Package ‘excursions’

February 26, 2020

Type Package

Title Excursion Sets and Contour Credibility Regions for Random Fields

Version 2.4.5

Date 2020-02-24

Description Functions that compute probabilistic excursion sets, contour credibility regions, contour avoiding regions, and simultaneous confidence bands for latent Gaussian random processes and fields. The package also contains functions that calculate these quantities for models estimated with the INLA package. The main references for excursions are Bolin and Lindgren (2015) <doi:10.1111/rssb.12055>, Bolin and Lindgren (2017) <doi:10.1080/10618600.2016.1228537>, and Bolin and Lindgren (2018) <doi:10.18637/jss.v086.i05>. These can be generated by the citation function in R.

Depends R (>= 3.2.0), Matrix, sp

Suggests INLA (>= 0.0-1468840039), testthat, rgeos

Imports graphics, methods, stats

Additional_repositories https://inla.r-inla-download.org/R/stable

License GPL (>= 3)

Copyright The R package and code, and the main programs, were written by and are Copyright by David Bolin and Finn Lindgren, and are redistributable under the GNU Public License, version 3 or later. The package also includes code from the libraries CAMD from the SuiteSparse collection of Tim Davis, as well as functions from the GNU Scientific library and the RngStreams library by Pierre L'Ecuyer. For details see the COPYRIGHTS file.

NeedsCompilation yes

RoxygenNote 7.0.2

Author David Bolin [cre, aut], Finn Lindgren [aut]

Maintainer David Bolin <davidbolin@gmail.com>

Repository CRAN

Date/Publication 2020-02-26 13:10:06 UTC
**Description**

`excursions` is one of the main functions in the package with the same name. The function is used for calculating excursion sets, contour credible regions, and contour avoiding sets for latent Gaussian models. Details on the function and the package are given in the sections below.

**Usage**

```r
excursions(
  alpha,
  u,
  mu,
  Q,
  type,
  n.iter = 10000,
  Q.chol,
  F.limit,
  vars,
  rho,
  reo,
  method = "EB",
```

---

**R topics documented:**

- `excursions-package` ............................................. 2
- `continuous` ..................................................... 6
- `contourmap` .................................................... 8
- `contourmap.colors` ............................................. 11
- `contourmap.inla` ............................................. 12
- `contourmap.mc` ............................................... 15
- `excursions.inla` ............................................. 17
- `excursions.mc` .............................................. 20
- `excursions.variances` ....................................... 22
- `gaussint` ..................................................... 23
- `require.nowarnings` .......................................... 26
- `simconf` ...................................................... 27
- `simconf.inla` ................................................. 28
- `simconf.mc` .................................................... 31
- `simconf.mixture` ............................................. 32
- `submesh.grid` ................................................. 34
- `submesh.mesh` ................................................ 35
- `summary.excurobj` .......................................... 36
- `tricontour` ................................................... 37

---

**Index**

| excursions-package | Excursion Sets and Contour Credibility Regions for Random Fields | 41 |
Arguments

- **alpha**: Error probability for the excursion set.
- **u**: Excursion or contour level.
- **mu**: Expectation vector.
- **Q**: Precision matrix.
- **type**: Type of region:
  - '>' positive excursion region
  - '<' negative excursion region
  - '!=' contour avoiding region
  - '=' contour credibility region
- **n.iter**: Number or iterations in the MC sampler that is used for approximating probabilities. The default value is 10000.
- **Q.chol**: The Cholesky factor of the precision matrix (optional).
- **F.limit**: The limit value for the computation of the F function. F is set to NA for all nodes where F<1-F.limit. Default is F.limit = alpha.
- **vars**: Precomputed marginal variances (optional).
- **rho**: Marginal excursion probabilities (optional). For contour regions, provide \( P(X > u) \).
- **reo**: Reordering (optional).
- **method**: Method for handling the latent Gaussian structure:
  - 'EB' Empirical Bayes (default)
  - 'QC' Quantile correction, rho must be provided if QC is used.
- **ind**: Indices of the nodes that should be analysed (optional).
- **max.size**: Maximum number of nodes to include in the set of interest (optional).
- **verbose**: Set to TRUE for verbose mode (optional).
- **max.threads**: Decides the number of threads the program can use. Set to 0 for using the maximum number of threads allowed by the system (default).
- **seed**: Random seed (optional).

Details

The estimation of the region is done using sequential importance sampling with **n.iter** samples. The procedure requires computing the marginal variances of the field, which should be supplied if available. If not, they are computed using the Cholesky factor of the precision matrix. The cost of this step can therefore be reduced by supplying the Cholesky factor if it is available.
The latent structure in the latent Gaussian model can be handled in several different ways. The default strategy is the EB method, which is exact for problems with Gaussian posterior distributions. For problems with non-Gaussian posteriors, the QC method can be used for improved results. In order to use the QC method, the true marginal excursion probabilities must be supplied using the argument \( \rho \). Other more complicated methods for handling non-Gaussian posteriors must be implemented manually unless INLA is used to fit the model. If the model is fitted using INLA, the method `excursions.inla` can be used. See the Package section for further details about the different options.

**Value**

`excursions` returns an object of class "excurobj". This is a list that contains the following arguments:

- **E** Excursion set, contour credible region, or contour avoiding set
- **G** Contour map set. \( G = 1 \) for all nodes where the \( \mu > u \).
- **M** Contour avoiding set. \( M = -1 \) for all non-significant nodes. \( M = 0 \) for nodes where the process is significantly below \( u \) and \( M = 1 \) for all nodes where the field is significantly above \( u \). Which values that should be present depends on what type of set that is calculated.
- **F** The excursion function corresponding to the set \( E \) calculated or values up to \( F \).lim
- **rho** Marginal excursion probabilities
- **mean** The mean \( \mu \).
- **vars** Marginal variances.
- **meta** A list containing various information about the calculation.

**Package**

`excursions` contains functions that compute probabilistic excursion sets, contour credibility regions, contour avoiding regions, contour map quality measures, and simultaneous confidence bands for latent Gaussian random processes and fields.

**Excursion sets, contour credibility regions, and contour avoiding regions**

The main functions for computing excursion sets, contour credibility regions, and contour avoiding regions are

- `excursions` The main function for Gaussian models.
- `excursions.inla` Interface for latent Gaussian models estimated using INLA.
- `excursions.mc` Function for analyzing models that have been estimated using Monte Carlo methods.

The output from the functions above provides a discrete domain estimate of the regions. Based on this estimate, the function `continuous` computes a continuous domain estimate.


**Contour map quality measures**
The package provides several functions for computing contour maps and their quality measures. These quality measures can be used to decide on an appropriate number of contours to use for the contour map.

The main functions for computing contour maps and the corresponding quality measures are

- `contourmap` The main function for Gaussian models.
- `contourmap.inla` Interface for latent Gaussian models estimated using INLA.
- `contourmap.mc` Function for analyzing models that have been estimated using Monte Carlo methods.

Other noteworthy functions relating to contour maps are `tricontour` and `tricontourmap`, which compute contour curves for functions defined on triangulations, as well as `contourmap.colors` which can be used to compute appropriate colors for displaying contour maps.


**Simultaneous confidence bands**

The main functions for computing simultaneous confidence bands are

- `simconf` Function for analyzing Gaussian models.
- `simconf.inla` Function for analyzing latent Gaussian models estimated using INLA.
- `simconf.mc` Function for analyzing models estimated using Monte Carlo methods.
- `simconf.mixture` Function for analyzing Gaussian mixture models.


**Author(s)**

David Bolin <davidbolin@gmail.com> and Finn Lindgren <finn.lindgren@gmail.com>

**References**


**See Also**

`excursions.inla`, `excursions.mc`

**Examples**

```r
## Create a tridiagonal precision matrix
n = 21
Q.x = sparseMatrix(i=c(1:n, 2:n), j=c(1:n, 1:(n-1)), x=c(rep(1, n), rep(-0.1, n-1)),
dims=c(n, n), symmetric=TRUE)
## Set the mean value function
```
mu.x = seq(-5, 5, length=n)

## calculate the level 0 positive excursion function
res.x = excursions(alpha=1, u=0, mu=mu.x, Q=Q.x,
    type='>', verbose=1, max.threads=2)

## Plot the excursion function and the marginal excursion probabilities
plot(res.x$F, type="l",
    main="Excursion function (black) and marginal probabilities (red)'")
lines(res.x$rho, col=2)

---

**continuous**  
*Calculate continuous domain excursion and credible contour sets*

### Description

Calculates continuous domain excursion and credible contour sets

### Usage

```r
continuous(
    ex,
    geometry,
    alpha,
    method = c("log", "linear", "step"),
    output = c("sp", "inla"),
    subdivisions = 1,
    calc.credible = TRUE
)
```

### Arguments

- **ex**: An excurobj object generated by a call to `excursions` or `contourmap`.
- **geometry**: Specification of the lattice or triangulation geometry of the input. One of `list(x,y)`, `list(loc,dims)`, `inla.mesh.lattice`, or `inla.mesh`, where `x` and `y` are vectors, `loc` is a two-column matrix of coordinates, and `dims` is the lattice size vector. The first three versions are all treated topologically as lattices, and the lattice boxes are assumed convex.
- **alpha**: The target error probability. A warning is given if it is detected that the information `ex` isn’t sufficient for the given `alpha`. Defaults to the value used when calculating `ex`.
- **method**: The spatial probability interpolation transformation method to use. One of `log`, `linear`, or `step`. For `log`, the probabilities are interpolated linearly in the transformed scale. For `step`, a conservative step function is used.
- **output**: Specifies what type of object should be generated. `sp` gives a SpatialPolygons object, and `inla` gives an `inla.mesh.segment` object.
subdivisions

The number of mesh triangle subdivisions to perform for the interpolation of the excursions or contour function. 0 is no subdivision. The setting has a small effect on the evaluation of $P0$ for the log method (higher values giving higher accuracy) but the main effect is on the visual appearance of the interpolation. Default=1.

calc.credible

Logical, if TRUE (default), calculate credible contour region objects in addition to avoidance sets.

Value

A list:

- **M**: SpatialPolygons or inla.mesh.segment object. The subsets are tagged, so that credible regions are tagged "-1", and regions between levels are tagged as.character(0:nlevels).
- **F**: Interpolated F function.
- **G**: Contour and inter-level set indices for the interpolation.
- **F.geometry**: Mesh geometry for the interpolation.
- **P0**: $P0$ measure based on interpolated F function (only for contourmap input).

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

References


Examples

```r
if (require.nowarnings("INLA")) {
  #Generate mesh and SPDE model
  n.lattice = 10 #Increase for more interesting, but slower, examples
  x=seq(from=0,to=10,length.out=n.lattice)
  lattice=inla.mesh.lattice(x=x,y=x)
  mesh=inla.mesh.create(lattice=lattice, extend=FALSE, refine=FALSE)
  spde <- inla.spde2.matern(mesh, alpha=2)

  #Generate an artificial sample
  sigma2.e = 0.01
  n.obs=100
  obs.loc = cbind(runif(n.obs)*diff(range(x))+min(x),
                  runif(n.obs)*diff(range(x))+min(x))
  Q = inla.spde2.precision(spde, theta=c(log(sqrt(0.5)), log(sqrt(1))))
  x = inla.qsample(Q=Q)
```
\begin{verbatim}
A = inla.spde.make.A(mesh=mesh,loc=obs.loc)
Y = as.vector(A %*% x + rnorm(n.obs)*sqrt(sigma2.e))

## Calculate posterior
Q.post = (Q + (t(A) %*% A)/sigma2.e)
mu.post = as.vector(solve(Q.post,(t(A) %*% Y)/sigma2.e))
vars.post = excursions.variances(chol(Q.post))

## Calculate contour map with two levels
map = contourmap(n.levels = 2, mu = mu.post, Q = Q.post,
alpha=0.1, F.limit = 0.1,max.threads=1)

## Calculate the continuous representation
sets <- continuous(map, mesh, alpha=0.1)

## Plot the results
reo = mesh$idx$lattice
cols = contourmap.colors(map, col=heat.colors(100, 1),
credible.col = grey(0.5, 1))
names(cols) = as.character(-1:2)

par(mfrow = c(2,2))
image(matrix(mu.post[reo],n.lattice,n.lattice),
main="mean",axes=FALSE)
image(matrix(sqrt(vars.post[reo]),n.lattice,n.lattice),
main="sd", axes = FALSE)
image(matrix(map$M[reo],n.lattice,n.lattice),col=cols,axes=FALSE)
idx.M = setdiff(names(sets$M), "-1")
plot(sets$M[idx.M], col=cols[idx.M])
\end{verbatim}

---

**contourmap**

Contour maps and contour map quality measures for latent Gaussian models

**Description**

contourmap is used for calculating contour maps and quality measures for contour maps for Gaussian models.

**Usage**

```r
contourmap(
  mu,
  Q,
  vars,
  n.levels,
  ind,
  levels,
  type = c("standard", "pretty", "equalarea", "P0-optimal", "P1-optimal", "P2-optimal"),
)```
compute = list(F = TRUE, measures = NULL),
use.marginals = TRUE,
alpha,
F.limit,
n.iter = 10000,
verbose = FALSE,
max.threads = 0,
seed = NULL
)

Arguments

mu Expectation vector.
Q Precision matrix.
vars Precomputed marginal variances (optional).
n.levels Number of levels in contour map.
ind Indices of the nodes that should be analyzed (optional).
levels Levels to use in contour map.
type Type of contour map. One of:
  • 'standard' Equidistant levels between smallest and largest value of the posterior mean (default).
  • 'pretty' Equally spaced 'round' values which cover the range of the values in the posterior mean.
  • 'equalarea' Levels such that different spatial regions are approximately equal in size.
  • 'P0-optimal' Levels chosen to maximize the P0 measure.
  • 'P1-optimal' Levels chosen to maximize the P1 measure.
  • 'P2-optimal' Levels chosen to maximize the P2 measure.
compute A list with quality indices to compute
  • 'F': TRUE/FALSE indicating whether the contour map function should be computed (default TRUE).
  • 'measures': A list with the quality measures to compute ("P0", "P1", "P2") or corresponding bounds based only on the marginal probabilities ("P0-bound", "P1-bound", "P2-bound").
use.marginals Only marginal distributions are used when finding P-optimal maps (default TRUE).
alpha Maximal error probability in contour map function (default=1).
F.limit The limit value for the computation of the F function. F is set to NA for all nodes where F<1-F.limit. Default is F.limit = alpha.
n.iter Number or iterations in the MC sampler that is used for calculating the quantities in compute. The default value is 10000.
verbose Set to TRUE for verbose mode (optional).
max.threads Decides the number of threads the program can use. Set to 0 for using the maximum number of threads allowed by the system (default).
seed Random seed (optional).
Details

The Gaussian model is specified using the mean \textit{mu} and the precision matrix \textit{Q}. The contour map is then computed for the mean, using either the contour levels specified in \textit{levels}, or \textit{n.levels} contours that are placed according to the argument type.

A number of quality measures can be computed based on the specified contour map and the Gaussian distribution. What should be computed is specified using the compute argument. For details on these quantities, see the reference below.

Value

\texttt{contourmap} returns an object of class "excurobj". This is a list that can contains the following arguments:

- \texttt{u} Contour levels used in the contour map.
- \texttt{n.levels} The number of contours used.
- \texttt{u.e} The values associated with the level sets \textit{G}_k.
- \texttt{G} A vector which shows which of the level sets \textit{G}_k each node belongs to.
- \texttt{map} Representation of the contour map with map[i]=u.e[k] if i is in \textit{G}_k.
- \texttt{F} The contour map function (if computed).
- \texttt{M} Contour avoiding sets (if \texttt{F} is computed). \texttt{M} = -1 for all non-significant nodes and \texttt{M} = \texttt{k} for nodes that belong to \textit{M}_k.
- \texttt{P0/P1/P2} Calculated quality measures (if computed).
- \texttt{P0bound/P1bound/P2bound} Calculated upper bounds quality measures (if computed).
- \texttt{meta} A list containing various information about the calculation.

Author(s)

David Bolin <davidbolin@gmail.com>

References


See Also

\texttt{contourmap.inla, contourmap.mc, contourmap.colors}
Examples

n = 10
Q = Matrix(toeplitz(c(1, -0.5, rep(0, n-2))))
mu = seq(-5, 5, length=n)
lp <- contourmap(mu, Q, n.levels = 2,
    compute=list(F=FALSE, measures = c("P1","P2"),
    max.threads=1)
#Plot the contourmap
plot(lp$map)
#Display the quality measures
cat(c(lp$P1,lp$P2))

contourmap.colors

Define a color map for displaying contour maps.

Description

contourmap.colors calculates suitable colours for displaying contour maps.

Usage

contourmap.colors(lp, zlim, col, credible.col)

Arguments

lp  A contourmap calculated by contourmap, contourmap.inla, or contourmap.mc
zlim The range that should be used (optional). The default is the range of the mean value function used when creating the contourmap.
col The colormap that the colours should be taken from.
credible.col The color that should be used for displaying the credible regions for the contour curves (optional).

Value

A color map.

Author(s)

David Bolin <davidbolin@gmail.com>

Examples

n = 10
Q = Matrix(toeplitz(c(1, -0.5, rep(0, n-2))))
map <- contourmap(mu = seq(-5, 5, length=n), Q, n.levels = 2,
    compute=list(F=FALSE), max.threads=1)
cols = contourmap.colors(map, col=heat.colors(100, 1),
    credible.col = grey(0.5, 1))
contourmap.inla

Contour maps and contour map quality measures for latent Gaussian models

Description

An interface to the contourmap function for latent Gaussian models calculated using the INLA method.

Usage

contourmap.inla(
    result.inla,
    stack,
    name = NULL,
    tag = NULL,
    method = "QC",
    n.levels,
    type = c("standard", "pretty", "equalarea"),
    compute = list(F = TRUE, measures = NULL),
    alpha,
    F.limit,
    n.iter = 10000,
    verbose = FALSE,
    max.threads = 0,
    seed = NULL,
    ind,
    ...
)

Arguments

result.inla Result object from INLA call.
stack The stack object used in the INLA call.
name The name of the component for which to do the calculation. This argument should only be used if a stack object is not provided, use the tag argument otherwise.
tag The tag of the component in the stack for which to do the calculation. This argument should only be used if a stack object is provided, use the name argument otherwise.
method Method for handling the latent Gaussian structure. Currently only Empirical Bayes (EB) and Quantile corrections (QC) are supported.
n.levels Number of levels in contour map.
type Type of contour map. One of:
• 'standard' Equidistant levels between smallest and largest value of the posterior mean (default).
• 'pretty' Equally spaced 'round' values which cover the range of the values in the posterior mean.
• 'equalarea' Levels such that different spatial regions are approximately equal in size.

compute A list with quality indices to compute
• 'F': TRUE/FALSE indicating whether the contour map function should be computed (default TRUE)
• 'measures': A list with the quality measures to compute ("P0", "P1", "P2") or corresponding bounds based only on the marginal probabilities ("P0-bound", "P1-bound", "P2-bound")

alpha Maximal error probability in contour map function (default=1)
F.limit The limit value for the computation of the F function. F is set to NA for all nodes where F<1-F.limit. Default is F.limit = alpha.
n.iter Number or iterations in the MC sampler that is used for calculating the quantities in compute. The default value is 10000.
verbose Set to TRUE for verbose mode (optional)
max.threads Decides the number of threads the program can use. Set to 0 for using the maximum number of threads allowed by the system (default).
seed Random seed (optional).
ind If only a part of a component should be used in the calculations, this argument specifies the indices for that part (optional).
...
Additional arguments to the contour map function. See the documentation for contourmap for details.

Details
The INLA approximation of the quantity of interest is in general a weighted sum of Gaussian distributions with different parameters. If method = 'EB' is used, then the contour map is computed for the mean of the component in the weighted sum that has parameters with the highest likelihood. If on the other hand method='QC', then the contour map is computed for the posterior mean reported by INLA. If the EB method also is used in INLA, then this reported posterior mean is equal to the mean of the component with the highest likelihood. Therefore, method='EB' is appropriate if the EB method also is used in INLA, but method='QC' should be used in general.

The n.levels contours in the contour map are are placed according to the argument type. A number of quality measures can be computed based based on the specified contour map and the distribution of the component of interest. What should be computed is specified using the compute argument. For details on these quantities, see the reference below.

Value
contourmap.inla returns an object of class "excurobj" with the same elements as returned by contourmap.
contourmap.inla

Note

This function requires the INLA package, which is not a CRAN package. See http://www.r-inla.org/download for easy installation instructions.

Author(s)

David Bolin <davidbolin@gmail.com>

References


See Also

contourmap, contourmap.mc, contourmap.colors

Examples

```r
if (require.nowarnings("INLA")) {
# Generate mesh and SPDE model
n.lattice <- 10 # increase for more interesting, but slower, examples
x <- seq(from = 0, to = 10, length.out = n.lattice)
lattice <- inla.mesh.lattice(x = x, y = x)
mesh <- inla.mesh.create(lattice = lattice, extend = FALSE, refine = FALSE)
spde <- inla.spde2.matern(mesh, alpha = 2)
# Generate an artificial sample
sigma2.e = 0.01
n.obs=100
obs.loc = cbind(runif(n.obs)*diff(range(x))+min(x),
                runif(n.obs)*diff(range(x))+min(x))
Q = inla.spde2.precision(spde, theta=c(log(sqrt(0.5)), log(sqrt(1))))
x = inla.qsample(Q=Q)
A = inla.spde.make.A(mesh=mesh, loc=obs.loc)
Y = as.vector(A %% x + rnorm(n.obs) * sqrt(sigma2.e))

## Estimate the parameters using INLA
mesh.index = inla.spde.make.index(name="field",n.spde=spde$n.spde)
ef = list(c(mesh.index,list(Intercept=1)))
s.obs = inla.stack(data=list(y=Y), A=list(A), effects=ef, tag="obs")
s.pre = inla.stack(data=list(y=NA), A=list(1), effects=ef,tag="pred")
stack = inla.stack(s.obs,s.pre)
formula = y ~ -1 + Intercept + f(field, model=spde)
result = inla(formula=formula, family="normal", data = inla.stack.data(stack),
             control.predictor=list(A=inla.stack.A(stack),compute=TRUE),
             control.compute = list(config = TRUE),
             num.threads = 1)
}
```
## Calculate contour map with two levels
map = contourmap.inla(result, stack = stack, tag = 'pred',
        n.levels = 2, alpha=0.1, F.limit = 0.1,
        max.threads = 1)

## Plot the results
cols = contourmap.colors(map, col=heat.colors(100, 1),
        credible.col = grey(0.5, 1))
image(matrix(map$M[mesh$idx$lattice], n.lattice, n.lattice), col = cols)

---

### contourmap.mc

**Contour maps and contour map quality measures using Monte Carlo samples**

**Description**

*contourmap.mc* is used for calculating contour maps and quality measures for contour maps based on Monte Carlo samples of a model.

**Usage**

```r
contourmap.mc(
    samples,
    n.levels,
    ind,
    levels,
    type = c("standard", "equalarea", "P0-optimal", "P1-optimal", "P2-optimal"),
    compute = list(F = TRUE, measures = NULL),
    alpha,
    verbose = FALSE
)
```

**Arguments**

- **samples**: Matrix with model Monte Carlo samples. Each column contains a sample of the model.
- **n.levels**: Number of levels in contour map.
- **ind**: Indices of the nodes that should be analyzed (optional).
- **levels**: Levels to use in contour map.
- **type**: Type of contour map. One of:
  - 'standard': Equidistant levels between smallest and largest value of the posterior mean (default).
  - 'pretty': Equally spaced 'round' values which cover the range of the values in the posterior mean.
• 'equalarea' Levels such that different spatial regions are approximately equal in size.
• 'P0-optimal' Levels chosen to maximize the P0 measure.
• 'P1-optimal' Levels chosen to maximize the P1 measure.
• 'P2-optimal' Levels chosen to maximize the P2 measure.

compute A list with quality indices to compute

• 'F': TRUE/FALSE indicating whether the contour map function should be computed (default TRUE).
• 'measures': A list with the quality measures to compute ("P0", "P1", "P2") or corresponding bounds based only on the marginal probabilities ("P0-bound", "P1-bound", "P2-bound").

alpha Maximal error probability in contour map function (default=0.1).
verbose Set to TRUE for verbose mode (optional).

Details

The contour map is computed for the empirical mean of the samples. See contourmap and contourmap.inla for further details.

Value

contourmap returns an object of class "excurobj". This is a list that can contains the following arguments:

u Contour levels used in the contour map.
n.levels The number of contours used.
u.e The values associated with the level sets G_k.
G A vector which shows which of the level sets G_k each node belongs to.
map Representation of the contour map with map[i]=u.e[k] if i is in G_k.
F The contour map function (if computed).
M Contour avoiding sets (if F is computed). \( M = -1 \) for all non-significant nodes and \( M = k \) for nodes that belong to \( M_k \).
P0/P1/P2 Calculated quality measures (if computed).
P0bound/P1bound/P2bound Calculated upper bounds quality measures (if computed).
meta A list containing various information about the calculation.

Author(s)

David Bolin <davidbolin@gmail.com>

References


See Also
contourmap, contourmap.inla, contourmap.colors

Examples

n = 100
Q = Matrix(toeplitz(c(1, -0.5, rep(0, n-2))))
mu = seq(-5, 5, length=n)
## Sample the model 100 times (increase for better estimate)
X = mu + solve(chol(Q), matrix(rnorm(n=n*100), nrow=n, ncol=100))

lp <- contourmap.mc(X, nlevels = 2, compute=list(F=FALSE, measures = c("P1","P2")))

# plot contourmap
plot(lp$map)
# display quality measures
c(lp$P1, lp$P2)

Description
Excursion sets and contour credible regions for latent Gaussian models calculated using the INLA method.

Usage

excursions.inla(
    result.inla, stack, name = NULL, tag = NULL, ind = NULL,
    method, alpha = 1, F.limit, u, u.link = FALSE, type,
    n.iter = 10000, verbose = 0, max.threads = 0, seed = NULL
)
Arguments

result.inla  Result object from INLA call.
stack        The stack object used in the INLA call.
name         The name of the component for which to do the calculation. This argument
             should only be used if a stack object is not provided, use the tag argument otherwise.
tag          The tag of the component in the stack for which to do the calculation. This ar-
             gument should only be used if a stack object is provided, use the name argument
             otherwise.
ind          If only a part of a component should be used in the calculations, this argument
             specifies the indices for that part.
method       Method for handling the latent Gaussian structure:
             • 'EB' Empirical Bayes
             • 'QC' Quantile correction
             • 'NI' Numerical integration
             • 'NIQC' Numerical integration with quantile correction
             • 'iNIQC' Improved integration with quantile correction
alpha        Error probability for the excursion set of interest. The default value is 1.
F.limit      Error probability for when to stop the calculation of the excursion function. The
             default value is alpha, and the value cannot be smaller than alpha. A smaller
             value of F.limit results in a smaller computation time.
u            Excursion or contour level.
u.link       If u.link is TRUE, u is assumed to be in the scale of the data and is then trans-
             formed to the scale of the linear predictor (default FALSE).
type         Type of region:
             • '>' positive excursions
             • '<' negative excursions
             • '!=' contour avoiding function
             • '=' contour credibility function
n.iter        Number of iterations in the MC sampler that is used for approximating proba-
             bilities. The default value is 10000.
verbose      Set to TRUE for verbose mode (optional).
max.threads  Decides the number of threads the program can use. Set to 0 for using the
             maximum number of threads allowed by the system (default).
seed         Random seed (optional).

Details

The different methods for handling the latent Gaussian structure are listed in order of accuracy and
computational cost. The EB method is the simplest and is based on a Gaussian approximation of
the posterior of the quantity of interest. The QC method uses the same Gaussian approximation but
improves the accuracy by modifying the limits in the integrals that are computed in order to find the
region. The other three methods are intended for Bayesian models where the posterior distribution for the quantity of interest is obtained by integrating over the parameters in the model. The NI method approximates this integration in the same way as is done in INLA, and the NIQC and iNIQC methods combine this approximation with the QC method for improved accuracy.

If the main purpose of the analysis is to construct excursion or contour sets for low values of \( \alpha \), we recommend using QC for problems with Gaussian likelihoods and NIQC for problems with non-Gaussian likelihoods. The reason for this is that the more accurate methods also have higher computational costs.

**Value**

`excursions.inla` returns an object of class "excurobj". This is a list that contains the following arguments:

- **E**: Excursion set, contour credible region, or contour avoiding set
- **F**: The excursion function corresponding to the set \( E \) calculated for values up to \( F \).limit
- **G**: Contour map set. \( G = 1 \) for all nodes where the \( \mu > u \).
- **M**: Contour avoiding set. \( M = -1 \) for all non-significant nodes. \( M = 0 \) for nodes where the process is significantly below \( u \) and \( M = 1 \) for all nodes where the field is significantly above \( u \). Which values that should be present depends on what type of set that is calculated.
- **rho**: Marginal excursion probabilities
- **mean**: Posterior mean
- **vars**: Marginal variances
- **meta**: A list containing various information about the calculation.

**Note**

This function requires the INLA package, which is not a CRAN package. See [http://www.r-inla.org/download](http://www.r-inla.org/download) for easy installation instructions.

**Author(s)**

David Bolin <davidbolin@gmail.com> and Finn Lindgren <finn.lindgren@gmail.com>

**References**


**See Also**

`excursions, excursions.mc`
Examples

```r
## In this example, we calculate the excursion function
## for a partially observed AR process.

if (require.nowarnings("INLA")) {
  ## Sample the process:
  rho = 0.9
  tau = 15
  tau.e = 1
  n = 100
  x = 1:n
  mu = 10*((x<n/2)*(x-n/2) + (x>=n/2)*(n/2-x)+n/4)/n
  Q = tau* sparseMatrix(i=c(1:n, 2:n), j=c(1:n, 1:(n-1)),
                         x=c(1, rep(1+rho^2, n-2), 1, rep(-rho, n-1)),
                         dims=c(n, n), symmetric=TRUE)
  X = mu + solve(chol(Q), rnorm(n))

  ## measure the sampled process at n.obs random locations
  ## under Gaussian measurement noise.
  n.obs = 50
  obs.loc = sample(1:n, n.obs)
  A = sparseMatrix(i=1:n.obs, j=obs.loc, x=rep(1, n.obs), dims=c(n.obs, n))
  Y = as.vector(A %*% X + rnorm(n.obs)/sqrt(tau.e))

  ## Estimate the parameters using INLA
  ef = list(c(list(ar=x), list(cov=mu)))
  s.obs = inla.stack(data=list(y=Y), A=list(A), effects=ef, tag="obs")
  s.pre = inla.stack(data=list(y=NA), A=list(1), effects=ef, tag="pred")
  stack = inla.stack(s.obs, s.pre)
  formula = y ~ -1 + cov + f(ar,model="ar1")
  result = inla(formula=formula, family="normal", data = inla.stack.data(stack),
                control.predictor=list(A=inla.stack.A(stack), compute=TRUE),
                control.compute = list(config = TRUE))

  ## calculate the level 0 positive excursion function
  res.qc = excursions.inla(result, stack = stack, tag = 'pred', alpha=0.99, u=0,
                           method='QC', type='>', max.threads=2)

  ## plot the excursion function and marginal probabilities
  plot(res.qc$rho,type='l',
       main="marginal probabilities (black) and excursion function (red)")
  lines(res.qc$F,col=2)
}
```

description

excursions.mc is used for calculating excursion sets, contour credible regions, and contour avoiding sets based on Monte Carlo samples of models.
Usage

```r
excursions.mc(
    samples, alpha, u, type, rho, reo, ind, max.size, verbose = FALSE
)
```

Arguments

- **samples**: Matrix with model Monte Carlo samples. Each column contains a sample of the model.
- **alpha**: Error probability for the excursion set.
- **u**: Excursion or contour level.
- **type**: Type of region:
  - '>' positive excursions
  - '<' negative excursions
  - '!=' contour avoiding function
  - '=' contour credibility function
- **rho**: Marginal excursion probabilities (optional). For contour regions, provide \( P(X > u) \).
- **reo**: Reordering (optional).
- **ind**: Indices of the nodes that should be analysed (optional).
- **max.size**: Maximum number of nodes to include in the set of interest (optional).
- **verbose**: Set to TRUE for verbose mode (optional).

Value

`excursions` returns an object of class "excurobj". This is a list that contains the following arguments:

- **E**: Excursion set, contour credible region, or contour avoiding set.
- **G**: Contour map set. \( G = 1 \) for all nodes where the \( \mu > u \).
- **M**: Contour avoiding set. \( M = -1 \) for all non-significant nodes. \( M = 0 \) for nodes where the process is significantly below \( u \) and \( M = 1 \) for all nodes where the field is significantly above \( u \). Which values that should be present depends on what type of set that is calculated.
- **F**: The excursion function corresponding to the set \( E \) calculated for values up to \( F \text{.limit} \).
excursions.variances

rho  Marginal excursion probabilities
mean  The mean mu.
vars  Marginal variances.
meta  A list containing various information about the calculation.

Author(s)
David Bolin <davidbolin@gmail.com> and Finn Lindgren <finn.lindgren@gmail.com>

References

See Also
excursions, excursions.inla

Examples

```r
## Create mean and a tridiagonal precision matrix
n = 101
mu.x = seq(-5, 5, length=n)
Q.x = Matrix(toeplitz(c(1, -0.1, rep(0, n-2))))
## Sample the model 100 times (increase for better estimate)
X = mu.x + solve(chol(Q.x),matrix(rnorm(n=n*1000),nrow=n,ncol=1000))
## calculate the positive excursion function
res.x = excursions.mc(X,alpha=0.05,type='>',u=0)
## Plot the excursion function and the marginal excursion probabilities
plot(res.x$F, type="l",
     main="Excursion function (black) and marginal probabilites (red)'",
     lines(res.x$F, type="l",
     main="Excursion function (black) and marginal probabilites (red)')
lines(res.x$rho, col=2)
```

excursions.variances  Calculate variances from a sparse precision matrix

Description
excursions.variances calculates the diagonal of the inverse of a sparse symmetric positive definite matrix Q.

Usage

```r
excursions.variances(L, Q, max.threads = 0)
```
Arguments

L  Cholesky factor of precision matrix.
Q  Precision matrix.
max.threads  Decides the number of threads the program can use. Set to 0 for using the maximum number of threads allowed by the system (default).

Details

The method for calculating the diagonal requires the Cholesky factor, L, of Q, which should be supplied if available. If Q is provided, the cholesky factor is calculated and the variances are then returned in the same ordering as Q. If L is provided, the variances are returned in the same ordering as L, even if L@invpivot exists.

Value

A vector with the variances.

Author(s)

David Bolin <davidbolin@gmail.com>

Examples

```r
## Create a tridiagonal precision matrix
n = 21
Q = Matrix(toeplitz(c(1, -0.1, rep(0, n-2))))
v2 = excursions.variances(Q=Q,max.threads=2)
## var2 should be the same as:
v1 = diag(solve(Q))
```

Description

`gaussint` is used for calculating n-dimensional Gaussian integrals

\[
\int_a^b \frac{|Q|^{1/2}}{(2\pi)^{n/2}} \exp\left(-\frac{1}{2}(x - \mu)^T Q (x - \mu)\right) dx
\]

A limit value `lim` can be used to stop the integration if the sequential estimate goes below the limit, which can result in substantial computational savings in cases when one only is interested in testing if the integral is above the limit value. The integral is calculated sequentially, and estimates for all subintegrals are also returned.
Usage

```r
gaussint(
  mu,
  Q.chol,
  Q,
  a,
  b,
  lim = 0,
  n.iter = 10000,
  ind,
  use.reordering = c("natural", "sparsity", "limits"),
  max.size,
  max.threads = 0,
  seed
)
```

Arguments

- **mu**: Expectation vector for the Gaussian distribution.
- **Q.chol**: The Cholesky factor of the precision matrix (optional).
- **Q**: Precision matrix for the Gaussian distribution. If Q is supplied but not Q.chol, the cholesky factor is computed before integrating.
- **a**: Lower limit in integral.
- **b**: Upper limit in integral.
- **lim**: If this argument is used, the integration is stopped and 0 is returned if the estimated value goes below `lim`.
- **n.iter**: Number of iterations in the MC sampler that is used for approximating probabilities. The default value is 10000.
- **ind**: Indices of the nodes that should be analyzed (optional).
- **use.reordering**: Determines what reordering to use:
  - "natural": No reordering is performed.
  - "sparsity": Reorder for sparsity in the cholesky factor (MMD reordering is used).
  - "limits": Reorder by moving all nodes with a=-Inf and b=Inf first and then reordering for sparsity (CAMD reordering is used).
- **max.size**: The largest number of sub-integrals to compute. Default is the total dimension of the distribution.
- **max.threads**: Decides the number of threads the program can use. Set to 0 for using the maximum number of threads allowed by the system (default).
- **seed**: The random seed to use (optional).
gaussint

Details

The function uses sequential importance sampling to estimate the Gaussian integral, and returns all computed sub-integrals. This means that if, for example, the function is used to compute $P(x > 0)$ for an $n$-dimensional Gaussian variable $x$, then all integrals $P(x_1 > 0, ..., x_i > 0)$ for $i = 1, ..., n$ are computed.

If one is only interested in whether $P(x > 0) > \alpha$ or not, then one can stop the integration as soon as $P(x_1 > 0, ..., x_i > 0) < \alpha$. This can save a lot of computation time if $P(x_1 > 0, ..., x_i > 0) < \alpha$ for $i$ much smaller than $n$. This limit value is specified by the `lim` argument.

Which reordering to use depends on what the purpose of the calculation is and what the integration limits are. However, in general the `limits` reordering is typically most appropriate since this combines sparsity (which improves accuracy and reduces computational cost) with automatic handling of dimensions with limits $a = -\infty$ and $b = \infty$, which do not affect the probability but affect the computation time if they are not handled separately.

Value

A list with elements

- $P$ Value of the integral.
- $\mathbf{E}$ Estimated error of the $P$ estimate.
- $\mathbf{Pv}$ A vector with the estimates of all sub-integrals.
- $\mathbf{Ev}$ A vector with the estimated errors of the $\mathbf{Pv}$ estimates.

Author(s)

David Bolin <davidbolin@gmail.com>

References


Examples

```r
## Create mean and a tridiagonal precision matrix
n = 11
mu.x = seq(-5, 5, length=n)
Q.x = Matrix(toeplitz(c(1, -0.1, rep(0, n-2))))
## Calculate the probability that the variable is between mu-3 and mu+3
prob = gaussint(mu=mu.x, Q=Q.x, a=mu.x-3, b=mu.x+3, max.threads=2)
prob$P
```
require.nowarnings  

Warnings free loading of add-on packages

Description

Turn off all warnings for require(), to allow clean completion of examples that require unavailable Suggested packages.

Usage

require.nowarnings(package, lib.loc = NULL, character.only = FALSE)

Arguments

package  
The name of a package, given as a character string.

lib.loc  
a character vector describing the location of R library trees to search through, or NULL. The default value of NULL corresponds to all libraries currently known to .libPaths(). Non-existent library trees are silently ignored.

character.only  
a logical indicating whether package can be assumed to be a character string.

Details

require(package) acts the same as require(package, quietly = TRUE) but with warnings turned off. In particular, no warning or error is given if the package is unavailable. Most cases should use requireNamespace(package, quietly = TRUE) instead, which doesn’t produce warnings.

Value

require.nowarnings returns (invisibly) TRUE if it succeeds, otherwise FALSE

See Also

require

Examples

## This should produce no output:
if (require.nowarnings(nonexistent)) {
  message("Package loaded successfully")
}
simconf

**Simultaneous confidence regions for Gaussian models**

**Description**

`simconf` is used for calculating simultaneous confidence regions for Gaussian models $x$. The function returns upper and lower bounds $a$ and $b$ such that $P(a < x < b) = 1 - \alpha$.

**Usage**

```r
simconf(
  alpha,
  mu,
  Q,
  n.iter = 10000,
  Q.chol,
  vars,
  ind = NULL,
  verbose = 0,
  max.threads = 0,
  seed = NULL
)
```

**Arguments**

- `alpha`: Error probability for the region.
- `mu`: Expectation vector for the Gaussian distribution.
- `Q`: Precision matrix for the Gaussian distribution.
- `n.iter`: Number of iterations in the MC sampler that is used for approximating probabilities. The default value is 10000.
- `Q.chol`: The Cholesky factor of the precision matrix (optional).
- `vars`: Precomputed marginal variances (optional).
- `ind`: Indices of the nodes that should be analyzed (optional).
- `verbose`: Set to TRUE for verbose mode (optional).
- `max.threads`: Decides the number of threads the program can use. Set to 0 for using the maximum number of threads allowed by the system (default).
- `seed`: Random seed (optional).

**Details**

The pointwise confidence bands are based on the marginal quantiles, meaning that $a_{\text{marginal}} = \mu + q_{\alpha}$ and $b_{\text{marginal}} = \mu + q_{1-\alpha}$, where $\mu$ is the mean and $q_{\alpha}$ is a vector with the alpha-quantiles of $x-\mu$.

The simultaneous confidence bands are defined as $a = \mu + c \cdot q_{\alpha}$ and $b = \mu + c \cdot q_{1-\alpha}$, where $c$ is a constant computed such that $P(a < x < b) = 1 - \alpha$. 
Value

An object of class "excurobj" with elements

- **a**
  - The lower bound.
- **b**
  - The upper bound.
- **a.marginal**
  - The lower bound for pointwise confidence bands.
- **b.marginal**
  - The upper bound for pointwise confidence bands.

Author(s)

David Bolin <davidbolin@gmail.com> and Finn Lindgren <finn.lindgren@gmail.com>

References


See Also

simconf.inla, simconf.mc, simconf.mixture

Examples

```r
## Create mean and a tridiagonal precision matrix
n = 11
mu.x = seq(-5, 5, length=n)
Q.x = Matrix(toeplitz(c(1, -0.1, rep(0, n-2))))
## calculate the confidence region
conf = simconf(0.05, mu.x, Q.x, max.threads=2)
## Plot the region
plot(mu.x, type="l", ylim=c(-10, 10),
    main="Mean (black) and confidence region (red)'
    lines(conf$a, col=2)
    lines(conf$b, col=2)
```

---

**simconf.inla**

*Simultaneous confidence regions for latent Gaussian models*

Description

**simconf.inla** is used for calculating simultaneous confidence regions for latent Gaussian models estimated using INLA.
Usage

simconf.inla(
    result.inla,
    stack,
    name = NULL,
    tag = NULL,
    ind = NULL,
    alpha,
    method = "NI",
    n.iter = 10000,
    verbose = 0,
    link = FALSE,
    max.threads = 0,
    seed = NULL,
    inla.sample = TRUE
)

Arguments

result.inla  Result object from INLA call.
stack         The stack object used in the INLA call.
name          The name of the component for which to do the calculation. This argument should only be used if a stack object is not provided, use the tag argument otherwise.
tag           The tag of the component in the stack for which to do the calculation. This argument should only be used if a stack object is provided, use the name argument otherwise.
ind           If only a part of a component should be used in the calculations, this argument specifies the indices for that part.
alpha         Error probability for the region.
method        Method for handling the latent Gaussian structure:
               • 'EB' Empirical Bayes (Gaussian approximation of posterior).
               • 'NI' Numerical integration (Calculation based on the Gaussian mixture approximation of the posterior, as calculated by INLA).
n.iter         Number or iterations in the MC sampler that is used for approximating probabilities. The default value is 10000.
verbose       Set to TRUE for verbose mode (optional).
link          Transform output to the scale of the data using the link function as defined in the model estimated with INLA (default FALSE).
max.threads   Decides the number of threads the program can use. Set to 0 for using the maximum number of threads allowed by the system (default).
seed          Random seed (optional).
inla.sample   Set to TRUE if inla.posterior.sample should be used for the MC integration.
Details

See simconf for details.

Value

An object of class "excurobj" with elements

- **a**: The lower bound.
- **b**: The upper bound.
- **a.marginal**: The lower bound for pointwise confidence bands.
- **b.marginal**: The upper bound for pointwise confidence bands.

Note

This function requires the INLA package, which is not a CRAN package. See [http://www.r-inla.org/download](http://www.r-inla.org/download) for easy installation instructions.

Author(s)

David Bolin <davidbolin@gmail.com>

References


See Also

simconf, simconf.mc, simconf.mixture

Examples

```r
if (require.nowarnings("INLA")) {
  n <- 10
  x <- seq(0, 6, length.out=n)
  y <- sin(x) + rnorm(n)
  mu <- 1:n
  result <- inla(y ~ 1 + f(mu, model='rw2'),
                 data=list(y=y, mu=mu), verbose=FALSE,
                 control.compute = list(config=TRUE),
                 num.threads = 1)

  res <- simconf.inla(result, name='mu', alpha = 0.05, max.threads = 1)

  plot(result$summary.random$mu$mean, ylim=c(-2,2))
  lines(res$a)
  lines(res$b)
}
```
Simultaneous confidence regions using Monte Carlo samples

Description

`simconf.mc` is used for calculating simultaneous confidence regions based on Monte Carlo samples. The function returns upper and lower bounds \( a \) and \( b \) such that \( P(a < x < b) = 1 - \alpha \).

Usage

```r
simconf.mc(samples, alpha, ind, verbose = FALSE)
```

Arguments

- **samples**: Matrix with model Monte Carlo samples. Each column contains a sample of the model.
- **alpha**: Error probability for the region.
- **ind**: Indices of the nodes that should be analyzed (optional).
- **verbose**: Set to TRUE for verbose mode (optional).

Details

See `simconf` for details.

Value

An object of class "excurobj" with elements

- **a**: The lower bound.
- **b**: The upper bound.
- **a.marginal**: The lower bound for pointwise confidence bands.
- **b.marginal**: The upper bound for pointwise confidence bands.

Author(s)

David Bolin <davidbolin@gmail.com>

See Also

- `simconf`
- `simconf.inla`
Examples

```r
## Create mean and a tridiagonal precision matrix
n = 11
mu.x = seq(-5, 5, length=n)
Q.x = Matrix(toeplitz(c(1, -0.1, rep(0, n-2))))
## Sample the model 100 times (increase for better estimate)
X = mu.x + solve(chol(Q.x), matrix(rnorm(n=n*100), nrow=n, ncol=100))
## calculate the confidence region
conf = simconf.mc(X, 0.2)
## Plot the region
plot(mu.x, type="l", ylim=c(-10, 10),
     main="Var Mean (black) and confidence region (red)")
lines(conf$a, col=2)
lines(conf$b, col=2)
```

---

simconf.mixture

Simultaneous confidence regions for Gaussian mixture models

Description

simconf.mixture is used for calculating simultaneous confidence regions for Gaussian mixture models. The distribution for the process \( x \) is assumed to be

\[
\text{latex}
\]

The function returns upper and lower bounds \( a \) and \( b \) such that \( P(a < x < b) = 1 - \alpha \).

Usage

```r
simconf.mixture(
  alpha,
  mu,
  Q,
  w,
  ind,
  n.iter = 10000,
  vars,
  verbose = 0,
  max.threads = 0,
  seed = NULL,
  mix.samp = TRUE
)
```

Arguments

- `alpha`: Error probability for the region.
- `mu`: A list with the \( k \) expectation vectors \text{latex}.
- `Q`: A list with the \( k \) precision matrices \text{latex}.
simconf.mixture

w 
A vector with the weights for each class in the mixture.

ind 
Indices of the nodes that should be analyzed (optional).

n.iter 
Number or iterations in the MC sampler that is used for approximating probabilities. The default value is 10000.

vars 
A list with precomputed marginal variances for each class (optional).

verbose 
Set to TRUE for verbose mode (optional).

max.threads 
Decides the number of threads the program can use. Set to 0 for using the maximum number of threads allowed by the system (default).

seed 
Random seed (optional).

mix.samp 
If TRUE, the MC integration is done by directly sampling the mixture, otherwise sequential integration is used.

Details
See simconf for details.

Value

# @return An object of class "exurobj" with elements

a 
The lower bound.

b 
The upper bound.

a.marginal 
The lower bound for pointwise confidence bands.

b.marginal 
The upper bound for pointwise confidence bands.

Author(s)
David Bolin <davidbolin@gmail.com>

References


See Also

simconf, simconf.inla, simconf.mc

Examples

n = 11
K = 3
mu <- Q <- list()
for(k in 1:K){
  mu[[k]] = k*0.1 + seq(-5, 5, length=n)
Q[[k]] = Matrix(toeplitz(c(1, -0.1, rep(0, n-2))))
}
## calculate the confidence region
conf = simconf.mixture(0.05, mu, Q, w = rep(1/3,3), max.threads=2)

## Plot the region
plot(mu[[1]],type="l")
lines(mu[[2]])
lines(mu[[3]])
lines(conf$a, col=2)
lines(conf$b, col=2)

---

**submesh.grid**

*Extract a part of a grid*

**Description**

Extracts a part of a grid.

**Usage**

```r
submesh.grid(z, grid = NULL)
```

**Arguments**

- `z`: A matrix with values indicating which nodes that should be present in the sub-mesh.
- `grid`: A list with locations and dimensions of the grid.

**Value**

An `inla.mesh` object.

**Note**

This function requires the INLA package, which is not a CRAN package. See [http://www.r-inla.org/download](http://www.r-inla.org/download) for easy installation instructions.

**Author(s)**

Finn Lindgren <finn.lindgren@gmail.com>
## Not run:
```r
if (require(INLA)) {
  nxy = 40
  x = seq(from=0, to=4, length.out=nxy)
  lattice = inla.mesh.lattice(x=x, y=x)
  mesh = inla.mesh.create(lattice=lattice, extend=FALSE, refine=FALSE)

  # extract a part of the mesh inside a circle
  xy.in <- rowSums((mesh$loc[,1:2]-2)^2) < 1
  submesh <- submesh.grid(matrix(xy.in, nxy, nxy),
                          list(loc=mesh$loc, dim=c(nxy, nxy)))
  plot(mesh$loc[,1:2])
  lines(2 + cos(seq(0, 2*pi, length.out=100)), 2 + sin(seq(0, 2*pi, length.out=100)))
  plot(submesh, add=TRUE)
  points(mesh$loc[xy.in, 1:2], col="2")
}
## End(Not run)
```

---

### submesh.mesh

#### Extract a part of a mesh

## Description

Extracts a part of a mesh

## Usage

```r
submesh.mesh(z, mesh)
```

## Arguments

- `z`: A matrix with values indicating which nodes that should be present in the sub-
  mesh.
- `mesh`: An `inla.mesh` object.

## Value

An `inla.mesh` object.

## Note

This function requires the INLA package, which is not a CRAN package. See [http://www.r-inla.org/download](http://www.r-inla.org/download) for easy installation instructions.

### Author(s)

Finn Lindgren <finn.lindgren@gmail.com>
### Examples

```r
## Not run:
if (require(INLA)) {
  nxy = 30
x = seq(from=0, to=4, length.out=nxy)
lattice = inla.mesh.lattice(x=x, y=x)
mesh = inla.mesh.create(lattice=lattice, extend=FALSE, refine=FALSE)

# extract a part of the mesh inside a circle
xy.in <- rowSums((mesh$loc[, 1:2] - 2)^2) < 1
submesh <- excursions:::submesh.mesh(matrix(xy.in, nxy, nxy), mesh)
plot(mesh$loc[, 1:2])
lines(2 + cos(seq(0, 2*pi, length.out=100)), 2 + sin(seq(0, 2*pi, length.out=100)))
plot(submesh, add=TRUE)
points(mesh$loc[xy.in, 1:2], col="2")
}
## End(Not run)
```

### summary.excurobj

**Summarise excurobj objects**

**Description**

Summary method for class "excurobj"

**Usage**

```r
## S3 method for class 'excurobj'
summary(object, ...)

## S3 method for class 'summary.excurobj'
print(x, ...)
```

**Arguments**

- `object` an object of class "excurobj", usually, a result of a call to `excursions`.
- `x` an object of class "summary.excurobj", usually, a result of a call to `summary.excurobj`.
- `...` further arguments passed to or from other methods.
tricontour

Calculate contour curves on a triangulation

Description

Calculates contour curves and/or regions between them, for functions defined on a triangulation

Usage

tricontour(
  x,
  z,
  nlevels = 10,
  levels = pretty(range(z, na.rm = TRUE), nlevels),
  ...
)

## S3 method for class 'inla.mesh'
tricontour(
  x,
  z,
  nlevels = 10,
  levels = pretty(range(z, na.rm = TRUE), nlevels),
  ...
)

## S3 method for class 'matrix'
tricontour(
  x,
  z,
  nlevels = 10,
  levels = pretty(range(z, na.rm = TRUE), nlevels),
  loc,
  ...
)

## S3 method for class 'list'
tricontour(
  x,
  z,
  nlevels = 10,
  levels = pretty(range(z, na.rm = TRUE), nlevels),
  loc,
  type = c("+", "-"),
  tol = 1e-07,
  ...
)
tricontourmap(
  x,
  z,
  nlevels = 10,
  levels = pretty(range(z, na.rm = TRUE), nlevels),
  ...
)

## S3 method for class 'inla.mesh'
tricontourmap(
  x,
  z,
  nlevels = 10,
  levels = pretty(range(z, na.rm = TRUE), nlevels),
  ...
)

## S3 method for class 'matrix'
tricontourmap(
  x,
  z,
  nlevels = 10,
  levels = pretty(range(z, na.rm = TRUE), nlevels),
  loc,
  ...
)

## S3 method for class 'list'
tricontourmap(
  x,
  z,
  nlevels = 10,
  levels = pretty(range(z, na.rm = TRUE), nlevels),
  loc,
  type = c("+", "-"),
  tol = 1e-07,
  output = c("sp", "inla.mesh.segment"),
  ...
)

Arguments

x             An object generated by a call to \texttt{inla.mesh.2d} or \texttt{inla.mesh.create}, a triangle-
vertex index matrix, or a list of triangulation information, \texttt{list(loc, graph=list(tv))}.

z             A vector containing the values to be contoured (NAs are allowed).

nlevels       Number of contour levels desired, if and only if \texttt{levels} is not supplied.

levels        Numeric vector of levels at which to calculate contour lines.
... Additional arguments passed to the other methods.

**loc**
coordinate matrix, to be supplied when `x` is given as a triangle-vertex index matrix only.

**type**
"+" or "-", indicating positive or negative association. For +, the generated contours enclose regions where \( u_1 \leq z < u_2 \), for - the regions fulfil \( u_1 < z \leq u_2 \).

**tol**
tolerance for determining if the value at a vertex lies on a level.

**output**
The format of the generated output. Implemented options are "sp" (default) and "inla.mesh.segment" (requires the INLA package).

**Value**
For `tricontour`, a list some of the same fields that `inla.mesh.segment` objects have:

- **loc**
  A coordinate matrix
- **idx**
  Contour segment indices, as a 2-column matrix, each row indexing a single segment
- **grp**
  A vector of group labels. Each segment has a label, in \( 1,...,n\text{levels}\times2+1 \), where even labels indicate interior on-level contour segments, and odd labels indicate boundary segments between levels.

For `tricontourmap`, a list:

- **contour**
  A list of sp or `inla.mesh.segment` objects defining countour curves (level sets)
- **map**
  A list of sp or `inla.mesh.segment` objects enclosing regions between level sets

**Author(s)**
Finn Lindgren <finn.lindgren@gmail.com>

**Examples**

```r
if (require.nowarnings("INLA")) {
  ## Generate mesh and SPDE model
  n.lattice <- 20 #increase for more interesting, but slower, examples
  x <- seq(from = 0, to = 10, length.out = n.lattice)
  lattice <- inla.mesh.lattice(x = x, y = x)
  mesh <- inla.mesh.create(lattice = lattice, extend = FALSE, refine = FALSE)
  spde <- inla.spde2.matern(mesh, alpha = 2)

  ## Generate an artificial sample
  sigma2.e <- 0.01
  n.obs <- 1000
  obs.loc <- cbind(runif(n.obs) * diff(range(x)) + min(x),
                   runif(n.obs) * diff(range(x)) + min(x))
  Q <- inla.spde2.precision(spde, theta = c(log(sqrt(0.5)), log(sqrt(1))))
  x <- inla.qsample(Q = Q)
  A <- inla.spde.make.A(mesh = mesh, loc = obs.loc)
}```
Y <- as.vector(A %*% x + rnorm(n.obs) * sqrt(sigma2.e))

## Calculate posterior
Q.post <- (Q + (t(A) %*% A)/sigma2.e)
mu.post <- as.vector(solve(Q.post,(t(A) %*% Y)/sigma2.e))

## Calculate continuous contours
tric <- tricontour(mesh, z = mu.post,
                   levels = as.vector(quantile(x, c(0.25, 0.75))))

## Discrete domain contours
map <- contourmap(n.levels = 2, mu = mu.post, Q = Q.post,
                   alpha=0.1, compute = list(F = FALSE), max.threads=1)

## Calculate continuous contour map
setsc <- tricontourmap(mesh, z = mu.post,
                        levels = as.vector(quantile(x, c(0.25, 0.75))))

## Plot the results
reo <- mesh$idx$lattice
idx.setsc <- setdiff(names(setsc$map), "-1")
cols2 <- contourmap.colors(map, col=heat.colors(100, 0.5),
                           credible.col = grey(0.5, 0))
names(cols2) <- as.character(-1:2)
par(mfrow = c(1,2))
image(matrix(mu.post[reo], n.lattice, n.lattice),
       main = "mean", axes = FALSE)
plot(setsc$map[idx.setsc], col = cols2[idx.setsc])
par(mfrow = c(1,1))
}
Index

continuous, 4, 6
contourmap, 5, 6, 8, 14, 16, 17
contourmap.colors, 5, 10, 11, 14, 17
contourmap.inla, 5, 10, 12, 16, 17
contourmap.mc, 5, 10, 14, 15

excursions, 4, 6, 19, 22, 36
excursions (excursions-package), 2
excursions-package, 2
excursions.inla, 4, 5, 17, 22
excursions.mc, 4, 5, 19, 20
excursions.variances, 22

gaussint, 23

print.excurobj(summary.excurobj), 36
print.summary.excurobj
 (summary.excurobj), 36

require, 26
require.nowarnings, 26

simconf, 5, 27, 30, 31, 33
simconf.inla, 5, 28, 28, 31, 33
simconf.mc, 5, 28, 30, 31, 33
simconf.mixture, 5, 28, 30, 32
submesh.grid, 34
submesh.mesh, 35
summary.excurobj, 36, 36

tricontour, 5, 37
tricontourmap, 5
tricontourmap(tricontour), 37