Package ‘osmgeosample’

February 7, 2021

Title Construction of Geostatistical Sampling Designs with OSM Data

Version 0.1

Maintainer Henry Crosby <henry.crosby@warwick.ac.uk>

Imports dplyr, geoR, graphics, mapview, nngeo, osmdata, pdist, processx, rgdal, sp, splancs, stats, qpdf, Rcpp, tibble, sf, methods

Depends R (>= 3.0.0)

Suggests PrevMap, rmarkdown, markdown, leafem, viridisLite,raster, knitr, testthat

Description The utilisation of the functionality provided by the 'OSMData' and 'geosample' to allow users to create spatially continuous or discrete random samples from a pre-defined spatial border using OSM data. Reference: Joao Porto de Albuquerque, Godwin Yeboah, Vangelis Pitidis, Philipp Ulbrich (2019) <doi:10.13140/RG.2.2.13710.20804>.

License GPL (>= 2)

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

VignetteBuilder knitr

NeedsCompilation yes

LinkingTo Rcpp

Author Henry Crosby [cre],
    Godwin Yeboah [aut],
    Joao Porto De Albuquerque [aut]

Repository CRAN

Date/Publication 2021-02-07 10:40:02 UTC

R topics documented:

  osm_contin_inhibit_sample ........................................... 2
  osm_discrete_inhibit_sample ......................................... 5
  osm_random_sample .................................................... 9
osm_contin_inhibit_sample

**Description**

Draws a spatially continuous sample of locations within a polygonal sampling region according to an **inhibitory plus close pairs** specification. The region can be defined using OSM data or a user defined polygon.

**Usage**

```r
osm_contin_inhibit_sample(
  bounding_geom = NULL,
  boundary = 0,
  buff_dist = 0,
  buff_epsg = 4326,
  sample_size,
  plotit = TRUE,
  plotit_leaflet = TRUE,
  delta,
  delta.fix = FALSE,
  k = 0,
  rho = NULL,
  ntries = 10000
)
```

**Arguments**

- `bounding_geom` a sf or sp object (with $N \geq \text{size}$) where each line corresponds to one spatial location. It should contain values of 2D coordinates, data and, optionally, covariate(s) value(s) at the locations. This argument must be provided when sampling from a 'discrete' set of points, see 'type' below for details.

- `boundary` a categorical variable to determine whether the exact boundary provided (boundary = 0), the bounding box (boundary = 1) or a buffer around the boundary (boundary = 2) is used for sampling. Default is boundary = 0.

- `buff_dist` if boundary = 2 then this value determines the size of the buffer by distance. The default is buff_dist is NULL).

- `buff_epsg` if boundary = 2 then this value determines the local geographic grid reference so that the buffer can be calculated in meters. The default is buff_epsg = 4326 which will use decimal degrees instead of meters. As an example, 27700 relates to the British National Grid.

- `sample_size` a non-negative integer giving the total number of locations to be sampled.

- `plotit` 'logical' specifying if graphical output is required. Default is plotit = TRUE.
plotit_leaflet  'logical' specifying if leaflet (html) graphical output is required. This is prioritised over plotit if both are selected. Default is plotit_leaflet = TRUE.

delta  minimum permissible distance between any two locations in preliminary sample. This can be allowed to vary with the number of 'close pairs' if a simple inhibitory design is compared to one of the inhibitory plus close pairs design.

delta.fix  'logical' specifies whether delta is fixed or allowed to vary with number of close pairs k. Default is delta.fix = FALSE.

k  number of locations in preliminary sample to be replaced by near neighbours of other preliminary sample locations to form close pairs (integer between 0 and size/2). A simple inhibitory design is generated when k = 0.

rho  maximum distance between the two locations in a 'close-pair'.

ntries  number of rejected proposals after which the algorithm will terminate.

Details

To draw a simple inhibitory (SI) sample of size n from a spatially continuous region A, with the property that the distance between any two sampled locations is at least delta, the following algorithm is used.

- Step 1. Set i = 1 and generate a point x_1 uniformly distributed on D.
- Step 2. Generate a point x uniformly distributed on D and calculate the minimum, d_{min}, of the distances from x_i to all x_j : j ≤ i.
- Step 3. If d_{min} ≥ δ, increase i by 1, set x_i = x and return to step 2 if i ≤ n, otherwise stop;
- Step 4. If d_{min} < δ, return to step 2 without increasing i.

Sampling close pairs of points.

For some purposes, it is desirable that a spatial sampling scheme include pairs of closely spaced points, resulting in an inhibitory plus close pairs (ICP) design. In this case, the above algorithm requires the following additional steps to be taken. Let k be the required number of close pairs. Choose a value rho such that a close pair of points will be a pair of points separated by a distance of at most rho.

- Step 5. Set j = 1 and draw a random sample of size 2 from integers 1, 2, ..., n, say (i_1, i_2);
- Step 6. Replace x_{i_1} by x_{i_2} + u, where u is uniformly distributed on the disc with centre x_{i_2} and radius rho, increase i by 1 and return to step 5 if i ≤ k, otherwise stop.

When comparing a SI design to one of the ICP designs, the inhibitory components should have the same degree of spatial regularity. This requires δ to become a function of k namely

\[ \delta_k = \delta_0 \sqrt{\frac{n}{(n-k)}} \]

with \( \delta_0 \) held fixed.
osm_contin_inhibit_sample

Value

a list with the following four components:

size: the total number of sampled locations.
delta: the value of \( \delta \) after taking into account the number of close pairs \( k \). If delta.fix = TRUE, this will be \( \delta \) input by the user.
\( k \): the number of close pairs included in the sample (for inhibitory plus close pairs design).
sample.locs: a sf or sp object containing coordinates of dimension \( n \) by 2 containing the sampled locations.

Note

If 'delta' is set to 0, a completely random sample is generated. In this case, 'close pairs' are not permitted and rho is irrelevant.

Author(s)

Henry J. Crosby <henry.crosby@warwick.ac.uk>
Godwin Yeboah <godwin.yeboah@warwick.ac.uk>
J. Porto De Albuquerque <J.Porto@warwick.ac.uk>

References


See Also

osm_random_sample and osm_discrete_inhibit_sample

Examples

```r
## Not run:
library(sp)
bounding_geom<- SpatialPolygonsDataFrame(SpatialPolygons(list(Polygons(list(Polygon(
   cbind(
     c(3.888959,3.888744,3.888585,3.888355,3.888355,3.887893,3.887504,
     3.886955,3.886565,3.886303,3.886159,3.885650,3.885650,
     3.885595,3.885404,3.885444,3.885897,3.88692,3.887241,
     3.888068,3.888323,3.888697,3.889150,3.889548,3.88990,
     3.890184,3.890828,3.891258,3.891807,3.892061,3.892292,
     3.892689,3.893294,3.893008,3.89367,3.888959),
     c(7.379483,7.379785,7.380024,7.380294,7.380294,7.380986,
     7.381448,7.381861,7.382243,7.382474,7.383277,7.383468,
     7.383890,7.384263,7.384669,7.385258,7.385313,7.385194,)
```
osm_discrete_inhibit_sample

7.384868, 7.384900, 7.385051, 7.385067, 7.384955, 7.384749,
7.384526, 7.384120, 7.384090, 7.384080, 7.384430, 7.384478,
7.384629, 7.384772, 7.383269, 7.380963, 7.379483)), ID=1)),

data.frame( ID=1))
proj4string(bounding_geom) <- CRS('+proj=longlat +datum=WGS84')

set.seed(15892)

osm_contin_inhibit_sample(bounding_geom = bounding_geom, boundary = 0,
buff_dist=0,
buff_epsg = NULL, sample_size = 50, plotit = TRUE, plotit_leaflet = TRUE,
delta=50, delta.fix = FALSE,k=7,rho=1, ntries = 10)

## End(Not run)

---

osm_discrete_inhibit_sample

*OSM discrete Inhibitory sampling.*

**Description**

Draw a spatially discrete sample from a specified set of OSM spatial locations within a polygonal sampling region according to an 'inhibitory plus close pairs' specification.

**Usage**

```r
osm_discrete_inhibit_sample(
  bounding_geom = NULL,
  key = NULL,
  value = NULL,
  data_return = c("osm_polygons", "osm_points", "osm_multipolygons", "multilines", "lines"),
  boundary = 0,
  buff_dist = 0,
  buff_epsg = 4326,
  join_type = "within",
  sample_size,
  plotit = TRUE,
  plotit_leaflet = TRUE,
  delta,
  delta.fix = FALSE,
  k = 0,
  cp.criterion = NULL,
  zeta,
  ntries = 10000,
  boundary_or_feature = "boundary",
  join_features_to_osm = FALSE,
  feature_geom = NULL
)
```

Arguments

bounding_geom A sf or sp with each line corresponding to one spatial location. It should contain values of 2D coordinates. This argument must be provided when sampling from a 'discrete' set of locations defined in OSM.

key A feature key as defined in OSM. An example is 'building'.

value A value for a feature key (key); can be negated with an initial exclamation mark, value = '!this', and can also be a vector, value = c ('this', 'that'). More details at [https://wiki.openstreetmap.org/wiki/Map_Features](https://wiki.openstreetmap.org/wiki/Map_Features).

data_return A list which specifies what data types (as specified in OSM) you want returned. More than one can be selected. The options are 'osm_polygons', 'osm_points', 'osm_multipolygons', 'osm_multilines', 'osm_lines'.

boundary A categorical variable to determine whether the exact boundary (boundary = 0), the bounding box (boundary = 1) or a buffer around the boundary (boundary = 2) is used for sampling. The default is boundary = 0.

buff_dist If boundary = 2 then this value determines the size of the buffer by distance. The default is buff_dist is NULL.

buff_epsg If boundary = 2 then this value determines the local geographic grid reference so that the buffer can be calculated in meters. The default is buff_epsg = 4326 which will use decimal degrees instead of meters. As an example, 27700 relates to the British National Grid.

join_type A text value to determine how to spatially join all features with the boundary. The options are 'within' or 'intersect'.

sample_size A non-negative integer giving the total number of locations to be sampled.

plotit A 'logical' input specifying if a graphical output is required. Default is plotit = TRUE.

plotit_leaflet A 'logical' input specifying if leaflet (html) graphical output is required. This is prioritised over plotit if both are selected. Default is plotit_leaflet = TRUE.

delta The minimum permissible distance between any two locations in preliminary sample. This can be allowed to vary with number of 'close pairs' if a simple inhibitory design is compared to one of the inhibitory plus close pairs design.

delta.fix A 'logical' input which specifies whether 'delta' is fixed or allowed to vary with number of close pairs k. Default is delta.fix = FALSE.

k The number of close-pair locations in the sample. It must be an integer between 0 and size/2.

cp.criterion The criterion for choosing close pairs k. The 'cp.zeta' criterion chooses locations not included in the initial sample, from the uniform distribution of a disk with radius 'zeta' (NB: zeta argument must be provided for this criterion). The 'cp.neighb' criterion chooses nearest neighbours amongst locations not included in the initial sample ('zeta' becomes trivial for 'cp.neighb' criterion).

zeta The maximum permissible distance (radius of a disk with center $x^{*}_j, j = 1, \ldots, k$) within which a close-pair point is placed. See Details.
**osm_discrete_inhibit_sample**

- **ntrials**: The number of rejected proposals after which the algorithm terminates.

- **boundary_or_feature**: Specifies whether the user inputs a boundary or a set of user-inputted features. For example, if the user selects 'boundary', they can provide a spatial data frame or OSM locality which will query the osm features within that boundary or locality. If the user selects 'feature' then they can provide a data frame of features that they want to sample.

- **join_features_to_osm**: A TRUE or FALSE variable which allows the user to specify whether they want their feature geom to be spatially joined to OSM features. The output sampling data frame will have an additional column showing the joined OSM id.

- **feature_geom**: Is a user inputted data frame of features that are required to be sampled.

**Details**

To draw a sample of size \( n \) from a population of spatial locations \( X_i : i = 1, \ldots, N \), with the property that the distance between any two sampled locations is at least \( \delta \), the function implements the following algorithm.

1. Draw an initial sample of size \( n \) completely at random and call this \( x_i : i = 1, \ldots, n \).
2. Set \( i = 1 \).
3. Calculate the smallest distance, \( d_{\min} \), from \( x_i \) to all other \( x_j \) in the initial sample.
4. If \( d_{\min} \geq \delta \), increase \( i \) by 1 and return to step 2 if \( i \leq n \), otherwise stop.
5. If \( d_{\min} < \delta \), draw an integer \( j \) at random from \( 1, 2, \ldots, N \), set \( x_i = X_j \) and return to step 3.

Samples generated in this way exhibit more regular spatial arrangements than would random samples of the same size. The degree of regularity achievable will be influenced by the spatial arrangement of the population \( X_i : i = 1, \ldots, N \), the specified value of \( \delta \) and the sample size \( n \). For any given population, if \( n \) and/or \( \delta \) is too large, a sample of the required size with the distance between any two sampled locations at least \( \delta \) will not be achievable; the algorithm will then find \( n_s < n \) points that can be placed for the given parameters.

**Sampling close pairs of points.**

For some purposes, typically when using the same sample for parameter estimation and spatial prediction, it is desirable that a spatial sampling scheme include pairs of closely spaced points \( x \). The function offers two ways of specifying close pairs, either as the closest available unsampled point to an existing sampled point (cp.criterion = cp.neighb), or as a random choice from amongst all available unsampled points within distance \( \zeta \) of an existing sampled point (cp.criterion = cp.zeta). The algorithm proceeds as follows.

Let \( k \) be the required number of close pairs.

1. Construct a simple inhibitory design \( SI(n-k,\delta) \).
2. Sample \( k \) from \( x_1, \ldots, x_{n-k} \) without replacement and call this set \( x_j : j = 1, \ldots, k \).
3. For each \( x_j : j = 1, \ldots, k \), select a close pair \( x_{n-k+j} \) according to the specified criterion.
**Note:** Depending on the spatial configuration of potential sampling locations and, when the selection criterion \( \text{cp.criterion} = \text{cp.zeta} \), the specified value of \( \text{zeta} \), it is possible that one or more of the selected points \( x_j \) in Step 2 will not have an eligible “close pair”. In this case, the algorithm will try find an alternative \( x_j \) and report a warning if it fails to do so.

**Value**

a list with the following four components:

- `unique.locs`: the number of unique sampled locations.
- `delta`: the value of \( \delta \) after taking into account the number of close pairs \( k \). If `delta.fix = TRUE`, this will be \( \delta \) input by the user.
- `k`: the number of close pairs included in the sample (for inhibitory plus close pairs design).
- `sample.locs`: a `sf` or `sp` object containing the final sampled locations and any associated values.

**Note**

If 'delta' is set to 0, a completely random sample is generated. In this case, 'close pairs' are not permitted and 'zeta' becomes trivial.

**Author(s)**

Henry J. Crosby <henry.crosby@warwick.ac.uk>
Godwin Yeboah <godwin.yeboah@warwick.ac.uk>
J. Porto De Albuquerque <J.Porto@warwick.ac.uk>

**References**


**Examples**

```r
## Not run: library(sp)
bounding_geom <- SpatialPolygonsDataFrame(SpatialPolygons(list(Polygons(list(Polygon(
  cbind(3.888959,3.888744,3.888505,3.888355,3.887893,3.887504,3.886955,
    3.886565,3.886303,3.886159,3.885650,3.885650,3.885595,3.885404,
    3.885444,3.885897,3.886692,3.887241,3.888068,3.888323,3.888697,
    3.889150,3.889548,3.889090,3.890184,3.890628,3.891258,3.891807,
    3.892061,3.892292,3.892689,3.893294,3.893008,3.893676,3.888959),
  c(7.379483,7.379785,7.380024,7.380294,7.380629,7.380986,7.381448,
    7.381861,7.382243,7.382474,7.383277,7.383468,7.383890,7.384263,
    7.384669,7.385258,7.385313,7.385194,7.384868,7.384900,7.385051,)),
```

```
osm_random_sample

Spatially random sampling.

Description

This function draws a spatially random sample from a either (1) a discrete set of OSM features defined in the function parameters or (2) a continuous surface defined by a user defined geographical region.

Usage

```r
osm_random_sample(
  bounding_geom = NULL,
  key = NULL,
  value = NULL,
  boundary_or_feature = "boundary",
  feature_geom = NULL,
  data_return = c("osm_polygons", "osm_points", "osm_multipolygons", "multilines", "lines"),
  boundary = 0,
  buff_dist = 0,
  buff_epsg = 4326,
  join_type = "within",
  dis_or_cont,
  sample_size,
  join_features_to_osm,
  plotit = TRUE,
  plotit_leaflet = TRUE
)
```
Arguments

bounding_geom
a sf or sp object (with $N \geq \text{size}$) where each line corresponds to one spatial location. It should contain values of 2D coordinates, data and, optionally, covariate(s) value(s) at the locations. This argument must be provided when sampling from a 'discrete' set of points, see 'type' below for details.

key
A feature key as defined in OSM. An example is 'building'.

value
a value for a feature key (key); can be negated with an initial exclamation mark, value = '!this', and can also be a vector, value = c ('this', 'that').

boundary_or_feature
specifies whether the user inputs a boundary or a set of user-inputted features. For example if the user selects "boundary", they can provide a spatial data frame or OSM locality which will query the osm features within that boundary or locality. If the user select "feature" then they can provide a data frame of features that they want to sample

feature_geom
is a user inputted data frame of features that are required to be sampled.

data_return
specifies what data types (as specified in OSM) you want returned. More than one can be selected. The options are 'osm_polygons', 'osm_points', 'osm_multipolygons','osm_multilines'.

boundary
categorical variable to determine whether the exact boundary provided (boundary = 0), the bounding box (boundary = 1) or a buffer around the boundary (boundary = 2) is used for sampling. Default is boundary = 0.

buff_dist
if boundary = 2 then this value determines the size of the buffer by distance. The default is buff_dist is NULL).

buff_epsg
if boundary = 2 then this value determines the local geographic grid reference so that the buffer can be calculated in meters. The default is buff_epsg = 4326 which will use decimal degrees instead of meters. As an example, 27700 relates to the British National Grid.

join_type
a text value to determine how to spatially join all features with the boundary. The options are 'within' or 'intersect'.

dis_or_cont
random sampling, a choice of either 'discrete', from a set of $N$ potential sampling points or 'continuum' from independent, completely random points.

sample_size
a non-negative integer giving the total number of locations to be sampled.

join_features_to_osm
is a TRUE or FALSE variable which allows the user to specify whether they want their feature geom to be spatially joined to OSM features. The output sampling data frame will have an additional column showing the joined OSM id.

plotit
'logical' specifying if graphical output is required. Default is plotit = TRUE.

plotit_leaflet
'logical' specifying if leaflet (html) graphical output is required. This is prioritised over plotit if both are selected. Default is plotit_leaflet = TRUE.

Value

a df object named 'results' of dimension $n$ by 4 containing the final sampled osm_ids, centroid locations (named x, y) and whether the instance is in the selected sample (named inSample with a value of 0/1), if sampling from a 'discrete' set of points. A df object of dimension $n$ by 3 containing the serial id and centroid locations for all sample instances, if sampling from a 'continuum'.

osm_random_sample
Author(s)

Henry J. Crosby <henry.crosby@warwick.ac.uk>
Godwin Yeboah <godwin.yeboah@warwick.ac.uk>
J. Porto De Albuquerque <J.Porto@warwick.ac.uk>

References


Examples

## Not run:
library(sp)
bounding_geom <-
SpatialPolygonsDataFrame(
  SpatialPolygons(
    SpatialPolygons(
      Polygon(
        cbind(
          c(3.888959, 3.888744, 3.888585, 3.888355, 3.888793, 3.887504, 3.886955,
            3.886565, 3.886303, 3.886159, 3.885650, 3.885650, 3.885595, 3.885404,
            3.885444, 3.885897, 3.886692, 3.887241, 3.888068, 3.888323, 3.888697,
            3.889150, 3.889548, 3.889890, 3.890184, 3.890828, 3.891258, 3.891807,
            3.892061, 3.892292, 3.892689, 3.893294, 3.893008, 3.893676, 3.889594),
          c(7.379483, 7.379785, 7.380024, 7.380294, 7.380629, 7.380966, 7.381448,
            7.381861, 7.382243, 7.382474, 7.383277, 7.383468, 7.383890, 7.384263,
            7.384669, 7.385258, 7.385313, 7.385194, 7.384868, 7.384900, 7.385051,
            7.385067, 7.384955, 7.384749, 7.384526, 7.384120, 7.384009, 7.384080,
            7.384430, 7.384478, 7.384629, 7.384772, 7.383269, 7.380963, 7.379483))),
  ID=1)),
data.frame( ID=1))
proj4string(bounding_geom) <- CRS('+proj=longlat +datum=WGS84')

set.seed(15892)
xy.sample <- osm_random_sample(buff_dist=NULL,
  boundary_or_feature = "boundary",
  bounding_geom = bounding_geom,
  key = 'building', value = NULL, boundary = 0,
  buff_epsg = NULL, join_type = 'intersect',
  dis_or_cont = 'discrete', sample_size = 70,
  plotit = TRUE, plotit_leaflet = TRUE,
  data_return = c('osm_polygons'))

## End(Not run)
Index

osm_contin_inhibit_sample, 2
osm_discrete_inhibit_sample, 5
osm_random_sample, 4, 9