Appendix

On the Computation of Large Spatial Datasets With the M function

Eric Marcon

Florence Puech

June 4, 2025

Abstract

Increasing access to large geo-referenced datasets, coupled with the development of computing power, has encouraged the search for suitable spatial statistic tools. In this line, distance-based methods have been largely developed in many scientific fields to detect spatial concentration, dispersion or independence of entities at any distance and without bias. In a recent article, Tidu et al. (2024) highlight the qualities of the M function (Marcon and Puech, 2010), a relative distance-based measure, but they also express reservations for the computation times required. In our article, we propose a methodological work that seeks to specify the processing of large spatialised datasets with the M function by using R software. We appraise the computational performance of M into two ways. At first, a precise evaluation of the computational time and memory requirements for geo-referenced data is carried out using the *dbmss* package in R by means of performance tests. Then, as suggested by Tidu et al. (2024), we consider an approximation of the geographical positions of the entities. The extent of the deterioration in the estimate of M is estimated and discussed, as are gains in computation time. We give evidence that the individual location approximation generates information loss at small distances, implying a trade-off between the smallest distance at which spatial interactions can be detected and computing performance. We recommend designing the analysis of large datasets taking it into account. The R code used in the article is given for the reproducibility of our results.

The code used in the article "On the Computation of Large Spatial Datasets With M" is detailed here.

1 Data simulation

1.1 Drawing the points

A set of points is drawn at random with the following parameters:

- the number of points,
- the proportion of controls,
- the shape and scale of the gamma distribution.

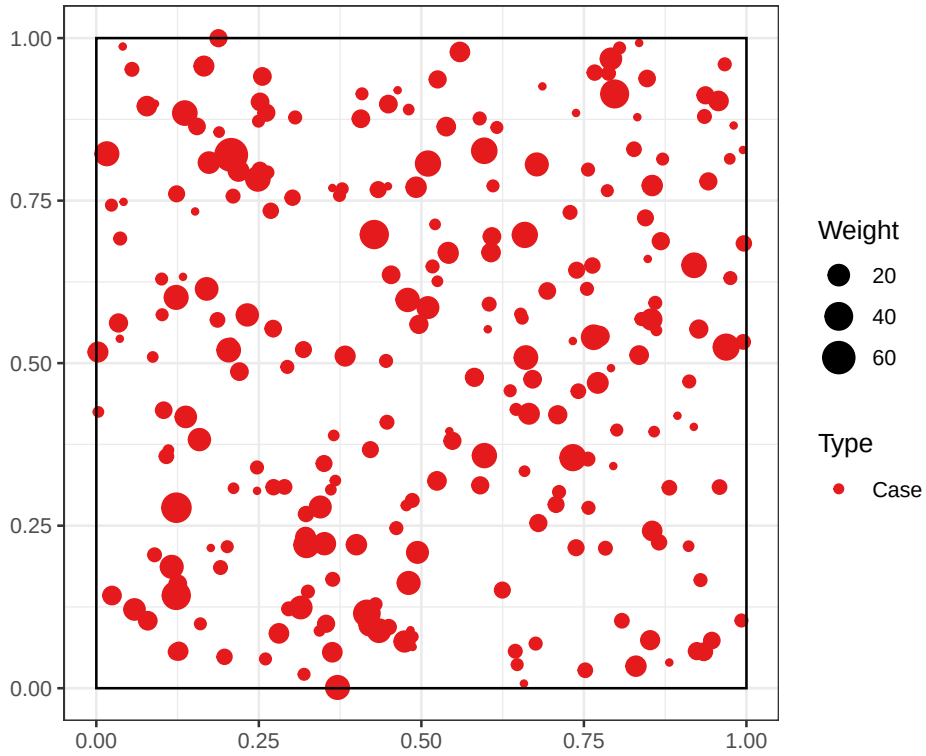
```
library("tidyverse")
library("spatstat")
library("dbmss")

par_points_nb <- 5000
par_case_ratio <- 1/20
par_size_gamma_shape <- 0.95
par_size_gamma_scale  <- 10
```

The `X_csr()` function is used to draw a series of points according to certain parameters. The `points_nb` argument, which sets the number of points, can be modified; the other parameters have their values set above.

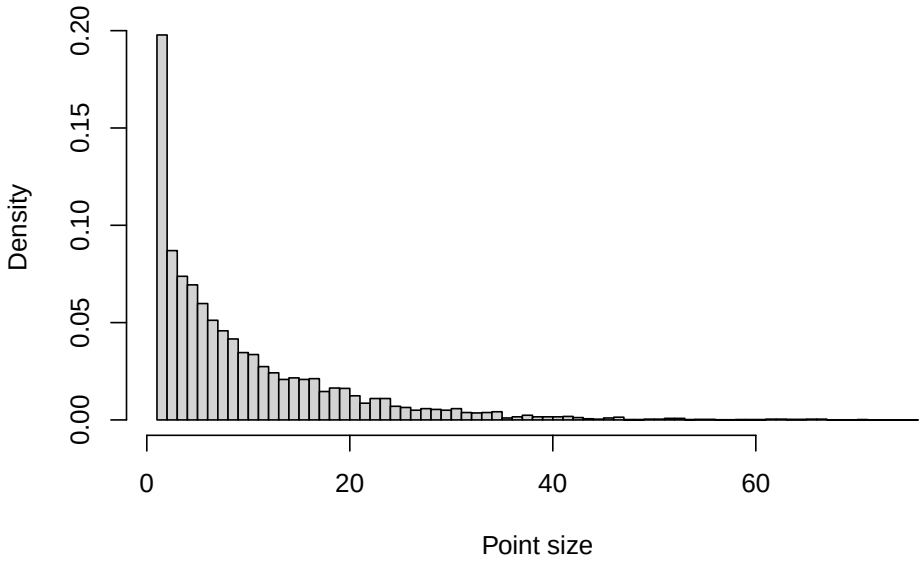
```
X_csr <- function(
    points_nb,
    case_ratio = par_case_ratio,
    size_gamma_shape = par_size_gamma_shape,
    size_gamma_scale = par_size_gamma_scale) {
  points_nb %>%
    runifpoint() %>%
    as.wmppp() ->
    X
  cases_nb <- round(points_nb *  case_ratio)
  controls_nb <- points_nb - cases_nb
  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"])
```



The size distribution is shown in the histogram below:

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



The `X_matern()` function is used to draw a semiset of points whose Cases are concentrated by a Matérn (1960) process. The parameters are

- κ: the expected number of clusters,
- `scale`: their radius.

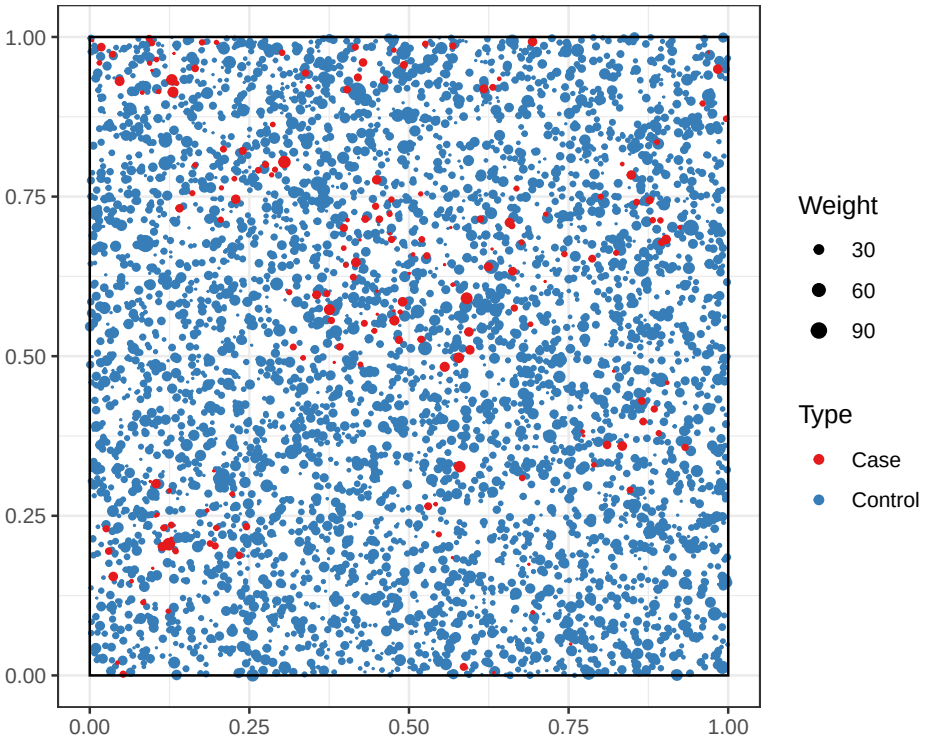
```
# Expected number of clusters
par_kappa <- 20
# Cluster radius
par_scale <-  0.1
```

The function code is as follows:

```
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) +
  scale_size(range = c(0, 3))
```



1.1.1 Space grid

The number of rows and columns is set:

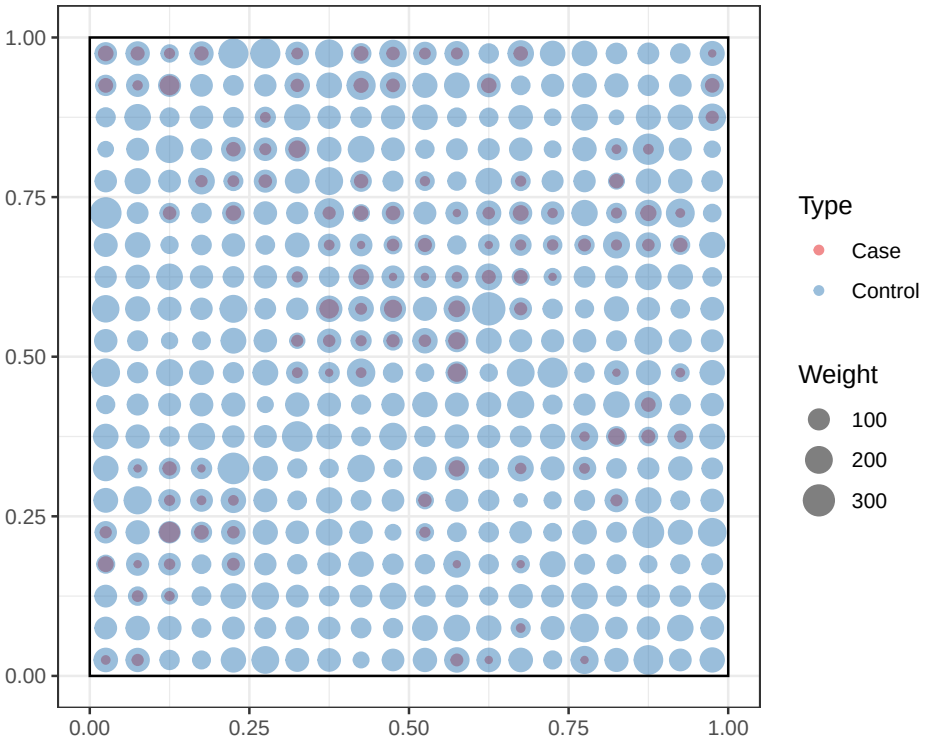
```
# Number of rows and columns
par_partitions <- 20
```

The `group_points()` function gathers all the points it contains at the centre of each grid cell. This simulates the usual approximation of the position of the points in an administrative unit to the position of its centre. The position of the points is slightly noisy to enable M to be calculated. The `group_points_to_plot()` function merges the points to produce a map.

```r
# 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,
      .keep = "unused"
    ) %>%
    rename(x = x_cell, y = y_cell)  %>%
    as.wmppp(window = X$window, unitname = X$window$units) %>%
    rjitter()
}
# Group points and merge them
group_points_to_plot <- function(X, partitions = par_partitions) {
  X %>%
    with(tibble(
      x,
      y,
      PointType = marks$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)
}
```

The figure is obtained using the following code:

```r
X %>% group_points_to_plot() %>% autoplot(alpha = 0.5)
```



2 Calculation of M

The distances at which the M function is calculated are chosen from `r`.

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

2.1 Necessary data

In the *dbmss* package (Marcon et al., 2015), the function applies to a set of points, object of class `wmppp`, or to a matrix of distances, object of class `Dtable`.

We start with an array (data.frame) containing the columns `x`, `y`, `PointType` and `PointWeight`. The spatial co-ordinates of the points are given by the `x` and `y` columns.

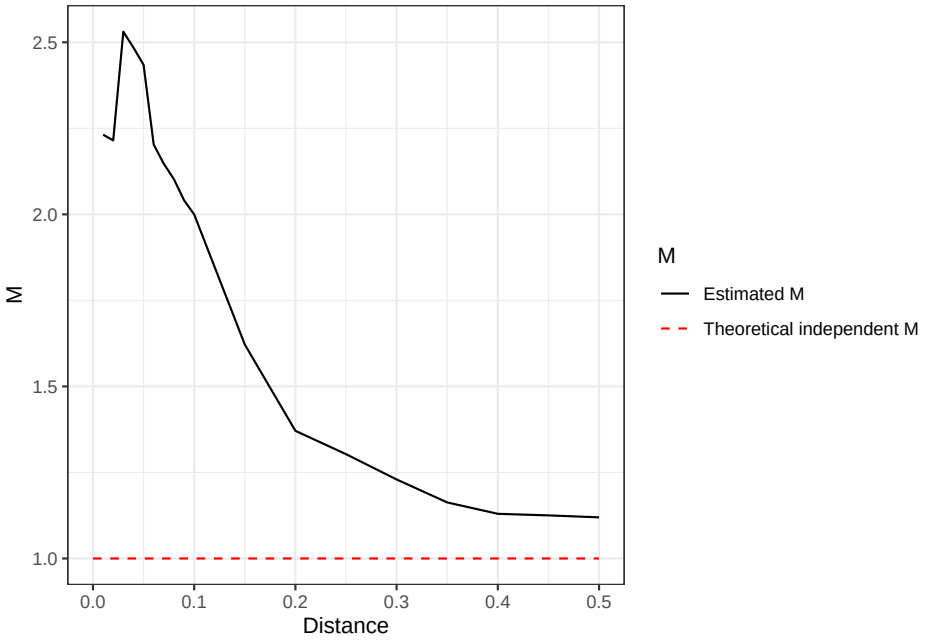
```
# Extract a dataframe from the point set
points_df <- with(X, data.frame(x, y, marks))
head(points_df)
```

```
##           x          y PointWeight PointType
## 1 0.4550716 0.31775637           4   Control
## 2 0.6463730 0.04396279           2   Control
## 3 0.9156488 0.23361975           2   Control
## 4 0.9724551 0.87464816           1   Control
## 5 0.8907927 0.26205266           4   Control
## 6 0.9561687 0.96813173           7   Control
```

2.2 Set of points

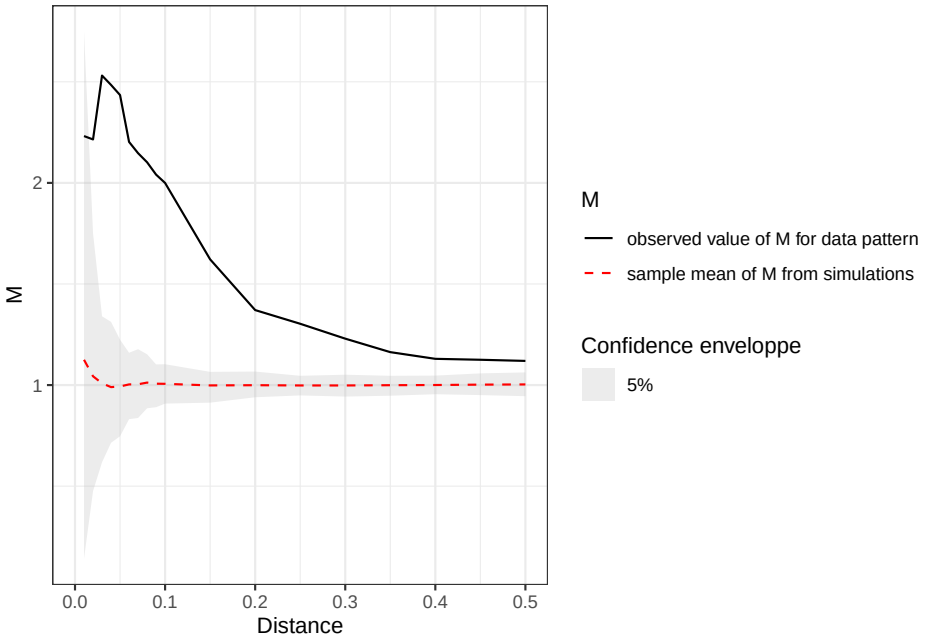
The `Mhat()` function is used to estimate the value of the M function.

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



The `Menvelope()` function is used to calculate the confidence interval of the function value under the null hypothesis of random location of the points. The global confidence interval (Duranton and Overman, 2005) is calculated by specifying the argument `Global = TRUE`.

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



2.3 Distance matrix

The `as.Dtable()` function is used to create a `Dtable` object.

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

It can also be created from a distance matrix obtained in another way, containing non-Euclidean distances for example (transport time, road distance, etc.).

```
# 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)
```

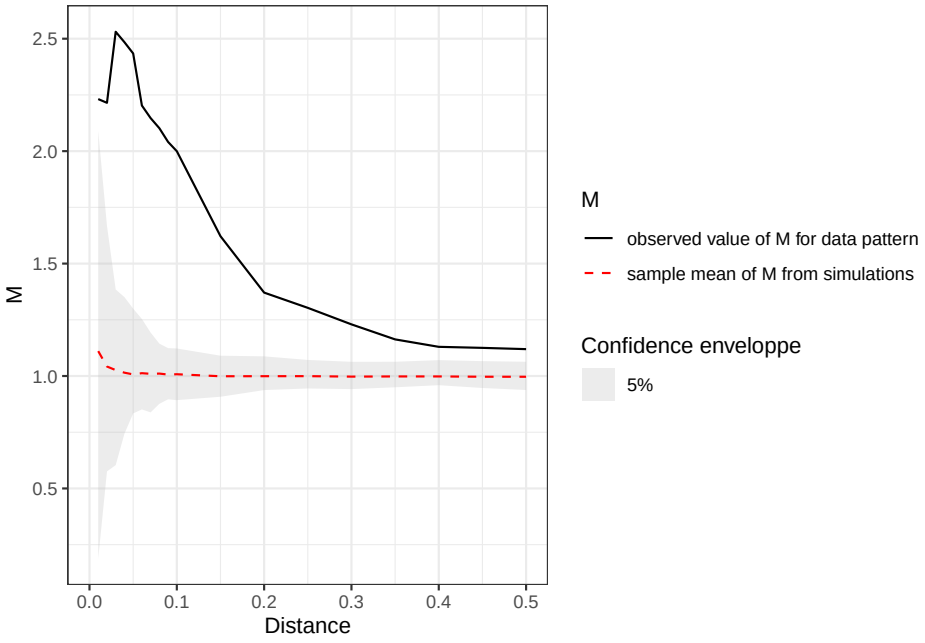
```
## $Dmatrix
##      [,1] [,2]
## [1,]    0    1
## [2,]    1    0
##
## $n
## [1] 2
##
## $marks
## $marks$PointType
## [1] Type1 Type2
## Levels: Type1 Type2
##
## $marks$PointWeight
## [1] 2 3
##
##
## attr(,"class")
## [1] "Dtable"
```

The `Mhat()` and `MEnvelope()` functions are the same as for point sets.

```
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 Performance of M

The `X_to_M()` function calculates the M function and returns the vector of its values for each distance. It is useful for measuring execution times.

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

3.1 Calculation time

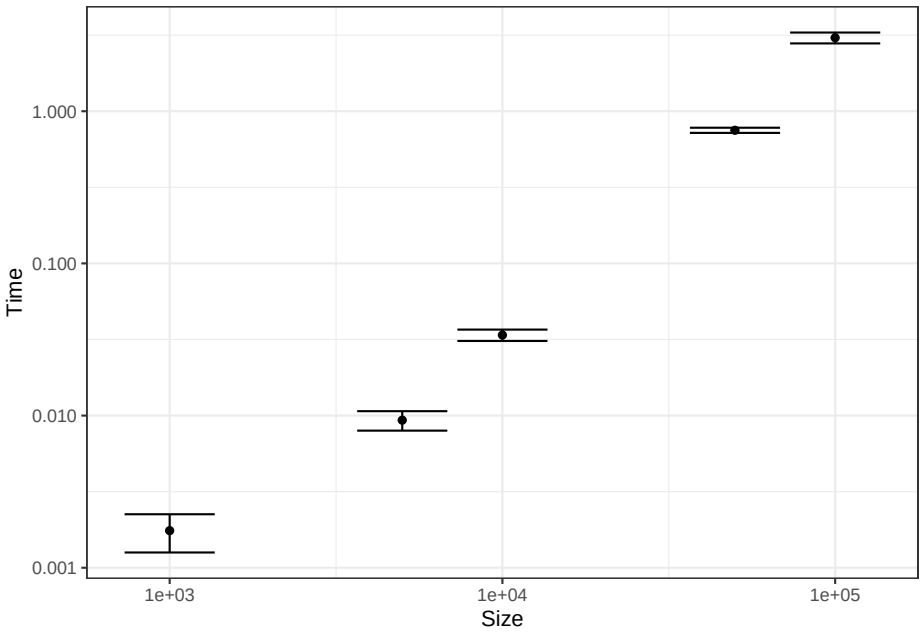
The time required for an exact calculation is evaluated for a range of numbers of points specified in `X_sizes`.

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

The `test_time()` function is used to measure the execution time of an evaluation of the M function.

```
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()
```



The calculation time is related to the size of the set of points by a power law.

```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
##  0.50039 -0.50218 -0.36531  0.05939  0.30771
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -18.3221     1.2837  -14.27 0.000745
## logSize       1.6613     0.1346   12.34 0.001146
##
## (Intercept) ***
## logSize     **
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4947 on 3 degrees of freedom
## Multiple R-squared:  0.9807, Adjusted R-squared:  0.9742
## F-statistic: 152.3 on 1 and 3 DF,  p-value: 0.001146
```

The *microbenchmark* package proposed by Mersmann (2023) is used to compare the computation time of the function between a set of points and a matrix of distances.

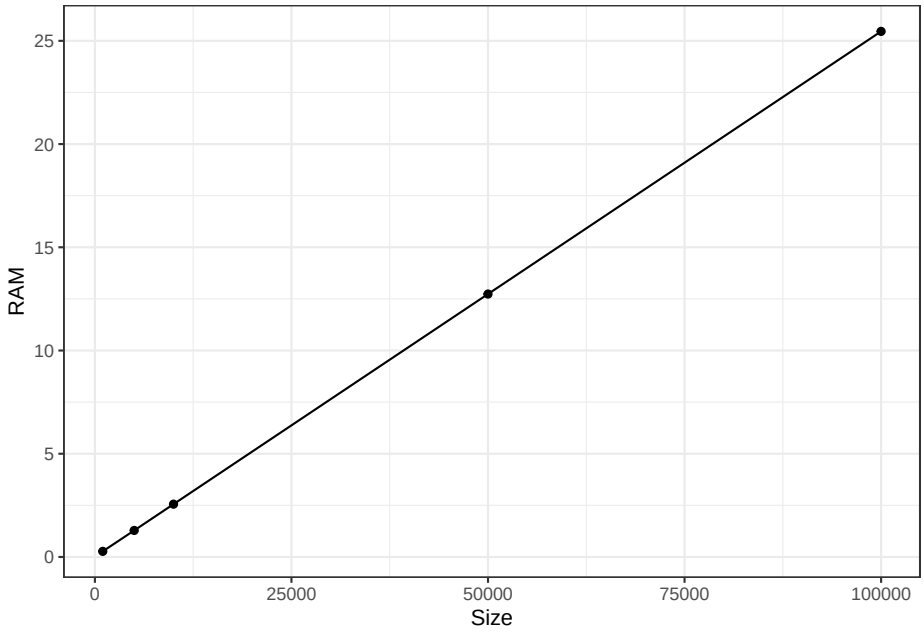
The calculation of distances is extremely fast in the `Mhat()` function: the matrix saves time, but the complete processing from a matrix is ultimately longer.

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

3.2 Memory

The memory used is evaluated with the *profmem* package (Bengtsson, 2021).

```r
# RAM
library("profmem")
test_ram <- function(points_nb) {
  X <- X_csr(points_nb)
  profmem(X_to_M(X)) %>%
    pull("bytes") %>%
    sum(na.rm = TRUE)
}
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()
```



The memory required (in MB) increases linearly with the number of points.

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

```
##
## Call:
## lm(formula = RAM ~ Size, data = M_ram)
##
## Residuals:
##          1          2          3          4
##  0.0032318 -0.0021685 -0.0011180 -0.0002664
##          5
##  0.0003211
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.431e-02  1.400e-03   10.22    0.002
## Size        2.544e-04  2.786e-08 9131.23  2.9e-12
##
## (Intercept) **
## Size        ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.00235 on 3 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 8.338e+07 on 1 and 3 DF,  p-value: 2.897e-12
```

4 Effects of approximating the position of the points

The number of test repetitions is set by `simulations_n`.

```
simulations_n <- 100
```

4.1 Case of an aggregated distribution (Matérn)

`X_matern_list` contains `simulations_n` drawn from the set of points. `X_matern_grouped_list` contains the same simulations, whose points have been grouped in the grid cells.

```
# Simulate X
X_matern_list <- replicate(
  simulations_n,
  expr = X_matern(par_points_nb),
  simplify = FALSE
)
# Group points and compute M
X_matern_grouped_list <- lapply(
  X_matern_list,
  FUN = group_points,
  partitions = par_partitions
)
```

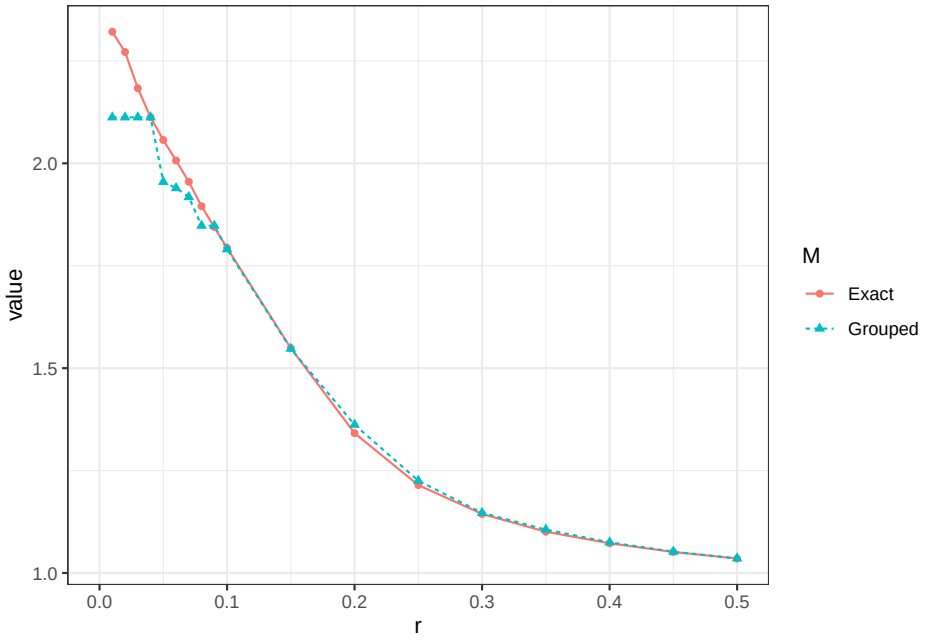
To assess the effect of the position approximation, the exact calculation and the calculation on the grid points are performed on each set of points.

```
library("pbapply")
# Compute M
M_matern_original <- pbsapply(X_matern_list, FUN = X_to_M)
M_matern_grouped <- pbsapply(X_matern_grouped_list, FUN = X_to_M)
```

The approximate calculation is very fast because it reduces the number of points to the number of cells, provided you take advantage of this in the code used for the calculation. This is not the case here: the *dbmss* package does not provide for this approximation. The `M_hat()` function is therefore applied to the grouped set of points, but calculated in the same way as with the original set of points.

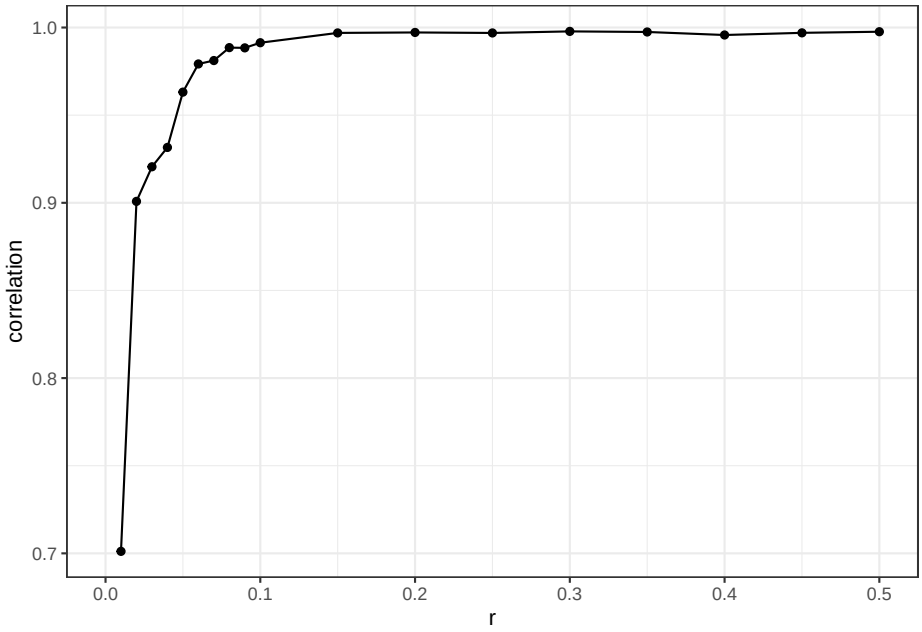
The mean values of the M estimates are shown below.

```
tibble(
  r,
  Exact = 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, shape = M, linetype  = M)) +
  geom_line() +
  geom_point()
```



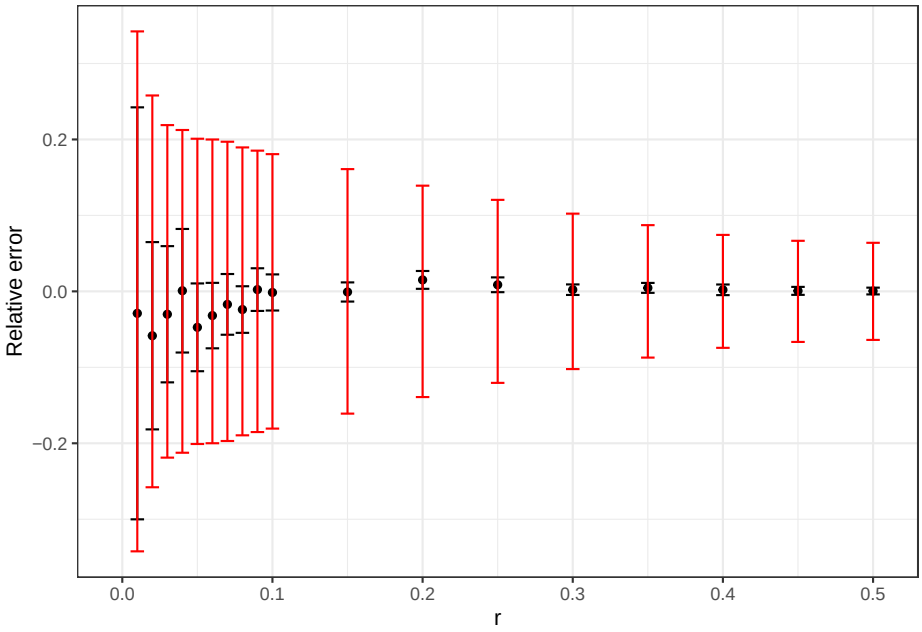
The correlation between the M values estimated by each method is calculated at each distance r.

```
# 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()
```



The correlation is very high as soon as the distance taken into account exceeds the grid cell. The values are then compared.

```
# 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")
```



The figure above shows, in red, the variability of the value of M (its coefficient of variation) over the course of the simulations. By definition, the mean value is error-free. At short distances, the values of M vary greatly for the same point process, depending on the stochasticity of its runs. As M is a cumulative function, it stabilises as distance increases.

The average relative error (to the exact value of M), due to the approximation of the position of the points, is shown in black, with its standard deviation normalised by the exact value of M. It is small, less than 10%, even at short distances.

4.2 Case of a completely random distribution (CSR)

`X_csr_list` contains `simulations_n` draws from the set of points.

```
# Simulate X
X_csr_list <- replicate(
  simulations_n,
  expr = X_csr(par_points_nb),
  simplify = FALSE
)
# Group points and compute M
X_csr_grouped_list <- lapply(
  X_csr_list,
  FUN = group_points,
  partitions = par_partitions
)
```

The exact calculation and the calculation on the points of the grid are carried out on each set of points.

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

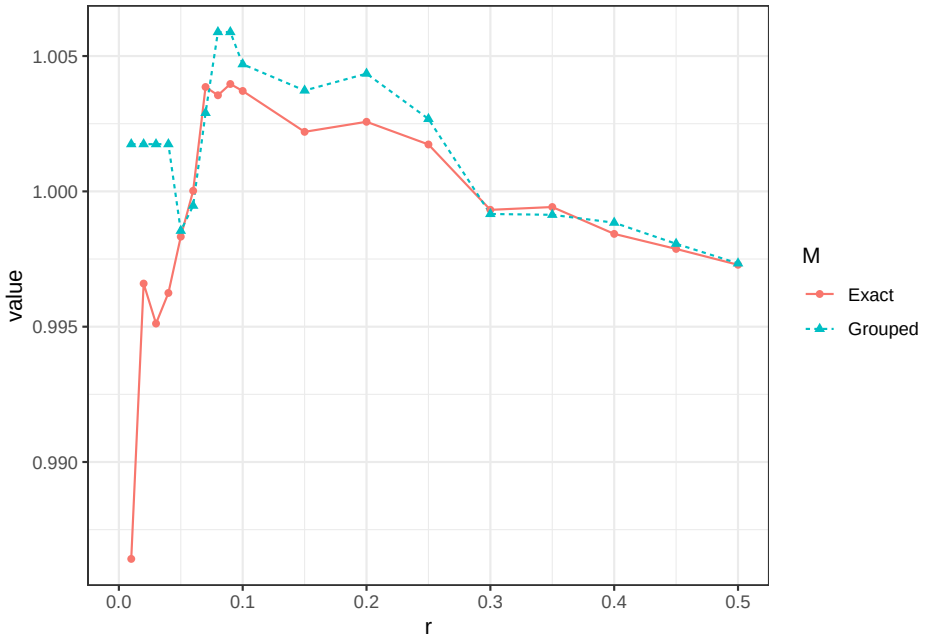
```
##    user  system elapsed
##   3.129   0.173   1.830
```

```
system.time(M_csr_grouped <- sapply(X_csr_grouped_list, FUN = X_to_M))
```

```
##    user  system elapsed
##   3.018   0.112   1.682
```

The average values are shown below.

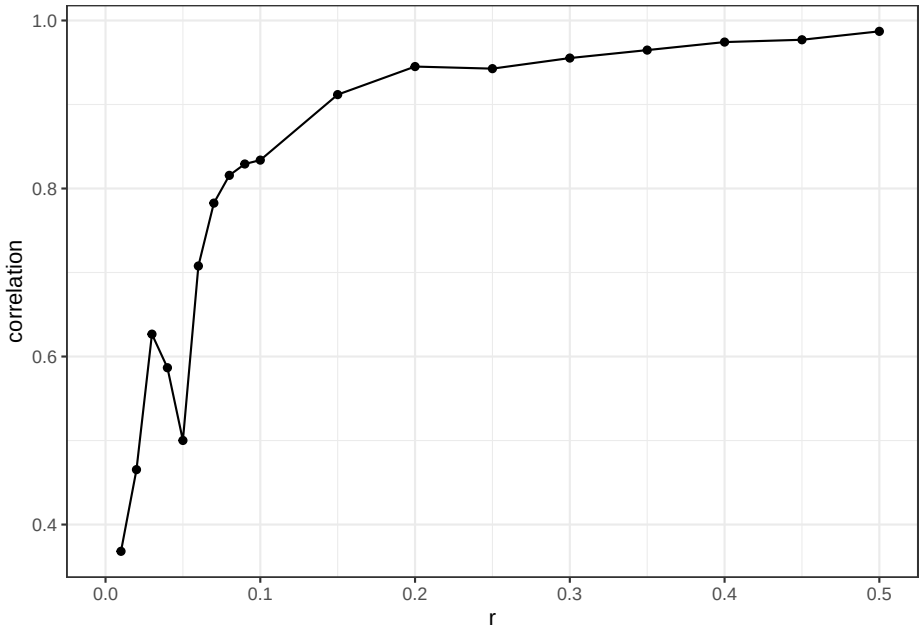
```
tibble(
  r,
  Exact = rowMeans(M_csr_original),
  Grouped =  rowMeans(M_csr_grouped)
) %>%
  pivot_longer(
    cols = !r,
    names_to = "M",
    values_to = "value"
  ) %>%
  ggplot(aes(x = r, y = value, color = M, shape = M, linetype  = M)) +
  geom_line() +
  geom_point()
```



The correlation between the M values calculated by each method is calculated at each distance r.

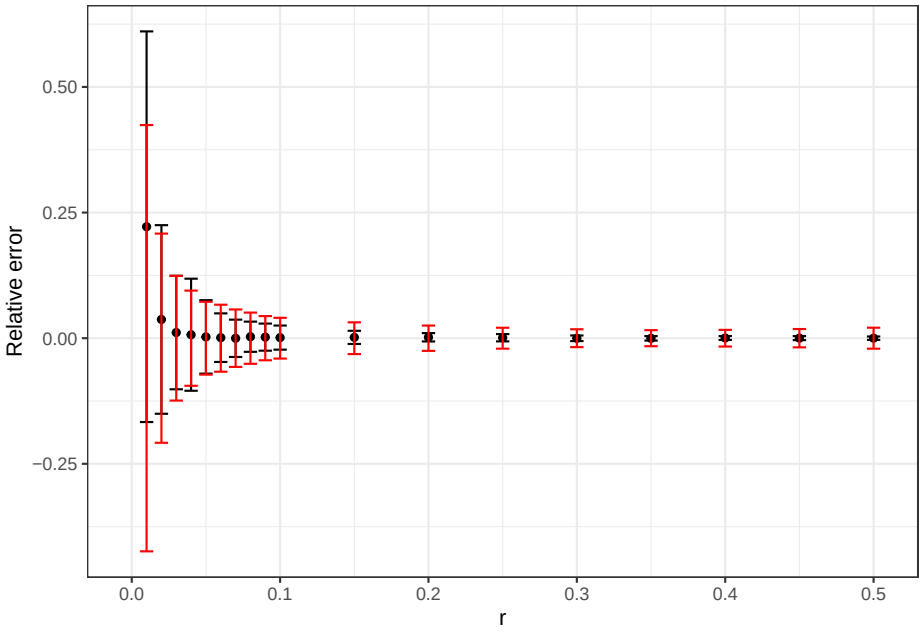
```
# Correlation
sapply(
  r,
  FUN = M_cor,
  M_original = M_csr_original,
  M_grouped = M_csr_grouped
) %>%
  t() %>%
```

```
  as_tibble() %>%
  rename(r = V1, correlation = V2) %>%
  ggplot(aes(x = r, y = correlation)) +
    geom_point() +
    geom_line()
```



In the absence of spatial structure, correlations are much weaker. The values are compared.

```
# 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")
```



The figure above is constructed in the same way as for aggregated point sets. In the absence of spatial structure, the value of M varies much less.

In the presence of a spatial structure, the estimation error is large at short distances. It becomes negligible beyond the grid cell.

References

Bengtsson, H. (2021). A Unifying Framework for Parallel and Distributed Processing in R using Futures. *The R Journal 13*(2), 208.

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.

Matérn, B. (1960). Spatial variation. *Meddelanden från Statens Skogsforskningsinstitut 49*(5), 1–144.

Mersmann, O. (2023). *Microbenchmark: Accurate Timing Functions.*

Tidu, A., F. Guy, and S. Usai (2024). Measuring Spatial Dispersion: An Experimental Test on the M-Index. *Geographical Analysis 56*(2), 384–403.