Simulating scenarios of feature distribution

Bernardo Niebuhr

03 December, 2021

Contents

Introduction

There are a few big questions that underlie the assessment of the effects of anthropogenic infrastructure on wildlife. When performing environmental impact assessments, one aims not only to to find which factors affect wildlife and how strongly, but also (i) at which spatial scale there are effects and (ii) how these effects sum and interact when (as it is often the case) there are multiple infrastructures and vectors of landscape modification. The first question is also knows as the Zone of Influence (ZoI) problem. The second is generally tackled in the context of cumulative impact assessment.

In the main text of this manuscript, we argue that the cumulative impact measure - based on the density of infrastructure features - might better represent the cumulative effect of multiple features in the landscape than considering only the distance to the feature nearest to a given location. This, however, might vary depending on the number of features in the landscape, how these features are distributed in space, and what is the ZoI of each of those features.

Here we simulate some landscapes with point-type infrastructure spread following different patterns, calculate the distance to the nearest feature and the cumulative impact (density) of features, at multiple scales, and assess when and how these variables might represent different sources of spatial variation.

Simulate landscapes

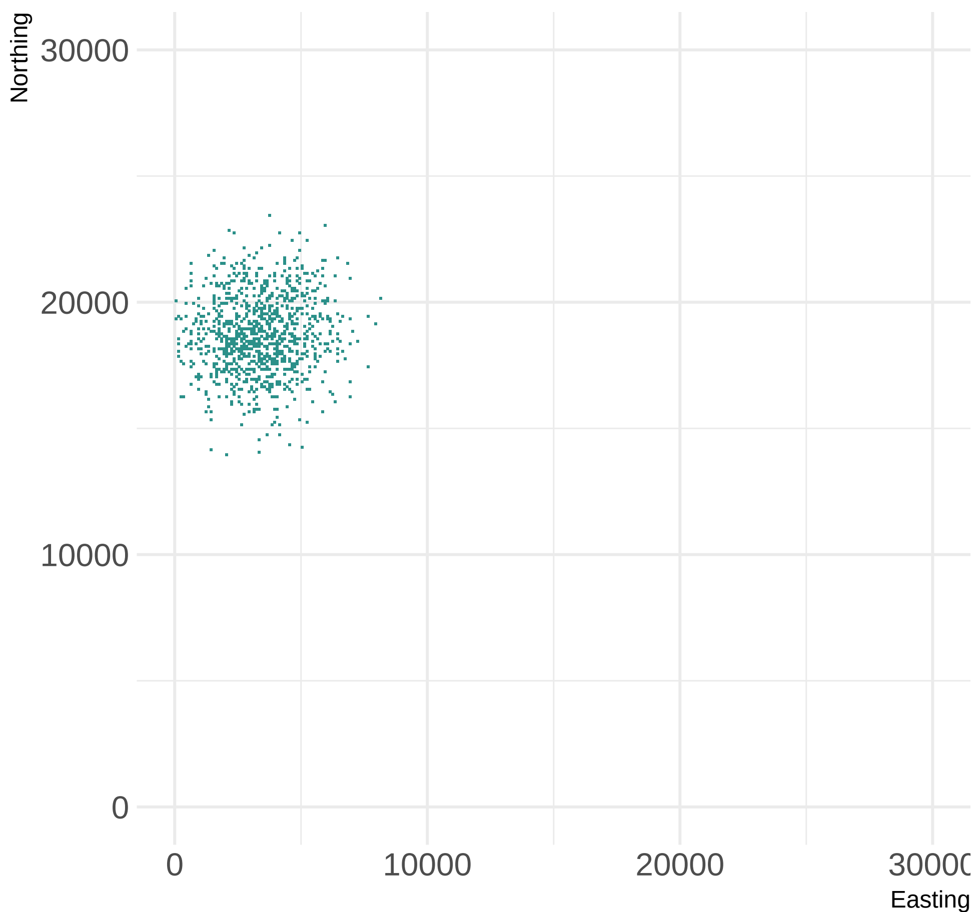
First we simulate some landscape using point-type infrastructure as an example. They could represent the spatial location of houses, cabins, or wind turbines, for example. To do this we'll use a few functions designed within the R package `oneimpact`. We also load other packages from data manipulation and plotting.

```
# Load packages

# data manipulation
library(dplyr)
library(purrr)
library(ggplot2)
library(ggpubr)
library(lsr) # for correlations

# spatial packages
library(mobsim)
library(raster)
library(landscapetools)
library(sf)
library(spatstat)
```

```r
# oneimpact package
library(oneimpact)
```

We set a 30x30 km^2 landscape and can simulate points following different spatial patterns. Here is an example landscape where the points are spread close to a single center (e.g. houses in a village).

```r
set.seed(1234)

# simulate a single patch
nfeat <- 1000 # number of features
ext <- 30000 # extension of the landscape
nc <- 1 # number of centers or patches
wd <- ext / 20 # width of the patch
pts <- set_points(
  n_features = 1000, centers = nc,
  width = wd, res = 100,
  extent_x = c(0, ext), extent_y = c(0, ext)
)

landscapetools::show_landscape(pts$rast, legend.position = "none")
```



We can now simulate landscapes with different patterns. We start with 3 spatial distribution of points: regular, random, and clumped distribution of points. For the clumped distribution, we use two scenarios, where the point features are spread in either 5 or only 1 focal patch. Each scenario is simulated with 10, 100, and 1000 points, so that we have 12 scenarios in total.

```r
set.seed(123)

# random, gradient, single center, multiple centers
methods <- c("regular", "random", "mobsim")
name <- c("regular", "random", "clumped5", "clumped1")
nfeat <- c(10, 100, 1000) # number of features
res <- 100 # resolution
ext <- 30000 # extent of the landscape
nc <- c(5, 1) # number of centers for clumped
wd <- c(0.05) * ext # width of the "patches"

# parameters
parms_df1 <- expand.grid(
  method = methods, n_features = nfeat,
  centers = nc[1], width = wd
) # first 3 scenarios
parms_df2 <- expand.grid(
  method = methods[3], n_features = nfeat,
  centers = nc[2], width = wd
) # parameters for clumped1
parms_df <- dplyr::bind_rows(parms_df1, parms_df2) %>%
  dplyr::arrange(n_features, method)
# names of scenarios
scenarios <- paste0(rep(name, 3), "_", rep(nfeat, each = 4))

# simulate points
pts <- parms_df %>% purrr::pmap(set_points,
  res = res,
  extent_x = c(0, ext),
  extent_y = c(0, ext),
  buffer_around = 10000
)

landscapes <- purrr::map(pts, ~ .[[2]])
names(landscapes) <- scenarios
```

Here we visualize the scenarios with 1000 features.

```r
# show landscapes with n_features = 1000
# plot(landscapes, col = "black", nc = 4, legend = F)
# rasterVis::levelplot(stack(landscapes), layout = c(4,3), names.attr = scenarios,
#                       par.settings = GrTheme, colorkey = FALSE)
landscapetools::show_landscape(landscapes[9:12])
```

We also store some variables related the distance between points for each of these scenarios. These variables are: (i) the average distance between points, (ii) the average nearest neighbor distance, and (iii) the average isolation. The average is isolation is defined here as the mean nearest neighbor distance between random points created in the landscapes and the simulated point feature locations.

```
# isolation to random points
isolation <- function(x, n_rand = 100, ext = c(0, 1), lonlat = FALSE) {
  # create random points
  rand <- data.frame(x = runif(n_rand, ext[1], ext[2]), y = runif(n_rand, ext[1], ext[2]))
  # calc dist
  dists <- pointDistance(x, rand, lonlat = lonlat)
  # min dist (nearest neighbor)
  apply(dists, 2, min)
}

# mean isolation
mean_isolation <- function(x, n_rand = 100, ext = c(0, 1), lonlat = FALSE) {
  mean(isolation(x, n_rand = n_rand, ext = ext, lonlat = lonlat))
}

# points
pts_coords <- purrr::map(pts, first)
# names(pts_coords) <- scenarios

# calculate distances
dist_scenarios <- data.frame(
  scenario = scenarios,
  spatial_dist = rep(name, 3),
  n_features = rep(nfeat, each = 4),
```

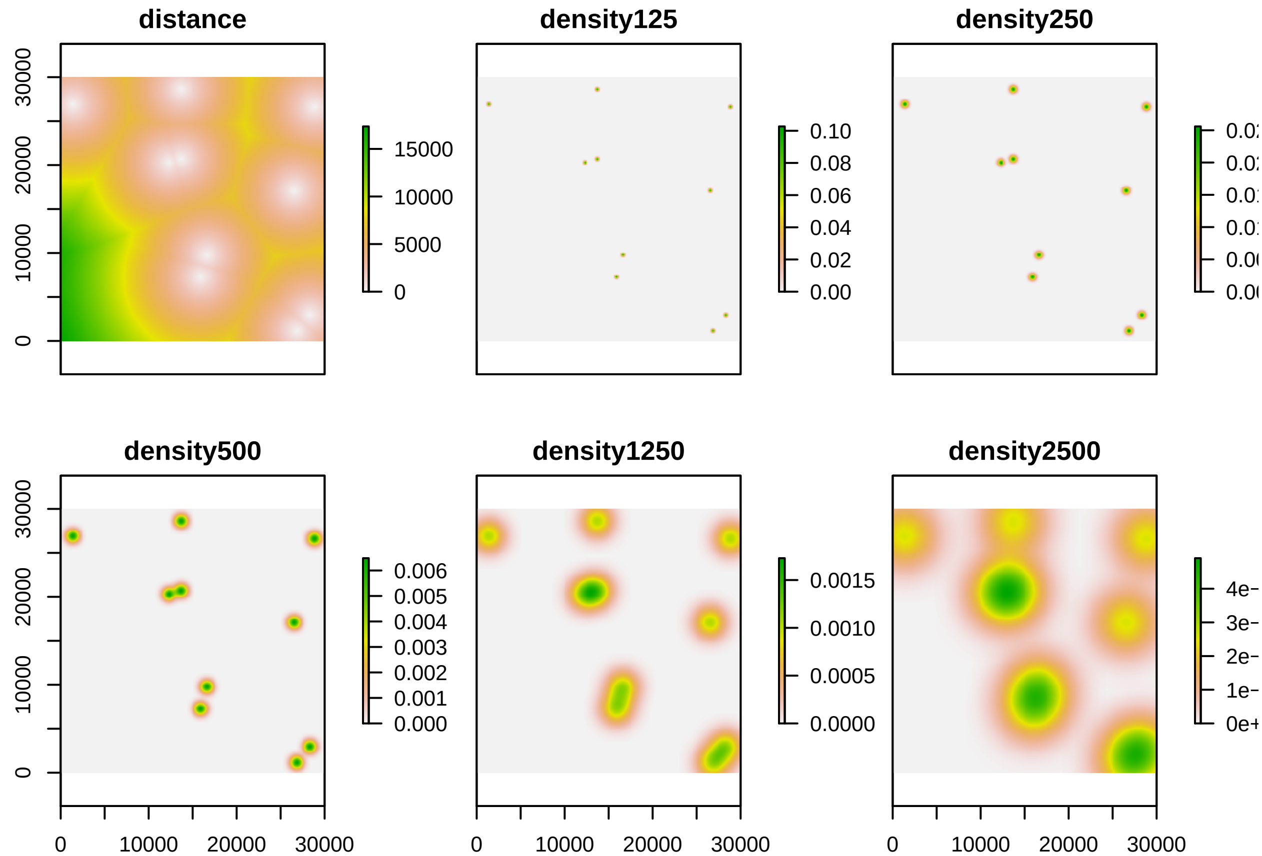
```r
  mean_dist = map_dbl(pts_coords, function(x) mean(dist(x))),
  mean_nndist = map_dbl(pts_coords, function(x) mean(spatstat.geom::nndist(x))),
  mean_isolation = map_dbl(pts_coords, mean_isolation, n_rand = 150, ext = c(0, ext))
)

dist_scenarios
```

```
##         scenario spatial_dist n_features mean_dist mean_nndist mean_isolation
## 1     regular_10      regular         10 15510.736   9486.8330      3910.7559
## 2      random_10       random         10 17450.119   5238.7641      5320.3333
## 3    clumped5_10     clumped5         10 15197.147   1155.7217      8224.4716
## 4    clumped1_10     clumped1         10  3029.684   1132.5368     11103.3612
## 5    regular_100      regular        100 15717.795   3000.0000      1182.9837
## 6     random_100       random        100 15252.246   1597.3085      1585.0777
## 7   clumped5_100     clumped5        100  9454.507    668.2567      5048.0218
## 8   clumped1_100     clumped1        100  2454.158    319.6537     13629.1097
## 9   regular_1000      regular       1000 15590.481    948.6833       371.6643
## 10   random_1000       random       1000 15765.710    486.1710       479.7201
## 11 clumped5_1000     clumped5       1000 14179.990    208.8593      4900.2276
## 12 clumped1_1000     clumped1       1000  2662.354    115.1179     10947.7893
```

Calculate distance and cumulative effects

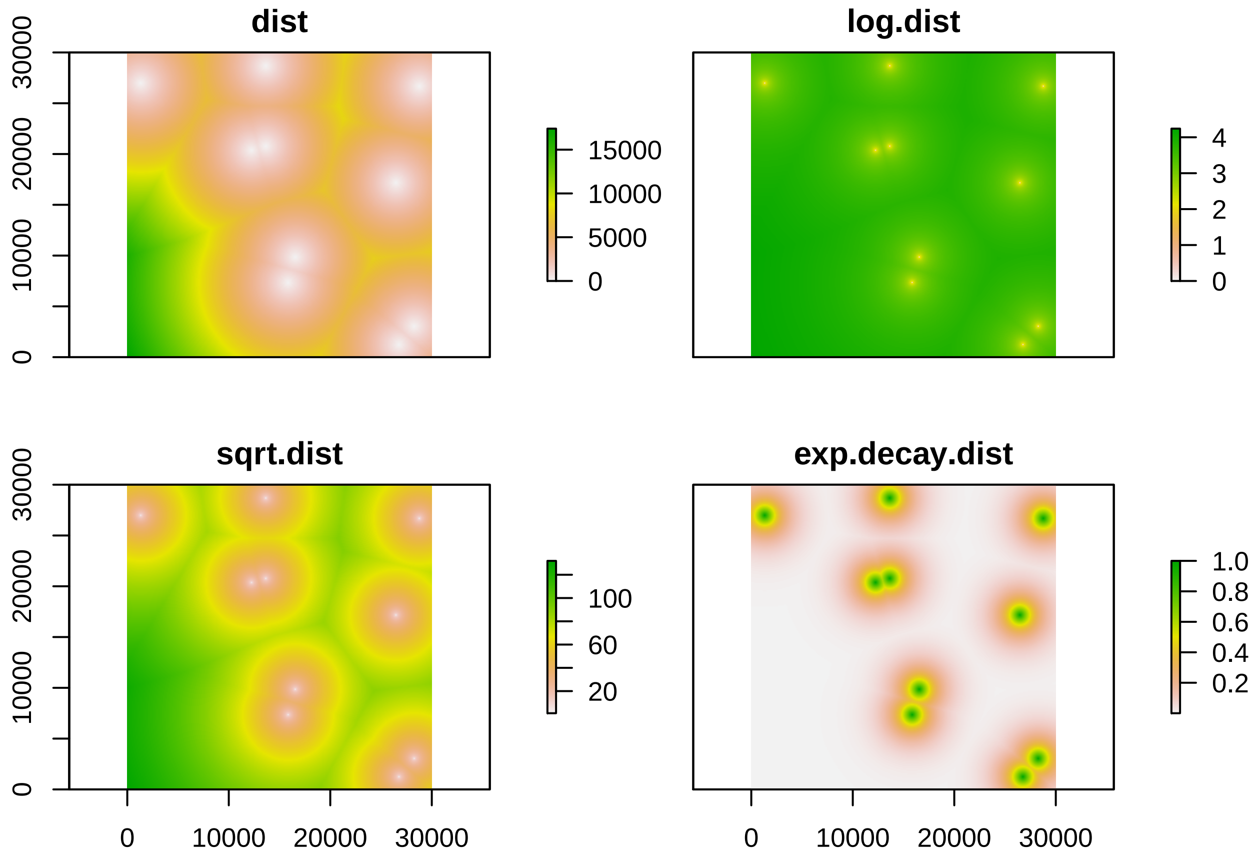
First we illustrate, for one of those scenarios, how the maps change as we calculate either the distance to the nearest feature or the density of features at different scales. For illustration purposes, we use here a Gaussian filter, where the scale corresponds to the standard deviation σ of the Gaussian distribution, and the moving window has a size of $3 \cdot \sigma$.

```r
# calculate distance and density at multiple scales for one input
scales <- c(250, 500, 1000, 2500, 5000) / 2 # scales for Gaussian filter
dist_dens1 <- calc_dist_dens(landscapes[[2]],
  type_density = "Gauss", scale = scales,
  extent_x_cut = c(0, ext), extent_y_cut = c(0, ext)
)
plot(dist_dens1, nc = 3)
```



We can see that the results are different if one transforms the distance variable, as it is generally done in ecological models. Below we show, as an example, how it looks like when the distances are log- and sqrt-transformed, or when an exponential decay distance is used instead. It also affects how distances and densities correlate, as we'll see.

```
# transformed distance
log_dist <- calc_dist(landscapes[[2]],
  transform_dist = "log", log_base = 10,
  extent_x_cut = c(0, ext), extent_y_cut = c(0, ext)
)
sqrt_dist <- calc_dist(landscapes[[2]],
  transform_dist = "sqrt",
  extent_x_cut = c(0, ext), extent_y_cut = c(0, ext)
)
exp_decay_dist <- calc_dist(landscapes[[2]],
  transform_dist = "exp_decay",
  exp_hl = 1000,
  extent_x_cut = c(0, ext), extent_y_cut = c(0, ext)
)

# combine
dists <- raster::stack(dist_dens1[[1]], log_dist, sqrt_dist, exp_decay_dist)

# plot
names(dists) <- c("dist", "log-dist", "sqrt-dist", "exp-decay-dist")
plot(dists, nc = 2)
```

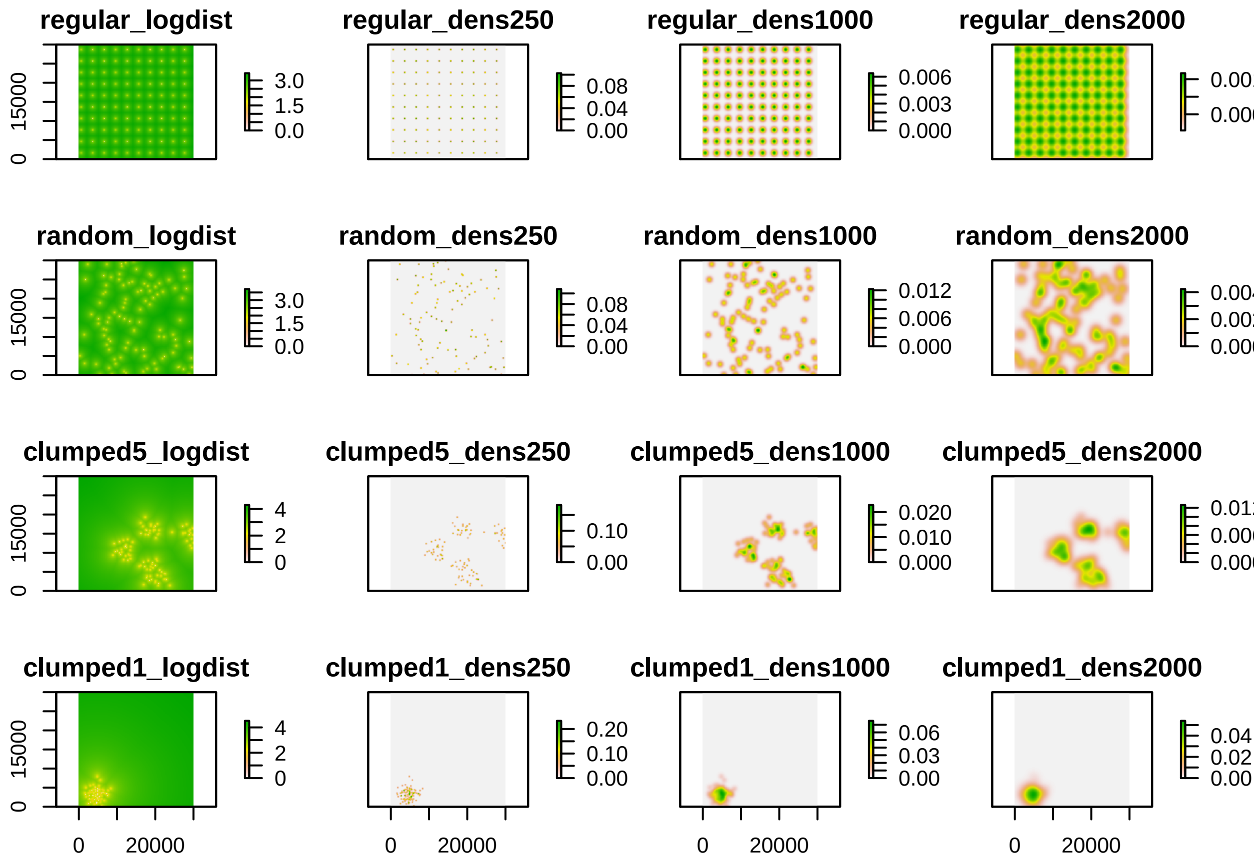


Visual comparison

Now we compare these variables visually for the different spatial point patterns, using the log-transformed distance, for now. We select the log-distance since it is a measure commonly used in ecological studies to include distance to the nearest features into the statistical models.

```
# redefine scales
scales <- c(250, seq(500, 5000, by = 500)) / 2 # scales for Gaussian filter

# calculate distance and density at multiple scales for each input
dist_dens <- purrr::map(landscapes, calc_dist_dens,
  type_density = "Gauss", scale = scales,
  # type_density = "circle", scale = scales,
  transform_dist = "log", log_base = 10,
  extent_x_cut = c(0, ext), extent_y_cut = c(0, ext)
)

# change names according to the scenario and the variable
vars <- c("logdist", paste0("dens", scales * 2))

for (i in 1:length(dist_dens)) {
  names(dist_dens[[i]]) <- paste(strsplit(names(dist_dens)[i], "_")[[1]][1], vars, sep = "_")
}

# slice a few of them to plot - for
slices <- c(1, 2, 4, 6) # as.vector(outer(c(1,2,4,6), (0:3)*6, FUN = "+"))
maps100 <- dist_dens[5:8]
```

```r
stk <- stack(maps100[[1]][[slices]], maps100[[2]][[slices]], maps100[[3]][[slices]], maps100[[4]][[slices]]
plot(stk, nc = 4)
```

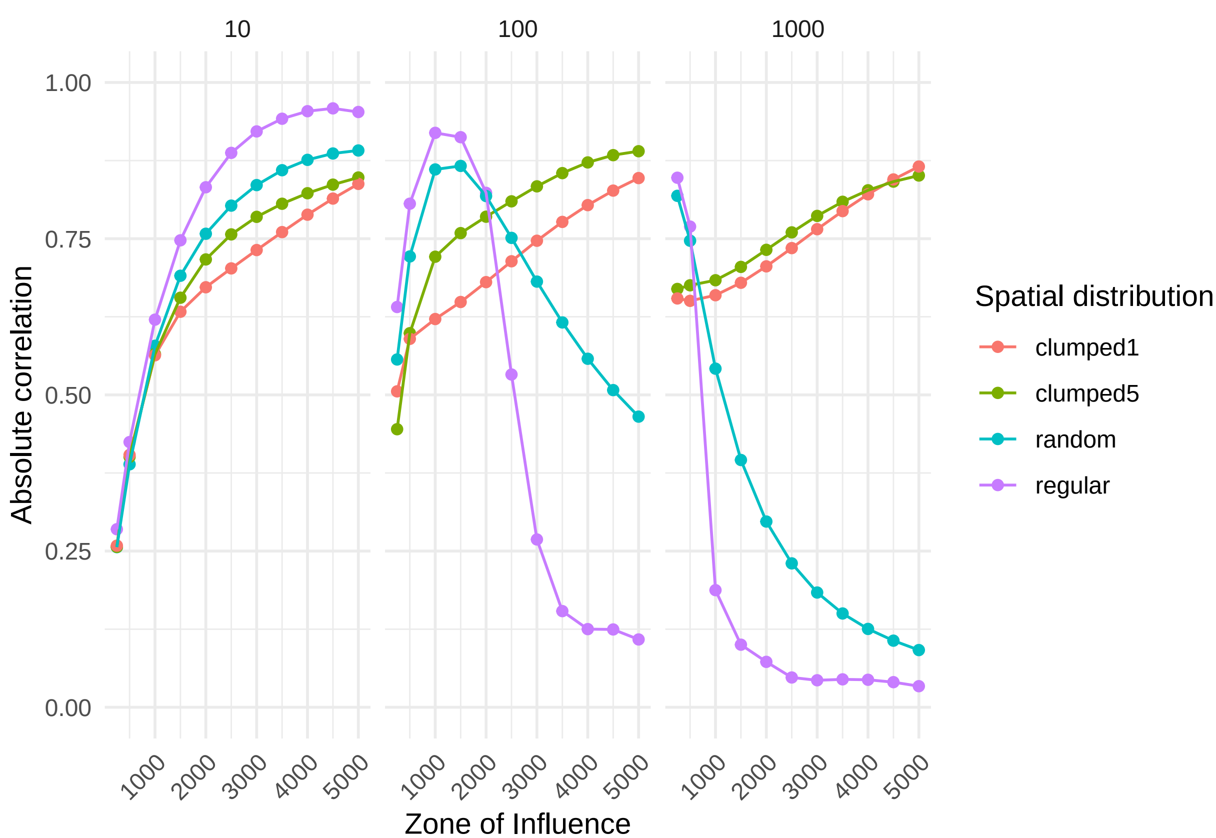


Calculate correlations between distance and cumulative impact (density)

Now we are going to check how correlated these variables are in space, to evaluate how much they can have different ecological interpretations.

```r
# Calculate pairwise Pearson correlation coefficient
# rast_cor <- layerStats(dist_dens, stat = "pearson")[[1]]
#
# # Extract the ones which are interesting for us
# dist_dens_cor <- rbind(rast_cor[1, 2:7], rast_cor[8, 9:14],
#                        rast_cor[15, 16:21], rast_cor[22, 23:28])
# colnames(dist_dens_cor) <- vars[2:7]
# rownames(dist_dens_cor) <- types
#
# # print
# dist_dens_cor

# put distances with rasters
scenarios_analysis_full <- dist_scenarios %>%
  tibble::as_tibble() %>%
  dplyr::mutate(
    rasts = dist_dens,
    rasts_df = purrr::map(rasts, as.data.frame),
    corr_raw = purrr::map(rasts_df, lsr::correlate),
    corr_list = purrr::map(corr_raw, function(x) x$correlation[1, 2:12]),
```

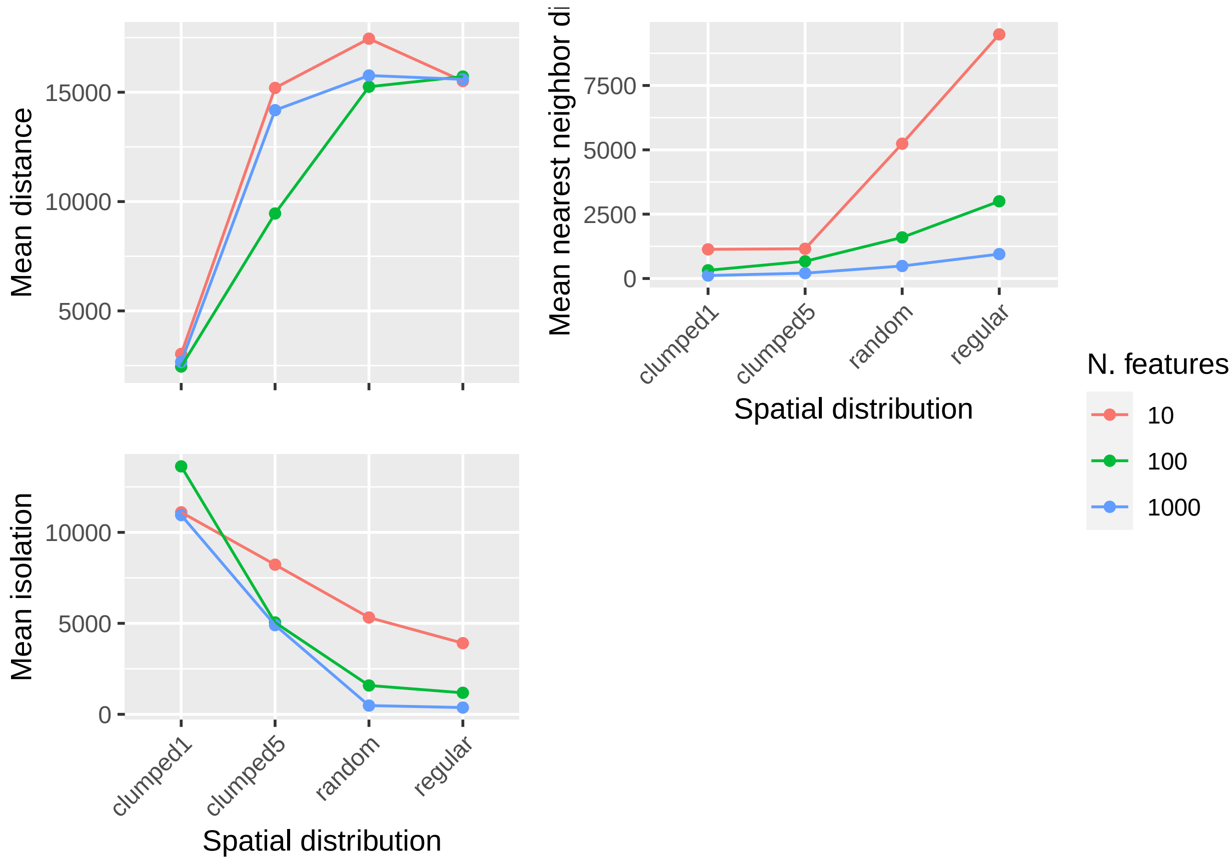
```
    corr_df = purrr::map(corr_list, tibble::enframe, name = "dens", value = "corr")
  ) %>%
  tidyr::unnest(corr_df) %>%
  dplyr::mutate(
    variable = gsub(".*_", "log-dist_", dens),
    scale = as.numeric(gsub("\\D+", "", variable))
  )

scenarios_analysis <- scenarios_analysis_full %>%
  dplyr::select(-c(rasts:corr_list)) %>%
  dplyr::mutate(n_features = factor(n_features))

# plot
scenarios_analysis %>%
  # dplyr::filter(n_features == 100) %>%
  ggplot(aes(scale, abs(corr), color = spatial_dist)) +
  geom_point() +
  geom_line() +
  ylim(0, 1) +
  facet_wrap(~n_features) +
  labs(
    x = "Zone of Influence",
    y = "Absolute correlation",
    color = "Spatial distribution"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1))
```



We can see that the pattern changes greatly with both the total number of features and their spatial distribution. For small density of features,

Now we use, instead of the name of the scenario, the average distance measures taken for each, to see if any response can be better interpreted in this terms. First, below we see how these variables change for each scenario and density of features.

```
# relationship between scenarios
ds1 <- dist_scenarios %>%
  ggplot(aes(x = spatial_dist, y = mean_dist, color = factor(n_features))) +
  geom_point() +
  geom_line(aes(group = n_features)) +
  labs(
    x = "",
    y = "Mean distance",
    color = "N. features"
  ) +
  theme(axis.text.x = element_blank())
ds2 <- dist_scenarios %>%
  ggplot(aes(x = spatial_dist, y = mean_nndist, color = factor(n_features))) +
  geom_point() +
  geom_line(aes(group = n_features)) +
  labs(
    x = "Spatial distribution",
    y = "Mean nearest neighbor dist.",
    color = "N. features"
  ) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1))
ds3 <- dist_scenarios %>%
  ggplot(aes(x = spatial_dist, y = mean_isolation, color = factor(n_features))) +
  geom_point() +
  geom_line(aes(group = n_features)) +
  labs(
    x = "Spatial distribution",
    y = "Mean isolation",
    color = "N. features"
  ) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1))

# combine
ggpubr::ggarrange(ds1, ds2, ds3,
  nrow = 2, ncol = 2, common.legend = T,
  legend = "right"
)
```

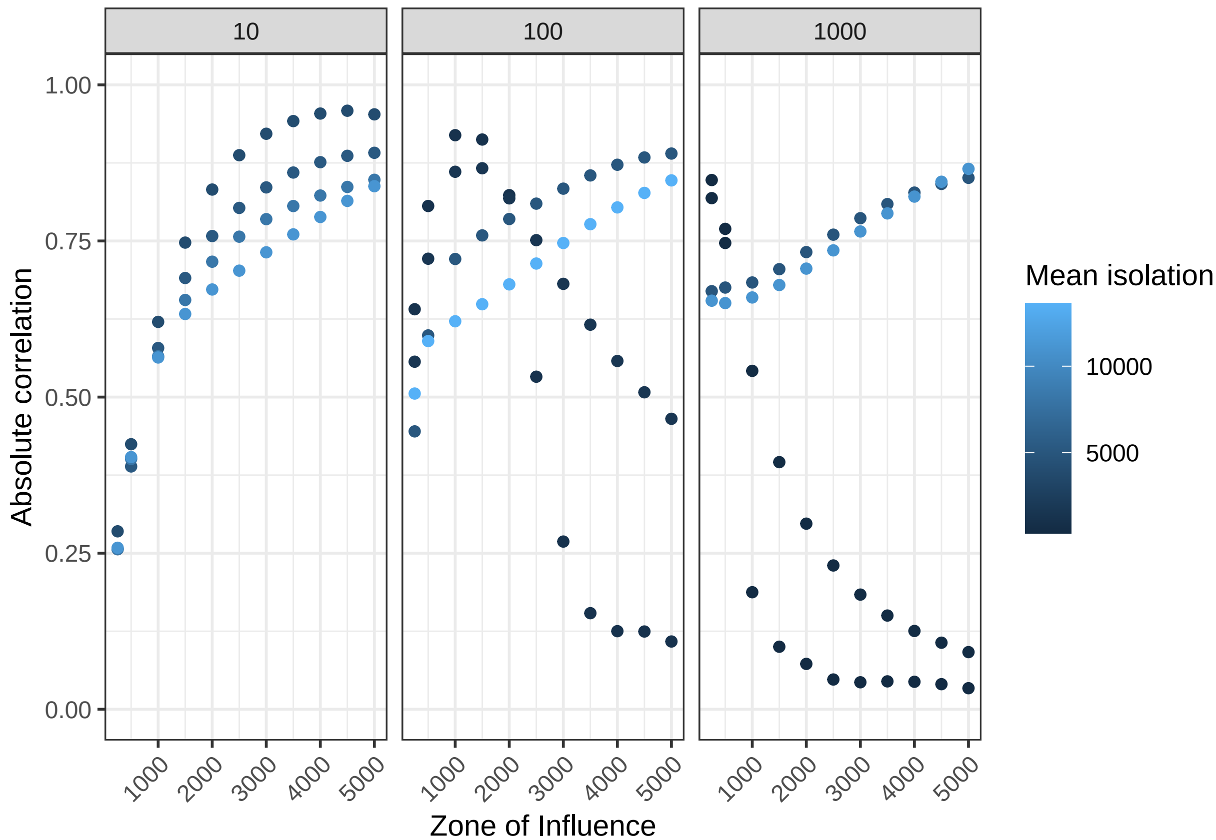


As expected, the mean distance between points and the mean distance to the nearest neighbor point increase when we depart from a more clumped scenario to a random or regular distribution. They represent the distance relationship between points only, regardless of the size of the landscape

The mean isolation follows an opposite pattern: it decreases when we depart from clumped to random or regular distribution of points. In all cases, even though the general pattern is qualitatively the same, quantitatively these quantities vary with the number of features - obviously the distances or isolation are larger when the density of points is smaller.

We take the last measure - mean isolation - to represent the relationship between the spatial distribution of points and the correlation between log-distances and cumulative impact (density) measures.

```
scenarios_analysis %>%
  # dplyr::filter(n_features == 1000) %>%
  ggplot(aes(scale, abs(corr), color = mean_isolation)) +
  geom_point() +
  ylim(0, 1) +
  labs(
    x = "Zone of Influence",
    y = "Absolute correlation",
    color = "Mean isolation"
  ) +
  facet_wrap(~n_features) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1))
```



Include interpretation here.

Discussion