MT4537 - Spatial Statistics

Project 1 - Simulation and Model Fitting

Contents

Chapter One

Simulation

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 commented code for the simulation algorithm is contained in Section A.2 of the Appendix.

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 without having to repeat the 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 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 (i.e. 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 derive 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, y)-coordinate values used in the example in Section 1.1.2. These are rounded to 3 decimal places.

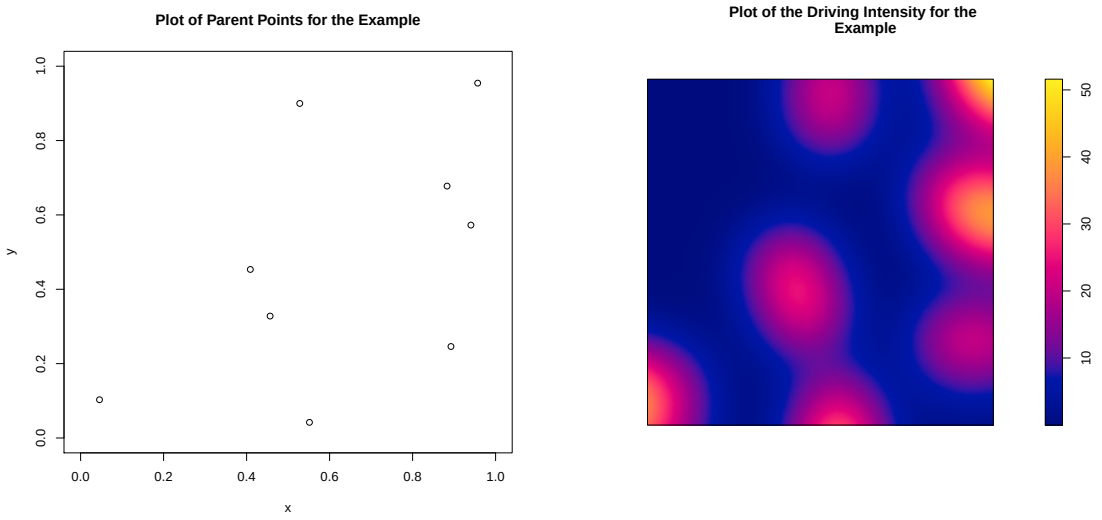

Figure 1.1: This contains the plots for the example in Section 1.1.2. The left hand plot is the parent points and the right hand plot is the resulting driving intensity using Gaussian densities and a standard deviation of 0.1. This was completed in a unit square.

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. As its inputs this function took 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* package 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 be 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 and 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 in red are those that are kept as part of the simulation. All the other points are removed from the simulation.

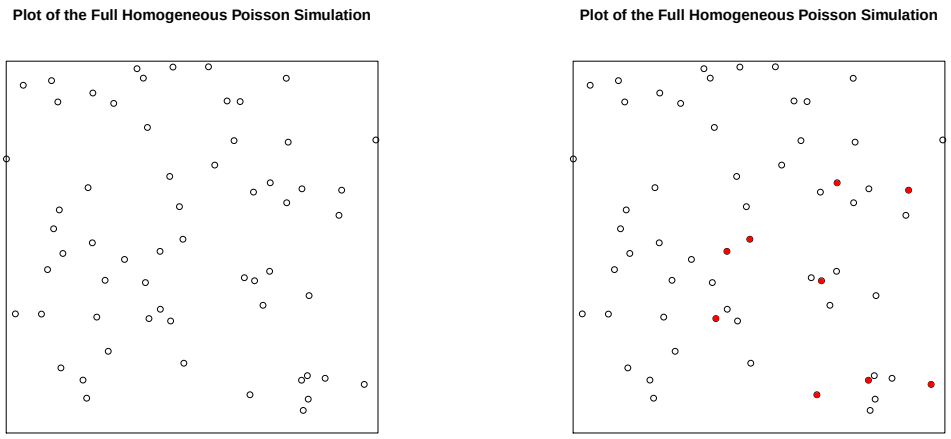

Figure 1.2: This contains the plots for the example in Section 1.1.3. The left hand plot is the simulation from the homogeneous Poisson process and the right hand plot is the thinned simulation where the red points were those that were kept as points in the simulation (the rest were removed). This was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

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. The code to create these realisations of the simulation algorithm is contained in Section A.3 of the Appendix.

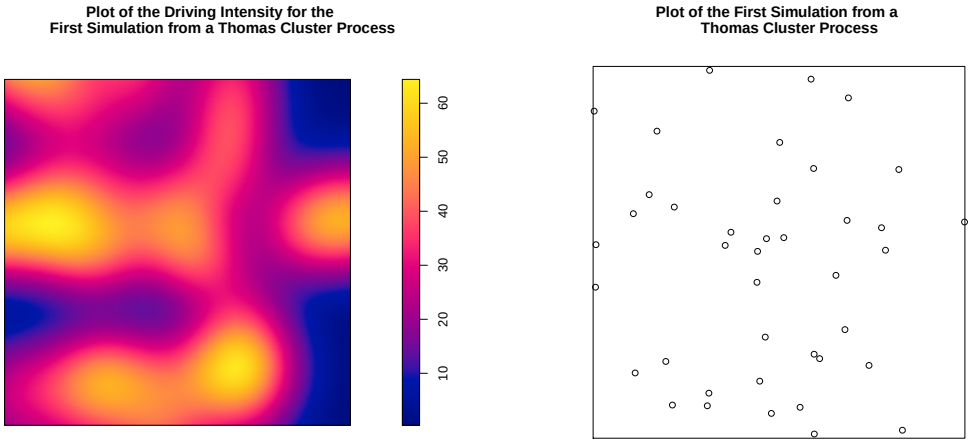

Figure 1.3: This contains the plots for the first realisation from the simulation of a Thomas cluster process. The left hand plot is the driving intensity and the right hand plot is the resulting set of spatially distributed points. This was simulated in a unit square.

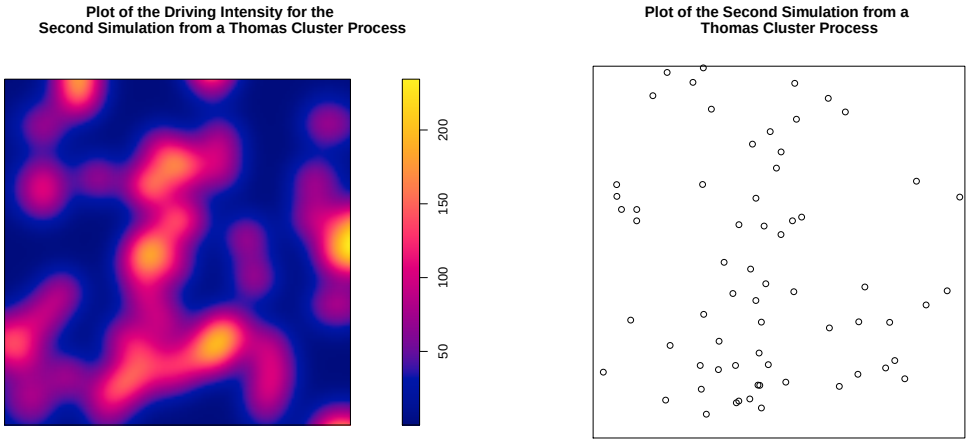

Figure 1.4: This contains the plots for the second realisation from the simulation of a Thomas cluster process. The left hand plot is the driving intensity and the right hand plot is the resulting set of spatially distributed points. This was simulated in a unit square.

1.3 Fitting Thomas and Matérn Process Models

1.3.1 Minimum Contrast Method

This is a common method used for estimating parameters for Poisson cluster processes which is based on the idea of least-squares estimation. It makes use of an estimated and theoretical summary characteristic and tries to find the parameter values which minimise the difference between these. The method can be seen as trying to minimise

$$\Delta(\boldsymbol{\theta}) = \int_{s_1}^{s_2} |\hat{S}(r) - S_{\boldsymbol{\theta}}(r)|^{\beta} dr$$

with respect to $\boldsymbol{\theta}$ where this is a vector of the parameters. The vector of parameters that minimise this expression are the estimators of the parameters. $S(r)$ is a summary characteristic depending on the distance from a point r. Generally, the other values included in the expression are arbitrary. However, it is normal practice to select $\beta = 2$. The limits of the integration expression remain arbitrary with no standard selection for these.

In reality, the formula given above for the minimum contrast method is discretised (broken up into discrete points) to make a summation formula. This means the method

now tries to minimise using

$$\int_{s_1}^{s_2} |\hat{S}(r) - S_{\boldsymbol{\theta}}(r)|^{\beta} dr \approx \sum_{i=0}^{k} |\hat{S}(\gamma_i) - S_{\boldsymbol{\theta}}(\gamma_i)|^{\beta}$$

where $S(r)$ and β remained defined as above. Meanwhile, $\gamma_0 = s_1$, $\gamma_k = s_2$ and $\gamma_i = \gamma_0 + i\delta$ where $\delta = \frac{s_2 - s_1}{k}$ and k is a positive integer. This is simply breaking the range of values for r into $k+1$ discrete points to evaluate the summary characteristic function $S(r)$ at to give the total difference. Taking the limit as k tends to ∞ of the sum will lead back to the initial integration definition for the method.

The summary characteristics commonly used are either

$$S(r) = K(r) \quad \text{or} \quad S(r) = g(r)$$

where $K(r)$ is Ripley's K-function and $g(r)$ is the pair correlation function. Throughout this section the minimum contrast method will be used with $K(r)$. $K(r)$ is defined to be the mean number of distinct points found within a distance r from any given point (not including the point itself). Whereas, $g(r)$ is defined in terms of $K(r)$ and contains the same information as it. In fact, it is a summary characteristic defined to be proportional to the derivative of $K(r)$. This can be written out as

$$g(r) = \frac{K'(r)}{2\pi r} \quad \text{for } r \geq 0.$$

1.3.2 Clip to Nominal Window

Both realisations of the simulation algorithm from Section 1.2 were clipped to a nominal window based on a simple rule. Each were clipped by double their *sigma* value from each edge. This meant the first simulation went from being in the whole unit square to the range $[0.2, 0.8]$ on both axes while the second simulation was moved into the range $[0.1, 0.9]$ on both axes. Plots of these new versions of the realisations are given in Figure 1.5.

1.3.3 Model Parameters and Pair Correlation Functions

Table 1.2 provides the values of the parameters for the two types of cluster models fitted to the realisations of the simulation algorithm. Both models had the same uniform density of 18 for the first realisation and 49 for the second as would be expected.

Figure 1.6 contains all the pair correlation functions for both realisations and both types of model. A translation edge correction was used when creating these plots. The fact

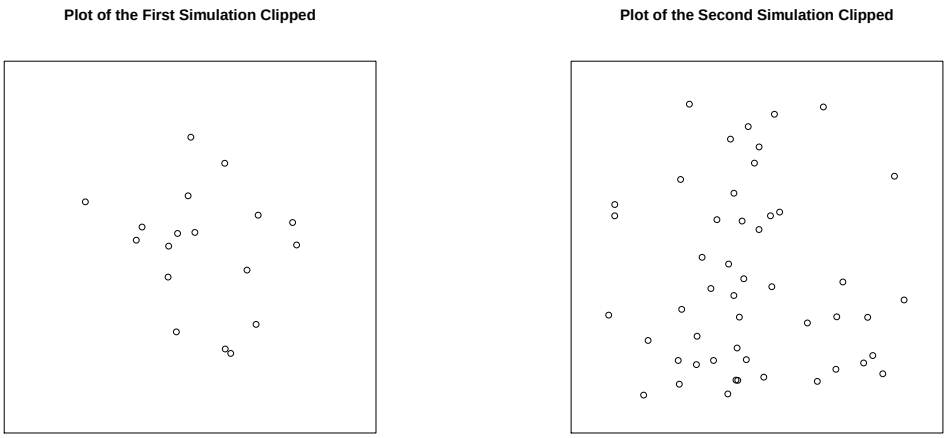

Figure 1.5: Plots of the two simulations from a Thomas process clipped to nominal windows. The left plot is the first simulation clipped to $[0.2, 0.8]$ on both axes while the right plot is the second clipped to $[0.1, 0.9]$ on both axes.

		Thomas	Matérn
Realisation 1	kappa	1.614	0.145
	scale	0.190	1.297
Realisation 2	kappa	18.619	18.614
	scale	$7.009\text{x}10^{-5}$	$1.319\text{x}10^{-4}$

Table 1.2: This contains the model parameters from the Thomas and Matérn cluster models fitted to the two realisations of the simulation algorithm.

that the black line in the two plots of the first column remains mostly above the dashed red line indicates that this is clustered data. This is what we would hope to see given that these are realisations of a simulation algorithm for clustered data. Neither model appears to do particularly well at recreating this clustering seen in the real data. The wildly different scales on the vertical axes in the second row make any inference difficult here. However, in the first row the Thomas model appears to recreate the clustering better then the Matérn model. Again this is as expected given these are simulations from a Thomas cluster process. Therefore I would conclude that the Thomas model is slightly better for modelling this simulated point process data.

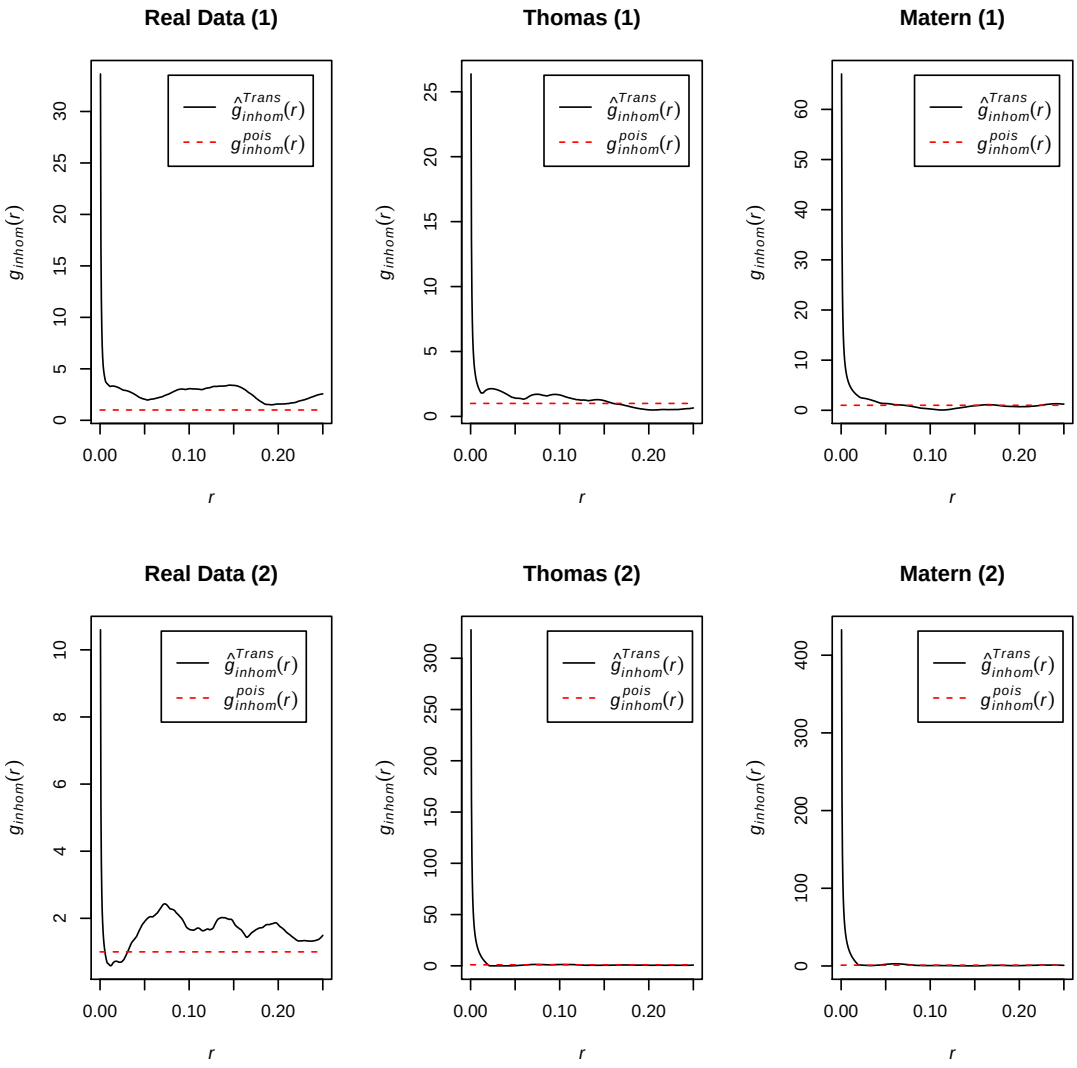

Figure 1.6: These are the pair correlation functions for the simulated data and both types of models. The first row contains all the plots relating to the first realisation and the second row contains all those relating to the second realisation.

Chapter Two

Model Fitting

The *bei* data set contains information on the location of 3604 trees and *bei.extra* provides additional covariates showing the elevation across the region and also the gradient. A plot of the *bei* data is provided on the left of Figure 2.1. Here we are trying to fit a model to describe the location of the trees across the region. The code used to run these models is contained in Section A.4.

2.1 Kernel Smooths

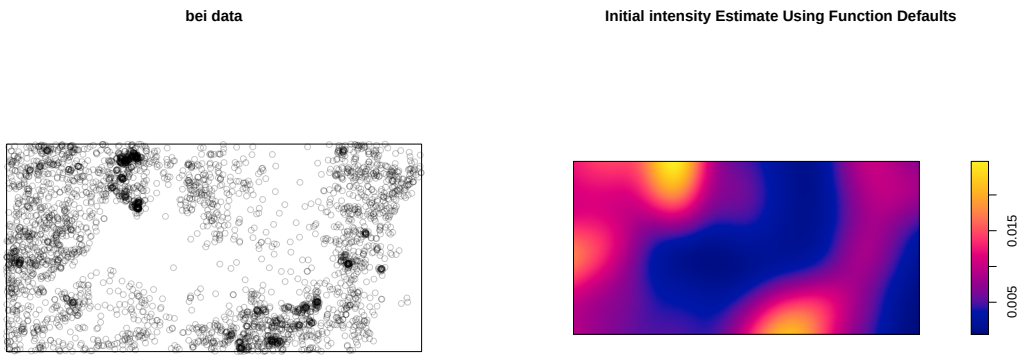

Figure 2.1: The plot on the left is the *bei* data set. On the right is an initial estimate of the intensity for this data using the *density* function with the arguments left at the default values.

The first step when trying to model this data is to fit a kernel smooth. This is fitted using the *density.ppp* function provided in *spatstat*. This function is simply placing symmetric kernels at every point in the pattern and then summing all the values of these kernels at every point. The shape and spread of the kernel can be controlled by specifying different types of kernels and specifying the bandwidth (*sigma*).

The initial smooth, depicted on the right of Figure 2.1, is fitted using the function defaults. It is difficult to tell with this many points which kernel is being used. Since it appears to be relatively smooth, it is unlikely to be the disc kernel but it is not easy to decide between the other three options (these being Gaussian, quartic and Epanechnikov). Looking at the help files for the function, the default kernel used is in fact the Gaussian. Again, just looking at the plot it is difficult to make any kind of guess as to the value being used for the bandwidth. However, the help files says that there is a "simple rule of thumb that depends only on the size of the window" to calculate a value for *sigma* when none is specified by the user. In this case, it appears to have a value of 50 and a window size of 1000m by 500m (this was concluded using trial and error). This implies the formula being used is either $0.05a$ or $0.1b$ where a is the length of the longest side while b is the length of the shorter side but this is only based on this one example so is not necessarily conclusive.

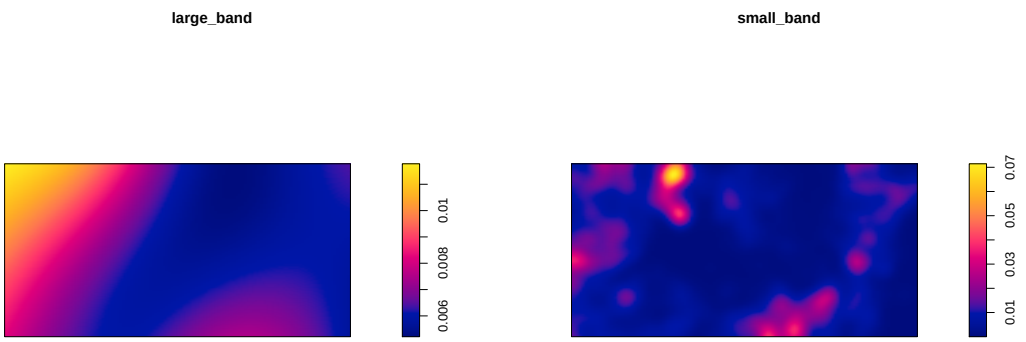

Figure 2.2: Initial estimate of the intensity for the *bei* data using the *density* function with Gaussian kernels but σ equal to 200 on the left and 20 on the right.

Figure 2.2 provides two markedly different intensity estimates. Also, these both differ from the initial estimate using the function defaults. The only change in these intensities

is the use of different values for the bandwidth. It is clear to see that the higher bandwidth leads to a much flatter intensity while the smaller value leads to a much more bumpy estimate of the intensity.

2.2 Fitting Poisson Process Models

This section covers the model fitting process used when adding covariates into the models through the *ppm* function in *spatstat*. The models being fitted here are nearly equivalent to a Poisson GLM. They use a log-link since the intensity being estimated must be strictly positive.

2.2.1 AIC

The model selection criteria used was Akaike's Information Criterion (AIC) which is calculated as follows:

$$AIC = -2log[L(\hat{\theta})] + 2q$$

where $L(\hat{\theta})$ is the likelihood function evaluated at its maximum which are the outputs of the modelling functions. q is defined to be the number of parameters estimated in the model and acts as a penalty for estimating more parameters. Model selection using AIC is carried out by selecting the model which has the lowest AIC value.

2.2.2 Using Coordinates

The first model fitted was just a constant model using the formula "$bei \sim 1$". Looking at the intensities estimated in Section 2.1 it is quite clear that this will not be a good model which should be evident in the AIC values. This model was included for completeness. The next model fitted was one that depended on x and y individually using the formula "$bei \sim x + y$" which was followed by the model based on the interaction term using the formula "$bei \sim x * y$". The final three models fitted were all polynomials of degree 2, 3 and 6 respectively. These polynomial models include coefficients for all the powers of the coordinates up to and include the degree given. For example, the form of the polynomial model of degree 3 is

$$log(\mathbb{E}(\lambda(x, y))) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 y + \beta_5 y^2 + \beta_6 y^3$$

where the β_is are the coefficients to be estimated in the model.

The AIC values for all these models are contained in Table 2.1. From Section 2.2.1, it is known that selection by AIC is done by choosing the model with the lowest value for the AIC. Looking at the table it can be seen that the models improve on each other as you move down the rows. Therefore, the polynomial model of degree 6 is the best of the models considered and so it should be brought forward in the modelling process. The intensity function for this model is provided in Figure 2.3. This intensity does appear to pick up the rough location of the dense locations but it does not appear as dense as the kernel smooth estimated previously.

Model Formula	AIC
1	42763.92
$x + y$	42545.68
$x * y$	42455.77
poly(x, 2) + poly(y, 2)	42234.23
poly(x, 3) + poly(y, 3)	42157.51
poly(x, 6) + poly(y, 6)	41966.5

Table 2.1: AIC values for the models using the (x, y)-coordinates.

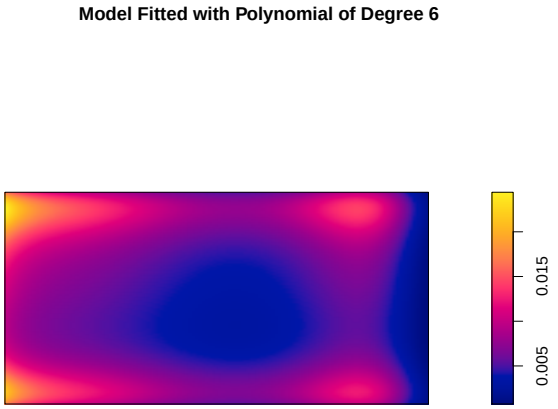

Figure 2.3: Intensity plot for the best model for the *bei* data set using only variants of the (x, y)-coordinates as covariates. This model was a polynomial model of degree 6.

2.2.3 Using Elevation and Gradient

In this section we look at fitting models using the extra available covariates in the data set. These include the elevation which is defined to be the distance above sea level in metres. Meanwhile, a measure of the gradient is present at each point in the region. This is defined as the norm of the elevation gradient so is in effect the magnitude of the change in the elevation.

A number of different models were fitted using these covariates. Throughout this section and the rest of this report, e represents the elevation and g represents the gradient. The modelling process began with a model that simply depended on the elevation which was fitted using the formula "$bei \sim e$". The next model was one that was based on only the gradient this time. This used the formula "$bei \sim g$". Following on, a model depending on a linear combination of these covariates was considered using the formula "$bei \sim e + g$" which led onto the model including the interaction of these covariates given by "$bei \sim e * g$".

Model Formula	AIC
e	42760.49
g	42382.81
$e + g$	42295.11
$e * g$	42229.24

Table 2.2: AIC values for the models using the elevation (e) and gradient (g).

The AIC values calculated for this set of models are given in Table 2.2. Again, the model with the lowest AIC value is said to be the best model for the data according to this selection criteria. This means that the model including the interaction term is the preferred one for this data. The fitted intensity function for this model is given on the right of Figure 2.4. This does not look very hopeful as there are not many similarities to the intensity function estimated using the kernels.

Parameter	Coefficient
Intercept	-4.403
e	-0.007
g	-36.500
$e : g$	0.293

Table 2.3: Parameter coefficients for the model including an interaction term for the elevation (e) and gradient (g).

The parameter estimates for the model with the interaction term are contained in

Table 2.3. These are all based on the log of the expected value of the trees at each point depending on the elevation and gradient at these points. This means it is difficult to interpret their exact meaning. However, we can look at the signs of these estimates. The intercept term is just the base line value of the intensity when both elevation and gradient are zero. Looking at the signs of the other cofficients shows that the intensity value decreases for both higher elevation and gradient since these are both negative. On the other hand, the positive value for the interaction term implies that the value of the intensity increases as the product of the elevation and gradient values increase.

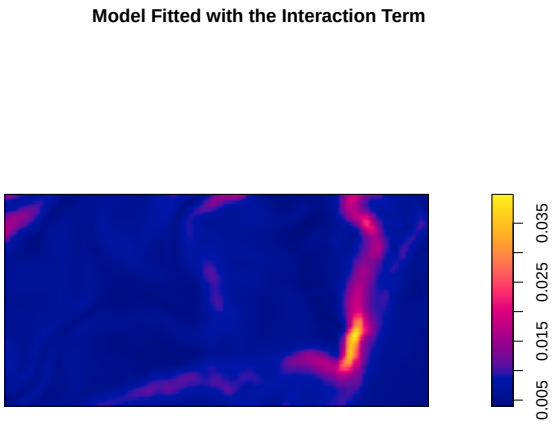

Figure 2.4: Intensity plot for the best model for the *bei* data set using only variants of the elevation and gradient covariates. This model was one including an interaction term.

Figure 2.5 provides the pair correlation functions ($g(r)$) both based on the real data and simulated data from this best model selected by AIC. This left hand plot indicates that in the real data there are more points seen closer to other points than would be expected under complete spatial randomness that only depended on the location or covariates. This is part of the assumptions of the models being fitted so far as indicated by the plot on the right which stays in and around the red line which shows the constant line at $g(r) = 1$ that would be expected under complete spatial randomness. This means the models being fitted are unlikely to be a very good fit since there appears to be some clustering processes underlying the data. Also, note the different scales on the vertical

axes.

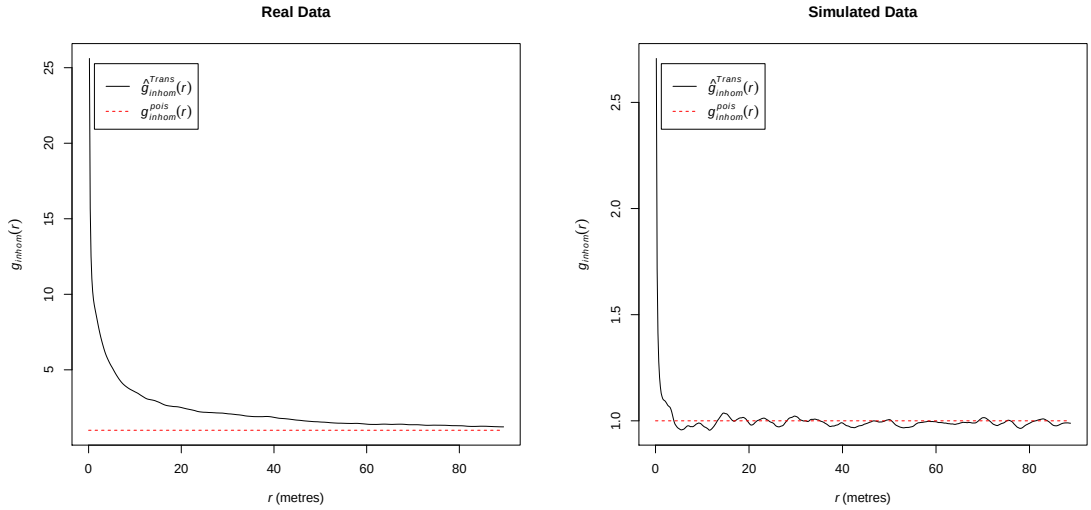

Figure 2.5: These are pair correlation functions for the *bei* data. The plot on the left is based on the original data while the right uses data simulated from the model including the interaction term.

2.3 Fitting Cox Models

2.3.1 Fitting the Model

Based on the conclusions from pair correlation function in the previous subsection, it may be useful to look at fitting Cox models. This is because the pair correlation plots appeared to indicate some clustering in the data which wasn't (and shouldn't be expected to due to the design of the models) being modelled by the Poisson process models. Cox models are designed to model the clustering processes underlying the intensity. Here we will consider only Thomas process models (assuming a Gaussian scattering at each parent point).

The preferred model from Section 2.2 (selected by AIC) was the polynomial of degree 6 for both of the (x, y)-coordinates so this model structure will be carried forward in this section. Figure 2.6 shows the pair correlation functions from the real data and simulated data from this preferred model. It is clear that the same issues arise as for the best elevation and gradient model considered in the previous section. These issues are the appearance of unmodelled clustering in the real data which means it is reasonable to look at fitting a clustering (Cox) model.

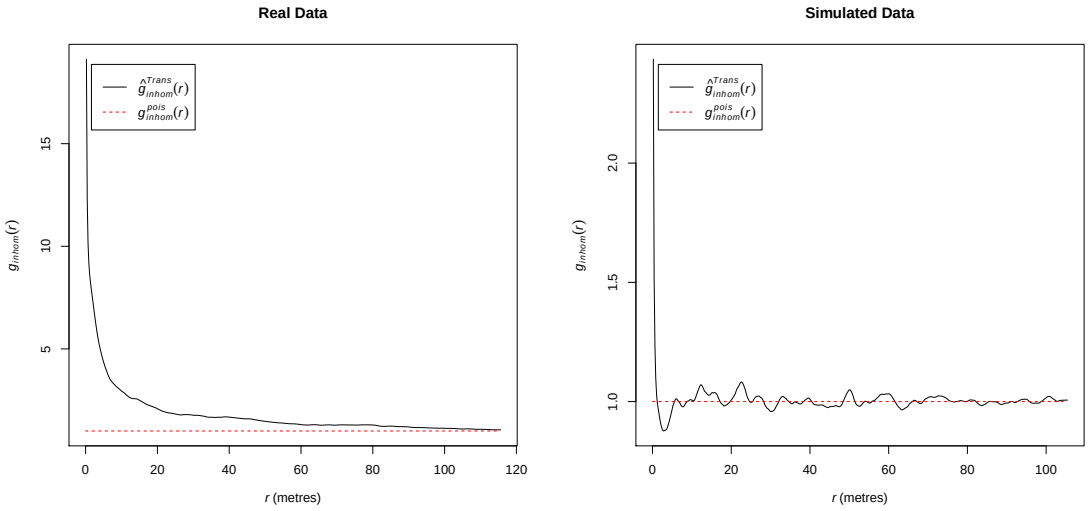

Figure 2.6: These are pair correlation functions for the *bei* data. The plot on the left is based on the original data while the right uses data simulated from the model including the interaction term.

Table 2.4 gives the coefficients for all the x and y parameters alongside the intercept and the fitted cluster parameters. This model was fitted using the method "*palm*" in the *kppm* function. The parameter coefficients for all the polynomial coordinate terms are exactly the same as those estimated in the standard Poisson process model discussed in Section 2.2.2. The same applies to the intercept term. This implies that the underlying intensity estimated is the same as in the previous section. The difference in the models is the cluster process being fitted. The *kappa* value gives the mean of the underlying parent Poisson process while the *scale* parameter gives the standard deviation of the normal densities placed at the parent points.

The intensity for this model, which is plotted in Figure 2.7, is the exact same as that which was estimated in Section 2.2.2. This is due to the fact that all the coefficients of the polynomial terms remained the same which forces the underlying intensity to remain the same. The plot on the left of the figure depicts the original data placed on top of the intensity which shows that this intensity appears to be quite a good fit for the data. The pair correlation functions are contained in Figure 2.8. Notice the different scales on the vertical axes. From the plots it again appears that the data are more clustered than the simulated data meaning the cluster model is not representing the data very well. This is because the plot on the right is in and around the value of 1 which is what is expected from completely spatially random point processes for most values of r (the radius of the

circle). This is particularly clear from 20m onwards.

Parameter	Coefficient	Parameter	Coefficient
Intercept	-5.013	-	-
x	-35.100	y	12.700
x^2	12.549	y^2	39.015
x^3	-25.274	y^3	-10.112
x^4	-30.247	y^4	-12.134
x^5	-18.423	y^5	1.177
x^6	-1.531	y^6	-9.294
kappa	0.007	scale	8.659

Table 2.4: Parameter coefficients for the Thomas cluster model using the polynomials of degree 6 for both of the (x, y)-coordinates.

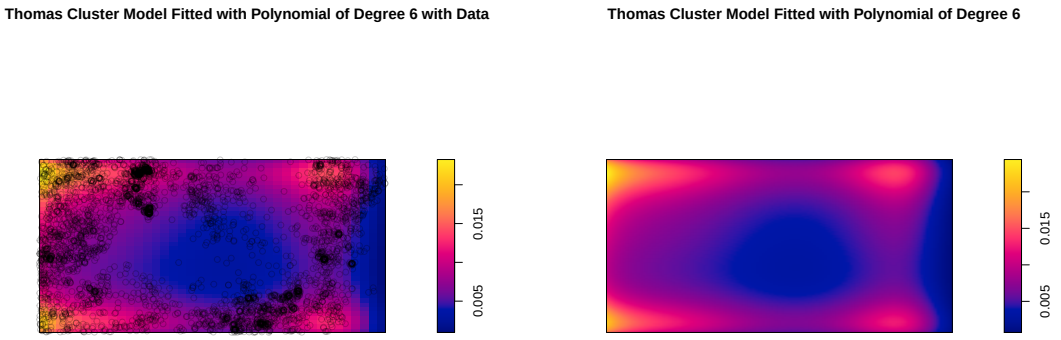

Figure 2.7: These are plots of the intensity function fitted to the *bei* data using polynomials of degree 6 in both the x and y directions. These were fitted as a Thomas cluster model. The left plot shows the original data overlayed onto the intensity while the right just shows the intensity surface.

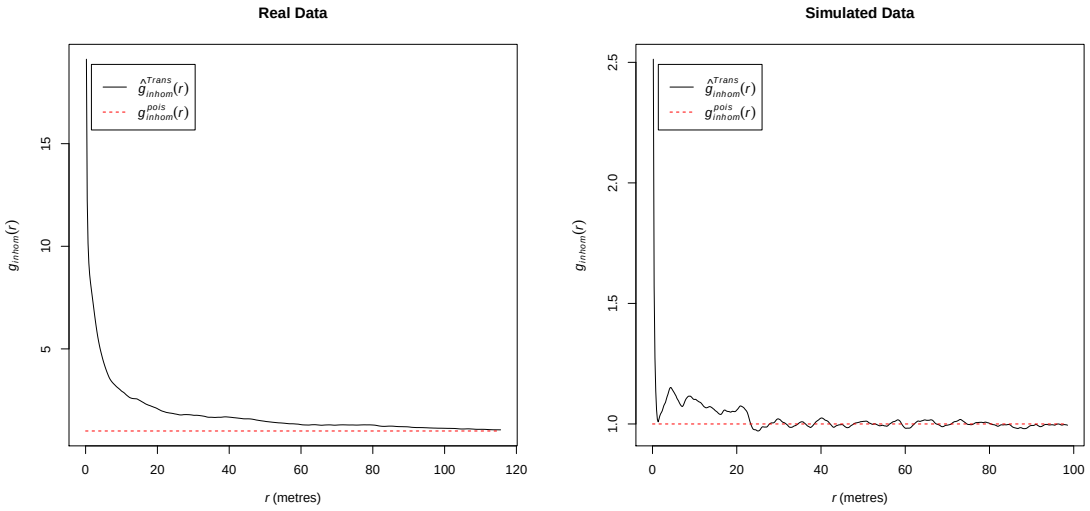

Figure 2.8: These are pair correlation functions for the *bei* data. The plot on the left is based on the original data while the right uses data simulated from the Thomas cluster model using the degree 6 polynomials in the coordinate directions.

2.3.2 Simulations from the Cox Model

Figure 2.9 below contains two different simulations from the Thomas model fitted above. These were made using a random seed of 500.

Generally, these simulations are a lot more dense than the original data appears to be. This is quite obvious in the central section of the simulations where there is a reasonably large number of points compared to Figure 2.11 where the central area is nearly empty. It does, however, seem to have picked out the areas of high density relatively well. Areas such as the top and bottom corners on the left hand side or the vertical belt towards the right are correctly more dense than the other areas.

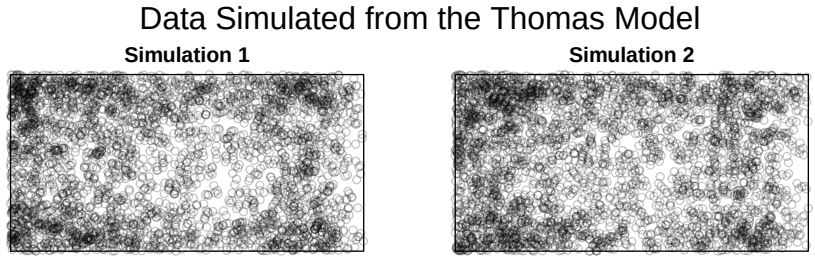

Figure 2.9: These are two simulations from the Thomas model fitted to the *bei* data in the previous section. These were created using a random seed of 500.

2.3.3 Selecting the Best Model

Figure 2.10 provides two different simulations from the best Poisson process model considered in Section 2.2. This is to try and allow some level of comparison between the models.

These simulations from the Poisson process model appear to be quite similar to those generated from the Thomas model. This is most likely because they are using identical driving intensities to generate the data. It is difficult to make any conclusions on which model is better for the data based on these simulated plots.

Overall, I would have to select the cluster model since the inhomogeneous summary characteristic functions implied quite heavily that there is some level of clustering in the underlying process generating the data. Therefore, it makes sense to choose a model which makes some attempt to model this clustering that is evident in the data set.

The next steps I would take in trying to model this data is by using some combinations of the (x, y)-coordinates and the elevation and gradient in a model together. I would also take a look at a number of different types of cluster models and check if any others perform better than the Thomas which was fitted in this section.

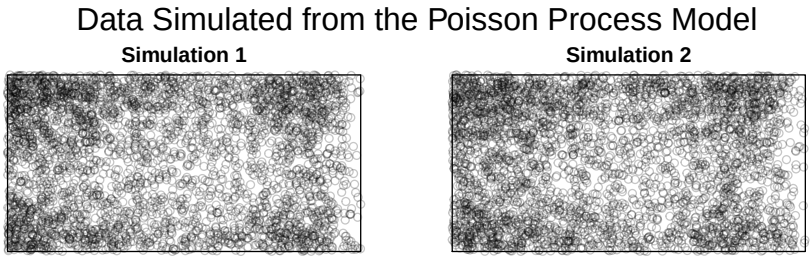

Figure 2.10: These are two simulations from the Poisson process model fitted to the *bei* data in Section 2.2. These were created using a random seed of 600.

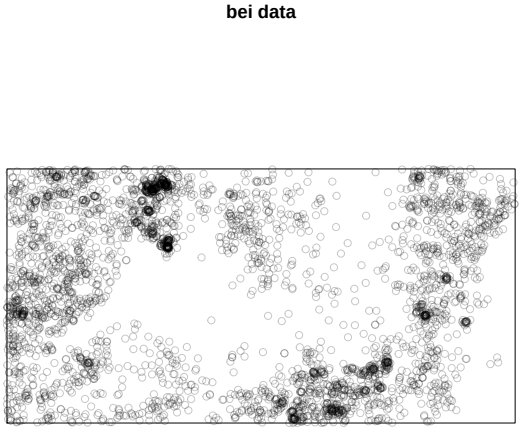

Figure 2.11: Plot of the original *bei* data set.

Appendix A

R Code

A.1 Simulation Algorithm Example

```
# Code used for the example in Section 1.1.2 and 1.1.3
# Commentary on this code is provided in those sections and
# on the actual functions used to carry out the simulation

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

```
# 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
#   xylim - limit of the window in both the x and y directions
#           (positive number)

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

thin_sim <- function(intense, xylim) {

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

  # 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)

  # Set size of window
  xylim <- 1

  # 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")

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

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

}
```

A.3 Run Simulations

```
# Description ------

# Script to run the simulations for the first part of the project,
# create the plots and fit the models as required

# Source the Functions ------

# Note that this also loads the spatstat library
source("src/sim_funcs.R")

# Create 2 Simulations, Clip to Window and Fit Models ------

sim1 <- ThomasSimul(30, 0.1, rand_seed = 200)
sim2 <- ThomasSimul(45, 0.05, rand_seed = 465)

# Save plots of these simulations and the driving intensity
plot(sim1$thin_sim, main = "Plot of the First Simulation from a
     Thomas Cluster Process")

plot(sim1$intense, main = "Plot of the Driving Intensity for the
     First Simulation from a Thomas Cluster Process")

plot(sim2$thin_sim, main = "Plot of the Second Simulation from a
     Thomas Cluster Process")

plot(sim2$intense, main = "Plot of the Driving Intensity for the
     Second Simulation from a Thomas Cluster Process")

# Clip to Nominal Window ------

# For simulation 1 clip to [0.2, 0.8] on each axis
```

```
# Get vector of indicies to remove
indremove <- NULL

for (i in 1:length(sim1$thin_sim$x)) {

  xi <- sim1$thin_sim$x[i]
  yi <- sim1$thin_sim$y[i]

  if (xi < 0.2 | xi > 0.8) {

    indremove <- c(indremove, i)

  } else if (yi < 0.2 | yi > 0.8) {

    indremove <- c(indremove, i)

  }

}

# Remove from ppp object
sim1$thin_sim$x <- sim1$thin_sim$x[-indremove]
sim1$thin_sim$y <- sim1$thin_sim$y[-indremove]

plot(sim1$thin_sim, main = "Plot of the First Simulation Clipped")

# For simulation 2 clip to [0.10, 0.90] on each axis

# Get vector of indicies to remove
indremove2 <- NULL

for (i in 1:length(sim2$thin_sim$x)) {

  xi <- sim2$thin_sim$x[i]
  yi <- sim2$thin_sim$y[i]
```

```
  if (xi < 0.1 | xi > 0.9) {

    indremove2 <- c(indremove2, i)

  } else if (yi < 0.1 | yi > 0.9) {

    indremove2 <- c(indremove2, i)

  }

}

# Remove from ppp object
sim2$thin_sim$x <- sim2$thin_sim$x[-indremove2]
sim2$thin_sim$y <- sim2$thin_sim$y[-indremove2]

plot(sim2$thin_sim, main = "Plot of the Second Simulation Clipped")

# Fit models ------

thom_sim1 <- kppm(sim1$thin_sim,
                  clusters = "Thomas",
                  method = "mincon")
mat_sim1 <- kppm(sim1$thin_sim,
                 clusters = "MatClust",
                 method = "mincon")

thom_sim2 <- kppm(sim2$thin_sim,
                  clusters = "Thomas",
                  method = "mincon")
mat_sim2 <- kppm(sim2$thin_sim,
                 clusters = "MatClust",
                 method = "mincon")


datPCF_tom1 <- pcfinhom(sim1$thin_sim,
```

```
                              thom_sim1,
                              correction = "translate")
datPCF_sim_tom1 <- pcfinhom(simulate(thom_sim1, nsim = 1)[[1]],
                                  thom_sim1,
                                  correction = "translate")

datPCF_mat1 <- pcfinhom(sim1$thin_sim,
                              mat_sim1,
                              correction = "translate")
datPCF_sim_mat1 <- pcfinhom(simulate(mat_sim1, nsim = 1)[[1]],
                                  mat_sim1,
                                  correction = "translate")

datPCF_tom2 <- pcfinhom(sim2$thin_sim,
                              thom_sim2,
                              correction = "translate")
datPCF_sim_tom2 <- pcfinhom(simulate(thom_sim2, nsim = 1)[[1]],
                                  thom_sim2,
                                  correction = "translate")

datPCF_mat2 <- pcfinhom(sim1$thin_sim,
                              mat_sim2,
                              correction = "translate")
datPCF_sim_mat2 <- pcfinhom(simulate(mat_sim2, nsim = 1)[[1]],
                                  mat_sim2,
                                  correction = "translate")

par(mfrow = c(2, 3))
plot(datPCF_tom1, main = "Real Data (1)")
plot(datPCF_sim_tom1, main = "Thomas (1)")
plot(datPCF_sim_mat1, main = "Matern (1)")
plot(datPCF_tom2, main = "Real Data (2)")
plot(datPCF_sim_tom2, main = "Thomas (2)")
plot(datPCF_sim_mat2, main = "Matern (2)")
```

A.4 Model Fitting Code

```
# Description ------

# Script to run the modelling for the bei data in the
# second part of the project

# Load packages and bei Data-------

library(spatstat)

dat <- bei
dat_extra <- bei.extra

plot(bei, main = "bei data")

# set the seed so simulations are reproducible
set.seed(150)

# Fit Kernel Smooth -------

# Fit kernel smooth as initial estimate of intensity
initial_est <- density(dat)

# Save plot of this intensity
plot(initial_est,
     main = "Initial intensity Estimate Using Function Defaults")

# Default kernel is Gaussian so this is use since it
# hasn't been overridden

# the bandwidth sigma is found by a "simple rule of thumb"
# relating only to the size of the window (assuming some
# 0.1 * shortest side length)
```

```
# Fit densities with a large and small bandwidth to give
# markedly different estimates of the intensity
large_band <- density(dat, sigma = 200)
small_band <- density(dat, sigma = 20)

plot(large_band)

plot(small_band)

# Fit Models Using ppm for x and y ———

mod_stat <- ppm(dat ~ 1)

mod_x_y <- ppm(dat ~ x + y)

mod_xy <- ppm(dat ~ x * y)

mod_poly2_xy <- ppm(dat ~ poly(x, 2) + poly(y, 2))

mod_poly3_xy <- ppm(dat ~ poly(x, 3) + poly(y, 3))

mod_poly6_xy <- ppm(dat ~ poly(x, 6) + poly(y, 6))

par(mfrow = c(2, 2), mar = c(0.5, 0.5, 0.5, 0.5))
plot(predict(mod_poly2_xy))
plot(predict(mod_poly3_xy))
plot(predict(mod_poly6_xy))
plot(density(dat, sigma = 50))

plot(predict(mod_poly6_xy),
     main = "Model␣Fitted␣with␣Polynomial␣of␣Degree␣6")

datPCF_x <- pcfinhom(dat,
                     mod_poly6_xy,
                     correction = "translate")
```

```
datPCF_sim_x <- pcfinhom(simulate(mod_poly6_xy, nsim = 1)[[1]],
                         mod_poly6_xy,
                         correction = "translate")

plot(datPCF_x, main = "Real Data")

plot(datPCF_sim_x, main = "Simulated Data")

# Fit Models Using ppm for Elevation and Gradient ------

mod_elev <- ppm(dat ~ elev, data = dat_extra)

mod_grad <- ppm(dat ~ grad, data = dat_extra)

mod_g_e <- ppm(dat ~ elev + grad, data = dat_extra)

mod_ge <- ppm(dat ~ elev * grad, data = dat_extra)

plot(predict(mod_ge),
     main = "Model Fitted with the Interaction Term")

datPCF <- pcfinhom(dat,
                   mod_ge,
                   correction = "translate")
datPCF_sim <- pcfinhom(simulate(mod_ge, nsim = 1)[[1]],
                       mod_ge,
                       correction = "translate")

plot(datPCF, main = "Real Data")

plot(datPCF_sim, main = "Simulated Data")

# Fit Thomas Models ------

mod_tom <- kppm(dat ~ poly(x, 6) + poly(y, 6),
                clusters = "Thomas",
```

```
                    method = "palm")

plot(predict(mod_tom),
     main = "Thomas Cluster Model Fitted with Polynomial of Degree 6")

plot(mod_tom,
     main = "Thomas Cluster Model Fitted with Polynomial of Degree 6 with Data

datPCF_tom <- pcfinhom(dat,
                       mod_tom,
                       correction = "translate")
datPCF_sim_tom <- pcfinhom(simulate(mod_tom, nsim = 1)[[1]],
                           mod_tom,
                           correction = "translate")

plot(datPCF_tom, main = "Real Data")

plot(datPCF_sim_tom, main = "Simulated Data")

# Simulations from the Cox Model ------

# Set a new seed for these simulations
set.seed(500)

# Simulate from the Thomas model fitted above
tomsim <- simulate(mod_tom, nsim = 2)

plot(tomsim,
     main = "Data Simulated from the Thomas Model")

# Simulations from the Poisson Model -------

set.seed(600)

poly6_sim <- simulate(mod_poly6_xy, nsim = 2)
```

```
plot(poly6_sim,
     main = "Data Simulated from the Poisson Process Model")
```