Sfislands: An R Package for Accommodating Islands and Disjoint Zones in Areal Spatial Modelling

by Kevin Horan, Katarina Domijan, and Chris Brunsdon

Abstract Fitting areal models which use a spatial weights matrix to represent relationships between geographical units can be a cumbersome task, particularly when these units are not well-behaved. The two chief aims of **sfislands** are to simplify the process of creating an appropriate neighbourhood matrix, and to quickly visualise the predictions of subsequent models. The package uses visual aids in the form of easily-generated maps to help this process. This paper demonstrates how **sfislands** could be useful to researchers. It begins by describing the package's functions in the context of a proposed workflow. It then presents two worked examples showing a selection of potential use-cases. These range from earthquakes in Indonesia, to river crossings in London. We aim to show how the **sfislands** package streamlines much of the human workflow involved in creating and examining such models.

1 Introduction

A key feature which differentiates spatial statistics is the non-independence of observations and the expectation that neighbouring units will be more similar than non-neighbouring ones (Tobler, 1970). If this is not accounted for, the assumptions of many types of models will be violated. The relationships between all spatial units in a study can be represented numerically in a spatial weights matrix. In order to build this, we must first decide on what constitutes being a neighbour. We might see this as a continuous relationship where degree of neighbourliness is a function of connectivity, which could be represented as some measure of distance. Alternatively it could be a binary situation where each pair of units either are (1) or are not (0) neighbours. This can be based on a condition such as contiguity of some sort, or a distance constraint. It is the job of the modeller to formulate a hypothesis which justifies their choice of neighbourhood structure.

For R users, the **spdep** package (Bivand, 2022) has long been popular for the creation of these matrices. More recently, in reference to the increasing use of **sf** structures (Pebesma, 2018), the **sfdep** package (Parry and Locke, 2024) has presented generally similar functionality by wrapping **spdep** functions with functions that follow the **sf** naming convention (function names starting with `st_`), as well as a "use a data.frame for everything" attitude.

The most appropriate form of neighbourhood structure will depend on the specific context. Briz-Redón et al. (2021) compared different structures in the context of COVID-19 data. They note that Earnest et al. (2007) found that distance-based matrices were more appropriate when examining birth defects in Australia, whereas Duncan et al. (2017) found that a first-order contiguity structure produced a better fit than others in the context of lip cancer incidence in Scotland.

The most commonly used neighbourhood structure is one based on first-order queen contiguity, where units are considered neighbours if they share at least a vertex or boundary. However, as the name suggests, this will lead to problems when non-contiguous units such as islands or exclaves are present. Less obviously, depending on how the geographic units are described, areas on either side of rivers may be inappropriately classified as neighbours or not neighbours. Furthermore, the presence of infrastructure such as tunnels, bridges or ferry services might be satisfactory to meet our hypothesis of the required degree of connectivity to be considered neighbours. Again, such information may not be apparent from a basic set of polygons. In order to create what a researcher considers to be an appropriate neighbourhood structure, incorporating all of the domain knowledge that

they might have about the system, it should be simple and intuitive to add and remove connections between spatial units. This might mean adding links to account for man-made infrastructure, or cutting links to incorporate natural barriers such as rivers or mountains.

The aim of **sfislands** (Horan et al., 2024) is to deal with the situations described above in a convenient and open manner. It allows us to set up a structure, quickly map it, and then examine whether or not we are happy with how it represents our hypothesis of relationships between units. The structure can then be edited and the process re-iterated until we have described a spatial relationship structure with which we are satisfied.

It should be noted that while this package offers convenient tools for the examination, visualisation, addition and removal of neighbourhood linkages between units, such an approach to dealing with disconnected units is not always appropriate and other methodologies are available. These issues are discussed in more depth by Bivand and Portnov (2004) and Freni-Sterrantino et al. (2018).

The above can be considered as the *pre-functions* of the package. A second category of features, which we refer to as *post-functions*, are for use after the creation of a model. Having fit a model with **mgcv** (Wood, 2011) in particular, the process of extracting estimates for certain types of effects can be somewhat awkward. These *post-functions* augment the original dataframe with these estimates and their standard errors in tidy format. They also allow for quick visualisation of the output in map form.

1.1 Typical use-cases

In this paper, we will look at two examples to show different use-cases for **sfislands**. The first example focuses on earthquakes in Indonesia. It shows a scenario where all of the functions are used, from setting up contiguities, to modelling and examining the predictions of the model. The second example looks at London and how, despite an absence of islands, the presence of a river means that some of the pre-functions of **sfislands** can be useful.

2 Why use sfislands?

Below, we outline some of the benefits of the package in the context of a proposed workflow for fitting areal spatial models.

Step 1: *Pre-functions* for setting up neighbourhood structure

1. It addresses an issue commonly seen in online help forums where an inexperienced user wishes to get started with a model but fails at the first hurdle because their neighbourhood structure contains empty records. **sfislands** will include a contiguity for all units.

2. It gives tools to immediately visualise this structure as a map.

3. These maps are created using **ggplot2** (Wickham, 2016), which allows users to apply additional styling and themes using **ggplot2** syntax.

4. As the nodes can be labelled by index, it makes it very easy to add and remove connections as appropriate with confidence and without reference to the names of these areas.

5. Connections which have been induced by a function from the package but which are not based on geographical contiguity can be accessed to ensure openness in the process.

Table 1: Pre-functions: setting up a neighbourhood structure.

function	purpose
st_bridges()	create a neighbourhood contiguity structure, with a k-nearest neighbours condition for islands
st_quickmap_nb()	check structure visually on map
st_check_islands()	check the contiguities which have been assigned to islands
st_force_join_nb()	enforce changes by adding connections
st_force_cut_nb()	enforce changes by removing connections

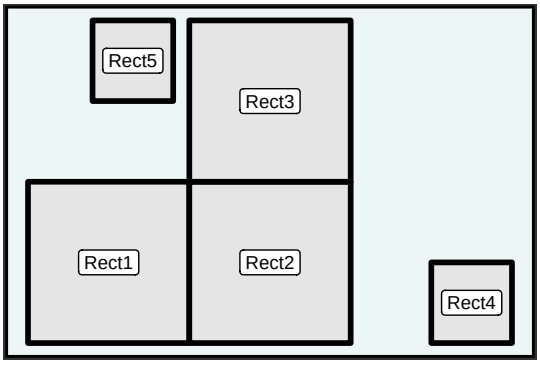

Figure 1: Simplified scenario with five rectangles.

Step 2: Modelling

These neighbourhood structures can be used in modelling packages such as **mgcv**, **brms** (Bürkner, 2017), **r-inla** (Bakka et al., 2018) and more.

Step 3: *Post-functions* for models

1. It simplifies the process of extracting estimates from models, such as those with random effects and Markov random field structures created using **mgcv**. Compatibility with more packages can be added at a future date.

2. These effects can be quickly visualised as **ggplot2** maps.

3 Pre-functions

The first group of functions, shown in Table 1, deals with the creation of a neighbourhood structure in the presence of discontiguities. The resultant structure can be quickly mapped to check if it is satisfactory. Connections can be forcibly added or removed by name or index number. By an iterative process of changes and examination of a quickly-generated guide map, a satisfactory structure can be decided upon.

We will now go through each function in more detail using the set of rectangles shown in Figure 1 for demonstration purposes. Rectangles 1, 2 and 3 are contiguous while 4 and 5 can be viewed as "islands".

3.1 st_bridges()

This function requires at least two arguments: an **sf** dataframe and, from that, the name of one column of unique row identifiers, ideally names, of each spatial unit. It creates a neighbourhood structure where non-island units are joined by first-order queen contiguity, while island units are joined to their k-nearest neighbours. The output is a *named* neighbourhood structure in either list or matrix form as desired, which can be either a standalone object

or included as an additional column in the original **sf** dataframe. While we have chosen to append the neighbourhood structure to the original data frame in this way by default, the user should be warned that any subsequent row sub-setting (filter) operation on this object will invalidate the list column involved. While it is not necessary in all modelling packages for the neighbourhood list or matrix to be *named*, it is good practice to do so and is mandatory when using, for example, **mgcv**.

One solution when confronted with islands in a dataset is to simply exclude them from the analysis. In the first two examples of using `st_bridges()`, we have chosen to ignore islands with the argument `remove_islands = TRUE` and to return a list and matrix structure respectively by specifying this in the `nb_structure` argument and choosing `add_to_dataframe = FALSE`:

```
# output a named list

st_bridges(rectangles,
           "name",
           remove_islands = TRUE,
           nb_structure = "list",
           add_to_dataframe = FALSE) |>
  head()

#> $Rect1
#> [1] 2 3
#>
#> $Rect2
#> [1] 1 3
#>
#> $Rect3
#> [1] 1 2

# output a named matrix

st_bridges(rectangles,
           "name",
           remove_islands = TRUE,
           nb_structure = "matrix",
           add_to_dataframe = FALSE) |>
  head()

#>       [,1] [,2] [,3]
#> Rect1    0    1    1
#> Rect2    1    0    1
#> Rect3    1    1    0
```

Alternatively, in the following examples, we choose to join islands to their 1 nearest neighbour, which is the default setting, and to return the output as a column called "nb" in the original **sf** dataframe (`add_to_dataframe = "TRUE"` is the default setting):

```
# output a named list as a column "nb" in original dataframe

st_bridges(rectangles,
           "name",
           link_islands_k = 1,
           nb_structure = "list") |>
  head()
```

```
#> Simple feature collection with 5 features and 2 fields
#> Geometry type: POLYGON
#> Dimension:     XY
#> Bounding box:  xmin: 0 ymin: 0 xmax: 6 ymax: 4
#> CRS:           NA
#>    name      nb                       geometry
#> 1 Rect1    2, 3 POLYGON ((0 0, 0 2, 2 2, 2 ...
#> 2 Rect2 1, 3, 4 POLYGON ((2 0, 2 2, 4 2, 4 ...
#> 3 Rect3 1, 2, 5 POLYGON ((2 2, 2 4, 4 4, 4 ...
#> 4 Rect4       2 POLYGON ((5 0, 5 1, 6 1, 6 ...
#> 5 Rect5       3 POLYGON ((0.8 3, 0.8 4, 1.8...

# output a named matrix as a column "nb" in original dataframe

st_bridges(rectangles,
           "name",
           link_islands_k = 1,
           nb_structure = "matrix") |>
  head()

#> Simple feature collection with 5 features and 2 fields
#> Geometry type: POLYGON
#> Dimension:     XY
#> Bounding box:  xmin: 0 ymin: 0 xmax: 6 ymax: 4
#> CRS:           NA
#>    name nb.1 nb.2 nb.3 nb.4 nb.5                       geometry
#> 1 Rect1    0    1    1    0    0 POLYGON ((0 0, 0 2, 2 2, 2 ...
#> 2 Rect2    1    0    1    1    0 POLYGON ((2 0, 2 2, 4 2, 4 ...
#> 3 Rect3    1    1    0    0    1 POLYGON ((2 2, 2 4, 4 4, 4 ...
#> 4 Rect4    0    1    0    0    0 POLYGON ((5 0, 5 1, 6 1, 6 ...
#> 5 Rect5    0    0    1    0    0 POLYGON ((0.8 3, 0.8 4, 1.8...
```

These structures can serve as the input to models in **brms**, **r-inla**, **rstan** (Stan Development Team, 2020) or **mgcv**. **brms** requires a matrix structure while **mgcv** models use a list. Rather than having a separate neighbours object, it is included in the original **sf** dataframe as a named list or matrix, in the spirit of the **sfdep** package.

3.2 st_quickmap_nb()

It is much more intuitive to examine these structures visually than in matrix or list format. This can be done with the `st_quickmap_nb()` function as shown in Figure 2.

```
# default is 'nodes = "point"'

st_bridges(rectangles,
           "name",
           link_islands_k = 1) |>
  st_quickmap_nb()
```

If we wish to make edits, it might be more useful to represent the nodes numerically rather than as points (Figure 3).

```
# with 'nodes = "numeric"'

st_bridges(rectangles,
           "name",
```

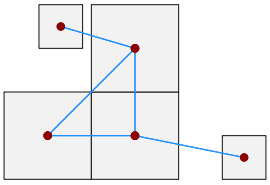

Figure 2: Queen contiguity and islands connected to nearest neighbour.

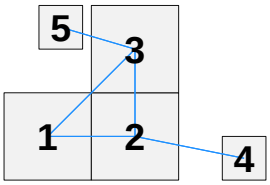

Figure 3: Queen contiguity and islands connected to nearest neighbour. Nodes are shown as numeric indices.

```
          link_islands_k = 1) |>
  st_quickmap_nb(nodes = "numeric")
```

3.3 st_check_islands()

This function will show us transparently what connections have been made which are not based on contiguity. It gives both the name and index number of each pair of added connections. In this example, two pairs have been added.

```
# show summary of non-contiguous connections in a dataframe

st_bridges(rectangles,
           "name",
           link_islands_k = 1) |>
  st_check_islands()

#>   island_names island_num nb_num nb_names
#> 1        Rect4          4      2    Rect2
#> 2        Rect5          5      3    Rect3
```

3.4 st_force_join_nb()

If we feel that 4 should also be connected to 3, this can be done by forcing a join (Figure 4).

```
# add an extra connection using numeric index

st_bridges(rectangles, "name",
           link_islands_k = 1) |>
  st_force_join_nb(3,4) |>
  st_quickmap_nb(nodes = "numeric")
```

3.5 st_force_cut_nb()

And perhaps there is a wide river between rectangles 1 and 2 which justifies removing the connection. We will edit it this time using names (Figure 5).

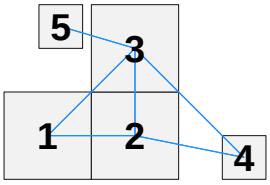

Figure 4: With an additional connection between 3 and 4.

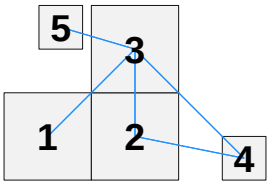

Figure 5: With the removal of connection between 1 and 2.

```
# remove an existing connection using unit name, not index

st_bridges(rectangles, "name",
           link_islands_k = 1) |>
  st_force_join_nb(3,4) |>
  st_force_cut_nb("Rect1","Rect2") |>
  st_quickmap_nb(nodes = "numeric")
```

Having decided upon an appropriate neighbourhood structure, the next step is to use this in the context of a model. The use of such structures is particularly associated with CAR (conditional autoregressive) or ICAR-type (intrinsic conditional autoregressive) models (Besag, 1974). These are often implemented in a Bayesian framework using **brms**, **r-inla** or **rstan**. For example, the **brms** ICAR structure requires the neighbourhood relationships to be in matrix form. The pre-functions will output the neighbourhood structure in the desired format for use in any of these frameworks. A convenient frequentist alternative is to use the **mgcv** package which requires a named list of neighbours. It has the functionality to create such models using `bs="mrf"`. It also has the ability to combine these with a hierarchical structure using `bs="re"`. While the outputs from the Bayesian structures mentioned above can be extracted in the same way as any other component of the model, it can be somewhat awkward to get the estimates from **mgcv** models. **sfislands** has two post-functions to conveniently extract and visualise these.

4 Post-functions

Table 2 shows the second set of functions in the package and their purpose.

4.1 st_augment()

This function augments the original dataframe with the estimated means and standard errors of the spatially varying predictions from a fitted **mgcv** model in a similar manner to how the **broom** package (Robinson et al., 2023) operates. The `geometry` column, as per convention, remains as the last column of the augmented dataframe, while the predictions are positioned

Table 2: Post-functions: tidy estimates from mgcv.

function	purpose
st_augment()	augment the original dataframe with model predictions
st_quickmap_preds()	generate quick maps of these predictions

Table 3: The naming procedure for augmented columns from different mgcv structures.

mgcv syntax	column name
s(region, bs = 're')	random.effect.region
s(region, covariate, bs = 're')	random.effect.covariate | region
s(sub-region, bs = 'mrf', xt = list(nb = data$nb))	mrf.smooth.sub-region
s(sub-region, by = covariate, bs = 'mrf', xt = list(nb = data$nb))	mrf.smooth.covariate | sub-region

immediately before it. [1] The spatially varying predictions which `st_augment()` extracts from an **mgcv** model are

- random effects (which are called in **mgcv** with `bs='re'`), and
- ICAR components (`bs='mrf'`).

Consider the model structure described in the code below using **mgcv** syntax. In this model y is the dependent variable which is being estimated with a fixed intercept, a fixed slope for some covariate, a set of random intercepts and slopes for the covariate at a *region* level, and a set of ICAR varying intercepts and slopes at a lower *sub-region* level.

```
# creating an mgcv model

mgcv::gam(
  y ~ covariate +                  # fixed intercept and effect for covariate
    s(region, bs = "re") +                 # random intercept at level region
    s(region, covariate, bs = "re") +         # random slopes at level region
    s(sub-region,
      bs = 'mrf',
      xt = list(nb = data$nb),
      k = k) +                    # ICAR varying intercept at level sub-region
    s(sub-region, by = covariate,
      bs = 'mrf',
      xt = list(nb = data$nb),
      k = k),           # ICAR varying slope for covariate at level sub-region
  data = data,
  method = "REML")
```

When labelling the new prediction columns which are augmented to the original dataframe from such a model, `st_augment()` follows the formula syntax of the **lme4** package (Bates et al., 2015), where the pipe symbol (`|`) indicates "*grouped by*". Table 3 shows how the augmented columns in this scenario would be named. Each column name begins with either `random.effect.` or `mrf.smooth.` as appropriate. An additional column is also added for the standard error of each prediction, as calculated by **mgcv**. These columns are named as above but with `se.` prepended (e.g. `se.random.effect.region`).

4.2 st_quickmap_preds()

These estimates can then be quickly mapped. As it is possible to include more than 1 spatially varying component, the output of this function is a list of plots. They can be viewed individually by indexing, or all at once using, for example, the `plotlist` argument from the `ggarrange()` function which is part of the **ggpubr** (Kassambara, 2023) package. We will see this function in practice in the following example. The maps which it generates are automatically titled and subtitled according to the type of effect. For example, the map showing predictions for `random.effect.region` will have "*region*" as its title and "*random.effect*" as its subtitle.

[1] In a similar way, `st_augment()` can also be used to append the random effects from **lme4** (Bates et al., 2015) and **nlme** (Pinheiro et al., 2023) models to an **sf** dataframe, which can then be easily mapped using `st_quickmap_preds()`. Compatibility with models created using different packages can be introduced in the future.

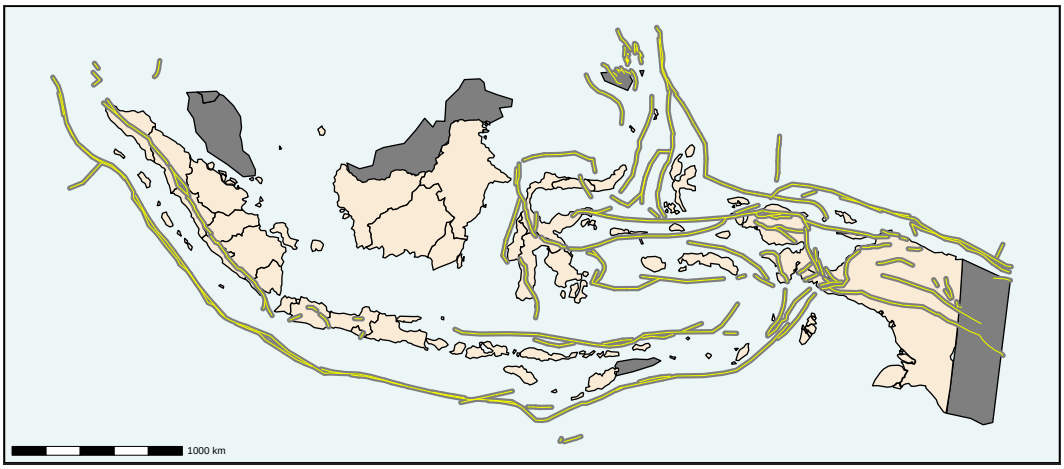

Figure 6: Indonesia faults. Surrounded by a 10 kilometre buffer.

5 Indonesia (example 1)

Modelling earthquakes in Indonesia serves as a good example to demonstrate this package. Firstly, Indonesia is composed of many islands. Secondly, earthquake activity is known to be associated with the presence of faults which exist below sea level and thus do not respect land boundaries. Therefore it is reasonable to expect similar behaviour in nearby provinces regardless of whether or not they are contiguous. We aim to model the incidence, or count per unit area, of earthquake activity by province across Indonesia, controlling for proximity to faults.

5.1 Data

The data for this section have been downloaded from the National Earthquake Information Center, USGS earthquake catalogue. The datasets with accompanying explanations are available at `https://github.com/horankev/quake_data`. They capture all recorded earthquakes in and close to Indonesia from the beginning of January 1985 to the end of December 2023. Figure 6 shows a map of Indonesia, divided into 33 provinces, with other neighbouring or bordering countries filled in grey. The many local faults which lie within 300km of the shore are shown in yellow with green outlines.

To get an interpretable measure of the concentration of faults in any area, these faults are transformed from linestrings to polygons by setting a buffer of 10km around them, which explains their green outline. Now both our faults and the sizes of provinces are in units of kilometres squared. This means we can generate a unitless metric of what proportion of any administrative unit is covered by these buffered faults. This measure across provinces is shown in Figure 7.

Earthquake incidence per province has been calculated as the total number of earthquakes with an epicentre within that province per unit area. We have restricted counts to earthquakes >5.5 on the moment magnitude scale, which is the point at which they are often labelled as potentially damaging.

The occurrences of these earthquakes are shown in Figure 8, their total per province in Figure 9, and finally, their incidence or count per square kilometre can be seen in Figure 10.

5.2 Model

As this is count data, we will model it as a Poisson distribution with λ as the mean count per province. For $i = 1, ..., n$ provinces, the dependent variable in this model is

$$y_i = \text{earthquake count}_i \tag{1}$$

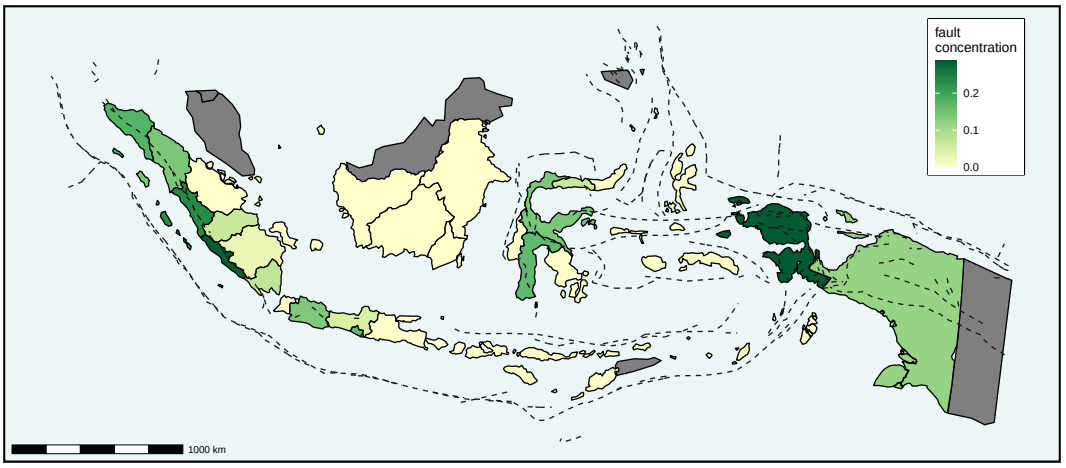

Figure 7: Indonesia fault concentration. Square kilometre of buffered fault per square kilometre of province area.

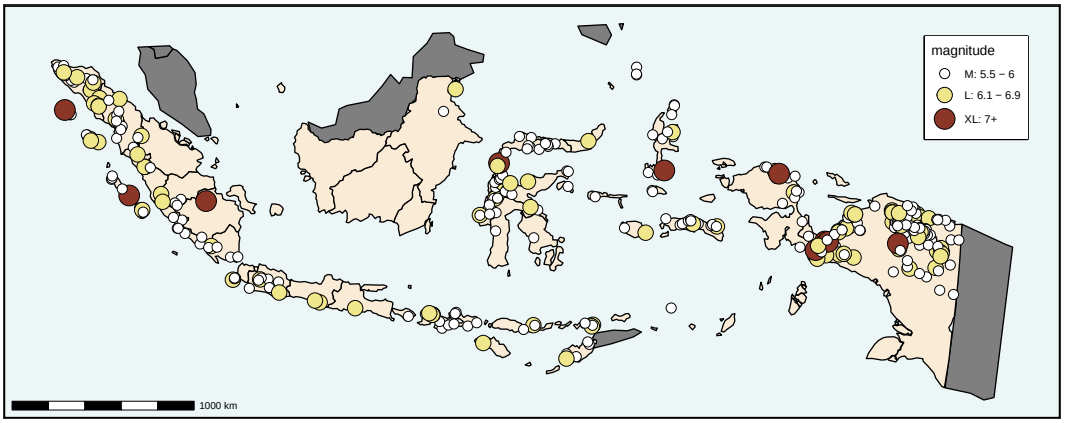

Figure 8: Earthquakes in Indonesia of magnitude > 5.5, 1985-2023. Categorised by magnitude as medium, large or extra-large.

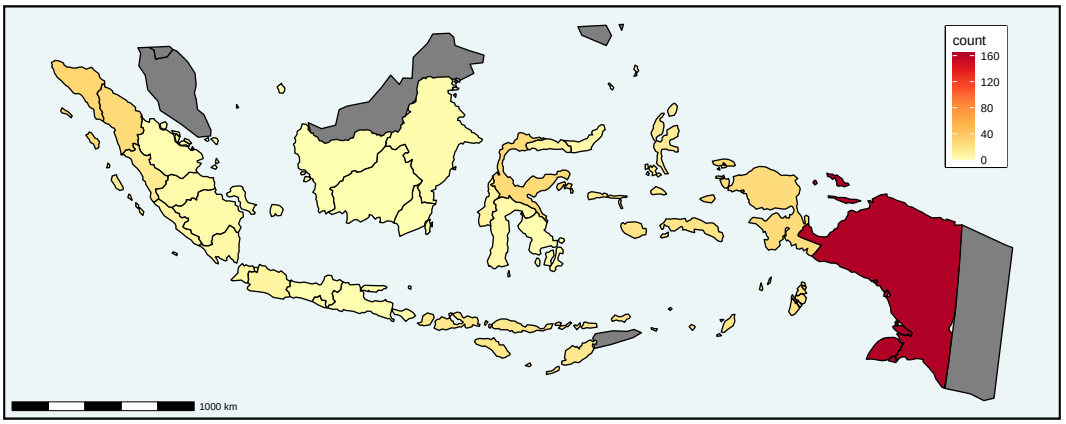

Figure 9: Earthquake count in Indonesia, 1985-2023, mag > 5.5: count by province.

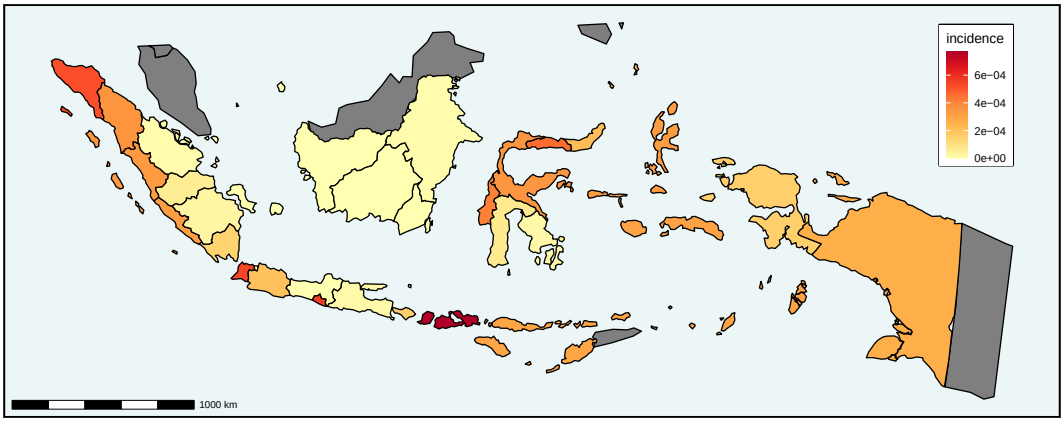

Figure 10: Earthquake incidence in Indonesia, 1985-2023, mag > 5.5: count per square kilometre by province.

while the explanatory variable is

$$x_i = \text{fault concentration}_i = \frac{\text{area of buffered faults in province}_i}{\text{province area}_i}. \tag{2}$$

Firstly, when excluding the incidence and just modelling counts, where y_i = earthquake count in province$_i$, the Poisson model is of the following form:

$$y_i|\lambda_i \sim \text{Pois}(\lambda_i) \tag{3}$$

with

$$E(y_i|\lambda_i) = \lambda_i. \tag{4}$$

We model

$$log(\lambda_i) = \beta_0 + \beta_1 x_i + \gamma_i. \tag{5}$$

where γ_i is a term with a correlation structure reflecting a province's location relative to other provinces.

We can describe these relationships by setting up a neighbourhood structure based on queen contiguity where a pair of provinces are considered neighbours if they share at least one point of boundary. This can be modelled as a Markov random field to generate an ICAR model with a spatially varying term. Each of these terms will be correlated with the others according to the neighbourhood structure we have defined.

The Markov random field here follows a multivariate Gaussian distribution. γ_i is a vector of province effects having a distribution with mean **0** and precision **P** where

$[\mathbf{p}]_{ij} = v_i$ if $i = j$ and v_i is the number of adjacent provinces to province i,

$[\mathbf{p}]_{ij} = -1$ if provinces i and j are adjacent, and

$[\mathbf{p}]_{ij} = 0$ otherwise.

A further constraint that $\Sigma_j \gamma_j = 0$ is applied so that the distribution is identifiable.

We now include an offset term (here, area) because we are more interested in modelling the incidence than in the actual count, such that

$$log(\frac{\lambda_i}{\text{area}_i}) = \beta_0 + \beta_1 x_i + \gamma_i \tag{6}$$

which is equivalent to

$$log(\lambda_i) = \beta_0 + \beta_1 x_i + \gamma_i + log(\text{area}_i). \tag{7}$$

We are still modelling $log(\lambda)$ rather than the incidence, but we are adding an offset to adjust for differing areas. Modelling $log(\lambda)$ and adding an offset is equivalent to modelling

incidence, and coefficients can be interpreted that way.

When interpreting the estimated coefficients of the model, it can be useful to look at it in the following form:

$$\lambda_i = e^{\beta_0+\beta_1 x_i+\gamma_i}\text{area}_i. \tag{8}$$

Having described the type of model we wish to implement, we now show how **sfislands** can be used to streamline the process.

5.3 Pre-functions

Such models, however, cannot incorporate locations which have no neighbours. In the case of Indonesia, this is quite problematic. It is composed of many islands. The estimated count of islands according to Andréfouët et al. (2022) is 13,558. While it is not unusual for a country to have a number of often small offshore islands, Indonesia is entirely composed of (at least portions of) an archipelago of islands, so many of these islands or groups of islands are individual provinces in their own right. We might like to hypothesise that just because a province is a disconnected island, this should not mean that it is independent of other nearby provinces in terms of earthquake incidence. A standard first-order queen contiguity structure would mean the exclusion of disconnected units entirely from the model. An alternative strategy of assigning neighbour status based on a distance metric would overcome this, but the threshold size of distance necessary for such a structure might be inappropriately large for the non-islands provinces. Many extra unwanted contiguities could be added when only those related to disconnected units were desired. We would like to use a compromise between these two strategies.

In this case, we use `st_bridges()` for setting up the queen contiguity structure as usual, but with the additional stipulation that unconnected units (provinces which are islands or collections of islands) are considered neighbours to their k nearest provinces. For this example, we have set the value of k to 2. The resulting neighbourhood structure is shown in Figure 11. Note how it can be styled with a combination of internal arguments (size, colour, fill etc.) and additional **ggplot2** layers.

```
# join islands to k=2 nearest neighbours
# various arguments exist for altering colours and sizes
# additional ggplot themes and layers can be added

st_bridges(provinces_df, "province", link_islands_k = 2) |>
  st_quickmap_nb(fillcol = "antiquewhite1",
                 bordercol = "black", bordersize = 0.5,
                 linkcol = "darkblue", linksize = 0.8,
                 pointcol = "red", pointsize = 2) +
  theme(panel.background = element_rect(fill = "#ECF6F7", colour = "black",
                                        linewidth=1.5),
        axis.text = element_blank()) +
  geom_sf(data=nearby_countries_df,
          fill="gray50", linewidth=0.5, colour="black")
```

This neighbourhood structure now has no unconnected provinces so it is suitable for use in an ICAR model. However, if we are not entirely happy with this structure because of some domain knowledge about the inter-relationships between certain island provinces, we might wish to

- add some additional contiguities using `st_force_join_nb()`
- and remove one using `st_force_cut_nb()`.

To cater for the possibility that a modeller might not be familiar with the names of the various geographic units but still wishes to enforce alterations to their relationships, we

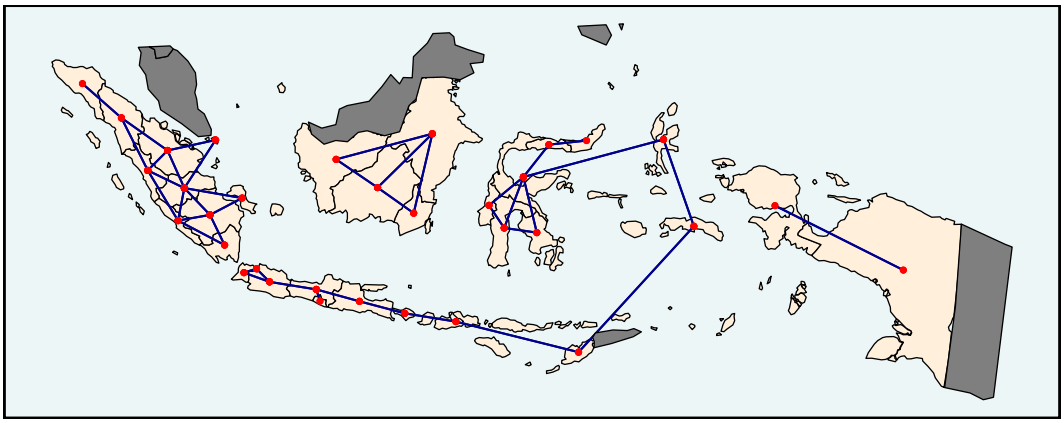

Figure 11: Neighbourhood structure for Indonesian provinces created by `st_bridges()` with `k=2`.

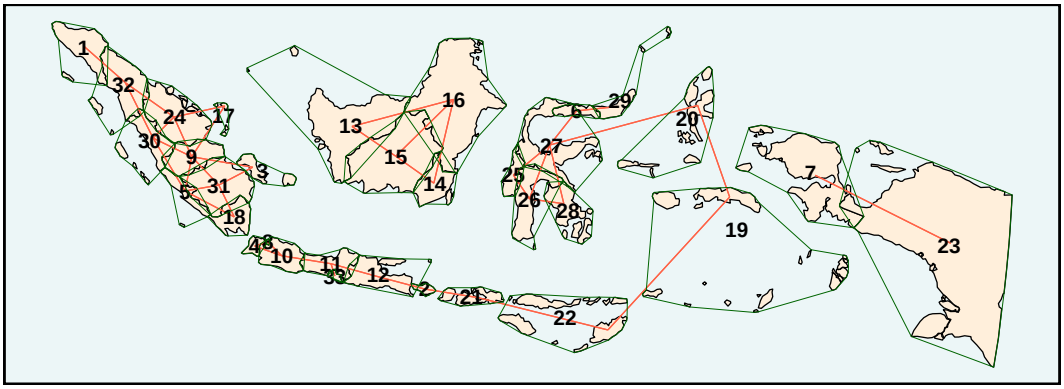

Figure 12: Neighbourhood structure for Indonesian provinces. Viewed with `st_quickmap_nb()`, using the arguments `nodes = 'numeric'` and `concavehull = TRUE`.

can look at a map (Figure 12) where the nodes are shown by index number instead of as points (using the argument `nodes='numeric'`). This makes it easy to cut and join neighbour connectivities as desired. Furthermore, there is an option to show concave hulls drawn around each unit (using `concavehull = TRUE`). This is also shown in Figure 12. These shapes are not used in the assignment of contiguities but it can be useful to see them in a situation such as Indonesia where many individual provinces are actually multipolygons of more than one island. Without them, it is not clear whether an island is a province in its own right, or which group of islands together form one province.

```
# with 'concavehull = TRUE' and 'nodes = "numeric"'

st_bridges(provinces_df, "province", link_islands_k = 2) |>
  st_quickmap_nb(fillcol = "antiquewhite1",
                 bordercol = "black", bordersize = 0.5,
                 linkcol = "tomato", linksize = 0.5,
                 nodes = "numeric",
                 numericcol = "black", numericsize = 6,
                 concavehull = TRUE,
                 hullcol = "darkgreen", hullsize = 0.2) +
  theme(panel.background = element_rect(fill = "#ECF6F7", colour = "black",
                                        linewidth=1.5),
        axis.text = element_blank())
```

Having enforced some adjustments to the neighbourhood structure, outlined in the code below, the new structure can be seen in Figure 13. Edge effects have also been mitigated by imposing additional connections on the two extreme provinces (1 and 23), which would

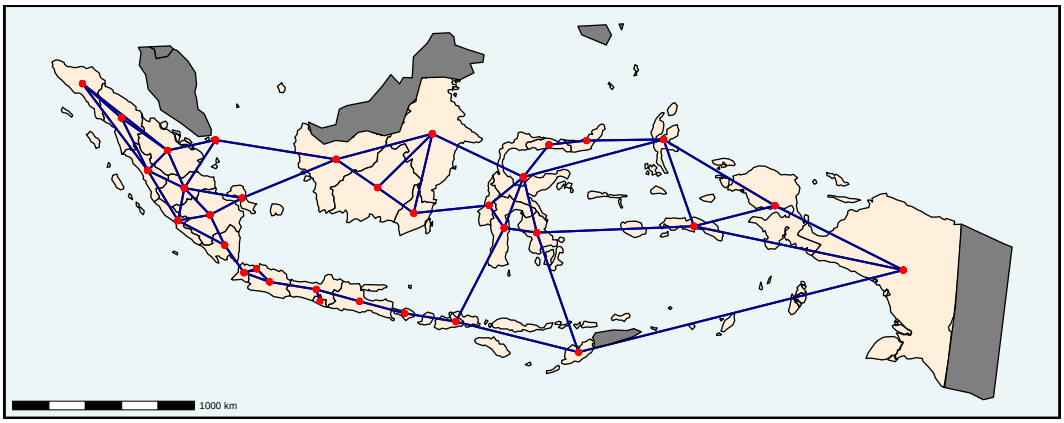

Figure 13: Neighbourhood structure for Indonesian provinces, after alterations using `st_force_join()` and `st_force_cut()`. As many connections are being enforced at once, we can feed these to the function as a data frame rather than using consecutive function calls as before.

otherwise have only one neighbour, so that they now also include their two next closest neighbours.

```
# a series of forced joins and cuts by index number

joins_df <- tribble(
  ~x, ~y,
  1, 24,
  1, 30,
  3, 13,
  13, 17,
  14, 25,
  20, 29,
  19, 23,
  16, 27,
  22, 23,
  7, 19,
  7, 20,
  19, 28,
  4, 18,
  21, 26,
  22, 28
)

st_bridges(provinces_df, "province", link_islands_k = 2) |>
  st_force_join_nb(xy_df = joins_df) |>
  st_force_cut_nb(19,22) |>
  st_quickmap_nb(fillcol = "antiquewhite1",
                 bordercol = "black", bordersize = 0.5,
                 linkcol = "darkblue", linksize = 0.8,
                 pointcol = "red", pointsize = 2) +
  theme(panel.background = element_rect(fill = "#ECF6F7", colour = "black",
                                        linewidth=1.5),
        axis.text = element_blank()) +
  geom_sf(data=nearby_countries_df,
          fill="gray50", linewidth=0.5, colour="black") +
  annotation_scale()
```

5.4 mgcv model

We now create the ICAR model using, in this case, the **mgcv** package. We will be able to use the output of `st_bridges`, which we have named `prep_data`, as both the data source for the model and the neighbourhood structure (by specifying the column `nb` which contains the neighbourhood list).

```
mod_pois_mrf <- gam(damaging_quakes_total ~
                      fault_concentration +
                      s(province, bs='mrf', xt=list(nb=prep_data$nb), k=24) +
                      offset(log(area_province)),
                    data=prep_data, method="REML",family = "poisson")
```

We can see from the summary below that the adjusted R-squared is **0.983** and deviance explained is **93.3%**. The coefficient for `fault_concentration` confirms an expected positive mean global association between earthquake and fault incidence.

```
#>
#> Family: poisson
#> Link function: log
#>
#> Formula:
#> damaging_quakes_total ~ fault_concentration + s(province, bs = "mrf",
#>     xt = list(nb = prep_data$nb), k = 24) + offset(log(area_province))
#>
#> Parametric coefficients:
#>                     Estimate Std. Error z value Pr(>|z|)
#> (Intercept)          -9.5648     0.1744 -54.845  < 2e-16 ***
#> fault_concentration   5.9971     1.9245   3.116  0.00183 **
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Approximate significance of smooth terms:
#>               edf Ref.df Chi.sq p-value
#> s(province) 19.19     23  166.6  <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> R-sq.(adj) =  0.983   Deviance explained = 93.3%
#> -REML = 104.81  Scale est. = 1         n = 33
```

Returning to the initial question, what is the additional risk level of earthquakes in a province, having controlled for the concentration of faults? This can be seen as a measure of the activity level of faults locally and it is spatially smoothed by the autoregressive process. It is represented in the model summary by the component `s(province)`. However, the extraction of individual predictions for this component for each province from the `mgcv` model requires a number of steps. We now demonstrate how these are streamlined into a single function by the **sfislands** package.

5.5 Post-functions

The function `st_augment()` allows us to add the spatially varying predictions from the model as new columns to the original dataframe in a process similar to that of the **broom** package. For instance, we see from the output of the following code chunk that the original dataframe is now augmented with columns called `mrf.smooth.province` and `se.mrf.smooth.province` which show the predictions for the γ_i component and their standard errors. Note that this is how we would expect them to be named, based on the previous

discussion surrounding Table 3. They are positioned immediately before the final `geometry` column of the `sf` dataframe, and after the neighbours list column, `nb`.

```
# column names of augmented dataframe

mod_pois_mrf |>
  st_augment(prep_data) |>
  names() |>
  dput()

#> c("province", "province_id", "S", "M", "L", "XL", "quake_total",
#> "quake_density", "damaging_quakes_total", "damaging_quakes_density",
#> "area_fault_within", "area_province", "fault_concentration",
#> "nb", "mrf.smooth.province", "se.mrf.smooth.province", "geometry"
#> )
```

This output can now be piped into the `st_quickmap_preds()` function to get a quick visualisation of these estimates for γ_i on a map, as shown in Figure 14. Again, note that the title and subtitle of the image are as previously discussed.

```
# st_quickmap_preds() outputs a list of ggplots

plot_mrf <- mod_pois_mrf |>
  st_augment(prep_data) |>
  st_quickmap_preds(scale_low = "darkgreen",
                    scale_mid = "ivory",
                    scale_high = "darkred",
                    scale_midpoint = 0)

# in this case, there is only one plot in the list
# so we call it by index
# it is then supplemented with additional ggplot functions

plot_mrf[[1]] +
  coord_sf(datum=NA, default = TRUE) +
  theme(panel.background = element_rect(fill = "#ECF6F7", colour = "black",
                                        linewidth=1.5),
        axis.text = element_blank()) +
  geom_sf(data=provinces_df, fill=NA, colour="black", linewidth=0.5) +
  geom_sf(data=nearby_countries_df, fill="gray50", colour="black",
          linewidth=0.5) +
  labs(fill="relative\nincidence") +
  annotation_scale() +
  coord_sf(datum=NA, default = TRUE) +
  theme(legend.position = "inside",
        legend.position.inside = c(0.92,0.77),
        legend.box.background = element_rect(colour = "black", linewidth = 1),
        legend.title = element_text())
```

If we wish to apply the inverse link function (the exponential function in the case of this Poisson model) to map these values to a more interpretable scale, this will not be generated by the function `st_quickmap_preds()`. Instead, we must use the augmented dataframe which is produced by `st_augment()` and create the appropriate extra column with the usual **tidyverse** `mutate()` function. This allows us to produce the map in Figure 15. As these coefficients are multiplicatively related to the earthquake incidence, values below 1 imply an earthquake incidence which is lower than expected.

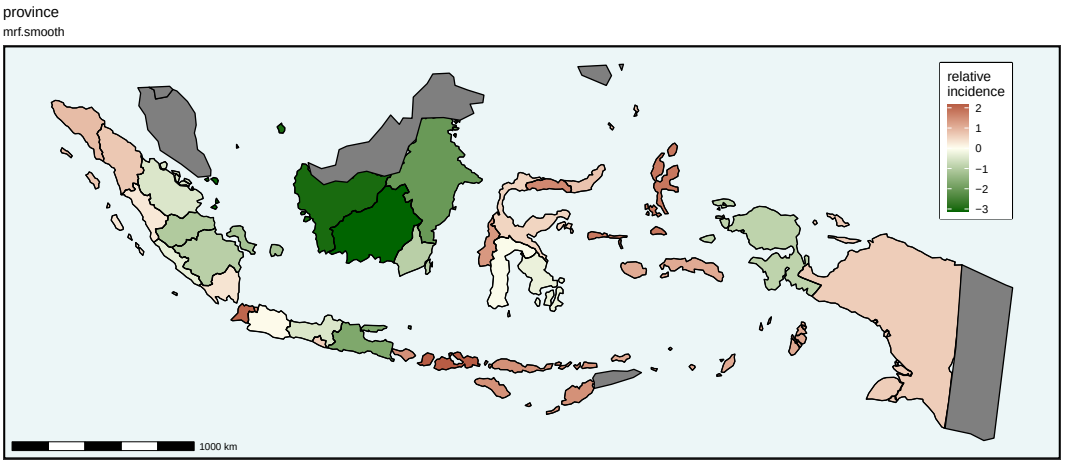

Figure 14: Estimates of γ_i shown as a map using `st_quickmap_preds()`.

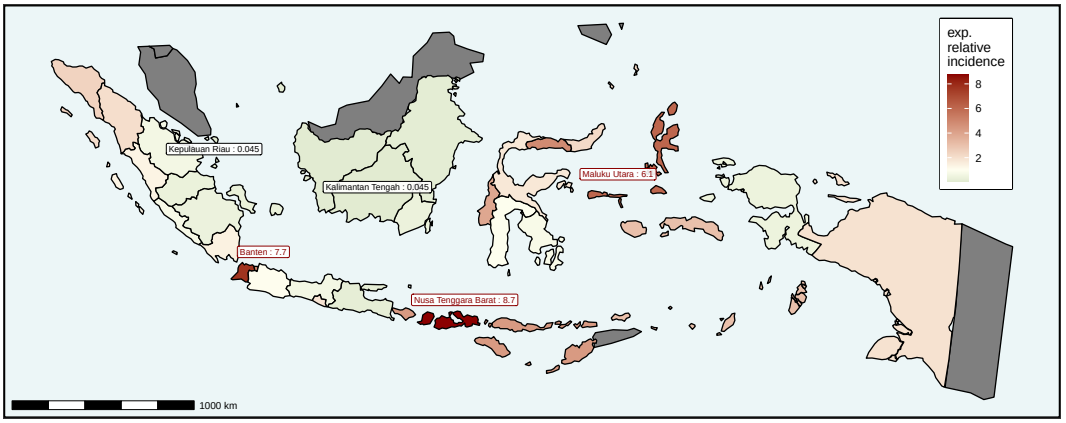

Figure 15: Map showing estimates of $\exp(\gamma_i)$. This is produced by adding an additional column to the dataframe produced by `st_augment()`.

The provinces with the 3 most elevated incidences are labelled in red. We can see that, controlling for the effects of proximity to faults, the province of Nusa Tenggara Barat has 8.7 times the expected incidence, or number of major earthquakes per square kilometre. The two lowest-scoring provinces, labelled in green, have essentially no incidence of earthquake epicentres within their boundaries, controlling for what their proximity to faults alone would suggest.

5.6 Workflow summary

In this example, we have gone through a number of stages carefully, making changes to contiguities that we deemed appropriate as we went. However, in practice, at least in a first iteration, it might not be necessary to go through all of these steps. A rough and ready model, complete with spatially varying coefficients and visual output, can be generated with **sfislands** using nothing more than three or four lines of code, such as the following:

```
# workflow:

# 1. set up neighbourhood structure

prep_data <- st_bridges(provinces_df, "province")

# 2. define model
```

```
mod <- gam(quake_mlxl_total ~
                  fault_concentration +
                  s(province, bs='mrf', xt=list(nb=prep_data$nb), k=22) +
                  offset(log(area_province)),
        data=prep_data, method="REML",family = "poisson")

# 3. augment tidy estimates

tidy_ests <- st_augment(mod, prep_data)

# 4. visualise them

st_quickmap_preds(tidy_ests)
```

6 London (example 2)

The next example looks only at using the *pre-functions* of **sfislands**, but in a situation where the presence of actual *islands* is not the problem we seek to deal with. Consider the wards of London (sourced from the Greater London Authority's London Datastore) and available at https://github.com/horankev/london_liverpool_data. In Figure 16 the `st_bridges()` function is applied to them to construct a queen contiguity neighbourhood structure. As can be seen from the `st_check_islands()` function, this collection of London wards contains no isolated units.

```
st_bridges(london, "GSS_CODE") |>
  st_check_islands()

#> No disconnected units were found in original data

#> [1] 0
```

The `st_quickmap_nb()` function gives an immediate visual representation of the structure. [2] Because this map is created using **ggplot2**, it can be easily supplemented by adding a layer showing the course of the river Thames which is also visible in Figure 16.

```
# same as sfdep:st_contiguity() as there are no islands
# an extra layer for the river Thames

st_bridges(london, "GSS_CODE") |>
  st_quickmap_nb() +
  geom_sf(data=thames, colour="blue", linewidth=1.5) +
  theme(panel.background = element_rect(fill = "#F6F3E9", colour = "black",
                                        linewidth=1.5))
```

When a study area has a river running through it, problems can arise with constructing appropriate neighbourhood structures. Depending on how the geometries are defined, the presence of a river can cause problems in two ways. In one situation, the river could be expressed as a polygon in its own right meaning that, using the condition of queen contiguity, it severs any potential contiguity between units on either side of its banks. In this situation, no spatial units will be neighbours with the units directly across the river from them. At the other extreme, if the river is not included as a geometry (as is the case here) all units on opposing banks are automatically considered neighbours.

Depending on the presence of river crossings, two areas which are physically quite close but on opposing banks might be very distinct. If there is no means of crossing the river

[2] `st_quickmap_nb()` can also be used to visualise any contiguity structure created by `spdep` or `sfdep` as long as that structure is included in an `sf` dataframe as a column named `nb`.

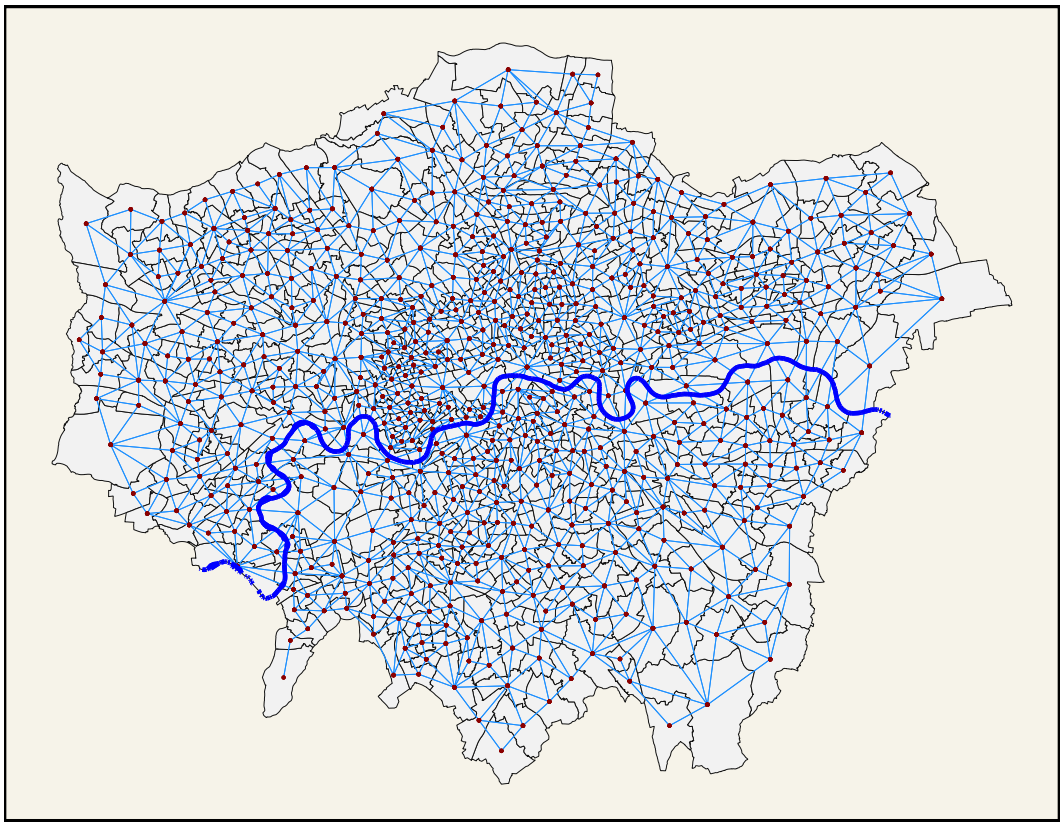

Figure 16: Wards of Greater London. Queen contiguity.

within a reasonable distance, somebody living on the banks of a river might be more likely to go about their life primarily on their side of the river, despite the short distance as the crow flies of facilities on the other side. This could be relevant in terms of, say, modelling of house prices where we might want to incorporate issues such as local amenities into a neighbourhood structure.

sfislands provides convenient functions for this sort of situation. Let us start by restricting our wards of interest to just those which are on either side of the river Thames. Figure 17 shows the resultant contiguities when the river is ignored.

```
# which wards are alongside the river

riverside <- thames |> st_intersects(london) |> unlist() |> unique()

# only map these wards

st_bridges(london[riverside,],"NAME") |>
  st_quickmap_nb(linksize = 0.5) +
  geom_sf(data=thames, colour="blue", linewidth=1.5) +
  annotation_scale(location="br") +
  coord_sf(datum=NA) +
  theme(panel.background = element_rect(fill = "#F6F3E9", colour = "black",
                                        linewidth=1.5))
```

In order to take account of actual connectivity, we can add a layer showing the road and pedestrian bridges or tunnels. Details of these were sourced from the Wikipedia (2024) article titled *"List of crossings of the River Thames"*. In Figure 18, we have also drawn a 1 kilometre buffer around each crossing. This was chosen as an arbitrary measure of what might be considered a "reasonable" distance within which to consider opposing banks as being connected. The vast majority of units on opposing banks have access to a river

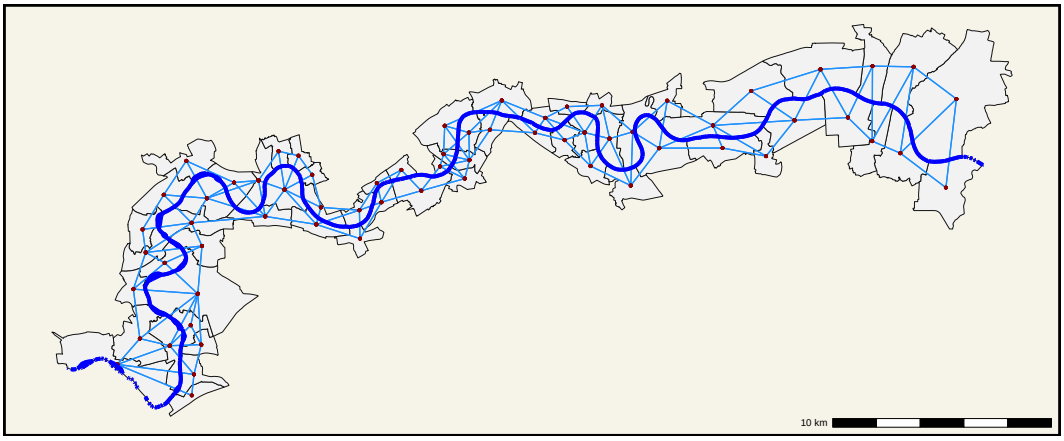

Figure 17: Riverside wards of Greater London. Queen contiguity disregarding river.

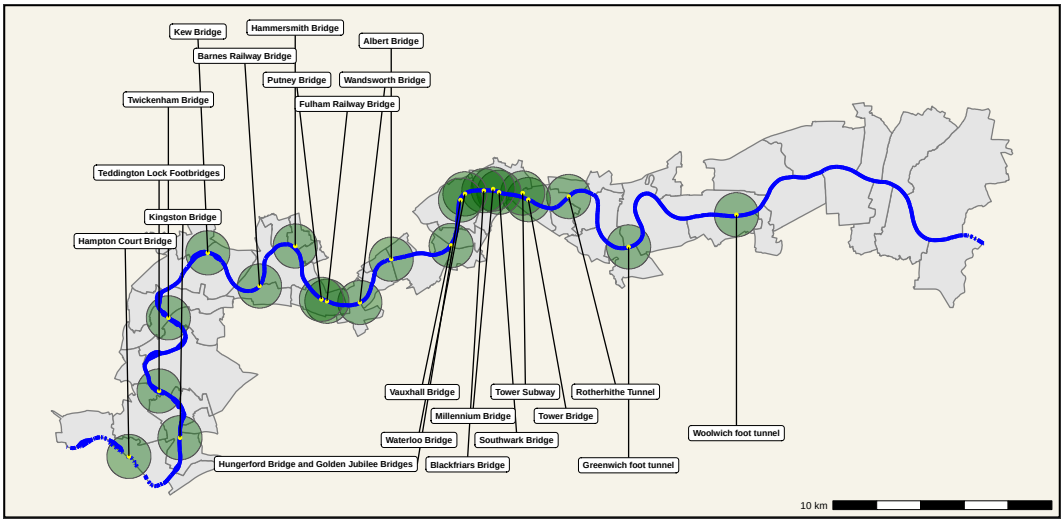

Figure 18: Riverside wards of Greater London. Road and pedestrian crossing and tunnels surrounded by 1 kilometre buffer shaded green.

crossing within this threshold and thus should be considered as neighbours. Only the extreme eastern units and one to the south west should not have a connection across the river according to this criterion.

In order to identify the changes we wish to make, we use the `nodes = "numeric"` argument in `st_quickmap_nb()`. Now we can identify each unit by its position in the contiguity structure. Here we have shaded in pink the units which are not within 1 kilometre of a river crossing (see Figure 19).

```
# with 'nodes = "numeric"'

st_bridges(london[riverside,],"NAME") |>
  st_quickmap_nb(nodes = "numeric", numericsize = 4, linksize = 0.5) +
  geom_sf(data=no_touch_buffer, fill="pink", alpha=0.3) +
  geom_sf(data=crossings_roadped |> st_buffer(1000),
          fill="darkgreen", alpha=0.3) +
  geom_sf(data=thames, colour="blue", linewidth=1.5) +
  geom_sf(data=crossings_roadped, size=1, colour="yellow") +
  annotation_scale(location="br") +
  coord_sf(datum=NA) +
  theme(panel.background = element_rect(fill = "#F6F3E9", colour = "black",
                                        linewidth=1.5))
```

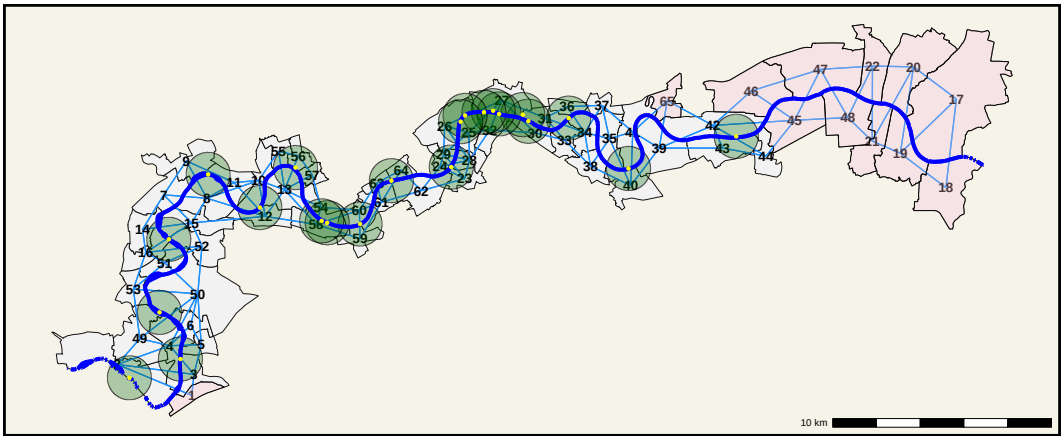

Figure 19: Riverside wards of Greater London. Index number for each ward shown at centroid. Wards which are not within 1 kilometre of a crossing are shaded pink.

This allows us to easily cut the ties across the river for these units by using the function `st_force_cut_nb()`. [3] Having made these adjustments, `st_quickmap_nb()` now shows a connectivity structure (Figure 20) which reflects our hypothesis of how influence should extend across the river in the presence or absence of crossings.

This example shows that the pre-functions of **sfislands** have uses for situations which do not involve islands. They can be used to apply domain knowledge to easily design the most appropriate neighbourhood structure.

```
# enforce cuts for the links where there is no crossing

cut_df <- tribble(
  ~x, ~y,
  18, 17,
  19, 17,
  19, 20,
  20, 21,
  21, 22,
  47, 48,
  45, 46,
  45, 47,
  39, 65,
  1, 2
)
st_bridges(london[riverside,], "NAME") |>
  st_force_cut_nb(xy_df = cut_df) |>
  st_quickmap_nb(bordercol = "black", bordersize = 0.5, linksize = 0.5) +
  geom_sf(data=no_touch_buffer, fill = "pink", alpha = 0.3) +
  geom_sf(data=crossings_roadped |> st_buffer(1000),
          fill= "darkgreen", alpha = 0.3) +
  geom_sf(data=thames, colour = "blue", linewidth = 1.5) +
  annotation_scale(location = "br") +
  coord_sf(datum=NA) +
  theme(panel.background = element_rect(fill = "#F6F3E9", colour = "black",
                                        linewidth = 1.5))
```

[3] While we are using the index of the units in this example, the function also accepts names as arguments which may be more convenient in some circumstances.

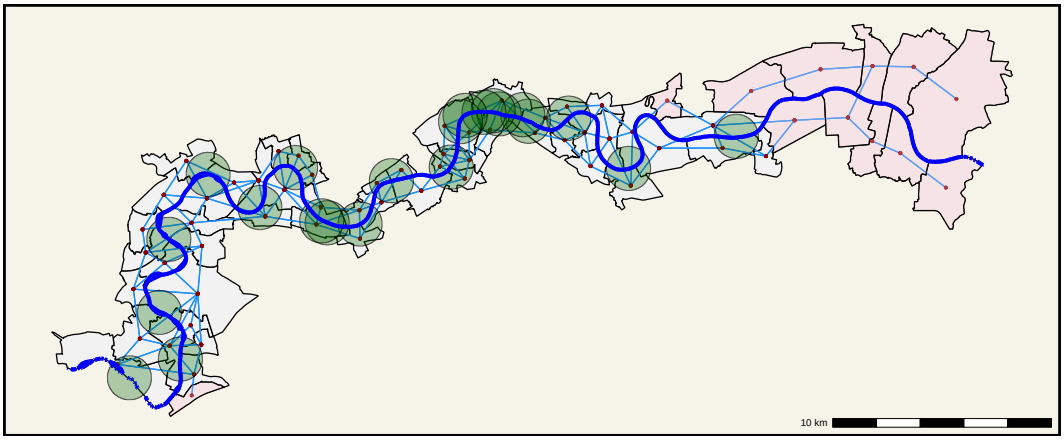

Figure 20: Riverside wards of Greater London. Contiguities across the river have been cut for the pink wards.

7 Summary

These examples have shown the varying scenarios in which **sfislands** can be useful. It aims to contribute to spatial modelling by making an awkward area less awkward. Rather than having a default attitude of ignoring islands when building neighbourhood structures based on contiguity, it offers a convenient system for enforcing linkages if that is deemed to be the most appropriate course of action. Even when no islands are present, it provides a simple procedure for tailoring a neighbourhood structure with bespoke contiguities to match a given hypothesis. It also provides helper functions to use these structures in spatial regression models, notably those built with **mgcv**, which streamline the human effort necessary to examine the estimates. In future, compatibility with other modelling packages can be added to broaden the package's capabilities.

8 Acknowledgements

This publication has emanated from research conducted with the financial support of Taighde Éireann – Research Ireland under Grant number 18/CRT/6049. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

References

S. Andréfouët, M. Paul, and A. R. Farhan. Indonesia's 13558 islands: A new census from space and a first step towards a One Map for Small Islands Policy. *Marine Policy*, 135: 104848, 2022. URL http://dx.doi.org/10.1016/j.marpol.2021.104848. [p95]

H. Bakka, H. Rue, G.-A. Fuglstad, A. Riebler, D. Bolin, E. Krainski, D. Simpson, and F. Lindgren. Spatial modelling with R-INLA: A review, 2018. URL https://arxiv.org/abs/1802.06350. [p86]

D. Bates, M. Mächler, B. Bolker, and S. Walker. Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, 67(1):1–48, 2015. URL https://doi.org/10.18637/jss.v067.i01. [p91]

J. Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 36(2):192–225, 1974. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1974.tb00999.x. [p90]

R. S. Bivand. R packages for analyzing spatial data: A comparative case study with areal data. *Geographical Analysis*, 54(3):488–518, 2022. URL https://doi.org/10.1111/gean.12319. [p84]

R. S. Bivand and B. A. Portnov. Exploring spatial data analysis techniques using R: The case of observations with no neighbors. In L. Anselin, R. J. G. M. Florax, and S. J. Rey, editors, *Advances in Spatial Econometrics*, Advances in Spatial Science, chapter 6, pages 121–142. Springer, 2004. URL https://ideas.repec.org/h/spr/adspcp/978-3-662-05617-2_6.html. [p85]

Á. Briz-Redón, A. Iftimi, J. F. Correcher, J. De Andrés, M. Lozano, and C. Romero-García. A comparison of multiple neighborhood matrix specifications for spatio-temporal model fitting: A case study on COVID-19 data. *Stochastic Environmental Research and Risk Assessment*, 36(1):271–282, 2021. URL http://dx.doi.org/10.1007/s00477-021-02077-y. [p84]

P.-C. Bürkner. brms: An R package for Bayesian multilevel models using Stan. *Journal of Statistical Software*, 80(1):1–28, 2017. URL https://doi.org/10.18637/jss.v080.i01. [p86]

E. W. Duncan, N. M. White, and K. Mengersen. Spatial smoothing in Bayesian models: A comparison of weights matrix specifications and their impact on inference. *International Journal of Health Geographics*, 16(1), 2017. URL http://dx.doi.org/10.1186/s12942-017-0120-x. [p84]

A. Earnest, G. Morgan, K. Mengersen, L. Ryan, R. Summerhayes, and J. Beard. Evaluating the effect of neighbourhood weight matrices on smoothing properties of conditional autoregressive (CAR) models. *International Journal of Health Geographics*, 6(1):54, 2007. URL http://dx.doi.org/10.1186/1476-072X-6-54. [p84]

A. Freni-Sterrantino, M. Ventrucci, and H. Rue. A note on intrinsic conditional autoregressive models for disconnected graphs. *Spatial and Spatio-temporal Epidemiology*, 26:25–34, 2018. URL http://dx.doi.org/10.1016/j.sste.2018.04.002. [p85]

K. Horan, K. Domijan, and C. Brunsdon. *sfislands: Streamlines the process of fitting areal spatial models*, 2024. URL https://horankev.github.io/sfislands/. R package version 1.1.2. [p85]

A. Kassambara. *ggpubr: 'ggplot2'-based publication ready plots*, 2023. URL https://CRAN.R-project.org/package=ggpubr. R package version 0.6.0. [p91]

J. Parry and D. H. Locke. *sfdep: Spatial dependence for simple features*, 2024. URL https://sfdep.josiahparry.com. R package version 0.2.4. [p84]

E. Pebesma. Simple features for R: Standardized support for spatial vector data. *The R Journal*, 10(1):439–446, 2018. URL https://doi.org/10.32614/RJ-2018-009. [p84]

J. Pinheiro, D. Bates, and R Core Team. *nlme: Linear and nonlinear mixed effects models*, 2023. URL https://CRAN.R-project.org/package=nlme. R package version 3.1-164. [p91]

D. Robinson, A. Hayes, and S. Couch. broom: Convert statistical objects into tidy tibbles. 2023. URL https://CRAN.R-project.org/package=broom. [p90]

Stan Development Team. RStan: the R interface to Stan, 2020. URL http://mc-stan.org/. [p88]

W. R. Tobler. A computer movie simulating urban growth in the Detroit region. *Economic Geography*, 46(sup1):234–240, 1970. URL https://www.tandfonline.com/doi/abs/10.2307/143141. Publisher: Routledge. [p84]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. URL https://ggplot2.tidyverse.org. [p85]

Wikipedia. List of crossings of the River Thames — Wikipedia, the free encyclopedia, 2024. URL http://en.wikipedia.org/w/index.php?title=List%20of%20crossings%20of%20the%20River%20Thames&oldid=1184426738. Accessed 2024-03-15. [p102]

S. N. Wood. Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. 73:3–36, 2011. URL https://CRAN.R-project.org/web/packages/mgcv/index.html. [p85]

Kevin Horan
Hamilton Institute, Maynooth University
Maynooth
Co. Kildare, Ireland
https://github.com/horankev
ORCiD: 0009-0003-9378-0084
kevin.horan.2021@mumail.ie

Katarina Domijan
Department of Mathematics and Statistics, Maynooth University
Maynooth
Co. Kildare, Ireland
ORCiD: 0000-0002-4268-2236
katarina.domijan@mu.ie

Chris Brunsdon
National Centre for Geocomputation, Maynooth University
Maynooth
Co. Kildare, Ireland
ORCiD: 0000-0003-4254-1780
Christopher.Brunsdon@mu.ie