TMA4250-Project-2

Ole Riddervold, Ole Kristian Skogly

2023-03-06

Loading data

```
cells   <- read.table("data/cells.dat", skip=3, col.names=c('x', 'y'))
redwood <- read.table("data/redwood.dat", skip=3, col.names=c('x', 'y'))
pines   <- read.table("data/pines.dat", skip=3, col.names=c('x', 'y'))
```

Exploring data

```
head(cells)
```

```
##       x     y
## 1 0.350 0.025
## 2 0.062 0.362
## 3 0.938 0.400
## 4 0.462 0.750
## 5 0.462 0.900
## 6 0.737 0.237
```

```
head(redwood)
```

```
##       x     y
## 1 0.364 0.082
## 2 0.898 0.082
## 3 0.864 0.180
## 4 0.966 0.541
## 5 0.864 0.902
## 6 0.686 0.328
```

```
head(pines)
```

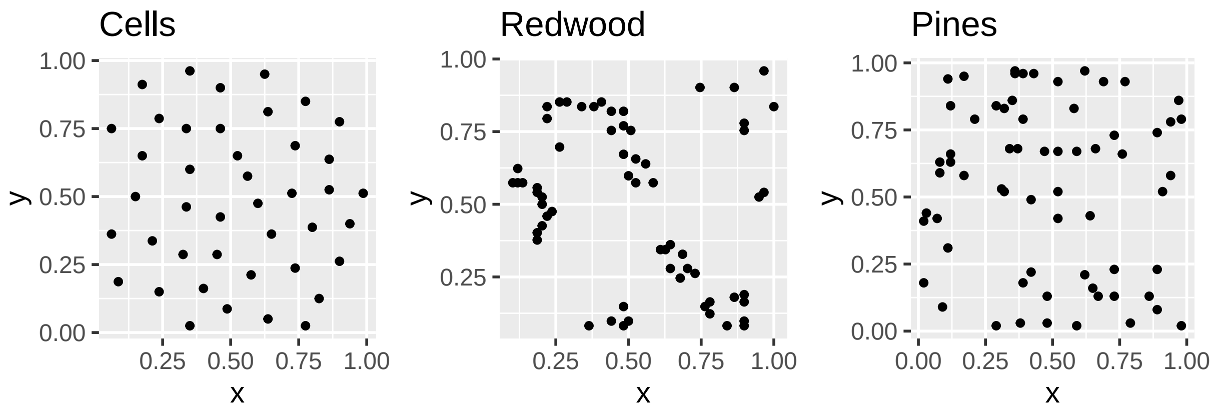
```
##      x    y
## 1 0.09 0.09
## 2 0.59 0.02
## 3 0.86 0.13
## 4 0.42 0.22
## 5 0.02 0.41
## 6 0.08 0.59
```

Problem 1

a)

```r
display_point_patterns <- function(dataframes, titles) {
  # Plots point patterns for dataframes with point data
  # Args: Dataframes and titles as lists of the same lengths
  # Returns: None
  n <- length(dataframes)
  do.call(
    wrap_plots,
    lapply(1:n, function(i) {
      ggplot(dataframes[[i]]) + geom_point(aes(x, y), size=1) +
        xlab('x') + ylab('y') + labs(title=titles[i]) +
        coord_fixed()
    })
  )
}
```

```r
plot_titles = c(
  "Cells",
  "Redwood",
  "Pines"
)

# png("figures/1_a.png", width=3600, height=1200, units="px", res=300) # Saving

display_point_patterns(list(
  cells,
  redwood,
  pines
), plot_titles)
```



For the Cells dataset, the points seem to be uniformly spread across the whole window. This would make sense since cells exist in very large quantities, and repel each other to a certain degree (https://www.hcplive.com/view/cells_repel). This will result in a uniform distribution of points when we pick a small window.

Comparing the last two dataset, it is evident that redwood trees have a tendency to grow in clusters along streams and other bodies of water (https://www.fs.usda.gov/research/treesearch/41153), which is very evident in the point patern plotted above. Pines, however, tend to follow a more random pattern of growth (https://www.researchgate.net/publication/227015905_The_problem_of_accuracy_in_environmental_analysis), giving the random pattern that is seen in the rightmost plot of Figure (FIGREF).

b)

The L-function is defined by

$$L(r) = \sqrt[d]{\frac{K(r)}{b_d}} \tag{1}$$

where

$$K(r) = \frac{1}{\lambda}\mathbb{E}_{\mathbf{0}}\left[N\left(B_r(\mathbf{0}\backslash\{\mathbf{0}\})\right)\right] \tag{2}$$

$$b_d = \nu(B_1(\mathbf{0})) \tag{3}$$

, $B_r(\mathbf{x})$ denotes a ball of radius r centered in $\mathbf{x}$ in the relevant space (here $\mathbb{R}^2$), and ν denotes the volume-function. The K-function $K(r)$ can be interpreted as the ratio between the expected number of points in a ball centered at $\mathbf{0}$ excluding the origin itself, *given* that there is a point at the origin, and the rate of the point process itself. Therefore one can say that the L-function is a variant of the K-function that takes the curse of dimensionality, i.e. the fact that points get further apart the larger the dimension, into account. In $\mathbb{R}^2$, we have

$$L(r) = \sqrt{(}\frac{K(r)}{\pi}) \tag{4}$$

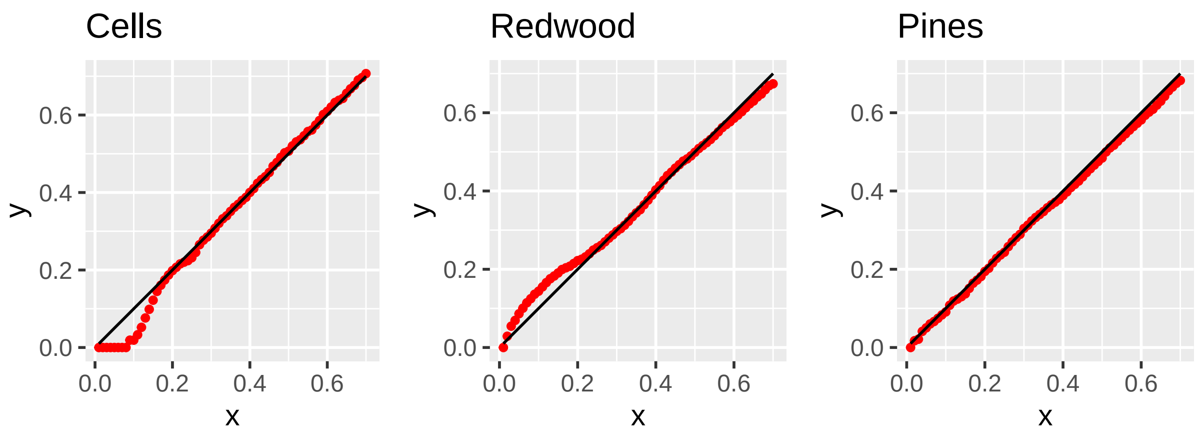
Estimation of the L-function:

```
pp.cells   <- ppinit("data/cells.dat")
pp.redwood <- ppinit("data/redwood.dat")
pp.pines   <- ppinit("data/pines.dat")

L.cells   <- data.frame(Kfn(pp.cells, fs=1.0)[1:2])
L.redwood <- data.frame(Kfn(pp.redwood, fs=1.0)[1:2])
L.pines   <- data.frame(Kfn(pp.pines, fs=1.0)[1:2])
```

L-function plotting:

```
display_L_functions <- function(dataframes, titles) {
  # Plots K-functions and affine lines with growth 1 for comparison
  # Args: Dataframes and titles as same length lists
  # Returns: None
  n <- length(dataframes)
  do.call(
    wrap_plots,
    lapply(1:n, function(i) {
      ni <- nrow(dataframes[[i]])
      df <- dataframes[[i]] %>% mutate(xline=1:ni/100, yline=1:ni/100)
      ggplot(df) + geom_point(aes(x, y), size=1, col="red") +
        xlab('x') + ylab('y') + labs(title=titles[i]) +
        geom_line(aes(xline, yline)) +
        coord_fixed()
    })
  )
}
```

```
# png("figures/1_b.png", width=3600, height=1200, units="px", res=300) # Saving

display_L_functions(list(
  L.cells,
  L.redwood,
  L.pines
), titles=plot_titles)
```



If, for small distances, the L-function diverges from the theoretical straight function (a straight line), then that is a implication that there is either clustering or repulsion in the dataset. For each of the datasets provided: - Cells: The L-function goes below the theoretical L-function, meaning that there are fewer points than expected in the ball with small r. This implies that there is repulsion in the point process, and a Gibbs process might be more suitable. - Redwood: The L-function is slightly higher than the theoretical function for small r, which is an implication of clustering, as discussed in 1a), and a Neymann-Scott process may therefore be more suitable. - Pines: The estimated L-function is mostly a straight line corresponding to the theoretical function, implying that a Poisson point process is a suitable model.

c)

For a homogeneous Poisson point process on a window $W \subseteq \mathbb{R}^m$, we have the convenient result:

$$[\mathbf{x}_1. \cdots, \mathbf{x}_n | N(W) = n] \overset{\text{i.i.d.}}{\sim} U(W) \tag{5}$$

In this case, $m = 2$ and $W = [0, 1] \times [0, 1]$. The points are therefore very easy to simulate, as they follow a uniform distribution on W.

```r
simulate_R2_unif <- function(n, m) {
  # Simulates a homogeneous Poisson point process on R^2 in m realizations on [0, 1]x[0, 1].
  # Args: n (amount of points to simulate in each realization) and m (amount of realizations)
  # Returns: mxnx2 array.
  dims <- c(m, n, 2)
  realizations <- array(runif(m*n*2), dims)
  return(realizations)
}
```

```r
calculate_R2_L_functions <- function(realizations) {
  # Given realizations of some point process in the form of an array, calculates
  # the L-function for each realiztion and returns as an array.
  # Args: realizations (mxnx2 array)
  #         - m: Amount of realzations
  #         - n: Amount of points per realization
  # Returns: mxn-array
  d <- dim(realizations)
  L <- array(rep(0, prod(d[1:2])), d[1:2])
  for (i in 1:d[1]) {
    df <- data.frame(realizations[i, , ])
    colnames(df) <- c('x', 'y')
    L[i, ] <- Kfn(df, fs=1.0)$y
  }
  return(L)
}
```

```r
R2_L_function_stats <- function(L_functions) {
  # Given an array of m L-functions at n points, calculates mean and 90%
  # confidence intervals.
  # Args: L_functions (mxn)
  # Returns: Dataframe of L-functions
  stats <- L_functions %>% melt() %>%
    group_by(Var2) %>%
    summarize(
      mean=mean(value),
      CI90=qt(0.05, length(value))*sd(value)*sqrt(1 + 1/length(value))
    )
  return(stats)
}
```

```r
simulate_R2_ppp_Lfn <- function(n, m) {
  # Simulates a Poisson point process conditional on N = n, and returning
  # a dataframe containing empirically estimated L-function and 90% CI.
  return(
    simulate_R2_unif(n, m) %>%
      calculate_R2_L_functions() %>%
      R2_L_function_stats()
  )
}
```

```r
display_L_functions_with_ppp_sim <- function(dataframes, titles, m=100) {
  # Plots K-functions and affine lines with growth 1 for comparison
  # Args: Dataframes and titles as same length lists
  # Returns: None
  n <- length(dataframes)
  do.call(
    wrap_plots,
    lapply(1:n, function(i) {
      ni <- nrow(dataframes[[i]])
      simi <- simulate_R2_ppp_Lfn(n=ni, m=m)
      df <- dataframes[[i]] %>% mutate(Lmean=simi$mean, LCI=simi$CI90)
      ggplot(df) + geom_point(aes(x, y), size=1, col="red") +
        geom_line(aes(x, Lmean)) +
        geom_ribbon(aes(x, ymin=Lmean-LCI, ymax=Lmean+LCI), alpha=0.25) +
        xlab('x') + ylab('y') + labs(title=titles[i]) +
        coord_fixed()
    })
  )
}
```

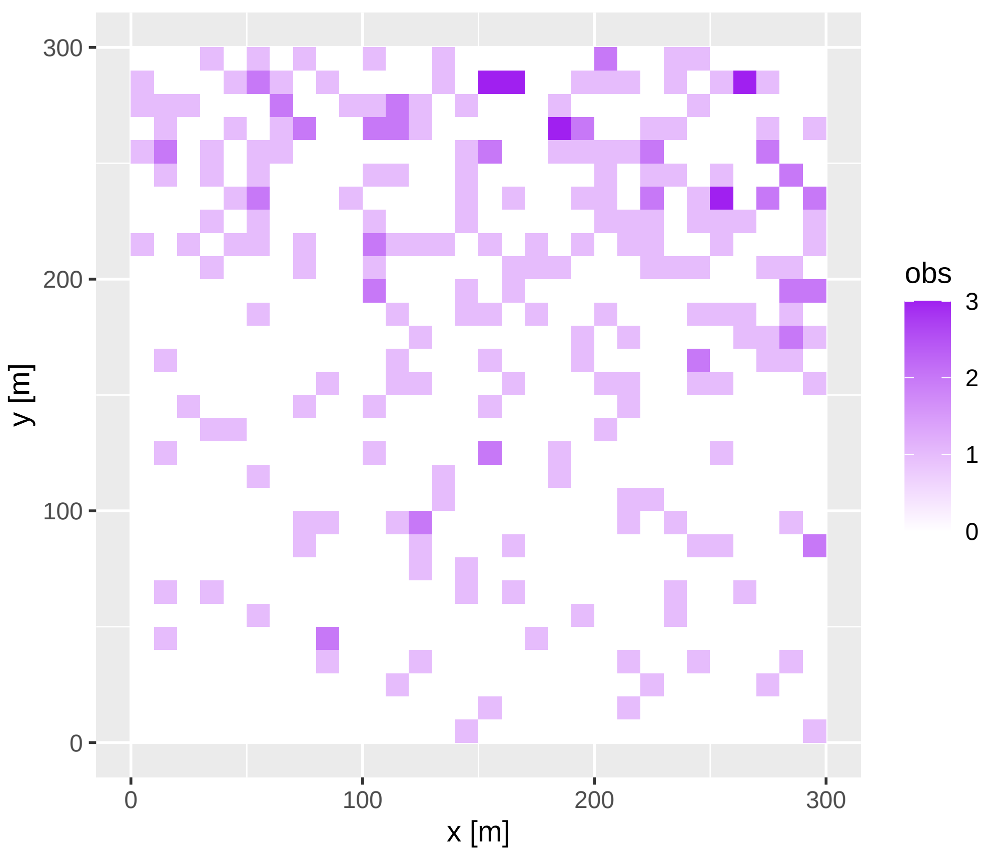
```
png("figures/1_c.png", width=3600, height=1200, units="px", res=300) # Saving

display_L_functions_with_ppp_sim(list(
  L.cells,
  L.redwood,
  L.pines
), titles=plot_titles)
```

Problem 2

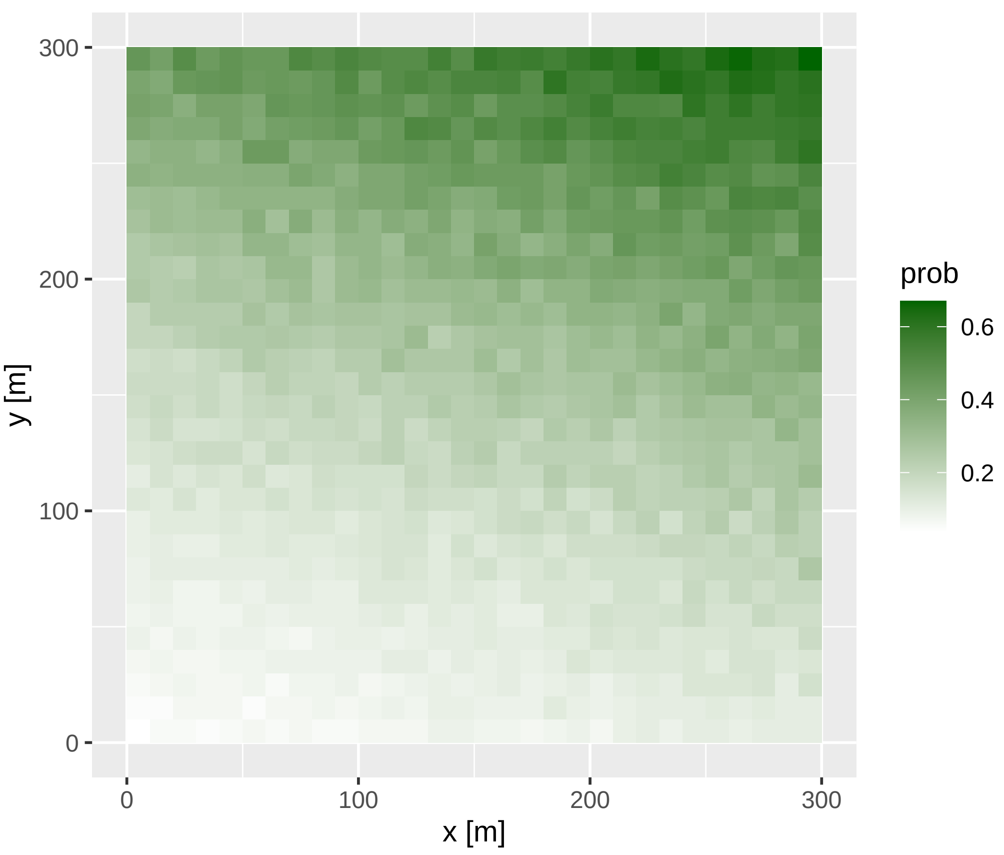
a)

```
obspines <- read.table("data/obspines.txt", skip=1, col.names=c('x', 'y', 'obs'))
obsprob <- read.table("data/obsprob.txt", skip=1, col.names=c('x', 'y', 'prob'))
```

```
# png("figures/2_a1.png", width=1200, height=1200, units="px", res=300) # Saving
ggplot(obspines) +
  geom_tile(aes(x, y, fill=obs)) + scale_fill_gradient(low="white", high="purple") +
  xlab('x [m]') + ylab('y [m]') +
  coord_fixed()
```



```
# png("figures/2_a2.png", width=1200, height=1200, units="px", res=300) # Saving
ggplot(obsprob) +
  geom_tile(aes(x, y, fill=prob)) + scale_fill_gradient(low='white', high='darkgreen') +
  xlab('x [m]') + ylab('y [m]') +
  coord_fixed()
```



b)

Assuming that the pine trees follow a Poisson point process, which supports the conclusions made in 1). It can be proved that if $\alpha : W \to [0,1]$ denotes detection probabilities on W (which in this case is defined as $W = [0,300] \times [0,300]$), and $N(\mathbf{x})$ is a homogeneous Poisson P.P. with rate λ_N then the observed points $M(\mathbf{x})$ follow an inhomogeneous Poisson P.P. with rate given by

$$\lambda_M(\mathbf{x}) = \lambda_N \alpha(\mathbf{x}) \ \forall\, \mathbf{x} \in W \tag{6}$$

Thus, we get that

$$M|N \sim \text{Poisson}(\nu_{\lambda_M}(W)) \tag{7}$$

$$\nu_{\lambda_M}(W) = \lambda_N \int_W \alpha_(\mathbf{x}) \mathrm{d}\mathbf{x} \tag{8}$$

Consequently, the point locations are distributed uniformly with pdf

$$f(\mathbf{x}_1, \cdots, \mathbf{x}_m | M(W) = m) = \prod_{i=1}^{m} \frac{\lambda_M(\mathbf{x}_i)}{\nu_{\lambda_M}(W)} = \left(\frac{\lambda_N}{\nu_{\lambda_M}(W)}\right)^m \prod_{i=1}^{m} \alpha(\mathbf{x}_i) \tag{9}$$

c)

We want to determine C such that

$$\hat{\Lambda} = \frac{1}{C} \sum_{i,j} M_{ij} \tag{10}$$

is unbiased.

Estimate from data:

```r
remote_estimate_lambda <- function(obs, prob, grid_size) {
  # Estimates lambda based on observed points and detection probabilities
  # Args: obs (Amount of observations in a gridpoint), alpha (detection probabilities)
  # Returns: (float) Estimate of lambda
  C <- grid_size*sum(prob) # Normalizing constant
  lambda_hat <- sum(obs)/C
  return(lambda_hat)
}
```

```r
grid_size <- 10^2 # For this particular problem
lambda_hat <- remote_estimate_lambda(obspines$obs, obsprob$prob, grid_size)
lambda_hat
```

```
## [1] 0.0104077
```

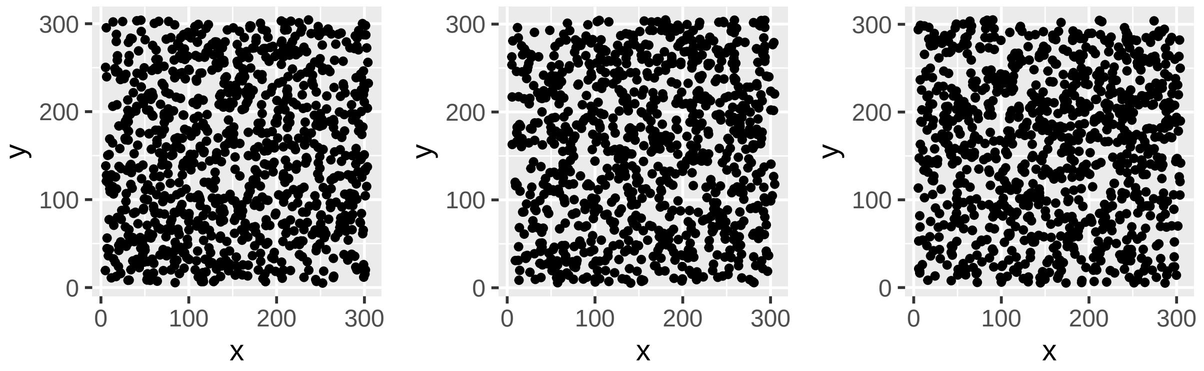
With the given data, the estimate is approximately $\hat{\lambda}_N = 0.0104$.

The following algorithm takes a grid in the form of a dataframe with x- and y-coordinates and corresponding rate λ, and simulates a homogeneous P.P.P. in each point using the rate.

```r
simulate_pois_dyn_rate <- function(lambda, displacement = 0) {
  # Given some changing rate lambda, draws corresponding amount of samples
  # for each rate.
  # Args:
  #   lambda (array-like): Varying rate of size m
  # Returns:
  #   Poisson samples of size m
  k <- length(lambda)
  return(
    lapply(
      1:k,
      function(i) rpois(1, lambda[i])
    ) %>% unlist() + displacement
  )
}

simulate_ppp_on_grid <- function(grid, poisson_displacement=0) {
  # Simulates a inhomogeneous Poisson point process on a square grid.
  # Args:
  #   grid  (Dataframe): Coordinates and rate (columns x, y, lambda)
  #   cell_size (float): Length of the given cell
  # Returns:
  #   Matrix of points
  k <- nrow(grid)
  n <-  simulate_pois_dyn_rate(grid$lambda, poisson_displacement) # BETTER WAY OF DOING THIS?
  points <- array(
    runif(2*sum(n), min=0, max=cell_size),
    c(sum(n), 2)
  ) %>% data.frame()
  colnames(points) <- c('x', 'y')
  # Adding x- and y-values to each point:
  ind <- 0
  for (i in 1:k) {
    if (n[i] > 0) {
      for (j in 1:n[i]) {
        points$x[ind+j] <- points$x[ind+j] + grid$x[i]
        points$y[ind+j] <- points$y[ind+j] + grid$y[i]
      }
      ind <- ind + n[i]
    }
  }
  return(points)
}
```

```r
cell_size <- 10
prior.grid <- obsprob %>% select(c('x', 'y')) %>% mutate(lambda=cell_size^2*lambda_hat)
prior.points_homogeneous_ppp <- replicate(3, simulate_ppp_on_grid(prior.grid), simplify=FALSE)
```

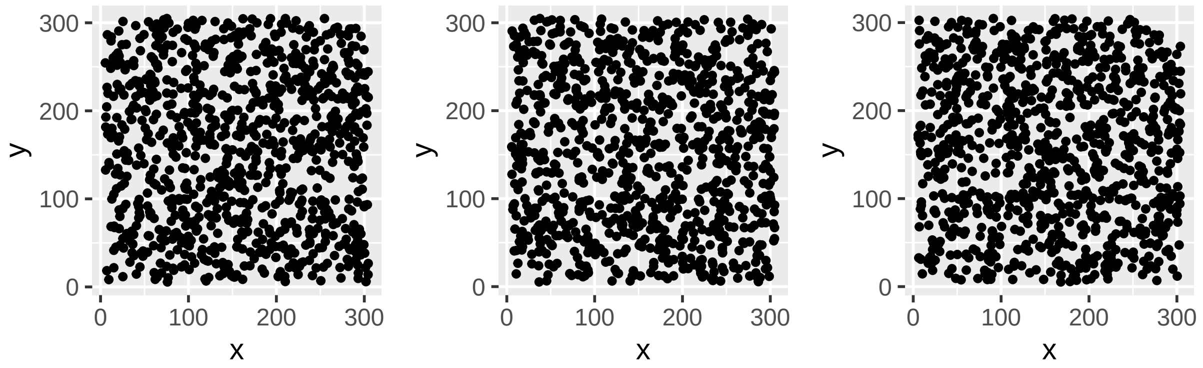
```r
# png("figures/2_c.png", width=1200, height=1200, units="px", res=300) # Saving

display_point_patterns(
  prior.points_homogeneous_ppp,
  titles=c("", "", "")
)
```



d)

```r
posterior.grid <- obsprob %>% mutate(lambda=cell_size^2*lambda_hat*(1-prob)) %>% select(-prob)
posterior.points_inhomogeneous_ppp <- replicate(3, simulate_ppp_on_grid(
  posterior.grid,
  poisson_displacement=obspines$obs
), simplify=FALSE)
```

```r
# png("figures/2_d.png", width=1200, height=1200, units="px", res=300) # Saving

display_point_patterns(
  posterior.points_inhomogeneous_ppp,
  titles=c("", "", "")
)
```



It is clear that this point pattern is slightly more centered towards the "unknown" lower left of W, the area where α is small, and points are harder to detect. Nevertheless, even though the process is centered around this area due to the rate given by $\lambda_N(1-\alpha(\mathbf{x}))$, the points are still mostly uniformly spread, since the Poisson distribution for $\mathbf{N}|\mathbf{M}$ is displaced by m, which needs to be compensated for after simulation from the distribution by adding the observations $\mathbf{M}$. Naturally, $\mathbf{M}$ will tend to be larger where α is larger, which evens out the skew which can be found in the derived distribution.

e)

```r
n <- 500

# Prior:
prior.sim <- replicate(n, simulate_pois_dyn_rate(prior.grid$lambda)) %>% melt() %>% # Simulating and cr
  group_by(Var1) %>% summarize(mean=mean(value), std=sd(value)) %>%                 # Grouping and gett
  bind_cols(obsprob %>% select(c('x', 'y'))) %>% select(-Var1)                      # Concatenating gri
prior.sim
```

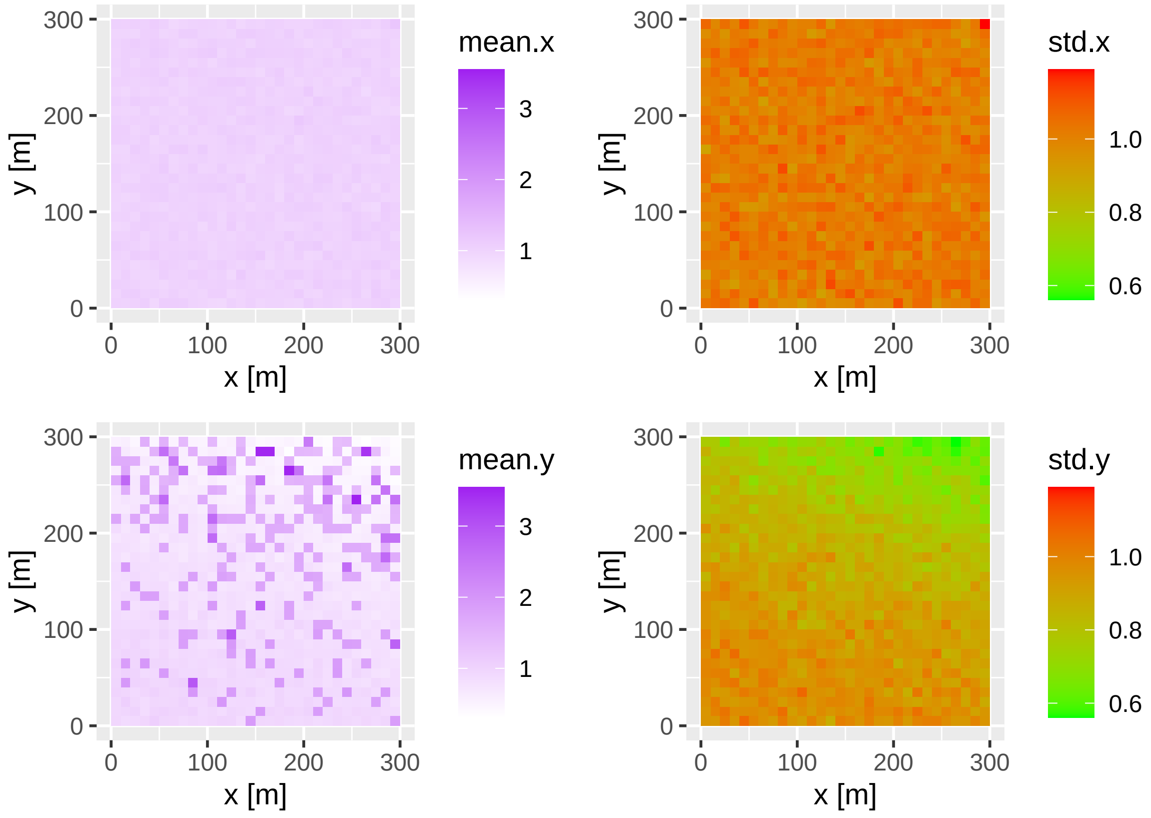
```
## # A tibble: 900 x 4
##     mean   std     x     y
##    <dbl> <dbl> <int> <int>
##  1 1.03  1.04      5     5
##  2 1.03  1.05     15     5
##  3 1.05  1.07     25     5
##  4 1.10  1.03     35     5
##  5 0.954 0.985    45     5
##  6 1.12  1.10     55     5
##  7 1.11  0.977    65     5
##  8 1.10  1.01     75     5
##  9 1.01  0.967    85     5
## 10 1.00  0.974    95     5
## # ... with 890 more rows
```

```r
# Posterior:
posterior.sim <- replicate(n, simulate_pois_dyn_rate(posterior.grid$lambda, displacement=obspines$obs))
  group_by(Var1) %>% summarize(mean=mean(value), std=sd(value)) %>%
  bind_cols(obsprob %>% select(c('x', 'y'))) %>% select(-Var1)
posterior.sim
```

```
## # A tibble: 900 x 4
##     mean   std     x     y
##    <dbl> <dbl> <int> <int>
##  1 0.958 0.948     5     5
##  2 0.892 0.943    15     5
##  3 0.918 1.02     25     5
##  4 0.968 0.943    35     5
##  5 1.08  1.07     45     5
##  6 0.976 0.995    55     5
##  7 0.966 0.958    65     5
##  8 0.914 0.932    75     5
##  9 0.942 1.02     85     5
## 10 0.918 0.945    95     5
## # ... with 890 more rows
```

```r
df_merged <- merge(prior.sim, posterior.sim, by=c('x', 'y'))
# colnames(df_merged)[3:length(df_merged)] <- c(TeX("Prior $\\hat{\\mu}$"), TeX("Prior $\\hat{\\sigma}$
# df_merged

generate_boundary_indices <- function(fnc) {
  # Generates either min or max for mean and std respectively to fit the
  # plotting scheme below of df_merged.
  val <- rep(0, 4)
  val[c(1, 3)] <- df_merged %>% select(c('mean.x', 'mean.y')) %>% fnc()
  val[c(2, 4)] <- df_merged %>% select(c('std.x', 'std.y')) %>% fnc()
  return(val)
}

# Mins and maxes for each of the four plots:
mins <- generate_boundary_indices(min)
maxs <- generate_boundary_indices(max)
col.lwr <- c("white", "green", "white", "green") # Thoughts: Low std: Good (green)
col.upr <- c("purple", "red", "purple", "red")   # Thoughts: High std: Bad (red)

# png("figures/2_e.png", width=1600, height=1200, units="px", res=300) # Saving

do.call(
  wrap_plots,
  lapply(1:4, function(i) {
    df_merged %>% {
      ggplot(.) +
        geom_tile(aes_string(x='x', y='y', fill=names(.)[i+2])) +
        scale_fill_gradient(low=col.lwr[i], high=col.upr[i],limits=c(mins[i], maxs[i])) +
        xlab('x [m]') + ylab('y [m]') + labs(cbar="test") +
        coord_fixed()
    }
  })
)
```

```
## Warning: `aes_string()` was deprecated in ggplot2 3.0.0.
## i Please use tidy evaluation ideoms with `aes()`
```



There is a clear distinction between the prior Poisson point process and the posterior, in that the gradient caused by the detection probabilities is very apparent. One can also see that there is significantly less variance in general for the posterior values, especially at the top right, where there is a high detection probability.