Traitement des données de grande taille avec M

Eric Marcon

Florence Puech

15 mars 2024

Résumé

Cet article méthodologique porte sur une mesure relative de la concentration spatiale en espace continu, la fonction M, proposée par Marcon and Puech (2010). L'objectif de notre contribution est d'apporter des réponses au traitement de jeu de données spatialisées de grande taille avec M sous R. Plus précisément, les deux sources d'incertitude pour le traitement de jeux de données importants sont analysées : l'estimation des erreurs liées à l'approximation de la position géographique des observations et les délais de calcul. Dans un premier temps, nous testons l'impact de l'approximation de la position géographique sur la précision de M et du temps de calcul associé, comme proposé par Tidu et al. (2023). Dans un second temps, les besoins en temps et en mémoire du calcul exact avec le package dbmss sous R sont évalués. Le code R est fourni pour la reproductibilité des résultats.

1 Motivation

L'accès grandissant à d'importants jeux de données individuelles et spatialisées et une plus grande puissance de calcul ont encouragé le développement d'outils d'analyse statistiques permettant de traiter au mieux de telles données [Baddeley2016a]. Des études empiriques à des niveaux géographiques très fins ont ainsi pu été proposées au cours des dernières années pour des jeux de données de grande taille. Une attention particulière a porté sur la détection des structures spatiales (attraction, répulsion, indépendance) de données individuelles spatialisées à partir d'analyses ne reposant plus des données zonées mais sur des données géolocalisées. Ce type d'approche a l'avantage de préserver les positions exactes des entités analysées et par conséquent de ne pas gommer les spécificités individuelles. Diverses études ont montré toute l'importance de retenir une telle méthodologie dans des domaines très variés comme en géographie (**?**, **?**), en économie (**?**, **?**), en écologie (**?**, **?**), en biologie (**?**) etc. Dans un article récent, Tidu et al. (2023) soulignent tout l'intérêt d'une mesure statistique particulière, la fonction M proposée par Marcon and Puech (2010). Cette

mesure, que nous dénommerons M dans la suite de l'article, permet de mettre en évidence et les structures spatiales au sein d'une distribution spatialisée (attraction, répulsion, indépendance) à partir d'une étude fondée sur les distances séparant les entités analysées. Toutefois, si cette mesure permet de préserver toute la richesse des données individuelles géolocalisées, elle nécessite un temps de calcul plus long que d'autres mesures fondées sur les distances puisqu'elle est à la fois une mesure cumulative et relative (voir **?** pour une revue de littérature sur les avantages et les limites d'une dizaine mesures existantes fondées sur les distances). Tidu et al. (2023) proposent de limiter les temps de calculs de M en introduisant une erreur de positionnement volontaire des entités analysées. Ainsi, dans leur étude, les établissements industriels en Sardaigne (Italie) ne sont plus localisés à leur adresse postale exacte mais au centroïde de leur municipalité. Ce repositionnement permet de réduire les temps de calcul car le nombre de distances possibles entre les établissements est, de fait, limité aux distances séparant les centroïdes des communes. Dans notre article, nous proposons dans un premier temps d'étudier la généralisation de cette méthode d'approximation géographique des localisations des entités analysées. Ce travail méthodologique reposant sur un nombre d'emplacements des entités volontairement limité permettra de quantifier l'importance de la détérioration de l'information que cette approche crée. Dans un second temps, nous montrons les avantages de l'utilisation du package *dbmss* (Marcon et al., 2015) pour effectuer ces approximations.

Le plan de l'article est le suivant. Une première section fournit le code nécessaire à la création de jeux de données. Des jeux de points de grande taille (de l'ordre de 100 000 points) complètement aléatoires ou (géographiquement) concentrés sont tirés. L'approximation de leur position consiste à les rassembler au centre des cases d'une grille, selon l'approche de Tidu et al. (2023) qui les positionnent au centre des unités administratives dans lesquelles ils se trouvent. La deuxième section détaille l'utilisation du package *dbmss* pour calculer la fonction M et son intervalle de confiance à partir d'un tableau donnant la position et les caractéristiques des points ou bien une matrice de distances entre eux. La troisième section teste l'impact de l'approximation des points. Enfin, la dernière section mesure la performance de *dbmss* en fonction de la taille du jeu de points.

2 Simulation des données

Les jeux de données que nous allons considérer dans cet article sont obtenus par simulation. Le code R est donné, ce qui permet une parfaite reproductibilité les exemples traités (ou d'en développer d'autres).

2.1 Tirage des points

Un jeu de points est tiré par un processus binomial dans une fenêtre carrée de côté 1. On associe à chaque point une marque qualitative : « Cas » ou « Contrôle ». La majorité des points sont des « Contrôles » et une partie constitue des « Cas », dont la structure spatiale est étudiée. Le poids des points est

tiré dans une loi gamma dont les paramètres de forme et d'échelle sont libres.

Les paramètres sont :

- le nombre de points,
- la proportion de contrôles,
- la forme et l'échelle de la loi gamma.

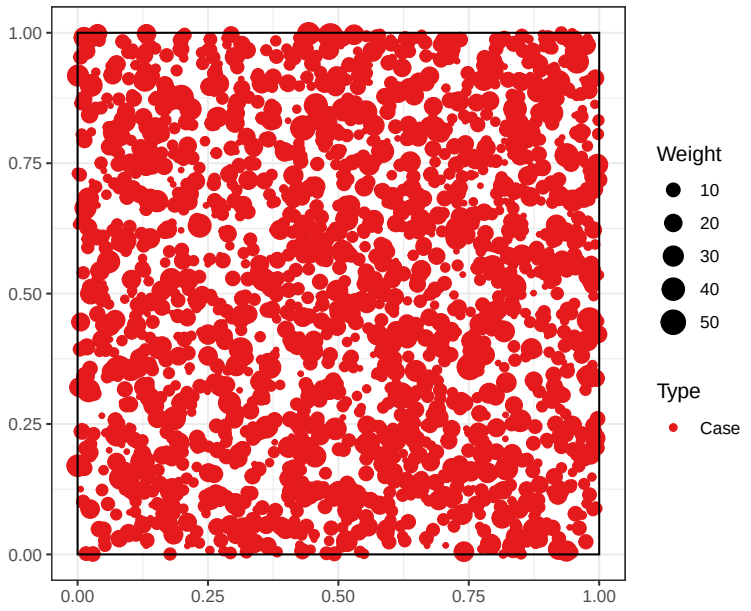
```
par_points_nb <- 40000
par_case_ratio <- 1/20
par_size_gamma_shape <- 0.95
par_size_gamma_scale  <- 10
```

Dans notre exemple, 40 000 points sont tirés et un point sur vingt est un Cas.

La fonction `X_csr()` permet de tirer un semis de points selon des paramètres déterminés. L'argument `points_nb` qui fixe le nombre de points peut être modifié ; les autres paramètres ont leur valeur fixée plus haut.

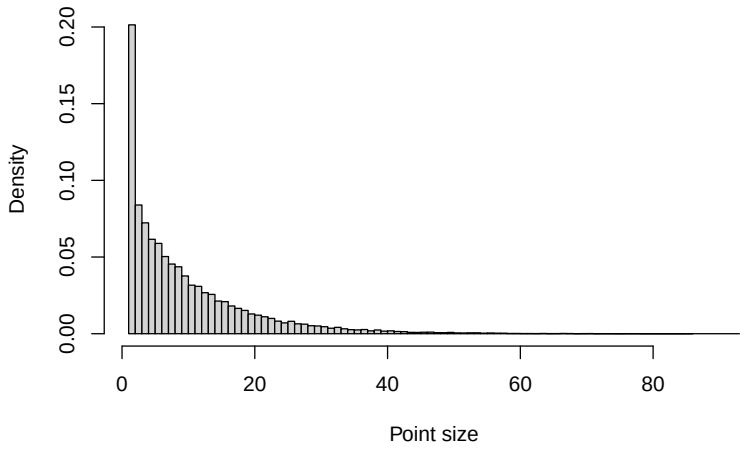
```
library("tidyverse")
library("spatstat")
library("dbmss")
X_csr <- function(
    points_nb,
    case_ratio = par_case_ratio,
    size_gamma_shape = par_size_gamma_shape,
    size_gamma_scale = par_size_gamma_scale) {
  points_nb %>%
    runifpoint() %>%
    as.wmppp() ->
    X
  cases_nb <- round(points_nb *  case_ratio)
  controls_nb <- points_nb - cases_nb
  c(rep("Control", controls_nb), rep("Case", cases_nb)) %>%
    as.factor() ->
    X$marks$PointType
  rgamma(
    X$n,
    shape = size_gamma_shape,
    scale = size_gamma_scale
  ) %>%
    ceiling() ->
    X$marks$PointWeight
  X
}

# Example
X <- X_csr(par_points_nb)
# Map the cases
autoplot(X[X$marks$PointType == "Case"])
```



Seuls les points « Cas » sont représentés sur la fenêtre ci-dessus. La taille des points est une marque quantitative (`Weight`) : graphiquement la surface des points est proportionnelle à la taille des points. Etant donné le grand nombre de Cas sur la fenêtre, la distribution de la taille de ces points n'est que peu lisible. Une analyse complémentaire est nécessaire.

```
# Point size distribution
hist(
  X$marks$PointWeight,
  breaks = unique(X$marks$PointWeight),
  main = "",
  xlab = "Point size"
)
```



Dans cet exemple, le tirage des points est complètement aléatoire (*complete spatial randomness* : CSR), c'est-à-dire qu'il n'y a pas ici de simulation d'attraction ou de dispersion de points qui pourraient générer des concentrations spatiales de points (agrégats) ou, au contraire, des régularités spatiales (dispersions).

La fonction `X_matern()` permet de tirer un semis de points dont les Cas sont concentrés par un processus de **?**. Les paramètres sont :

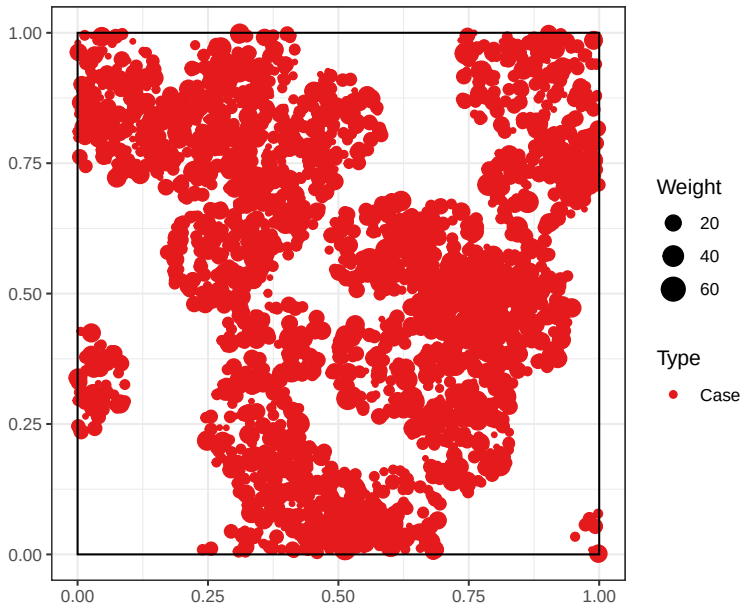
- κ : le nombre d'agrégats attendu,
- `scale` : leur rayon.

```
# Expected number of clusters
par_kappa <- 20
# Cluster radius
par_scale <-  0.1
```

Le code de la fonction est le suivant :

```
X_matern <- function(
    points_nb,
    case_ratio = par_case_ratio,
    kappa = par_kappa,
    scale = par_scale,
    size_gamma_shape = par_size_gamma_shape,
    size_gamma_scale = par_size_gamma_scale) {
  cases_nb <- round(points_nb *  case_ratio)
  controls_nb <- points_nb - cases_nb
  # CSR controls
  controls_nb %>%
    runifpoint() %>%
    superimpose(
      # Matern cases
      rMatClust(
        kappa = kappa,
        scale = scale,
        mu = cases_nb / kappa
      )
    ) %>%
    as.wmppp() ->
    X
  # Update the number of cases
  cases_nb <- X$n - controls_nb
  c(rep("Control", controls_nb), rep("Case", cases_nb)) %>%
    as.factor() ->
    X$marks$PointType
  rgamma(
    X$n,
    shape = size_gamma_shape,
    scale = size_gamma_scale
  ) %>%
    ceiling() ->
    X$marks$PointWeight
  X
}

# Example
X <- X_matern(par_points_nb)
# Map the cases
autoplot(X[X$marks$PointType == "Case"])
```



Les Cas apparaissent dans la fenêtre ci-dessus : les agrégats des Cas sont nettement visibles.

2.2 Maillage de l'espace

Considérons la simulation des Cas obtenus par le processus de processus de Matérn et découpons la fenêtre en une grille. Un maillage est de l'espace précédemment considéré est obtenu.

```
# Number of rows and columns
par_partitions <- 20
```

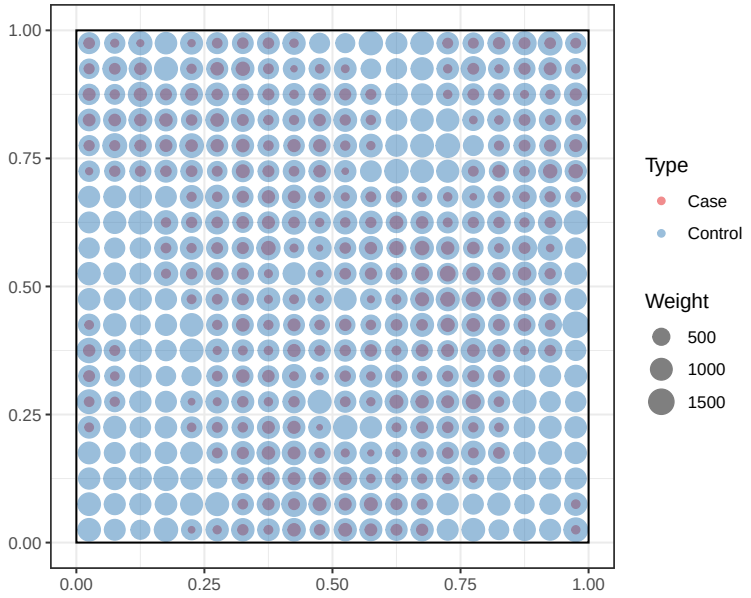
La fonction `group_points()` rassemble au centre de chaque cellule de la grille tous les points qu'elle contient. Cela permet de simuler l'approximation habituelle de la position des points d'une unité administrative sur la position de son centre.

```
# Group points into cells
group_points <- function(X, partitions = par_partitions) {
X %>%
  with(tibble(
    x,
    y,
    PointType = marks$PointType,
    PointWeight = marks$PointWeight)
  ) %>%
  mutate(
    x_cell = ceiling(x * partitions) / partitions - 1 / 2 / partitions,
    y_cell = ceiling(y * partitions) / partitions - 1 / 2 / partitions
  ) %>%
  group_by(PointType, x_cell, y_cell) %>%
  summarise(n = n(), PointWeight = sum(PointWeight)) %>%
```

```
  rename(x = x_cell, y = y_cell) %>%
  as.wmppp(window = X$window, unitname = X$window$units)
}
```

La position recentrée est présentée sur la carte suivante. Chaque cellule ne contient plus qu'un seul point de chaque type dont le poids est la somme des poids des points individuels. L'absence de Cas dans une cellule est aisément détectable (point unicolore bleu), tout comme la forte présence de Cas dans une cellule (point bicolore mais à dominante rouge).

```
group_points(X) %>% autoplot(alpha = 0.5)
```



3 Calcul de M

Les distances auxquelles la fonction M est calculées sont choisies dans `r`.

```
r <- c((0:9) / 100, (2:10) / 20)
```

3.1 Données nécessaires

Dans le package *dbmss*, la fonction s'applique à un jeu de points, objet de classe `wmppp`, ou à une matrice de distance, objet de classe `Dtable`.

Nous partons d'un tableau (data.frame) contenant les colonnes `x`, `y`, `PointType` et `PointWeight`.

Les coordonnées spatiales des points sont données par les colonnes `x` et `y`. Aucune grille n'est considérée dans un premier temps.

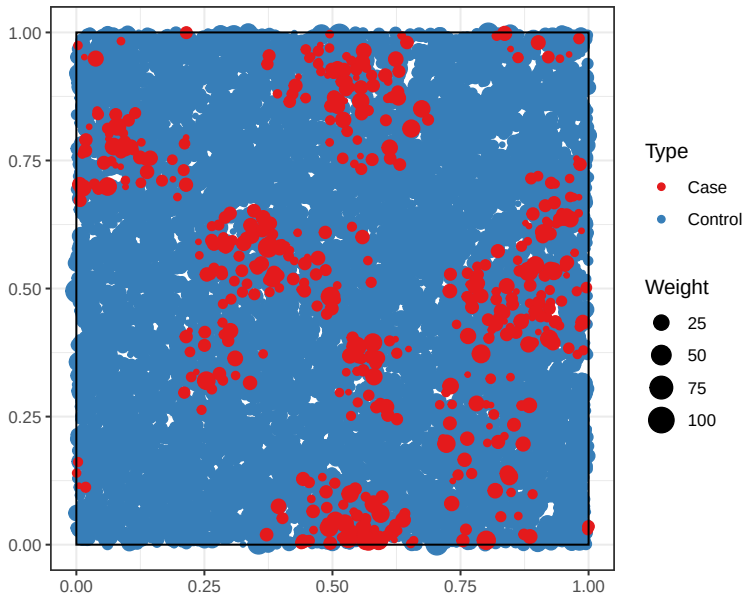
```
# Draw 10000 points and make a dataframe
X_matern(points_nb = 1E4) %>%
  with(data.frame(x, y, marks)) ->
  points_df
head(points_df)
```

```
##            x         y PointWeight PointType
## 1 0.89231474 0.1307877           1   Control
## 2 0.04432755 0.7844362          46   Control
## 3 0.78102343 0.7502537          10   Control
## 4 0.72095334 0.8236009           4   Control
## 5 0.95409741 0.2098661           2   Control
## 6 0.86374118 0.3331368           7   Control
```

3.2 Jeu de points

La fonction `wmppp()` permet de créer le jeu de points à partir du tableau. La fenêtre est créée automatiquement si elle n'est pas précisée. Ici, c'est un carré de côté 1.

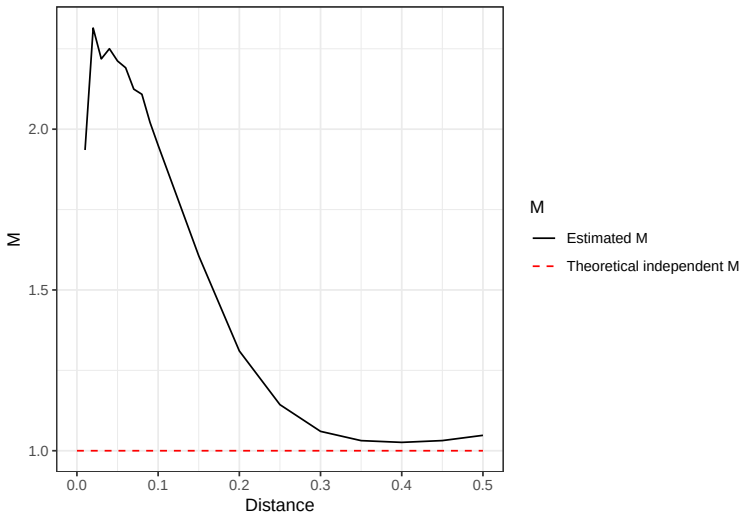
```
library("dbmss")
X <- wmppp(points_df, window = square(1))
autoplot(X)
```



La fonction `Mhat()` permet d'estimer la fonction. La valeur de référence théorique de M est 1 car cette fonction rapporte la proportion de Cas jusqu'à une distance r à celle observée sur toute la fenêtre. L'agrégation des Cas sera mise en évidence par des valeurs de M supérieures à 1 (la présence relative de Cas est plus importante localement que sur l'ensemble de la fenêtre) et la dispersion des Cas par des valeurs inférieures à 1. On observe que M détecte une agglomération des Cas, ce qui est en accord avec la simulation de ce type de points (les

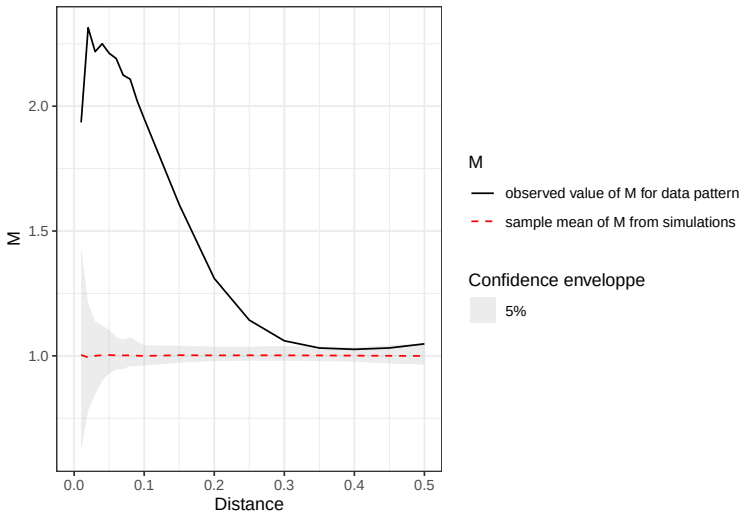
contrôles ayant localisation complètement aléatoire sur la fenêtre). L'avantage d'une fonction fondée sur les distances est nettement visible : elle permet de détecter exactement à quelle(s) distance(s) les phénomènes d'attraction se produisent et sont les plus importants (pour les fonctions dont les valeurs peuvent être comparées à différents rayons, comme M).

```
X %>%
  Mhat(r = r, ReferenceType = "Case") %>%
  autoplot()
```



La fonction `Menvelope()` permet de calculer l'intervalle de confiance de la valeur de la fonction sous l'hypothèse nulle de localisation aléatoire des points. L'intervalle de confiance global (Duranton and Overman, 2005) est calculé en précisant l'argument `Global = TRUE`. Sur le graphique ci-dessous, l'intervalle de confiance, simulé à 95%, apparaît en gris et est centré sur la valeur 1.

```
X %>%
  MEnvelope(r = r, ReferenceType = "Case", Global = TRUE) %>%
  autoplot()
```



3.3 Matrice de distances

La fonction `as.Dtable()` permet de créer un objet `Dtable`.

```
d_matrix <- as.Dtable(points_df)
```

Il peut aussi être créé à partir d'une matrice de distances obtenue autrement, contenant par exemple des distances non euclidiennes (temps de transport, distance routière…).

```
# A Dtable containing two points
Dmatrix <- matrix(c(0, 1, 1, 0), nrow = 2)
PointType <- c("Type1", "Type2")
PointWeight <- c(2, 3)
Dtable(Dmatrix, PointType, PointWeight)
```

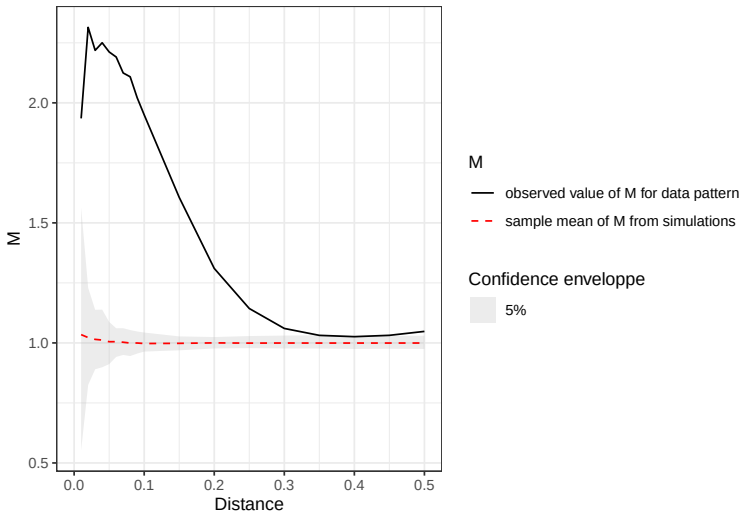
```
## $Dmatrix
##      [,1] [,2]
## [1,]    0    1
## [2,]    1    0
##
## $n
## [1] 2
##
## $marks
## $marks$PointType
## [1] Type1 Type2
## Levels: Type1 Type2
##
## $marks$PointWeight
## [1] 2 3
##
##
## attr(,"class")
## [1] "Dtable"
```

Les fonctions `Mhat()` et `MEnvelope()` sont les mêmes que pour les jeux de points.

```
identical(
  Mhat(X, r = r, ReferenceType = "Case", NeighborType = "Control"),
  Mhat(d_matrix, r = r, ReferenceType = "Case", NeighborType = "Control")
)
```

```
## [1] TRUE
```

```
d_matrix %>%
  MEnvelope(r = r, ReferenceType = "Case", Global = TRUE) %>%
  autoplot()
```



3.4 Performance

Le package *microbenchmark* proposé par **?** est retenu pour chronométrer le temps d'exécution de la fonction *M*. Le calcul des distances est extrêmement rapide dans la fonction `Mhat()` : la matrice le fait économiser, mais le traitement complet à partir d'une matrice est finalement plus long.

```
library("microbenchmark")
(
  mb <- microbenchmark(
    Mhat(X, r = r, ReferenceType = "Case", NeighborType = "Control"),
    Mhat(d_matrix, r = r, ReferenceType = "Case", NeighborType = "Control"),
    times = 4L
  )
)
```

```
## Unit: milliseconds
##                                                                                 expr
##         Mhat(X, r = r, ReferenceType = "Case", NeighborType = "Control")
##  Mhat(d_matrix, r = r, ReferenceType = "Case", NeighborType = "Control")
##       min       lq     mean   median       uq
##  17.97968 18.12893 18.57384 18.63833 19.01875
##  37.85978 37.92666 48.86461 39.59725 59.80255
##       max neval
##  19.03902     4
##  78.40415     4
```

4 Tests

La fonction `X_to_M()` calcule la fonction M et renvoie le vecteur de ses valeurs pour chaque distance. Elle est utile pour mesurer les temps d'exécution.

```
# Compute M
X_to_M <- function(X) {
  X %>%
    Mhat(r = r, ReferenceType = "Case") %>%
    pull("M")
}
```

Le nombre de répétition des tests est fixé par `simulations_n`.

```
simulations_n <- 10
```

L'effet de l'approximation de la localisation est testé d'abord sur un jeu de points agrégé, similaire aux données réelles de Tidu et al. (2023). Dans un deuxième temps, le cas d'un jeu de points non structuré est traité.

4.1 Cas d'une distribution agrégée (Matérn)

`X_matern_list` contient `simulations_n` tirages du jeu de points.

```
# Simulate X
X_matern_list <- replicate(
  simulations_n,
  expr = X_matern(par_points_nb),
  simplify = FALSE
)
```

Pour évaluer l'effet de l'approximation de la position, le calcul exact et le calcul sur les points de la grille sont effectués sur chaque jeu de points.

```
library("pbapply")
# Compute M
system.time(M_matern_original <- pbsapply(X_matern_list, FUN = X_to_M))
```

```
##    user  system elapsed
##  12.410   0.081   3.255
```

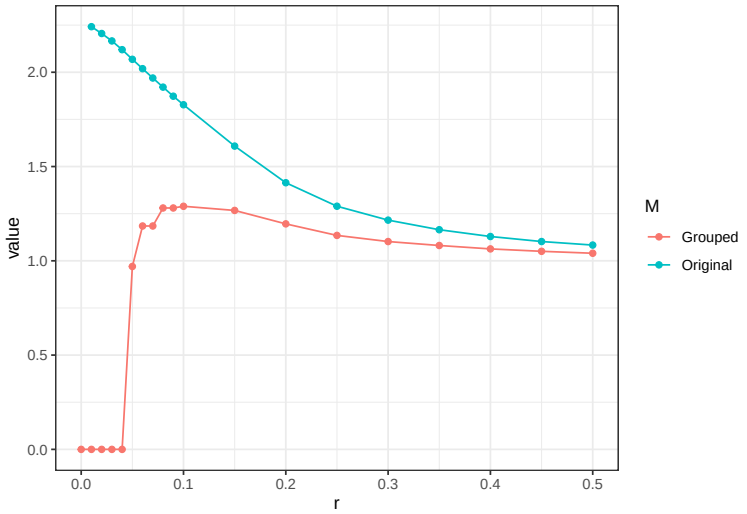
```
# Group points and compute M
X_matern_grouped_list <- lapply(
  X_matern_list,
  FUN = group_points,
  partitions = par_partitions
)
# Compute M
system.time(M_matern_grouped <- sapply(X_matern_grouped_list, FUN = X_to_M))
```

```
##    user  system elapsed
##   0.037   0.005   0.025
```

Le calcul approximé est très rapide parce qu'il réduit le nombre de points à celui du nombre de cellules.

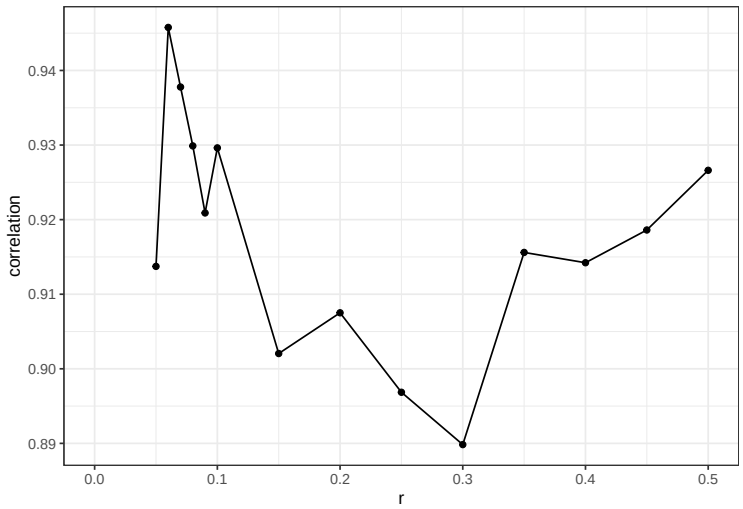
Les valeurs moyennes sont présentées ci-dessous.

```r
tibble(
  r,
  Original = rowMeans(M_matern_original),
  Grouped =  rowMeans(M_matern_grouped)
) %>%
  pivot_longer(
    cols = !r,
    names_to = "M",
    values_to = "value"
  ) %>%
  ggplot(aes(x = r, y = value, color = M)) +
  geom_line() +
  geom_point()
```



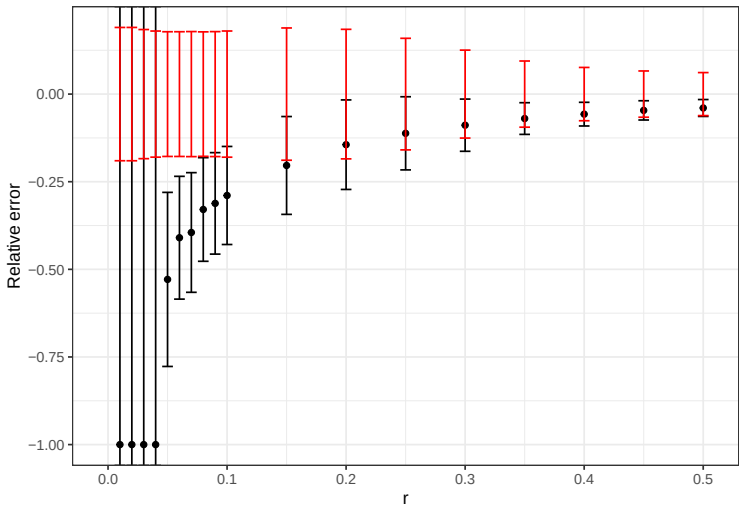
La corrélation entre les valeurs de M estimées par chaque méthode est calculée à chaque distance.

```r
# Correlation
M_cor <- function(r_value, M_original, M_grouped) {
  r_index <- which(r == r_value)
  # Return
  c(
    # Distance
    r_value,
    # Correlation
    cor(M_original[r_index, ], M_grouped[r_index, ])
  )
}
sapply(
  r,
  FUN = M_cor,
  M_original = M_matern_original,
  M_grouped = M_matern_grouped
) %>%
  t() %>%
  as_tibble() %>%
  rename(r = V1, correlation = V2) %>%
  ggplot(aes(x = r, y = correlation)) +
    geom_point() +
    geom_line()
```



La corrélation est très élevée dès que la distance prise en compte dépasse la maille de la grille. Les valeurs sont ensuite comparées.

```
# Compare values
M_bias <- function(r_value, M_original, M_grouped) {
  r_index <- which(r == r_value)
  # Return
  c(
    # Distance
    r_value,
    # Relative error
    mean((M_grouped[r_index, ] - M_original[r_index, ]) / M_original[r_index, ]),
    # Standardised error sd
    sd(M_grouped[r_index, ] - M_original[r_index, ]) / mean(M_grouped[r_index, ]),
    # Coefficient of variation
    sd(M_original[r_index, ] / mean(M_original[r_index, ]))
  )
}
sapply(
  r,
  FUN = M_bias,
  M_original = M_matern_original,
  M_grouped = M_matern_grouped
) %>%
  t() %>%
  as_tibble() %>%
  rename(r = V1, `Relative error` = V2, `Error CV` = V3, `M CV` = V4) %>%
  ggplot() +
    geom_point(aes(x = r, y = `Relative error`)) +
    geom_errorbar(
      aes(
        x = r,
        ymin = `Relative error` - `Error CV`,
        ymax = `Relative error` + `Error CV`
      )
    ) +
    geom_errorbar(aes(x = r, ymin = -`M CV`, ymax = `M CV`), col = "red")
```



La figure ci-dessus montre, en rouge, la variabilité de la valeur de M (son coefficient de variation) au cours des simulations. Par définition, la valeur moyenne est sans erreur. L'erreur relative (à la valeur exacte de M) moyenne est présentée en noir, avec son écart-type normalisé par la valeur exacte de M.

Bien que les corrélations soient très grandes, l'erreur relative dépasse 25% jusqu'à 2 fois la taille de la maille.

4.2 Cas d'une distribution complètement aléatoire (CSR)

`X_csr_list` contient `simulations_n` tirages du jeu de points.

```
# Simulate X
X_csr_list <- replicate(
  simulations_n,
  expr = X_csr(par_points_nb),
  simplify = FALSE
)
```

Le calcul exact et le calcul sur les points de la grille sont effectués sur chaque jeu de points.

```
# Compute M
system.time(M_csr_original <- pbsapply(X_csr_list, FUN = X_to_M))
```

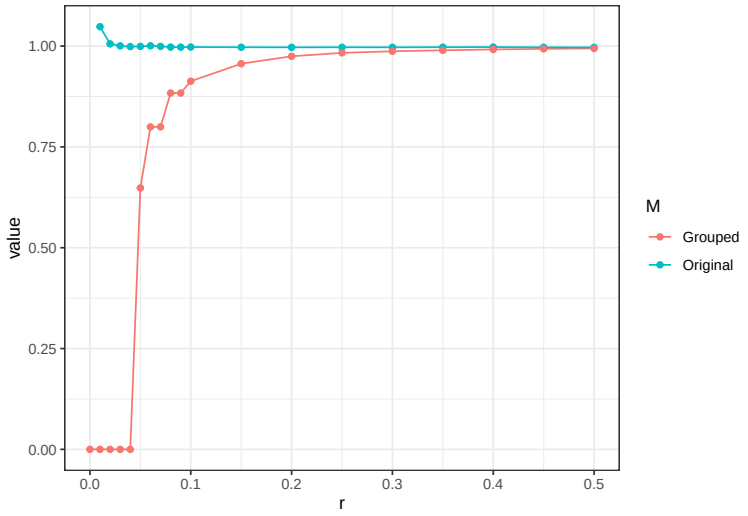
```
##    user  system elapsed
##  13.755   0.163   3.734
```

```
# Group points and compute M
X_csr_grouped_list <- lapply(
  X_csr_list,
  FUN = group_points,
  partitions = par_partitions
)
# Compute M
system.time(M_csr_grouped <- sapply(X_csr_grouped_list, FUN = X_to_M))
```

```
##    user  system elapsed
##   0.062   0.004   0.040
```

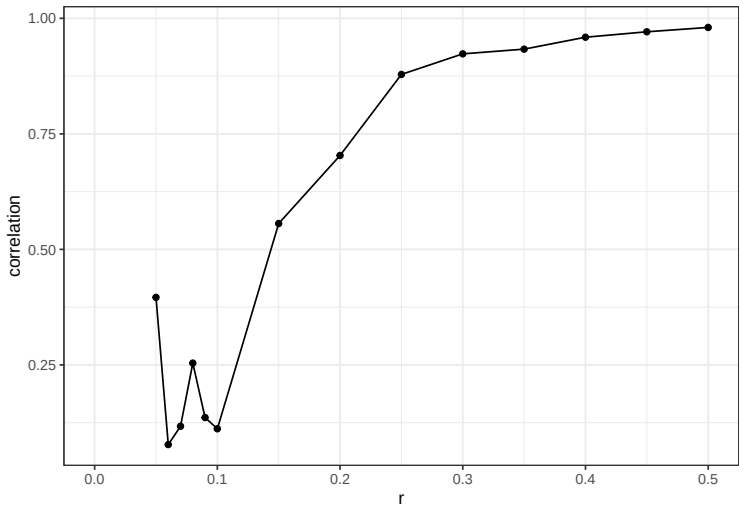
Les valeurs moyennes sont présentées ci-dessous.

```
tibble(
  r,
  Original = rowMeans(M_csr_original),
  Grouped =  rowMeans(M_csr_grouped)
) %>%
  pivot_longer(
    cols = !r,
    names_to = "M",
    values_to = "value"
  ) %>%
  ggplot(aes(x = r, y = value, color = M)) +
  geom_line() +
  geom_point()
```



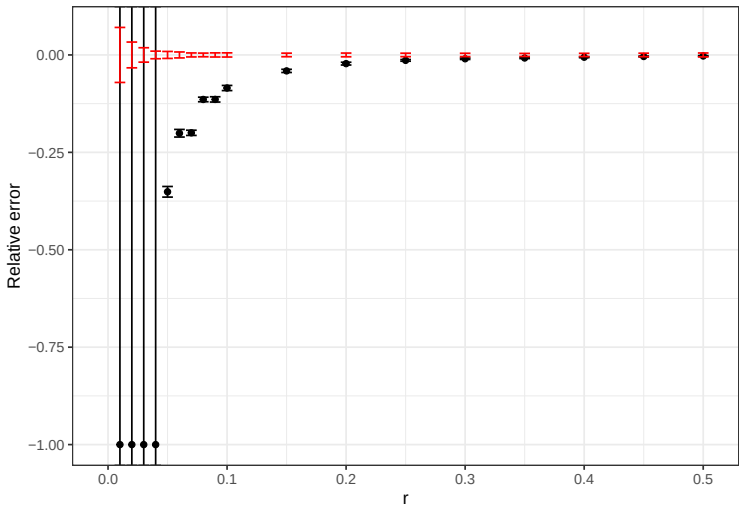
La corrélation entre les valeurs de M calculées par chaque méthode est calculée à chaque distance.

```
# Correlation
sapply(
  r,
  FUN = M_cor,
  M_original = M_csr_original,
  M_grouped = M_csr_grouped
) %>%
  t() %>%
  as_tibble() %>%
  rename(r = V1, correlation = V2) %>%
  ggplot(aes(x = r, y = correlation)) +
    geom_point() +
    geom_line()
```



En absence de structure spatiale, les corrélations sont bien plus faibles. Les valeurs sont comparées.

```
# Compare values
sapply(
  r, FUN = M_bias,
  M_original = M_csr_original,
  M_grouped = M_csr_grouped
) %>%
  t() %>%
  as_tibble() %>%
  rename(r = V1, `Relative error` = V2, `Error CV` = V3, `M CV` = V4) %>%
  ggplot() +
    geom_point(aes(x = r, y = `Relative error`)) +
    geom_errorbar(
      aes(
        x = r,
        ymin = `Relative error` - `Error CV`,
        ymax = `Relative error` + `Error CV`
      )
    ) +
    geom_errorbar(aes(x = r, ymin = -`M CV`, ymax = `M CV`), col = "red")
```



La valeur moyenne de M est 1 à toutes les distances par construction : les cas et les contrôles sont distribués complètement aléatoirement.

Les barres rouges sur la figure ci-dessus représentent l'écart-type empirique de la valeur de M, calculé à partir des simulations. Les points noirs montrent l'erreur apportée par l'approximation, mesurée par l'écart moyen entre les valeurs de M calculées avec ou sans approximation. Les barres d'erreur sont l'écart-type de cette différence.

On constate que l'approximation sous-estime systématiquement M. L'erreur est maximale jusqu'à la taille de la grille : tous les points d'une même cellule sont placés artificiellement en son centre. Elle chute brutalement au-delà de ce seuil mais reste importante jusqu'à 4 fois la taille de la grille.

Notre conclusion prudente dans ce cas est que l'approximation de doit pas être utilisée pour étudier les interactions à courte distance.

Le test sur les corrélations effectué ici est beaucoup plus sévère que dans Tidu et al. (2023) : les points n'ont aucune structure, donc M permet de détecter les petites variations aléatoires des différents tirages. En présence d'une structure spatiale, les valeurs de M sont nettement mieux corrélées, mais dans tous les cas l'erreur d'estimation est grande.

5 Performance de M

5.1 Temps de calcul

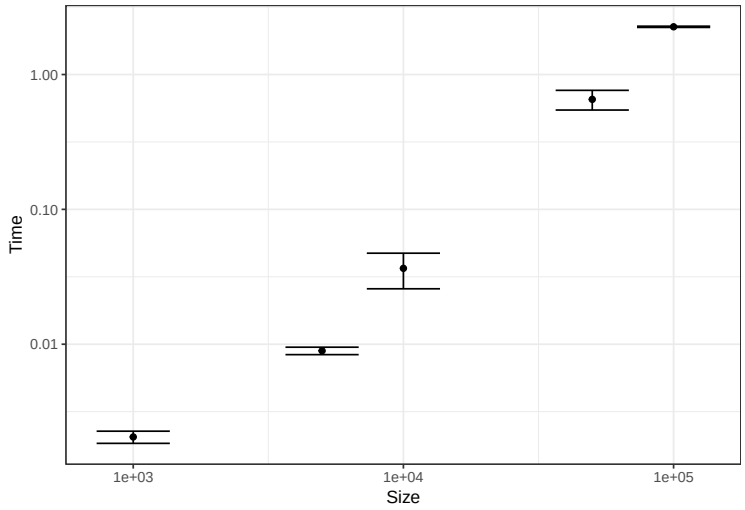
Le temps de calcul nécessaire au calcul exact est évalué pour une gamme de nombres de points précisée dans `X_sizes`.

```
X_sizes <- c(1000, 5000, 10000, 50000, 100000)
```

La fonction `test_time()` permet de mesurer le temps d'exécution d'une évaluation de la fonction M.

```r
library("microbenchmark")
test_time <- function(points_nb) {
  X <- X_csr(points_nb)
  microbenchmark(X_to_M(X), times = 4L) %>%
    pull("time")
}

X_sizes %>%
  sapply(FUN = test_time) %>%
  as_tibble() %>%
  pivot_longer(cols = everything()) %>%
  rename(Size = name) %>%
  group_by(Size) %>%
  summarise(Time = mean(value) / 1E9, sd = sd(value) / 1E9) %>%
  mutate(
    Size = as.double(
      plyr::mapvalues(
        .$Size,
        from = paste0("V", seq_along(X_sizes)),
        to = X_sizes
      )
    )
  ) -> M_time
M_time %>%
  ggplot(aes(x = Size, y = Time)) +
    geom_point() +
    geom_errorbar(aes(ymin = Time - sd, ymax = Time + sd)) +
    scale_x_log10() +
    scale_y_log10()
```



Le temps de calcul est lié à la taille du jeu de points par une loi puissance.

```r
# Model
M_time %>%
  mutate(logTime = log(Time), logSize = log(Size)) ->
  M_time_log
M_time_lm <- lm(logTime ~ logSize, data = M_time_log)
summary(M_time_lm)
```

```
##
## Call:
```

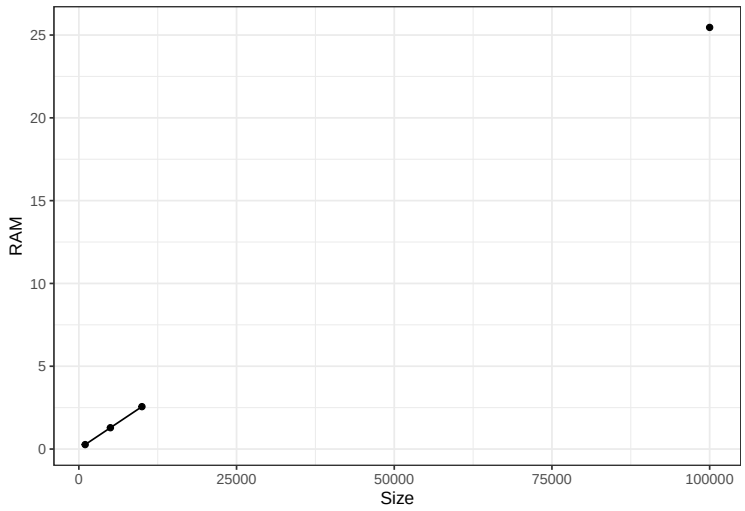
```
## lm(formula = logTime ~ logSize, data = M_time_log)
##
## Residuals:
##        1        2        3        4        5
##  0.48675 -0.57396 -0.25605  0.09729  0.24597
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.5421     1.2551  -13.98 0.000793
## logSize       1.5731     0.1316   11.95 0.001260
##
## (Intercept) ***
## logSize     **
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4837 on 3 degrees of freedom
## Multiple R-squared:  0.9794, Adjusted R-squared:  0.9726
## F-statistic: 142.9 on 1 and 3 DF,  p-value: 0.00126
```

Le temps de calcul augmente moins vite que le carré du nombre de points. Il peut être estimé très précisément ($R^2 = 0.98$) par la relation $t = t_0(n/n_o)^p$ où t est le temps estimé pour n points connaissant le temps t_0 (ex. : 2.26) pour n_0 points (ex. : 100000) et p la relation de puissance (1.57).

5.2 Mémoire

La mémoire utilisée est évaluée pour les mêmes tailles de données.

```
# RAM
library("profmem")
test_ram <-function(points_nb) {
  X <- X_csr(points_nb)
  profmem(X_to_M(X)) %>%
    pull("bytes") %>%
    sum()
}
sapply(X_sizes, FUN = test_ram) %>%
  tibble(Size = X_sizes, RAM = . / 2^20) ->
  M_ram
M_ram %>%
  ggplot(aes(x = Size, y = RAM)) +
    geom_point() +
    geom_line()
```



La mémoire nécessaire (en Mo) augmente linéairement avec le nombre de points et n'est jamais critique pour des tailles de jeux de points traitables dans des temps raisonnables. Cela permet de relativiser la conclusion de Tidu et al. (2023) sur la puissance et le temps de calculs nécessaires en utilisant M sur des jeux de données de taille importante.

```
# Model
lm(RAM ~ Size, data = M_ram) %>% summary()
```

```
##
## Call:
## lm(formula = RAM ~ Size, data = M_ram)
##
## Residuals:
##          1          2          3          5
## -6.138e-04 -2.093e-05  6.973e-04 -6.254e-05
##
## Coefficients:
##              Estimate Std. Error  t value
## (Intercept) 1.475e-02  4.029e-04    36.62
## Size        2.545e-04  8.008e-09 31775.07
##             Pr(>|t|)
## (Intercept) 0.000745 ***
## Size         9.9e-10 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0006585 on 2 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 1.01e+09 on 1 and 2 DF,  p-value: 9.904e-10
```

La mémoire utilisée par les objets `Dtable` pour le calcul de M à partir d'une matrice de distance est bien supérieure : c'est celle d'une matrice numérique, de l'ordre de 8 octets fois le nombre de points au carré, soit 800 Mo pour 10000 points seulement.

6 Conclusion

Le temps de calcul de M est de l'ordre de 6 secondes pour un jeu de 100 000 points sur un ordinateur portable (processeur Intel i7-1360P 2.20 GHz), et nécessite 25 Mo de RAM. Le calcul d'un intervalle de confiance à partir de 1000 simulations prend donc moins de deux heures.

Pour un jeu de cinq millions de points, le temps de calcul attendu est $6 \times 50^{1.8} = 6860$ secondes, près de deux heures. 1000 simulations nécessiteraient alors environ trois mois. Le calcul des distances est parallélisé : un serveur de calcul augmenterait drastiquement la performance mais au prix d'une complexité de mise en oeuvre qui limite son usage.

En se limitant à la puissance de calcul d'un ordinateur personnel, le calcul exact se justifie donc pleinement pour des données de l'ordre de 10^5 points : quelques heures suffisent à calculer des intervalles de confiance.

Au-delà, l'approximation de la localisation permet de ramener la taille du jeu de points à celle du nombre de localisations retenues. Le prix à payer est l'absence d'information à l'échelle des unités géographiques élémentaires (les cellules de la grille ici), et une erreur relative importante. Si les valeurs de M sont utilisées comme covariables dans un modèle (par exemple pour expliquer la croissance des points), alors cette imprécision est acceptable parce que la corrélation entre leur valeur exacte et leur valeur approximée est élevée, dès que les points présentent une structure spatiale.

Références

Duranton, G. and H. G. Overman (2005). Testing for localisation using micro-geographic data. *Review of Economic Studies 72*(4), 1077–1106.

Marcon, E. and F. Puech (2010). Measures of the geographic concentration of industries : Improving distance-based methods. *Journal of Economic Geography 10*(5), 745–762.

Marcon, E., S. Traissac, F. Puech, and G. Lang (2015). Tools to characterize point patterns : dbmss for R. *Journal of Statistical Software 67*(3), 1–15.

Tidu, A., F. Guy, and S. Usai (2023, November). Measuring Spatial Dispersion : An Experimental Test on the M -Index. *Geographical Analysis*, gean.12381.