Geographic operations and meshes

J Matthiopoulos

2022-06-16

Setting up

The libraries needed

```
library(sf)
library(raster)
library(rmapshaper)
library(tidyr)
library(ggplot2)
library(inlabru)
library(INLA)
library(sp)
```

Importing the data

The necessary geographic data are in package `raster`. The `getData` command fetches geographic data for anywhere in the world. Data are read from files that are first downloaded if necessary. The data names are as follows:

- `alt`: Altitude (elevation); the data were aggregated from SRTM 90 m resolution data between -60 and 60 latitude.
- `GADM`: A database of global administrative boundaries.
- `worldclim`: A database of global interpolated climate data.
- `SRTM`: The hole-filled CGIAR-SRTM digital elevation (90 m resolution).
- `countries`: Polygons for all countries at a higher resolution than the `wrld_simpl` data in the maptools package.

Note that the `terra` package, that is compatible with the new changes in GDAL and PROJ, has now been created as a replacement for the raster library. I will need to explore equivalent ways of obtaining these data via `terra`.

```
uk_mask <- getData('GADM', country='GBR', level=1)
uk_alt <- getData("alt", country='GBR', mask=TRUE)
England <- uk_mask[uk_mask$NAME_1 == "England",]
class(England)
```

```
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

```
class(uk_alt)
```

```
## [1] "RasterLayer"
## attr(,"package")
## [1] "raster"
```

Any shape file in my system can be read directly using the `st_read()` (an `sf` command for reading simple features from files or databases, or retrieving layer names and their geometry type(s)). In this example, the England SpatialPolygonsDataFrame will be converted to a simple feature object that we can manipulate and visualize within the tidyverse DSLs. The CRS for spatial objects of class `sf` or `stars` can be retrieved using the `st_crs` function, or be set or changed via `st_set_crs` using pipeline command (notice that simply replacing the CRS does not re-project the data, we should use `st_transform` for this).

In the code below `st_transform()` (Equivalent to `spTransform()`) is used to project the original CRS using the EPSG code for the BNG and change the units from meters to km by accessing the PROJ.4 string attribute.

```
# build an sf object
England_sf = st_as_sf(England) %>%  st_transform(crs = 27700)
England_sf = st_transform(England_sf, gsub("units=m","units=km",st_crs(England_sf)$proj4string))
```

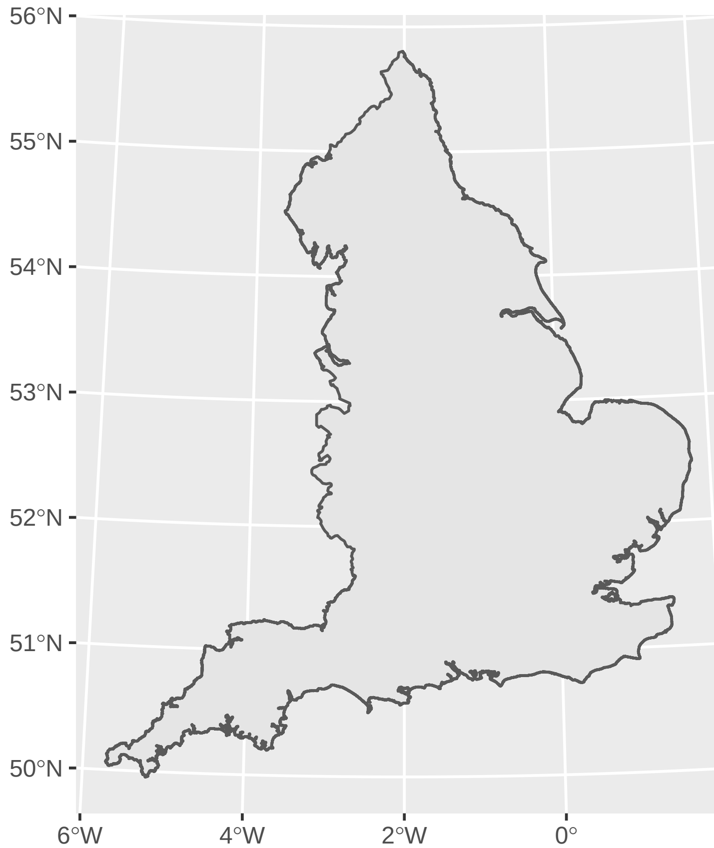
For simplicity, remove all of the smaller detached islands by using the `ms_filter_islands` function in the `rmapshaper` package [1]

```
# Remove detached polygons with an area less than 2000 km
England_mainland <- ms_filter_islands(England_sf, min_area = 2000)
England_mainland
```

```
## Simple feature collection with 1 feature and 10 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: 134.0774 ymin: 11.09554 xmax: 655.6956 ymax: 656.7911
## CRS:           +proj=tmerc +lat_0=49 +lon_0=-2 +k=0.9996012717 +x_0=400000 +y_0=-100000 +ellps=airy +
##   GID_0         NAME_0   GID_1   NAME_1 VARNAME_1 NL_NAME_1
## 1   GBR United Kingdom GBR.1_1 England      <NA>      <NA>
##                              TYPE_1 ENGTYPE_1 CC_1 HASC_1
## 1 Home Nation|Constituent Country   Kingdom <NA>   <NA>
##                           geometry
## 1 POLYGON ((564.9785 102.5189...
```

```
#Plot the resulting simple feature object using geom_sf within ggplot.
ggplot()+
  geom_sf(data=England_mainland)
```

[1] This package fully supports sf or sfc polygons object as well. It is used to edit and simplify `geojson`, `Spatial`, and `sf` objects. Performs topologically-aware polygon simplification, as well as other operations such as clipping, erasing, dissolving, and converting 'multi-part' to 'single-part' geometries. It relies on the `geojsonio` package for working with `geojson` objects, the `sf` package for working with `sf` objects, and the `sp` and `rgdal` packages for working with `Spatial` objects.



We can query information on the CRS and projection as follows:

```
#  retrieve the PROJ.4 attribute
st_crs(England_mainland)$proj4string
```

```
## [1] "+proj=tmerc +lat_0=49 +lon_0=-2 +k=0.9996012717 +x_0=400000 +y_0=-100000 +ellps=airy +units=km +
```

```
# check whether  longitude-latitude projection is still being applied
st_is_longlat(England_mainland)
```

```
## [1] FALSE
```

```
# Check the spatial units of our projection
st_crs(England_mainland)$units
```

```
## [1] "km"
```

Two-dimensional mesh for spatial problems

There are several arguments that can be used to build the mesh. This vignette will only cover a two-dimensional mesh construction using the inla.mesh.2d. function. However, a one-dimensional mesh specification can be created using the inla.mesh.1d function . The arguments for a two-dimensional mesh construction are the following:

```
args(inla.mesh.2d)
```

```
## function (loc = NULL, loc.domain = NULL, offset = NULL, n = NULL,
##     boundary = NULL, interior = NULL, max.edge = NULL, min.angle = NULL,
##     cutoff = 1e-12, max.n.strict = NULL, max.n = NULL, plot.delay = NULL,
##     crs = NULL)
## NULL
```

First, some reference about the study region is needed, which can be provided by either:

- The location of points, supplied on the `loc` argument [2].
- The domain extent which can be supplied as a single polygon on the `loc.domain` argument.
- A boundary of the region defined by a set of polygons (e.g a polygon defining the coastline of the study) supplied on the `boundary` argument.

Note that if either (1) the location of points or (2) the domain extent are specified, the mesh will be constructed based on a convex hull (a polygon of triangles out of the domain area). Alternatively, it possible to include a non-convex hull as a boundary in the mesh construction instead of the location or loc.domain arguments. This will result in the triangulation to be constrained by the boundary. A non-convex hull mesh can also be created by building a boundary for the points using the inla.nonconvex.hull() function. Finally, the other compulsory argument that needs to be specified is the max.edge which determines the largest allowed triangle length (the lower the value for max.edge the higher the resolution). The value supplied to this argument can be either a scalar, in which case the value controls the triangle edge lengths in the inner domain, or a length two vector that controls the edge lengths in the inner domain and in the outer extension respectively. Notice that The value (or values) passed to the max.edge function must be on the same scale unit as the coordinates. To illustrate the different options when building a mesh I will use the number of dragonflies records on the British Dragonfly Society Recording Scheme (2020) in the west coast of England.

The final step is to transform sf-class objects to a sp spatial-structure. The we can use this object to produce the mesh and fit our model.

```
# Build the mesh

England_mainland_sp <- as(England_mainland, "Spatial")

england.bdry <- England_mainland_sp %>% inla.sp2segment()

max.edge = 20
mesh = inla.mesh.2d(boundary = england.bdry,
                    crs = st_crs(England_mainland),offset = 50,
                    max.edge = c(1.5, 2.5) * max.edge,
                    cutoff = 15)

plot(mesh)
```

[2] Matrix of point locations to be used as initial triangulation nodes. Can alternatively be a `SpatialPoints` or `SpatialPointsDataFrame` object.

Constrained refined Delaunay triangulation