Package: intamapInteractive (via r-universe)

August 26, 2024

Version 1.2-6

Date 2023-10-30

Title Interactive Add-on Functionality for 'intamap'

Maintainer Jon Skoien <jon.skoien@gmail.com>

Depends intamap

Imports spatstat.geom, automap, gstat, methods, sp, spcosa, sf

Description The methods in this package adds to the functionality of the 'intamap' package, such as bias correction and network optimization. Pebesma et al (2010) gives an overview of the methods behind and possible usage <doi:10.1016/j.cageo.2010.03.019>.

License GPL (>= 2)

NeedsCompilation no

Author Edzer Pebesma [aut], Jon Skoien [aut, cre], Olivier Baume [ctb], A Chorti [ctb], Dionisis Hristopulos [ctb], Stepahnie Melles [ctb], Giannis Spiliopoulos [ctb]

Date/Publication 2023-10-30 13:30:05 UTC

Repository https://jskoien.r-universe.dev

RemoteUrl https://github.com/cran/intamapInteractive

RemoteRef HEAD

RemoteSha e9283d7f3564854ac4b8303799ea3d4b121e22ce

Contents

`intamapInteractive-package` *Interactive functionality added to the intamap package*

Description

This package provides some added functionality to the `link[intamap]{intamap-package}` for automatic interpolation of environmental variables. Whereas `link[intamap]{intamap-package}` was specifically developed as a statistical back-end for a Web Processing Service (WPS), this package offers some functionality that is not possible to access through such a WPS.

The methods in this package can mainly be put into three groups:

bias correction methods for estimating and possible correct for biases between measurement networks, due to differences in measurement strategies, measurement devices, or (unknown) post-processing of data

segmentation method for segmentation of data, based on their measurement density

network optimization methods for optimizing a measurement network (adding or removing observation points), based on different criteria

---

`anisotropyChoice` *anisotropyChoice*

Description

This function combines segmentation of scattered 2D data and estimation of anisotropy parameters using the CTI method.

Usage

```
anisotropyChoice(object)
```

Arguments

`object`	An Intamap type object containing one `SpatialPointsDataFrame` with `observations`.

Details

The function AnisotropyChoice function employs the doSegmentation function to automatically separate the original dataset into clusters based on the sampling density and the spatial locations of the data (see doSegmentation for details). The results of the segmentation procedure and the anisotropy analysis per cluster are returned in a matrix of dimension [cl]x5, where [cl] is the number of clusters . Each row of the matrix contains the cluster index, the anisotropy ratio, the anisotropy direction, the number of cluster points and the area inside the convex hull of the cluster. In addition, a single set of anisotropy parameters is returned in the element anisPar. These parameters are calculated using weighted averages of the covariance Hessian matrix estimates in each cluster. The weights are based on the area enclosed by the convex hull of each cluster.

Value

object: A modified Intamap type object is returned, which contains the results of the anisotropy parameter estimation. The anisotropy parameters are returned in the element anisPar as described below.

anisPar — List element in object that contains a list with the following elements:

- ratio A coarse-grained anisotropy ratio for all the data
- direction A coarse-grained anisotropy orientation for all the data
- clusters A matrix of dimension [cl]x5 which determines the anisotropy per cluster. Each row of clusters gives the (cluster id, anisotropy ratio, anisotropy direction, number of points, area) for each cluster detected.

clusters — list element added to the original object containing the segmentation results.

- **index** Index array identifying the cluster in which each observation point belongs. Zero value means that the observations has been removed.
- **clusterNumber** Number of clusters detected.

Note

This function uses the akima package to perform "bilinear" and "bicubic" interpolation for the estimation of spatial derivatives

Author(s)

D.T. Hristopulos, G.Spiliopoulos, A.Chorti

References

[1] http://www.intamap.org

[2] A. Chorti and D. T. Hristopulos (2008). Non-parametric Identification of Anisotropic (Elliptic) Correlations in Spatially Distributed Data Sets, IEEE Transactions on Signal Processing, 56(10), 4738-4751 (2008).

[3] D. T. Hristopulos, M. P. Petrakis, G. Spiliopoulos, A. Chorti (2009). Non-parametric estimation of geometric anisotropy from environmental sensor network measurements, StatGIS 2009: Geoinformatics for Environmental Surveillance Proceedings (ed. G. Dubois).

Examples

```
library(gstat)
data(walker)
object=createIntamapObject(observations=walker)
object=anisotropyChoice(object)

print(summary(object$clusters$index))
print(object$anisPar)
```

`biasCorr` *Bias correction*

Description

Identifies and removes biases from measurement networks

Usage

```
biasCorr(object,regCode = "regCode",...)
```

Arguments

`object`	Data frame with observations with same format as `observations` described in the presentation of the `intamap-package`)
`regCode`	the column name of regions in the data polygons, if existing
`...`	further arguments to the bias correction methods called, see details below

Details

Many data sets can consist of data coming from a large number of different measurement networks, using different measurement devices or applying different methods for post-processing the observations. Some of these networks can exist in the same area, e.g. when different authorities are measuring the same, but at different locations (one of them in cities, the other one close to lakes), some networks will only exist as neighbouring networks (networks operated by a municipality or a country). Local networks can also be grouped together as one national data-base, which can again be merged into an international data-base.

One challenge with the merging into data-bases is that there will be inconsistencies between measurements in the different networks, which will again cause difficulties when attempting to map the observations, as done in the intamap-package. The intention of this function is therefore to call other functions that are able to identify and remove such differences, which can be referred to as biases between the networks.

There are at the moment two methods available for bias correction, "UK" and "LM". "UK" is a universal kriging based approach implemented in `findBiasUK`. This method can only deal with biases between neigbouring networks, but is well capable of taking covariates into account. "LM" is based on local methods for estimating differences between networks, and is implemented in

findLocalBias and findRegionalBias. The choice between the methods is given by the parameter biasRemovalMethod in the parameter element of the object, set in getIntamapParams, called from createIntamapObject.

The function will remove biases according to the settings of the parameters removeBias. Below is a list of the functions available for bias corrections. See each individual function for more information about usage.

findBiasUK The universal kriging based function for finding biases between neighbouring networks

findLocalBias Find biases for ovelapping networks

removeLocalBias Removes biases between ovelapping networks

findBoundaryLines Find points that define adjacent boundaries between regions

findRegionalBias Find biases for neighbouring networks

removeRegionalBias Remove biases between neighbouring networks

Value

Data frame with observations, with the identified biases removed.

Author(s)

Jon Olav Skoien

References

Skoien, J. O., O. P. Baume, E. J. Pebesma, and G. B. M. Heuvelink. 2010. Identifying and removing heterogeneities between monitoring networks. Environmetrics 21(1), 66-84.

See Also

findLocalBias

Examples

```
data(meuse)
data(meuse.grid)
observations = data.frame(x = meuse$x,y = meuse$y,value = log(meuse$zinc))
coordinates(observations) = ~x+y
gridded(meuse.grid) = ~x+y
pBoundaries = spsample(observations, 8, "regular",bb = bbox(observations) +
              matrix(c(-400,-400,400,400),ncol=2),offset=c(0,0))
gridded(pBoundaries) = TRUE
cs = pBoundaries@grid@cellsize[1]/2
dx = cs/5

Srl = list()
nb = dim(coordinates(pBoundaries))[1]
for (i in 1:nb) {
  pt1 = coordinates(pBoundaries)[i,]
```

```
  x1 = pt1[1]-cs
  x2 = pt1[1]+cs
  y1 = pt1[2]-cs
  y2 = pt1[2]+cs

  boun = data.frame(x=c(seq(x1,x2,dx),rep(x2,11),seq(x2,x1,-dx),rep(x1,11)),
                    y=c(rep(y1,11),seq(y1,y2,dx),rep(y2,11),seq(y2,y1,-dx)))
  coordinates(boun) = ~x+y
  boun = Polygon(boun)
  Srl[[i]] = Polygons(list(boun),ID = as.character(i))
}
pBoundaries = SpatialPolygonsDataFrame(SpatialPolygons(Srl),
                                       data = data.frame(ID=c(1:nb)))
observations$ID = over(observations, geometry(pBoundaries))
blines = findBoundaryLines(pBoundaries,regCode = "ID")


object = createIntamapObject(observations,meuse.grid,boundaryLines = blines,
  params = list(removeBias = "regionalBias"))
object = biasCorr(object,regCode= "ID")
object$regionalBias$regionalBias
pBoundaries$bias = NA
pBoundaries$bias[object$regionalBias$regionalBias$ID] = object$regionalBias$regionalBias$ols
spplot(pBoundaries,"bias",sp.layout = list(list("sp.points",observations)))
```

`calculateMukv` *MUKV computation*

Description

Computes mean universal kriging variance (MUKV) for given geostatistical parameters

Usage

```
calculateMukv(observations, predGrid, model, formulaString, fun, ...)
```

Arguments

`observations`	`SpatialPoints` or `SpatialPointsDataFrame` with observation locations and possible covariates
`predGrid`	`Spatial` object with coordinates of prediction locations (usually `SpatialGrid` or `SpatialGridDataFrame` when independent covariate predictor variables are used)
`model`	Variogram model:object of class `variogramModel`, of the form created by `vgm`
`formulaString`	formula that defines the dependent variable as a linear model of independent variables; suppose the dependent variable has name `z`, for ordinary and simple kriging use the formula `z~1`; for universal kriging, suppose `z` is linearly

	dependent on x and y, use the formula z~x+y. The formulaString defaults to "value~1" if value is a part of the data set. If not, the first column of the data set is used.
fun	alternative penalty function, needs to be a function which can take the same arguments as calculateMukv
...	other arguments to be passed on at lower level functions

Details

This function computes kriging on the predGrid with krige function, and averages the kriging variance (MUKV). With covariates, the function takes a universal kriging model into account.

Value

MUKV value

Author(s)

S.J. Melles, O. Baume, J. Skoien

Examples

```
# load data:
library(gstat)
data(meuse)
coordinates(meuse) = ~x+y
data(meuse.grid)
coordinates(meuse.grid) = ~x+y
gridded(meuse.grid) = TRUE
meuse.grid$soil = factor(meuse.grid$soil)

# estimate variogram:
smplvarUK = variogram(zinc~dist+ffreq+soil, meuse)
plot(smplvarUK)
vfitUK = fit.variogram(variogram(zinc~dist+ffreq+soil, meuse), vgm(1, "Exp", 300, 1))
plot(smplvarUK, vfitUK)

calculateMukv(meuse, meuse.grid, vfitUK, zinc~dist+ffreq+soil)
```

---

doSegmentation *Spatial Segmentation - Clustering for Scattered Observations*

Description

This function performs segmentation of scattered 2D data based on sampling density and location.

Usage

```
doSegmentation(object)
```

Arguments

`object`	An Intamap type object containing the element (list) `observations`, which includes the coordinates of the observation locations

Details

This function performs segmentation of scattered 2D data based on sampling density and location. Let us assume that `No` is the number of observation locations. If `No`< 200, then a single cluster is returned.

(1) The segmentation algorithm first removes isolated distant points, if there are any, from the observation locations. Points (xi,yi) are characterized as 'isolated' and 'distant' if they satisfy the following conditions : $abs(xi-mean(x)) > 4 *std(x) or abs(yi-mean(y)) > 4 *std(y)$ and distance from closest neighbor $> sqrt((std(x)/2)^2+(std(y)/2)^2)$. After the first step the size of the original dataset is reduced to N (N= No - isolated points) points.

(2) A sampling density matrix (lattice) consisting of N cells that cover the study area is constructed. Each cell is assigned a density value equal to the number of observation points inside the cell. In addition, each observation point is assigned the sampling density value of the containing cell.

(3) Unsupervised clustering edge detection is used to determine potential cluster perimeters.

(4) Each closed region's perimeter is labeled with a different cluster (segment) number.

(5) All observation points internal to a cluster perimeter are assigned to the specific cluster.

(6) Each cluster that contains fewer than 50 observation points is rejected.

(7) The observation points that have not initially been assigned to a cluster and those belonging to rejected (small) clusters are assigned at this stage. The assignment takes into account both the distance of the points from the centroids of the accepted clusters as well as the mean sampling density of the clusters.

Note: The `No`< 200 empirical constraint is used to avoid extreme situations in which the sampling density is concentrated inside a few cells of the background lattice, thereby inhibiting the edge detection algorithm.

Value

A modified Intamap object which additionally includes the list element `clusters`. This element is a list that contains (i) the indices of removed points from `observations`; (ii) the indices of the clusters to which the remaining observation points are assigned and (iii) the number of clusters detected.

`clusters` list element added to the original object containing the segmentation results.

- **rmdist** Indices of removed points.
- **index** Index array identifying the cluster in which each observation point belongs.
- **clusterNumber** Number of clusters detected.

Author(s)

A. Chorti, Spiliopoulos Giannis, Hristopulos Dionisis

References

[1] D. T. Hristopulos, M. P. Petrakis, G. Spiliopoulos, A. Chorti (2009). Non-parametric estimation of geometric anisotropy from environmental sensor network measurements, StatGIS 2009: Geoinformatics for Environmental Surveillance Proceedings (ed. G. Dubois).

Examples

```
library(gstat)

data(walker)
# coordinates(walker)=~X+Y
object=createIntamapObject(observations=walker)
object=doSegmentation(object)

print(summary(object$clusters$index))
```

---

findBiasUK *Finding the regional biases using GLM*

Description

Method for identifying regional biases (in most cases biases between countries)

Usage

```
findBiasUK(object)
```

Arguments

object	an object of class SpatialPointsDataFrame, at least containing observations and a regional identification code (regCode)

Value

A data.frame with the biases for each country with uncertainty.

Author(s)

Olivier Baume

See Also

findRegionalBias

`findBoundaryLines` *Finding the regional boundaries*

Description

Method for identifying points on the boundaries between regions (in most cases biases between countries)

Usage

```
findBoundaryLines(polygons, projOrig, projNew, regCode = "regCode")
```

Arguments

`polygons`	A `SpatialPolygonsDataFrame` with the polygons defining the boundaries of each separate region.
`projOrig`	The original projection of the boundaries
`projNew`	If a different projection is wanted for the output
`regCode`	the column name of regions in the data polygons

Details

This function finds the points defining the boundary between two polygons and passes a `SpatialPointsDataFrame` with these points back. The result in mainly used by `findRegionalBias` for estimation of regional biases. The function is based on the boundary between the polygons being defined by the same points.

Value

A `SpatialPointsDataFrame` with points defining the boundaries between regions.

Author(s)

Jon Olav Skoien

References

Skoien, J. O., O. P. Baume, E. J. Pebesma, and G. B. M. Heuvelink. 2010. Identifying and removing heterogeneities between monitoring networks. Environmetrics 21(1), 66-84.

Examples

```
data(meuse)
observations = data.frame(x = meuse$x,y = meuse$y,value = log(meuse$zinc))
coordinates(observations) = ~x+y
pBoundaries = spsample(observations, 10, "regular", bb = bbox(observations) +
                         matrix(c(-400,-400,400,400),ncol=2),offset=c(0,0))
gridded(pBoundaries) = TRUE
```

```
cs = pBoundaries@grid@cellsize[1]/2

Srl = list()
nb = dim(coordinates(pBoundaries))[1]
for (i in 1:nb) {
  pt1 = coordinates(pBoundaries)[i,]
  x1 = pt1[1]-cs
  x2 = pt1[1]+cs
  y1 = pt1[2]-cs
  y2 = pt1[2]+cs

  boun = data.frame(x=c(x1,x2,x2,x1,x1),y=c(y1,y1,y2,y2,y1))
  coordinates(boun) = ~x+y
  boun = Polygon(boun)
  Srl[[i]] = Polygons(list(boun),ID = as.character(i))
}
pBoundaries = SpatialPolygonsDataFrame(SpatialPolygons(Srl),
                                       data = data.frame(ID=c(1:nb)))
observations$ID = over(observations, geometry(pBoundaries))
blines = findBoundaryLines(pBoundaries, regCode = "ID")
```

`findLocalBias` *Finds (and removes) biases between overlapping networks*

Description

The function tries to identify differences between different networks of observation stations that share a region. From these differences, biases are estimated, and can be removed.

Usage

```
findLocalBias(object, gid = "group",
              formulaString = value ~ 1, regCode="regCode",...)
removeLocalBias(object, localBias, gid = "group", formulaString = value ~ 1,
                regCode = "regCode")
```

Arguments

`object`	data frame with observations
`gid`	name of column identifying groups of local networks
`formulaString`	formula that defines the dependent variable as a linear model of independent variables; suppose the dependent variable has name `z`, for ordinary and simple kriging use the formula `z~1`; for universal kriging, suppose `z` is linearly dependent on `x` and `y`, use the formula `z~x+y`
`regCode`	the column name of regions in the `object`, if existing
`localBias`	List of data frames, for a single region, or for each of the regions, each containing biases for different networks in the region(s), result of `findLocalBias`
`...`	arguments to be passed to sub-functions

Details

`findLocalBias` tries to identify biases between overlapping networks, i.e. when there is no boundary between different networks sampling the same type of data. This can typically happen if different governmental bodies are responsible for different types of measurement, e.g. one measuring the situation around populated areas, the other one measuring close to water bodies.

The function will then try to find the difference between the different networks, and estimate the individual bias for each network, relative to a reference value, usually the average of all networks. The method is not recommended if there can be assumed to be a dependency beteween the process and the networks.

`removeLocalBias` removes the bias estimated in `findLocalBias`.

Value

From `findLocalBias`: A list consisting of one element for each regional network, or an element `single` if only one regional network is apparent. Each of these elements is again a list consisting of several other elements, where `bias` is the interesting one. The remaining elements are only necessary for debugging purposes. The elements D, V and Q refers to the matrices with same names in Skoien et al. (2009), i.e. the relationship matrix, the variance matrix and the difference matrix.

From `removeLocalBias`: A `SpatialPointsDataFrame` with the biases subtracted.

Author(s)

Jon Olav Skoien

References

Skoien, J. O., O. P. Baume, E. J. Pebesma, and G. B. M. Heuvelink. 2010. Identifying and removing heterogeneities between monitoring networks. Environmetrics 21(1), 66-84.

Examples

```
# Assuming that the soil type is the source of biases
data(meuse)
coordinates(meuse) = ~x+y


lb = findLocalBias(meuse,gid = "soil",formulaString=as.formula(zinc~1))
lb$single$bias

meuseUnbias = removeLocalBias(meuse,localBias = lb, gid = "soil",
    formulaString = zinc~1)
```

findRegionalBias *Find and/or remove regional biases*

Description

Method for identifying regional biases (in most cases biases between countries)

Usage

```
findRegionalBias(object,boundaryLines,
                 formulaString = value~1,
                 minKrige = 5, regCode = "regCode", unbias = "default")
removeRegionalBias(object, regionalBias, formulaString = value~1, regCode = "regCode")
```

Arguments

object	an object of class SpatialPointsDataFrame, at least containing observations and a regional identification code (regCode)
boundaryLines	SpatialPointsDataFrame with points defining the boundaries between regions. This can be found using findBoundaryLines.
formulaString	formula that defines the dependent variable as a linear model of independent variables; suppose the dependent variable has name z, for ordinary and simple kriging use the formula z~1; for universal kriging, suppose z is linearly dependent on x and y, use the formula z~x+y
minKrige	Setting a minimum number of observations necessary for kriging
regCode	the column name of regions in the data polygons, if existing
unbias	defines if a particular data dependent function should be used to set unbiasedness constraints for the biases. "default" gives one additional constraint, assuming that the average of the biases should be equal to zero. See also details below.
regionalBias	List of data frames, one for each region, each containing biases for different networks in the region.

Details

This methods attempts to find biases between regional networks that are separated by a boundary, based on line kriging along these boundaries. A typical example of such networks would be different national networks, with the country borders as boundaryLines, but also other boundaries can be considered. Further details can be found in Skoien et al. (2009).

The parameter unbias can be used to name the unbiasedness function if the user needs a different unbiasedness constraint than the default one. Such a function (with unbias = "new" above) should be similar to the following:

```
unBias.new = function(cDiff,uRegCode) {
  D = cDiff$D
  Q = cDiff$Q
```

```
    V = cDiff$V
#
    D = rbind(D,0)
    cd = dim(D)[1]
    ino = which(uRegCode == "NO")
    iis = which(uRegCode == "IS")
    iuk = which(uRegCode == "UK" | uRegCode == "GB")
    if (length(iis) > 0) {
      D[cd,ino] = .5
      D[cd,iuk] = .5
      D[cd,iis]= -1
      Q[cd] = 0
      V[cd] = max(V)
      cd = cd+1
      D = rbind(D,0)
    }
    cd = cd + 1
    D = rbind(D,0)
    D[cd,] = 1
    Q[cd] = 0
    V[cd] = min(V)
    cDiff$D = D
    cDiff$Q = Q
    cDiff$V = V
    return(cDiff)
  }
```

The last part is similar to unbias.default. In the other part is solving the problem where there are no boundaries between Iceland and any other countries. This would cause a missing constraint when searching for the biases, which will make it impossible to find a solution. The solution here sets the bias for Iceland equal to the average of the bias for Norway and United Kingdom. Note that the real bias for Iceland is not really estimated in this case, this construction is mainly to make sure that the system can be solved. If one were only interested in the bias, it would in this case be better to remove Iceland from the data set, as a real bias is not possible to find.

Value

For `findRegionalBias`; a `data.frame` with the biases for each country with uncertainty.

For `removeRegionalBias`; a `data.frame` with observations, with biases removed

Author(s)

Jon Olav Skoien

References

Skoien, J. O., O. P. Baume, E. J. Pebesma, and G. B. M. Heuvelink. 2010. Identifying and removing heterogeneities between monitoring networks. Environmetrics 21(1), 66-84.

Examples

```
library(intamapInteractive)
data(meuse)
observations = data.frame(x = meuse$x,y = meuse$y,value = log(meuse$zinc))
coordinates(observations) = ~x+y
pBoundaries = spsample(observations, 10, "regular",bb = bbox(observations) +
              matrix(c(-400,-400,400,400),ncol=2),offset=c(0,0))
gridded(pBoundaries) = TRUE
cs = pBoundaries@grid@cellsize[1]/2

Srl = list()
nb = dim(coordinates(pBoundaries))[1]
for (i in 1:nb) {
  pt1 = coordinates(pBoundaries)[i,]
  x1 = pt1[1]-cs
  x2 = pt1[1]+cs
  y1 = pt1[2]-cs
  y2 = pt1[2]+cs

  boun = data.frame(x=c(x1,x2,x2,x1,x1),y=c(y1,y1,y2,y2,y1))
  coordinates(boun) = ~x+y
  boun = Polygon(boun)
  Srl[[i]] = Polygons(list(boun),ID = as.character(i))
}
pBoundaries = SpatialPolygonsDataFrame(SpatialPolygons(Srl),
                                       data = data.frame(ID=c(1:nb)))
observations$ID = over(observations, geometry(pBoundaries))
blines = findBoundaryLines(pBoundaries, regCode = "ID")
rb = findRegionalBias(observations, blines, value~1, regCode = "ID")
rb$regionalBias

obs2 = removeRegionalBias(observations, rb, value~1, regCode = "ID")
```

`optimizeNetwork` *Optimization of networks*

Description

Optimizes the sampling design of observation point locations using a varity of methods including spatial coverage by k means (as described in `spcosa`) or by maximizing nearest neighbour distances and spatial simulated annealing (SSA, as described in `ssaOptim`) using MUKV as a criterion (`calculateMukv`) .

Usage

```
optimizeNetwork(observations, predGrid, candidates, method, action,
```

```
nDiff, model, criterion = "MUKV", plotOptim = TRUE, nGridCells,
nTry, nr_iterations = 10000, formulaString, fun, ...)
```

Arguments

observations	object of class Spatial* with coordinates and possible covariates
predGrid	object of class Spatial* used when method = "ssa". predGrid should contain the coordinates of prediction locations for optimization. Usually predGrid is a SpatialGrid / SpatialPixels or a SpatialGridDataFrame / SpatialPixelsDataFrame when independent covariate predictor variables are used
candidates	when method = "manual" or method = "ssa", candidates is the study area of class SpatialPolygonsDataFrame; for other methods, when action = add, candidates are points or polygons of class Spatial*
method	"spcov" for spatial coverage, "ssa" for spatial simulated annealing or "manual" for manual processing of the network
action	character string indicating which action to perform: "add" to add new measurement stations to the existing network or "del" to delete existing stations
nDiff	number of stations to add or delete
model	variogram model to consider when method = "ssa" and criterion = "mukv"; object of class variogramModel, as generated by vgm
criterion	Only in use for method "ssa": character string, "mukv"
plotOptim	logical; if TRUE, creates a plot of the result as optimization progresses; TRUE by default
nGridCells	when method is "spcov" and action is "add": the approximate number gridcells to explore within the candidate map as locations for new observations
nTry	when method is "spcov" and action is "add": nTry is the number of initial configurations to try. The method will keep the best solution in order to reduce the risk of ending up with an unfavorable solution
nr_iterations	number of iterations to process before stoping. The default coolingFactor in ssaOptim is also a function of number of iterations. Refer to ssaOptim for more details
formulaString	When method = "ssa", this formula defines the dependent variable as a linear model of independent variables; suppose the dependent variable has name z, for ordinary and simple kriging use the formula z~1; for universal kriging, suppose z is linearly dependent on x and y, use the formula z~x+y. The formulaString defaults to "value~1" if value is a part of the data set. If not, the first column of the data set in observations is used.
fun	Alternative objective function for optimization, the input and output should match the ones of (calculateMukv (except for fun)
...	other arguments to be passed on to lower level functions

Details

This function contains different methods to optimally add or remove point locations to or from a measurement network (Baume et al. 2011). Points can be added or deleted in the following ways:

1. manually
2. using a spatial coverage approach by k means to add stations (as described in spcosa, Brus et al. 2006) using a spatial coverage approach by maximizing mean nearest neighbour distances to remove stations (as described in spCovDel)
3. or using spatial simulated annealing with mean universal kriging variance as a criterion (calculateMukv, Brus & Heuvelink 2007, Melles et al. 2011)

The results of different methods can be checked using the function calculateMukv, which returns mean universal kriging variance for an optimized network.

The user should be aware of the following limitations:

1. method = "ssa" is only implemented for criterion = "mukv"
2. Input candidates should preferably be a continuous domain such as SpatialPolygons
3. method = "ssa" with criterion = "mukv" makes it possible to assume a linear relationship between independent variables in predGrid and dependent variables at observation locations using universal kriging (krige). However, a correct estimate of mean universal kriging variance requires that the independent covariate variables be known at candidate locations. Thus it is necessary to have complete spatial coverage for all covariate predictors in the linear model. Covariate information must be available at both new candidate measurement locations and prediction locations. This information is acquired (or sampled) from predGrid at candidate locations during SSA using a call to over by default. But see ssaOptim for more details and an option to interpolate these values for candidate locations from predGrid.
4. Note that it is not recommended to use independent variables which differ strongly in magnitude (as for traditional universal kriging)
5. If no formulaString is supplied, an ordinary kriging formula is assumed, and optimization will proceed using mean ordinary kriging variance

Value

Object of class SpatialPoints* with spatial coordinates of optimized locations (including observation locations when action = "add")

Author(s)

O. Baume, S.J. Melles, J. Skoien

References

O. P. Baume, A. Gebhardt, C. Gebhardt, G. B. M. Heuvelink, J. Pilz (2011). Network optimization algorithms and scenarios in the context of automatic mapping, Computers and Geosciences, 37: 289-294 (2011).

S. J. Melles, G. B. M. Heuvelink, C. J. W. Twenhofel, U. Stohlker (2011). Optimizing the spatial pattern of networks for monitoring radioactive releases, Computers and Geosciences, 37: 280-288 (2011).

D. J. Brus, G. B. M. Heuvelink (2007). Optimization of sample patterns for universal kriging of environmental variables, Geoderma, 138: 86-95 (2007).

D. J. Brus, J. de Gruijter, J. van Groenigen (2006). Designing spatial coverage samples using the k-means clustering algorithm. In A. McBratney M. Voltz and P. Lagacherie, editor, Digital Soil Mapping: An Introductory Perspective, Developments in Soil Science, vol. 3., Elsevier, Amsterdam.

See Also

ssaOptim, spCovDel, spCovAdd, calculateMukv, stratify

Examples

```
# load data:
library(gstat)
data(meuse)
coordinates(meuse) = ~x+y
data(meuse.grid)
coordinates(meuse.grid) = ~x+y
gridded(meuse.grid) = TRUE
predGrid = meuse.grid

# estimate variograms (OK/UK):
vfitOK = fit.variogram(variogram(zinc~1, meuse), vgm(1, "Exp", 300, 1))
vfitUK = fit.variogram(variogram(zinc~x+y, meuse), vgm(1, "Exp", 300, 1))
vfitRK = fit.variogram(variogram(zinc~dist+ffreq+soil, meuse), vgm(1, "Exp", 300, 1))

# study area of interest:
bb = bbox(predGrid)
boun = SpatialPoints(data.frame(x=c(bb[1,1],bb[1,2],bb[1,2],bb[1,1],bb[1,1]),
                                y=c(bb[2,1],bb[2,1],bb[2,2],bb[2,2],bb[2,1])))
Srl = Polygons(list(Polygon(boun)),ID = as.character(1))
candidates = SpatialPolygonsDataFrame(SpatialPolygons(list(Srl)),
                                      data = data.frame(ID=1))


# add 20 more points assuming OK model (SSA method):
 optimOK <- optimizeNetwork(meuse, meuse.grid, candidates = candidates,
  method= "ssa", action= "add", nDiff = 20, model = vfitOK, criterion="MUKV",
  nr_iterations=10000, nmax=40)


# add 20 more points assuming UK model (SSA method):
optimUK <- optimizeNetwork(meuse, meuse.grid, candidates = candidates,
   method = "ssa", action = "add", nDiff = 20, model=vfitUK, criterion="MUKV",
   nr_iterations = 10000, nmax = 40, formulaString = zinc~x+y)

# add 20 more points with auxiliary variables (SSA method):
optimRK <- optimizeNetwork(meuse, meuse.grid, candidates=candidates,
   method="ssa", action="add", nDiff=4, model=vfitRK, criterion="MUKV",
   nr_iterations=10000, formula=zinc~dist+ffreq+soil, nmax=200)

# add optimally 20 stations from current network with method "spcov"
# (spatial coverage method)
```

```
optimSC = optimizeNetwork(meuse, meuse.grid, candidates, method = "spcov",
          action = "add", nDiff = 10, model = model, criterion = "MUKV", plotOptim = TRUE,
            nGridCells = 10000,nTry = 100 )

# delete optimally 10 stations from current network with method "manual"
if (interactive()) optimMAN = optimizeNetwork(meuse, meuse.grid, candidates, method = "manual",
          action = "del", nDiff = 10, model = model, criterion = "MUKV", plotOptim = TRUE )


# comparison of results with ordinary kriging variogram, otherwise add formulaString
# ssa method, assuming ordinary kriging
calculateMukv(optimOK, predGrid, vfitOK)

# ssa method, using spatial location as covariates
calculateMukv(optimUK, predGrid, vfitUK, zinc~x+y)

# ssa method, using other variables as covariates
calculateMukv(optimRK, predGrid, vfitRK, zinc~dist+ffreq+soil)

# spcov method
calculateMukv(optimSC, predGrid, vfitOK)

# 10 stations manually deleted
if (interactive()) calculateMukv(optimMAN, predGrid, vfitOK, zinc~1)
```

---

spCovAdd *Spatial coverage method to add new measurements*

Description

This function spCovAdd allows to build optimization scenarios based on spatial coverage method.

Usage

```
spCovAdd( observations, candidates, nDiff, nGridCells, plotOptim = TRUE, nTry, ... )
```

Arguments

observations	object of class data.frame with x,y coordinates
candidates	a SpatialPolygonsDataFrame to explore: in use when optimizing the implementation of new measurement stations to an existing network
nDiff	number of stations to add or delete
nGridCells	number of grid cells to work on spatial coverage strafication
plotOptim	logical; to plot the result or not
nTry	the method will try nTry initial configurations and will keep the best solution in order to reduce the risk of ending up with an unfavorable solution
...	other arguments to be passed on at lower level functions such as stratify

Details

This function allows to build optimization scenarios based on spatial coverage method. The scenario action is "add". To add new measurement locations to the running network, the function uses function `stratify` from package `spcosa`. Function stratify adds new strata to the domain study.

Value

`data.frame` of optimized locations

Author(s)

Olivier Baume

References

D. J. Brus, J. de Gruijter, J. van Groenigen (2006). Designing spatial coverage samples using the k-means clustering algorithm. In A. McBratney M. Voltz and P. Lagacherie, editor, Digital Soil Mapping: An Introductory Perspective, Developments in Soil Science, vol. 3., Elsevier, Amsterdam.

`spCovDel` *Optimize the network with spatial coverage methods*

Description

The function spCovDel allows to build optimization scenarios based on spatial coverage method.

Usage

```
spCovDel(observations, candidates, nDiff, plotOptim = TRUE, ...)
```

Arguments

`observations`	object of class `data.frame` with x,y coordinates
`candidates`	not compulsory used only for plotting purpose – a `SpatialPolygonsDataFrame` describing the study area
`nDiff`	number of stations to add or delete
`plotOptim`	logical; to plot the result or not
`...`	other arguments to be passed on at lower level functions such as `nndist`

Details

This function allows to build optimization scenarios based on spatial coverage method. When action is "del", the function maximizes the mean distance of measurements with direct neighbours using function `nndist`. The heuristic search uses a swapping algorithm to converge more rapidly to the best solution.

Value

data.frame of optimized locations

Author(s)

Olivier Baume

---

ssaOptim *Spatial simulated annealing (SSA) for optimization of sampling designs using a geostatistical measure of spatial prediction error*

Description

Spatial simulated annealing uses slight perturbations of previous sampling designs and a random search technique to solve spatial optimization problems. Candidate measurement locations are iteratively moved around and optimized by minimizing the mean universal kriging variance (calculateMukv). The approach relies on a known, pre-specified model for underlying spatial variation (variogramModel).

Usage

```
ssaOptim(observations, predGrid, candidates, action, nDiff, model,
         nr_iterations, plotOptim = TRUE, formulaString = NULL,
         coolingFactor = nr_iterations/10, covariates = "over", fun, ...)
```

Arguments

observations	object of class Spatial with coordinates and possible covariates
predGrid	object of class Spatial* used when method = "ssa". predGrid should contain the coordinates of prediction locations for optimization. Usually predGrid is a SpatialGrid / SpatialPixels or a SpatialGridDataFrame / SpatialPixelsDataFrame when independent covariate predictor variables are used
candidates	candidates is the study area of class SpatialPolygonsDataFrame
action	character string indicating which type of action to perform: "add" to add new measurement stations to the existing network or "del" to delete existing stations
nDiff	number of stations to add or delete
model	variogram model:object of class variogramModel, as generated by vgm
nr_iterations	number of iterations to process before stopping. The default coolingFactor is also a function of number of iterations.
plotOptim	logical; if TRUE, creates a plot of the result as optimization progresses; TRUE by default

formulaString	formula that defines the dependent variable as a linear model of independent variables; suppose the dependent variable has name z, for ordinary and simple kriging use the formula z~1; for universal kriging, suppose z is linearly dependent on x and y, use the formula z~x+y. The formulaString defaults to "value~1" if value is a part of the data set. If not, the first column of the data set is used.
coolingFactor	variable defining how fast the algorithm will cool down. With SSA, worsening designs are accepted with a decreasing probability (generally set to p $<$ 0.2 to avoid selection of local minima). The coolingFactor dictates the rate at which p decreases to zero. Commonly p is set to exponentially decrease or cool as a function of number of iterations to ensure convergence (Brus & Heuvelink 2007, Melles et al. 2011). Smaller numbers give quicker cooling; higher numbers give slower cooling.
covariates	character string defining whether possible covariates should be found by "over" or "krige", see also details below
fun	Alternative objective function for optimization, the input and output should match the ones of (calculateMukv (except for fun)
...	other arguments to be passed on to lower level functions

Details

The default version of this function applies spatial simulated annealing for optimization with the MUKV criterion (calculateMukv). With covariates, the function takes a universal kriging model into account. When optimizing a sampling design using SSA and criterion = "mukv", measurement values at new sampling locations are not required in order to calculate prediction error variance (criterion = "mukv"). This attractive property allows one to estimate the kriging prediction error variance prior to collecting the data (i.e., the dependent variable is unknown at new candidate locations), and it is this property that is used in the SSA optimization procedure after (Brus & Heuvelink 2007, Melles et al. 2011).

A stopping criterion countMax is implemented in lower level functions to end the optimization procedure after 200 search steps have occurred without an improvement in the design. If this stopping criterion is reached before nr_iterations, SSA will terminate.

method = "ssa" with criterion = "mukv" makes it possible to assume a linear relationship between independent variables in predGrid and dependent variables at observation locations using universal kriging (krige). However, a correct estimate of mean universal kriging variance requires that the independent covariate variables be known at candidate locations. Thus it is necessary to have complete spatial coverage for all covariate predictors in the linear model. Covariate information must be available at both new candidate measurement locations and prediction locations. This is not possible (except for the measurement locations themselves). Instead, these are estimated from the prediction locations.

There are two possible methods to attain information on covariates at the candidate locations, and the method can be set using the argument covariates: over and krige. over finds the value of covariates at new locations by overlaying candidate locations on the prediction grid and taking the value of the nearest neighbour. The second method uses kriging to estimate covariate values at new locations from predGrid. The first method is generally faster, the second method is most likely more exact, particularly if the resolution of predGrid is low in relation to the spatial correlation

lengths of the covariates. Both methods are approximations that may influence the criterion used for optimization with increasing numbers of points added.

It is possible to submit an alternative function fun as objective function. This function should take at least the observation locations and the predGrid as input, and return a value which should be minimized. See also calculateMukv for more information about arguments to this function.

Value

SpatialPointsDataFrame with optimized locations

Author(s)

O. Baume, S.J. Melles, J. Skoien

References

D. J. Brus, G. B. M. Heuvelink (2007). Optimization of sample patterns for universal kriging of environmental variables, Geoderma, 138: 86-95 (2007).

S. J. Melles, G. B. M. Heuvelink, C. J. W. Twenhofel, U. Stohlker (2011). Optimizing the spatial pattern of networks for monitoring radioactive releases, Computers and Geosciences, 37: 280-288 (2011).

Examples

```
# load data:
library(gstat)
data(meuse)
coordinates(meuse) = ~x+y
data(meuse.grid)
coordinates(meuse.grid) = ~x+y
gridded(meuse.grid) = TRUE
predGrid = meuse.grid

# estimate variograms (OK/UK):
vfitOK = fit.variogram(variogram(zinc~1, meuse), vgm(1, "Exp", 300, 1))
vfitUK = fit.variogram(variogram(zinc~x+y, meuse), vgm(1, "Exp", 300, 1))
vfitRK = fit.variogram(variogram(zinc~dist+ffreq+soil, meuse), vgm(1, "Exp", 300, 1))

# study area of interest:
bb = bbox(predGrid)
boun = SpatialPoints(data.frame(x=c(bb[1,1],bb[1,2],bb[1,2],bb[1,1],bb[1,1]),
                                y=c(bb[2,1],bb[2,1],bb[2,2],bb[2,2],bb[2,1])))
Srl = Polygons(list(Polygon(boun)),ID = as.character(1))
candidates = SpatialPolygonsDataFrame(SpatialPolygons(list(Srl)),
                                      data = data.frame(ID=1))

# add 20 more points assuming OK model (SSA method):
optimOK <- ssaOptim(meuse, meuse.grid, candidates = candidates, covariates = "over",
            nDiff = 20, action = "add", model = vfitOK, nr_iterations = 10000,
            formulaString = zinc~1, nmax = 40, countMax = 200)
```

```
# add 20 more points assuming UK model (SSA method):
optimUK <- ssaOptim(meuse, meuse.grid, candidates = candidates, covariates = "over",
            nDiff = 20, action = "add", model = vfitUK, nr_iterations = 10000,
            formulaString = zinc~x+y, nmax = 40, countMax = 200)

# add 20 more points with auxiliary variables (SSA method):
optimRK <- ssaOptim(meuse, meuse.grid, candidates = candidates, covariates = "over",
            nDiff = 20, action = "add", model = vfitRK, nr_iterations = 10000,
            formulaString = zinc~dist+ffreq+soil, nmax = 40, countMax = 200)
```

Index