Appendix

Computation of Large Spatial Datasets with the M function

Eric Marcon

Florence Puech

January 29, 2026

Abstract

Increasing access to large geo-referenced datasets, coupled with the development of computing power, has encouraged the search for suitable spatial statistical tools. Distance-based methods have been extensively developed in several scientific fields to detect spatial concentration, dispersion or independence of entities at any distance and without any bias. Recently, Tidu et al. (2024) highlighted the qualities of Marcon and Puech's M function, a relative distance-based measure, and also expressed reservations about the computation time required. Herein, we propose a methodology that specifies the processing of large spatialized datasets with the M function using R software. The computational performance of M was conducted using two methods: (i) a precise evaluation of the computational time and memory requirements for geo-referenced data was conducted using the *dbmss* package in R via performance tests, and (ii) based on Tidu et al. (2024), we considered an approximation of the geographical positions of the entities. The deterioration extent of the M results was estimated and discussed as the gains it provides in computation time. We provided evidence that the individual location approximation generated information loss at substantially small distances, implying a trade-off between the smallest distance at which spatial interactions could be detected and computing performance. The R code used in the article is given for the reproducibility of our results.

The code used in the article "Computation of Large Spatial Datasets with the M function" 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.

In this appendix, the number of points of the examples is reduced to 5000 to limit computing overhead. The parameter `par_points_nb` can be set to 100,000 to reproduce the exact figures of the article.

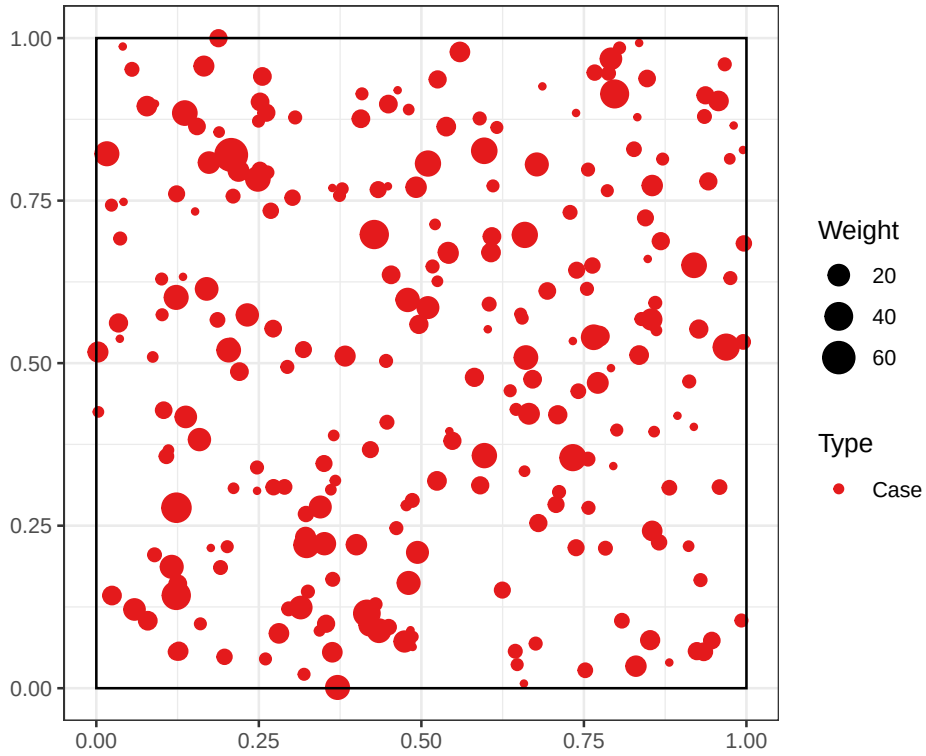
```
library("tidyverse")
library("spatstat")
library("dbmss")

par_points_nb <- 5000 # 1E5 in the article
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.

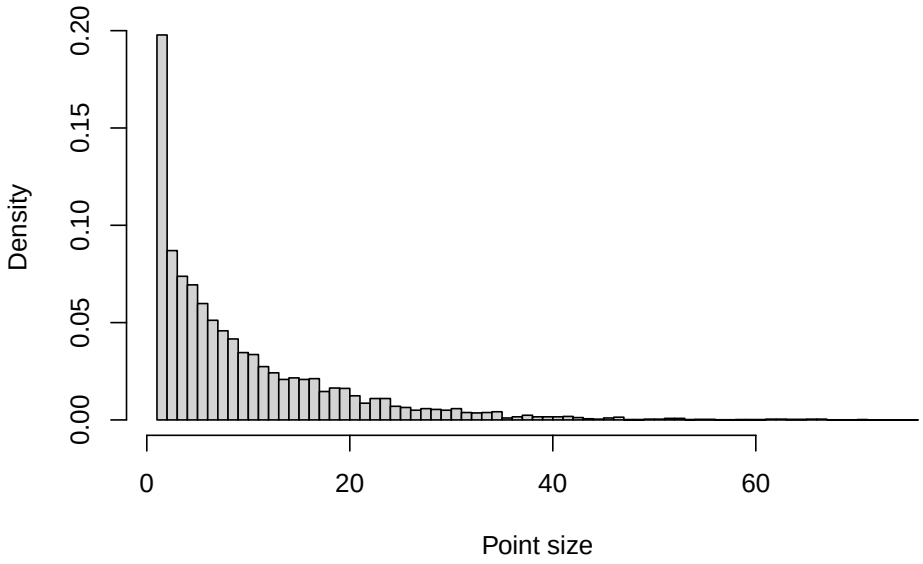
```
X_csr <- function(
    points_nb,
    case_ratio = par_case_ratio,
    size_gamma_shape = par_size_gamma_shape,
    size_gamma_scale = par_size_gamma_scale) {
  points_nb %>%
    runifpoint() %>%
    as.wmppp() ->
    X
  cases_nb <- round(points_nb *  case_ratio)
  controls_nb <- points_nb - cases_nb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

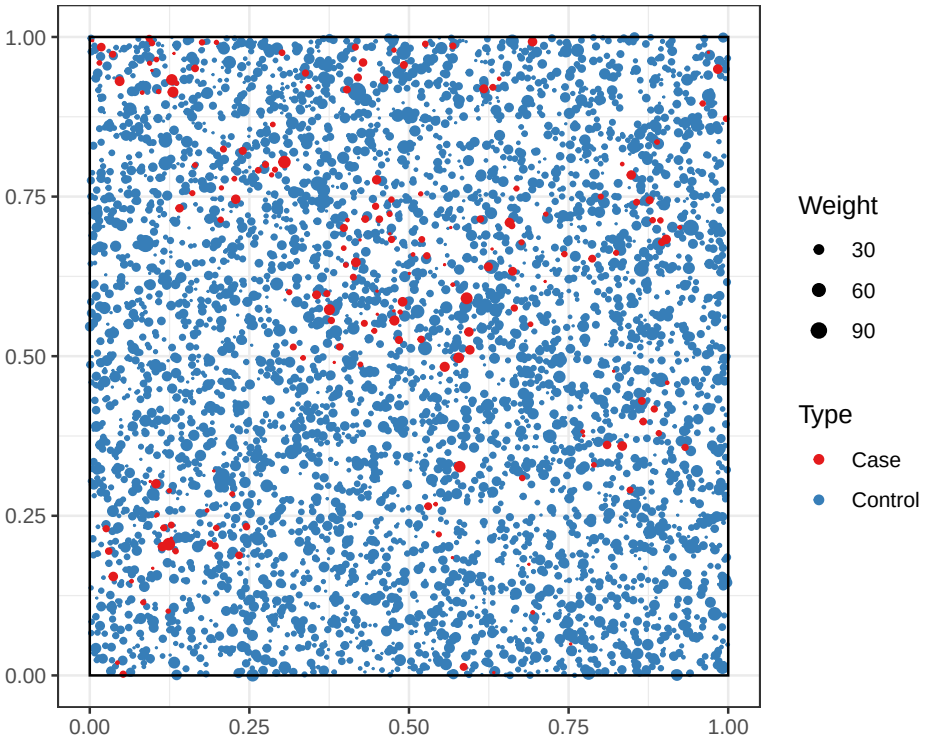
```
# Expected number of clusters
par_kappa <- 20
# Cluster radius
par_scale <-  0.1
```

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))
```



1.2 Gridding the space

The number of rows and columns is set:

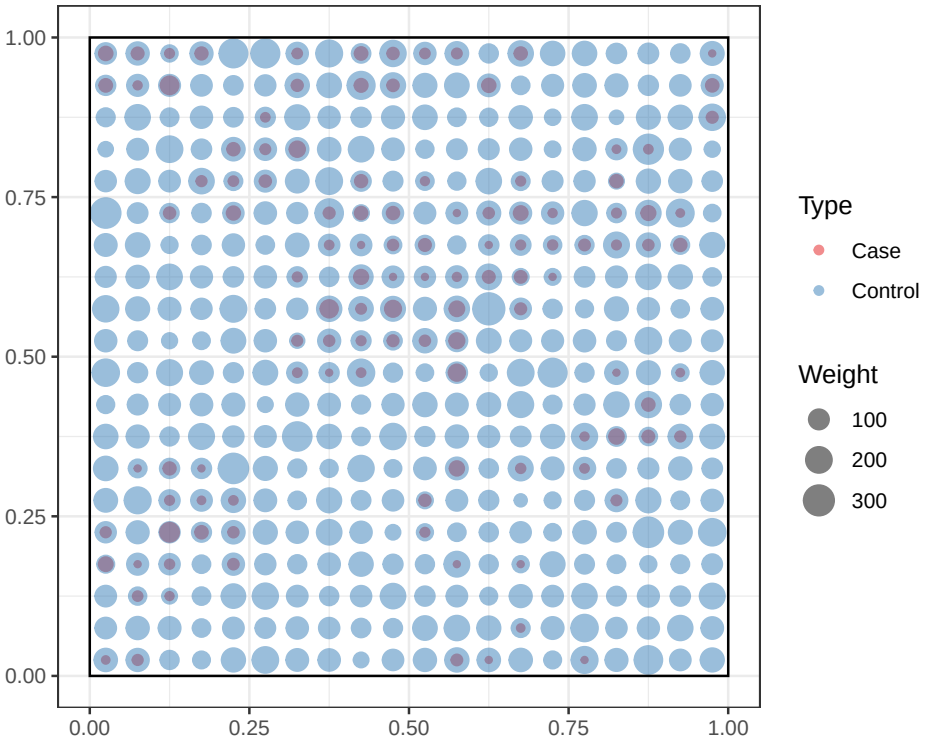
```
# Number of rows and columns
par_partitions <- 20
```

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 Computing M with the *dbmss* package

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.

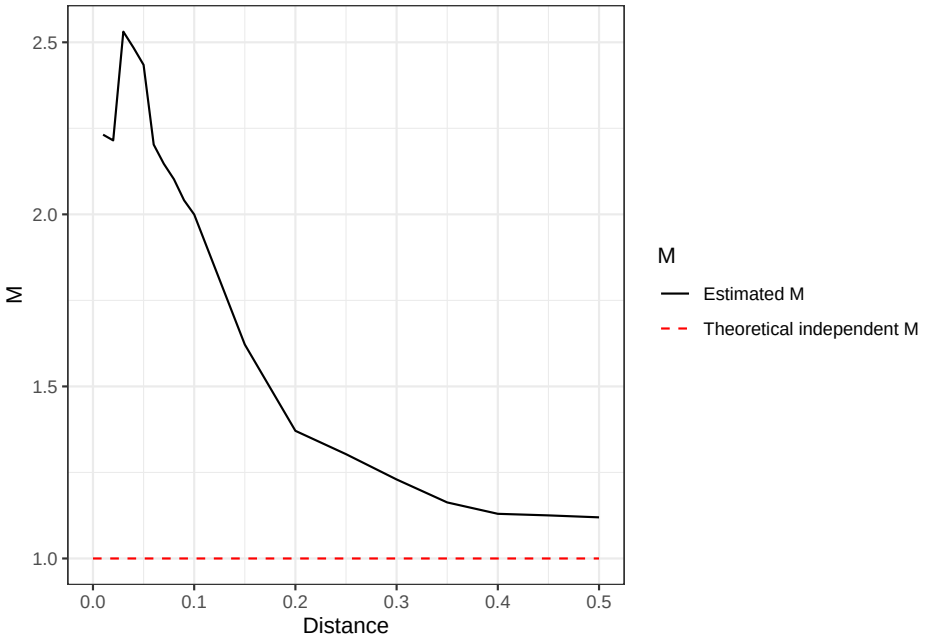
```
# Extract a dataframe from the point set
points_df <- with(X, data.frame(x, y, marks))
head(points_df)
```

```
##           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

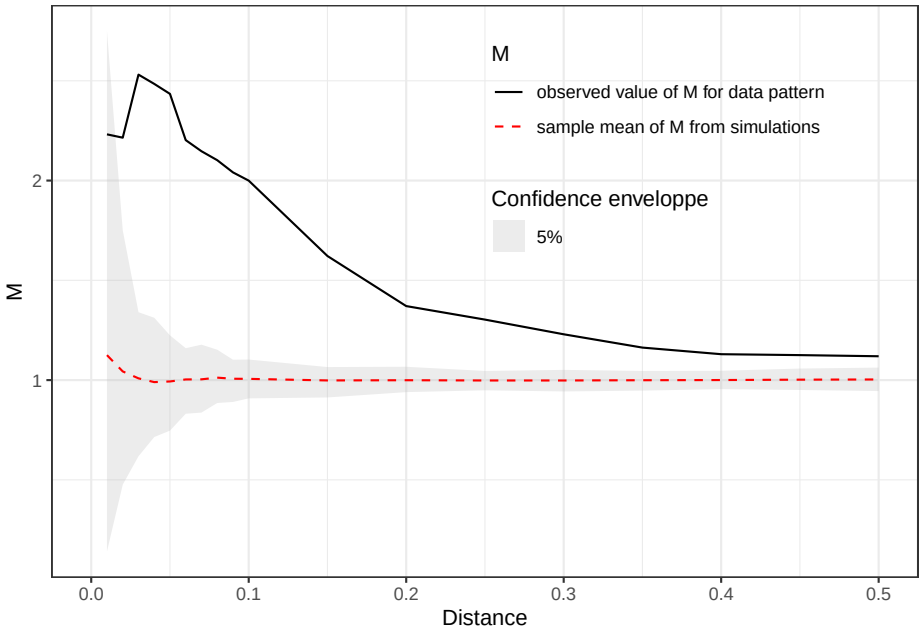
The `Mhat()` function is used to estimate the value of the M function.

```
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() +
  theme(
    legend.position = c(0.7, 0.75),
    legend.background = element_blank()
  )
```



The confidence envelope contains 95% of the simulated values of the M function under the null hypothesis of independence between points. It is much smaller in the article because it decreases with the number of points.

2.3 Distance matrix

The code of this section will not run with datasets over 10,000 points because the required memory to store a distance matrix is 8 bytes times the squared number of plots (i.e. approximately 8 GB for 10,000 points). The maximum size of vectors, that is also that of the number of cells in a matrix, in R is 2147483647. If the number of points is greater the square root of this value, the distance matrix can't be created, whatever the available RAM.

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

```
if (par_points_nb <= 10000) {
  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
```

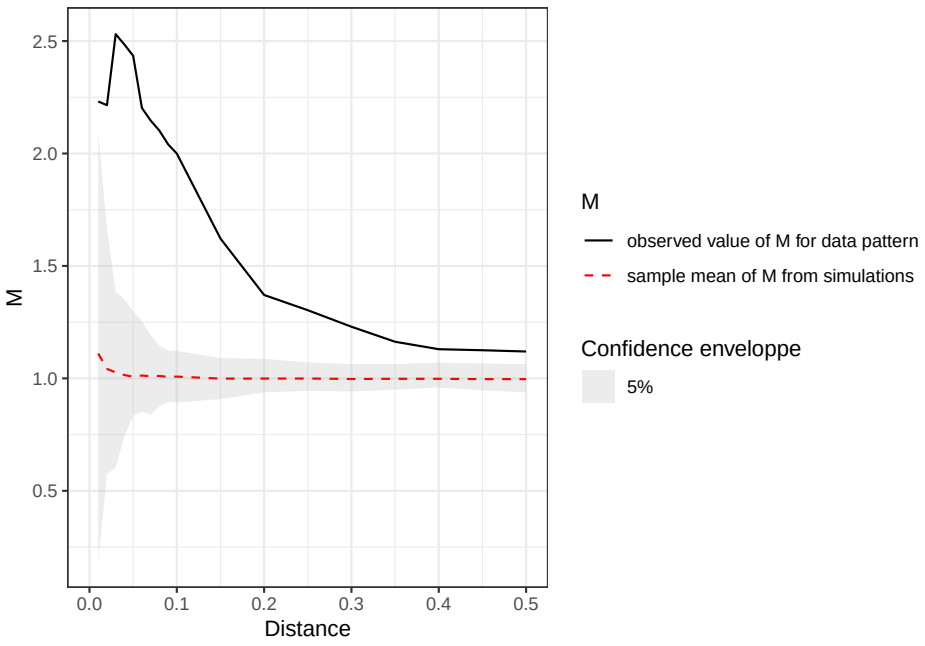
```
##
## $n
## [1] 2
##
## $marks
## $marks$PointType
## [1] Type1 Type2
## Levels: Type1 Type2
##
## $marks$PointWeight
## [1] 2 3
##
##
## attr(,"class")
## [1] "Dtable"
```

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

```
if (exists("d_matrix")) {
  identical(
    Mhat(X, r = r, ReferenceType = "Case", NeighborType = "Control"),
    Mhat(d_matrix, r = r, ReferenceType = "Case", NeighborType = "Control")
  )
}
```

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

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



3 Computational performance

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 Computing time

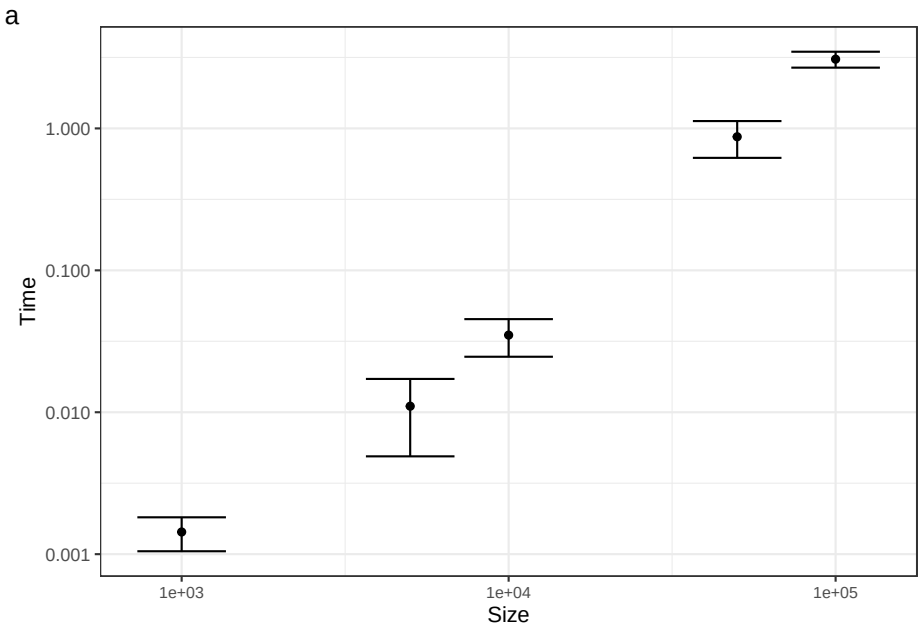
The time required for an exact calculation is evaluated for a range of numbers of points specified in `X_sizes`.

```
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 = 10L) %>%
    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() +
    labs(tag = "a") +
    theme(axis.text.x = element_text(size = 8))
```



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

```
# 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
##  0.3747 -0.3287 -0.3569  0.1169  0.1941
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -18.6997     0.9792   -19.1 0.000314
## logSize       1.7049     0.1027    16.6 0.000475
##
## (Intercept) ***
## logSize     ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3774 on 3 degrees of freedom
## Multiple R-squared:  0.9892, Adjusted R-squared:  0.9856
## F-statistic: 275.7 on 1 and 3 DF,  p-value: 0.0004755
```

The model is estimated from the average computation time of each point set size.

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. Distance matrices are thus useful only in relatively small datasets (less than 4.634×10^4 points by R limitations, possibly much less depending on available RAM) when distances are not euclidean.

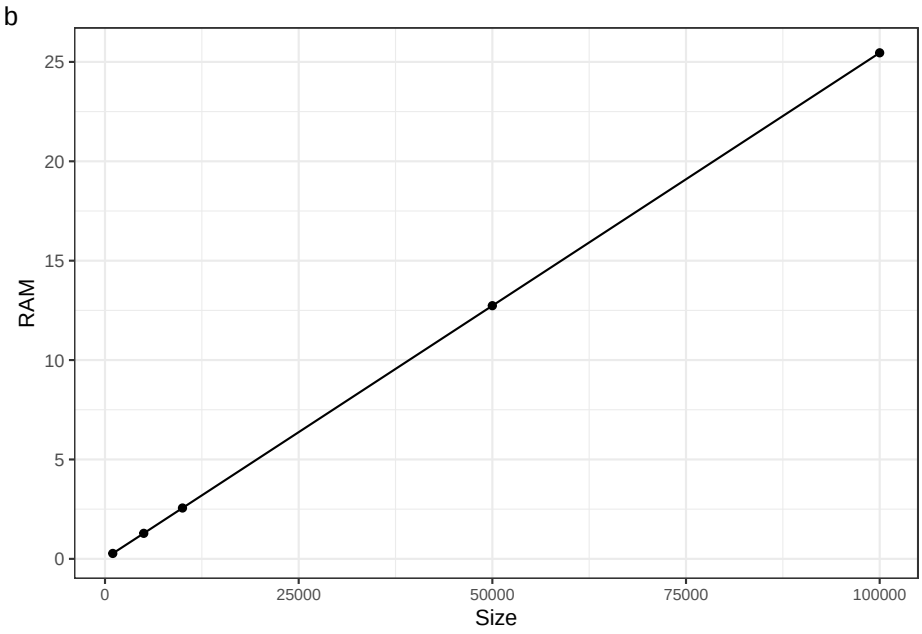
```
library("microbenchmark")
if (exists("d_matrix")) {
  mb <- microbenchmark(
    Mhat(X, r = r, ReferenceType = "Case", NeighborType = "Control"),
    Mhat(d_matrix, r = r, ReferenceType = "Case", NeighborType = "Control"),
    times = 10L
  )
  mb
}
```

```
## 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
##   6.329211  8.848087 12.75493 14.17036 14.89624
##  16.035387 19.755522 21.08945 20.17633 20.59746
##       max neval
##  17.94722    10
##  29.33255    10
```

3.2 Memory

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

```
# 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() +
    labs(tag = "b") +
    theme(axis.text.x = element_text(size = 8))
```



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
##  0.0032971 -0.0022080 -0.0011395 -0.0002819
##          5
##  0.0003323
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.435e-02  1.428e-03   10.05   0.0021
## Size        2.544e-04  2.842e-08 8953.87 3.07e-12
##
## (Intercept) **
## Size        ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.002397 on 3 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 8.017e+07 on 1 and 3 DF,  p-value: 3.072e-12
```

4 Effects of approximating the position of points

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

```
simulations_n <- 20
```

4.1 Test setting

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

The number of points of each point pattern is 100,000 in the article: several hours are necessary to compute the tests. In this appendix, it is set to 5000 in the parameter `par_points_comp_nb` to limit computer overhead and allow testing the code. It can be set to 100,000 to reproduce the tests of the article.

```
# Number of points for comparison simulations
par_points_comp_nb <- 5000 # 1E5 in the article
# Simulate Matérn point patterns
X_matern_list <- replicate(
  simulations_n,
  expr = X_matern(par_points_comp_nb),
  simplify = FALSE
)
# Group points
X_matern_grouped_list <- lapply(
  X_matern_list,
  FUN = group_points,
  partitions = par_partitions
)
# Simulate CSR point patterns
X_csr_list <- replicate(
  simulations_n,
  expr = X_csr(par_points_comp_nb),
  simplify = FALSE
)
# Group points and compute M
X_csr_grouped_list <- lapply(
  X_csr_list,
  FUN = group_points,
  partitions = par_partitions
)
```

The M function is computed on each point pattern.

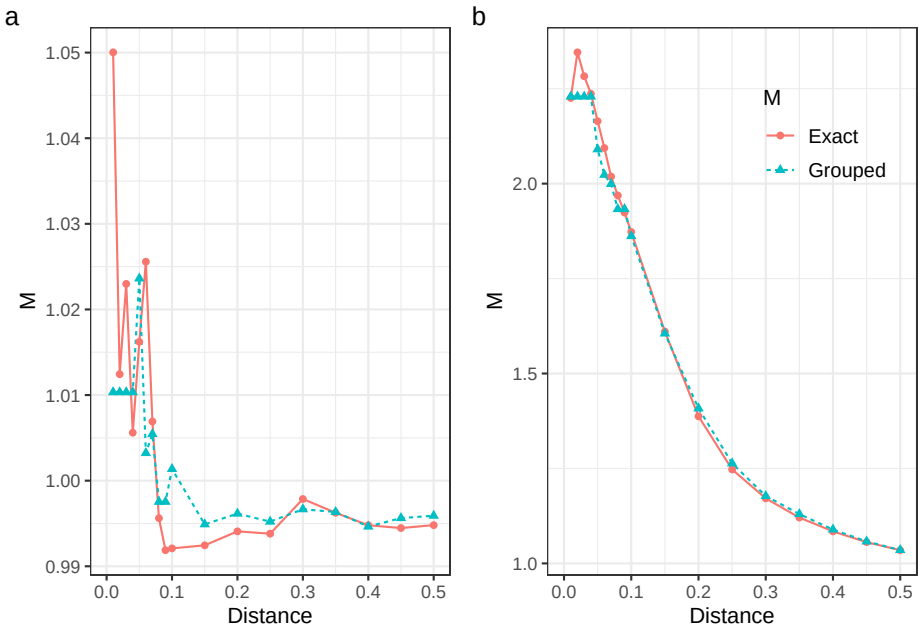
```
library("pbapply")
# Compute M
M_matern_original <- pbsapply(X_matern_list, FUN = X_to_M)
M_matern_grouped <- pbsapply(X_matern_grouped_list, FUN = X_to_M)
M_csr_original <- pbsapply(X_csr_list, FUN = X_to_M)
M_csr_grouped <- pbsapply(X_csr_grouped_list, FUN = X_to_M)
```

The average values of M across simulations are plotted.

```
library("gridExtra")
# Figure: exact and grouped M(r)
Mapproxfig <- function(M_original, M_grouped, tag) {
  tibble(
    r,
    Exact = rowMeans(M_original),
    Grouped =  rowMeans(M_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() +
    labs(x = "Distance", y = "M", tag = tag) +
    theme(
      legend.position = c(0.75, 0.8),
      legend.background = element_blank(),
      legend.text = element_text(size = 10)
    )
}

# Plot
grid.arrange(
  Mapproxfig(M_csr_original, M_csr_grouped, "a") +
    theme(legend.position = "none"),
  Mapproxfig(M_matern_original, M_matern_grouped, "b"),
  ncol = 2
)
```



4.2 Results

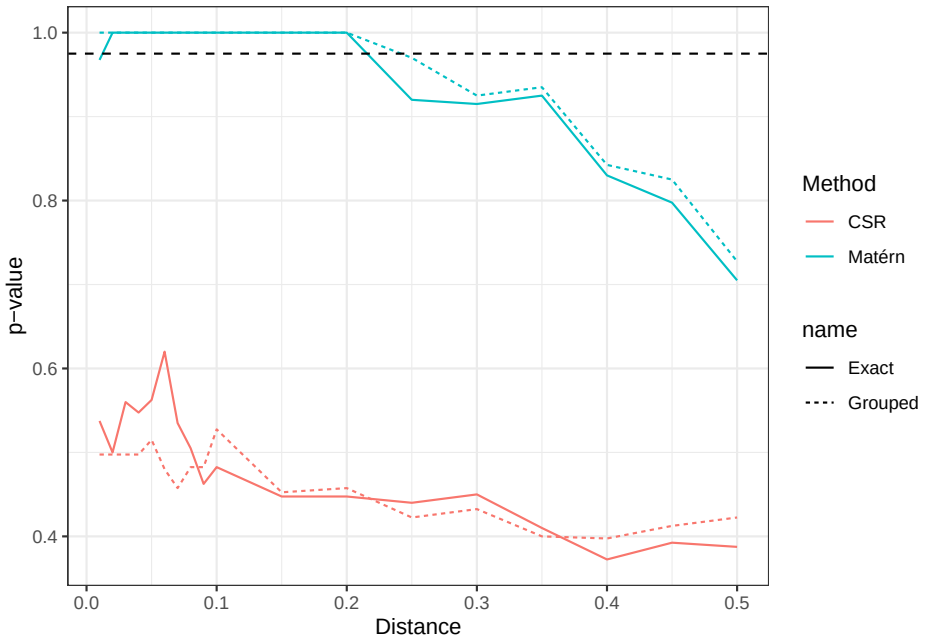
The following code compares the p-values to reject the null hypothesis of random location of points in both CSR and Matérn point patterns at each distance.

```
# Function to calculate the departure of the grouped point pattern M p-value
# against H0 from that of the real point pattern
p_departure <- function(
    X_list,
    M_simulations = simulations_n,
    verbose = interactive()
  ) {
  # Group the points
  X_grouped_list <- lapply(
```

```
    X_list,
    FUN = group_points,
    partitions = par_partitions
  )
  # Prepare arrays to save the results
  quantiles_exact <-
    quantiles_grouped <-
    matrix(NA, nrow = length(r) - 1, ncol = simulations_n)
  for (pp_i in seq_len(simulations_n)) {
    # Iterate across the point patterns
    if (verbose) cat("Point pattern", pp_i, "of", simulations_n,"\n")
    # Calculate M with simulations
    envelope_exact <- MEnvelope(
      X_list[[pp_i]],
      r = r,
      NumberOfSimulations = M_simulations,
      ReferenceType = "Case",
      parallel = TRUE,
      verbose = FALSE,
    )
    # Individual simulations are in an attribute.
    # Extract them in a dataframe, delete the first row (distances)
    # Each column contains M(r) for a simulation
    sims_exact <- as.data.frame(attr(envelope_exact, "simfuns"))[, -1]
    # Quantiles of each M(r) in the simulations
    quantiles_exact[, pp_i] <- sapply(
      # Remove distance 0
      seq_along(r)[-1],
      FUN = function(r_i) {
        # Quantile function of the simulated M(r)
        ecdf(as.numeric(sims_exact[r_i, ]))(envelope_exact$obs[r_i])
      }
    )
    # M grouped, idem
    envelope_grouped <- MEnvelope(
      X_grouped_list[[pp_i]],
      r = r,
      NumberOfSimulations = M_simulations,
      ReferenceType = "Case",
      parallel = TRUE,
      verbose = FALSE,
    )
    sims_grouped <- as.data.frame(attr(envelope_grouped, "simfuns"))[, -1]
    quantiles_grouped[, pp_i] <- sapply(
      # Remove distance 0
      seq_along(r)[-1],
      FUN = function(r_i) {
        # Quantile function of the simulated M(r)
        ecdf(as.numeric(sims_grouped[r_i, ]))(envelope_grouped$obs[r_i])
      }
    )
  }
  return(
    tibble(
      Exact = rowMeans(quantiles_exact),
      Grouped = rowMeans(quantiles_grouped),
    )
  )
}

# Parallelize M simulations
# (https://ericmarcon.github.io/dbmss/articles/parallel.html)
library("doFuture")
plan(multisession)
departure_csr <- p_departure(X_csr_list)
departure_matern <- p_departure(X_matern_list)
plan(sequential)
```

```r
# Plot the p-values
departure_csr %>%
  mutate(Pattern = "CSR", Distance = r[-1]) %>%
  bind_rows(
    departure_matern %>%
      mutate(Pattern = "Matérn", Distance = r[-1])
  ) %>%
  pivot_longer(Exact:Grouped) %>%
  ggplot() +
  geom_line(aes(x = Distance, y = value, color = Pattern, linetype = name)) +
  geom_hline(yintercept = .975, linetype = 2) +
  labs(color = "Method", y = "p-value")
```



4.3 Correlation between exact and approximated point patterns

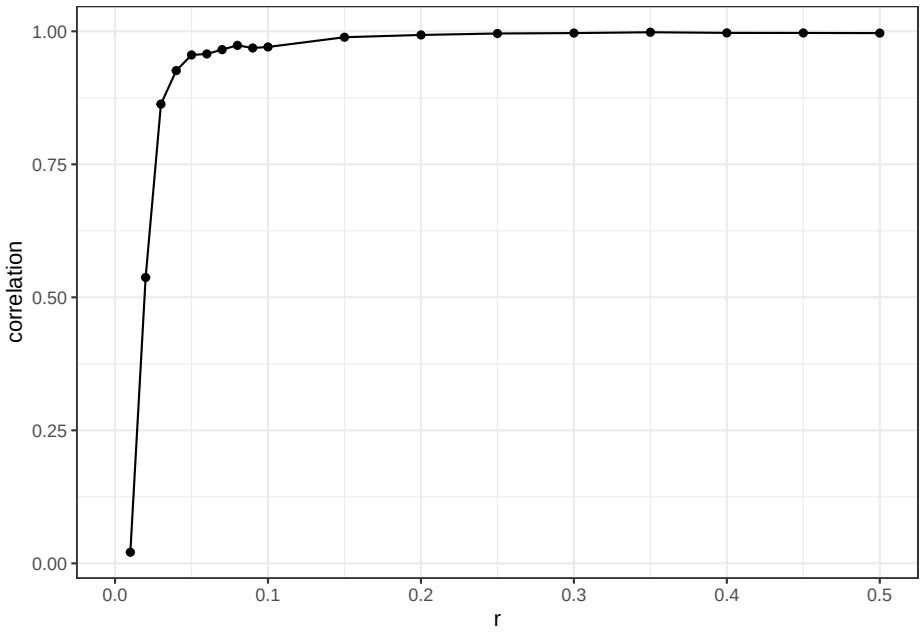
Tidu et al. (2024) tested the effect of grouping the points by the correlation between exact and approximated M values. To allow comparing our results to theirs, the same approach is applied below but it is not retained in the article because all correlations shown below mainly depend on the number of points.

4.3.1 Case of an aggregated distribution (Matérn)

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

```r
# Correlation
M_cor <- function(r_value, M_original, M_grouped) {
  r_index <- which(r == r_value)
  # Return
  c(
    # Distance
    r_value,
    # Correlation
```

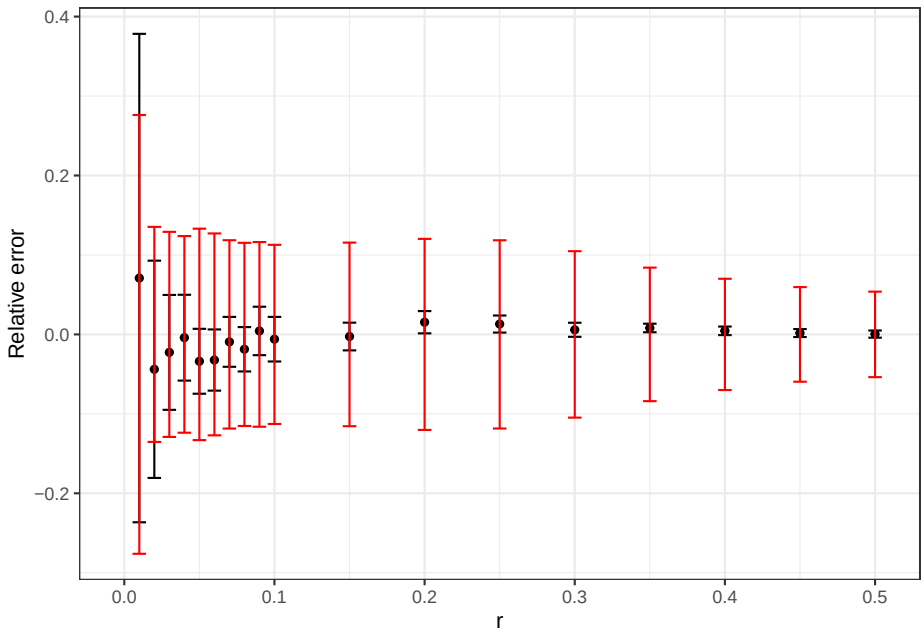
```
    cor(M_original[r_index, ], M_grouped[r_index, ])
  )
}
sapply(
  r,
  FUN = M_cor,
  M_original = M_matern_original,
  M_grouped = M_matern_grouped
) %>%
  t() %>%
  as_tibble() %>%
  rename(r = V1, correlation = V2) %>%
  ggplot(aes(x = r, y = correlation)) +
    geom_point() +
    geom_line()
```



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")
```



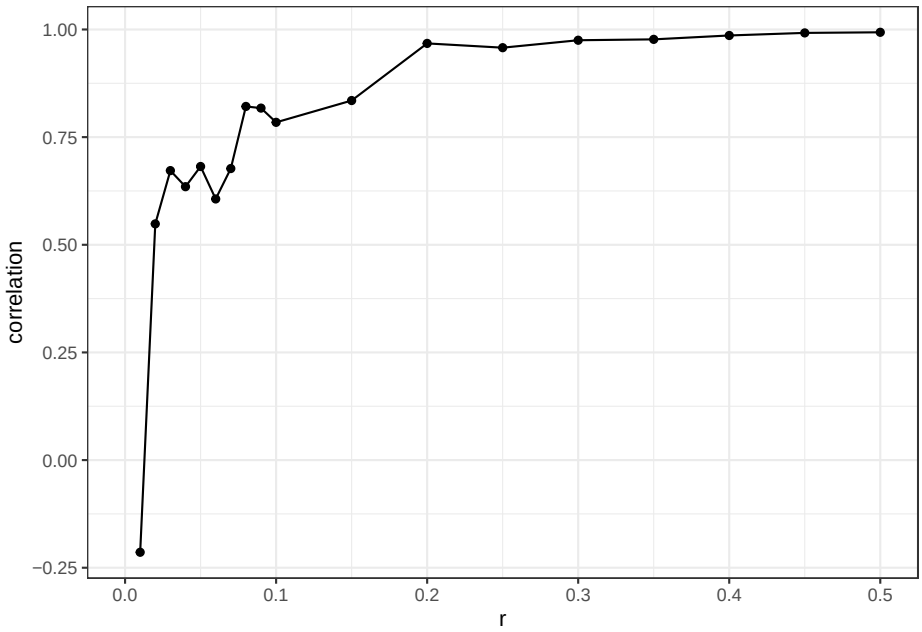
The figure above shows, in red, the variability of the value of M (its coefficient of variation) over the course of the simulations. By definition, the mean value is error-free. At short distances, the values of M vary greatly for the same point process, depending on the stochasticity of its runs. As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

4.4 Case of a completely random distribution (CSR)

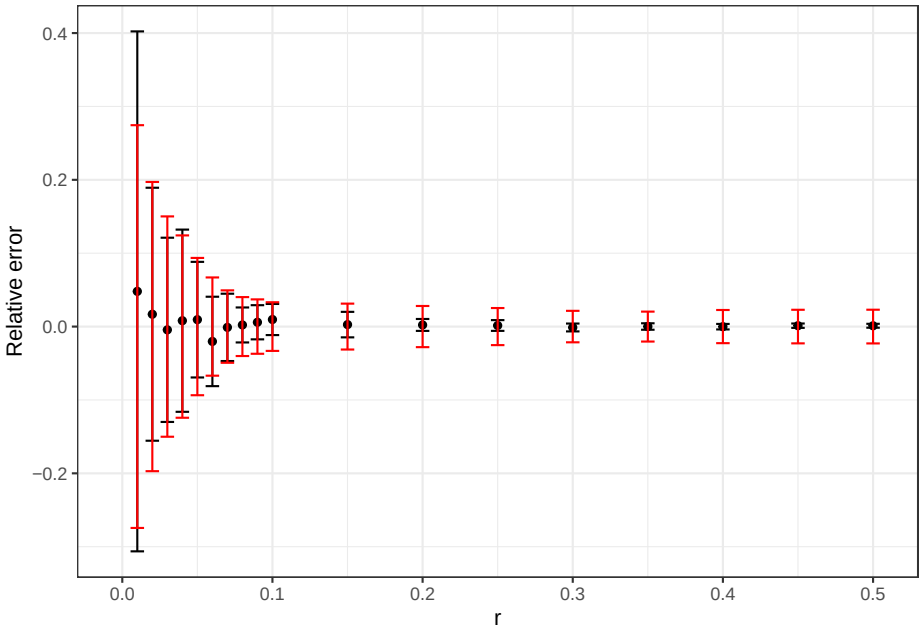
The correlation between the M values calculated by each method is calculated at each distance r.

```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.

```r
# 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.

5 Discussion and conclusion

The last code chunk predicts the computation time of large point patterns.

```
# Time to compute 100000 points, in seconds
(Mtime_1E5_s <- ceiling(M_time$Time[5]))
```

```
## [1] 4
```

```
# 1000 simulations, in minutes
(Mtime_1E5_1000_min <- 1000 * Mtime_1E5_s / 60)
```

```
## [1] 66.66667
```

```
# 5 million points, in minutes
(
  Mtime_5E6_min <- exp(
    predict(
      M_time_lm,
      newdata = data.frame(logSize = log(5E6))
    )
  ) / 60
)
```

```
##        1
## 33.26947
```

```r
# 1000 simulations, in days
(Mtime_5E6_1000_d <- 1000 * Mtime_5E6_min / 60 / 24)
```

```
##       1
## 23.1038
```

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., 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.