.1641   0.24437
lag.Pct_brevet -0.038048   0.056688 -0.6712   0.50211
```

```
lag.NivVieMed  -0.102531   0.172405 -0.5947   0.55204

Lambda: 0.8976, LR test value: 464.09, p-value: < 2.22e-16
Asymptotic standard error: 0.018242
    z-value: 49.204, p-value: < 2.22e-16
Wald statistic: 2421, p-value: < 2.22e-16

Log likelihood: -1367.25 for error model
ML residual variance (sigma squared): 10.046, (sigma: 3.1696)
Nagelkerke pseudo-R-squared: 0.78871
Number of observations: 506
Number of parameters estimated: 13
AIC: 2760.5, (AIC for lm: 3222.6)
```

Effets directs, indirects et totaux

```
## Effets directs, indirects et totaux (uniquement les coefficients)
impacts(Modele.DurbinErreur, listw = W.Rook, R = 999)
```

```
Impact measures (SDEM, glht):
                Direct    Indirect       Total
Pct0_14    -0.081997642 -0.17674683 -0.25874447
Pct_65     -0.026329370  0.01053248 -0.01579689
Pct_Img     0.004656039  0.09278511  0.09744115
Pct_brevet  0.009784961 -0.03804813 -0.02826317
NivVieMed  -0.167855498 -0.10253070 -0.27038620
```

```
## Effets directs, indirects et totaux (coefficients, valeurs de z et de p)
summary(impacts(Modele.DurbinErreur, listw = W.Rook, R = 999), zstats = TRUE, short = TRUE)
```

```
Impact measures (SDEM, glht, n):
                Direct    Indirect       Total
Pct0_14    -0.081997642 -0.17674683 -0.25874447
Pct_65     -0.026329370  0.01053248 -0.01579689
Pct_Img     0.004656039  0.09278511  0.09744115
Pct_brevet  0.009784961 -0.03804813 -0.02826317
NivVieMed  -0.167855498 -0.10253070 -0.27038620
========================================================
Standard errors:
               Direct   Indirect      Total
Pct0_14    0.04169878 0.10234506 0.13146453
Pct_65     0.03471367 0.08918350 0.11192948
Pct_Img    0.03102807 0.07970364 0.09949722
Pct_brevet 0.02388387 0.05668833 0.07344175
NivVieMed  0.06800483 0.17240549 0.21172909
```

```
============================================================
Z-values:
               Direct   Indirect      Total
Pct0_14    -1.9664279 -1.7269698 -1.9681695
Pct_65     -0.7584727  0.1180989 -0.1411325
Pct_Img     0.1500589  1.1641264  0.9793354
Pct_brevet  0.4096890 -0.6711810 -0.3848379
NivVieMed  -2.4682880 -0.5947067 -1.2770385

p-values:
           Direct   Indirect Total
Pct0_14    0.049249 0.084173 0.049049
Pct_65     0.448168 0.905989 0.887765
Pct_Img    0.880718 0.244373 0.327414
Pct_brevet 0.682034 0.502105 0.700358
NivVieMed  0.013576 0.552040 0.201589
```

Dépendance spatiale du modèle SDEM?

```
test <- moran.mc(resid(Modele.DurbinErreur), W.Rook, nsim=999)
print(test)
```

```
    Monte-Carlo simulation of Moran I

data:  resid(Modele.DurbinErreur)
weights: W.Rook
number of simulations + 1: 1000

statistic = -0.010362, observed rank = 376, p-value = 0.624
alternative hypothesis: greater
```

Avec une valeur du I de Moran de -0.01 (p = 0.624), les résidus du modèle SDEM ne sont pas spatialement autocorrélés.

7.1.3 Quel modèle choisir?

7.1.3.1 Tests du multiplicateur de Lagrange sur le modèle MCO

L'utilisation des tests du multiplicateur de Lagrange (simple et robuste) a été largement popularisée par Anselin *et al.* (1996) pour vérifier si le recours à un modèle autorégressif est nécessaire, comparativement à un modèle de régression classique (MCO). Les tests sont calculés sur le modèle MCO avec la fonction `lm.LMtests` et une matrice de pondération spatiale. Ces tests permettent aussi de choisir entre les modèles SAR et SEM. La démarche suivante peut être utilisée pour choisir un modèle :

1. Si toutes les valeurs des tests (simples et robustes) sont non significatives (p > 0,05), alors le recours à un modèle autorégressif n'est pas nécessaire. Nous pouvons conserver le modèle de régression classique (MCO).
2. Si les valeurs de `LMlag` ou `RLMlag` sont non significatives (p > 0,05), alors le recours au modèle SAR n'est pas nécessaire.
3. Si les valeurs de `LMerr` ou `RLMerr` sont non significatives (p > 0,05), alors le recours au modèle SEM n'est pas nécessaire.
4. Si les valeurs de `RLMlag` et `RLMerr` sont significatives (p < 0,001), nous choisissons le modèle ayant la plus forte statistique.

```
lag_test <- lm.LMtests(model = Modele.MCO,
                   listw = W.Rook,
                   test = c("LMlag","LMerr","RLMlag","RLMerr"))
summary(lag_test)
```

```
    Rao's score (a.k.a Lagrange multiplier) diagnostics for spatial
    dependence
data:
model: lm(formula = NO2 ~ Pct0_14 + Pct_65 + Pct_Img + Pct_brevet +
NivVieMed, data = LyonIris)
test weights: listw

         statistic parameter p.value
RSlag    554.65778         1  <2e-16 ***
RSerr    432.83282         1  <2e-16 ***
adjRSlag 122.56452         1  <2e-16 ***
adjRSerr   0.73955         1  0.3898
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Dans les résultats ci-dessous, nous ne retenons pas le modèle SEM car la valeur de 0.74 pour le `adjRSerr` n'est pas significative (p = 0.39). En revanche, les valeurs de `RSlag` et de `adjRSlag` (555 et `RLMlag` (123) sont significatives, ce qui justifie la sélection du modèle SAR.

7.1.3.2 Comparaison des modèles mixtes et non mixtes

Nous avons vu qu'il existe deux modèles mixtes (SDM et SDEM). Il convient alors de vérifier si le recours d'un modèle mixte est justifié comparativement à un modèle non mixte. Dans le code ci-dessous, nous vérifions si le modèle SDM est statistiquement différent du modèle SAR avec les fonctions `LR.Sarlm` et `anova`. Les résultats signalent un écart significatif des valeurs du log-vraisemblance (26,101, p < 0,001). Par conséquent, ce modèle mixte a un apport significatif.

```
## SDM et SEM sont-ils significativement différents?
LR.Sarlm(Modele.DurbinSpatial, Modele.SAR)
```

```
    Likelihood ratio for spatial linear models

data:
Likelihood ratio = 26.101, df = 5, p-value = 8.528e-05
sample estimates:
Log likelihood of Modele.DurbinSpatial           Log likelihood of Modele.SAR
                             -1353.106                               -1366.157
```

```
anova(Modele.DurbinSpatial, Modele.SAR)
```

```
                     Model df    AIC  logLik Test L.Ratio    p-value
Modele.DurbinSpatial     1 13 2732.2 -1353.1    1
Modele.SAR               2  8 2748.3 -1366.2    2  26.101 8.5283e-05
```

À l'inverse, la différence entre les valeurs du log-vraisemblance des modèles SDEM et SEM n'est pas significative (4,9728, p = 0,42), signalant que l'utilisation d'un modèle SDEM comparativement à un modèle SEM n'est pas nécessaire.

```
## SDEM et SEM sont-ils significativement différents?
LR.Sarlm(Modele.DurbinErreur, Modele.SEM)
```

```
    Likelihood ratio for spatial linear models

data:
Likelihood ratio = 4.9728, df = 5, p-value = 0.4192
sample estimates:
Log likelihood of Modele.DurbinErreur          Log likelihood of Modele.SEM
                             -1367.250                               -1369.737
```

```
anova(Modele.DurbinErreur, Modele.SEM)
```

```
                    Model df    AIC  logLik Test L.Ratio p-value
Modele.DurbinErreur     1 13 2760.5 -1367.2    1
Modele.SEM              2  8 2755.5 -1369.7    2  4.9728  0.4192
```

7.1.3.3 Mesures AIC et BIC et dépendance spatiale

Le critère d'information d'Akaike (AIC) et le critère d'information bayésien (BIC) sont largement utilisés pour évaluer la qualité d'ajustement du modèle. Plus leurs valeurs sont faibles, meilleur est le modèle. Il est donc possible de comparer leurs valeurs pour les différents modèles (MCO, SLX, SAR, SEM, SDM et SDEM). Nous pouvons aussi comparer l'autocorrélation spatiale des résidus des modèles avec le I de Moran.

```
## Valeurs d'AIC et de BIC
AICs <- AIC(Modele.MCO, Modele.SLX, Modele.SAR, Modele.SEM,
            Modele.DurbinSpatial, Modele.DurbinErreur)
BICs <- BIC(Modele.MCO, Modele.SLX, Modele.SAR, Modele.SEM,
            Modele.DurbinSpatial, Modele.DurbinErreur)
## Autocorrélation spatiale des résidus
IMoran.MCO <- moran.mc(resid(Modele.MCO), W.Rook, nsim=999)
IMoran.SLX <- moran.mc(resid(Modele.SLX), W.Rook, nsim=999)
IMoran.SLM <- moran.mc(resid(Modele.SAR), W.Rook, nsim=999)
IMoran.SEM <- moran.mc(resid(Modele.SEM), W.Rook, nsim=999)
IMoran.DurbinS <- moran.mc(resid(Modele.DurbinSpatial), W.Rook, nsim=999)
IMoran.DurbinE <- moran.mc(resid(Modele.DurbinErreur), W.Rook, nsim=999)
MoranI.s <- c(IMoran.MCO$statistic, IMoran.SLX$statistic,
              IMoran.SLM$statistic, IMoran.SEM$statistic,
              IMoran.DurbinS$statistic, IMoran.DurbinE$statistic)
MoranI.p <- c(IMoran.MCO$p.value, IMoran.SLX$p.value,
              IMoran.SLM$p.value, IMoran.SEM$p.value,
              IMoran.DurbinS$p.value, IMoran.DurbinE$p.value)
## Tableau
Comparaison <- data.frame(Modele = c("MCO", "SLX", "SAR", "SEM", "Durbin S", "Durbin E"),
                          AIC = AICs$AIC,
                          BIC = BICs$BIC,
                          dl = AICs$df,
                          MoranI = MoranI.s,
                          MoranIp = MoranI.p)
Comparaison
```

```
    Modele      AIC      BIC dl       MoranI MoranIp
1      MCO 3366.626 3396.212  7  0.587312061   0.001
2      SLX 3222.594 3273.313 12  0.604660275   0.001
3      SAR 2748.314 2782.126  8 -0.014281059   0.660
4      SEM 2755.474 2789.286  8 -0.011826605   0.599
5 Durbin S 2732.212 2787.157 13 -0.004612686   0.549
6 Durbin E 2760.501 2815.446 13 -0.010361653   0.604
```

Quelques lignes de code suffisent pour créer deux graphiques permettant de comparer visuellement les résultats des différents modèles (figure 7.5). Les résultats démontrent que :

- Les modèles MCO et SLX ont un problème de dépendance spatiale puisque leurs résidus sont significativement autocorrélés spatialement. Par conséquent, ils ne devraient pas être retenus.
- Les modèles SDM, SAR et SEM sont les plus performants avec les valeurs d'AIC les plus faibles.

```
library(ggplot2)
library(ggpubr)
## Graphique pour l'autocorrélation spatiale
g1 <- ggplot(data=Comparaison, aes(x=reorder(Modele,MoranI), y=MoranI)) +
```

```
      geom_segment(aes(x=reorder(Modele, MoranI),
                       xend=reorder(Modele, MoranI),
                       y=0, yend=MoranI)) +
  geom_point( size=4,fill="red",shape=21)+
  xlab("Modèle") + ylab("I de Moran")+
  labs(title="Autocorrélation spatiale des résidus",
       caption="Plus la valeur du I de Moran est faible, \nmoins il y a d'autocorrélation spatiale.")
## Graphique pour les valeurs d'AIC
g2 <- ggplot(data=Comparaison, aes(x=reorder(Modele,AIC), y=AIC)) +
  geom_segment(aes(x=reorder(Modele, AIC),
                   xend=reorder(Modele, AIC),
                   y=0, yend=AIC)) +
  geom_point( size=4,fill="red",shape=21)+
  xlab("Modèle") + ylab("AIC")+
  labs(title="Qualité d'ajustement du modèle",
       caption="Plus la valeur d'AIC est faible, \nplus le modèle est performant.")
## Figure avec les deux graphiques
ggarrange(g1, g2)
```

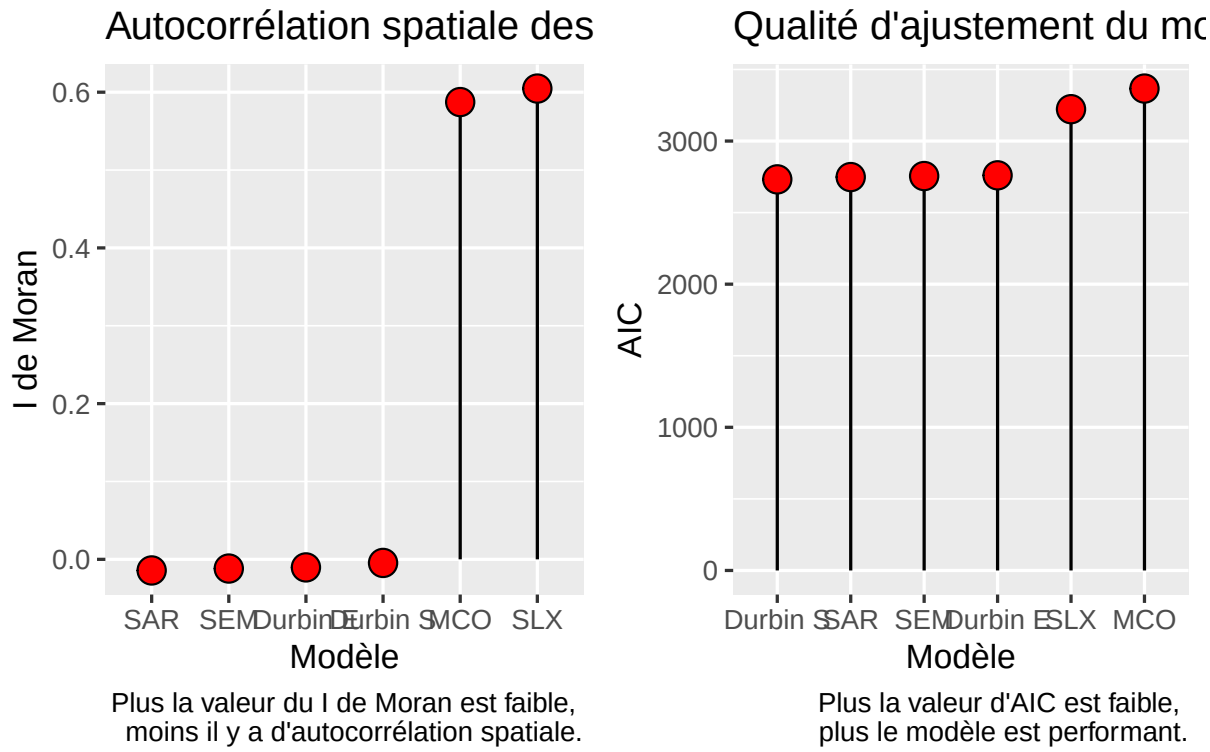


Figure 7.5 – Comparaison des différents modèles

7.2 Modèles généralisés additifs (GAM) avec une spline bivariée sur les coordonnées géographiques

Les modèles généralisés additifs (*Generalized additive models* en anglais) permettent d'intégrer à la fois des effets linéaires et des effets non linéaires avec des *splines*. Ils peuvent alors être utilisés en intégrant une *spline* bivariée sur les coordonnées géographiques des centroïdes des entités spatiales.

⚠ **Attention**

Modèles généralisés additifs
Pour une description détaillée des modèles généralisés additifs, nous vous invitons vivement à lire le chapitre suivant (Apparicio et Gelb 2022).

7.2.1 Principe de base d'un GAM intégrant l'espace

Avec une *spline* bivariée sur les coordonnées géographiques, l'équation d'un modèle généralisé additif s'écrit :

$$g(Y) = \beta_0 + X\beta + s(CoordX, CoordY) + \epsilon \tag{7.8}$$

avec :

- y, la variable dépendante.
- β_0, la constante.
- X, les variables indépendantes.
- β, les coefficients des variables dépendantes.
- $s(CoordX, CoordY)$, *spline* bivariée sur les coordonnées x et y.
- ϵ, les résidus.

L'intérêt de recourir à une *spline* bivariée sur les coordonnées géographiques est double :

1. Contrôler l'effet de la localisation sur la variable dépendante (y). Les coefficients des autres variables indépendantes sont ainsi obtenus une fois l'espace pris en compte.
2. Évaluer l'effet de la localisation (patron spatial), une fois les autres variables indépendantes contrôlées. Autrement dit, toutes choses étant égales par ailleurs, quel est l'effet de la localisation sur la variable dépendante?

7.2.2 Construction d'un modèle GAM dans R

7.2.2.1 Réalisation du modèle GAM

Pour construire des modèles GAM dans R, nous utilisons la fonction `gam` du *package* `mgcv` (Wood 2011).

```r
library(mgcv)
## Ajout des coordonnées X et Y dans LyonIris
xy <- st_coordinates(st_centroid(LyonIris))
LyonIris$X <- xy[,1]
LyonIris$Y <- xy[,2]
## Construction du modèle GAM
Modele.GAM1 <- gam(NO2 ~  Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed+
                          s(X, Y),    # spline sur les coordonnées X, Y
                   data = LyonIris)  # dataframe
## Résultats du modèle
summary(Modele.GAM1)
```

```
Family: gaussian
Link function: identity

Formula:
NO2 ~ Pct0_14 + Pct_65 + Pct_Img + Pct_brevet + NivVieMed + s(X,
    Y)

Parametric coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 30.402313   2.156322  14.099   <2e-16 ***
Pct0_14     -0.053641   0.046224  -1.160    0.246
Pct_65      -0.056310   0.038476  -1.463    0.144
Pct_Img      0.008033   0.035397   0.227    0.821
Pct_brevet   0.043874   0.027518   1.594    0.112
NivVieMed   -0.043282   0.072461  -0.597    0.551
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
        edf Ref.df     F p-value
s(X,Y) 26.5  28.62 27.22  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.715   Deviance explained = 73.3%
GCV = 18.814  Scale est. = 17.605    n = 506
```

Les résultats ci-dessus signalent que la localisation a un effet très significatif puisque (s(X,Y) = 26,5 avec $p < 0{,}001$). Notez que la valeur de p permet de déterminer si la *spline* bivariée (et donc l'espace) a ou non un effet significatif. Si la valeur de p est supérieure à 0,05, alors il n'est pas nécessaire de conserver la *spline* bivariée sur les coordonnées géographiques.

De plus, le code ci-dessous permet de constater que le modèle GAM est plus performant que le modèle linéaire multiple classique (MCO).

```
anova(Modele.MCO, Modele.GAM1)
```

```
Analysis of Variance Table

Model 1: NO2 ~ Pct0_14 + Pct_65 + Pct_Img + Pct_brevet + NivVieMed
Model 2: NO2 ~ Pct0_14 + Pct_65 + Pct_Img + Pct_brevet + NivVieMed + s(X,
    Y)
  Res.Df     RSS   Df Sum of Sq      F    Pr(>F)
1  500.0 22346.0
2  473.5  8336.1 26.5     14010 30.029 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Nous pouvons aussi introduire une *spline* plus complexe en augmentant le nombre de nœuds à 40.

```
Modele.GAM2 <- gam(NO2 ~  Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed+
                          s(X, Y, k= 40), data = LyonIris)
summary(Modele.GAM2)
```

```
Family: gaussian
Link function: identity

Formula:
NO2 ~ Pct0_14 + Pct_65 + Pct_Img + Pct_brevet + NivVieMed + s(X,
    Y, k = 40)

Parametric coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 30.372093   2.088160  14.545   <2e-16 ***
Pct0_14     -0.055294   0.044285  -1.249   0.2124
Pct_65      -0.049122   0.036870  -1.332   0.1834
Pct_Img      0.002226   0.033722   0.066   0.9474
Pct_brevet   0.052229   0.026315   1.985   0.0478 *
NivVieMed   -0.050991   0.070081  -0.728   0.4672
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
         edf Ref.df     F p-value
s(X,Y) 35.68  38.51 24.27  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.747   Deviance explained = 76.7%
GCV = 17.014  Scale est. = 15.613    n = 506
```

La valeur plus faible d'AIC pour le second modèle GAM signale qu'il est plus performant que le premier.

```
AIC(Modele.MCO, Modele.GAM1, Modele.GAM2)
```

```
                  df      AIC
Modele.MCO   7.00000 3366.626
Modele.GAM1 33.50046 2920.682
Modele.GAM2 42.67520 2868.355
```

7.2.2.2 Visualisation de l'effet de l'espace

Pour visualiser les prédictions du modèle dans l'espace, toutes choses étant égales par ailleurs, nous utilisons la fonction `vis.gam` (figure 7.6). Les contours signalent qu'au centre de Lyon, les valeurs de dioxyde d'azote sont les plus élevées et dépassent même 40 μg/m^3.

```
vis.gam(Modele.GAM2, view=c("X", "Y"), plot.type = "contour", color="terrain")
```

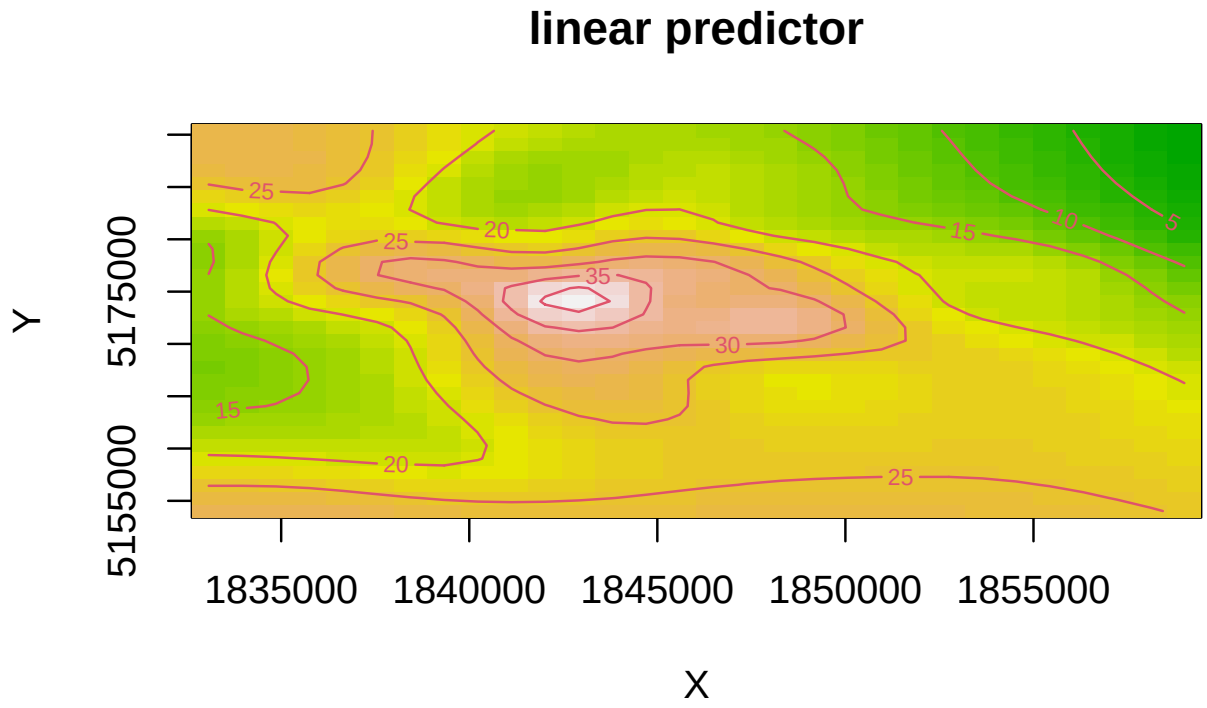


Figure 7.6 – Visualisation des prédictions dans l'espace avec la fonction `vis.gam`

Toutefois, il est plus intéressant de la représenter dans un *raster*, une fois contrôlées les autres variables indépendantes. Pour ce faire, six étapes sont nécessaires :

1. Créer une *grid*.
2. Fixer les autres paramètres à leur moyenne respective.

3. Calculer la prédiction pour la localisation.
4. Centrer la prédiction.
5. Construire le *raster* avec les prédictions.
6. Découper et cartographier le *raster*.

```
library(raster)
library(terra)
## Étape 1 : création d'une grid pour la prédiction de 100 mètres de résolution spatiale
Xcoords <- seq(min(LyonIris$X-100), max(LyonIris$X+100), by=100)
Ycoords <- seq(min(LyonIris$Y-100), max(LyonIris$Y+100), by=100)
PredDF <- expand.grid(Xcoords,Ycoords)
names(PredDF) <- c("X","Y")
## Étape 2 : fixation de tous les autres paramètres à leur moyenne
for(Var in c("VegHautPrt","Pct0_14","Pct_65","Pct_Img","Pct_brevet", "NivVieMed")){
  PredDF[[Var]] <- mean(LyonIris[[Var]])
}
## Étape 3 : calcul de la prédiction
PredDF$PM25 <- predict(Modele.GAM2,newdata=PredDF)
## Étape 4 : centrage de la prédiction (sans la constante)
PredDF$CenterPredPM25 <- PredDF$PM25 - mean(PredDF$PM25)
### Étape 5 : construction du raster
rasterGAM <- rasterFromXYZ(PredDF[, c("X", "Y", "CenterPredPM25")])
crs(rasterGAM) <- crs(as(LyonIris, "Spatial"))
rasterGAM <- rast(rasterGAM)
### Étape 6 : découpage et cartographie du raster
LyonIris.SpatVector <- vect(LyonIris)
rasterGAM <- terra::mask(rasterGAM, LyonIris.SpatVector)
terra::plot(rasterGAM)
```

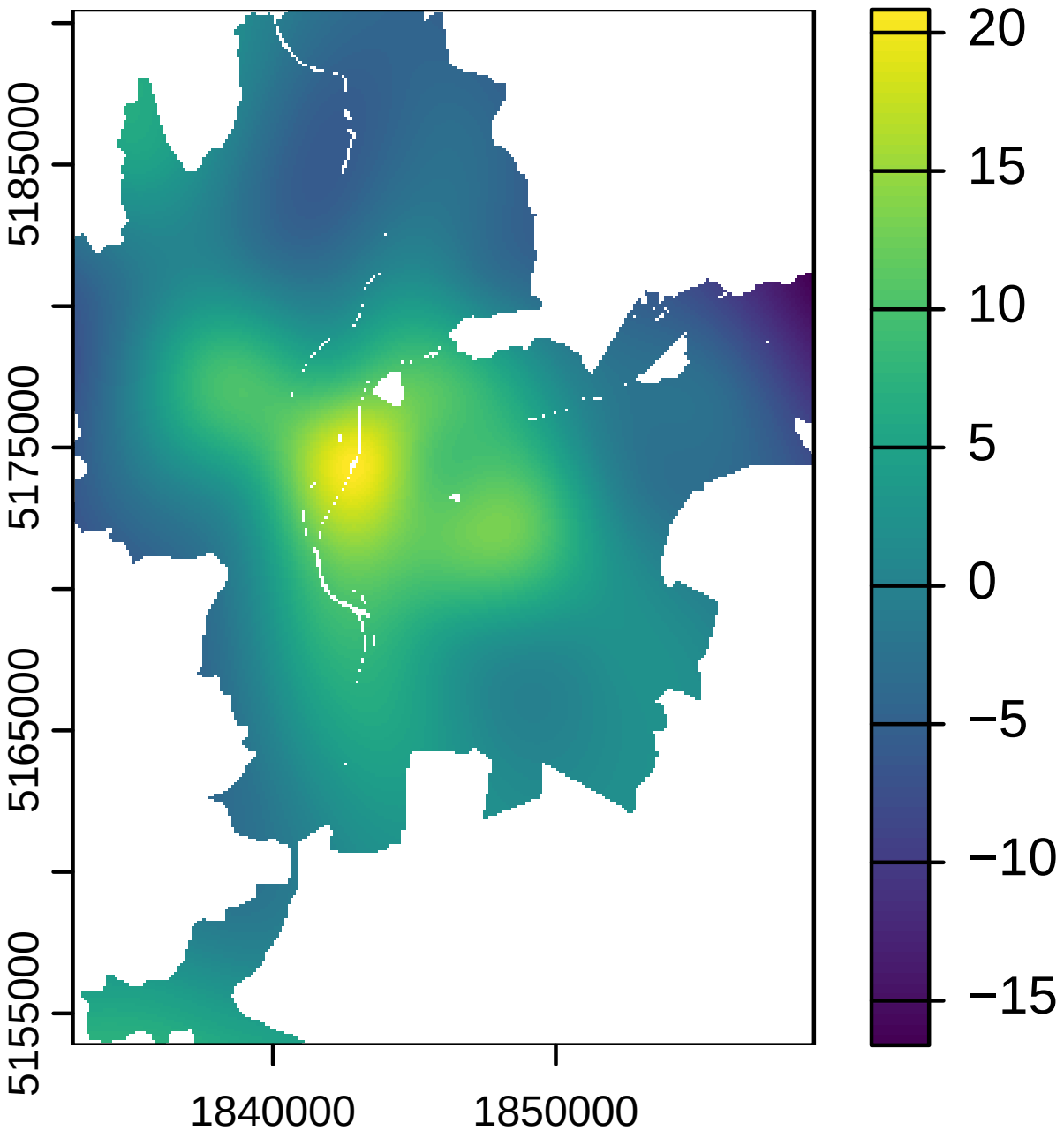


Figure 7.7 – Visualisation de l'effet de la localisation centrée sur zéro

La figure 7.7 signale que dans le centre de Lyon, le dioxyde d'azote est plus élevé de 10 à 20 $\mu g/m^3$, toutes choses étant égales par ailleurs. À l'inverse, dans les zones périphériques, il est faible. Cela signale un net patron spatial décroissant du centre vers la périphérie.

7.2.2.3 Dépendance spatiale du modèle GAM

Par contre, bien que l'autocorrélation spatiale des résidus du modèle GAM soit plus faible que pour le modèle MCO (I de Moran de 0,337 contre 0,570 avec $p < 0{,}001$), il reste que le problème de la dépendance spatiale n'est pas corrigé.

```
moran.mc(resid(Modele.GAM2), W.Rook, nsim=999)
```

```
    Monte-Carlo simulation of Moran I

data:  resid(Modele.GAM2)
weights: W.Rook
number of simulations + 1: 1000

statistic = 0.33664, observed rank = 1000, p-value = 0.001
alternative hypothesis: greater
```

7.3 Régression géographiquement pondérée

La régression géographiquement pondérée (*geographically weighted regression* - GWR, en anglais) a été proposée par Fotheringham *et al.* (2003) pour modéliser une variable continue. Depuis, plusieurs extensions ont été proposées, notamment des GWR mixtes, des GWR logistiques ou Poisson. Dans le cadre de cette section, nous abordons uniquement sa forme classique (variable dépendante continue).

7.3.1 Principe de base

Pourquoi recourir à la GWR?

Dans la section 7.1, nous avons vu que les modèles autorégressifs visent à contrôler la **dépendance spatiale** d'un modèle de régression classique (MCO), afin d'améliorer l'estimation des coefficients de régression. L'objectif des modèles de régression géographiquement pondérée est différent : ils visent à analyser les variations spatiales de la relation entre la variable dépendante et les variables indépendantes.

Autrement dit, les modèles GWR visent à explorer **l'instabilité spatiale du modèle MCO** afin d'analyser localement la relation entre la variable dépendante et les variables indépendantes. Pour une description détaillée en français de la GWR, consultez Apparicio *et al.* (2007).

Formulation de la GWR

Contrairement à la régression linéaire classique et aux modèles spatiaux autorégressifs qui produisent une équation pour l'ensemble du tableau de données, la GWR produit une équation pour chaque unité spatiale *i* et ainsi, des valeurs locales de R^2, β_0, β_k, *t* de Student, etc. La résolution de cette équation de régression locale est aussi basée sur la méthode des moindres carrés et sur une matrice de pondération W(i) dont les valeurs décroissent en fonction de la distance séparant les unités *i* et *j*. Autrement dit, plus *j* est proche de *i*, plus sa pondération est élevée et donc plus son « rôle » dans la détermination de l'équation de régression locale de *i* est important.

De la sorte, la GWR est une extension de la régression linéaire multiple classique où (u_i, v_i) représente les coordonnées géographiques du centroïde de l'unité spatiale et où les paramètres β_0 et β_k peuvent varier dans l'espace (équation 7.9).

$$y_i = \beta_0(u_i, v_i) + \sum_{j=1}^{k} \beta_j(u_i, v_i) x_{ij} + \epsilon_i \tag{7.9}$$

avec :

- (u_i, v_i), les coordonnées géographiques de l'unité spatiale *i*.
- y_i, la variable dépendante pour l'unité spatiale *i*.
- $\beta_0(u_i, v_i)$, la constante pour l'unité spatiale *i* aux coordonnées géographiques (u_i, v_i).
- $\beta_j(u_i, v_i)$, le coefficient de régression pour la variable x_j (avec *k* variables indépendantes) pour l'unité spatiale *i* aux coordonnées géographiques (u_i, v_i).
- x_{ij}, la valeur de la variable indépendante x_j pour l'unité spatiale *i*.
- ϵ_i, le terme d'erreur pour l'unité spatiale *i*.

Fotheringham *et al.* (2003) proposent deux fonctions *kernel* pour définir la pondération W(i) dans le modèle GWR : une fonction gaussienne (équation 7.10) et une fonction bicarrée (équation 7.11) où d_{ij} représente la distance euclidienne entre les points *i* et *j* et *b*, le rayon de zone d'influence autour du point *i* (*bandwidth*). Il existe une différence fondamentale entre les deux : la fonction gaussienne accorde un poids non nul à tous les points de l'espace d'étude aussi loin soient-ils, tandis que la fonction bicarrée ne tient pas compte des points distants à plus de *b* mètres de *i*, tel qu'illustré à la figure 7.8 avec une valeur fixée à 5000 mètres en guise d'exemple.

$$w_{ij} = exp[-.5(d_{ij}/b)^2] \tag{7.10}$$

$$w_{ij} = [1 - (d_{ij}/b)^2]^2 \text{ si } d_{ij} < b, \text{ sinon } w_{ij} = 0 \tag{7.11}$$

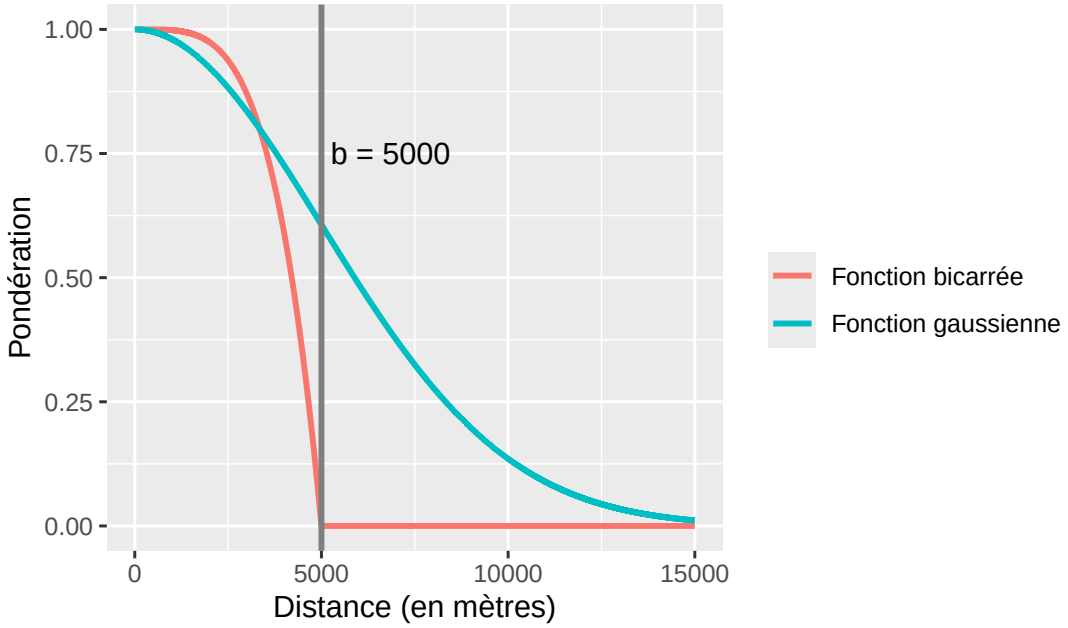


Figure 7.8 – Fonctions kernel pour définir la matrice de pondération W(i)

Dans le modèle GWR, la valeur de b est soit fixée par la personne utilisatrice, soit optimisée avec la valeur de CV (*cross-validation*) ou celle de l'AIC. Notez qu'il est possible d'optimiser la taille de la zone d'influence à partir de la distance euclidienne ou du nombre de plus proches voisins.

7.3.2 Construction et analyse du modèle GWR dans R

Pour construire un modèle GWR dans R, nous utilisons le *package* `spgwr` (Bivand et Yu 2023). La construction d'un modèle GWR comprend les étapes suivantes :

1. Sélection de la taille de la zone d'influence (*bandwidth*) optimale.
2. Réalisation de la GWR avec la taille de la zone d'influence optimale.
3. Comparaison des modèles MCO et GWR.
4. Cartographie des résultats du modèle GWR (R^2, coefficients, valeurs de t, etc.).

7.3.2.1 Définition de la taille de la zone d'influence

La sélection de la taille de la zone d'influence optimale est réalisée avec la fonction `gwr.sel` pour laquelle :

- le paramètre `gweight` permet de spécifier une fonction *kernel* gaussienne (`gwr.gauss`) ou bicarrée (`gwr.gauss`).
- le paramètre `adapt` permet de spécifier si vous optimisez le nombre de plus proches voisins (`adapt=TRUE`) ou la distance (`adapt=FALSE`).

```
library(spgwr)
## Optimisation du nombre de voisins avec le CV
bwaCV.voisins  <- gwr.sel(NO2 ~  Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed,
                          data = LyonIris,
                          method = "cv",          # Méthode cv ou AIC
                          gweight=gwr.bisquare,   # gwr.gauss ou gwr.bisquare
                          adapt=TRUE,
                          verbose = FALSE,
                          RMSE = TRUE,
                          longlat = FALSE,
                          coords=cbind(LyonIris$X,LyonIris$Y))
## Optimisation du nombre de voisins avec l'AIC
bwaAIC.voisins <- gwr.sel(NO2 ~  Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed,
                          data = LyonIris,
                          method = "AIC",          # Méthode cv ou AIC
                          gweight=gwr.bisquare,    # gwr.gauss ou gwr.bisquare
                          adapt=TRUE,              # adaptatif
                          verbose = FALSE,
                          RMSE = TRUE,
                          longlat = FALSE,
                          coords=cbind(LyonIris$X,LyonIris$Y))
## Optimisation de la distance avec le CV
bwnaCV.dist <- gwr.sel(NO2 ~  Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed,
```

```
                          data = LyonIris,
                          method = "cv",     # méthode cv ou AIC
                          gweight=gwr.Gauss, # gwr.gauss ou gwr.bisquare
                          adapt=FALSE,       # non adaptatif
                          verbose = FALSE,
                          RMSE = TRUE,
                          longlat = FALSE,
                          coords=cbind(LyonIris$X,LyonIris$Y))
## Optimisation de la distance avec l'AIC
bwnaAIC.dist <- gwr.sel(NO2 ~  Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed,
                        data = LyonIris,
                        method = "AIC",      # méthode cv ou AIC
                        gweight=gwr.Gauss,   # gwr.gauss ou gwr.bisquare
                        adapt=FALSE,         # non adaptatif
                        RMSE = TRUE,
                        verbose = FALSE,
                        longlat = FALSE,
                        coords=cbind(LyonIris$X,LyonIris$Y))
## Affichage des résultats d'optimisation
cat("Sélection de la taille de la zone optimale (bandwidth)",
    "\n avec le nombre de plus proches voisins :",
    "\n  CV =", round(bwaCV.voisins,4), "nombre de voisins =",
    round(bwaCV.voisins*nrow(LyonIris)),
    "\n  AIC =", round(bwaAIC.voisins,4), "nombre de voisins =",
    round(bwaAIC.voisins*nrow(LyonIris)),
    "\nSélection de la taille de la zone optimale (bandwidth) avec la distance :",
    "\n  CV =", round(bwnaCV.dist, 0),  "mètres",
    "\n  AIC =", round(bwnaAIC.dist, 0), "mètres")
```

```
Sélection de la taille de la zone optimale (bandwidth)
 avec le nombre de plus proches voisins :
  CV = 0.1818 nombre de voisins = 92
  AIC = 0.1067 nombre de voisins = 54
Sélection de la taille de la zone optimale (bandwidth) avec la distance :
  CV = 1315 mètres
  AIC = 1662 mètres
```

Les résultats ci-dessus montrent que le nombre de plus proches voisins pourrait être de 92 selon l'approche *cross-validation* et de 54 selon la méthode basée sur l'AIC. Si la valeur de b est basée sur la distance, elle serait alors optimale à 1315 et 1662 mètres selon les deux méthodes.

7.3.2.2 Réalisation de la GWR

Avec la fonction `gwr`, nous estimons un modèle GWR avec un *kernel* bicarré et un nombre optimisé de plus voisins selon la méthode CV, soit 92.

```
Modele.GWR <- gwr(NO2 ~ Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed,
              data = LyonIris,
              adapt=bwaCV.voisins,
              gweight=gwr.bisquare,
              hatmatrix=TRUE,
              se.fit=TRUE,
              coords=cbind(LyonIris$X,LyonIris$Y),
              longlat=F)
```

Le code ci-dessous permet de renvoyer les statistiques univariées des coefficients des 506 régressions locales, réalisées pour chacune des 506 entités spatiales (IRIS), et les statistiques d'ajustement du modèle (AIC, R^2 global, etc.)

```
Modele.GWR
```

```
Call:
gwr(formula = NO2 ~ Pct0_14 + Pct_65 + Pct_Img + Pct_brevet +
    NivVieMed, data = LyonIris, coords = cbind(LyonIris$X, LyonIris$Y),
    gweight = gwr.bisquare, adapt = bwaCV.voisins, hatmatrix = TRUE,
    longlat = F, se.fit = TRUE)
Kernel function: gwr.bisquare
Adaptive quantile: 0.1818192 (about 92 of 506 data points)
Summary of GWR coefficient estimates at data points:
                  Min.   1st Qu.    Median   3rd Qu.      Max.  Global
X.Intercept. 12.429518 30.249511 38.619342 48.038863 60.098584 49.4330
Pct0_14      -1.094802 -0.360556 -0.215643 -0.047687  0.382801 -0.5335
Pct_65       -0.715331 -0.158253 -0.031353  0.076086  0.464992 -0.1505
Pct_Img      -0.331892 -0.049146  0.077177  0.240755  0.670433  0.2829
Pct_brevet   -0.655221 -0.221655 -0.084954  0.047835  0.598456 -0.2400
NivVieMed    -1.140895 -0.560649 -0.214717  0.193768  1.311228 -0.3162
Number of data points: 506
Effective number of parameters (residual: 2traceS - traceS'S): 107.1278
Effective degrees of freedom (residual: 2traceS - traceS'S): 398.8722
Sigma (residual: 2traceS - traceS'S): 4.118116
Effective number of parameters (model: traceS): 81.53263
Effective degrees of freedom (model: traceS): 424.4674
Sigma (model: traceS): 3.992025
Sigma (ML): 3.656286
AICc (GWR p. 61, eq 2.33; p. 96, eq. 4.21): 2945.674
AIC (GWR p. 96, eq. 4.22): 2829.504
Residual sum of squares: 6764.424
Quasi-global R2: 0.783008
```

7.3.2.3 Comparaison des modèles MCO et GWR

Le R^2 global du modèle GWR est bien supérieur au modèle classique MCO (0,783 contre 0,283). Fotheringham *et al.* (2003) proposent plusieurs tests pour comparer les modèles GWR et classique qui sont implémentés dans

le *package* `spgwr` (fonctions `anova(Modele.GWR)`, `anova(Modele.GWR, approx=TRUE)`, `LMZ.F1GWR.test(Modele.GWR)`, `LMZ.F2GWR.test(Modele.GWR)`.

Si les valeurs de p de ces tests sont inférieures à 0,05, alors le modèle GWR améliore de façon significative la capacité prédictive du modèle de régression globale, ce que confirment les résultats ci-dessous.

```
anova(Modele.GWR)
```

```
Analysis of Variance Table
                    Df  Sum Sq Mean Sq F value
OLS Residuals     6.00 22346.0
GWR Improvement 101.13 15581.6 154.078
GWR Residuals   398.87  6764.4  16.959  9.0854
```

```
anova(Modele.GWR, approx=TRUE)
```

```
Analysis of Variance Table
approximate degrees of freedom (only tr(S))
                     Df  Sum Sq Mean Sq F value
OLS Residuals     6.000 22346.0
GWR Improvement  75.533 15581.6 206.290
GWR Residuals   424.467  6764.4  15.936  12.945
```

```
LMZ.F1GWR.test(Modele.GWR)
```

```

    Leung et al. (2000) F(1) test

data:  Modele.GWR
F = 0.37946, df1 = 430.81, df2 = 500.00, p-value < 2.2e-16
alternative hypothesis: less
sample estimates:
SS OLS residuals SS GWR residuals
       22346.021         6764.424
```

```
LMZ.F2GWR.test(Modele.GWR)
```

```

    Leung et al. (2000) F(2) test

data:  Modele.GWR
F = 3.4476, df1 = 142.92, df2 = 500.00, p-value < 2.2e-16
alternative hypothesis: greater
sample estimates:
  SS OLS residuals SS GWR improvement
          22346.02           15581.60
```

Un autre test (`LMZ.F3GWR.test`) permet de répondre à la question suivante : est-ce que les coefficients de régression du modèle GWR varient spatialement de façon significative? Les résultats ci-dessous démontrent que c'est le cas pour toutes les variables indépendantes et la constante ($p < 0{,}001$).

```
LMZ.F3GWR.test(Modele.GWR)
```

```
Leung et al. (2000) F(3) test

            F statistic Numerator d.f. Denominator d.f.     Pr(>)
(Intercept)      2.2771       134.3880           430.81 1.629e-10 ***
Pct0_14          2.7767       141.7244           430.81 5.636e-16 ***
Pct_65           2.0918       169.0472           430.81 8.399e-10 ***
Pct_Img          1.9486       106.4400           430.81 1.550e-06 ***
Pct_brevet       2.4445       121.2830           430.81 1.629e-11 ***
NivVieMed        3.6926       138.8118           430.81 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

7.3.2.4 Cartographie des résultats du modèle GWR

Dans un premier temps, nous ajoutons les valeurs locales des R^2, des coefficients de régression et des valeurs de t dans la couche `sf`. Notez que les résultats locaux de la GWR sont stockés dans l'objet `Modele.GWR$SDF`.

```
## Récupération du R carré local
LyonIris$GWR.R2 <- Modele.GWR$SDF$localR2
## Récupération des coefficients de régression et calcul des valeurs de t locales
names(Modele.GWR$SDF)
```

```
 [1] "sum.w"              "(Intercept)"        "Pct0_14"
 [4] "Pct_65"             "Pct_Img"            "Pct_brevet"
 [7] "NivVieMed"          "(Intercept)_se"     "Pct0_14_se"
[10] "Pct_65_se"          "Pct_Img_se"         "Pct_brevet_se"
[13] "NivVieMed_se"       "gwr.e"              "pred"
[16] "pred.se"            "localR2"            "(Intercept)_se_EDF"
[19] "Pct0_14_se_EDF"     "Pct_65_se_EDF"      "Pct_Img_se_EDF"
[22] "Pct_brevet_se_EDF"  "NivVieMed_se_EDF"   "pred.se"
```

```
VarsIndep <- c("Pct0_14", "Pct_65", "Pct_Img", "Pct_brevet", "NivVieMed")
for(e in VarsIndep){
  # Nom des nouvelles variables
  var.coef <- paste0("GWR.", "B_", e)
  var.t    <- paste0("GWR.", "T_", e)
  # Récupération des coefficients pour les variables indépendantes
  LyonIris[[var.coef]] <- Modele.GWR$SDF[[e]]
```

```
  # Calcul des valeurs de t pour les variables indépendantes
  LyonIris[[var.t]]    <- Modele.GWR$SDF[[e]] / Modele.GWR$SDF[[paste0(e, "_se")]]
}
```

Cartographie des R^2 locaux

Le code ci-dessous permet ensuite de cartographier les R^2 locaux de la GWR (figure 7.9).

```
library(tmap)
tm_shape(LyonIris)+
  tm_borders(col="gray25", lwd=.5)+
  tm_fill(col="GWR.R2",
          palette="YlOrBr",
          n=5, style="quantile",
          legend.format = list(text.separator = "à"),
          title = "R2 locaux")+
  tm_layout(frame=FALSE)+
  tm_scale_bar(breaks=c(0,5))
```

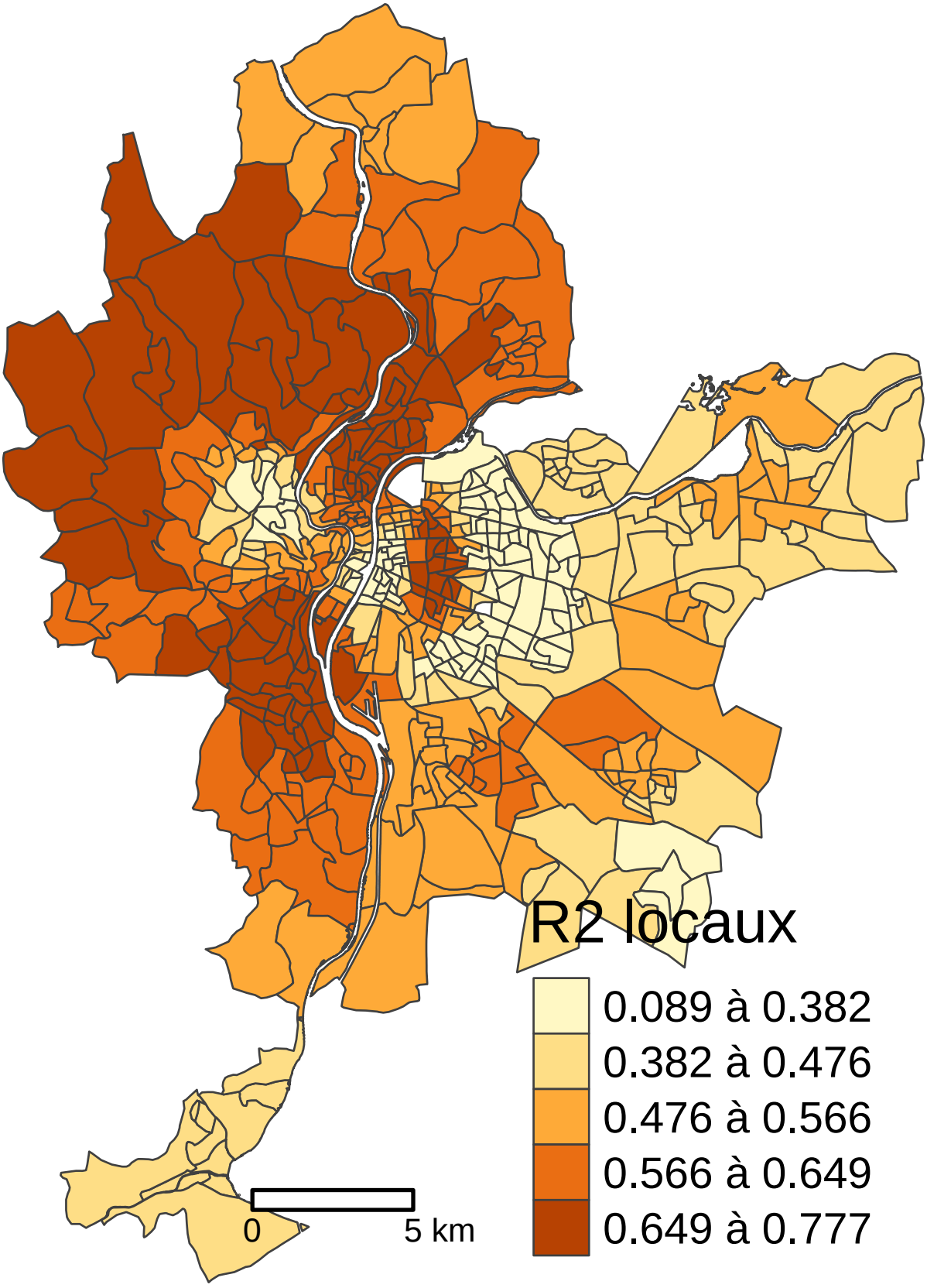


Figure 7.9 – Cartographie des R carrés locaux de la GWR

Cartographie des coefficients de régression

Le code ci-dessous permet ensuite de cartographier les coefficients locaux de la GWR (figure 7.10).

```
Carte1 <- tm_shape(LyonIris)+ tm_borders(col="gray25", lwd=.5)+
          tm_fill(col="GWR.B_Pct0_14", palette="YlOrBr", n=4, style="pretty",
                  legend.format = list(text.separator = "à"),
                  title = "Moins de 15 ans (%)")+
          tm_layout(frame=FALSE, legend.outside = TRUE)+tm_scale_bar(breaks=c(0,5))
Carte2 <- tm_shape(LyonIris)+ tm_borders(col="gray25", lwd=.5)+
          tm_fill(col="GWR.B_Pct_65", palette="YlOrBr", n=4, style="pretty",
                  legend.format = list(text.separator = "à"),
                  title = "65 ans et plus (%)")+
          tm_layout(frame=FALSE, legend.outside = TRUE)+tm_scale_bar(breaks=c(0,5))
Carte3 <- tm_shape(LyonIris)+ tm_borders(col="gray25", lwd=.5)+
          tm_fill(col="GWR.B_Pct_Img", palette="YlOrBr", n=4, style="pretty",
                  legend.format = list(text.separator = "à"),
                  title = "Immigrants (%)")+
          tm_layout(frame=FALSE, legend.outside = TRUE)+tm_scale_bar(breaks=c(0,5))
Carte4 <- tm_shape(LyonIris)+ tm_borders(col="gray25", lwd=.5)+
          tm_fill(col="GWR.B_Pct_brevet", palette="YlOrBr", n=4, style="pretty",
                  legend.format = list(text.separator = "à"),
                  title = "Faible scolarité (%)")+
          tm_layout(frame=FALSE, legend.outside = TRUE)+tm_scale_bar(breaks=c(0,5))
Carte5 <- tm_shape(LyonIris)+ tm_borders(col="gray25", lwd=.5)+
          tm_fill(col="GWR.B_NivVieMed", palette="YlOrBr", n=4, style="pretty",
                  legend.format = list(text.separator = "à"),
                  title = "Niveau de vie (€1000)")+
          tm_layout(frame=FALSE, legend.outside = TRUE)+tm_scale_bar(breaks=c(0,5))
tmap_arrange(Carte1, Carte2, Carte3, Carte4, Carte5, ncol = 2, nrow=3)
```

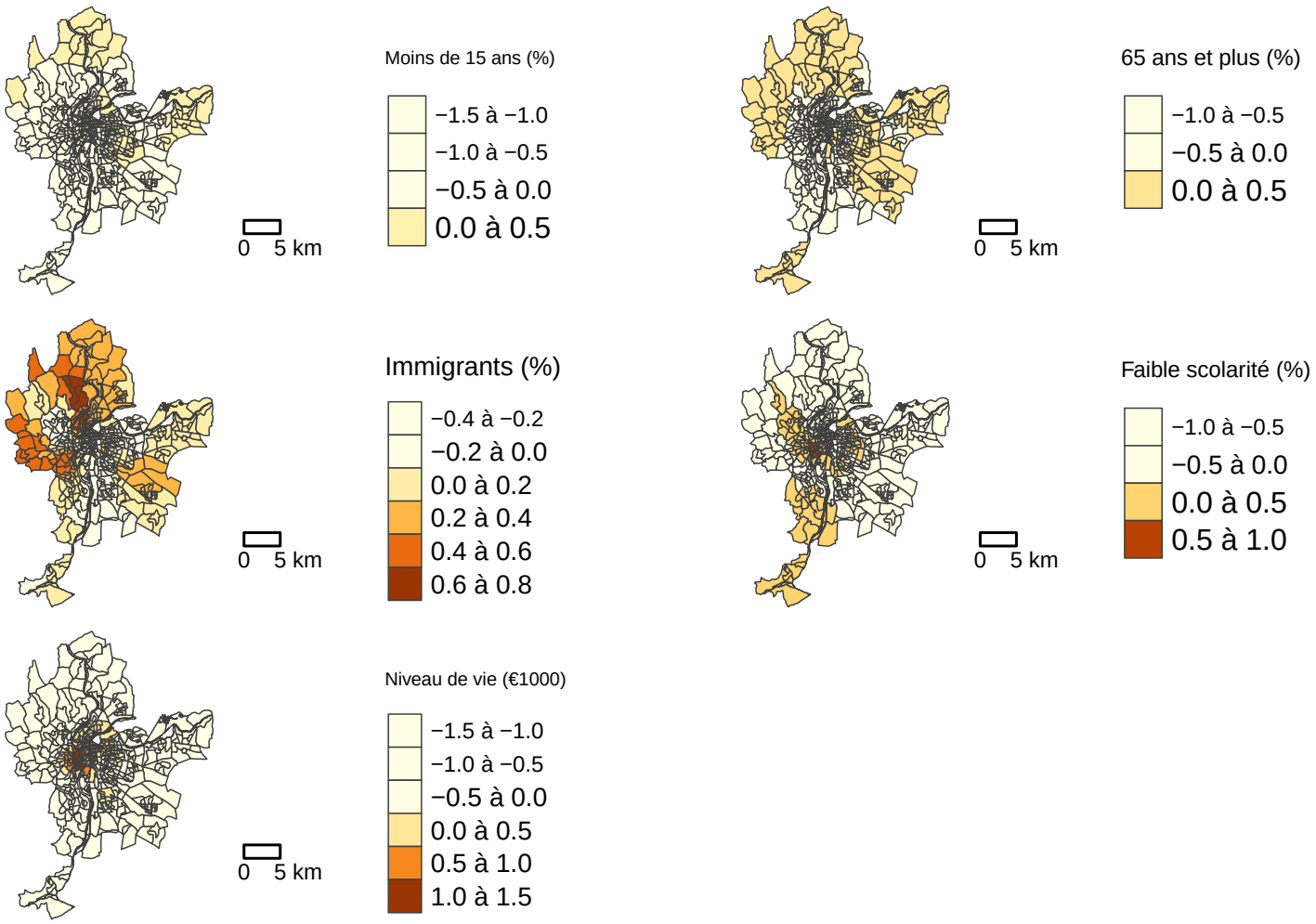


FIGURE 7.10 – Cartographie des coefficients de régression de la GWR

Cartographie des valeurs de *t*

Pour cartographier les valeurs de *t*, nous utilisons les seuils de ± 1,96, 2,58 et 3,29, indiquant des seuils de signification à 5 %, 1 % et 0,1 % (figure 7.11).

```
classes.intervalles = c(-Inf, -3.29, -2.58, -1.96, 1.96, 2.58, 3.29, Inf)
Carte1 <- tm_shape(LyonIris)+ tm_borders(col="gray25", lwd=.5)+
          tm_fill(col="GWR.T_Pct0_14", palette="-RdBu",
                  breaks = classes.intervalles,
                  legend.format = list(text.separator = "à"),
                  title = "Moins de 15 ans (%)")+
          tm_layout(frame=FALSE, legend.outside = TRUE)
Carte2 <- tm_shape(LyonIris)+ tm_borders(col="gray25", lwd=.5)+
          tm_fill(col="GWR.T_Pct_65", palette="-RdBu",
                  breaks = classes.intervalles,
                  legend.format = list(text.separator = "à"),
```

```
                  title = "65 ans et plus (%)")+
          tm_layout(frame=FALSE, legend.outside = TRUE)
Carte3 <- tm_shape(LyonIris)+ tm_borders(col="gray25", lwd=.5)+
          tm_fill(col="GWR.T_Pct_Img", palette="-RdBu",
                  breaks = classes.intervalles,
                  legend.format = list(text.separator = "à"),
                  title = "Immigrants (%)")+
          tm_layout(frame=FALSE, legend.outside = TRUE)
Carte4 <- tm_shape(LyonIris)+ tm_borders(col="gray25", lwd=.5)+
          tm_fill(col="GWR.B_Pct_brevet", palette="-RdBu",
                  breaks = classes.intervalles,
                  legend.format = list(text.separator = "à"),
                  title = "Faible scolarité (%)")+
          tm_layout(frame=FALSE, legend.outside = TRUE)
Carte5 <- tm_shape(LyonIris)+ tm_borders(col="gray25", lwd=.5)+
          tm_fill(col="GWR.T_NivVieMed", palette="-RdBu",
                 breaks = classes.intervalles,
                  legend.format = list(text.separator = "à"),
                 title = "Niveau de vie (€1000)")+
          tm_layout(frame=FALSE, legend.outside = TRUE)+
          tm_scale_bar(breaks=c(0,5))
tmap_arrange(Carte1, Carte2, Carte3, Carte4, Carte5, ncol = 2, nrow=3)
```

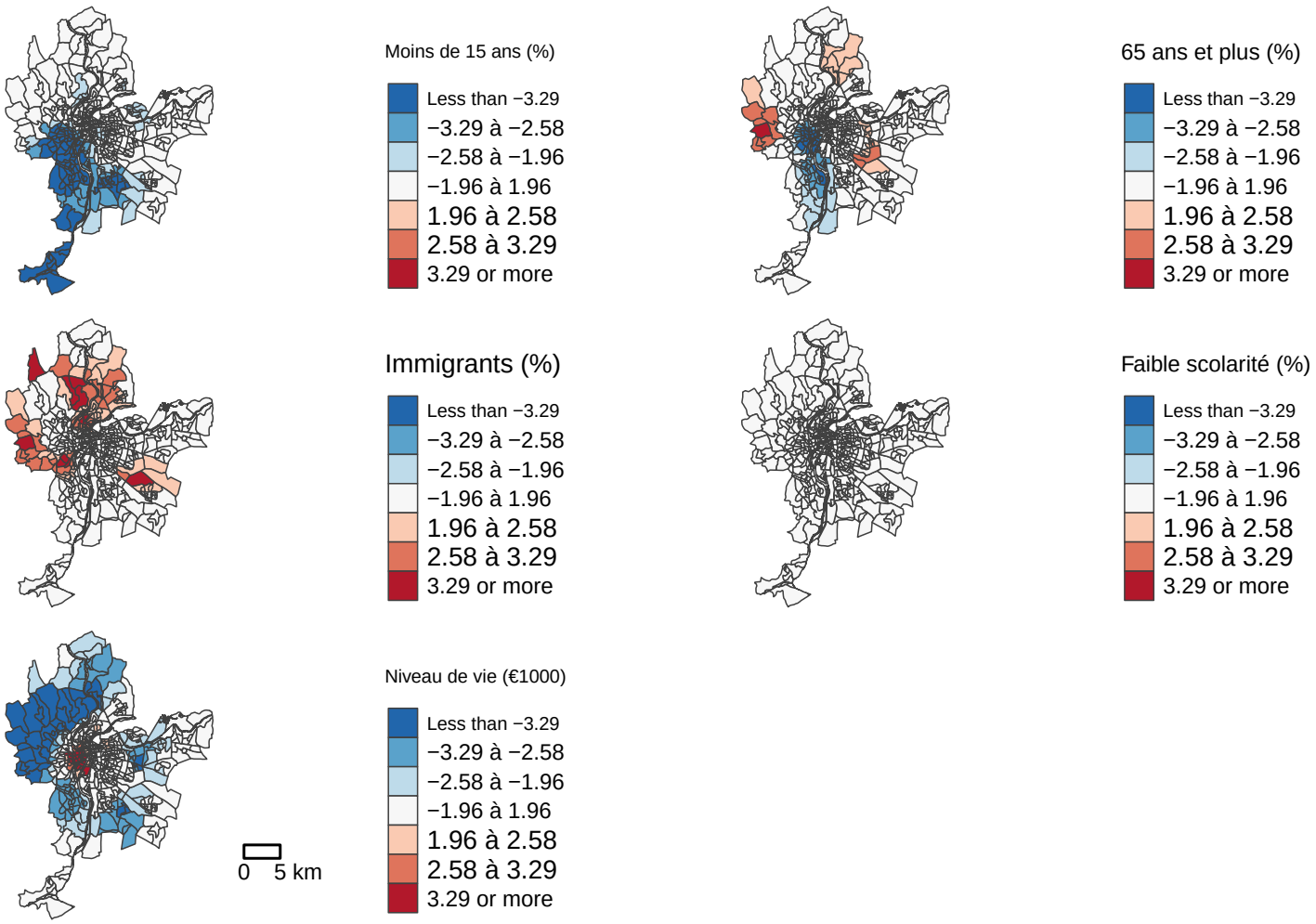


FIGURE 7.11 – Cartographie des valeurs de t de la GWR

Cartographie du nombre de variables significatives

Nous pouvons aussi cartographier le nombre de variables localement significatives aux seuils de 5 % et 1 %.

```
## Identifier la variable plus significative avec les valeurs de t
VarsT <- paste0("GWR.T_", c("Pct0_14", "Pct_65", "Pct_Img", "Pct_brevet", "NivVieMed"))
Lyon.df  <- st_drop_geometry(LyonIris)
Lyon.df <- abs(Lyon.df[,VarsT])
PlusSign <- VarsT[apply(Lyon.df[VarsT],1,which.max)]
PlusSign <- substr(PlusSign, 7, nchar(PlusSign))
MaxAbsTvalue <- apply(Lyon.df[VarsT], 1, max)
PlusSign <- ifelse(MaxAbsTvalue<1.96, "Aucune", PlusSign)
## Nombre de variables significatives au seuil de 5%, soit abs(t)= 1,96)
LyonIris$NbSignif_1.96 <- as.factor(rowSums(Lyon.df > 1.96))
LyonIris$NbSignif_2.58 <- as.factor(rowSums(Lyon.df > 2.58))
LyonIris$PlusSign      <- as.factor(PlusSign)
## Cartographie
```

```
Carte1 <- tm_shape(LyonIris)+ tm_borders(col="gray25", lwd=.5)+
          tm_fill(col="NbSignif_1.96", palette="Reds",
                  title = "Sign. au seuil de 5%")+
          tm_layout(frame=FALSE)+ tm_scale_bar(breaks=c(0,5))
Carte2 <- tm_shape(LyonIris)+ tm_borders(col="gray25", lwd=.5)+
          tm_fill(col="NbSignif_2.58", palette="Reds",
                  title = "Sign. au seuil de 1%")+
          tm_layout(frame=FALSE)
tmap_arrange(Carte1, Carte2, ncol=2, nrow=1)
```

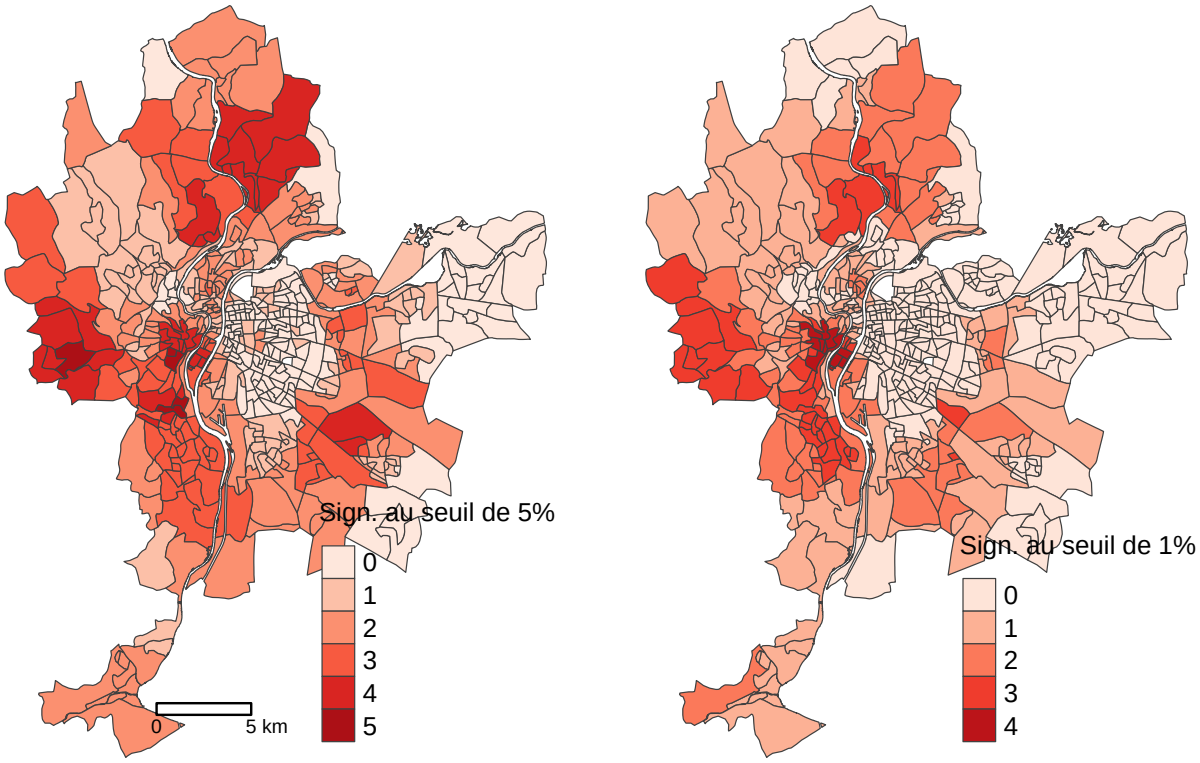


FIGURE 7.12 – Nombre de variables significatives aux seuils de 5% et 1%

Cartographie de la variable la plus significative avec la valeur de *t*

Finalement, le code ci-dessous permet de repérer la variable la plus significative au seuil de 5 %, c'est-à-dire avec la plus forte valeur absolue pour la valeur de *t*.

```
tm_shape(LyonIris)+ tm_borders(col="gray25", lwd=.5)+
          tm_fill(col="PlusSign", palette="Set1",
                  title = "Variable la plus significative")+
          tm_layout(frame=FALSE)+ tm_scale_bar(breaks=c(0,5))
```

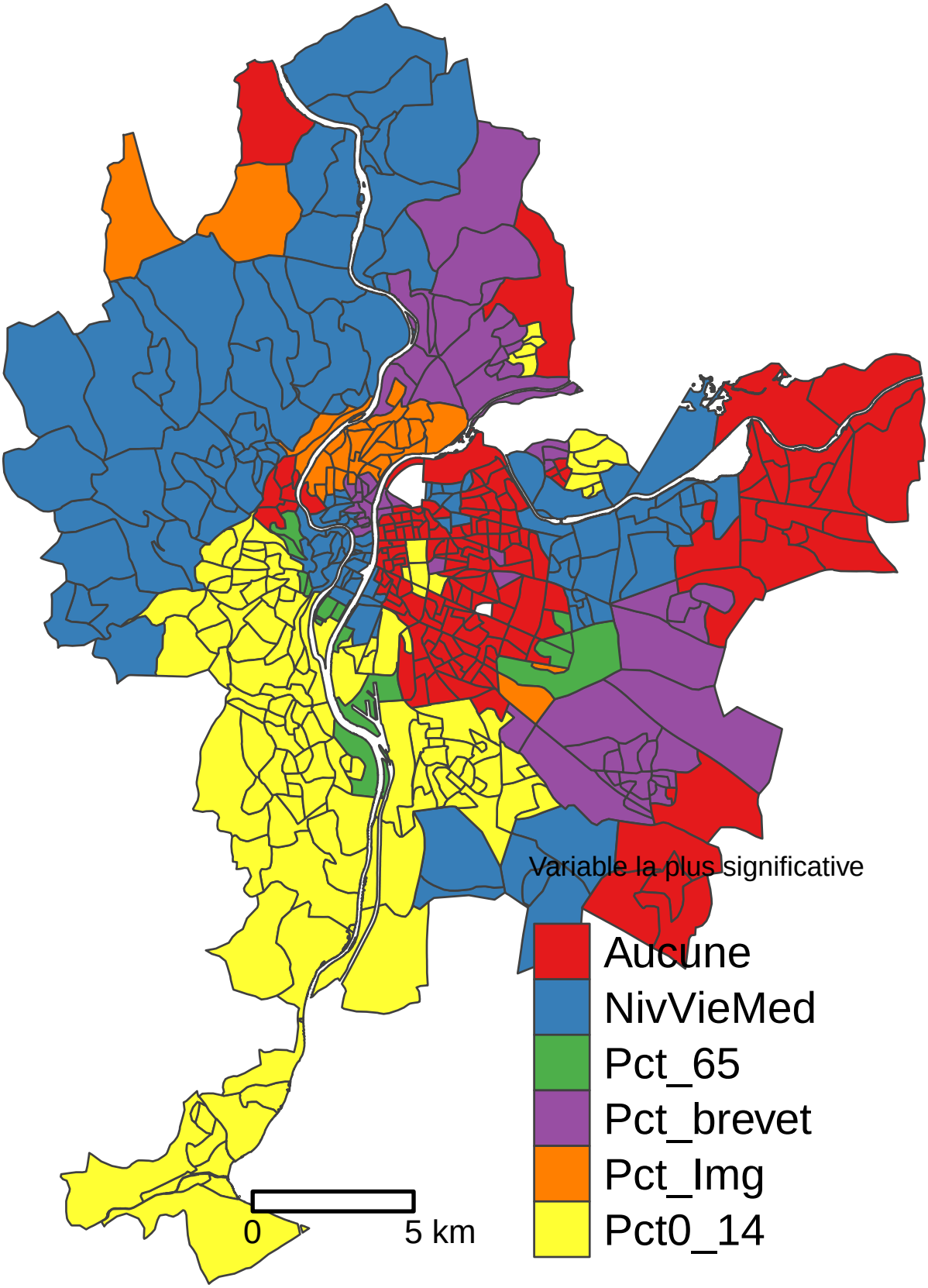


Figure 7.13 – Variable indépendante la plus significative au seuil de 5 %

Aller plus loin

Extensions de la GWR classique
À titre de rappel, la GWR classique permet de modéliser une variable dépendante continue. Plusieurs extensions ont été proposées, notamment :

- La GWR mixte qui permet de spécifier des variables indépendantes variant spatialement et d'autres étant fixes (Fotheringham, Brunsdon et Charlton 2003).
- Les GWR logistique (pour une variable dépendante binaire) et Poisson (pour une variable dépendante de comptage) (Fotheringham, Brunsdon et Charlton 2003).
- La régression géographiquement et temporellement pondérée (*Geographical and temporal weighted regression* – GTWR) (Fotheringham, Crespo et Yao 2015).
- La régression géographiquement pondérée multiéchelle (*Multiscale geographically weighted regression* – MGWR) (Fotheringham, Yang et Kang 2017).
- L'analyse en composantes principales géographiquement pondérée (*Geographically weighted principal components analysis* – GWR PCA) (Harris, Brunsdon et Charlton 2011).

Aller plus loin

Autres méthodes de régression spatiales
Comme signalé en introduction, si vous souhaitez maîtriser une plus large éventail de méthodes de régression spatiale, nous vous recommandons la lecture d'un autre ouvrage de la Série *Un grand Bol d'R* intitulé **Méthodes de régression spatiale : un grand bol d'R** (Apparicio et Gelb 2025).

7.4 Quiz de révision du chapitre

Questions

- **Qu'est-ce que la dépendance spatiale d'un modèle de régression?**
  - Lorsque les variables indépendantes sont fortement corrélées entre elles.
  - Lorsque les résidus du modèle sont fortement autocorrélés spatialement.

  Relisez au besoin la section 7.1.1.
- **Dans un modèle SLX, l'autocorrélation est introduite au niveau de :**
  - Variable dépendante
  - Variables indépendantes
  - Terme d'erreur
  - Variable dépendante et variables indépendantes
  - Variable dépendante et terme d'erreur

  Relisez au besoin la section 7.1.2.1.
- **Dans un modèle SAR, l'autocorrélation est introduite au niveau de :**
  - Variable dépendante
  - Variables indépendantes
  - Terme d'erreur

  Relisez au besoin la section 7.1.2.2.

— **Dans un modèle SEM, l'autocorrélation est introduite au niveau de :**

  — Variable dépendante
  — Variables indépendantes
  — Terme d'erreur

  Relisez au besoin la section 7.1.2.3.

— **Dans un modèle mixte SDM, l'autocorrélation est introduite au niveau de :**

  — Variable dépendante
  — Variables indépendantes
  — Terme d'erreur

  Relisez au besoin la section 7.1.2.4.

— **Dans un modèle SDEM, l'autocorrélation est introduite au niveau de :**

  — Variable dépendante
  — Variables indépendantes
  — Terme d'erreur

  Relisez au besoin la section 7.1.2.5.

— **Comment est intégré l'espace dans un modèle généralisé additif?**

  — Avec une variable spatialement décalée
  — Avec une spline bivariée sur les coordonnées x et y

  Relisez au besoin la section 7.2.1.

— **Un modèle de régression géographiquement pondérée permet d'explorer**

  — L'instabilité spatiale du modèle
  — La dépendance du modèle

  Relisez le deuxième encadré à la section 7.3.1.

— **Un modèle de régression géographiquement pondérée produit autant de régressions que d'entités spatiales dans le jeu de données à l'étude.**

  — Vrai
  — Faux

  Relisez le deuxième encadré à la section 7.3.1.

Réponses

— Qu'est-ce que la dépendance spatiale d'un modèle de régression?
  — Lorsque les résidus du modèle sont fortement autocorrélés spatialement.
— Dans un modèle SLX, l'autocorrélation est introduite au niveau de :
  — Variables indépendantes
— Dans un modèle SAR, l'autocorrélation est introduite au niveau de :
  — Variable dépendante
— Dans un modèle SEM, l'autocorrélation est introduite au niveau de :
  — Terme d'erreur
— Dans un modèle mixte SDM, l'autocorrélation est introduite au niveau de :
  — Variable dépendante
  — Variables indépendantes

- Dans un modèle SDEM, l'autocorrélation est introduite au niveau de :
  - Variables indépendantes
  - Terme d'erreur
- Comment est intégré l'espace dans un modèle généralisé additif?
  - Avec une spline bivariée sur les coordonnées x et y
- Un modèle de régression géographiquement pondérée permet d'explorer
  - L'instabilité spatiale du modèle
- Un modèle de régression géographiquement pondérée produit autant de régressions que d'entités spatiales dans le jeu de données à l'étude.
  - Vrai

7.5 Exercices de révision

Exercice

Exercice 1. Réalisation de modèles de régression autorégressifs spatiaux

```
library(sf)
library(spatialreg)
# Matrice de contiguïté selon le partage d'un segment (Rook)
load("data/chap07/DonneesLyon.Rdata")
Rook <- poly2nb(LyonIris, queen=FALSE)
Rook <- poly2nb(LyonIris, queen=FALSE)
W.Rook <- nb2listw(Rook, zero.policy=TRUE, style = "W")
# Modèles
formule <- "PM25 ~  Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed"
Modele.SLX <- à compléter
Modele.SAR <- à compléter
Modele.SEM <- à compléter
Modele.DurbinSpatial <- à compléter
Modele.DurbinErreur <- à compléter
```

Correction à la section 12.7.1.

Exercice

Exercice 2. Réalisation d'un modèle GAM

```
library(sf)
library(mgcv)
load("data/chap07/DonneesLyon.Rdata")
# Ajout des coordonnées x et y
xy <- à compléter
LyonIris$X <- à compléter
LyonIris$Y <- à compléter
# Construction du modèle
formule <- "PM25 ~  Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed"
Modele.GAM2 <- gam(NO2 ~  à compléter
                          à compléter,
                          data = LyonIris)
summary(Modele.GAM2)
```

Correction à la section 12.7.2.

Exercice

Exercice 2. Réalisation d'un modèle GWR

```
library(sf)
library(spgwr)
load("data/chap07/DonneesLyon.Rdata")
# Ajout des coordonnées x et y
xy <- à compléter
LyonIris$X <- à compléter
LyonIris$Y <- à compléter
# Optimisation du nombre de voisins avec le CV
formule <- "PM25 ~ Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed"
bwaCV.voisins  <- gwr.sel(à compléter)
# Optimisation du nombre de voisins avec l'AIC
bwaCV.voisins  <- gwr.sel(à compléter)
# Réalisation de la GWR
Modele.GWR <- gwr(à compléter)
# Affichage des résultats
Modele.GWR
```

Correction à la section 12.7.3.

8 Méthodes de classification non supervisée spatiale

Les méthodes de classification visent à regrouper des observations d'un jeu de données en plusieurs classes en fonction de leurs caractéristiques évaluées à partir de plusieurs variables. Appliquées à une couche spatiale (polygones, points, lignes), il s'agit alors de classifier les unités spatiales sur la base de plusieurs de leurs attributs mesurés à partir de variables. Dans le cadre de ce chapitre, nous abordons deux principales familles de méthodes de classification non supervisée : **celle avec une contrainte spatiale** (algorithmes AZP, SKATER, REDCAP) et **celle avec une dimension spatiale** (*ClustGeo* et classification floue c-moyennes spatiale).

Sommairement, les deux dernières méthodes sont des extensions spatiales de la classification ascendante hiérarchique (CAH) et de l'algorithme flou c-moyennes. Par conséquent, la lecture de ce chapitre nécessite de bien maîtriser le fonctionnement de la CAH et des k-moyennes (*k-means*). Si ce n'est pas le cas, nous vous invitons vivement à lire le chapitre suivant (Apparicio et Gelb 2022).

⚠ **Attention**

Bref retour sur les méthodes de classification
Il existe de nombreuses méthodes de classification. Nous distinguons habituellement plusieurs familles de méthodes de classification, celles non supervisées versus supervisées et celles strictes versus floues :

- **Les méthodes de classification non supervisée** « […] relèvent de la statistique exploratoire multidimensionnelle et permettent de regrouper automatiquement les observations sans avoir de connaissance préalable sur la nature des classes présentes dans l'ensemble de données (Lebart, Morineau et Piron 1995). Les méthodes les plus connues dans ce domaine sont l'algorithme de Classification ascendante hiérarchique (CAH) et la méthode des k-moyennes (*k-means*) » (Jérémy Gelb et Apparicio 2021, 1). À cela s'ajoutent d'autres méthodes comme les k-médianes (Jain et Dubes 1988) ou encore les k-médoïdes (Kaufman 1990) et la classification mixte combinant k-moyennes et CAH (Lebart, Morineau et Piron 1995). Pour regrouper les observations, ces méthodes (CAH, k-moyennes, k-médianes, k-médoïdes, classification mixte) sont basées sur la distance (proximité) entre les observations tandis que d'autres méthodes sont basées sur la densité des observations (algorithmes DBSCAN, HDBSCAN, STDBSCAN, OPTICS abordés à la section 4.1).
- **Les méthodes de classification supervisée** « […] permettent d'affecter des observations à partir d'un échantillon déjà classifié, souvent appelé classe d'entraînement. Parmi les méthodes supervisées les plus connues, on retrouve les forêts d'arbres décisionnels, les réseaux de neurones artificiels et l'analyse factorielle discriminante » (Jérémy Gelb et Apparicio 2021, 1). Plus exactement, ces méthodes visent à apprendre des règles basées sur les attributs des observations pour déterminer à quel groupe chaque observation doit être attribuée. Ces règles peuvent ensuite être utilisées pour déterminer la catégorie de nouvelles observations.
- Que la classification soit ou non supervisée, « on distingue généralement les **méthodes strictes** (ou de partition) des **méthodes floues**. […] Dans une classification stricte, chaque observation appartient à une seule classe : mathématiquement, l'appartenance à une classe donnée est binaire (0 ou 1), tandis que dans une classification floue, chaque observation a une probabilité d'appartenance variant de 0 à 1 pour chacune des classes » (Jérémy Gelb et Apparicio 2021, 1-2).

Pourquoi recourir à des méthodes de classification non supervisée spatiale?

Dans un article récent, Gelb et Apparicio (2021) identifient deux principales limites à l'application d'une méthode de classification non supervisée a-spatiale (comme le CAH et le *k-means*) sur des données spatiales :

1. **La non-prise en compte de la dimension spatiale constitue une perte d'information :** « […] une partie de l'information propre aux données, à savoir leur localisation, n'est pas prise en compte dans le processus de classification. Or, la dimension géographique est souvent très structurante; par conséquent, l'occulter revient à perdre une quantité non négligeable d'information. Il convient toutefois de nuancer quelque peu ce propos. Généralement, la cartographie des méthodes de classification non supervisée a-spatiale (CAH et *k-means*) révèlent des effets de voisinage, d'autant plus que les variables introduites dans la classification sont fortement autocorrélées positivement » (Jérémy Gelb et Apparicio 2021, 6).
2. **Limiter l'effet de mitage** : « […] dans un contexte d'autocorrélation spatiale positive, des observations proches spatialement devraient plus vraisemblablement appartenir au même groupe. Avec les méthodes de classification a-spatiale, il est fréquent d'observer des phénomènes de mitage, c'est-à-dire des observations appartenant à un groupe *b* et isolées au milieu d'un ensemble d'observations appartenant au groupe *a*. Ce phénomène peut s'expliquer par la présence ici et là d'autocorrélation spatiale locale négative, c'est-à-dire des observations dont les caractéristiques sémantiques diffèrent de leurs voisines. Souvent, la dissimilarité sémantique entre ces observations est négligeable et ne justifie pas cette rupture spatiale » (Jérémy Gelb et Apparicio 2021, 6).

Package

Liste des *packages* utilisés dans ce chapitre

- Pour importer et manipuler des fichiers géographiques :
  - `sf` pour importer et manipuler des données vectorielles.
  - `spdep` pour construire des matrices de pondération spatiale.
- Pour construire des cartes et des graphiques :
  - `tmap` est certainement le meilleur *package* pour la cartographie.
  - `ggplot2` pour construire des graphiques.
  - `ggpubr` pour combiner des graphiques.
- Pour les méthodes de classification avec une contrainte spatiale :
  - `rgeoda` pour les algorithmes AZP, SKATER et REDCAP.
  - `spdep` pour l'algorithme SKATER.
- Pour les méthodes de classification avec une dimension spatiale :
  - `ClustGeo` pour la méthode ClustGeo.
  - `geocmeans` pour la classification k-moyennes floue et spatiale.

8.1 Méthodes de classification non supervisée avec contrainte spatiale

Nous avons vu que l'objectif d'une méthode non supervisée appliquée à des données spatiales est de regrouper en *n* classes les unités spatiales d'une couche géographique. Prenons l'exemple de quatre variables environnementales cartographiées à la figure 8.1 pour les IRIS de la ville de Lyon, dont trois considérées comme des nuisances (bruit, dioxyde d'azote et particules fines) et une considérée comme avantageuse (végétation).

Il est possible de regrouper les unités spatiales avec ou sans contrainte spatiale :

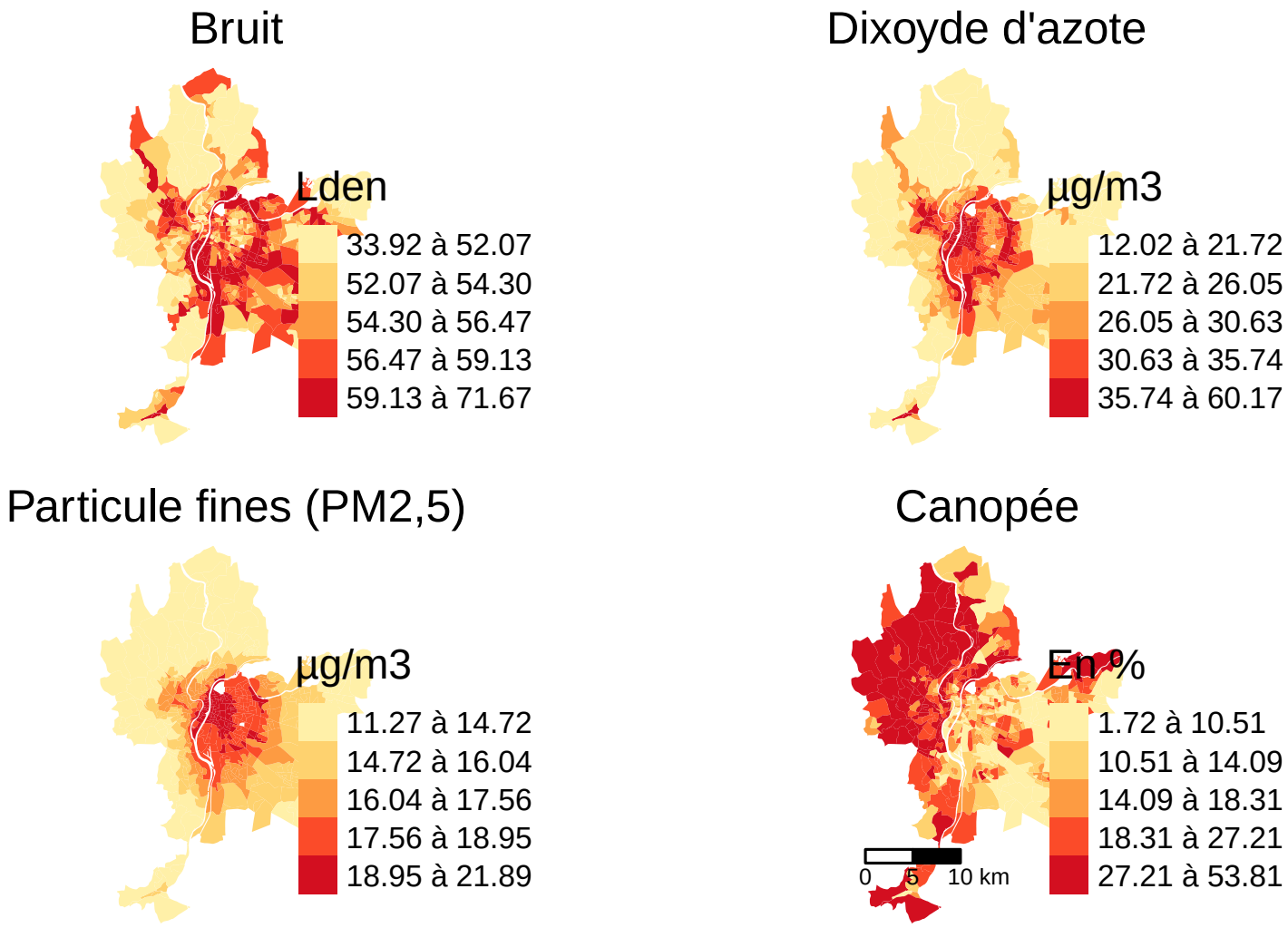


FIGURE 8.1 – Cartographie des variables environnementales

— **Sans contrainte spatiale**, nous cherchons à regrouper les IRIS (unités spatiales) avec des valeurs similaires pour les quatre variables retenues (L_{den}, NO_2, $PM_{2,5}$ et pourcentage de canopée). Cette approche est illustrée à la figure 8.2 (a) avec l'algorithme k-moyennes (*k-means* en anglais) avec cinq classes.
— **Avec contrainte spatiale**, nous cherchons à regrouper les IRIS (unités spatiales) avec des valeurs similaires pour les quatre variables retenues, tout en nous assurant que les regroupements forment des régions avec une absence de mitage. Cette approche est illustrée à la figure 8.2 (b) avec l'algorithme SKATER (*Spatial 'K'luster Analysis by Tree Edge Removal*) avec cinq classes. Autrement dit, l'objectif des méthodes de classification non supervisée avec contrainte spatiale est d'agréger n unités spatiales en m régions non discontinues (avec $n < m$) et cohérentes du point de vue de leurs attributs (Openshaw et Rao 1995, 428).

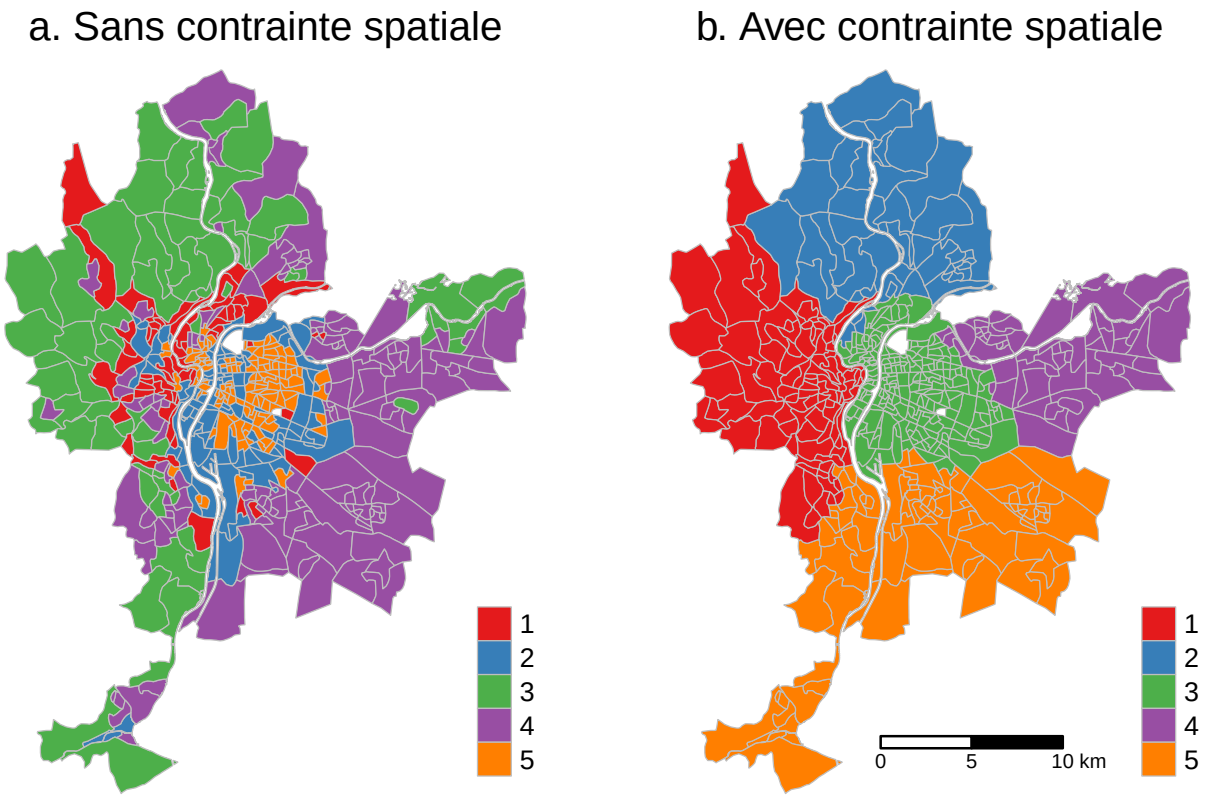


Figure 8.2 – Classification non supervisée avec et sans contrainte spatiale

Objectif

Intérêt et limites des méthodes de classification avec une contrainte spatiale
Selon Gelb et Apparicio (2021, 7), le résultat d'une méthode de classification avec une contrainte spatiale est « la création de régions très cohérentes spatialement, c'est-à-dire avec une absence de mitage. Autrement dit, avec ces méthodes, il n'est pas possible d'identifier de groupes qui seraient spatialement discontinus, c'est-à-dire composés de plusieurs ensembles régionaux séparés. L'impossibilité d'obtenir du mitage au sein des différentes régions peut masquer la présence de valeurs fortement dissemblables localement, malgré la prise en compte de l'espace. Or, ces observations systématiquement différentes de leurs voisines doivent faire l'objet d'une attention particulière dans les exercices de classification intégrant l'espace, ce que ne permettent pas ces méthodes d'agrégation spatiale.
Les limites de ces méthodes, particulièrement celles relatives au mitage, ont conduit plus récemment à la mise au point de nouvelles méthodes incluant l'espace dans le processus de classification, sans imposer une contrainte de contiguïté. Plus spécifiquement, ces nouvelles méthodes sont des modifications des algorithmes classiques, tels que la CAH ou le FCM, pour intégrer la dimension spatiale en parallèle à la dimension sémantique des données. En d'autres termes, l'espace n'est plus intégré comme une contrainte dans les algorithmes de classification, mais plutôt comme une donnée supplémentaire ».

Les principaux algorithmes de classification non supervisée avec contrainte spatiale (*Spatially Constrained Clustering Methods* en anglais) sont :

- La méthode de zonage automatique (*Automatic Zoning Procedure* en anglais) (AZP) proposée par Openshaw (1977), puis améliorée par Openshaw et Rao (1995).
- L'algorithme SKATER (*Spatial 'K'luster Analysis by Tree Edge Removal*) (Assunção et al. 2006).
- L'algorithme REDCAP (*Regionalization with dynamically constrained agglomerative clustering and partitioning*) (Guo 2008).
- L'algorithme du *max-p-regions problem* (Duque, Anselin et Rey 2012).

Pour mettre en œuvre ces différents algorithmes, nous utilisons le *package* `rgeoda` (Li et Anselin 2023). Notez que l'algorithme SKATER est aussi implémenté dans le *package* `spded` (fonction `skater`).

8.1.1 Algorithmes AZP

L'algorithme AZP (*Automatic Zoning Problem*) est une approche itérative et heuristique visant à regrouper des polygones adjacents en m régions, tout en maximisant la variance interrégionale (variance interclasse) et en minimisant la variance intrarégionale (variance intraclasse) calculées sur les p variables. Autrement dit, il vise à créer des régions non discontinues les plus homogènes possibles et les plus dissemblables entre elles sur la base des p variables. Pour utiliser l'AZP, il faut spécifier le nombre de régions (m) désiré. Notez qu'il existe trois algorithmes pour l'AZP :

1. AZP (*Automatic Zoning Procedure*), soit la première version par Stan Openshaw (1977).
2. AZP-SA (*A simulated annealing AZP method*) (Openshaw et Rao 1995).
3. AZP-TABU (*A tabu search heuristic version of AZP*) (Openshaw et Rao 1995).

Pour une description détaillée de ces trois algorithmes, vous pouvez consulter Openshaw et Rao (1995) ou encore le lien suivant.

Appliquons ces algorithmes aux 506 IRIS de la ville de Lyon avec les quatre variables environnementales préalablement centrées réduites (bruit, dioxyde d'azote, particules fines et pourcentage de végétation) et une matrice de contiguïté selon le partage d'un nœud. Le *package* `rgeoda` comprend trois fonctions pour l'AZP : `azp_greedy` (AZP), `azp_sa` (AZP-SA),

`azp_tabu` (AZP-TABU). Pour l'exercice, nous fixons le nombre de régions à 5. Notez que par défaut, les variables seront centrées réduites (moyenne = 0 et écart-type = 1) avec le paramètre `scale_method="standardize"`.

```
library(rgeoda)
library(sf)
library(tmap)
## Variables
VarsEnv <- c("Lden", "NO2", "PM25", "VegHautPrt")
## Dataframe sans la géométrie et les quatre variables
load("data/chap08/DonneesLyon.Rdata")
Data <- st_drop_geometry(LyonIris[VarsEnv])
## Création d'une matrice de contiguïté avec rgeoda
queen_w <- queen_weights(LyonIris)
## Calcul des trois algorithmes
azp <- rgeoda::azp_greedy(p=5,       # Nombre de régions
                          w=queen_w, # Matrice contiguïté
                          df=Data,   # Tableau de données
                          scale_method = "standardize") # cote z
azp.sa <- rgeoda::azp_sa(p=5, w=queen_w, df=Data, cooling_rate = 0.85)
azp.tab <- rgeoda::azp_tabu(p=5, w=queen_w, df=Data, tabu_length = 10, conv_tabu = 10)
## Création des trois champs dans la couche de Lyon
LyonIris$Azp <- as.character(azp$Clusters)
LyonIris$Azp_sa <- as.character(azp.sa$Clusters)
LyonIris$Azp_tab <- as.character(azp.tab$Clusters)
```

Cartographions les résultats des trois algorithmes AZP (figure 8.3).

```
## Cartographie des résultats
Carte.AZP1 <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
              tm_fill(col="Azp", palette = "Set1", title ="")+
              tm_layout(frame=FALSE,
                main.title = "a. AZP",
                main.title.position = "center",
                main.title.size = 1)
Carte.AZP2 <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
              tm_fill(col="Azp_sa", palette = "Set1", title ="")+
              tm_layout(frame=FALSE,
                main.title = "b. AZP Simulated Annealing",
                main.title.position = "center",
                main.title.size = 1)
Carte.AZP3 <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
              tm_fill(col="Azp_tab", palette = "Set1", title ="")+
              tm_layout(frame=FALSE,
                main.title = "c. AZP Tabu Search",
                main.title.position = "center",
                main.title.size = 1)
tmap_arrange(Carte.AZP1, Carte.AZP2, Carte.AZP3, ncol = 2, nrow = 2)
```

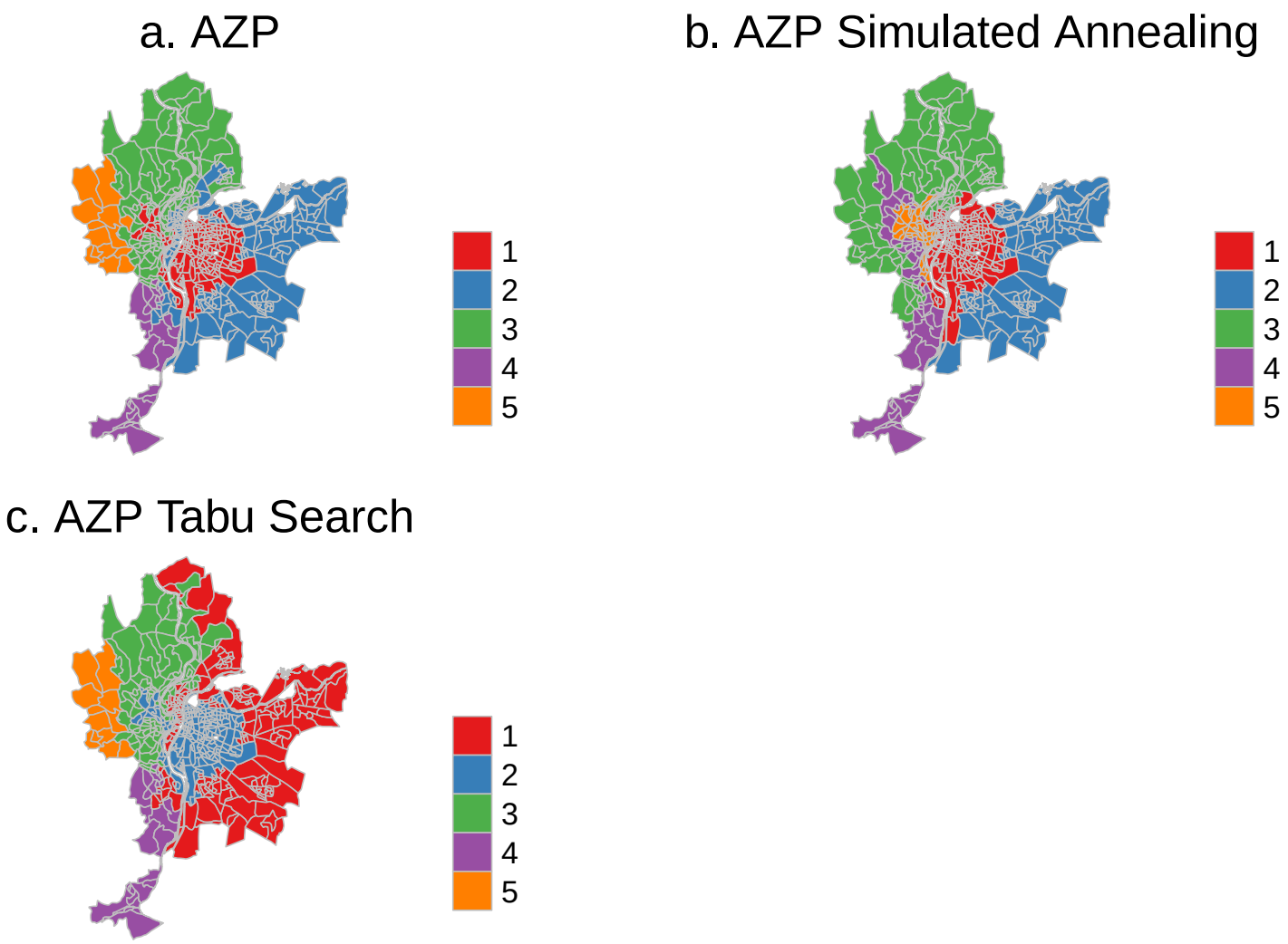


FIGURE 8.3 – Regroupements des 505 IRIS en cinq régions selon les trois algorithmes AZP

Par la suite, nous comparons les résultats obtenus des trois algorithmes en reportant :

1. Les variances totale, intrarégionale et interrégionale, et surtout le ratio entre les variances intergroupe et totale. Ce ratio varie de 0 à 1 et exprime la proportion de la variance des variables qui est expliquée par les différentes régions obtenues; plus il est élevé, meilleur est le résultat. Par conséquent, il peut être utilisé pour identifier la solution optimale entre les trois algorithmes.
2. Le nombre d'observations par région.
3. Les valeurs moyennes des variables centrées réduites par région.

```
## Calcul du ratio entre les variances intergroupe et totale
cat("Ratio des variances interrégionale et totale",
    "\nAZP : ", round(azp$`The ratio of between to total sum of squares`, 3),
    "\nAZP-SA : ", round(azp.sa$`The ratio of between to total sum of squares`, 3),
    "\nAZP-TABU : ", round(azp.tab$`The ratio of between to total sum of squares`, 3)
)
```

```
Ratio des variances interrégionale et totale
AZP :  0.436
AZP-SA :  0.518
AZP-TABU :  0.428
```

À la lecture des valeurs du ratio entre la variance interrégionale et la variance totale ci-dessus, la plus élevée est obtenue pour l'AZP-SA (0,518), suivie de celles de l'AZP (0,436) et de l'AZP-TABU (0,428). Nous retenons alors l'AZP-SA.

```
## Nombre d'observations par région
table(LyonIris$Azp)
```

```
  1   2   3   4   5
186 176 104  27  13
```

```
table(LyonIris$Azp_sa)
```

```
  1   2   3   4   5
221 107  87  51  40
```

```
table(LyonIris$Azp_tab)
```

```
  1   2   3   4   5
191 186  91  27  11
```

```
## Valeurs moyennes des variables centrées réduites par région
Data$Azp <- azp$Clusters
Data$Azp_sa <- azp.sa$Clusters
Data$Azp_tab <- azp.tab$Clusters
aggregate(cbind(Lden,NO2,PM25,VegHautPrt) ~ Azp, data = Data, FUN = mean)
```

```
  Azp     Lden      NO2     PM25 VegHautPrt
1   1 58.05464 35.21961 18.85212   15.45333
2   2 55.42159 26.45169 16.32884   13.98563
3   3 53.23122 23.63965 14.95293   30.42529
4   4 52.26742 22.99701 14.58065   20.39444
5   5 48.71652 18.24301 13.07043   33.07154
```

```
aggregate(cbind(Lden,NO2,PM25,VegHautPrt) ~ Azp_sa, data = Data, FUN = mean)
```

```
  Azp_sa     Lden      NO2     PM25 VegHautPrt
1      1 57.41952 35.02151 18.80132   14.41317
2      2 55.11019 22.52804 15.55562   14.12178
3      3 51.08297 20.48809 14.16951   28.66000
4      4 54.23625 24.47389 14.96836   22.30059
5      5 58.40430 33.55155 17.08480   28.83775
```

```
aggregate(cbind(Lden,NO2,PM25,VegHautPrt) ~ Azp_tab, data = Data, FUN = mean)
```

```
  Azp_tab     Lden      NO2     PM25 VegHautPrt
1       1 55.24285 26.04447 16.19300   14.67712
2       2 58.05464 35.21961 18.85212   15.45333
3       3 53.19298 23.97221 14.99671   31.90967
4       4 52.26742 22.99701 14.58065   20.39444
5       5 48.32861 17.74683 12.84853   31.68364
```

Les résultats finaux de l'AZP-SA sont présentés au tableau 8.1 et à la figure 8.4. L'analyse conjointe du tableau et de la carte permet ainsi d'interpréter chacune des classes. En guise d'exemple, nous pouvons conclure que :

- **La région 1** comprend 221 IRIS localisés au centre de la ville de Lyon et caractérisés par des niveaux moyens élevés de bruit (57,4), de dioxyde d'azote (35) et de particules fines (18,8) élevés et un faible pourcentage de canopée (14,4 %).
- Par contre, **la région 2** comprend 107 IRIS localisés à l'extrême ouest de la ville et caractérisés par les plus faibles niveaux de polluants (51,1, 20,5 et 14,2) et une forte moyenne pour la canopée (28,7 %).

Tableau 8.1 – Valeurs moyennes des variables pour les cinq régions obtenues par l'AZP-SA

Région	Lden	NO2	PM25	Végétation	Nombre d'IRIS
1	57,4	35,0	18,8	14,4	221
2	55,1	22,5	15,6	14,1	107
3	51,1	20,5	14,2	28,7	87
4	54,2	24,5	15,0	22,3	51
5	58,4	33,6	17,1	28,8	40

Nous avons vu que pour les algorithmes AZP, il faut spécifier le nombre de régions. Nous l'avons fixé arbitrairement à 5. Comme pour n'importe quelle méthode de classification non supervisée, déterminer le nombre de classes optimal est une étape cruciale qui peut s'appuyer sur différentes techniques, dont la méthode du coude basée sur l'inertie expliquée (ici le ratio entre les variances interrégionale et totale), l'indicateur de silhouette et la méthode GAP. Pour une description détaillée de ces méthodes, consultez la section suivante (Apparicio et Gelb 2022). Le code ci-dessous permet de réaliser un graphique avec les valeurs du ratio (inertie expliquée) obtenues avec l'algorithme AZP-TABU calculé pour 2 à 10 régions. À la lecture de la figure 8.5, nous observons deux ruptures (coudes) très nettes à 5 et 8.

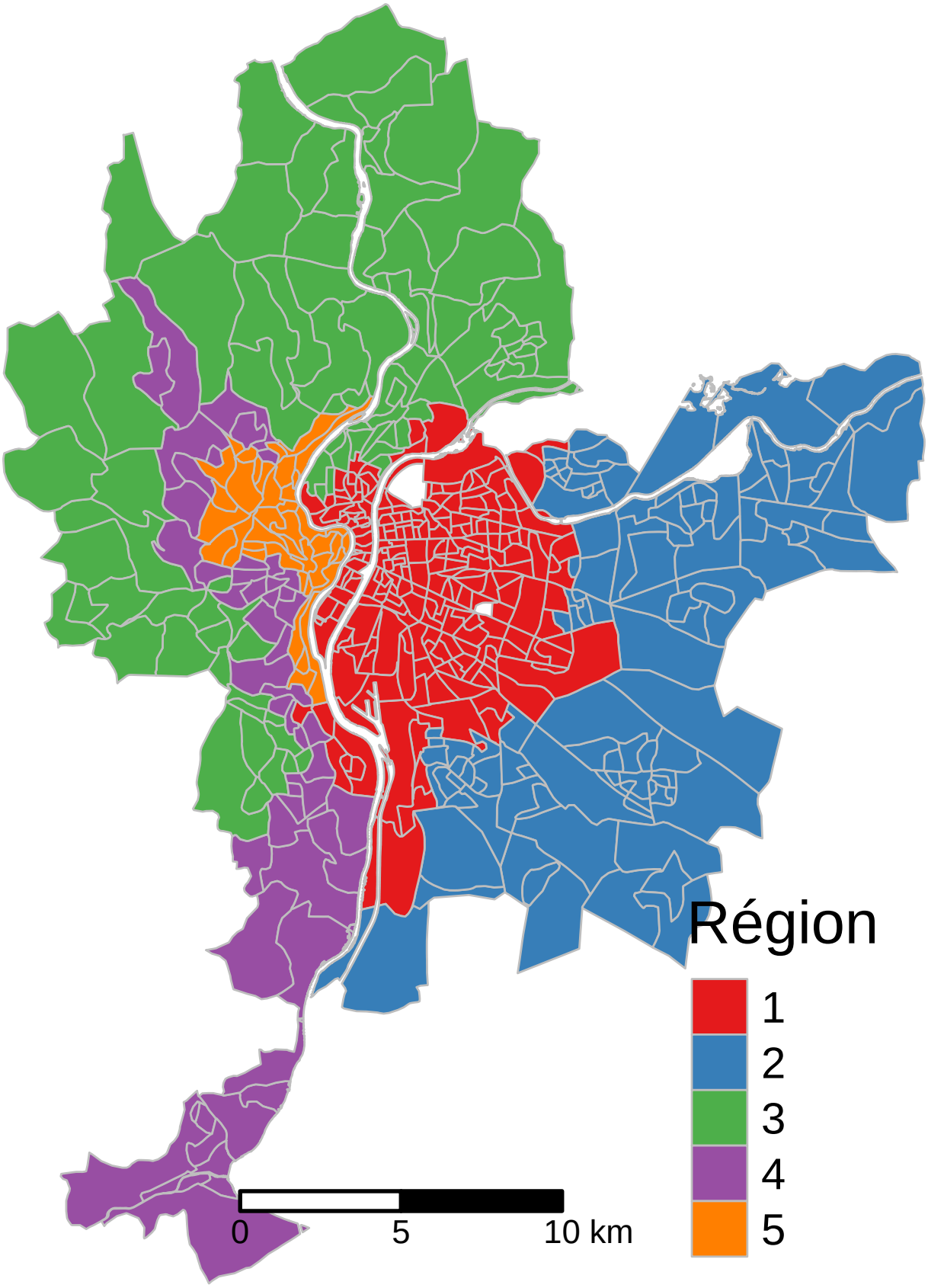


Figure 8.4 – Regroupement des IRIS en cinq régions selon l'AZP-TABU

```r
library(ggplot2)
nregions <- 2:10
Data <- data.frame(scale(st_drop_geometry(LyonIris)[VarsEnv]))
queen_w <- queen_weights(LyonIris)
inertie <- sapply(nregions, function(k){
  # calcul de l'AZP-TABU avec k
  resultat <- azp_tabu(p=k, w=queen_w, df=Data, tabu_length = 10, conv_tabu = 10)
  # récupération du ratio
  ratios <- resultat$`The ratio of between to total sum of squares`
  return(ratios)
})

df <- data.frame(k = nregions, ratio = inertie)
ggplot(df) +
  geom_line(aes(x = k, y = ratio)) +
  geom_point(aes(x = k, y = ratio), color = "red") +
  labs(x = "Nombre de régions", y = "Inertie expliquée (%)")
```

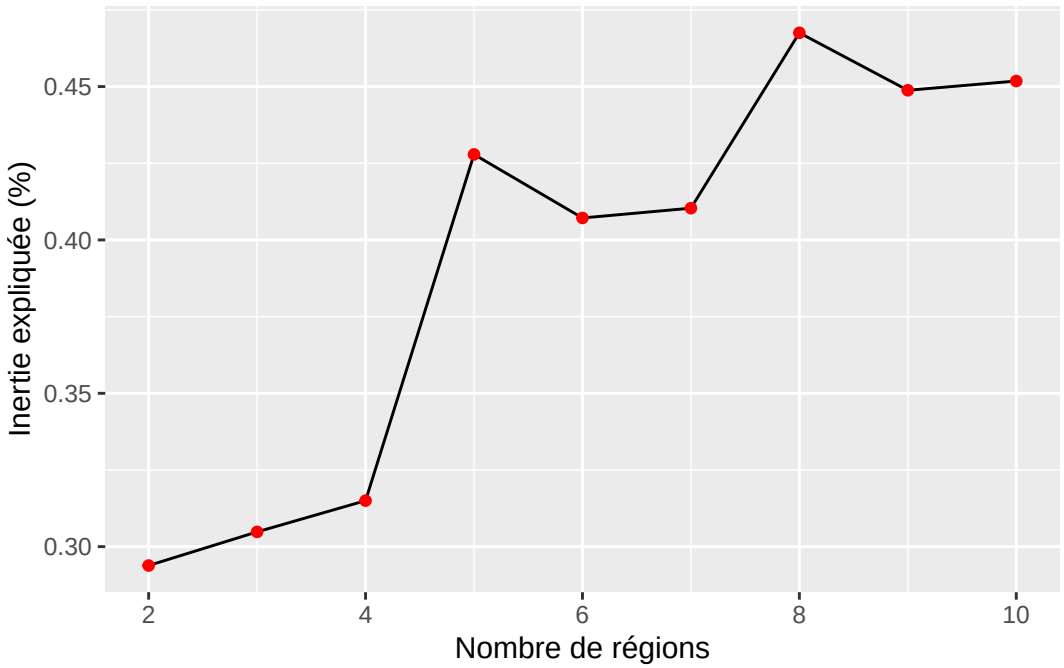


FIGURE 8.5 – Méthode du coude reposant sur l'inertie expliquée pour l'AZP-TABU

8.1.2 Algorithme SKATER

L'algorithme SKATER (*Spatial 'K'luster Analysis by Tree Edge Removal*) (Assunção et al. 2006) permet aussi de créer des régions sans discontinuité, en recourant à une technique de la théorie des graphes, soit celle de l'arbre couvrant de poids minimal (*minimum spanning tree*). Succinctement, la classification est obtenue avec les étapes suivantes :

1. Création d'un graphe de connectivité pour les polygones de la couche géographique. Dans ce graphe, les nœuds sont les centroïdes des polygones et les arêtes représentent les liaisons entre deux entités spatiales voisines.
2. Pour chaque arête, nous calculons la dissimilarité (appelée coût) des deux polygones voisins en fonction des p variables.
3. Pour chaque polygone, nous retenons l'arête avec le coût minimal. Autrement dit, pour chaque polygone, nous retenons son polygone voisin qui lui est le plus semblable selon les p variables. Nous obtenons ainsi l'arbre couvrant de poids minimal.
4. Cet arbre est ensuite élagué en supprimant les arêtes avec les plus forts coûts et en créant ainsi des sous-graphes en m régions sans discontinuité.

Pour une description plus détaillée de l'algorithme, consultez l'article d'Assunção *et al.* (2006).

Le code ci-dessous permet de centrer et de réduire les quatre variables (fonction `scale`) et de construire la matrice de voisinage entre les polygones de la couche `LyonIris` (fonction `poly2nb` de `spdep`).

```
library(spdep)
library(tmap)
## Variables
VarsEnv <- c("Lden", "NO2", "PM25", "VegHautPrt")
## Dataframe sans la géométrie et les quatre variables
load("data/chap08/DonneesLyon.Rdata")
Data <- st_drop_geometry(LyonIris[VarsEnv])
## Données centrées et réduites
LyonIrisZscore <- data.frame(scale(Data))
## Matrice voisinage
Lyon.nb <- poly2nb(LyonIris)
```

Calculons les coûts pour les arêtes reliant les nœuds avec la fonction `nbcosts`. Nous constatons que le polygone 1 est voisin des polygones 27, 26, 44 et 74 avec des coûts de 1,34, 1,74, 1,15 et 16,3. Par conséquent, parmi ses quatre voisins, le polygone 1 est le plus semblable au polygone 44 (coût minimal).

```
## Calcul des coûts pour les arêtes
lcosts <- nbcosts(Lyon.nb, LyonIrisZscore)
head(Lyon.nb, n=1)
```

```
[[1]]
[1] 27 36 44 73
```

```
head(lcosts, n=1)
```

```
[[1]]
[1] 1.343210 1.735894 1.153787 1.632583
```

À partir de ces coûts, nous pouvons trouver l'arbre couvrant de poids minimal (*minimum spanning tree*), objet dénommé ici `Lyon.mst` qui comprend trois colonnes :

— La première pour l'identifiant du polygone.
— La seconde pour l'identifiant du polygone voisin.
— La troisième pour la valeur du coût minimal (similarité selon les variables retenues).

```
## Matrice de pondération spatiale avec les coûts
Lyon.w <- nb2listw(Lyon.nb, lcosts, style="B")
### Trouver l'arbre couvrant de poids minimal
Lyon.mst <- mstree(Lyon.w)
head(Lyon.mst, n=3)
```

```
     [,1] [,2]      [,3]
[1,]  499  478 0.5025378
[2,]  478   46 0.4618205
[3,]   46    7 1.3665949
```

Le code ci-dessous permet de visualiser le graphe de connectivité et l'arbre couvrant de poids minimal (figure 8.6).

```
## Visualisation du graphe de connectivité
coords <- st_coordinates(st_centroid(LyonIris))
plot(st_geometry(LyonIris), border="gray", lwd=.5, col="wheat")
plot(Lyon.nb, coords, add=TRUE, col="red", lwd=1)
## Visualisation de l'arbre couvrant de poids minimal
plot(st_geometry(LyonIris), border="gray", lwd=.5, col="wheat")
plot(Lyon.mst, coords, col="blue", cex.lab=0.7, add=TRUE)
```

Le code ci-dessous permet de réaliser une classification SKATER avec cinq régions avec le *package* `spdep`.

```
## SKATER avec le package spdep
set.seed(123456789)
skater5.spdep <- spdep::skater(edges = Lyon.mst[,1:2], # premières colonnes de l'arbre
                               data = data.frame(LyonIrisZscore),
                               method = "euclidean",
                               ncuts = 4)  # k-1 régions
table(skater5.spdep$groups)
```

```
  1   2   3   4   5
 48 215  89  57  97
```

Toutefois, il est plus simple d'utiliser la fonction `skater` de `rgeoda` qui ne nécessite pas de créer au préalable l'arbre couvrant de poids minimal.

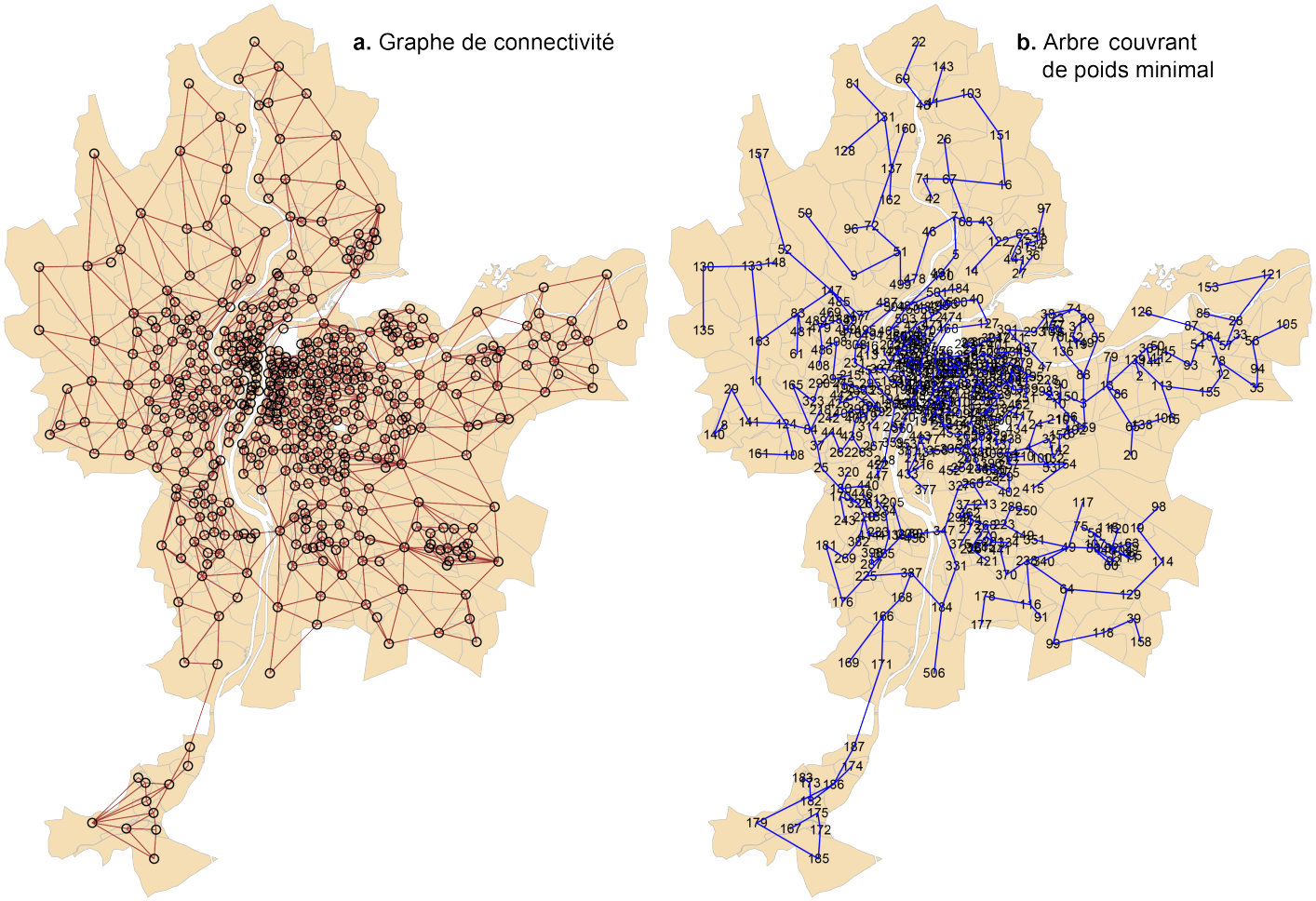


FIGURE 8.6 – Graphe de connectivité et arbre couvrant de poids minimal

```
## SKATER avec le package rgeoda
library(rgeoda)
Data <- st_drop_geometry(LyonIris[VarsEnv])
queen_w <- queen_weights(LyonIris)
skater5.rgeoda <- rgeoda::skater(k = 5,        # k-1 régions
                                 w = queen_w,  # matrice de contiguïté
                                 scale_method = "standardize",
                                 df = Data)    # dataframe
table(skater5.rgeoda$Clusters)
```

```
  1   2   3   4   5
248 125  51  48  34
```

La figure 8.7 démontre que les résultats obtenus sont légèrement différents avec les deux *packages*.

```
LyonIris$skater5spdep <- as.character(skater5.spdep$groups)
LyonIris$skater5rgeoda <- as.character(skater5.rgeoda$Clusters)

Carte.SkaterA <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
              tm_fill(col="skater5spdep", palette = "Set1", title ="")+
              tm_layout(frame=FALSE,
                main.title = "a. SKATER spdep",
                main.title.position = "center",
                main.title.size = 1)
Carte.SkaterB <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
              tm_fill(col="skater5rgeoda", palette = "Set1", title ="")+
              tm_layout(frame=FALSE,
                main.title = "b. SKATER rgeoda",
                main.title.position = "center",
                main.title.size = 1)
tmap_arrange(Carte.SkaterA, Carte.SkaterB)
```

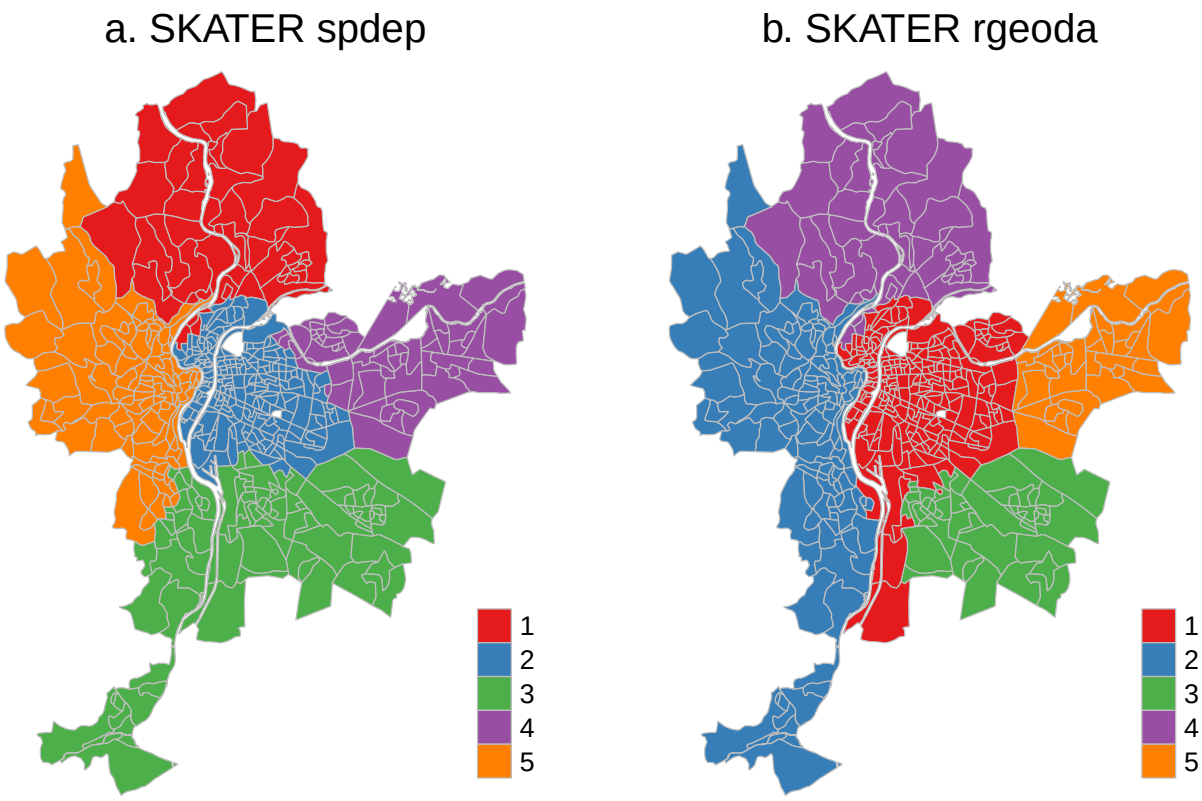


FIGURE 8.7 – Résultats de l'algorithme SKATER avec cinq classes obtenus avec les *packages* `spdep` et `rgeoda`

Aller plus loin

Algorithme SKATER avec un seuil minimal pour les classes

Dans une classification non supervisée avec une contrainte spatiale, il est possible de fixer un seuil minimal pour chaque région à partir d'une variable. L'exemple le plus classique est l'obtention de p régions qui doivent au moins avoir un nombre d'habitants fixé par la personne utilisatrice. Pour ce faire, nous utilisons deux paramètres de la fonction `spdep::skater`, soit `crit = 50000` pour fixer le seuil et `vec.crit = df$Population` pour indiquer le vecteur sur lequel est calculé le critère.

```
clus10_min <- spdep::skater(edges = ct_mst[,1:2],
                      # dataframe avec les variables centrées réduites
                      data = dfs,
                      # seuil fixé
                      crit = 50000,
                      # variable population du dataframe
                      vec.crit = df$Population,
                      ncuts = 4)
```

Fonction `skater` : différences entre les *packages* `rgeoda` et `spdep`

La fonction `skater` de `rgeoda` a deux principaux avantages :

1. Comme décrit précédemment, l'avantage de la fonction `skater` de `rgeoda` est qu'elle ne nécessite pas de calculer au préalable l'arbre couvrant de poids minimal.
2. scale_method = c("raw", "standardize", "demean", "mad", "range_standardize", "range_adjust") permet de transformer directement les variables. La méthode par défaut est la cote z (moyenne = 0 et écart-type = 1).

Avec la fonction `skater` de `spdep`, vous devez préalablement transformer vos variables et construire l'arbre couvrant de poids minimal. Par contre, elle intègre de nombreux types de distance pour évaluer la dissimilarité entre les unités spatiales avec le paramètre `method = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski", "mahalanobis")` tandis que le paramètre `distance_method = c("euclidean", "manhattan")` de `rgeoda` ne comprend que deux types de distance.

8.1.3 Algorithmes REDCAP

Les différentes versions de l'algorithme REDCAP (*Regionalization with dynamically constrained agglomerative clustering and partitioning*) proposé par Diansheng Guo (2008) sont aussi basées sur la construction d'un arbre (*spanning tree*) dont l'élagage est obtenu de cinq différentes façons :

— Premier ordre et saut minimal (*First-order and Single-linkage*) qui fournit un résultat identique à l'algorithme SKATER.
— Ordre complet et saut maximal (*Full-order and Complete-linkage*).
— Ordre complet et saut moyen (*Full-order and Average-linkage*).
— Ordre complet et saut minimal (*Full-order and Single-linkage*).
— Ordre complet et critère de Ward (*Full-order and Ward-linkage*).

Le code ci-dessous permet de calculer les cinq versions de l'algorithmes REDCAP avec cinq régions et de comparer leurs résultats à partir du ratio (entre les variances interrégionale et totale) et du nombre d'observations par région. Ce ratio

représente la part de la variance intra-groupe dans la variance totale des données. Plus ce ratio est grand, meilleure est la classification car elle résume une plus grande partie de la variance totale.

```
library(rgeoda)
library(sf)
## Préparation des données
Data <- st_drop_geometry(LyonIris[VarsEnv])
queen_w <- queen_weights(LyonIris)
## Algorithmes REDCAP
redcap5.A <- redcap(k = 5, w = queen_w, scale_method = "standardize", df = Data,
                    method = "firstorder-singlelinkage")
redcap5.B <- redcap(k = 5, w = queen_w, scale_method = "standardize", df = Data,
                    method = "fullorder-completelinkage")
redcap5.C <- redcap(k = 5, w = queen_w, scale_method = "standardize", df = Data,
                    method = "fullorder-averagelinkage")
redcap5.D <- redcap(k = 5, w = queen_w, scale_method = "standardize", df = Data,
                    method = "fullorder-singlelinkage")
redcap5.E <- redcap(k = 5, w = queen_w, scale_method = "standardize", df = Data,
                    method = "fullorder-wardlinkage")
## Comparaison des résultats
Ratios <- data.frame(Methode = c("firstorder-singlelinkage",
                                 "fullorder-completelinkage",
                                 "fullorder-averagelinkage",
                                 "fullorder-singlelinkage",
                                 "fullorder-wardlinkage"),
                     ratio = c(redcap5.A$`The ratio of between to total sum of squares`,
                               redcap5.B$`The ratio of between to total sum of squares`,
                               redcap5.C$`The ratio of between to total sum of squares`,
                               redcap5.D$`The ratio of between to total sum of squares`,
                               redcap5.E$`The ratio of between to total sum of squares`)
                     )
Nobs <- data.frame(rbind(table(redcap5.A$Clusters),
              table(redcap5.B$Clusters),
              table(redcap5.C$Clusters),
              table(redcap5.D$Clusters),
              table(redcap5.E$Clusters))
              )
names(Nobs) <- c("C1", "C2", "C3", "C4", "C5")
Ratios <- cbind(Ratios, Nobs)
Ratios
```

```
                   Methode     ratio  C1  C2  C3 C4 C5
1  firstorder-singlelinkage 0.4081577 248 125  51 48 34
2 fullorder-completelinkage 0.4728026 173 156 115 47 15
3  fullorder-averagelinkage 0.4993184 148 141 106 63 48
4   fullorder-singlelinkage 0.3976055 227 102  73 53 51
5     fullorder-wardlinkage 0.5185455 166 134 120 51 35
```

À la lecture des valeurs du ratio ci-dessus, la meilleure classification serait celle obtenue avec un ordre complet et le critère de Ward. Cartographions les résultats des quatre versions de l'algorithme RECAP avec un ordre complet (figure 8.8).

```
## Ajout des champs dans la couche
LyonIris$RC5.FOcompletelinkage  <- as.character(redcap5.B$Clusters)
LyonIris$RC5.FOaveragelinkage   <- as.character(redcap5.C$Clusters)
LyonIris$RC5.FOsinglelinkage    <- as.character(redcap5.D$Clusters)
LyonIris$RC5.FOwardlinkage      <- as.character(redcap5.E$Clusters)
## Cartographie des résultats
Carte.RCb <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
              tm_fill(col="RC5.FOcompletelinkage", palette = "Set1", title ="")+
              tm_layout(frame=FALSE,
                main.title = "a. Saut maximal",
                main.title.position = "center",
                main.title.size = 1)
Carte.RCc <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
              tm_fill(col="RC5.FOaveragelinkage", palette = "Set1", title ="")+
              tm_layout(frame=FALSE,
                main.title = "b. Saut moyen",
                main.title.position = "center",
                main.title.size = 1)
Carte.RCd <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
              tm_fill(col="RC5.FOsinglelinkage", palette = "Set1", title ="")+
              tm_layout(frame=FALSE,
                main.title = "c. Saut minimal",
                main.title.position = "center",
                main.title.size = 1)
Carte.RCe <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
              tm_fill(col="RC5.FOwardlinkage", palette = "Set1", title ="")+
              tm_layout(frame=FALSE,
                main.title = "d. Critère de Ward",
                main.title.position = "center",
                main.title.size = 1)
tmap_arrange(Carte.RCb, Carte.RCc, Carte.RCd, Carte.RCe, ncol = 2, nrow = 2)
```

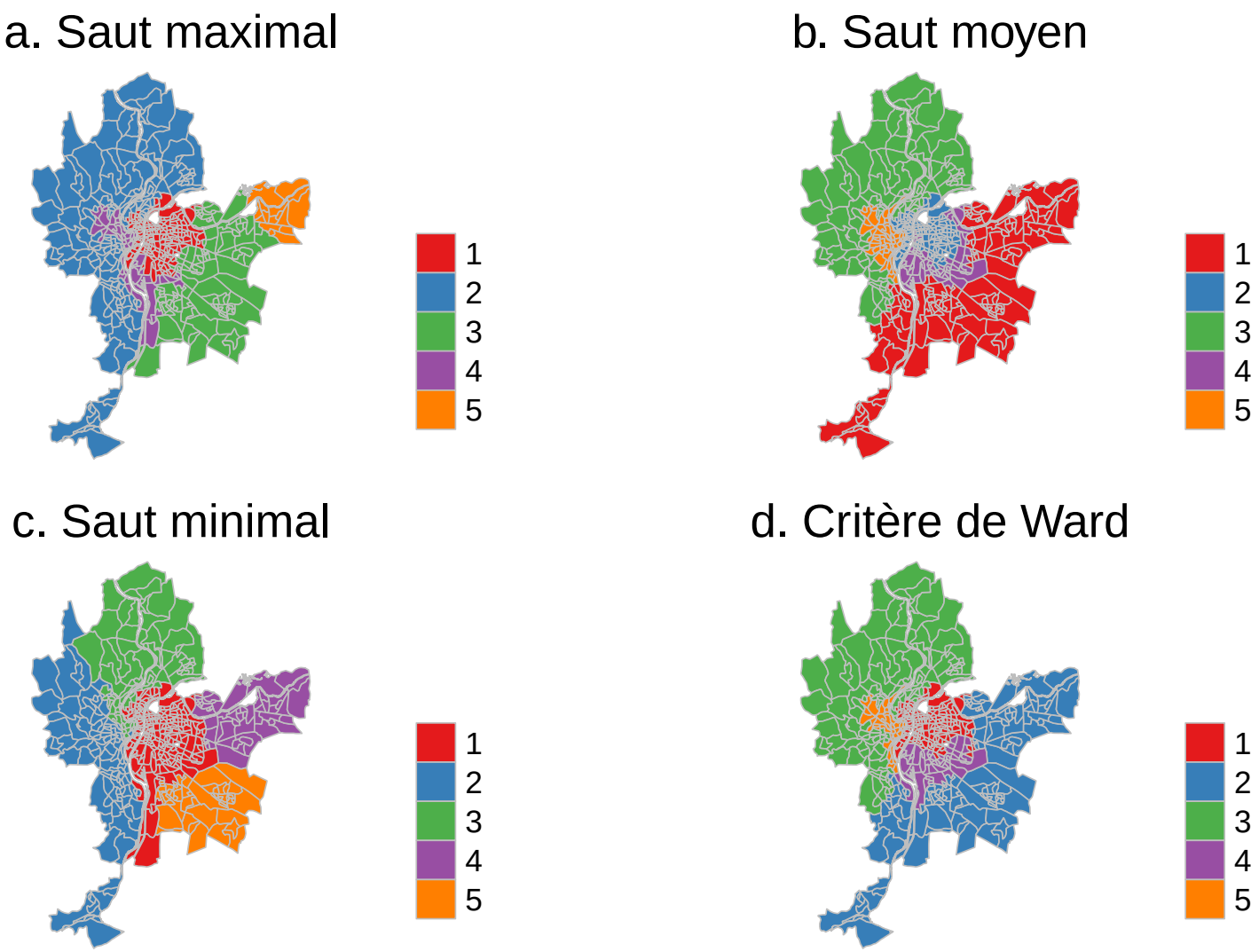


FIGURE 8.8 – Regroupements des 505 IRIS en cinq régions selon les quatre versions de l'algorithme REDCAP avec un lien complet

8.1.4 Algorithme du max-p-regions problem

Cet algorithme, proposé par Duque *et al.* (2012), n'est pas décrit ici. Notez qu'il peut être calculé avec trois fonctions du *package* `rgeoda`, soit `maxp_greedy`, `maxp_sa` et `maxp_tabu`.

8.2 Méthodes de classification non supervisée avec une dimension spatiale

Nous avons vu que les méthodes de classification avec une contrainte spatiale visent à obtenir des régions non discontinues, c'est-à-dire sans mitage spatial. L'objectif des méthodes de classification non supervisée avec une dimension spatiale est quelque peu différent : classifier les observations en tenant compte de l'espace (proximité, voisinage entre les unités spatiales) afin de limiter les effets de mitage, sans toutefois l'interdire.

Dans le cadre de cette section, nous décrivons deux de ces méthodes qui intègrent la dimension spatiale de manière différente :

1. **La méthode *ClustGeo***, qui est une extension de la classification ascendante hiérarchique, est une méthode de classification non supervisée, spatiale et stricte. Cette méthode repose sur deux matrices de dissimilarité : une **matrice des distances sémantiques (attributaires)** calculée sur les valeurs de plusieurs variables caractérisant les observations et une **matrice de distances** (euclidienne le plus souvent) entre les entités géographiques. Nous cherchons ainsi à regrouper les observations qui se ressemblent à la fois selon leurs attributs et selon leur proximité spatiale.
2. **La méthode k-moyennes spatiale et floue (*Spatial fuzzy c-means*)**, qui est une extension de la méthode k-moyennes, est une méthode de classification non supervisée, spatiale et floue. Cette méthode repose sur deux matrices de dissimilarité : une **matrice sémantique** calculée sur les valeurs de plusieurs variables caractérisant les entités géographiques et une **matrice sémantique spatialement décalée**. Nous cherchons ainsi à regrouper les observations qui se ressemblent à la fois selon leurs caractéristiques et celles de leurs unités spatiales adjacentes ou proches.

Autrement dit, dans la méthode *ClustGeo*, l'espace est introduit sous la forme d'une matrice de distances entre les entités spatiales (**agencement spatial**) tandis que dans la méthode du *Spatial fuzzy c-means*, il est introduit sous la forme d'une matrice de données sémantiques spatialement décalées (**information sémantique dans l'environnement immédiat**).

8.2.1 Classification ascendante hiérarchique spatiale (ClustGeo)

8.2.1.1 Description de la méthode ClustGeo

La méthode *ClustGeo*, proposée par Marie Chavent et ses collègues (2018), est une extension de la classification ascendante hiérarchique (CAH) qui intègre la dimension spatiale des entités géographiques. Cette méthode repose sur une idée brillante, soit de classer (regrouper) les observations (unités spatiales) en combinant deux matrices de dissimilarité :

— Une matrice sémantique calculée sur p variables caractérisant les unités spatiales (D_0).
— Une matrice spatiale calculée à partir des distances spatiales (habituellement euclidienne) entre les unités spatiales (D_1). Ces deux matrices sont ensuite fusionnées en une seule matrice finale représentant la combinaison de la distance spatiale et de la dissimilarité sémantique (attributaire) entre les observations. Notez qu'un paramètre α, variant de 0 à 1, permet de définir le poids de la matrice spatiale comparativement à celui de la sémantique :

  — Avec $\alpha = 0$, le poids accordé à la matrice spatiale est nul. Nous obtenons ainsi une CAH classique puisque seules les différences attributaires sont conservées.
  — Avec $\alpha = 1$, le poids accordé à la matrice spatiale est maximal; la classification est alors purement spatiale et ignore les différences attributaires.

Par conséquent, « […] l'enjeu principal est de fixer la valeur du paramètre α, considérant qu'une augmentation de α revient à améliorer l'inertie expliquée de la matrice spatiale, au détriment d'une perte de l'inertie expliquée sur le plan sémantique » (Jérémy Gelb et Apparicio 2021, 16).

8.2.1.2 Calcul de la CAH classique

> ⚠ **Attention**
>
> **Retour sur la classification ascendante hiérarchique (CAH)**
> Pour une description détaillée de la CAH, consultez la section suivante (Apparicio et Gelb 2022).

Le code ci-dessous permet de construire l'arbre de classification selon le critère de Ward à partir de la matrice sémantique (figure 8.9).

```
## Variables pour la CAH
VarsEnv <- c("Lden", "NO2", "PM25", "VegHautPrt")
## Dataframe sans la géométrie et les quatre variables
load("data/chap08/DonneesLyon.Rdata")
Data <- st_drop_geometry(LyonIris[VarsEnv])
## Centrage (moyenne = 0) et réduction des données (variance = 1)
DataZscore <- data.frame(scale(Data))
## Matrice sémantique : dissimilarité des observations selon les variables
Matrice.Semantique <- dist(DataZscore, method = "euclidean")
# Calcul du dendrogramme avec le critère WARD
Arbre <- hclust(Matrice.Semantique, method = "ward.D")
plot(Arbre, hang = -1, label = FALSE,
     main = "Dendrogramme \n(arbre de classification selon le critère de Ward)",
     sub = "", ylab = "Hauteur", xlab = ""
     )
```

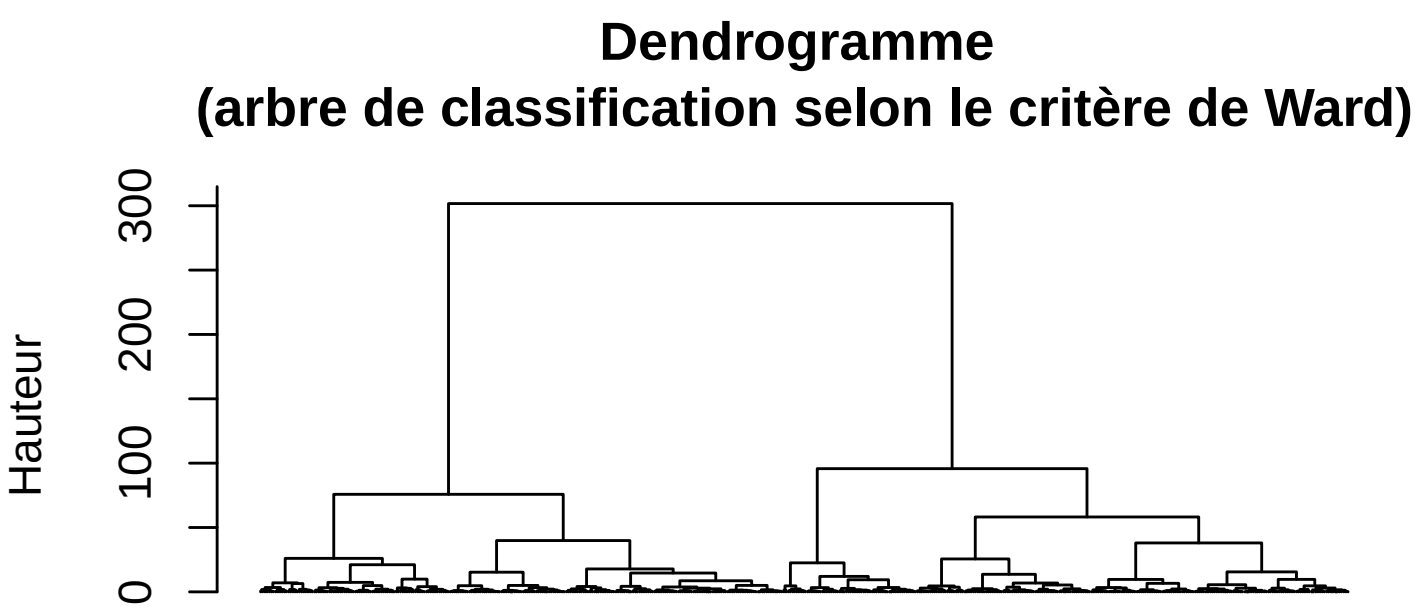


Figure 8.9 – Arbre de classification (dendrogramme)

À la lecture de la figure 8.10, nous ne détectons pas de seuils marqués dans l'inertie expliquée en fonction du nombre de groupes. Par conséquent, nous fixons arbitrairement le nombre de groupes à 5.

```
library(ggplot2)
# Fonction pour l'inertie expliquée par les classes
prop_inert_cutree <- function(K,tree,n){
  P <- cutree(tree,k=K)
  W <- sum(tree$height[1:(n-K)])
  Tot <- sum(tree$height)
  return(1-W/Tot)
}
# Inertie expliquée par des CAH de 2 à 10 classes
df.inertie <- data.frame(NGroupes = 2:10,
                         Inertie = sapply(2:10,
                                          prop_inert_cutree,
                                          tree=Arbre,
                                          n=nrow(DataZscore)))
ggplot(df.inertie)+
  geom_line(aes(x=NGroupes,y=Inertie))+
  geom_point(aes(x=NGroupes,y=Inertie), color = "red") +
  labs(y = "Inertie expliquée", x = "Nombre de groupes")
```

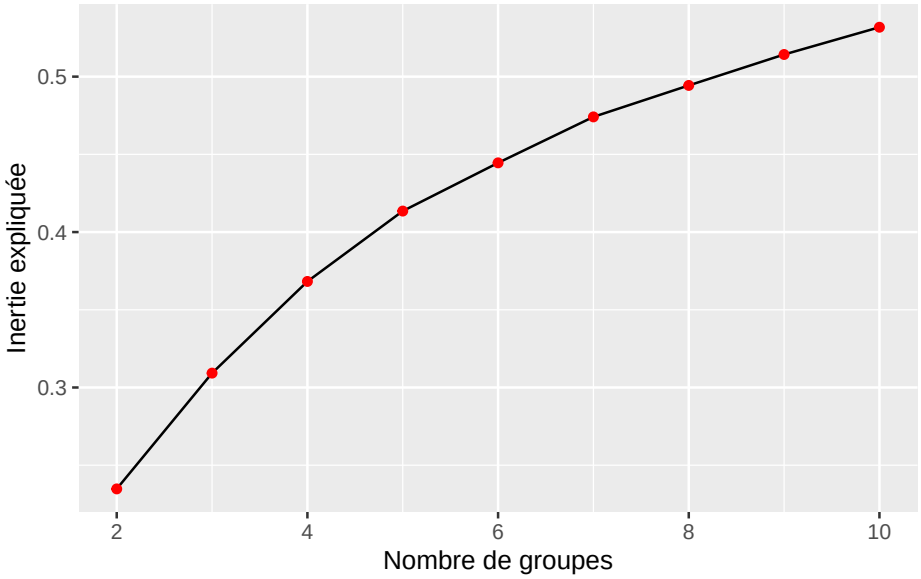


FIGURE 8.10 – Méthode du coude reposant sur l'inertie expliquée pour CAH

Nous pouvons visualiser l'arbre avec une coupure à cinq classes.

```
plot(Arbre, labels = FALSE,
     main = "Partition en 2, 5 ou 9 classes",
     xlab = "", ylab = "", sub = "", axes = FALSE, hang = -1)
rect.hclust(Arbre, 5, border = "red")
```


```
## Coupure de l'arbre à cinq classes
LyonIris$CAH5 <- as.character(cutree(Arbre, k=5))
## Nombre d'observations par classe
table(LyonIris$CAH5)
```

```
  1   2   3   4   5
 62 120  88  81 155
```

```
## Valeurs moyennes des classes
aggregate(cbind(Lden,NO2,PM25,VegHautPrt) ~ CAH5, data = st_drop_geometry(LyonIris), FUN = mean)
```

```
  CAH5     Lden      NO2     PM25 VegHautPrt
1    1 49.52246 19.11011 13.84968   34.64694
2    2 54.57170 22.29326 15.17248   12.34458
3    3 62.53950 37.18071 18.25763   18.25330
4    4 55.21535 25.85652 15.83278   26.67333
5    5 55.08405 34.17193 18.90686   13.44716
```

Les résultats de la CAH sont cartographiés à la figure 8.11 tandis que le nombre d'observations et les valeurs moyennes des classes sont reportés au tableau tableau 8.2.

```
library(tmap)
## Cartographie des résultats
Carte.CAH5 <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
              tm_fill(col="CAH5", palette = "Set1", title ="")+
              tm_layout(frame=FALSE,
                main.title = "CAH (critère de Ward)",
                main.title.position = "center",
                main.title.size = 1)
Carte.CAH5
```

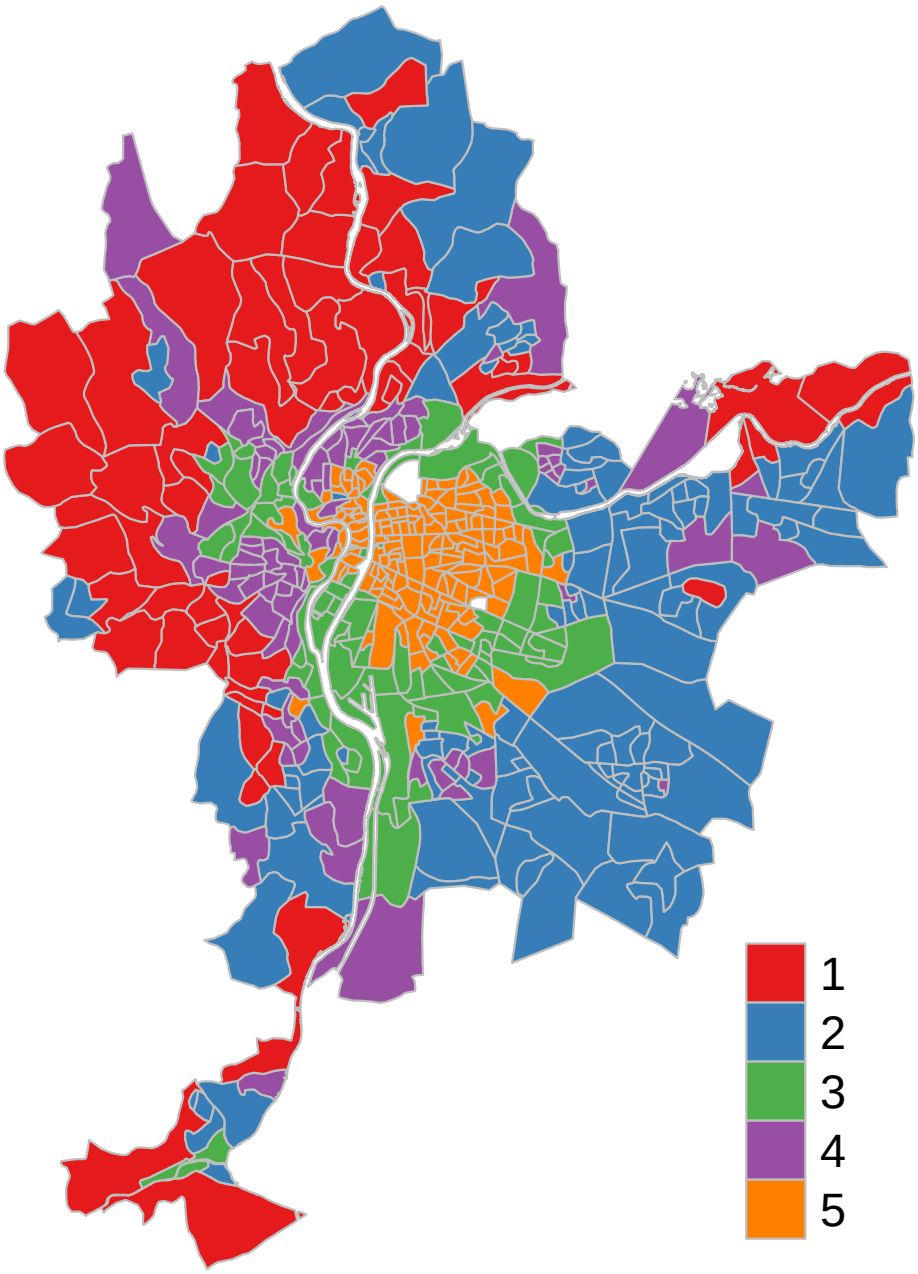


Figure 8.11 – CAH avec le critère de Ward avec cinq classes

Tableau 8.2 – Valeurs moyennes des variables pour les cinq classes obtenues avec la CAH

Classe	Lden	NO2	PM25	Végétation	Nombre d'IRIS
1	49,5	19,1	13,8	34,6	62
2	54,6	22,3	15,2	12,3	120

Tableau 8.2 – Valeurs moyennes des variables pour les cinq classes obtenues avec la CAH

Classe	Lden	NO2	PM25	Végétation	Nombre d'IRIS
3	62,5	37,2	18,3	18,3	88
4	55,2	25,9	15,8	26,7	81
5	55,1	34,2	18,9	13,4	155

8.2.1.3 Calcul de la méthode ClustGeo

Calcul des deux matrices (sémantique et spatiale)

Dans un premier temps, nous créons les matrices sémantique (D_0) et spatiale (D_1). Notez ici que les distances spatiales utilisées correspondent aux distances euclidiennes entre les centroïdes des IRIS.

```
library(sf)
library(ClustGeo)
## Variables
VarsEnv <- c("Lden", "NO2", "PM25", "VegHautPrt")
## Dataframe sans la géométrie et les quatre variables
load("data/chap08/DonneesLyon.Rdata")
Data <- st_drop_geometry(LyonIris[VarsEnv])
## Centrage (moyenne = 0) et réduction des données (variance = 1)
DataZscore <- data.frame(scale(Data))
## Matrice sémantique : dissimilarité des observations selon les variables
Matrice.Semantique <- dist(DataZscore, method = "euclidean")
## Matrice spatiale entre les unités spatiales
xy <- st_coordinates(st_centroid(LyonIris))
Matrice.Spatiale <- dist(xy, method = "euclidean")
```

Optimisation de la valeur de α

Pour la méthode *ClustGeo* (avec k = 5), nous évaluons l'impact du paramètre α pour des valeurs de 0 à 1, avec un saut de 0,05. Pour ce faire, nous utilisons la fonction `choicealpha` du *package* `ClustGeo`. À la lecture de la figure 8.12, nous constatons que :

- Plus la valeur de α augmente, plus l'inertie expliquée par la matrice sémantique diminue (trait noir) et inversement, plus l'inertie expliquée par la matrice spatiale est forte.
- Avec $\alpha = 0{,}30$, l'inertie expliquée par la matrice spatiale est de 50 % pour une perte d'inertie expliquée par la matrice sémantique de seulement 7 %. Avec $\alpha = 0{,}35$, nous constatons une chute importante de l'inertie sémantique expliquée. Par conséquent, nous retenons la valeur de 0,30 pour la méthode *ClustGeo*. Ce choix donne un excellent compromis entre la préservation de l'inertie expliquée des données attributaires et la cohérence spatiale de la classification finale.

```
alphas <- seq(0, 1, 0.05)
result <- choicealpha(D0 = Matrice.Semantique, # matrice sémantique
                      D1 = Matrice.Spatiale,   # matrice spatiale
```

```
                    range.alpha = alphas,    # valeurs de alpha
                    K = 5,                   # nombre de classes
                    wt = NULL, scale = TRUE, graph = FALSE)
# Graphique avec Alpha
df.alpha <- data.frame(result$Q)
df.alpha$alpha <- alphas
ggplot(df.alpha)+
  geom_line(aes(x=alphas,y= Q0), color = "black")+
  geom_point(aes(x=alphas,y= Q0), color = "black", size=3) +
  geom_line(aes(x=alphas,y= Q1), color = "red")+
  geom_point(aes(x=alphas,y= Q1), color = "red", size=3) +
  labs(y = "Pseudo-inertie",
       x = "Paramètre alpha",
       subtitle = "Matrice sémantique (noir) et matrice spatiale (rouge)")
```

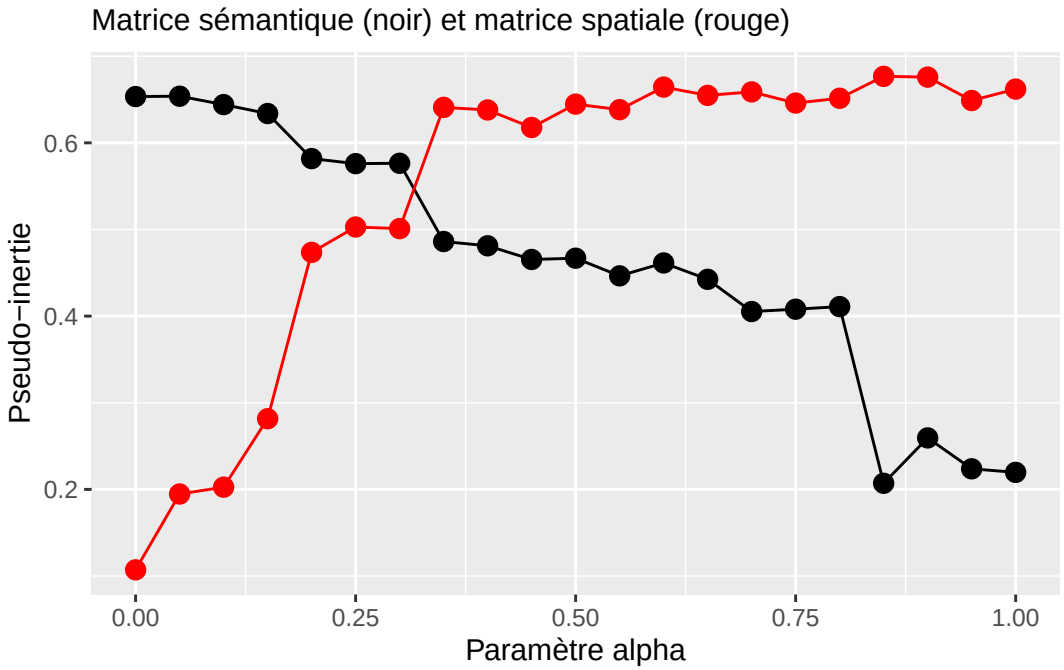


FIGURE 8.12 – Impact des deux matrices dans la classification

Réalisation de la méthode *ClustGeo*

```
## Dendrogramme avec ClustGeo
Arbre.ClustGeo <- hclustgeo(D0 = Matrice.Semantique, D1 = Matrice.Spatiale, alpha = 0.30)
## Coupure de l'arbre à cinq classes
LyonIris$ClustGeo5 <- as.character(cutree(Arbre.ClustGeo, k=5))
## Nombre d'observations par classe
table(LyonIris$ClustGeo5)
```

```
  1   2   3   4   5
101  73 102 145  85
```

```
## Valeurs moyennes des classes
aggregate(cbind(Lden,NO2,PM25,VegHautPrt) ~ ClustGeo5,
          data = st_drop_geometry(LyonIris), FUN = mean)
```

```
  ClustGeo5     Lden      NO2     PM25 VegHautPrt
1         1 51.99446 22.63948 14.69971   32.70396
2         2 55.46789 20.84670 15.00392   14.07849
3         3 61.41468 36.20785 18.23707   18.77059
4         4 54.98927 34.16558 18.93142   12.84883
5         5 54.05419 24.32165 15.45733   16.14224
```

```
## Cartographie des résultats
Carte.ClusteGeo <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
                    tm_fill(col="ClustGeo5", palette = "Set1", title ="")+
                    tm_layout(frame=FALSE,
                      main.title = "ClustGeo avec alpha = 0,30",
                      main.title.position = "center",
                      main.title.size = 1)
Carte.ClusteGeo
```

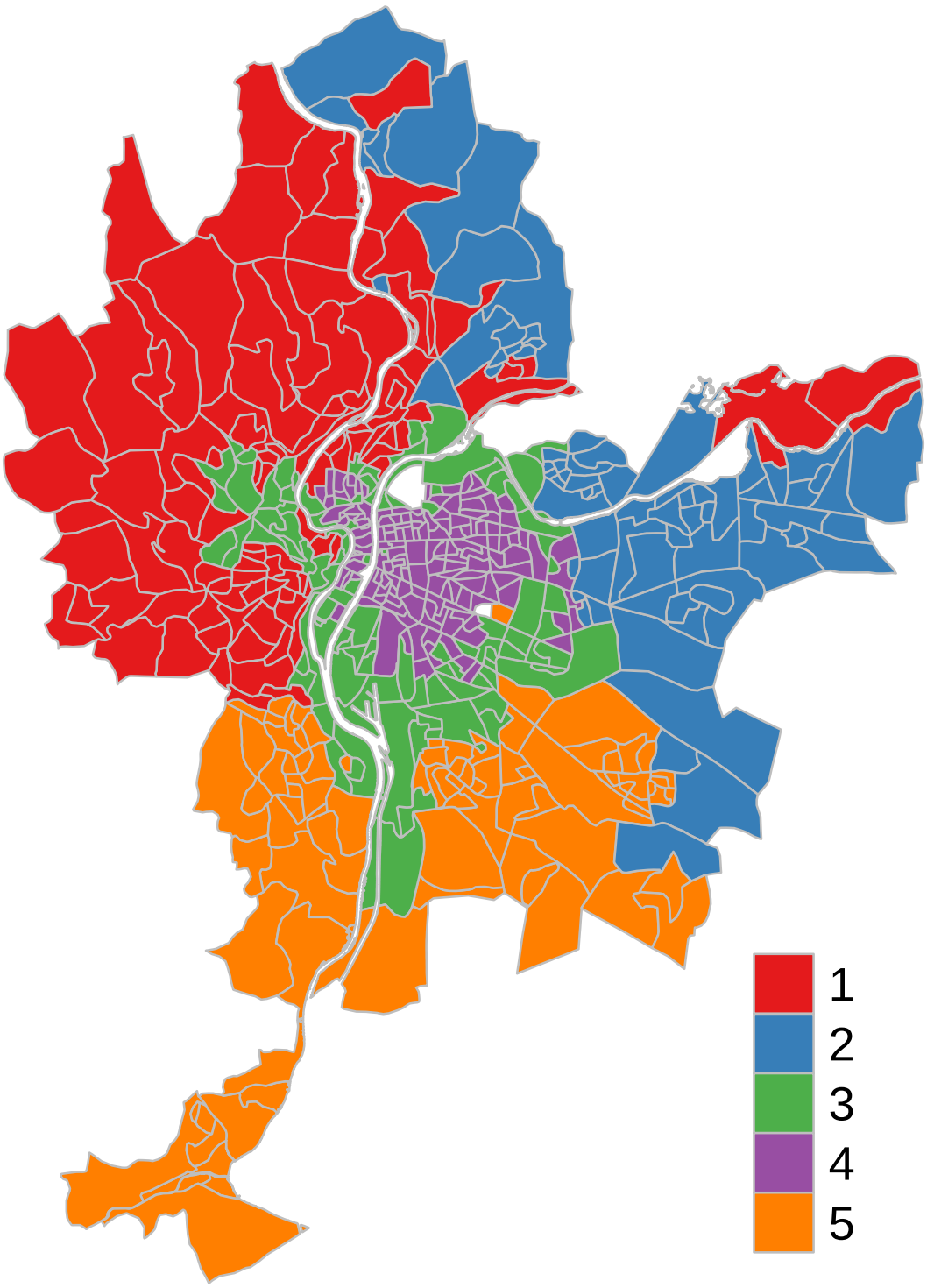


Figure 8.13 – Classification ClustGeo avec alpha = 0,30

Tableau 8.3 – Valeurs moyennes des variables pour cinq classes obtenues par la méthode ClustGeo (alpha = 0,30)

Classe	Lden	NO2	PM25	Végétation	Nombre d'IRIS
1	52,0	22,6	14,7	32,7	101
2	55,5	20,8	15,0	14,1	73
3	61,4	36,2	18,2	18,8	102
4	55,0	34,2	18,9	12,8	145
5	54,1	24,3	15,5	16,1	85

La comparaison des typologies obtenues avec la CAH et la ClusteGeo démontre clairement que l'introduction d'une matrice spatiale dans la classification *ClustGeo* génère des classes qui sont moins discontinues spatialement (figure 8.14).

```
tmap_arrange(Carte.CAH5, Carte.ClusteGeo, nrow = 1, ncol = 2)
```

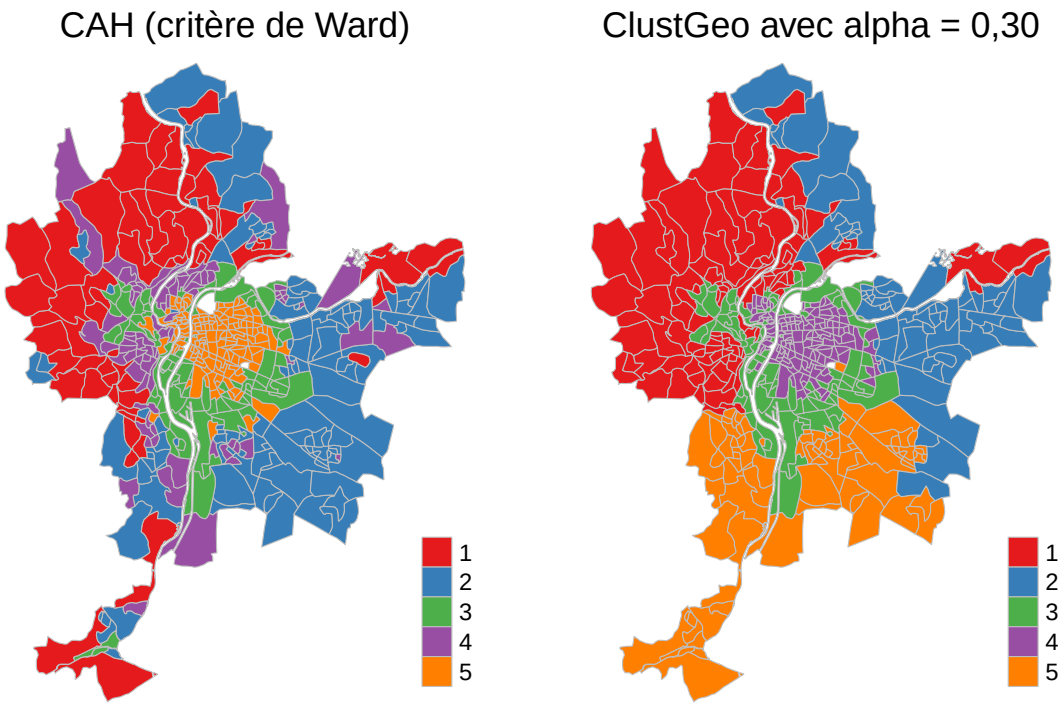


Figure 8.14 – Impact des deux matrices dans la classification

8.2.2 Spatial fuzzy c-means

La méthode SFCM (*Spatial fuzzy c-means*), proposée par Weiling Cai et ses collègues (2007), est une extension de FCM, soit une version floue de l'algorithme des k-moyennes. Le principe de base est le suivant :

« Comparativement au FCM classique, le SFCM introduit dans son calcul, en plus du jeu de données original (D_0), une version spatialement décalée (D_s) de ce dernier (Cai, Chen et Zhang 2007). En analyse d'image, cela revient à calculer D_s en appliquant un filtre moyen ou médian à D_0 (la médiane étant moins sensible aux valeurs extrêmes locales). Ce processus peut facilement s'appliquer à des entités géographiques vectorielles, en créant une matrice de pondération

spatiale W_{kl} (l étant les voisins de k et la diagonale de cette matrice valant 0) (Getis 2009) et en utilisant les poids de cette matrice dans le calcul d'une moyenne ou d'une médiane pondérée locale » (Jérémy Gelb et Apparicio 2021, 8).

Comme pour la méthode *ClustGeo*, il est possible de fixer une pondération à la dimension spatiale (D_s) avec un paramètre α, qui varie de 0 à ∞. Une valeur de 0 signale qu'aucun poids n'est accordé à la dimension spatiale, ce qui revient à calculer un FCM classique. Si $\alpha = 2$, alors la version spatialement décalée aura deux fois plus de poids dans la classification que le jeu de données original.

⚠ **Attention**

Retour sur une variable spatialement décalée et la classification k-moyennes
La notion de variable spatialement décalée a été abordée au chapitre 2 (figure 2.29). Pour une description détaillée de la classification k-moyennes, consultez la section suivante (Apparicio et Gelb 2022).

8.2.2.1 Calcul de la classification c-moyennes floue classique (fuzzy c-means)

À des fins de comparaison avec la CAH, nous proposons de calculer une classification c-moyennes floue classique (*fuzzy c-means*) avec cinq classes (k = 5). Puis, pour déterminer la valeur optimale de m (soit le degré de logique floue), nous calculons l'inertie expliquée et l'indice de silhouette pour des valeurs de m variant de 1,1 à 3 (avec un incrément de 0,1). Les résultats de ces deux indicateurs, présentés à la figure 8.15, signalent que :

- Plus le paramètre m augmente, plus l'inertie expliquée diminue (figure 8.15, a).
- La valeur de l'indice de silhouette est maximale avec m = 1,8 (figure 8.15, b).

```
library(geocmeans)
library(ggpubr)
## Variables pour la CAH
VarsEnv <- c("Lden", "NO2", "PM25", "VegHautPrt")
## Dataframe sans la géométrie et les quatre variables
load("data/chap08/DonneesLyon.Rdata")
Data <- st_drop_geometry(LyonIris[VarsEnv])
## Centrage (moyenne = 0) et réduction des données (variance = 1)
DataZscore <- data.frame(scale(Data))
## Dataframe pour les différents paramètres avec k = 5 et degré de flou
FCM_selection <- select_parameters(algo = "FCM",
                  data = DataZscore,
                  k = 5, # nous pourrions ici tester avec k=2:10
                  m = seq(1.1,3,0.1),
                  classidx = TRUE, spconsist = FALSE,
                  tol = 0.001, seed = 456,
                  verbose = FALSE)
```

```
[1] "number of combinaisons to estimate :  20"
```

```
# Graphique avec l'inertie expliquée
G1 <- ggplot(FCM_selection) +
      geom_line(aes(x = m, y = Explained.inertia)) +
      geom_point(aes(x = m,  y = Explained.inertia), color = "red")+
      labs(title ="a. Variation des données expliquées",
           y = "Inertie expliquée", x = "Paramètre m")
# Graphique avec l'indice de silhouette
G2 <- ggplot(FCM_selection) +
      geom_line(aes(x = m, y = Silhouette.index)) +
      geom_point(aes(x = m,  y = Silhouette.index), color = "red")+
      labs(title ="b. Consistance des groupes",
           y = "Critère de silhouette floue", x = "Paramètre m")
# Combinaison des deux graphiques dans la figure
ggarrange(G1, G2)
```

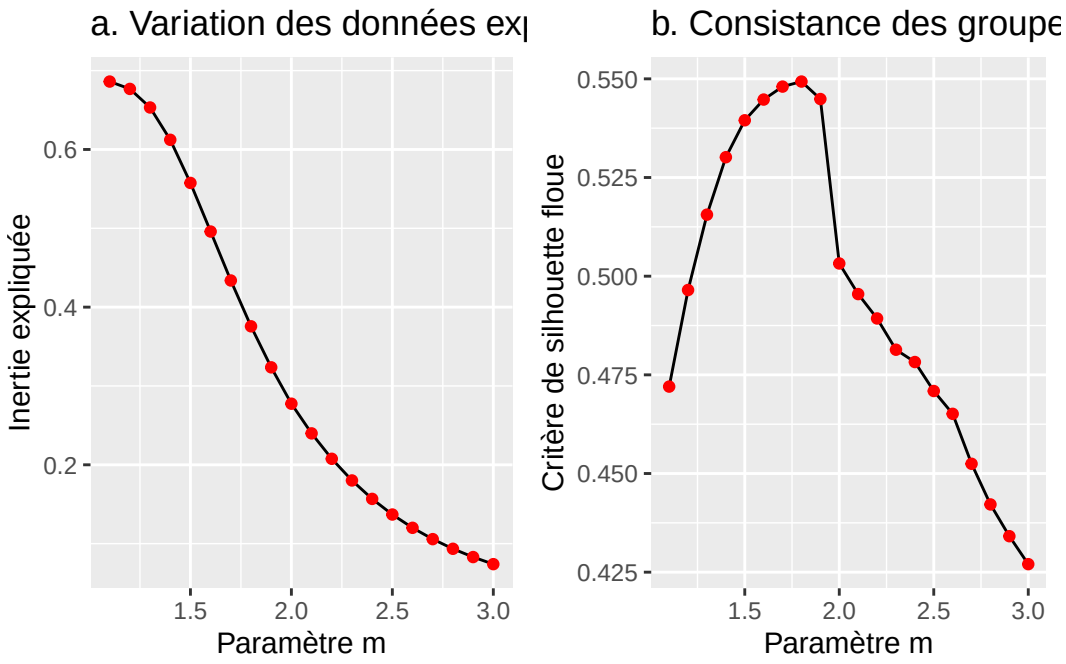


FIGURE 8.15 – Évaluation de la qualité de la classification FCM avec cinq classes selon le degré de flou (m)

Réalisons la classification c-moyennes floue et cartographions les probabilités d'appartenance à chacune des classes (figure 8.16), puis l'appartenance à une classe (figure 8.17).

```
## Classification du c-moyennes
FCM <- CMeans(DataZscore, k = 5, m = 1.8, tol = 0.0001, verbose = FALSE, seed = 456)
## Calcul des indicateurs de qualité
calcqualityIndexes(DataZscore, FCM$Belongings, 1.5)
```

```
$Silhouette.index
```

```
[1] 0.5502472

$Partition.entropy
[1] 0.9663788

$Partition.coeff
[1] 0.5143427

$XieBeni.index
[1] 1.269992

$FukuyamaSugeno.index
[1] 158.1419

$Explained.inertia
[1] 0.3757482
```

```
## Cartographie des probabilités d'appartenance à chaque classe
Cartes.FCM <- mapClusters(LyonIris,FCM$Belongings)
names(Cartes.FCM)
```

```
[1] "ProbaMaps"   "ClusterPlot"
```

```
tmap_arrange(Cartes.FCM$ProbaMaps, ncol = 2)
```

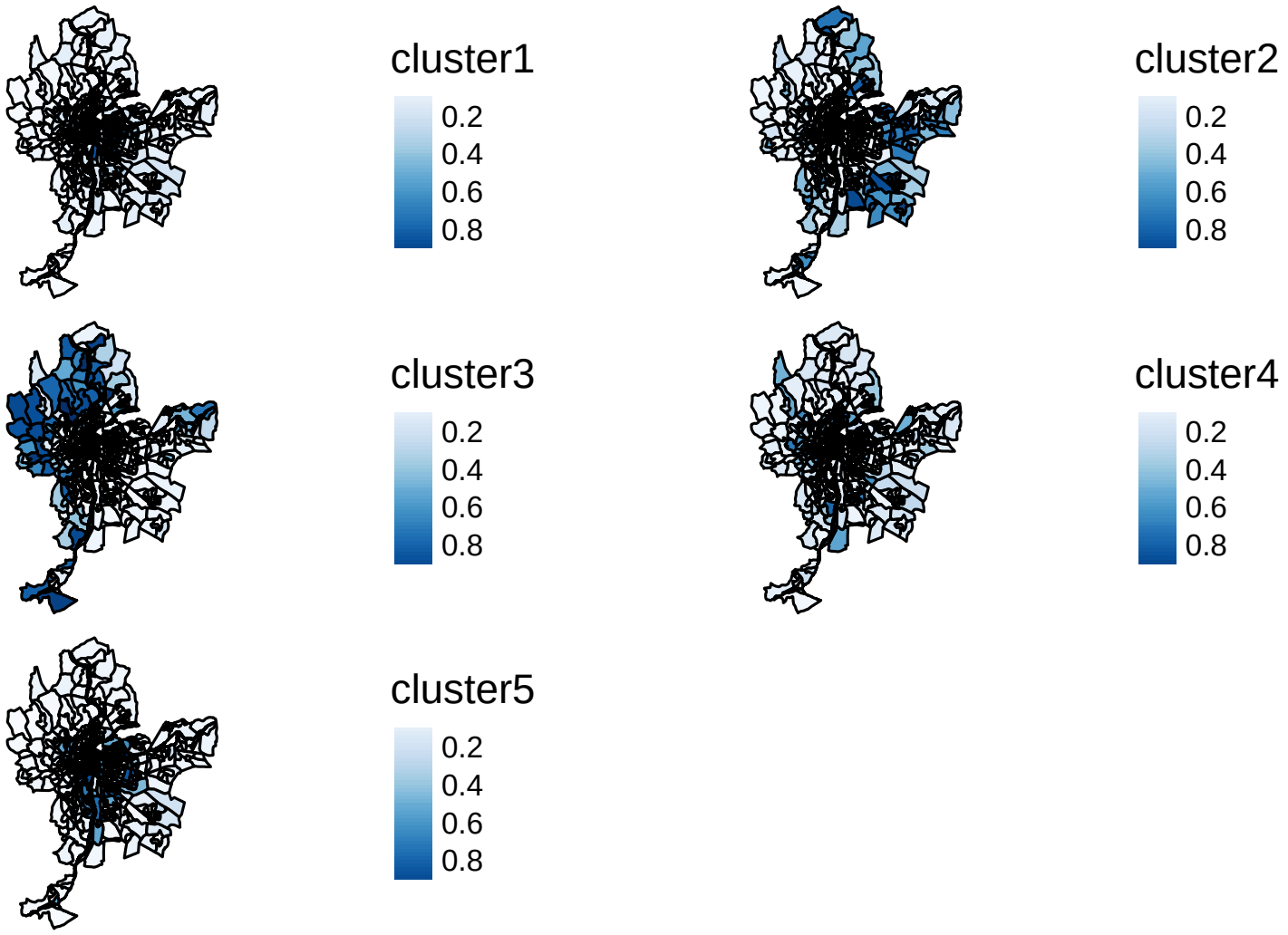


FIGURE 8.16 – Cartographie des probabilités d'appartenance aux cinq classes avec la classification c-moyennes

```
Cartes.FCM$ClusterPlot
```

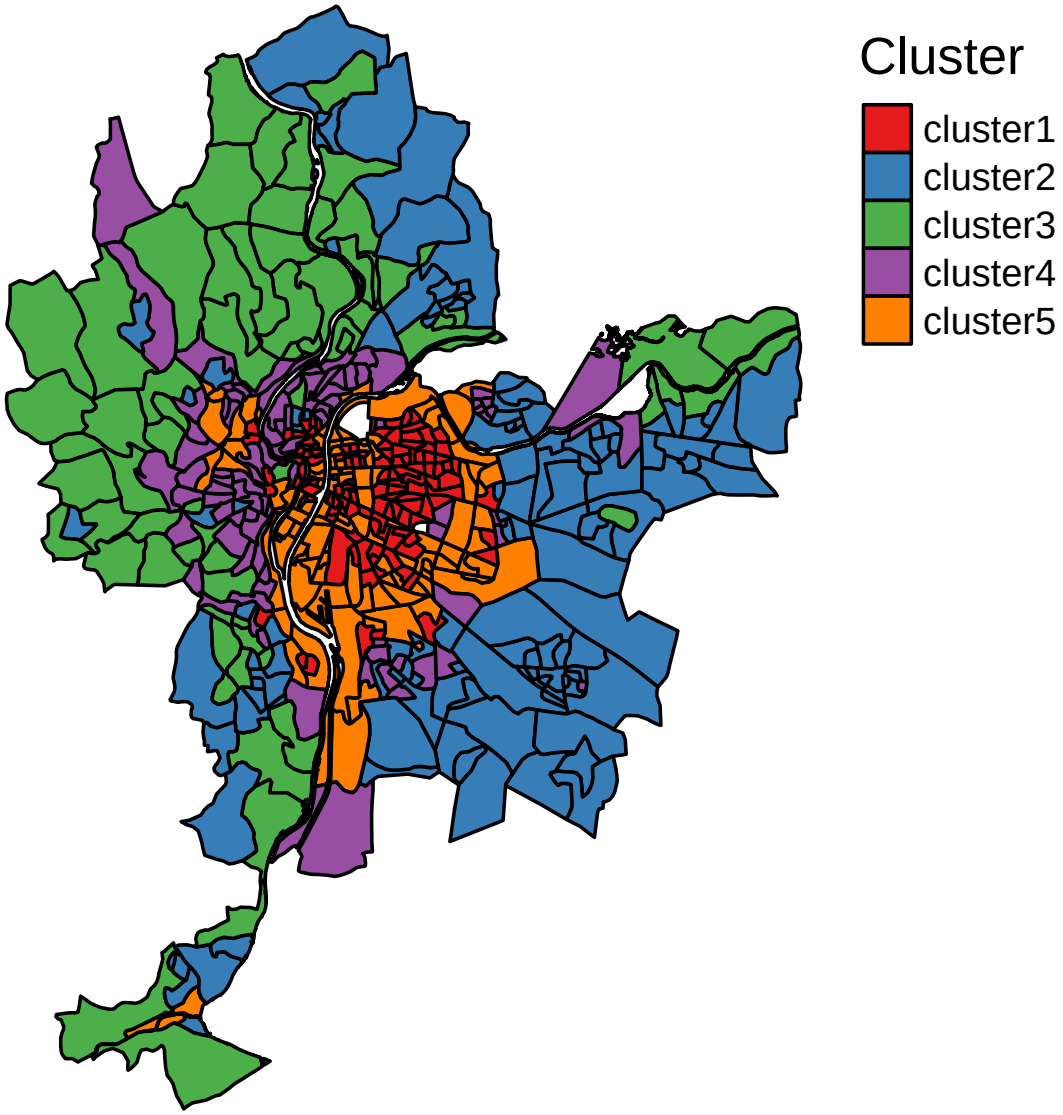


Figure 8.17 – Cartographie des classes issues de la classification c-moyennes

8.2.2.2 Calcul de la classification c-moyennes floue et spatiale (spatial fuzzy c-means)

Dans l'exemple ci-dessous, nous calculons une classification SFCM (*spatial fuzzy c-means*). Nous avons déterminé avec l'analyse du simple FCM que la valeur de 1,8 pour m semblait satisfaisante avec cinq groupes (k = 5). Nous devons maintenant déterminer la valeur de *alpha*, soit le poids accordé aux variables spatialement décalées comparativement aux données originales.

Pour cela, nous calculons toutes les valeurs possibles d'*alpha* entre 0 et 2 avec un intervalle de 0,05. Nous comparons ensuite les valeurs des indices de silhouette (qualité sémantique de la classification) et d'incohérence spatiale des différentes solutions.

```
library(spdep)
library(ggplot2)
# Création d'une matrice de contiguïté standardisée
Neighbours <- poly2nb(LyonIris, queen = TRUE)
WMat <- nb2listw(Neighbours, style="W", zero.policy = TRUE)
SFCM_selection <- select_parameters(algo = "SFCM",
                  data = DataZscore,
                  k = 5,
                  m = 1.8,
                  nblistw = WMat,
                  alpha = seq(0,2,0.05),
                  classidx = TRUE,
                  tol = 0.001, seed = 456,
                  spconsist  = TRUE,
                  verbose = FALSE)
```

```
[1] "number of combinaisons to estimate :  41"
```

```
graph1 <- ggplot(SFCM_selection) +
  geom_line(aes(x = alpha, y = Silhouette.index), color = 'black') +
  geom_point(aes(x = alpha, y = Silhouette.index), color = 'red')+
  labs(x = "Alpha", y = "Indice de silhouette", )

graph2 <- ggplot(SFCM_selection) +
  geom_line(aes(x = alpha, y = spConsistency), color = 'black') +
  geom_point(aes(x = alpha, y = spConsistency), color = 'red')+
  labs(x = "Alpha", y = "Indice d'incohérence spatiale")

ggarrange(graph1, graph2, ncol = 2, nrow = 1)
```

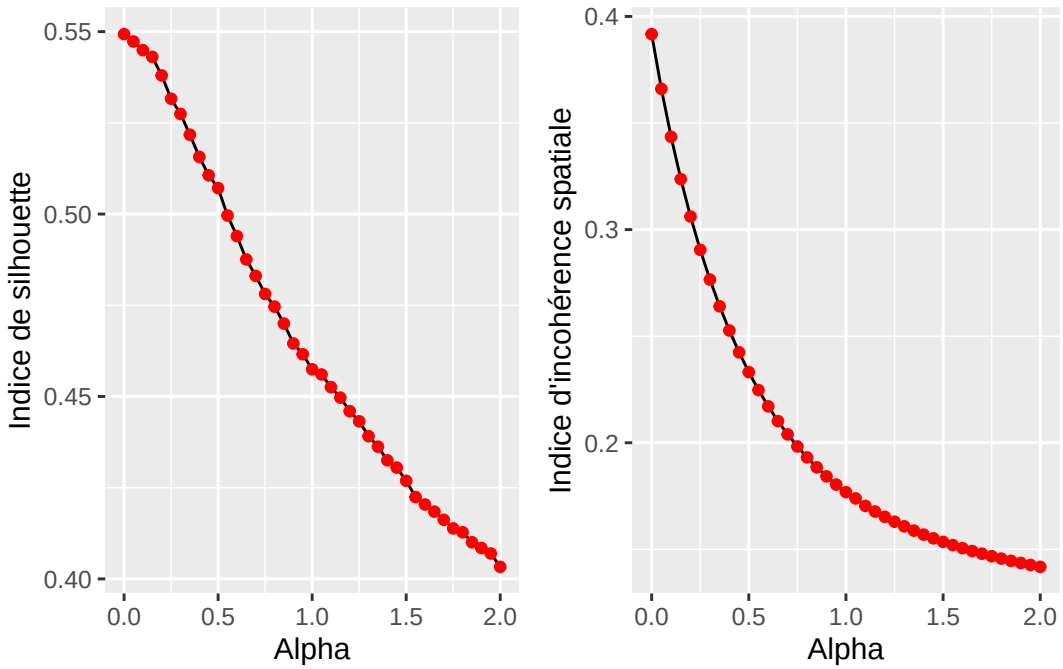


FIGURE 8.18 – Comparaison de différentes valeurs d'alpha pour le SFCM

La figure 8.18 démontre que l'augmentation d'*alpha* a pour effet de réduire la qualité sémantique de la classification (indice de silhouette), mais aussi de limiter l'inconsistance spatiale des résultats produits. Nous constatons aussi que l'augmentation d'*alpha* a un effet quasiment linéaire sur la dégradation de l'indice de silhouette et un effet curvilinéaire sur l'inconsistance spatiale. Cette dernière est donc plus impactée par les premières augmentations d'*alpha*. Nous proposons ici de retenir une solution avec α = 0,55, car elle correspond à une baisse relativement faible de l'indice de silhouette (0,55 versus 0,50) pour une augmentation importante de la cohérence spatiale des résultats (0,39 versus 0,22).

```
# Calcul du SFCM
SFCM <- SFCMeans(DataZscore, WMat,
                 k = 5,
                 m = 1.8,
                 alpha = 0.55,
                 tol = 0.0001, standardize = FALSE,
                 verbose = FALSE, seed = 456)
```

Résultats des indicateurs de qualité du SFCM

```
calcqualityIndexes(DataZscore, SFCM$Belongings, 1.5)
```

```
$Silhouette.index
[1] 0.4995905

$Partition.entropy
[1] 1.039552
```

```
$Partition.coeff
[1] 0.4726401

$XieBeni.index
[1] 2.011733

$FukuyamaSugeno.index
[1] 380.0203

$Explained.inertia
[1] 0.3371823
```

Cartographie des probabilités d'appartenance à chaque classe

La cartographie des probabilité d'appartenance pour chacune des cinq classes est présentée aux figures 8.19, 8.20, 8.21, 8.22 et 8.22.

```
Cartes.SFCM <- mapClusters(LyonIris, SFCM$Belongings, undecided = 0.45)
Cartes.SFCM$ProbaMaps[[1]]
```

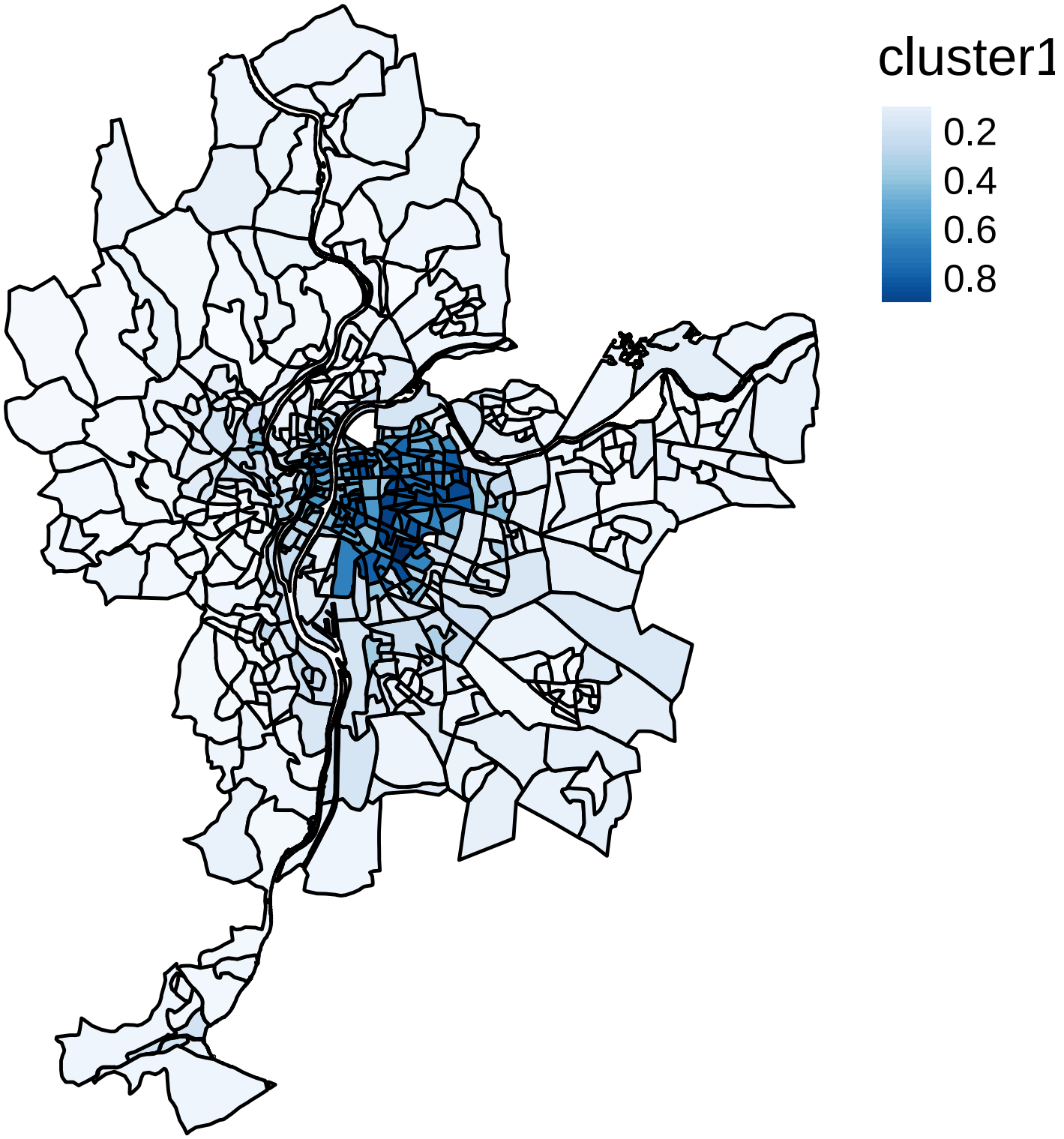


FIGURE 8.19 – Cartographie des probabilités d'appartenance issues de la classification SFCM (classe 1)

```
Cartes.SFCM$ProbaMaps[[2]]
```

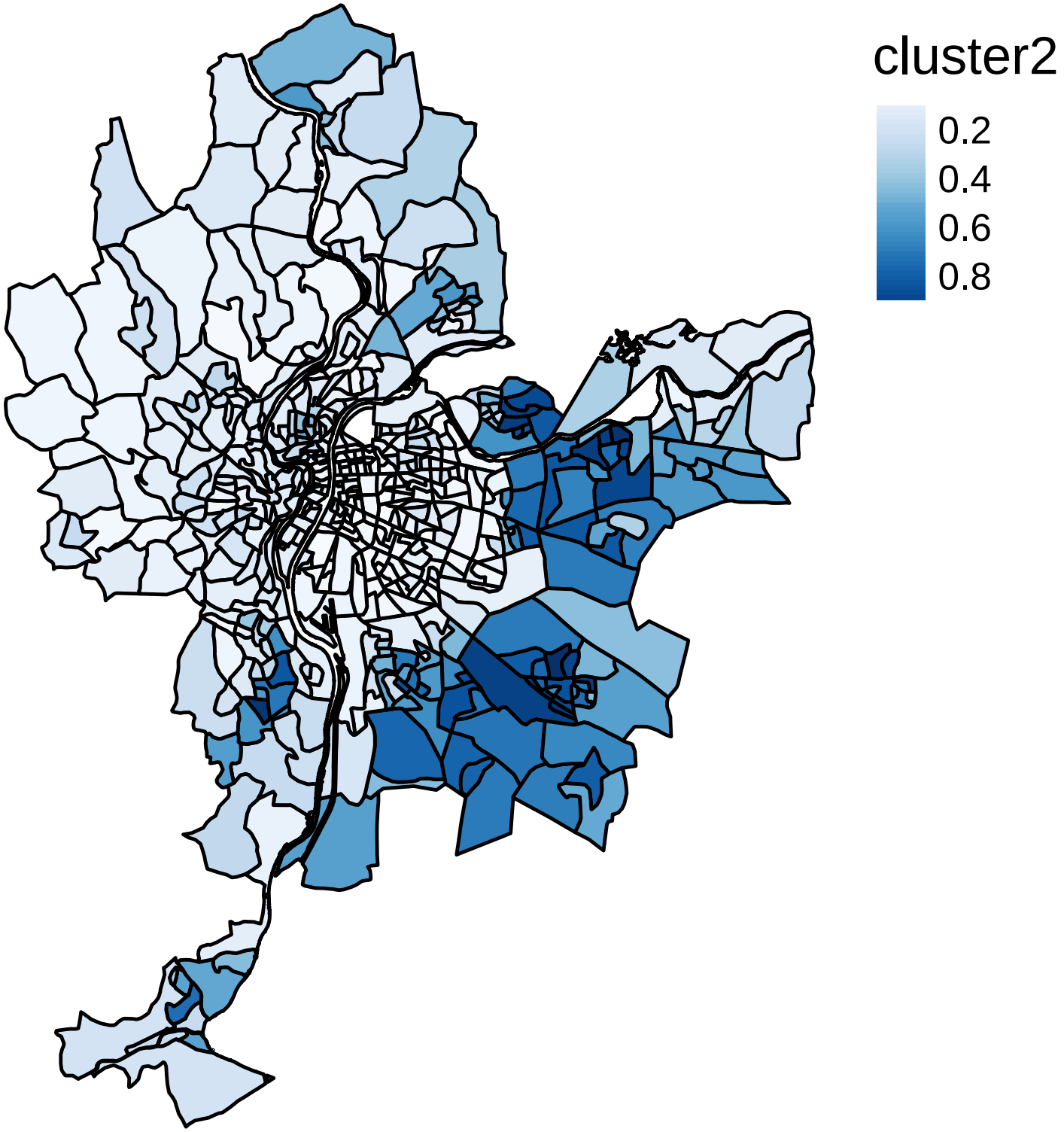


FIGURE 8.20 – Cartographie des probabilités d'appartenance issues de la classification SFCM (classe 2)

```
Cartes.SFCM$ProbaMaps[[3]]
```

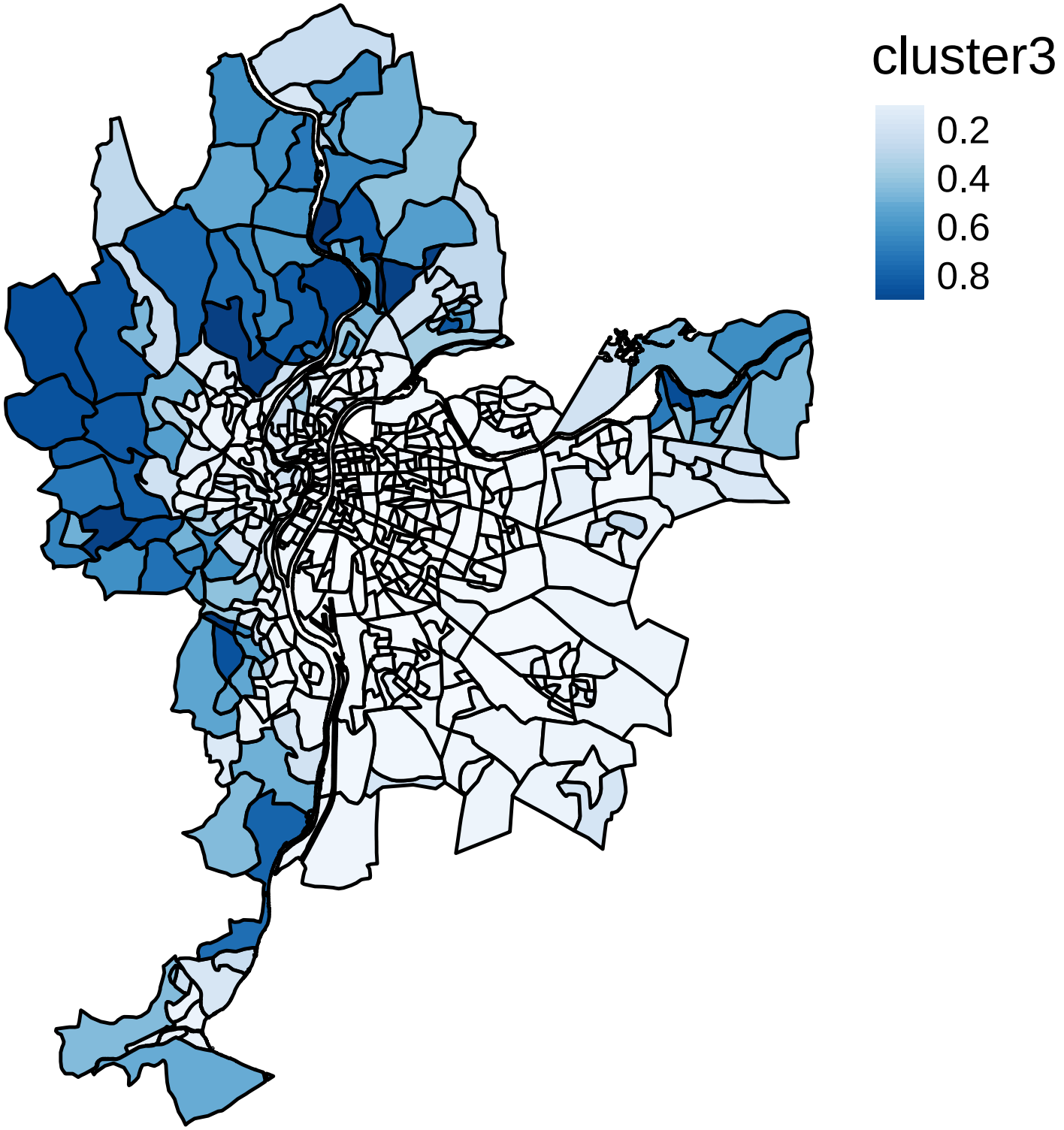


FIGURE 8.21 – Cartographie des probabilités d'appartenance issues de la classification SFCM (classe 3)

```
Cartes.SFCM$ProbaMaps[[4]]
```

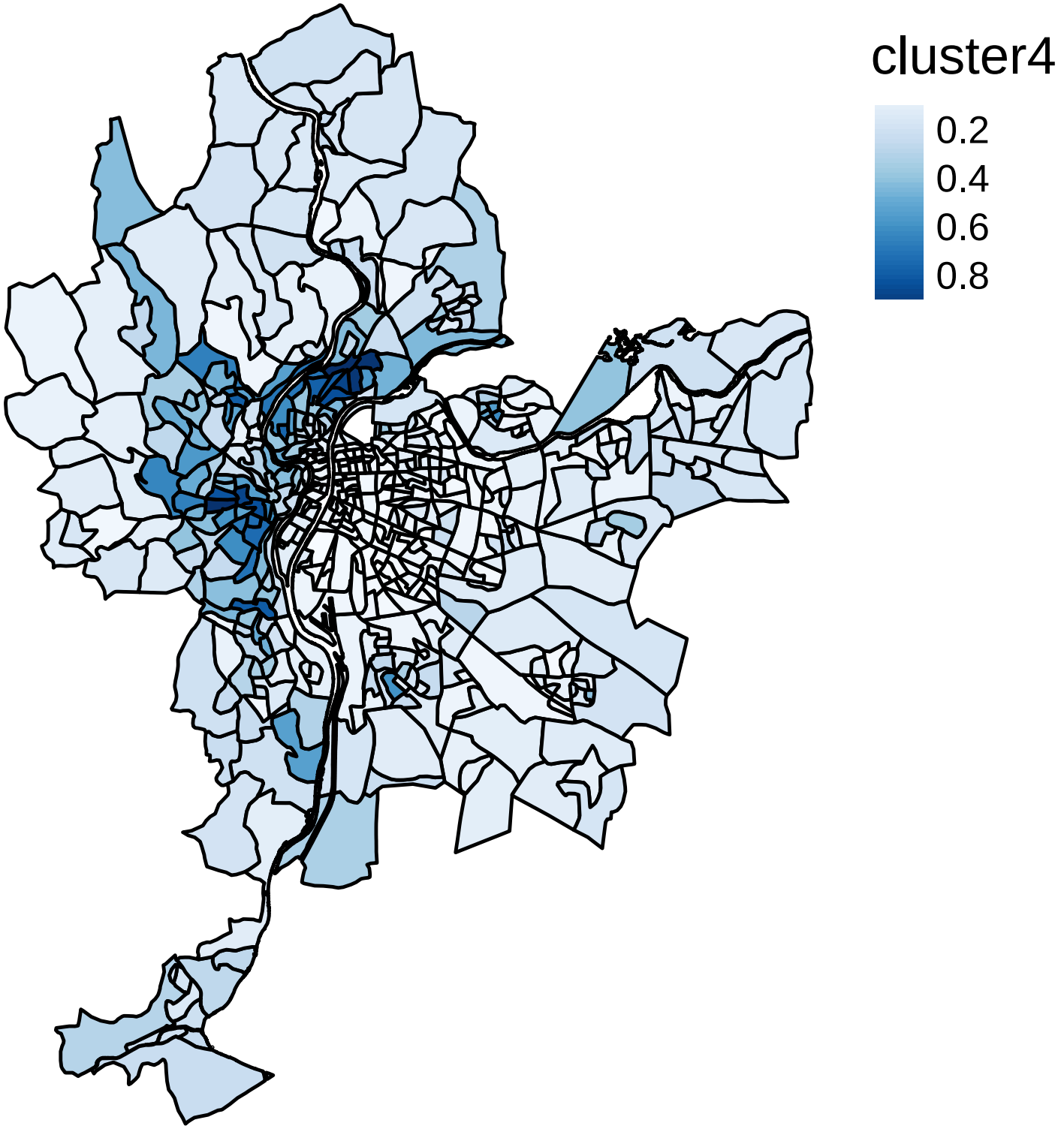


FIGURE 8.22 – Cartographie des probabilités d'appartenance issues de la classification SFCM (classe 4)

```
Cartes.SFCM$ProbaMaps[[5]]
```

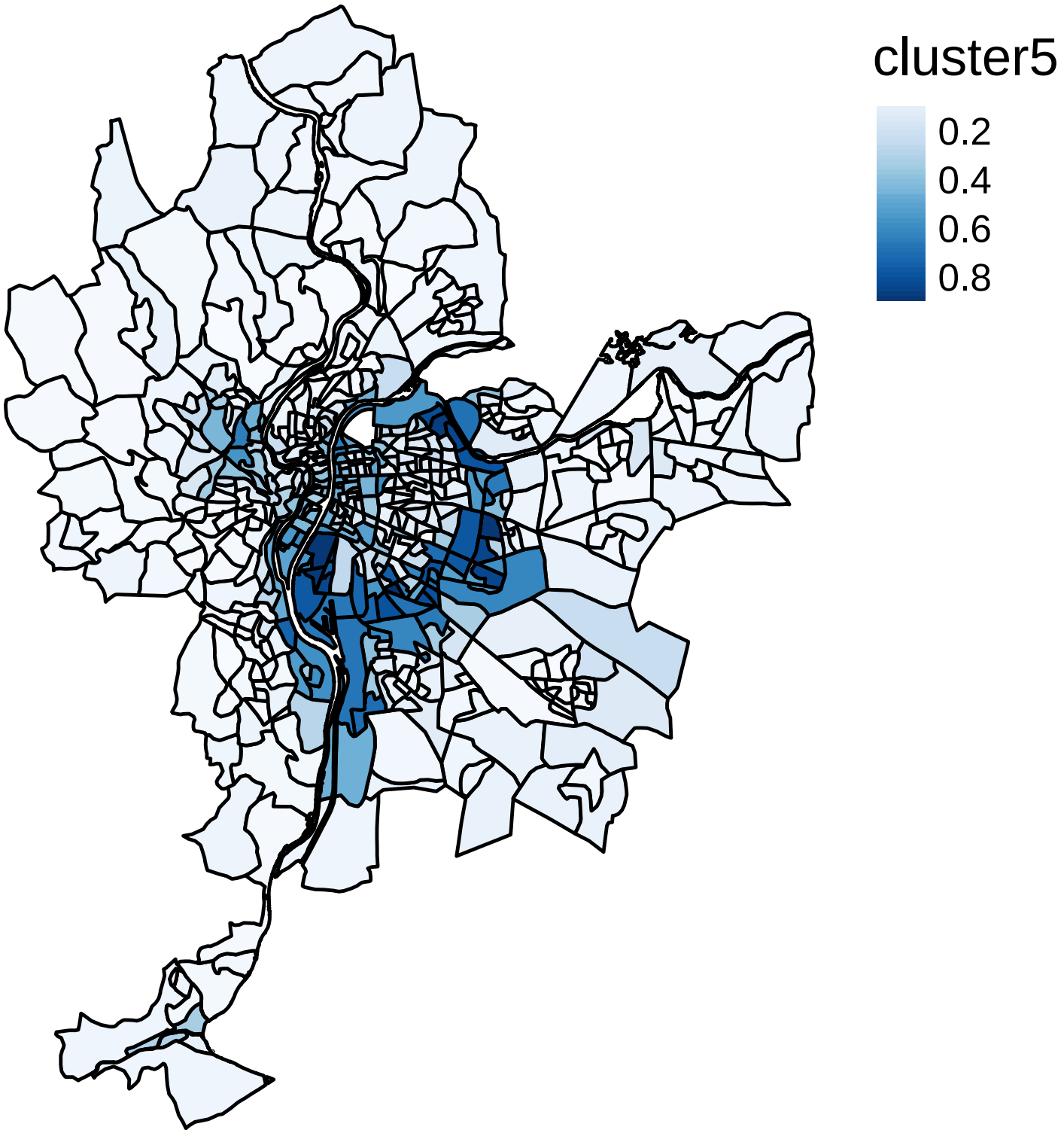


FIGURE 8.23 – Cartographie des probabilités d'appartenance issues de la classification SFCM (classe 5)

Cartographie des probabilités d'appartenance à chaque classe du SFCM (figure 8.24)

```
Cartes.SFCM$ClusterPlot
```

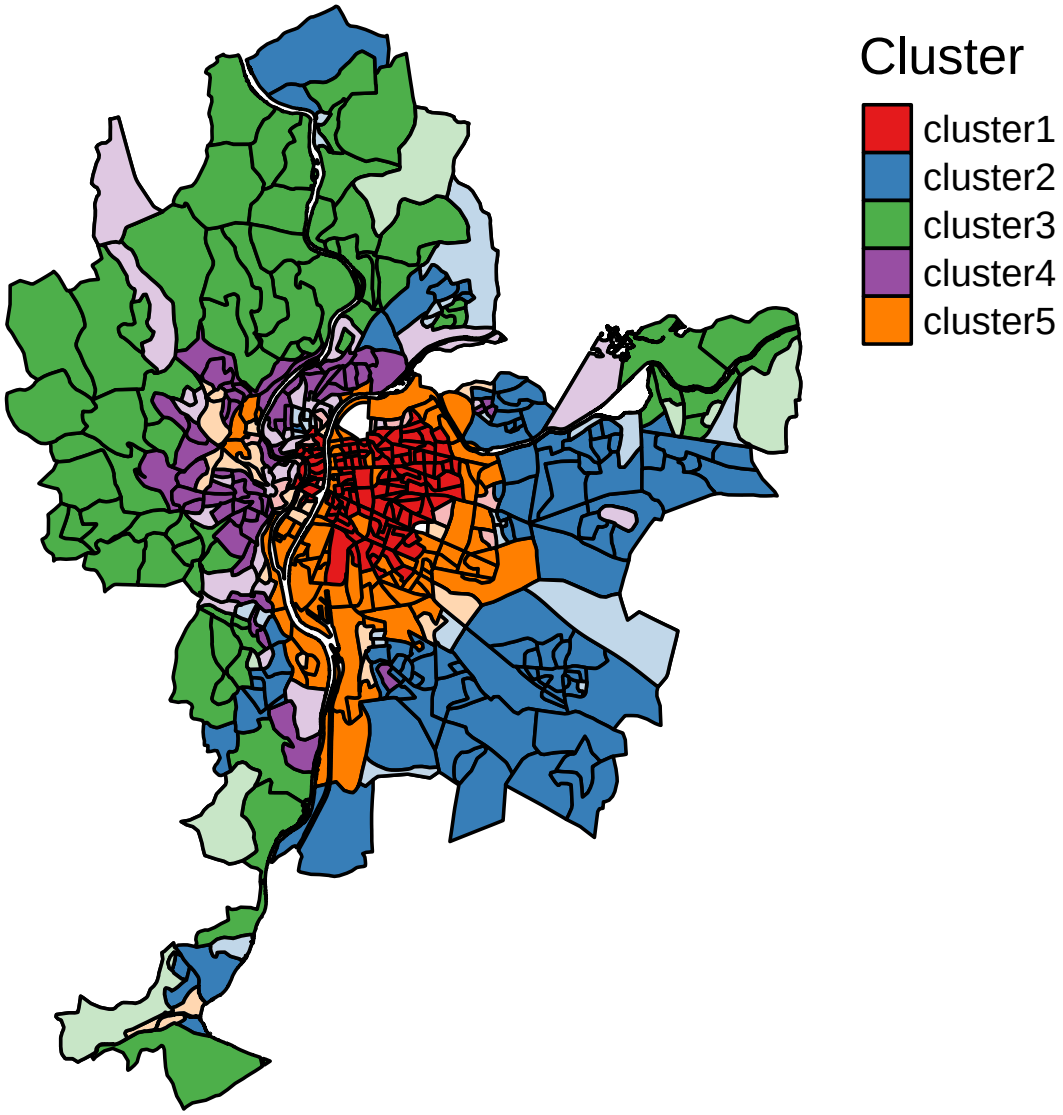


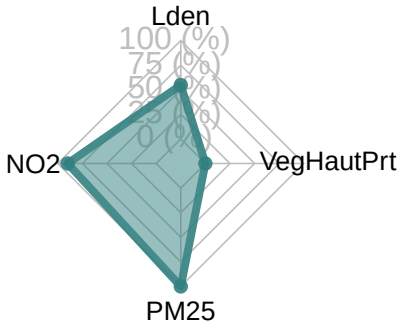
Figure 8.24 – Cartographie des classes issues de la classification SFCM

Il est intéressant de noter que les groupes construits par les deux algorithmes (ClustGeo et SFCM) produisent des solutions assez différentes. Nous retrouvons dans les deux classifications un groupe caractérisant les quartiers centraux, puis une distinction assez nette entre les secteurs est et ouest de l'agglomération. Cependant, *ClustGeo* regroupe les Iris du sud dans un groupe spécifique alors que SFCM a identifié deux groupes plus atypiques spatialement. Le groupe 5 correspond aux Iris situés le long du boulevard périphérique encerclant Lyon.

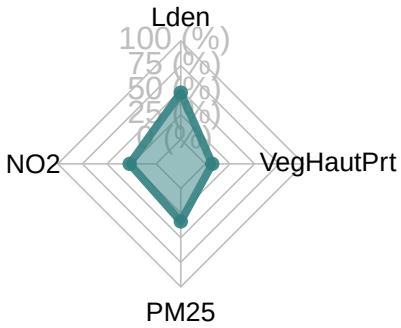
Le groupe 5 correspond donc à un ensemble d'Iris avec les plus hauts niveaux de concentration de pollutions atmosphérique et sonore (figure ci-dessous). Il est similaire en cela au groupe 1, qui est cependant marqué par des niveaux de bruit un peu moins élevés.

```
spiderPlots(Data,SFCM$Belongings,
            chartcolors = c("darkorange3",
                            "grey4",
                            "darkgreen",
                            "royalblue",
                            "blueviolet"))
```

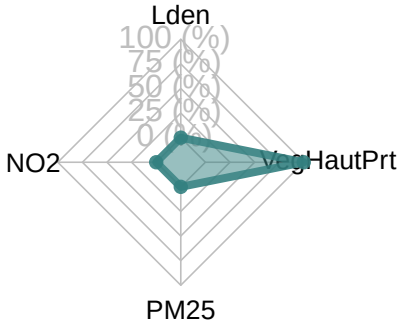
Group number : 1



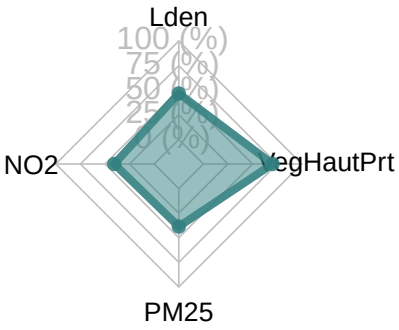
Group number : 2



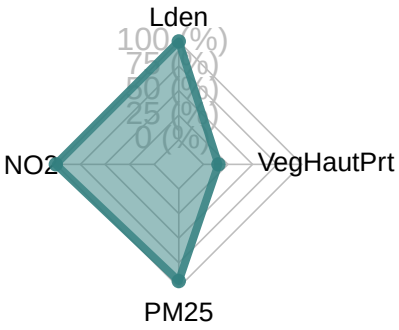
Group number : 3

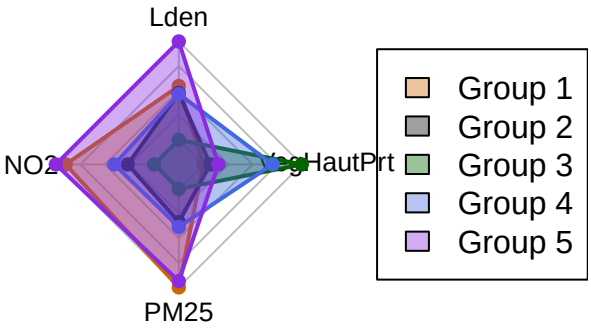


Group number : 4



Group number : 5





Aller plus loin

De multiples saveurs du SFCM

Plusieurs variantes du SFCM sont proposées par le *package* `geocmeans` qui offrent une grande flexibilité :

- La version généralisée du SFCM (appelée SFGCM) avec l'ajout d'un paramètre supplémentaire (*beta*) qui accélère la convergence du SFCM en limitant la probabilité attribuée au groupe le moins probable de chaque observation.
- Les versions robustes du SFCM et SFGCM, qui normalisent les distances calculées entre les observations pour réaliser des classifications non sphériques.
- L'ajout d'un groupe de résidus, permettant d'attribuer les observations très incertaines à un groupe « poubelle », ce qui évite d'influencer la cohérence des « vrais groupes » avec des observations très atypiques.

8.3 Quiz de révision du chapitre

Questions

- **Quelles sont les méthodes de classification non supervisée avec une contrainte spatiale?**
  - Algorithmes AZP (Automatic Zoning Problem).
  - Algorithmes SKATER.
  - Classification floue c-moyennes spatiale.
  - Algorithmes REDCAP.
  - Méthode ClustGeo.

  Relisez l'introduction du chapitre 8.
- **Quelles sont les méthodes de classification non supervisée avec une dimensions spatiale?**
  - Algorithmes AZP (Automatic Zoning Problem).
  - Algorithmes SKATER.
  - Classification floue c-moyenne spatiale.
  - Algorithmes REDCAP.
  - Méthode ClustGeo.

  Relisez l'introduction de la section 8.
- **La classification ascendante hiérarchique et la classification le k-moyennes sont des méthodes de classification non supervisée qui ne prennent pas en compte l'espace :**
  - Vrai
  - Faux

  Relisez au besoin la section 8.
- **Une méthode de classification avec contrainte spatiale produit des régions non discontinues (sans mitage spatial) :**
  - Vrai
  - Faux

  Relisez au besoin la section 8.1.
- **L'algorithme SKATER est basé sur :**
  - Une matrice de distances.
  - Un arbre couvrant de poids minimal.
  - Le I de Moran.

  Relisez au besoin la section 8.1.2.
- **La méthode ClustGeo est basée sur deux matrices :**
  - Une matrice sémantique calculée sur p variables caractérisant les unités spatiales (D0).
  - Une matrice spatiale calculée à partir des distances euclidiennes entre les unités spatiales (D1).
  - Une matrice spatialement décalée de la matrice sémantique original (Ds).

  Relisez au besoin la section 8.2.1.
- **La méthode c-moyennes spatiale floue est :**
  - Une matrice sémantique calculée sur p variables caractérisant les unités spatiales (D0).

  - Une matrice spatiale calculée à partir des distances euclidiennes entre les unités spatiales (D1).
  - Une matrice spatialement décalée de la matrice sémantique original (Ds).

  Relisez au besoin la section 8.2.2.
- **Le paramètre alpha de la méthode ClustGeo permet de définir le poids accordé à la matrice spatiale. Son intervalle de variation est de :**

  - 0 à 1.
  - 0 à l'infini.

  Relisez le deuxième encadré à la section 8.2.1.1.
- **Le paramètre alpha de la classification c-moyennes spatiale floue permet de définir le poids accordé à la matrice spatialement décalée. Son intervalle de variation est de :**

  - 0 à 1.
  - 0 à l'infini.

  Relisez la section 8.2.2.2.

Réponses

- Quelles sont les méthodes de classification non supervisée avec une contrainte spatiale?
  - Algorithmes AZP (Automatic Zoning Problem).
  - Algorithmes SKATER.
  - Algorithmes REDCAP.
- Quelles sont les méthodes de classification non supervisée avec une dimensions spatiale?
  - Classification floue c-moyenne spatiale.
  - Méthode ClustGeo.
- La classification ascendante hiérarchique et la classification le k-moyennes sont des méthodes de classification non supervisée qui ne prennent pas en compte l'espace :
  - Vrai
- Une méthode de classification avec contrainte spatiale produit des régions non discontinues (sans mitage spatial) :
  - Vrai
- L'algorithme SKATER est basé sur :
  - Un arbre couvrant de poids minimal.
- La méthode ClustGeo est basée sur deux matrices :
  - Une matrice sémantique calculée sur p variables caractérisant les unités spatiales (D0).
  - Une matrice spatiale calculée à partir des distances euclidiennes entre les unités spatiales (D1).
- La méthode c-moyennes spatiale floue est :
  - Une matrice sémantique calculée sur p variables caractérisant les unités spatiales (D0).
  - Une matrice spatialement décalée de la matrice sémantique original (Ds).
- Le paramètre alpha de la méthode ClustGeo permet de définir le poids accordé à la matrice spatiale. Son intervalle de variation est de :
  - 0 à 1.
- Le paramètre alpha de la classification c-moyennes spatiale floue permet de définir le poids accordé à la matrice spatialement décalée. Son intervalle de variation est de :
  - 0 à l'infini.

8.4 Exercices de révision

Exercice

Exercice 1. Réalisation de classification avec contrainte spatiale

```
library(rgeoda)
library(sf)
library(tmap)
## Préparation des données
load("data/chap08/DonneesLyon.Rdata")
VarSocioEco <- c("Pct0_14", "Pct_65", "Pct_Img", "Pct_brevet", "NivVieMed")
Data2 <- st_drop_geometry(LyonIris[VarSocioEco])
queen_w <- queen_weights(LyonIris)
## Classification avec k = 4
azp5_sa  <- à compléter
azp5_tab <- à compléter
skater5  <- rgeoda::skater(à compléter)
redcap5  <- à compléter
## Cartographie des résultats
LyonIris$SE.azp4_sa  <- à compléter
LyonIris$SE.azp4_tab <- à compléter
LyonIris$SE.skater4 <- à compléter
LyonIris$SE.recap4  <- à compléter
Carte1 <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
  tm_fill(col="SE.azp4_sa", palette = "Set1", title ="")+
  tm_layout(frame=FALSE,  main.title = "a. AZP-SA",
            main.title.position = "center", main.title.size = 1)
Carte2 <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
  tm_fill(col="SE.azp4_tab", palette = "Set1", title ="")+
  tm_layout(frame=FALSE,  main.title = "b. AZP-TABU",
            main.title.position = "center", main.title.size = 1)
Carte3 <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
  tm_fill(col="SE.skater4", palette = "Set1", title ="")+
  tm_layout(frame=FALSE,  main.title = "c. Skater",
            main.title.position = "center", main.title.size = 1)
Carte4 <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
  tm_fill(col="SE.recap4", palette = "Set1", title ="")+
  tm_layout(frame=FALSE,  main.title = "d. RECAP",
            main.title.position = "center", main.title.size = 1)

tmap_arrange(à compléter)
```

Correction à la section 12.8.1.

Partie 6. Échantillonnage, interpolation et désagrégation spatiales

9 Méthodes d'échantillonnage spatial

10 Méthodes d'interpolation spatiale

10.1 Méthodes déterministes

10.1.1 Interpolation polynomiale locale

10.1.2 Interpolation de l'inverse à la distance (IDW)

10.2 Méthodes géostatistiques

10.2.1 Krigeage simple

10.2.2 Krigeage universel

10.2.3 Co-krigeage

10.2.4 Krigeage résiduel

10.3 Quiz de révision du chapitre

10.4 Exercices de révision

11 Méthodes de déségrégation spatiale

11.1 Retour sur le MAUP

11.2 Carroyage et lissage

11.3 Interpolation pycnophylactique

11.4 Cartographie dasymétrique

11.5 Quiz de révision du chapitre

11.6 Exercices de révision

Partie 7. Exercices et bibliographie

12 Correction des exercices

12.1 Exercices du chapitre 1

12.1.1 Exercice 1

```
library(sf)
## Importation des deux couches
Arrond <- st_read("data/chap01/shp/Arrondissements.shp", quiet = TRUE)
Rues <- st_read("data/chap01/shp/Segments_de_rue.shp", quiet = TRUE)
## Création d'un objet sf pour l'arrondissement des Nations : requête attributive
table(Arrondissements$NOM)
Arrond.DesNations <- subset(Arrondissements,
                            NOM == "Arrondissement des Nations")
## Découper les rues avec le polygone de l'arrondissement des nations
Rues.DesNations <- st_intersection(Rues, Arrond.DesNations)
```

12.1.2 Exercice 2

```
library(sf)
library(tmap)
## Importation des deux couches
AD.RMRSherb <- st_read(dsn = "data/chap01/gpkg/Recen2021Sherbrooke.gpkg",
                       layer = "SherbAD", quiet = T)
HotelVille <- data.frame(ID = 1, Nom = "Hotel de Ville",
                         lon = -71.89306, lat = 45.40417)
HotelVille <- st_as_sf(HotelVille, coords = c("lon","lat"), crs = 4326)
## Changement de projection avant de s'assurer que les deux couches aient la même
HotelVille <- st_transform(HotelVille, st_crs(AD.RMRSherb))
## Ajout d'un champ pour la distance en km à l'hôtel de Ville pour les secteurs de recensement
AD.RMRSherb$DistHVKM <- as.numeric(st_distance(AD.RMRSherb,HotelVille)) / 1000
## Cartographie en quatre classes selon les quantiles
tmap_mode("plot")
tm_shape(AD.RMRSherb)+
  tm_fill(col= "DistHVKM",
          palette = "Reds",
          n=4,
```

```
          style = "quantile",
          title ="Distance à l'hôtel de Ville (km)")+
  tm_borders(col="black")
```

12.1.3 Exercice 3

```
library(sf)
## Importation de la couche des divisions de recensement du Québec
DR.Qc <- st_read(dsn = "data/chap01/gpkg/Recen2021Sherbrooke.gpkg",
                 layer = "DivisionsRecens2021", quiet = T)
## Importation du fichier csv des division de recensement
DR.Data <- read.csv("data/chap01/tables/DRQC2021.csv")
## Jointure attributive avec le champ IDUGD
DR.Qc <- merge(DR.Qc, DR.Data, by="IDUGD")
## Il y a déja deux champs dans la table pour calculer la densité de population :
## SUPTERRE : superficie en km2
## DRpop_2021 : population en 2021
DR.Qc$HabKm2 <- DR.Qc$DRpop_2021 / DR.Qc$SUPTERRE
head(DR.Qc, n=2)
summary(DR.Qc$HabKm2)
```

12.1.4 Exercice 4

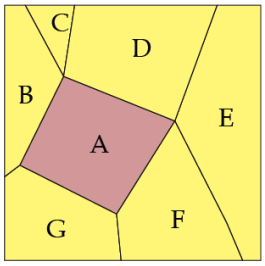
```
library(sf)
## Importation du réseau de rues
Rues <- st_read("data/chap01/shp/Segments_de_rue.shp", quiet=T)
unique(Rues$TYPESEGMEN)
## Sélection des tronçons autoroutiers
Autoroutes <- subset(Rues, TYPESEGMEN == "Autoroute")
## Création d'une couche sf pour le point avec les coordonnées
## en degrés (WGS84, EPSG : 4326) : -71.91688, 45.37579
Point1_sf <- data.frame(ID = 1,
                        lon = -71.91688, lat = 45.37579)
Point1_sf <- st_as_sf(Point1_sf, coords = c("lon","lat"), crs = 4326)
## Changement de projection avant de s'assurer que les deux couches aient la même
Point1_sf <- st_transform(Point1_sf, st_crs(Autoroutes))
## Trouver le tronçon autoroutier le plus proche
PlusProche <- st_nearest_feature(Point1_sf, Autoroutes)
print(PlusProche)
Point1_sf$AutoroutePlusProche <- as.numeric(st_distance(Point1_sf,
                                                        Autoroutes[PlusProche,]))
cat("Distance à l'autoroute la plus proche :", Point1_sf$AutoroutePlusProche, "m.")
```

```
## Zone tampon
ZoneTampon <- st_buffer(Point1_sf, Point1_sf$AutoroutePlusProche)
## Cartographie
tmap_mode("view")
tm_shape(ZoneTampon)+
  tm_borders(col= "black")+
tm_shape(Autoroutes)+
  tm_lines(col="red")+
tm_shape(Point1_sf)+
  tm_dots(col= "blue", shape=21, size = .2)
```

12.2 Exercices du chapitre 2

12.2.1 Exercice 1

A. Complétez les matrices de contiguité pour les entités spatiales ci-dessous.



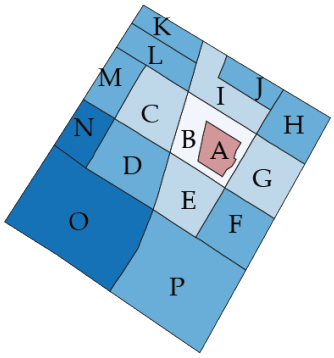
Matrice de contiguïté selon le partage d'un segment

	A	B	C	D	E	F	G
A	--	1	0	1	0	1	1
B	1	--	1	0	0	0	1
C	0	1	--	1	0	0	0
D	1	0	1	--	1	0	0
E	0	0	0	1	--	1	0
F	1	0	0	0	1	--	1
G	1	1	0	0	0	1	--

Matrice de contiguïté selon le partage d'un nœud

	A	B	C	D	E	F	G
A	--	1	1	1	1	1	1
B	1	--	1	1	0	0	1
C	1	1	--	1	0	0	0
D	1	0	1	--	1	1	0
E	1	0	0	1	--	1	0
F	1	0	0	0	1	--	1
G	1	1	0	0	0	1	--

B. Trouvez les polygones contigus aux différents ordres d'adjacence et selon le partage d'un segment pour le polygone A.



Polygones adjacents

Ordre 1 : B

Ordre 2 : C, E, I, G

Ordre 3 : D, F, J, H, M, K, L, P

Ordre 4 : O, N

FIGURE 12.1 – Exercice sur la contiguïté et les ordres d'adjacence

12.2.2 Exercice 2

```
library(sf)
library(spdep)
library(tmap)
## Importation de la couche des secteurs de recensement
SRQc <- st_read(dsn = "data/chap02/exercice/RMRQuebecSR2021.shp", quiet=TRUE)

## Matrice selon le partage d'un segment (Rook)
Rook <- poly2nb(SRQc, queen=FALSE)
W.Rook <- nb2listw(Rook, zero.policy=TRUE, style = "W")

## Coordonnées des centroïdes des entités spatiales
coords <- st_coordinates(st_centroid(SRQc))

## Matrices de l'inverse de la distance
# Trouver le plus proche voisin
k1 <- knn2nb(knearneigh(coords))
plusprochevoisin.max <- max(unlist(nbdists(k1,coords)))
# Voisins les plus proches avec le seuil de distance maximal
Voisins.DistMax <- dnearneigh(coords, 0, plusprochevoisin.max)
# Distances avec le seuil maximum
distances <- nbdists(Voisins.DistMax, coords)
# Inverse de la distance au carré
InvDistances2 <- lapply(distances, function(x) (1/x^2))
## Matrices de pondérations spatiales standardisées en ligne
W_InvDistances2Reduite <- nb2listw(Voisins.DistMax, glist = InvDistances2, style = "W")

## Matrice des plus proches voisins avec k = 2
k2 <- knn2nb(knearneigh(coords, k = 2))
W.k2 <-  nb2listw(k2, zero.policy=FALSE, style = "W")
```

12.2.3 Exercice 3

```
library(sf)
library(spdep)
library(tmap)
## Cartographie de la variable
tm_shape(SRQc)+
  tm_polygons(col="D1pct", title = "Premier décile de revenu (%)",
              style="quantile", n=5, palette="Greens")+
  tm_layout(frame = F)+tm_scale_bar(c(0,5,10))

## I de Moran avec la méthode Monte-Carlo avec 999 permutations
```

```
# utilisez la fonction moran.mc
# avec la matrice W.Rook
moran.mc(SRQc$D1pct, listw=W.Rook, zero.policy=TRUE,  nsim=999)
# avec la matrice W_InvDistances2
moran.mc(SRQc$D1pct, listw=W_InvDistances2Reduite, zero.policy=TRUE,  nsim=999)
# avec la matrice W.k2
moran.mc(SRQc$D1pct, listw=W.k2, zero.policy=TRUE,  nsim=999)
```

Les valeurs du I de Moran sont les suivantes : 0,69 pour la matrice *Rook*, 0,52 pour la matrice inverse de la distance au carré réduite et 0,75 pour la matrice selon le critère des deux plus proches voisins.

12.2.4 Exercice 4

```
####################
## Calcul du Z(Gi)
####################
SRQc$D1pct_localGetis <- localG(SRQc$D1pct,
                                W.Rook,
                                zero.policy=TRUE)
# Définition des intervalles et des noms des classes
classes.intervalles = c(-Inf, -3.29, -2.58, -1.96, 1.96, 2.58, 3.29, Inf)
classes.noms = c("Point froid (p = 0,001)",
                 "Point froid (p = 0,01)",
                 "Point froid (p = 0,05)",
                 "Non significatif",
                 "Point chaud (p = 0,05)",
                 "Point chaud (p = 0,01)",
                 "Point chaud (p = 0,001)")

## Création d'un champ avec les noms des classes
SRQc$D1pct_localGetisP <- cut(SRQc$D1pct_localGetis,
                              breaks = classes.intervalles,
                              labels = classes.noms)
## Cartographie
tm_shape(SRQc)+
  tm_polygons(col ="D1pct_localGetisP",
              title="Z(Gi)", palette="-RdBu", lwd = 1)+
  tm_layout(frame =F)

####################
## Typologie LISA
####################
## Cote Z (variable centrée réduite)
zx <- (SRQc$D1pct - mean(SRQc$D1pct))/sd(SRQc$D1pct)
```

```
## variable X centrée réduite spatialement décalée avec une matrice Rook
wzx <- lag.listw(W.Rook, zx)
## I de Moran local (notez que vous pouvez aussi utiliser la fonction localmoran_perm)
localMoranI  <- localmoran(SRQc$D1pct, W.Rook)
plocalMoranI <- localMoranI[, 5]
## Choisir un seuil de signification
signif = 0.05
## Construction de la typologie
Typologie <- ifelse(zx > 0 & wzx > 0, "1. HH", NA)
Typologie <- ifelse(zx < 0 & wzx < 0, "2. LL", Typologie)
Typologie <- ifelse(zx > 0 & wzx < 0, "3. HL", Typologie)
Typologie <- ifelse(zx < 0 & wzx > 0, "4. LH", Typologie)
Typologie <- ifelse(plocalMoranI > signif, "Non sign", Typologie)  # Non significatif
## Enregistrement de la typologie dans un champ
SRQc$TypoIMoran.D1pct <- Typologie
## Couleurs
Couleurs <- c("red", "blue", "lightpink", "skyblue2", "lightgray")
names(Couleurs) <- c("1. HH","2. LL","3. HL","4. LH","Non sign")
## Cartographie
tmap_mode("plot")
tm_shape(SRQc) +
  tm_polygons(col = "TypoIMoran.D1pct", palette = Couleurs,
              title ="Autocorrélation spatiale locale")+
  tm_layout(frame = FALSE)
```

12.3 Exercices du chapitre 3

12.3.1 Exercice 1

```
library(sf)
library(tmap)
## Importation des données
Arrondissements <-  st_read(dsn = "data/chap03/Arrondissements.shp", quiet=TRUE)
Incidents <- st_read(dsn = "data/chap03/IncidentsSecuritePublique.shp", quiet=TRUE)
## Changement de projection
Arrondissements <- st_transform(Arrondissements, crs = 3798)
Incidents <- st_transform(Incidents, crs = 3798)
## Couche pour les accidents
Accidents <- subset(Incidents, Incidents$DESCRIPTIO %in%
                      c("Accident avec blessés", "Accident mortel"))
## Coordonnées et projection cartographique
xy <- st_coordinates(Accidents)
ProjCarto <- st_crs(Accidents)
```

```
## Centre moyen
CentreMoyen <- data.frame(X = mean(xy[,1]),
                          Y = mean(xy[,2]))
CentreMoyen <- st_as_sf(CentreMoyen, coords = c("X", "Y"), crs = ProjCarto)
# Distance standard combiné
CentreMoyen$DS <- c(sqrt(mean((xy[,1] - mean(xy[,1]))**2 +
                              (xy[,2] - mean(xy[,2]))**2)))
CercleDS <- st_buffer(CentreMoyen, dist = CentreMoyen$DS)
head(CercleDS)
```

12.3.2 Exercice 2

```
library(sf)
library(tmap)
## Importation des données
SR <- st_read(dsn = "data/chap03/Recen2021Sherbrooke.gpkg",
              layer = "DR_SherbSRDonnees2021", quiet=TRUE)
## Couche pour les accidents pour l'année 2021
Acc2021 <- subset(Incidents, Incidents$DESCRIPTIO %in%
                    c("Accident avec blessés", "Accident mortel")
                  & ANNEE==2021)
## Nous nous assurons que les deux couches aient la même projection cartographique
SR <- st_transform(SR, st_crs(Acc2021))
## Calcul du nombre d'incidents par SR
SR$Acc2021 <- lengths(st_intersects(SR, Acc2021))
## Calcul du nombre de méfaits pour 1000 habitants
SR$DensiteMAcc2021Hab <- SR$Acc2021 / (SR$SRpop_2021 / 1000)
## Cartographie
tm_shape(SR)+
  tm_polygons(col="Acc2021", style="pretty",
              title="Nombre pour 1000 habitants",
              border.col = "black", lwd = 1)+
  tm_bubbles(size = "DensiteMAcc2021Hab", border.col = "black", alpha = .5,
             col = "aquamarine3", title.size = "Nombre", scale = 1.5)+
  tm_layout(frame = FALSE)+tm_scale_bar(text.size = .5, c(0, 5, 10))
```

12.3.3 Exercice 3

```
library(sf)
library(spatstat)
library(tmap)
library(terra)
```

```
## Importation des données
Arrondissements <-  st_read(dsn = "data/chap03/Arrondissements.shp", quiet=TRUE)
Incidents <- st_read(dsn = "data/chap03/IncidentsSecuritePublique.shp", quiet=TRUE)
## Changement de projection
Arrondissements <- st_transform(Arrondissements, crs = 3798)
Incidents <- st_transform(Incidents, crs = 3798)
## Couche pour les méfaits pour l'année 2021
M2021 <- subset(Incidents, DESCRIPTIO == "Méfait" & ANNEE==2021)
## Pour accélérer les calculs, nous retenons uniquement l'arrondissement des Nations
# Couche pour l'arrondissement des Nations
ArrDesNations <- subset(Arrondissements, NOM == "Arrondissement des Nations")
# Sélection des accidents localisés dans l'arrondissement Des Nations
RequeteSpatiale <- st_intersects(M2021, ArrDesNations, sparse = FALSE)
M2021$Nations <- RequeteSpatiale[, 1]
M2021Nations <- subset(M2021, M2021$Nations == TRUE)

## Conversion des données sf dans le format de spatstat
# la fonction as.owin est utilisée pour définir la fenêtre de travail
fenetre <- as.owin(ArrDesNations)
## Conversion des points au format ppp pour les différentes années
M2021.ppp <- ppp(x = st_coordinates(M2021Nations)[,1],
                 y = st_coordinates(M2021Nations)[,2],
                 window = fenetre, check = T)

## Kernel quadratique avec un rayon de 500 mètres et une taille de pixel de 50 mètres
kdeQ <- density.ppp(M2021.ppp, sigma=500, eps=50, kernel="quartic")
## Conversion en raster
RkdeQ <- terra::rast(kdeQ)*1000000
## Projection cartographique
crs(RkdeQ) <- "epsg:3857"
## Visualisation des résultats
tmap_mode("plot")
  tm_shape(RkdeQ) + tm_raster(style = "cont", palette="Reds", title = "Gaussien")+
  tm_shape(M2021Nations) + tm_dots(col = "black", size = 0.01)+
  tm_shape(ArrDesNations) + tm_borders(col = "black", lwd = 3)+
  tm_layout(frame = F)
```

12.4 Exercices du chapitre 4

12.4.1 Exercice 1

```
library(sf)
library(tmap)
```

```
library(dbscan)
library(ggplot2)
## Importation des données
Collissions <- st_read(dsn = "data/chap04/collisions.gpkg",
                       layer = "CollisionsRoutieres",
                       quiet = T)
## Collisions impliquant au moins une personne à vélo en 2020 et 2021
Coll.Velo <- subset(Collissions,
                    Collissions$NB_VICTIMES_VELO > 0 &
                      Collissions$AN %in% c(2020, 2021))
## Coordonnées géographiques
xy <- st_coordinates(Coll.Velo)
## Graphique pour la distance au quatrième voisin le plus proche
DistKplusproche <- kNNdist(xy, k = 4)
DistKplusproche <- as.data.frame(sort(DistKplusproche, decreasing = FALSE))
names(DistKplusproche) <- "distance"
ggplot(data = DistKplusproche)+
  geom_path(aes(x = 1:nrow(DistKplusproche), y = distance), size=1)+
  labs(x = "Points triés par ordre croissant selon la distance",
       y = "Distance au quatrième point le plus proche")+
  geom_hline(yintercept=250, color = "#08306b", linetype="dashed", size=1)+
  geom_hline(yintercept=500, color = "#00441b", linetype="dashed", size=1)+
  geom_hline(yintercept=1000, color = "#67000d", linetype="dashed", size=1)
## DBSCAN avec les quatre distances
set.seed(123456789)
dbscan250  <- dbscan(xy, eps = 250, minPts = 4)
dbscan500  <- dbscan(xy, eps = 500, minPts = 4)
dbscan1000 <- dbscan(xy, eps = 1000, minPts = 4)
## Affichage des résultats
dbscan250
dbscan500
dbscan1000
## Enregistrement dans la couche de points sf Coll.Velo
Coll.Velo$dbscan250 <- as.character(dbscan250$cluster)
Coll.Velo$dbscan500 <- as.character(dbscan500$cluster)
Coll.Velo$dbscan1000 <- as.character(dbscan1000$cluster)

Coll.Velo$dbscan250 <- ifelse(nchar(Coll.Velo$dbscan250) == 1,
                              paste0("0", Coll.Velo$dbscan250),
                              Coll.Velo$dbscan250)
Coll.Velo$dbscan500 <- ifelse(nchar(Coll.Velo$dbscan500) == 1,
                               paste0("0", Coll.Velo$dbscan500),
                               Coll.Velo$dbscan500)
Coll.Velo$dbscan1000 <- ifelse(nchar(Coll.Velo$dbscan1000) == 1,
                               paste0("0", Coll.Velo$dbscan1000),
                               Coll.Velo$dbscan1000)
```

```
## Extraction des agrégats
Agregats.dbscan250  <- subset(Coll.Velo, dbscan250 != "00")
Agregats.dbscan500  <- subset(Coll.Velo, dbscan500 != "00")
Agregats.dbscan1000 <- subset(Coll.Velo, dbscan1000 != "00")
## Cartographie des résultats
tmap_mode("view")
tm_shape(Agregats.dbscan250)+tm_dots(col="dbscan250", size = .05)
tm_shape(Agregats.dbscan500)+tm_dots(col="dbscan500", size = .05)
tm_shape(Agregats.dbscan1000)+tm_dots(col="dbscan1000", size = .05)
```

12.4.2 Exercice 2

```
library(sf)
library(tmap)
library(dbscan)
library(ggplot2)
## Importation des données
Collissions <- st_read(dsn = "data/chap04/collisions.gpkg", layer = "CollisionsRoutieres")
## Collisions impliquant au moins une personne à vélo en 2020 et 2021
Coll.Velo <- subset(Collissions,
                    Collissions$NB_VICTIMES_VELO > 0 &
                      Collissions$AN %in% c(2020, 2021))
## Coordonnées géographiques
xy <- st_coordinates(Coll.Velo)
Coll.Velo$x <- xy[,1]
Coll.Velo$y <- xy[,2]
## Conversion du champ DT_ACCDN au format Date
Coll.Velo$DT_ACCDN <- as.Date(Coll.Velo$DT_ACCDN)
## ST-DBSCAN avec eps1 = 500, esp2 = 30 et minpts = 4
Resultats.stdbscan <- stdbscan(x = Coll.Velo$x,
                               y = Coll.Velo$y,
                               time = Coll.Velo$DT_ACCDN,
                               eps1 = 500,
                               eps2 = 30,
                               minpts = 4)
## Enregistrement des résultats ST-DBSCAN dans la couche de points sf
Coll.Velo$stdbscan <- as.character(Resultats.stdbscan$cluster)
Coll.Velo$stdbscan <- ifelse(nchar(Coll.Velo$stdbscan) == 1,
                             paste0("0", Coll.Velo$stdbscan),
                             Coll.Velo$stdbscan)
## Nombre de points par agrégat avec la fonction table
table(Coll.Velo$stdbscan)
## Sélection des points appartenant à un agrégat avec la fonction subset
Agregats <- subset(Coll.Velo, stdbscan != "00")
```

```
## Conversion de la date au format POSIXct
Agregats$dtPOSIXct <- as.POSIXct(Agregats$DT_ACCDN, format = "%Y/%m/%d")
## Tableau récapitulatif
library("dplyr")
Tableau.stdbscan <-
  st_drop_geometry(Agregats) %>%
  group_by(stdbscan) %>%
  summarize(points = n(),
            date.min = min(DT_ACCDN),
            date.max = max(DT_ACCDN),
            intervalle.jours = as.numeric(max(DT_ACCDN)-min(DT_ACCDN)))
## Affichage du tableau
print(Tableau.stdbscan, n = nrow(Tableau.stdbscan))
## Construction du graphique
ggplot(Agregats) +
  geom_point(aes(x = dtPOSIXct,
                 y = stdbscan,
                 color = stdbscan),
             show.legend = FALSE) +
  scale_x_datetime(date_labels = "%Y/%m")+
  labs(x= "Temps",
       y= "Identifiant de l'agrégat",
       title = "ST-DBSCAN avec Esp1 = 1000, Esp2 = 21 et MinPts = 4")
## Création d'une couche pour les agrégats
stdbcan.Agregats <- subset(Coll.Velo, stdbscan != "00")
## Cartographie
tmap_mode("view")
  tm_shape(stdbcan.Agregats)+
    tm_dots(shape = 21, col="stdbscan", size=.025, title = "Agrégat")
```

12.5 Exercices du chapitre 5

12.5.1 Exercice 1

```
library(sf)
library(tmap)
library(r5r)

setwd("data/chap05/Laval")
rJava::.jinit()
options(java.parameters = "-Xmx2G")

# 1. Construction du réseau
```

```r
dossierdata <- paste0(getwd(),"/_DataReseau")
list.files(dossierdata)
r5r_core <- setup_r5(data_path = dossierdata,
                     elevation = "TOBLER",
                     verbose = FALSE, overwrite = FALSE)

# 2. Création de deux points
Pts <- data.frame(id = c("Station Morency", "Adresse 1"),
                  lon = c(-73.7199, -73.7183),
                  lat = c(45.5585,  45.5861))
Pts <- st_as_sf(Pts, coords = c("lon","lat"), crs = 4326)
StationMorency <- Pts[1,]
Adresse1 <- Pts[2,]

## 2.1. Trajets en automobile
Auto.1 <- detailed_itineraries(r5r_core = r5r_core,
                               origins = Adresse1,
                               destinations = StationMorency,
                               mode = "CAR",
                               shortest_path = FALSE,
                               drop_geometry = FALSE)
Auto.2 <- detailed_itineraries(r5r_core = r5r_core,
                               origins = StationMorency,
                               destinations = Adresse1,
                               mode = "CAR",
                               shortest_path = FALSE,
                               drop_geometry = FALSE)
## 2.2. Trajets en vélo
velo.1 <- detailed_itineraries(r5r_core = r5r_core,
                               origins = StationMorency,
                               destinations = Adresse1,
                               mode = "BICYCLE",
                               bike_speed = 12,  # par défaut 12
                               shortest_path = FALSE,
                               drop_geometry = FALSE)
velo.2 <- detailed_itineraries(r5r_core = r5r_core,
                               origins = Adresse1,
                               destinations = StationMorency,
                               mode = "BICYCLE",
                               bike_speed = 12,  # par défaut 12
                               shortest_path = FALSE,
                               drop_geometry = FALSE)
## 2.3. Trajets à pied
marche.1 <- detailed_itineraries(r5r_core = r5r_core,
                                 origins = StationMorency,
                                 destinations = Adresse1,
```

```
                                mode = "WALK",
                                walk_speed = 4.5,  # par défaut 3.6
                                shortest_path = FALSE,
                                drop_geometry = FALSE)
marche.2 <- detailed_itineraries(r5r_core = r5r_core,
                                origins = Adresse1,
                                destinations = StationMorency,
                                mode = "WALK",
                                walk_speed = 4.5,  # par défaut 12
                                shortest_path = FALSE,
                                drop_geometry = FALSE)

## 2.4. Trajets en transport en commun
dateheure.matin <- as.POSIXct("12-02-2024 08:00:00",
                              format = "%d-%m-%Y %H:%M:%S")
dateheure.soir  <- as.POSIXct("12-02-2024 18:00:00",
                              format = "%d-%m-%Y %H:%M:%S")
### Définir le temps de marche maximal
minutes_marches_max <- 20
TC.1 <- detailed_itineraries(r5r_core = r5r_core,
                                origins = Adresse1,
                                destinations = StationMorency,
                                mode = c("WALK", "TRANSIT"),
                                max_walk_time = minutes_marches_max,
                                walk_speed = 4.5,
                                departure_datetime = dateheure.matin,
                                shortest_path = FALSE,
                                drop_geometry = FALSE)
TC.2 <- detailed_itineraries(r5r_core = r5r_core,
                                 origins = StationMorency,
                                 destinations = Adresse1,
                                 mode = c("WALK", "TRANSIT"),
                                 max_walk_time = minutes_marches_max,
                                 walk_speed = 4.5,
                                 departure_datetime = dateheure.soir,
                                 shortest_path = FALSE,
                                 drop_geometry = FALSE)



# 4. Cartographie
  # - Map1.Aller : Marche (de la résidence à la station de métro)
  # - Map2.Aller : Vélo (de la résidence à la station de métro)
  # - Map3.Aller : Auto (de la résidence à la station de métro)
  # - Map4.Aller : Transport en commun (de la résidence à la station de métro)
tmap_mode(view)
Map1.Aller <- tm_shape(marche.1)+tm_lines(col="mode", lwd = 3,
```

```
                                      popup.vars = c("mode", "from_id", "to_id",
                                                     "segment_duration", "distance",
                                                     "total_duration", "total_distance"))+
              tm_shape(Adresse1)+tm_dots(col="green", size = .15)+
              tm_shape(StationMorency)+tm_dots(col="red", size = .15)

Map2.Aller <- tm_shape(velo.1)+tm_lines(col="mode", lwd = 3,
                                        popup.vars = c("mode", "from_id", "to_id",
                                                       "segment_duration", "distance",
                                                       "total_duration", "total_distance"))+
              tm_shape(Adresse1)+tm_dots(col="green", size = .15)+
              tm_shape(StationMorency)+tm_dots(col="red", size = .15)

Map3.Aller <- tm_shape(Auto.1)+tm_lines(col="mode", lwd = 3,
                                       popup.vars = c("mode", "from_id", "to_id",
                                                      "segment_duration", "distance",
                                                      "total_duration", "total_distance"))+
              tm_shape(Adresse1)+tm_dots(col="green", size = .15)+
              tm_shape(StationMorency)+tm_dots(col="red", size = .15)

Map4.Aller <- tm_shape(TC.1)+tm_lines(col="mode", lwd = 3,
                                       popup.vars = c("mode", "from_id", "to_id",
                                                      "segment_duration", "distance",
                                                      "total_duration", "total_distance"))+
              tm_shape(Adresse1)+tm_dots(col="green", size = .15)+
              tm_shape(StationMorency)+tm_dots(col="red", size = .15)

tmap_arrange(Map1.Aller, Map2.Aller, Map3.Aller, Map4.Aller, ncol = 2, nrow = 2)

## Réaliser une figure avec quatre figures pour les trajets retour :
  # - Map1.Retour : Marche (de la station de métro à la résidence)
  # - Map2.Retour : Vélo (de la station de métro à la résidence)
  # - Map3.Retour : Auto (de la station de métro à la résidence)
  # - Map4.Retour : Transport en commun (de la station de métro à la résidence)

Map1.Retour <- tm_shape(marche.2)+tm_lines(col="mode", lwd = 3,
                                        popup.vars = c("mode", "from_id", "to_id",
                                                       "segment_duration", "distance",
                                                       "total_duration", "total_distance"))+
  tm_shape(Adresse1)+tm_dots(col="red", size = .15)+
  tm_shape(StationMorency)+tm_dots(col="green", size = .15)

Map2.Retour <- tm_shape(velo.2)+tm_lines(col="mode", lwd = 3,
                                       popup.vars = c("mode", "from_id", "to_id",
                                                      "segment_duration", "distance",
                                                      "total_duration", "total_distance"))+
```

```
  tm_shape(Adresse1)+tm_dots(col="red", size = .15)+
  tm_shape(StationMorency)+tm_dots(col="green", size = .15)

Map3.Retour <- tm_shape(Auto.2)+tm_lines(col="mode", lwd = 3,
                                         popup.vars = c("mode", "from_id", "to_id",
                                                        "segment_duration", "distance",
                                                        "total_duration", "total_distance"))+
  tm_shape(Adresse1)+tm_dots(col="red", size = .15)+
  tm_shape(StationMorency)+tm_dots(col="green", size = .15)

Map4.Retour <- tm_shape(TC.2)+tm_lines(col="mode", lwd = 3,
                                       popup.vars = c("mode", "from_id", "to_id",
                                                      "segment_duration", "distance",
                                                      "total_duration", "total_distance"))+
  tm_shape(Adresse1)+tm_dots(col="red", size = .15)+
  tm_shape(StationMorency)+tm_dots(col="green", size = .15)

tmap_arrange(Map1.Retour, Map2.Retour, Map3.Retour, Map4.Retour, ncol = 2, nrow = 2)

# 5. Arrêt de java
r5r::stop_r5(r5r_core)
rJava::.jgc(R.gc = TRUE)
```

12.5.2 Exercice 2

```
## Construction du réseau
setwd("data/chap05/Laval")
rJava::.jinit()
options(java.parameters = "-Xmx2G")
dossierdata <- paste0(getwd(),"/_DataReseau")
list.files(dossierdata)
r5r_core <- setup_r5(data_path = dossierdata,
                     elevation = "TOBLER",
                     verbose = FALSE, overwrite = FALSE)

## Point pour la Station Morency
StationMorency <- data.frame(id = "Station Morency",
                             lon = -73.7199,
                             lat = 45.5585,  45.5861)
StationMorency <- st_as_sf(StationMorency,
                           coords = c("lon","lat"), crs = 4326)

# 1. Calcul d'isochrones à pied de 5, 10 et 15 minutes
Iso.Marche <- isochrone(r5r_core = r5r_core,
```

```
                        origins = StationMorency,
                        mode = "WALK",
                        cutoffs = c(5, 10, 15),
                        sample_size = .8,
                        time_window = 120,
                        progress = FALSE)
# 1.2. Isochrone à vélo de 5, 10 et 15 minutes
Iso.Velo <- isochrone(r5r_core = r5r_core,
                      origins = StationMorency,
                      mode = "BICYCLE",
                      cutoffs = c(10, 20, 30),
                      sample_size = .8,
                      time_window = 120,
                      progress = FALSE)

# 3. Cartographie les résultats
tmap_mode("view")
tmap_options(check.and.fix = TRUE)
Carte.Marche <- tm_shape(Iso.Marche)+
                    tm_fill(col="isochrone",
                            alpha = .4,
                            breaks = c(0, 5, 10, 15),
                            title ="Marche",
                            legend.format = list(text.separator = "à"))+
                    tm_shape(StationMorency)+tm_dots(col="darkred", size = .25)

Carte.Velo <- tm_shape(Iso.Velo)+
                    tm_fill(col="isochrone",
                            alpha = .4,
                            breaks = c(0, 5, 10, 15),
                            title ="Vélo",
                            legend.format = list(text.separator = "à"))+
                    tm_shape(StationMorency)+tm_dots(col="darkred", size = .25)


tmap_arrange(Carte.Marche, Carte.Velo, ncol = 2)

# 4. Arrêt de java
r5r::stop_r5(r5r_core)
rJava::.jgc(R.gc = TRUE)
```

12.6 Exercices du chapitre 6

12.6.1 Exercice 1

```
library(sf)
library(spNetwork)
library(future)

future::plan(future::multisession(workers = 5))
# Importation des données sur les collisions cycles et le réseau de rues
Collisions <- st_read(dsn = "data/chap06/Mtl/DonneesMTL.gpkg", layer="CollisionsAvecCyclistes", quiet=TRUE)
ReseauRues <- st_read(dsn = "data/chap06/Mtl/DonneesMTL.gpkg", layer="Rues", quiet=TRUE)
ReseauRues$LineID <- 1:nrow(ReseauRues)
LongueurKm <- sum(as.numeric(st_length(ReseauRues)))/1000
Collisions <- st_transform(Collisions, st_crs(ReseauRues))
cat("Informations sur les couches",
    "\n  Collisions avec cylistes :", nrow(Collisions),
    "\n  Réseau :", round(LongueurKm,3), "km")
# Cartographie
tmap_mode("view")
tm_shape(ReseauRues) + tm_lines("black") +
  tm_shape(Collisions) + tm_dots("blue", size = 0.025)+
tm_scale_bar(c(0,1,2), position = 'left')+
  tm_layout(frame = FALSE)


## Évaluation des bandwidths de 100 à 1200 avec un saut de 50
eval_bandwidth <- bw_cv_likelihood_calc.mc(
  bw_range = c(100,1200),
  bw_step = 50,
  lines = ReseauRues,
  events = Collisions,
  w = rep(1, nrow(Collisions)),
  kernel_name = 'quartic',
  method = 'discontinuous',
  adaptive = FALSE,
  max_depth = 10,
  digits = 1,
  tol = 0.1,
  agg = 5,
  grid_shape = c(5,5),
  verbose = TRUE)

## Graphique pour les bandwidths
ggplot(eval_bandwidth) +
```

```
  geom_path(aes(x = bw, y = cv_scores)) +
  geom_point(aes(x = bw, y = cv_scores), color = 'red')+
  labs(x = "Valeur de la bandwidth", y = "Valeur du CV")
```

12.6.2 Exercice 2

```
library(sf)
library(spNetwork)
library(future)
## Création des lixels d'une longueur de 100 mètres
lixels <- lixelize_lines(ReseauRues, 100, mindist = 50)
lixels_centers <- spNetwork::lines_center(lixels)
## Calcul de la NKDE continue
intensity <- nkde.mc(lines = ReseauRues,
                     events = Collisions,
                     w = rep(1, nrow(Collisions)),
                     samples = lixels_centers,
                     kernel_name = 'quartic',
                     bw = 500,
                     adaptive = FALSE,
                     method = 'continuous',
                     max_depth = 8,
                     digits = 1,
                     tol = 0.1,
                     agg = 5,
                     verbose = FALSE,
                     grid_shape = c(5,5))
lixels$density <- intensity * 1000
## Cartographie
tm_shape(lixels) +
  tm_lines("density", lwd = 1.5, n = 7, style = "fisher",
           legend.format = list(text.separator = "à"))+
  tm_layout(frame=FALSE)
```

12.7 Exercices du chapitre 7

12.7.1 Exercice 1

```
library(sf)
library(spatialreg)
# Matrice de contiguïté selon le partage d'un segment (Rook)
```

```
load("data/chap06/DonneesLyon.Rdata")
Rook <- poly2nb(LyonIris, queen=FALSE)
Rook <- poly2nb(LyonIris, queen=FALSE)
W.Rook <- nb2listw(Rook, zero.policy=TRUE, style = "W")
# Modèles
formule <- "PM25 ~ Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed"
Modele.SLX <- lmSLX(formule, listw=W.Rook,  data = LyonIris)  # dataframe
Modele.SAR <- lagsarlm(formule, listw=W.Rook,  data = LyonIris,  type = 'lag')
Modele.SEM <- errorsarlm(formule, listw=W.Rook, data = LyonIris)
Modele.DurbinSpatial <- lagsarlm(formule, listw = W.Rook, data = LyonIris, type = "mixed")
Modele.DurbinErreur <- errorsarlm(formule, listw=W.Rook, data = LyonIris, etype = 'emixed')
# Résultats des modèles
summary(Modele.SLX)
summary(Modele.SAR)
summary(Modele.SEM)
summary(Modele.DurbinSpatial)
summary(Modele.DurbinErreur)
```

12.7.2 Exercice 2

```
library(sf)
library(mgcv)
load("data/chap06/DonneesLyon.Rdata")
# Ajout des coordonnées x et y
xy <- st_coordinates(st_centroid(LyonIris))
LyonIris$X <- xy[,1]
LyonIris$Y <- xy[,2]
# Construction du modèle avec
formule <- "PM25 ~ Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed"
Modele.GAM2 <- gam(PM25 ~  Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed+
                           s(X, Y, k= 40),
                         data = LyonIris)
summary(Modele.GAM2)
```

12.7.3 Exercice 3

```
library(sf)
library(spgwr)
load("data/chap06/DonneesLyon.Rdata")
# Ajout des coordonnées x et y
xy <- st_coordinates(st_centroid(LyonIris))
LyonIris$X <- xy[,1]
```

```
LyonIris$Y <- xy[,2]
# Optimisation du nombre de voisins avec le CV
formule <- "PM25 ~ Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed"
bwaCV.voisins  <- gwr.sel(formule,
                          data = LyonIris,
                          method = "cv",
                          gweight=gwr.bisquare,
                          adapt=TRUE,
                          verbose = FALSE,
                          RMSE = TRUE,
                          longlat = FALSE,
                          coords=cbind(LyonIris$X,LyonIris$Y))
# Optimisation du nombre de voisins avec l'AIC
formule <- "PM25 ~ Pct0_14+Pct_65+Pct_Img+Pct_brevet+NivVieMed"
bwaCV.voisins  <- gwr.sel(formule,
                          data = LyonIris,
                          method = "AIC",
                          gweight=gwr.bisquare,
                          adapt=TRUE,
                          verbose = FALSE,
                          RMSE = TRUE,
                          longlat = FALSE,
                          coords=cbind(LyonIris$X,LyonIris$Y))
# Réalisation de la GWR
Modele.GWR <- gwr(formule,
              data = LyonIris,
              adapt=bwaCV.voisins,
              gweight=gwr.bisquare,
              hatmatrix=TRUE,
              se.fit=TRUE,
              coords=cbind(LyonIris$X,LyonIris$Y),
              longlat=F)
# Affichage des résultats
Modele.GWR
```

12.8 Exercices du chapitre 8

12.8.1 Exercice 1

```
library(rgeoda)
library(sf)
library(tmap)
## Préparation des données
load("data/Lyon.Rdata")
```

```r
VarSocioEco <- c("Pct0_14", "Pct_65", "Pct_Img", "Pct_brevet", "NivVieMed")
Data2 <- st_drop_geometry(LyonIris[VarSocioEco])
queen_w <- queen_weights(LyonIris)
## Classification avec k = 4
azp5_sa  <- azp_sa(p=4, w=queen_w, df=Data2, cooling_rate = 0.85)
azp5_tab <- azp_tabu(p=4, w=queen_w, df=Data2, tabu_length = 10, conv_tabu = 10)
skater5  <- rgeoda::skater(k=4, w=queen_w, df=Data2)
redcap5  <- redcap(k = 4, w = queen_w, df = Data2, method = "fullorder-wardlinkage")
## Cartographie des résultats
LyonIris$SE.azp4_sa  <- as.character(azp5_tab$Clusters)
LyonIris$SE.azp4_tab <- as.character(azp5_sa$Clusters)
LyonIris$SE.skater4 <- as.character(skater5$Clusters)
LyonIris$SE.recap4  <- as.character(redcap5$Clusters)
Carte1 <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
  tm_fill(col="SE.azp4_sa", palette = "Set1", title ="")+
  tm_layout(frame=FALSE,  main.title = "a. AZP-SA",
            main.title.position = "center", main.title.size = 1)
Carte2 <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
  tm_fill(col="SE.azp4_tab", palette = "Set1", title ="")+
  tm_layout(frame=FALSE,  main.title = "b. AZP-TABU",
            main.title.position = "center", main.title.size = 1)
Carte3 <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
  tm_fill(col="SE.skater4", palette = "Set1", title ="")+
  tm_layout(frame=FALSE,  main.title = "c. Skater",
            main.title.position = "center", main.title.size = 1)
Carte4 <- tm_shape(LyonIris)+tm_borders(col="gray", lwd=.5)+
  tm_fill(col="SE.recap4", palette = "Set1", title ="")+
  tm_layout(frame=FALSE,  main.title = "d. RECAP",
            main.title.position = "center", main.title.size = 1)

tmap_arrange(Carte1, Carte2, Carte3, Carte4)
```

Bibliographie

Abramson, Ian S. 1982. « On bandwidth variation in kernel estimates-a square root law. » *The annals of Statistics*: 1217-1223. https://www.jstor.org/stable/2240724.

Anselin, Luc. 1995. « Local indicators of spatial association—LISA. » *Geographical analysis* 27 (2): 93-115. https://doi.org/10.1111/j.1538-4632.1995.tb00338.x.

———. 1996. « The Moran scatterplot as an ESDA tool to Assess Local Instability in Spatial Association. » In *Spatial Analytical Perspectives on GIS*, sous la dir. de Manfred M Fischer, 121-137. Taylor & Francis Group. https://doi.org/10.1201/9780203739051.

———. 2019. « A local indicator of multivariate spatial association: extending Geary's C. » *Geographical Analysis* 51 (2): 133-150. https://doi.org/10.1111/gean.12164.

Anselin, Luc, Anil K Bera, Raymond Florax et Mann J Yoon. 1996. « Simple diagnostic tests for spatial dependence. » *Regional science and urban economics* 26 (1): 77-104. https://doi.org/10.1016/0166-0462(95)02111-6.

Anselin, Luc et Xun Li. 2019. « Operational local join count statistics for cluster detection. » *Journal of geographical systems* 21: 189-210. https://doi.org/10.1007/s10109-019-00299-x.

Anselin, Luc et Sergio J Rey. 2014. *Modern spatial econometrics in practice: A guide to GeoDa, GeoDaSpace and PySAL*. GeoDa Press LLC.

Apparicio, Philippe, Marie-Soleil Cloutier et Richard Shearmur. 2007. « The case of Montreal's missing food deserts: evaluation of accessibility to food supermarkets. » *International journal of health geographics* 6 (1): 1-13. https://doi.org/10.1186/1476-072X-6-4.

Apparicio, Philippe et Dubé Gelb Jérémy. 2025. *Méthodes de régression spatiale : un grand bol d'R*. FabriqueREL, Licence CC BY-SA. https://serieboldr.com/RegressionsSpatiales/.

Apparicio, Philippe et Jérémy Gelb. 2022. *Méthodes quantitatives en sciences sociales : un grand bol d'R*. FabriqueREL, Licence CC BY-SA. https://serieboldr.com/MethodesQuantitatives/.

Apparicio, Philippe, Jérémy Gelb, Anne-Sophie Dubé, Simon Kingham, Lise Gauvin et Éric Robitaille. 2017. « The approaches to measuring the potential spatial access to urban health services revisited: distance types and aggregation-error issues. » *International journal of health geographics* 16 (1): 32. https://doi.org/10.1186/s12942-017-0105-9.

Apparicio, Philippe et Anne-Marie Séguin. 2006. « Measuring the accessibility of services and facilities for residents of public housing in Montreal. » *Urban studies* 43 (1): 187-211. http://dx.doi.org/10.1080/00420980500409334.

Apparicio, Philippe, Anne-Marie Séguin et Xavier Leloup. 2007. « Modélisation spatiale de la pauvreté à Montréal: apport méthodologique de la régression géographiquement pondérée. » *The Canadian Geographer/Le Géographe canadien* 51 (4): 412-427. https://doi.org/10.1111/j.1541-0064.2007.00189.x.

Assunção, Renato M, Marcos Corrêa Neves, Gilberto Câmara et Corina da Costa Freitas. 2006. « Efficient regionalization techniques for socio-economic geographical units using minimum spanning trees. » *International Journal of Geographical Information Science* 20 (7): 797-811. https://doi.org/10.1080/13658810600665111.

Baddeley, Adrian, Ege Rubak et Rolf Turner. 2015. *Spatial point patterns: methodology and applications with R*. CRC press.

Baddeley, Adrian et Rolf Turner. 2005. « spatstat: An R Package for Analyzing Spatial Point Patterns. » *Journal of Statistical Software* 12 (6): 1-42. https://doi.org/10.18637/jss.v012.i06.

Barbonne, Rémy, Paul Villeneuve et Marius Thériault. 2007. « La dynamique spatiale des marchés locaux de l'emploi au sein du champ métropolitain de Québec, 1981–2001. » *The Canadian Geographer/Le Géographe canadien* 51 (3). Wiley Online Library: 303-322. https://doi.org/10.1111/j.1541-0064.2007.00180.x.

Birant, Derya et Alp Kut. 2007. « ST-DBSCAN: An algorithm for clustering spatial–temporal data. » *Data & knowledge engineering* 60 (1): 208-221. https://doi.org/10.1016/j.datak.2006.01.013.

Bivand, Roger, Giovanni Millo et Gianfranco Piras. 2021. « A review of software for spatial econometrics in R. » *Mathematics* 9 (11): 1276. https://doi.org/10.3390/math9111276.

Bivand, Roger, Edzer J Pebesma, Virgilio Gómez-Rubio et Edzer Jan Pebesma. 2008. *Applied spatial data analysis with R*. Vol. 747248717. Springer.

Bivand, Roger, Edzer Pebesma et Virgilio Gomez-Rubio. 2013. *Applied spatial data analysis with R, Second edition*. Springer, New York. https://asdar-book.org/.

Bivand, Roger et David WS Wong. 2018. « Comparing implementations of global and local indicators of spatial association. » *Test* 27 (3): 716-748. https://doi.org/10.1007/s11749-018-0599-x.

Bivand, Roger et Danlin Yu. 2023. *spgwr: Geographically Weighted Regression*. s.n. https://CRAN.R-project.org/package=spgwr.

Brewer, Cynthia A, Geoffrey W Hatchard et Mark A Harrower. 2003. « ColorBrewer in print: a catalog of color schemes for maps. » *Cartography and geographic information science* 30 (1): 5-32. https://doi.org/10.1559/152304003100010929.

Caha, Jan. 2023. *SpatialKDE: Kernel Density Estimation for Spatial Data*. s.n. https://CRAN.R-project.org/package=SpatialKDE.

Cai, Weiling, Songcan Chen et Daoqiang Zhang. 2007. « Fast and robust fuzzy c-means clustering algorithms incorporating local information for image segmentation. » *Pattern recognition* 40 (3): 825-838. https://doi.org/10.1016/j.patcog.2006.07.011.

Campello, Ricardo JGB, Davoud Moulavi et Jörg Sander. 2013. « Density-based clustering based on hierarchical density estimates. » In *Advances in Knowledge Discovery and Data Mining: 17th Pacific-Asia Conference, PAKDD 2013, Gold Coast, Australia, April 14-17, 2013, Proceedings, Part II 17*, 160-172. s.n. http://dx.doi.org/10.1007/978-3-642-37456-2_14.

Chavent, Marie, Vanessa Kuentz-Simonet, Amaury Labenne et Jérôme Saracco. 2018. « ClustGeo: an R package for hierarchical clustering with spatial constraints. » *Computational Statistics* 33 (4): 1799-1822. https://doi.org/10.1007/s00180-018-0791-1.

Cliff, Andrew David et J Keith Ord. 1981. *Spatial processes: models & applications*. Taylor & Francis.

Davies, T. M., J. C. Marshall et M. L. Hazelton. 2018. « Tutorial on kernel estimation of continuous spatial and spatiotemporal relative risk. » *Statistics in Medicine* 37 (7): 1191-1221.

Diggle, Peter. 1985. « A kernel method for smoothing point process data. » *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 34 (2). Wiley Online Library: 138-147.

Dijkstra, Edsger Wybe. 1959. « A note on two problems in connexion with graphs:(Numerische Mathematik, 1 (1959), p 269-271). » Stichting Mathematisch Centrum.

Douglas, David H et Thomas K Peucker. 1973. « Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. » *Cartographica: the international journal for geographic information and geovisualization* 10 (2): 112-122. https://doi.org/10.3138/FM57-6770-U75U-7727.

Dubé, Jean et Dièg0 Legros. 2014. *Econométrie spatiale appliquée des microdonnées*. ISTE Group.

Duque, Juan C, Luc Anselin et Sergio J Rey. 2012. « The max-p-regions problem. » *Journal of Regional Science* 52 (3): 397-419. https://doi.org/10.1111/j.1467-9787.2011.00743.x.

Ester, Martin, Hans-Peter Kriegel, Jörg Sander et Xiaowei Xu. 1996. « A density-based algorithm for discovering clusters in large spatial databases with noise. » In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 96:226-231. 34. s.n. https://dl.acm.org/doi/10.5555/3001460.3001507.

Fotheringham, A Stewart, Chris Brunsdon et Martin Charlton. 2003. *Geographically weighted regression: the analysis of spatially varying relationships*. John Wiley & Sons.

Fotheringham, A Stewart, Ricardo Crespo et Jing Yao. 2015. « Geographical and temporal weighted regression (GTWR). » *Geographical Analysis* 47 (4). Wiley Online Library: 431-452. https://doi.org/10.1111/gean.12071.

Fotheringham, A Stewart, Wenbai Yang et Wei Kang. 2017. « Multiscale geographically weighted regression (MGWR). » *Annals of the American Association of Geographers* 107 (6). Taylor & Francis: 1247-1265. https://doi.org/10.1080/24694452.

2017.1352480.

Garnier, Simon, Noam Ross, Robert Rudis, Antônio Pedro Camargo, Marco Sciaini et Cédric Scherer. 2021. *viridis - Colorblind-Friendly Color Maps for R*. s.n. https://sjmgarnier.github.io/viridis/.

Geary, Robert C. 1954. « The contiguity ratio and statistical mapping. » *The incorporated statistician* 5 (3): 115-146. https://doi.org/10.2307/2986645.

Gelb, Jeremy. 2021. « spNetwork: A Package for Network Kernel Density Estimation. » *The R Journal* 13 (2): 561-577. https://doi.org/10.32614/RJ-2021-102.

Gelb, Jérémy et Philippe Apparicio. 2021. « Apport de la classification floue c-means spatiale en géographie: essai de taxinomie socio-résidentielle et environnementale à Lyon. » *Cybergeo: European Journal of Geography*. https://doi.org/10.4000/cybergeo.36414.

Getis, Arthur. 2009. « Spatial weights matrices. » *Geographical Analysis* 41 (4): 404-410. https://doi.org/10.1111/j.1538-4632.2009.00768.x.

Getis, Arthur et J Keith Ord. 1992. « The analysis of spatial association by use of distance statistics. » *Geographical analysis* 24 (3): 189-206. https://doi.org/10.1111/j.1538-4632.1992.tb00261.x.

Gilardi, Andrea et Robin Lovelace. 2022. *osmextract: Download and Import Open Street Map Data Extracts*. s.n. https://CRAN.R-project.org/package=osmextract.

Giraud, Timothée et Nicolas Lambert. 2016. « cartography: Create and Integrate Maps in your R Workflow. » *JOSS* 1 (4). The Open Journal. https://doi.org/10.21105/joss.00054.

Guagliardo, Mark F. 2004. « Spatial accessibility of primary care: concepts, methods and challenges. » *International journal of health geographics* 3 (1): 1-13. https://doi.org/10.1186/1476-072x-3-3.

Guo, Diansheng. 2008. « Regionalization with dynamically constrained agglomerative clustering and partitioning (REDCAP). » *International Journal of Geographical Information Science* 22 (7): 801-823. https://doi.org/10.1080/13658810600665111.

Hahsler, Michael et Matthew Piekenbrock. 2022. *dbscan: Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and Related Algorithms*. s.n. https://CRAN.R-project.org/package=dbscan.

Hahsler, Michael, Matthew Piekenbrock et Derek Doran. 2019. « dbscan: Fast Density-Based Clustering with R. » *Journal of Statistical Software* 91 (1): 1-30. https://doi.org/10.18637/jss.v091.i01.

Harris, Paul, Chris Brunsdon et Martin Charlton. 2011. « Geographically weighted principal components analysis. » *International Journal of Geographical Information Science* 25 (10). Taylor & Francis: 1717-1736. https://doi.org/10.1080/13658816.2011.554838.

Harrower, Mark et Cynthia A Brewer. 2003. « ColorBrewer.org: an online tool for selecting colour schemes for maps. » *The Cartographic Journal* 40 (1): 27-37. http://dx.doi.org/10.1179/000870403235002042.

Hewko, Jared, Karen E Smoyer-Tomic et M John Hodgson. 2002. « Measuring neighbourhood spatial accessibility to urban amenities: does aggregation error matter? » *Environment and Planning A* 34 (7): 1185-1206. https://doi.org/10.1016/0038-0121(92)90004-O.

Hijmans, Robert J. 2022a. *raster: Geographic Data Analysis and Modeling*. R package version 3.5-15. https://CRAN.R-project.org/package=raster.

———. 2022b. *terra: Spatial Data Analysis*. s.n. https://CRAN.R-project.org/package=terra.

Hollister, Jeffrey, Tarak Shah, Alec L Robitaille, Marcus W Beck et Mike Johnson. 2023. « elevatr: access elevation data from various APIs. » *R package version 0.99.0* 3. https://doi.org/10.5281/zenodo.8335450.

Jain, Anil K et Richard C Dubes. 1988. *Algorithms for clustering data*. Prentice-Hall, Inc.

Jepson, Victoria, Philippe Apparicio et Thi-Thanh-Hien Pham. 2022. « Environmental equity and access to parks in Greater Montreal: an analysis of spatial proximity and potential congestion issues. » *Journal of Urbanism: International Research on Placemaking and Urban Sustainability*: 1-19. http://dx.doi.org/10.1080/17549175.2022.2150271.

Jombart, T, S Devillard, Anne-Béatrice Dufour et D Pontier. 2008. « Revealing cryptic spatial patterns in genetic variability by a new multivariate method. » *Heredity* 101 (1): 92-103. https://doi.org/10.1038/hdy.2008.34.

Kaufman, Leonard. 1990. « Partitioning around medoids (program pam). » *Finding groups in data* 344: 68-125. https:

//doi.org/10.1002/9780470316801.ch2.

Khan, Abdullah A. 1992. « An integrated approach to measuring potential spatial access to health care services. » *Socio-economic planning sciences* 26 (4): 275-287. https://doi.org/10.1016/0038-0121(92)90004-O.

Kulldorff, Martin. 1997. « A spatial scan statistic. » *Communications in Statistics-Theory and methods* 26 (6): 1481-1496. https://doi.org/10.1080/03610929708831995.

Lebart, Ludovic, Alain Morineau et Marie Piron. 1995. *Statistique exploratoire multidimensionnelle*. Vol. 3. Dunod Paris.

Lee, Sang-Il. 2001. « Developing a bivariate spatial association measure: an integration of Pearson's r and Moran's I. » *Journal of geographical systems* 3: 369-385. https://doi.org/10.1007/s101090100064.

LeSage, James P et R. Kelly Pace. 2008. *An introduction to spatial econometrics*. 123. CRC Press.

Levine, Ned. 2006. « Crime mapping and the Crimestat program. » *Geographical analysis* 38 (1): 41-56. https://doi.org/10.1111/j.0016-7363.2005.00673.x.

———. 2021. « CrimeStat IV. » In *The Encyclopedia of Research Methods in Criminology and Criminal Justice*, sous la dir. de JC Barnes et David R Forde, 1:28-32. John Wiley & Sons. https://doi.org/10.1002/9781119111931.ch6.

Li, Xun et Luc Anselin. 2023. *rgeoda: R Library for Spatial Data Analysis*. s.n. https://CRAN.R-project.org/package=rgeoda.

Loader, Clive. 2006. *Local regression and likelihood*. Springer Science & Business Media.

López Castro, Marco Antonio, Marius Thériault et Marie-Hélène Vandersmissen. 2015. « Évolution de la mobilité des membres de familles monoparentales dans la région métropolitaine de Québec de 1996 à 2006: comparaison entre les ménages matricentriques et patricentriques. » *Cahiers de géographie du Québec* 59 (167): 209-250. https://doi.org/10.7202/1036355ar.

Luo, Wei et Yi Qi. 2009. « An enhanced two-step floating catchment area (E2SFCA) method for measuring spatial accessibility to primary care physicians. » *Health & Place* 15 (4). Elsevier: 1100-1107. https://doi.org/10.1016/j.healthplace.2009.06.002.

Luo, Wei et Fahui Wang. 2003. « Measures of spatial accessibility to health care in a GIS environment: synthesis and a case study in the Chicago region. » *Environment and planning B: planning and design* 30 (6): 865-884. https://doi.org/10.1068/b29120.

McGrail, Matthew R. 2012. « Spatial accessibility of primary health care utilising the two step floating catchment area method: an assessment of recent improvements. » *International Journal of Health Geographics* 11. Springer: 1-12. https://doi.org/10.1186/1476-072X-11-50.

McGrail, Matthew R et John S Humphreys. 2009. « Measuring spatial accessibility to primary care in rural areas: Improving the effectiveness of the two-step floating catchment area method. » *Applied geography* 29 (4): 533-541. https://doi.org/10.1016/j.apgeog.2008.12.003.

Mitchel, Andy. 2005. *The ESRI Guide to GIS analysis, Volume 2: Spartial measurements and statistics*. ESRI press.

Moran, Patrick. 1950. « A test for the serial independence of residuals. » *Biometrika* 37 (1/2): 178-181. https://doi.org/10.2307/2332162.

Morgan, Malcolm, Young Marcus, Robin Lovelace et Layik Hama. 2019. « OpenTripPlanner for R. » *Journal of Open Source Software* 4 (44): 1926. https://10.21105/joss.01926.

Naimi, Babak, Nicholas AS Hamm, Thomas A Groen, Andrew K Skidmore, Albertus G Toxopeus et Sara Alibakhshi. 2019. « ELSA: Entropy-based local indicator of spatial association. » *Spatial statistics* 29: 66-88. https://doi.org/10.1016/j.spasta.2018.10.001.

Neuwirth, Erich. 2022. *RColorBrewer: ColorBrewer Palettes*. s.n. https://CRAN.R-project.org/package=RColorBrewer.

Ngui, André Ngamini et Philippe Apparicio. 2011. « Optimizing the two-step floating catchment area method for measuring spatial accessibility to medical clinics in Montreal. » *BMC health services research* 11 (1): 1-12. https://doi.org/10.1186/1472-6963-11-166.

Okabe, Atsuyuki et Kokichi Sugihara. 2012. *Spatial analysis along networks: statistical and computational methods*. John Wiley & Sons.

Openshaw, Stan. 1977. « A geographical solution to scale and aggregation problems in region-building, partitioning and spatial modelling. » *Transactions of the institute of british geographers*: 459-472. https://doi.org/10.2307/622300.

Openshaw, Stan et Liang Rao. 1995. « Algorithms for reengineering 1991 Census geography. » *Environment and planning A* 27 (3): 425-446. https://doi.org/10.1068/a270425.

Orava, Jan. 2011. « K-nearest neighbour kernel density estimation, the choice of optimal k. » *Tatra Mountains Mathematical Publications* 50 (1): 39-50.

Ord, J Keith et Arthur Getis. 1995. « Local spatial autocorrelation statistics: distributional issues and an application. » *Geographical analysis* 27 (4): 286-306. https://doi.org/10.1111/j.1538-4632.1995.tb00912.x.

Pebesma, Edzer. 2018. « Simple Features for R: Standardized Support for Spatial Vector Data. » *The R Journal* 10 (1): 439-446. https://doi.org/10.32614/RJ-2018-009.

Pebesma, Edzer et Roger Bivand. 2005. « Classes and methods for spatial data in R. » *R News* 5 (2): 9-13. https://CRAN.R-project.org/doc/Rnews/.

Penchansky, Roy et J William Thomas. 1981. « The concept of access: definition and relationship to consumer satisfaction. » *Medical care*: 127-140. https://doi.org/10.1097/00005650-198102000-00001.

Pereira, Rafael H. M., Marcus Saraiva, DanielHerszenhut, Carlos Kaue Vieira Braga et Matthew Wigginton Conway. 2021. « r5r: Rapid Realistic Routing on Multimodal Transport Networks with R5 in R. » *Findings*. https://doi.org/10.32866/001c.21262.

Sander, Jörg, Martin Ester, Hans-Peter Kriegel et Xiaowei Xu. 1998. « Density-based clustering in spatial databases: The algorithm gdbscan and its applications. » *Data mining and knowledge discovery* 2: 169-194. https://doi.org/10.1023/A:1009745219419.

Sokal, Robert R, Neal L Oden et Barbara A Thomson. 1998. « Local spatial autocorrelation in a biological model. » *Geographical Analysis* 30 (4): 331-354. https://doi.org/10.1111/j.1538-4632.1998.tb00406.x.

Steenberghen, Thérèse, Koen Aerts et Isabelle Thomas. 2010. « Spatial clustering of events on a network. » *Journal of Transport Geography* 18 (3): 411-418. https://doi.org/10.1016/j.jtrangeo.2009.08.005.

Tennekes, Martijn. 2018. « tmap: Thematic Maps in R. » *Journal of Statistical Software* 84 (6): 1-39. https://doi.org/10.18637/jss.v084.i06.

Tobler, Waldo R. 1970. « A computer movie simulating urban growth in the Detroit region. » *Economic geography* 46 (sup1): 234-240. https://doi.org/10.2307/143141.

Tveite, Håvard. 2020. « The QGIS Standard Deviational Ellipse Plugin. » http://plugins.qgis.org/plugins/SDEllipse/.

Visvalingam, Maheswari et James D Whyatt. 1993. « Line generalisation by repeated elimination of points. » *The cartographic journal* 30 (1): 46-51. https://doi.org/10.3138/FM57-6770-U75U-7727.

Wang, Bin, Wenzhong Shi et Zelang Miao. 2015. « Confidence analysis of standard deviational ellipse and its extension into higher dimensional Euclidean space. » *PloS one* 10 (3): e0118537. https://doi.org/10.1371%2Fjournal.pone.0118537.

Wickham, Hadley. 2016. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org.

Wong, David WS. 1999. « Geostatistics as measures of spatial segregation. » *Urban geography* 20 (7): 635-647. https://doi.org/10.2747/0272-3638.20.7.635.

Wong, David WS et Jay Lee. 2005. *Statistical analysis of geographic information with ArcView GIS and ArcGIS*. Wiley.

Wood, Simon N. 2011. « Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. » *Journal of the Royal Statistical Society Series B: Statistical Methodology* 73 (1): 3-36. https://doi.org/10.1111/j.1467-9868.2010.00749.x.

Yamada, Ikuho et Jean-Claude Thill. 2007. « Local indicators of network-constrained clusters in spatial point patterns. » *Geographical analysis* 39 (3). Wiley Online Library: 268-292. https://www.jstor.org/stable/40645354.