MT4537 - Spatial Statistics

Project 1 - Simulation and Model Fitting

Contents

Chapter One

Simulation

The* R *code associated with this project is provided in a GitHub repository which contains a README file that explains the layout of the relevant files and it can be accessed using the following link: https://github.com/alawrie751/MT4537Proj1

1.1 Description of the Simulation Process

This section provides a description of the process followed to create the simulation algorithm for a Thomas cluster process by deriving a random driving intensity and then using thinning to create a given realisation. For reference, the code for the simulation algorithm is contained in the *src/simul_funcs.R* file.

1.1.1 Usability and Reproducibility

The simulation algorithm is written using functions within *R* since this allows the simulations to be re-run multiple times easily and without having to repeat code. Making use of functions allowed inputs to be created that let the user control some elements of the simulations. These inputs included the mean of the distribution of the number of parents, the bandwidth used in creating the driving intensity, the dimension of the square in which the process would be simulated and a random seed.

Initially, there is a set of input checks which are just there to improve usability of the function. It ensures the user inputs are valid to reduce the chance of unexplained errors later in the process. The data type and limitations for each input is described above the function. The next step was to set the random seed. This allows the results of the simulation (ie. the realizations) to be reproduced if the same inputs are given to the function. If this was not included then the realizations would be different every time the function is run, even if all the other inputs remained the same.

1.1.2 Parent Points

Moving on to the distribution of the "parent" points, the theory of Neyman-Scott processes assumes that the parent points are generated from a homogeneous Poisson process. This means the number of these parent points can be given by a single random draw from a Poisson distribution. This is exactly what has been implemented in the simulation algorithm where the mean of this distribution is controlled by a user input which can be manipulated as required.

The next step is to place the required number of parent points randomly and independently across the region. Since these parents come from a homogeneous process they have equal probability of being placed anywhere within the region which implies that they can be placed using random draws from a uniform distribution. This is done in the algorithm by taking the given number of random draws from a uniform distribution with limits given by the length of the region to give the x-coordinates and another set of similar random draws giving the y-coordinates. These random draws were stored as vectors with equal length and converted to a *spatstat ppp* object for use in the next step.

A driving intensity was then derived using the randomly located parent points as the centres of Gaussian densities which were symmetric in every direction. These densities have a single parameter σ which is the standard deviation of the normal distribution being placed at each parent point. This can be controlled by the user and changed to fit the situation being simulated. This driving intensity was created using the *density* function from *spatstat* using Gaussian kernels and the *sigma* parameter as described above.

Now we will consider a short example of this part of the algorithm. The R code used for this example is contained in Section A.1 of the Appendix. First, set the seed to 123 and simulate a value from a Poisson distribution with mean 11. This returns a value of 9 parent points. The next step is to generate the (x, y)-coordinates for each of these 9 points using random draws from a uniform distribution in the range $[0, 1]$. This returns vectors for the x-coordinate and the y-coordinate, the values of which are contained in Table 1.1. These points can be plotted as seen on the left in Figure 1.1. The final step in this section of the algorithm is to devise the driving intensity from these parent points by placing symmetric Gaussian densities at these parent points with a standard deviation of 0.1. This intensity can be seen on the right side of Figure 1.1.

x	0.409	0.883	0.940	0.046	0.528	0.892	0.551	0.457	0.957
y	0.453	0.678	0.573	0.103	0.900	0.246	0.042	0.328	0.955

Table 1.1: x- and y-coordinate values used in the example in Section 1.1.2.

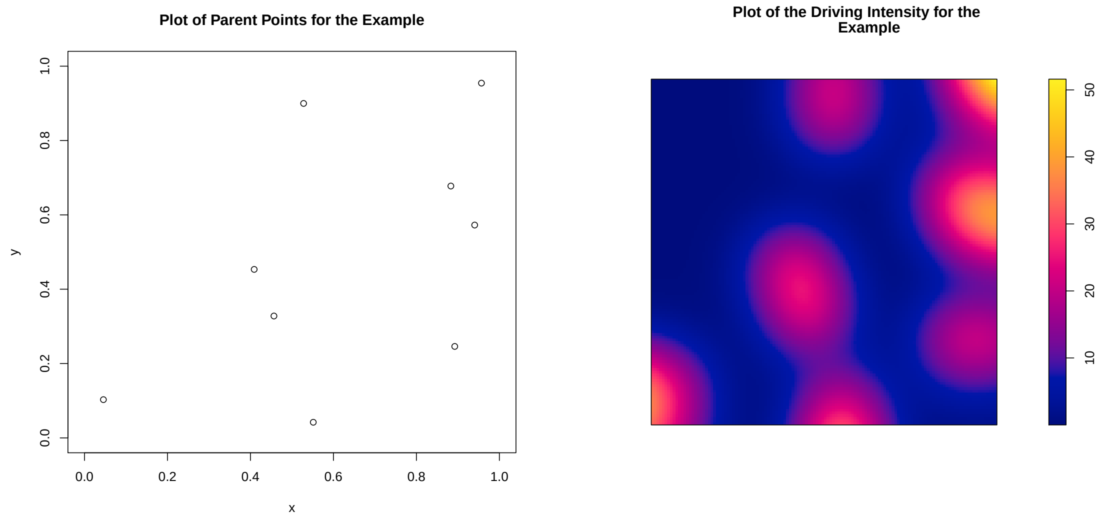

Figure 1.1: This contains the plots for the example in Section 1.1.2. The left hand plot is the parent points and the right hand plot is the resulting driving intensity using Gaussian densities and a standard deviation of 0.1. This was completed in a unit square.

1.1.3 Thinning

The thinning section of the algorithm took part in a separate function to avoid the main simulation function becoming too crowded. This function took as inputs the driving intensity based on the parent points derived in the previous step and the length of the sides of the square being simulated in.

This thinning process is completed by first generating a sample from a homogeneous Poisson process using the *rpoispp()* function from the *spatstat* library this time. The intensity for this homogeneous Poisson process is given by the maximum intensity value observed in the derived driving intensity that came from the parent points and is generated in the same window as specified earlier.

The thinning step is then applied to this realisation of the homogeneous Poisson process. Two empty vectors are created to store the (x, y)-coordinates that are to kept as points in the final simulation. The (x, y)-coordinates are extracted from the *ppp* object created during the simulation from the homogeneous Poisson. All the values in the previously derived driving intensity are transformed to probabilities by dividing each by the maximum value of the intensity observed so they are all now in the range $[0, 1]$. Now, for each set of (x, y)-coordinates, a Bernoulli random variable is simulated. This has the following PDF:

$$f(x) = \begin{cases} 1 & \text{with probability } p \\ 0 & \text{with probability } 1 - p \end{cases}$$

The probability p of returning a value of 1 is given by the value of the transformed density evaluated at the point under consideration. The point was kept and added to the initially empty storage vectors if a value of 1 is returned by the Bernoulli trial and is removed (not added to the storage vectors) from the simulation if 0 is returned.

The (x, y)-coordinates that are kept after the thinning process described above are then converted to a *ppp* object then returned. The overall simulation function then returns a list containing these simulation points as a *ppp* object and the driving intensity derived in Section 1.1.2.

Continuing the example from the previous section, the full simulation from a homogeneous Poisson process is created. This is shown on the left of Figure 1.2. The thinning process is then applied making use of the driving intensity depicted on the right of Figure 1.1. The plot on the right of Figure 1.2 shows all the points again but those points in red are the points that are kept as part of the simulation. All the other points are removed from the simulation.

1.2 Realisations from the Simulation

This section contains two realisations from the simulation algorithm described in the previous section. Figure 1.3 shows the plots from the first simulation where the mean number of parent points was 30, the standard deviation of the Gaussian densities placed at the parent points was 0.1, a random seed of 200 and the simulation was completed in the unit square. The plots for the second simulation are contained in Figure 1.4. This simulation was completed using a mean number of parents of 45, standard deviation of the Gaussian densities equal to 0.05 and a random seed of 465. This was again simulated within the unit square.

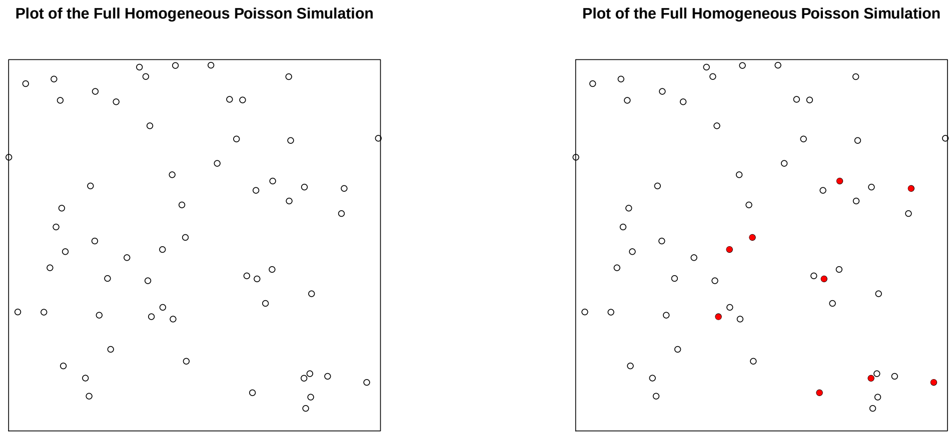

Figure 1.2: This contains the plots for the example in Section 1.1.3. The left hand plot is the simulation from the homogeneous Poisson process and the right hand plot is the thinned simulation where the red points were those that were kept as points in the simulation (the rest were removed). This was completed in a unit square.

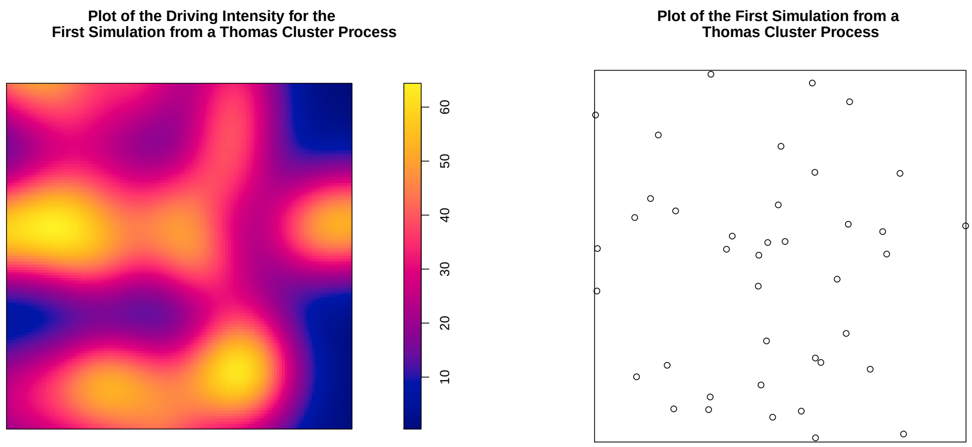

Figure 1.3: This contains the plots for the first realisation from the simulation of a Thomas cluster process. The left hand plot is the driving intensity and the right hand plot is the resulting set of spatially distributed points. This was simulated in a unit square.

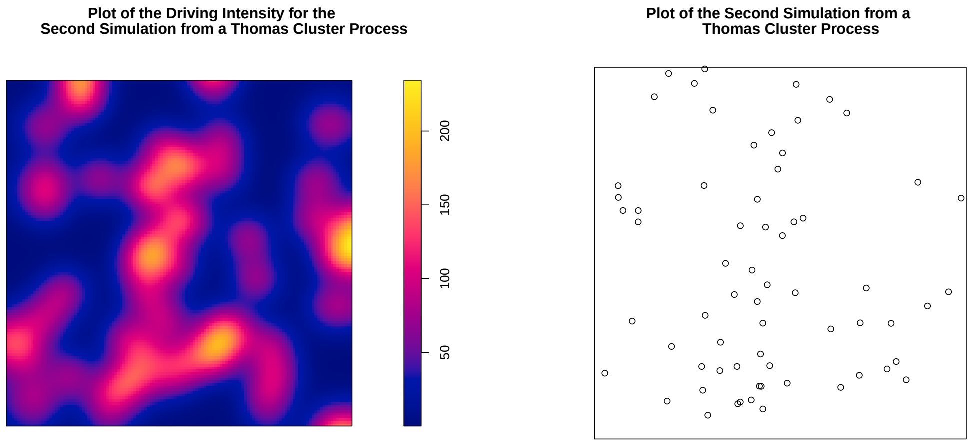

Figure 1.4: This contains the plots for the second realisation from the simulation of a Thomas cluster process. The left hand plot is the driving intensity and the right hand plot is the resulting set of spatially distributed points. This was simulated in a unit square.

Chapter Two

Model Fitting

2.1 Model Output and Comparison

Appendix A

R Code

A.1 Algorithm Example

```
# Code used for the example in Section 1.1.2 and 1.1.3

library(spatstat)

set.seed(123)

num_p <- rpois(1, 11)

x_p <- runif(num_p, 0, 1)
y_p <- runif(num_p, 0, 1)

plot(x_p, y_p, xlim = c(0, 1), ylim = c(0, 1), xlab = "x", ylab = "y",
     main = "Plot of Parent Points for the Example")

loc_par <- ppp(x_p, y_p, window = owin(c(0, 1), c(0, 1)))

drive_intense <- density(loc_par, sigma = 0.1, kernel = "gaussian")

plot(drive_intense, main = "Plot of the Driving Intensity for the
     Example")
```

```
full_sim <- rpoispp(max(drive_intense), win = owin(c(0, 1), c(0, 1)))

plot(full_sim, main = "Plot of the Full Homogeneous Poisson Simulation")

new_x <- NULL
new_y <- NULL

xvals <- full_sim$x
yvals <- full_sim$y

nvals <- length(xvals)

prob_dens <- as.function(drive_intense / max(drive_intense))

for (i in 1:nvals) {

  prob <- prob_dens(xvals[i], yvals[i])
  ind <- rbinom(1, 1, prob)

  if (ind == 1) {

    new_x <- c(new_x, xvals[i])
    new_y <- c(new_y, yvals[i])

  }
}

thinned <- ppp(new_x, new_y, window = owin(c(0, 1), c(0, 1)))

plot(full_sim, main = "Plot of the Full Homogeneous Poisson Simulation")
points(thinned, pch = 16, col = "red")
```

A.2 Simulation Functions

```r
# Description ----

# Script containing the functions to carry out the simulations for the first
# part of the project

# Load Packages ----

library(spatstat)

# Thinning Function ----

# Function to thin a simulation from a homogeneous Poisson process to an
# inhomogeneous Poisson process simulation

# Inputs:
#   intense - intensity given by the density function from spatstat
#   full_sim - ppp object which is a simulation from a homogeneous Poisson
#              with intensity equal to the maximum intensity seen across the
#              window
#   xylim - limit of the window in both the x and y directions (positive numbe

# Outputs:
#   ppp object containing the thinned simulation from a Thomas process

thin_sim <- function(intense, full_sim, xylim) {

  # Create variables to store a vector containing the points being kept
  new_x <- NULL
  new_y <- NULL

  # Extract the x and y values into vectors and find the length of the vectors
  xvals <- full_sim$x
  yvals <- full_sim$y

  nvals <- length(xvals)
```

```
  # Convert the derived intensity function to the range [0, 1] and then conver
  # it to a function type so that the value of the density at (x, y) is
  # extracted easily
  prob_dens <- as.function(intense / max(intense))

  for (i in 1:nvals) {

    # Find probability density at the given point then sample a Bernoulli
    # variable with probability of 1 equal to this probability
    prob <- prob_dens(xvals[i], yvals[i])
    ind <- rbinom(1, 1, prob)

    # If the indicator is 1 then keep the given point and add it to the vector
    if (ind == 1) {

      new_x <- c(new_x, xvals[i])
      new_y <- c(new_y, yvals[i])

    }

  }

  # Convert to a spatstat ppp object and return
  thinned <- ppp(new_x, new_y, window = owin(c(0, xylim), c(0, xylim)))

  return(thinned)

}


# Thomas Simulation Function ------

# Function to run the simulation of a Thomas cluster process using the
# thinning function above

# Inputs:
```

```
#   mu_p - mean of parent distribution (positive number)
#   sd_c - standard deviation of symmetric normal distribution of children
#          around parents (positive number)
#   xylim - limit of the window in both the x and y directions (positive numbe
#           has a default value of 1)
#   rand_seed - number to set the random seed to to allow reproducibility of
#               simulations (number, has an arbitrary default of 150)

# Outputs:
#   ppp object containing the simulation from a Thomas process

ThomasSimul <- function(mu_p, sd_c, xylim = 1, rand_seed = 150) {

  # Error traps to ensure that user inputs are valid
  if (!is.numeric(mu_p)) stop("invalid arguments. mu_p must be a numeric")
  if (!is.numeric(sd_c)) stop("invalid arguments. sd_c must be a numeric")
  if (!is.numeric(xylim)) stop("invalid arguments. xylim must be a numeric")
  if (!is.numeric(rand_seed)) stop("invalid arguments. rand_seed must be a nur

  if (mu_p <= 0) stop("invalid arguments. mu_p must be > 0")
  if (sd_c <= 0) stop("invalid arguments. sd_c must be > 0")
  if (xylim <= 0) stop("invalid arguments. xylim must be > 0")

  # Set random seed to allow reproducibility of simulations
  set.seed(rand_seed)

  # Randomly select number of parents from Poisson distribution of mean mu_p
  num_p <- rpois(1, mu_p)

  # Randomly generate coordinates for each of the num_p points, with equal
  # probability everywhere in the region of [0, xylim] in each direction
  x_p <- runif(num_p, 0, xylim)
  y_p <- runif(num_p, 0, xylim)
  #plot(x_p, y_p, xlim = c(0, xylim), ylim = c(0, xylim))

  # Convert to a spatstat ppp object
```

```
    loc_p <- ppp(x_p, y_p, window = owin(c(0, xylim), c(0, xylim)))

    # Calculate the driving intensity using Gaussian densities placed at each
    # parent location with standard deviation given by sd_c
    drive_intense <- density(loc_p, sigma = sd_c, kernel = "gaussian")

    # Create sample of a homogeneous Poisson process with density equal to the
    # highest seen in the derived driving intensity above
    large_sim <- rpoispp(max(drive_intense), win = owin(c(0, xylim), c(0, xylim)

    # Thin this using the thinning function described above
    thinned_sim <- thin_sim(drive_intense, large_sim, xylim)

    return(list(thin_sim = thinned_sim, intense = drive_intense))

}
```