Spatial Analysis of Madrid Traffic Data with MetricGraph

David Bolin and Alexandre Simas - INLA Workshop 2025

May 2025

1 Introduction

This tutorial explores spatial analysis of traffic data on road networks using the `MetricGraph` package. We'll work with real traffic data from Madrid, Spain, and learn how to:

1. Construct a metric graph from OpenStreetMap road network data
2. Add observations to the graph
3. Visualize traffic data on the network
4. Fit spatial models to analyze traffic patterns

The `MetricGraph` package provides tools for statistical analysis of data on network-structured domains, like road networks. It implements various types of Gaussian random fields on graphs and offers interfaces to INLA for Bayesian inference.

Let's begin by loading the necessary packages:

```r
library(data.table)
library(sf)
library(osmdata)
library(dplyr)
library(MetricGraph)
library(ggplot2)
library(inlabru)
```

2 Data Description

For this tutorial, we've prepared datasets containing traffic information for Madrid. The datasets include:

1. Traffic sensor locations in Madrid
2. Traffic measurements from these sensors during evening rush hour (6-7 PM) in March 2025

These data have been generated based on typical traffic patterns and distributions for educational purposes.

2.1 Loading the Data

For convenience, we've prepared data files containing: - The traffic sensor locations - The traffic measurements

```
Now we can load the pre-prepared data:


``` r
# Load the pre-prepared data files
madrid_traffic_avg_sf <- readRDS("madrid_traffic_avg.rds")
radar_locations_sf <- readRDS("radar_locations.rds")
```
```

```
# Examine the data
head(madrid_traffic_avg_sf)
```

```
## Simple feature collection with 6 features and 4 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -3.666633 ymin: 40.40367 xmax: -3.663641 ymax: 40.40921
## Geodetic CRS:  WGS 84
## # A tibble: 6 x 5
##                geometry avg_speed intensity occupation  load
##             <POINT [°]>     <dbl>     <dbl>      <dbl> <dbl>
## 1 (-3.664111 40.40921)      73       3064.        15       0
## 2 (-3.663641 40.40657)      58.4     3160.        17.5     0
## 3 (-3.666633 40.40434)      53       1075.        12.3     0
## 4 (-3.665753 40.40372)      50.0     1118.        10.2     0
## 5 (-3.666589 40.40367)      68.8     1395          5.6     0
## 6 (-3.665522 40.40554)      62.4     1476          6.2     0
```

3 Madrid Road Network as a Metric Graph

3.1 Understanding the Study Area

For this tutorial, we're focusing on a central area of Madrid defined by this bounding box:

```
# Define the Madrid bounding box used for this tutorial
small_bbox <- c(-3.73, 40.40, -3.65, 40.45)

# Create a simple data frame for the bounding box to visualize it
bbox_df <- data.frame(
  lon = c(small_bbox[1], small_bbox[3], small_bbox[3], small_bbox[1], small_bbox[1]),
  lat = c(small_bbox[2], small_bbox[2], small_bbox[4], small_bbox[4], small_bbox[2])
)
bbox_sf <- st_as_sf(bbox_df, coords = c("lon", "lat"), crs = 4326) %>%
  st_combine() %>%
  st_cast("POLYGON")
```

3.2 Exercise 1: Exploring the Graph Structure

Let us start by creating a graph from the OpenStreetMap data with some standard features.

```
madrid_sf_filtered <- opq(bbox = small_bbox) %>%
  add_osm_feature(key = "highway",
                  value = c("motorway", "trunk", "primary", "secondary")) %>%
  osmdata_sf()

madrid_graph_components <- graph_components$new(madrid_sf_filtered, perform_merges = TRUE,
tolerance = list(vertex_vertex = 0.001,
vertex_edge = 0.01))

madrid_graph <- madrid_graph_components$get_largest()
```

Let's explore the pre-loaded graph structure:

```
# Look at the graph structure
print(madrid_graph)
```

```
## A metric graph with  1328  vertices and  1496  edges.
##
## Vertices:
##   Degree 1: 54;  Degree 2: 981;  Degree 3: 199;  Degree 4: 91;  Degree 5: 3;
##   With incompatible directions:  6
##
## Edges:
##   Lengths:
##       Min: 0.001153276  ; Max: 1.256895  ; Total: 145.4731
##   Weights:
##       Columns: osm_id name access access:lanes admin_level alt_name bicycle bicycle:backward:conditi
##   That are circles:  0
##
## Graph units:
##   Vertices unit:  degree  ; Lengths unit:  km
##
## Longitude and Latitude coordinates:  TRUE
##   Which spatial package:  sp
##   CRS:  +proj=longlat +datum=WGS84 +no_defs
##
## Some characteristics of the graph:
##   Connected: TRUE
##   Has loops: FALSE
##   Has multiple edges: FALSE
##   Is a tree: FALSE
##   Distance consistent: unknown
```

```
##   Has Euclidean edges: unknown
```

```r
# What is the total length of the road network in the graph?
cat("Total length of road network:", round(sum(madrid_graph$edge_lengths), 2), "km\n")
```

```
## Total length of road network: 145.47 km
```

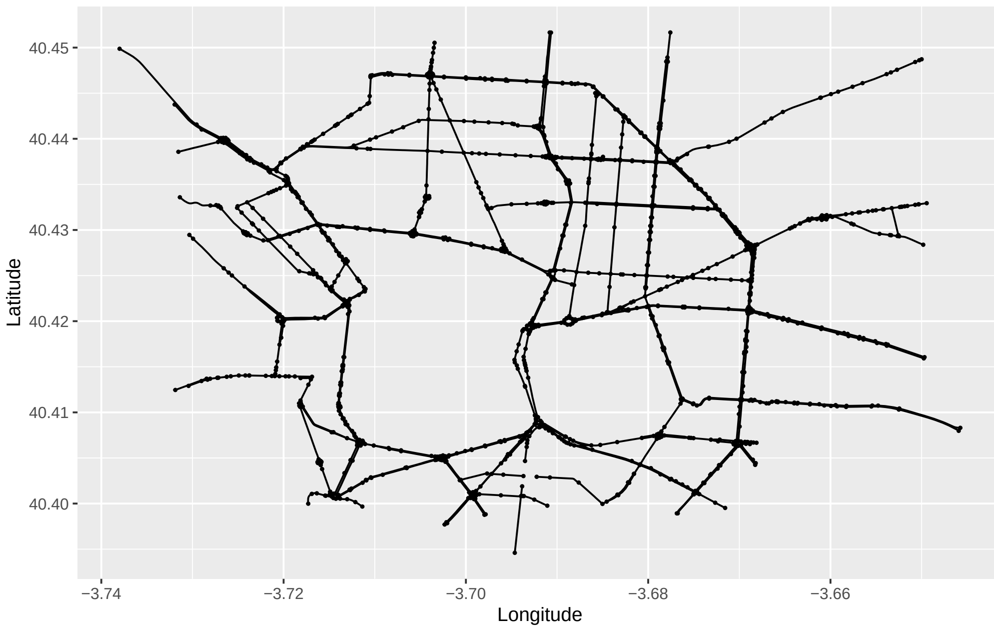
```r
# Compute additional characteristics
madrid_graph$compute_characteristics()
print(t(madrid_graph$characteristics))
```

```
##      has_loops connected has_multiple_edges is_tree
## [1,] FALSE     TRUE      FALSE              FALSE
```

3.3 Visualizing the Graph

Let's visualize our road network graph:

```r
# Plot the graph
p <- madrid_graph$plot(vertex_size = 0.5, edge_width = 0.5)
print(p)
```



4 Traffic Data on the Road Network

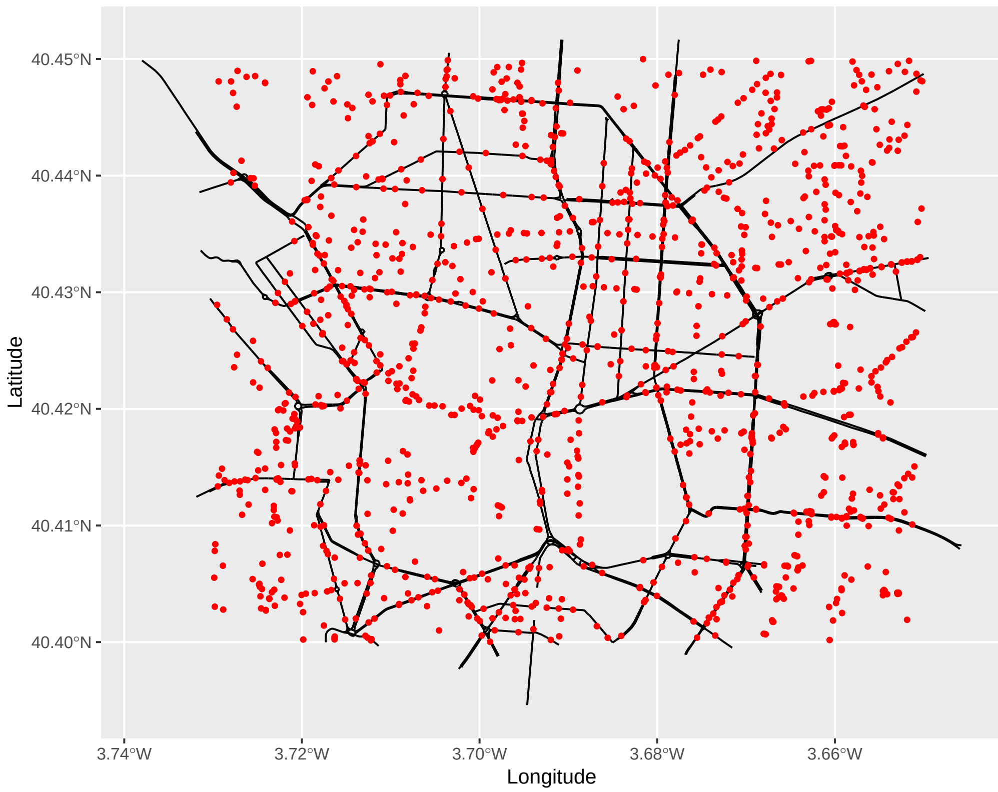
4.1 Examining Sensor Locations

Let's examine the traffic sensor locations in our study area:

```r
# How many sensors are in our study area?
cat("Number of sensors in the study area:", nrow(radar_locations_sf), "\n")
```

```
## Number of sensors in the study area: 1299
```

```r
# Plot the graph with the sensor locations
p <- madrid_graph$plot(vertex_size = 0, edge_width = 0.5)
p + geom_sf(data = radar_locations_sf, color = "red", size = 1)
```



4.2 Recreate the graph with more features

Let us recreate the graph with more features.

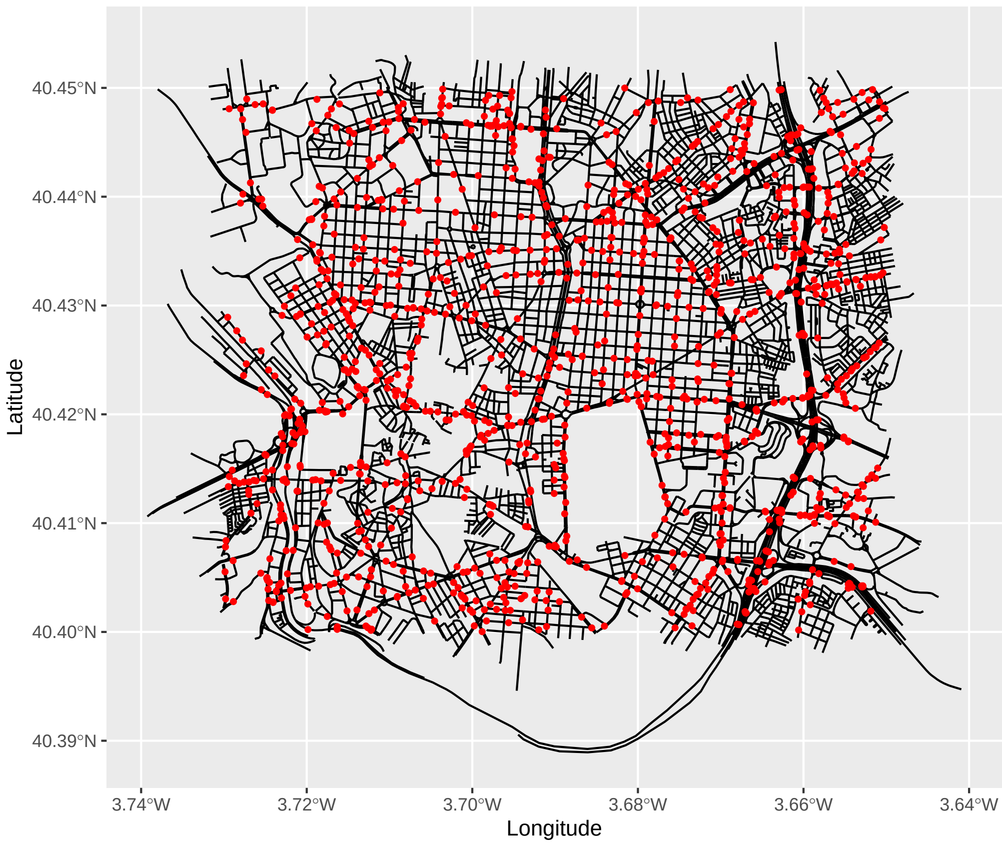
```
madrid_sf_new <- opq(bbox = small_bbox) %>%
  add_osm_feature(key = "highway",
                  value = c("motorway", "trunk", "primary", "secondary",
                            "tertiary", "residential", "unclassified",
                            "motorway_link", "trunk_link")) %>%
  osmdata_sf()

madrid_graph_components_new <- graph_components$new(madrid_sf_new, perform_merges = TRUE,tolerance = li

madrid_graph_new <- madrid_graph_components_new$get_largest()
```

4.3 Examining the new graph along with the data

```
# Plot the new graph with the sensor locations
p <- madrid_graph_new$plot(vertex_size = 0, edge_width = 0.5)
p + geom_sf(data = radar_locations_sf, color = "red", size = 1)
```



We can now add the data to the graph.

```
madrid_graph_new$add_observations(madrid_traffic_avg_sf)
```

```
## $removed
## [1] avg_speed  intensity  occupation load       .coord_x   .coord_y
## <0 rows> (or 0-length row.names)
```

4.4 Exercise 2: Checking Coordinate Reference Systems

A crucial step in spatial analysis is ensuring that all data uses the same coordinate reference system (CRS):

```
# Check the CRS of the radar locations
st_crs(radar_locations_sf)
```

```
## Coordinate Reference System:
##   User input: EPSG:4326
##   wkt:
## GEOGCRS["WGS 84",
##     ENSEMBLE["World Geodetic System 1984 ensemble",
##         MEMBER["World Geodetic System 1984 (Transit)"],
##         MEMBER["World Geodetic System 1984 (G730)"],
##         MEMBER["World Geodetic System 1984 (G873)"],
##         MEMBER["World Geodetic System 1984 (G1150)"],
##         MEMBER["World Geodetic System 1984 (G1674)"],
##         MEMBER["World Geodetic System 1984 (G1762)"],
##         MEMBER["World Geodetic System 1984 (G2139)"],
##         MEMBER["World Geodetic System 1984 (G2296)"],
##         ELLIPSOID["WGS 84",6378137,298.257223563,
```

```
##             LENGTHUNIT["metre",1]],
##         ENSEMBLEACCURACY[2.0]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["geodetic latitude (Lat)",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["geodetic longitude (Lon)",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     USAGE[
##         SCOPE["Horizontal component of 3D system."],
##         AREA["World."],
##         BBOX[-90,-180,90,180]],
##     ID["EPSG",4326]]
```

```r
# Check the CRS of the graph
print(madrid_graph)
```

```
## A metric graph with  1328  vertices and  1496  edges.
##
## Vertices:
##   Degree 1: 54;  Degree 2: 981;  Degree 3: 199;  Degree 4: 91;  Degree 5: 3;
##   With incompatible directions:  6
##
## Edges:
##   Lengths:
##       Min: 0.001153276  ; Max: 1.256895  ; Total: 145.4731
##   Weights:
##       Columns: osm_id name access access:lanes admin_level alt_name bicycle bicycle:backward:conditi
##   That are circles:  0
##
## Graph units:
##   Vertices unit:  degree  ; Lengths unit:  km
##
## Longitude and Latitude coordinates:  TRUE
##   Which spatial package:  sp
##   CRS:  +proj=longlat +datum=WGS84 +no_defs
##
## Some characteristics of the graph:
##   Connected: TRUE
##   Has loops: FALSE
##   Has multiple edges: FALSE
##   Is a tree: FALSE
##   Distance consistent: unknown

##   Has Euclidean edges: unknown
```

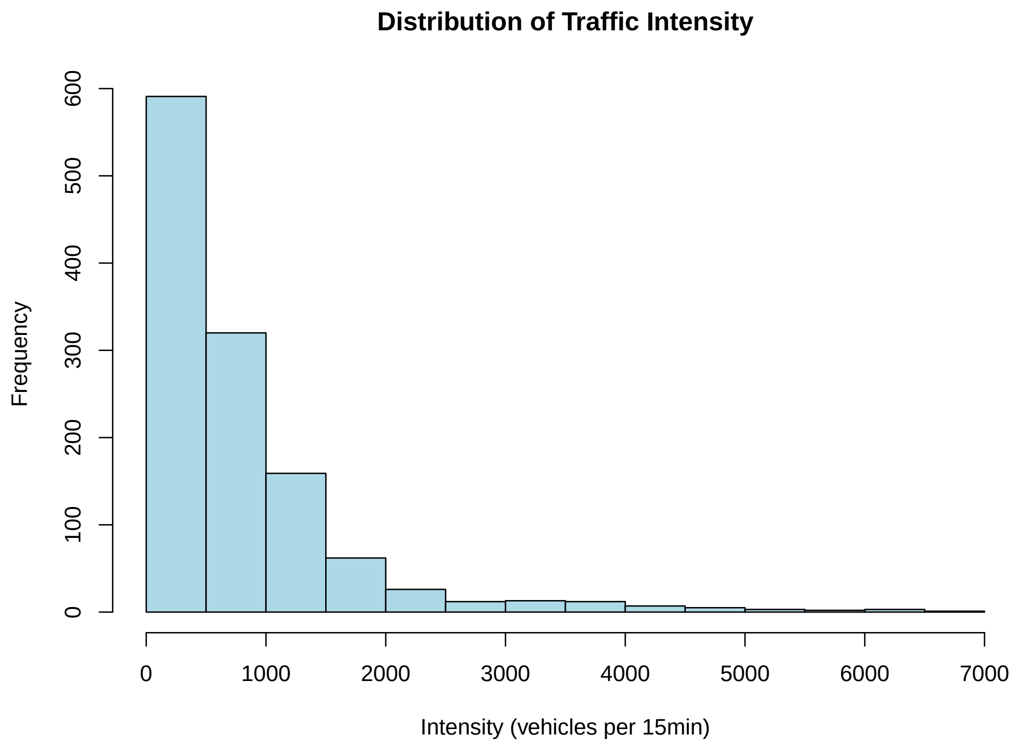
4.5 Examining Traffic Data

Now let's examine the pre-computed traffic measurements:

```r
# Basic statistics of the traffic data
summary(madrid_traffic_avg_sf)
```

```
##          geometry      avg_speed        intensity        occupation         load
##  POINT        :1216   Min.   : 0.000   Min.   :   0.0   Min.   : 0.00   Min.   :  0.00
##  epsg:4326    :   0   1st Qu.: 0.000   1st Qu.: 258.2   1st Qu.: 4.90   1st Qu.: 20.35
##  +proj=long...:   0   Median : 0.000   Median : 515.8   Median : 8.10   Median : 32.90
##                       Mean   : 6.071   Mean   : 788.8   Mean   :11.83   Mean   : 34.74
##                       3rd Qu.: 0.000   3rd Qu.:1003.5   3rd Qu.:14.04   3rd Qu.: 45.35
##                       Max.   :90.800   Max.   :6791.9   Max.   :99.05   Max.   :100.00
```

```r
# Histogram of traffic intensity
hist(madrid_traffic_avg_sf$intensity,
     main = "Distribution of Traffic Intensity",
     xlab = "Intensity (vehicles per 15min)",
     col = "lightblue")
```



5 Visualizing Traffic Data on the Graph

Let's check if the traffic data is already loaded into the graph:

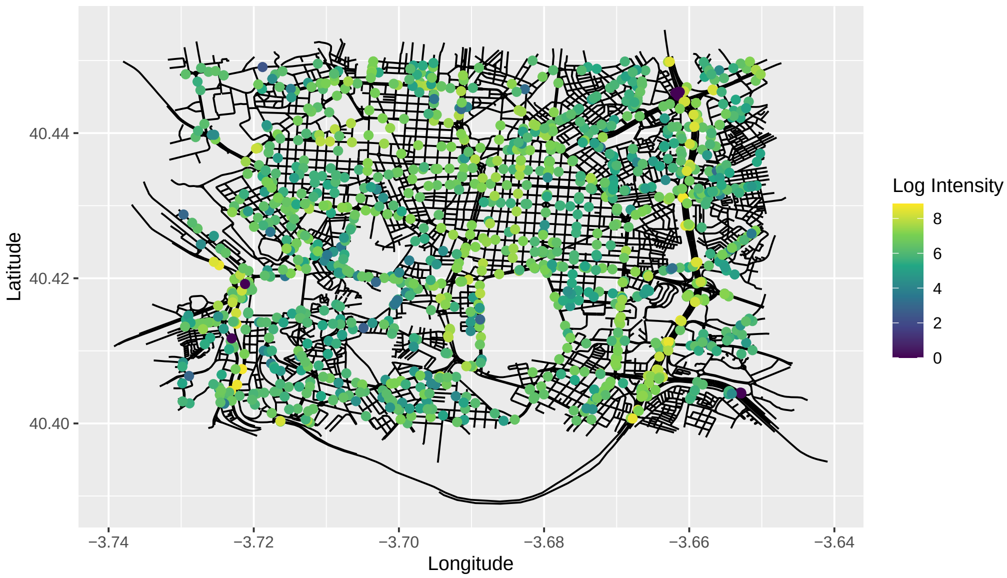
```r
madrid_graph_new$add_observations(madrid_traffic_avg_sf)
```

```
## $removed
## [1] avg_speed  intensity  occupation load       .coord_x   .coord_y
## <0 rows> (or 0-length row.names)
```

```r
madrid_graph_new$add_observations(
  madrid_graph_new$mutate(log_intensity = log(pmax(1, intensity))),
  clear_obs = TRUE
)

# Plot the data
madrid_graph_new$plot(data = "log_intensity",
```

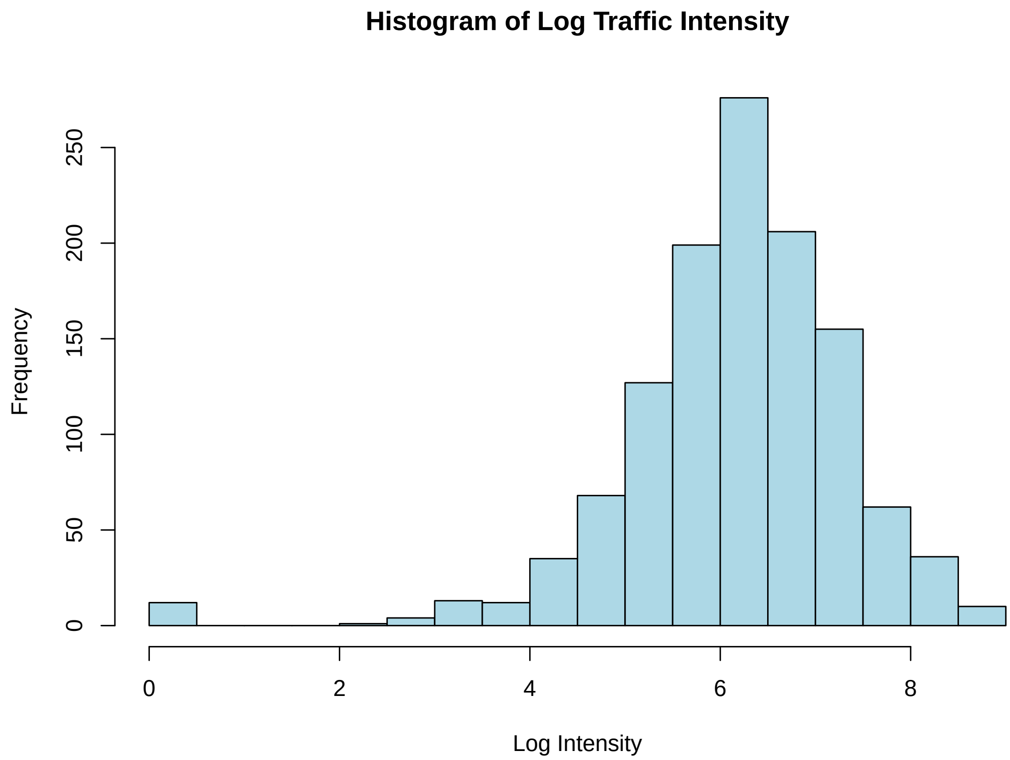
```
                   vertex_size = 0,
                   data_size = 2,
                   edge_width = 0.5) +
    scale_color_viridis_c(name = "Log Intensity")
```



6 Exploratory Data Analysis

Let's explore the relationships between different traffic metrics:

```
# Get the data from the graph
traffic_data <- madrid_graph_new$get_data()
# Histogram of log intensity
hist(traffic_data$log_intensity,
     main = "Histogram of Log Traffic Intensity",
     xlab = "Log Intensity",
     col = "lightblue",
     breaks = 20)
```



7 Spatial Modeling

Now let's fit a spatial model to the log intensity using a Whittle-Matérn random field with $\alpha = 1$, which corresponds to an exponential covariance model:

```
# Fit the model
fit_alpha1 <- graph_lme(log_intensity ~ 1,
                        graph = madrid_graph_new,
                        BC = 0,
                        model = list(type = "WhittleMatern", alpha = 1))

# Look at the model summary
summary(fit_alpha1)
```

```
##
## Latent model - Whittle-Matern with alpha = 1
##
## Call:
## graph_lme(formula = log_intensity ~ 1, graph = madrid_graph_new,
##     model = list(type = "WhittleMatern", alpha = 1), BC = 0)
##
## Fixed effects:
##             Estimate Std.error z-value Pr(>|z|)
## (Intercept)   6.1601    0.1421   43.36   <2e-16 ***
##
## Random effects:
##       Estimate Std.error z-value
## tau    0.41538   0.02282  18.205
## kappa  0.63398   0.18992   3.338
##
```

```
## Random effects (Matern parameterization):
##       Estimate Std.error z-value
## sigma   2.1380    0.2856   7.487
## range   3.1547    0.9201   3.429
##
## Measurement error:
##          Estimate Std.error z-value
## std. dev  0.71522   0.02888   24.77
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log-Likelihood:  -1741.588
## Number of function calls by 'optim' = 21
## Optimization method used in 'optim' = L-BFGS-B
##
## Time used to:     fit the model =  31.84341 secs
```

7.1 Interpreting the Model Results

The estimated parameters of the Whittle-Matérn model provide insights into the spatial correlation of traffic intensity:

```r
# Extract parameters from the model
summary(fit_alpha1)
```

```
##
## Latent model - Whittle-Matern with alpha = 1
##
## Call:
## graph_lme(formula = log_intensity ~ 1, graph = madrid_graph_new,
##     model = list(type = "WhittleMatern", alpha = 1), BC = 0)
##
## Fixed effects:
##             Estimate Std.error z-value Pr(>|z|)
## (Intercept)   6.1601    0.1421   43.36   <2e-16 ***
##
## Random effects:
##       Estimate Std.error z-value
## tau    0.41538   0.02282  18.205
## kappa  0.63398   0.18992   3.338
##
## Random effects (Matern parameterization):
##       Estimate Std.error z-value
## sigma   2.1380    0.2856   7.487
## range   3.1547    0.9201   3.429
##
## Measurement error:
##          Estimate Std.error z-value
## std. dev  0.71522   0.02888   24.77
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log-Likelihood:  -1741.588
## Number of function calls by 'optim' = 21
## Optimization method used in 'optim' = L-BFGS-B
```

```
##
## Time used to:     fit the model =  31.84341 secs
```

- **Intercept**: The average log traffic intensity across the network
- **Range**: The effective distance (in kilometers) at which spatial correlation becomes negligible
- **Sigma**: The standard deviation of the spatial process, indicating the variability of traffic intensity
- **Nugget**: The measurement error or micro-scale variation

7.2 Predicting on the Graph

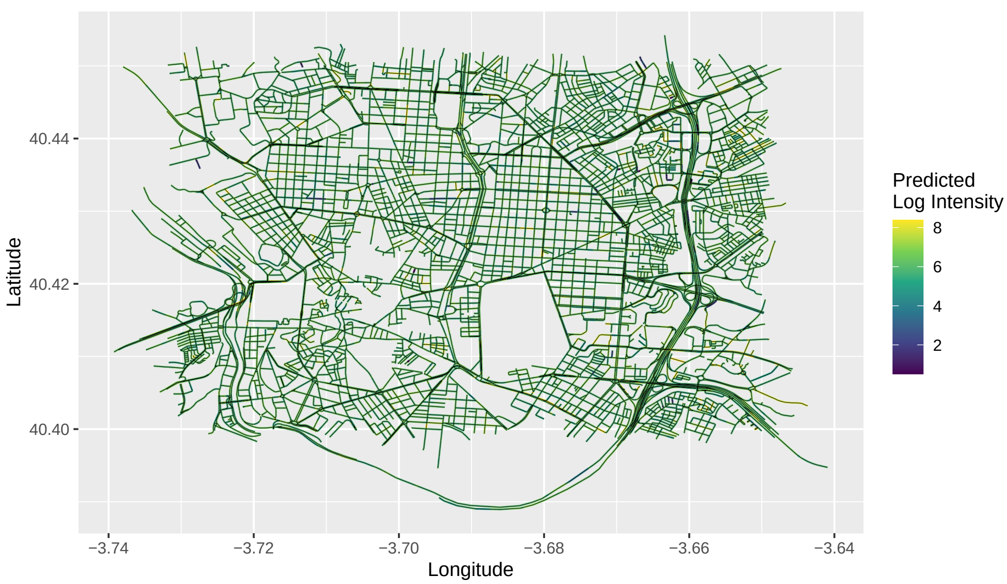
We can use our fitted model to make predictions at unobserved locations on the graph:

```r
# Build a mesh on the graph for prediction (if not already done)
if (length(madrid_graph_new$mesh$VtE) == 0) {
  madrid_graph_new$build_mesh(h = 0.01)
}

# Create a data frame with the prediction locations
# Extract mesh vertices as sf object
mesh_vertices_df <- data.frame(
  edge_number = madrid_graph_new$mesh$VtE[,1],
  distance_on_edge = madrid_graph_new$mesh$VtE[,2]
)

# Make predictions
pred <- predict(fit_alpha1, newdata = mesh_vertices_df,
                normalize = TRUE)

# Plot the predictions
madrid_graph_new$plot_function(X = pred$mean,
                               vertex_size = 0,
                               edge_width = 0.5) +
  scale_color_viridis_c(name = "Predicted\nLog Intensity")
```



7.3 Fitting the model with inlabru interface

Let us create the SPDE model object:

```r
spde_model <- graph_spde(madrid_graph_new, alpha = 1)
```

Now, let us create the data object:

```r
data_spde <- graph_data_spde(spde_model, loc_name = "loc")
```

Let us now fit the model:

```r
f.s <- log_intensity ~ Intercept(1) +
                    field(loc, model = spde_model)
fit_alpha1_inlabru <- bru(f.s,
                    data = data_spde[["data"]])
```

Now, let us look at the model summary:

```r
summary(fit_alpha1_inlabru)
```

```
## inlabru version: 2.12.0
## INLA version: 25.03.13
## Components:
## Intercept: main = linear(1), group = exchangeable(1L), replicate = iid(1L), NULL
## field: main = cgeneric(loc), group = exchangeable(1L), replicate = iid(1L), NULL
## Likelihoods:
##   Family: 'gaussian'
##     Tag: ''
##     Data class: 'metric_graph_data', 'list'
##     Response class: 'numeric'
##     Predictor: log_intensity ~ .
##     Used components: effects[Intercept, field], latent[]
## Time used:
##     Pre = 1.9, Running = 1.3, Post = 0.173, Total = 3.38
## Fixed effects:
##            mean    sd 0.025quant 0.5quant 0.975quant  mode kld
## Intercept 6.008 0.162      5.663    6.014      6.316 6.014   0
##
## Random effects:
##   Name     Model
##     field CGeneric
##
## Model hyperparameters:
##                                         mean    sd 0.025quant 0.5quant 0.975quant  mode
## Precision for the Gaussian observations 1.98 0.160      1.685    1.976      2.315 1.963
## Theta1 for field                        0.88 0.054      0.772    0.881      0.984 0.884
## Theta2 for field                        1.86 0.472      1.033    1.830      2.876 1.684
##
## Deviance Information Criterion (DIC) ...............: 3156.56
## Deviance Information Criterion (DIC, saturated) ....: 1747.61
## Effective number of parameters .....................: 529.61
##
## Watanabe-Akaike information criterion (WAIC) ...: 3159.00
## Effective number of parameters .................: 414.03
##
## Marginal log-Likelihood:  -1762.88
##  is computed
## Posterior summaries for the linear predictor and the fitted values are computed
## (Posterior marginals needs also 'control.compute=list(return.marginals.predictor=TRUE)')
```

Now, let us see the estimated parameters:

```
spde_result <- spde_metric_graph_result(fit_alpha1_inlabru,
                    "field", spde_model)

summary(spde_result)
```

```
##          mean       sd 0.025quant 0.5quant 0.975quant    mode
## sigma 3.11227 0.706029    2.05712  3.00056     4.8127 2.83778
## range 7.20692 3.888480    2.82428  6.16900    17.5581 4.70193
```

8 Exercise 3: Further analysis

- Consider removing the zero values from the data.
- Try pruning the graph.
- Try fitting a model with smoothness parameter alpha = 2.
- Try fitting the model estimating the smoothness parameter by using `rSPDE` interface.
- Try fitting the model with `INLA` interface.
- Check the connected components of the graph and consider whether more features should be added to the graph, or if the tolerances should be changed.