Package 'BayesNSGP'

October 12, 2022

Title Bayesian Analysis of Non-Stationary Gaussian Process Models

Description Enables off-the-shelf functionality for fully Bayesian, nonstationary Gaussian process modeling. The approach to nonstationary modeling involves a closed-form, convolution-based covariance function with spatially-varying parameters; these parameter processes can be specified either deterministically (using covariates or basis functions) or stochastically (using approximate Gaussian processes). Stationary Gaussian processes are a special case of our methodology, and we furthermore implement approximate Gaussian process inference to account for very large spatial data sets (Finley, et al (2017) <arXiv:1702.00434v2>). Bayesian inference is carried out using Markov chain Monte Carlo methods via the 'nimble' package, and posterior prediction for the Gaussian process at unobserved locations is provided as a post-processing step.

Version 0.1.2

Date 2022-01-07

Maintainer Daniel Turek <danielturek@gmail.com>

Author Daniel Turek, Mark Risser

Depends R (>= 3.4.0),nimble

Imports FNN,Matrix,methods,StatMatch

License GPL-3

Encoding UTF-8

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2022-01-09 01:52:42 UTC

R topics documented:

`calcQF` *Calculate the Gaussian quadratic form for the NNGP approximation*

Description

`calcQF` calculates the quadratic form in the multivariate Gaussian based on the NNGP approximation, for a specific parameter combination. The quadratic form is `t(u)C^{-1}v`.

Usage

```
calcQF(u, v, AD, nID)
```

Arguments

`u`	Vector; left product.
`v`	Vector; right product
`AD`	N x (k+1) matrix; the first k columns are the 'A' matrix, and the last column is the 'D' vector. Represents the Cholesky of `C^{-1}`.
`nID`	N x k matrix of neighbor indices.

Value

A list with two components: (1) an N x 2 array containing the same spatial coordinates, ordered by MMD, and (2) the same thing, but with any NA values removed.

`calculateAD_ns` *Calculate A and D matrices for the NNGP approximation*

Description

`calculateAD_ns` calculates A and D matrices (the Cholesky of the precision matrix) needed for the NNGP approximation.

Usage

```
calculateAD_ns(
  dist1_3d,
  dist2_3d,
  dist12_3d,
  Sigma11,
  Sigma22,
  Sigma12,
  log_sigma_vec,
  log_tau_vec,
  nID,
  N,
  k,
  nu,
  d
)
```

Arguments

`dist1_3d`	N x (k+1) x (k+1) array of distances in the x-coordinate direction.
`dist2_3d`	N x (k+1) x (k+1) array of distances in the y-coordinate direction.
`dist12_3d`	N x (k+1) x (k+1) array of cross-distances.
`Sigma11`	N-vector; 1-1 element of the Sigma() process.
`Sigma22`	N-vector; 2-2 element of the Sigma() process.
`Sigma12`	N-vector; 1-2 element of the Sigma() process.
`log_sigma_vec`	N-vector; process standard deviation values.
`log_tau_vec`	N-vector; nugget standard deviation values.
`nID`	N x k matrix of neighbor indices.
`N`	Scalar; number of data measurements.
`k`	Scalar; number of nearest neighbors.
`nu`	Scalar; Matern smoothness parameter.
`d`	Scalar; dimension of the spatial domain.

Value

A N x (k+1) matrix; the first k columns are the 'A' matrix, and the last column is the 'D' vector.

`calculateU_ns` *Calculate the (sparse) matrix U*

Description

`calculateU_ns` calculates the (sparse) matrix U (i.e., the Cholesky of the inverse covariance matrix) using a nonstationary covariance function. The output only contains non-zero values and is stored as three vectors: (1) the row indices, (2) the column indices, and (3) the non-zero values. NOTE: this code assumes the all inputs correspond to the ORDERED locations.

Usage

```
calculateU_ns(
  dist1_3d,
  dist2_3d,
  dist12_3d,
  Sigma11,
  Sigma22,
  Sigma12,
  log_sigma_vec,
  log_tau_vec,
  nu,
  nID,
  cond_on_y,
  N,
  k,
  d,
  M = 0
)
```

Arguments

`dist1_3d`	N x (k+1) x (k+1) array of distances in the x-coordinate direction.
`dist2_3d`	N x (k+1) x (k+1) array of distances in the y-coordinate direction.
`dist12_3d`	N x (k+1) x (k+1) array of cross-distances.
`Sigma11`	N-vector; 1-1 element of the Sigma() process.
`Sigma22`	N-vector; 2-2 element of the Sigma() process.
`Sigma12`	N-vector; 1-2 element of the Sigma() process.
`log_sigma_vec`	N-vector; process standard deviation values.
`log_tau_vec`	N-vector; nugget standard deviation values.
`nu`	Scalar; Matern smoothness parameter.

nID	N x k matrix of (ordered) neighbor indices.
cond_on_y	A matrix indicating whether the conditioning set for each (ordered) location is on the latent process (y, 1) or the observed values (z, 0). Calculated in sgvSetup.
N	Scalar; number of data measurements.
k	Scalar; number of nearest neighbors.
d	Scalar; dimension of the spatial domain.
M	Scalar; number of prediction sites.

Value

Returns a sparse matrix representation of the Cholesky of the precision matrix for a fixed set of covariance parameters.

---

conditionLatentObs *Assign conditioning sets for the SGV approximation*

Description

conditionLatentObs assigns q_y(i) vs q_z(i) following Section 5.1 in Katzfuss and Guinness (2018). This function only needs to be run once per SGV analysis.

Usage

```
conditionLatentObs(nID, coords_ord, N)
```

Arguments

nID	N x k matrix of neighbor indices.
coords_ord	N x 2 matrix of locations.
N	Scalar; number of locations (observed only!).

Value

A matrix indicating whether the conditioning set for each location is on the latent process (y, 1) or the observed values (z, 0).

determineNeighbors *Determine the k-nearest neighbors for each spatial coordinate.*

Description

determineNeighbors returns an N x k matrix of the nearest neighbors for spatial locations coords, with the ith row giving indices of the k nearest neighbors to the ith location, which are selected from among the 1,...(i-1) other spatial locations. The first row is -1's, since the first location has no neighbors. The i=2 through i=(k+1) rows each necessarily contain 1:i.

Usage

```
determineNeighbors(coords, k)
```

Arguments

coords	N x 2 array of N 2-dimensional (x,y) spatial coordinates.
k	Scalar; number of neighbors

Value

An N x k matrix of nearest neighbor indices

Examples

```
coords <- cbind(runif(100), runif(100))
determineNeighbors(coords, 20)
```

dmnorm_nngp *Function for the evaluating the NNGP approximate density.*

Description

dmnorm_nngp (and rmnorm_nngp) calculate the approximate NNGP likelihood for a fixed set of parameters (i.e., A and D matrices). Finally, the distributions must be registered within nimble.

Usage

```
dmnorm_nngp(x, mean, AD, nID, N, k, log)
```

Arguments

x	N-vector of data.
mean	N-vector with current values of the mean
AD	N x (k+1) matrix; the first k columns are the 'A' matrix, and the last column is the 'D' vector.
nID	N x k matrix of neighbor indices.
N	Scalar; number of data measurements.
k	Scalar; number of nearest neighbors.
log	Scalar; should the density be on the log scale (1) or not (0).

Value

The NNGP approximate density.

---

dmnorm_sgv *Function for the evaluating the SGV approximate density.*

Description

dmnorm_sgv (and rmnorm_sgv) calculate the approximate SGV likelihood for a fixed set of parameters (i.e., the U matrix). Finally, the distributions must be registered within nimble.

Usage

```
dmnorm_sgv(x, mean, U, N, k, log = 1)
```

Arguments

x	Vector of measurements
mean	Vector of mean valiues
U	Matrix of size N x 3; representation of a sparse N x N Cholesky of the precision matrix. The first two columns contain row and column indices, respectively, and the last column is the nonzero elements of the matrix.
N	Number of measurements in x
k	Number of neighbors for the SGV approximation.
log	Logical; should the density be evaluated on the log scale.

Value

Returns the SGV approximation to the Gaussian likelihood.

`inverseEigen` *Calculate covariance elements based on eigendecomposition components*

Description

`inverseEigen` calculates the inverse eigendecomposition – in other words, the covariance elements based on the eigenvalues and vectors. For a 2x2 anisotropy (covariance) matrix, we parameterize the three unique values in terms of the two log eigenvalues and a rotation parameter on the rescaled logit. The function is coded as a `nimbleFunction` (see the `nimble` package) but can also be used as a regular R function.

Usage

```
inverseEigen(eigen_comp1, eigen_comp2, eigen_comp3, which_Sigma)
```

Arguments

`eigen_comp1`	N-vector; contains values of the log of the first anisotropy eigenvalue for a set of locations.
`eigen_comp2`	N-vector; contains values of the log of the second anisotropy eigenvalue for a set of locations.
`eigen_comp3`	N-vector; contains values of the rescaled logit of the anisotropy rotation for a set of locations.
`which_Sigma`	Scalar; one of `(1,2,3)`, corresponding to which covariance component should be calculated (Sigma11, Sigma22, or Sigma12, respectively).

Value

A vector of anisotropy values (Sigma11, Sigma22, or Sigma12; depends on `which_Sigma`) for the corresponding set of locations.

Examples

```
# Generate some eigendecomposition elements (all three are real-valued)
eigen_comp1 <- rnorm(10)
eigen_comp2 <- rnorm(10)
eigen_comp3 <- rnorm(10)
inverseEigen( eigen_comp1, eigen_comp2, eigen_comp3, 2) # Return the Sigma22 values
```

matern_corr *Calculate a stationary Matern correlation matrix*

Description

matern_corr calculates a stationary Matern correlation matrix for a fixed set of locations, based on a range and smoothness parameter. This function is primarily used for the "npGP" and "approxGP" models. The function is coded as a nimbleFunction (see the nimble package) but can also be used as a regular R function.

Usage

```
matern_corr(dist, rho, nu)
```

Arguments

dist	N x N matrix; contains values of pairwise Euclidean distances in the x-y plane.
rho	Scalar; "range" parameter used to rescale distances
nu	Scalar; Matern smoothness parameter. nu = 0.5 corresponds to the Exponential correlation; nu = Inf corresponds to the Gaussian correlation function.

Value

A correlation matrix for a fixed set of stations and fixed parameter values.

Examples

```
# Generate some coordinates
coords <- cbind(runif(100),runif(100))
nu <- 2
# Calculate distances -- can use nsDist to calculate Euclidean distances
dist_list <- nsDist(coords, isotropic = TRUE)
# Calculate the correlation matrix
corMat <- matern_corr(sqrt(dist_list$dist1_sq), 1, nu)
```

nimble_sparse_chol *nimble_sparse_chol*

Description

nimble_sparse_chol

Usage

```
nimble_sparse_chol(i, j, x, n)
```

Arguments

i	Vector of row indices.
j	Vector of column indices.
x	Vector of values in the matrix.
n	Length of the vector

`nimble_sparse_crossprod` *nimble_sparse_crossprod*

Description

nimble_sparse_crossprod

Usage

```
nimble_sparse_crossprod(i, j, x, z, n, subset, transp)
```

Arguments

i	Vector of row indices.
j	Vector of column indices.
x	Vector of values in the matrix.
z	Vector to calculate the cross-product with.
n	Length of the vector
subset	Optional vector of rows to include in the calculation.
transp	Optional indicator of using the transpose

`nimble_sparse_solve` *nimble_sparse_solve*

Description

nimble_sparse_solve

Usage

```
nimble_sparse_solve(i, j, x, z)
```

Arguments

i	Vector of row indices.
j	Vector of column indices.
x	Vector of values in the matrix.
z	Vector to calculate the cross-product with.

nimble_sparse_tcrossprod *nimble_sparse_tcrossprod*

Description

nimble_sparse_tcrossprod

Usage

```
nimble_sparse_tcrossprod(i, j, x, subset)
```

Arguments

i	Vector of row indices.
j	Vector of column indices.
x	Vector of values in the matrix.
subset	Optional vector of rows to include in the calculation.

nsCorr *Calculate a nonstationary Matern correlation matrix*

Description

nsCorr calculates a nonstationary correlation matrix for a fixed set of locations, based on vectors of the unique anisotropy parameters for each station. Since the correlation function uses a spatially-varying Mahalanobis distance, this function requires coordinate- specific distance matrices (see below). The function is coded as a nimbleFunction (see the nimble package) but can also be used as a regular R function.

Usage

```
nsCorr(dist1_sq, dist2_sq, dist12, Sigma11, Sigma22, Sigma12, nu, d)
```

Arguments

dist1_sq	N x N matrix; contains values of pairwise squared distances in the x-coordinate.
dist2_sq	N x N matrix; contains values of pairwise squared distances in the y-coordinate.
dist12	N x N matrix; contains values of pairwise signed cross- distances between the x- and y-coordinates. The sign of each element is important; see nsDist function for the details of this calculation. in the x-coordinate.
Sigma11	Vector of length N; contains the 1-1 element of the anisotropy process for each station.

Sigma22	Vector of length N; contains the 2-2 element of the anisotropy process for each station.
Sigma12	Vector of length N; contains the 1-2 element of the anisotropy process for each station.
nu	Scalar; Matern smoothness parameter. nu = 0.5 corresponds to the Exponential correlation; nu = Inf corresponds to the Gaussian correlation function.
d	Scalar; dimension of the spatial coordinates.

Value

A correlation matrix for a fixed set of stations and fixed parameter values.

Examples

```
# Generate some coordinates and parameters
coords <- cbind(runif(100),runif(100))
Sigma11 <- rep(1, 100) # Identity anisotropy process
Sigma22 <- rep(1, 100)
Sigma12 <- rep(0, 100)
nu <- 2
# Calculate distances
dist_list <- nsDist(coords)
# Calculate the correlation matrix
corMat <- nsCorr(dist_list$dist1_sq, dist_list$dist2_sq, dist_list$dist12,
                 Sigma11, Sigma22, Sigma12, nu, ncol(coords))
```

---

nsCrosscorr *Calculate a nonstationary Matern cross-correlation matrix*

Description

nsCrosscorr calculates a nonstationary cross-correlation matrix between two fixed sets of locations (a prediction set with M locations, and the observed set with N locations), based on vectors of the unique anisotropy parameters for each station. Since the correlation function uses a spatially-varying Mahalanobis distance, this function requires coordinate- specific distance matrices (see below). The function is coded as a nimbleFunction (see the nimble package) but can also be used as a regular R function.

Usage

```
nsCrosscorr(
  Xdist1_sq,
  Xdist2_sq,
  Xdist12,
  Sigma11,
  Sigma22,
```

```
  Sigma12,
  PSigma11,
  PSigma22,
  PSigma12,
  nu,
  d
)
```

Arguments

Xdist1_sq	M x N matrix; contains values of pairwise squared cross-distances in the x-coordinate.
Xdist2_sq	M x N matrix; contains values of pairwise squared cross-distances in the y-coordinate.
Xdist12	M x N matrix; contains values of pairwise signed cross/cross- distances between the x- and y-coordinates. The sign of each element is important; see nsDist function for the details of this calculation. in the x-coordinate.
Sigma11	Vector of length N; contains the 1-1 element of the anisotropy process for each observed location.
Sigma22	Vector of length N; contains the 2-2 element of the anisotropy process for each observed location.
Sigma12	Vector of length N; contains the 1-2 element of the anisotropy process for each observed location.
PSigma11	Vector of length N; contains the 1-1 element of the anisotropy process for each prediction location.
PSigma22	Vector of length N; contains the 2-2 element of the anisotropy process for each prediction location.
PSigma12	Vector of length N; contains the 1-2 element of the anisotropy process for each prediction location.
nu	Scalar; Matern smoothness parameter. nu = 0.5 corresponds to the Exponential correlation; nu = Inf corresponds to the Gaussian correlation function.
d	Scalar; dimension of the spatial domain.

Value

A M x N cross-correlation matrix for two fixed sets of stations and fixed parameter values.

Examples

```
# Generate some coordinates and parameters
coords <- cbind(runif(100),runif(100))
Sigma11 <- rep(1, 100) # Identity anisotropy process
Sigma22 <- rep(1, 100)
Sigma12 <- rep(0, 100)
Pcoords <- cbind(runif(200),runif(200))
PSigma11 <- rep(1, 200) # Identity anisotropy process
PSigma22 <- rep(1, 200)
```

```
PSigma12 <- rep(0, 200)
nu <- 2
# Calculate distances
Xdist_list <- nsCrossdist(coords, Pcoords)
# Calculate the correlation matrix
XcorMat <- nsCrosscorr(Xdist_list$dist1_sq, Xdist_list$dist2_sq, Xdist_list$dist12,
   Sigma11, Sigma22, Sigma12, PSigma11, PSigma22, PSigma12, nu, ncol(coords))
```

`nsCrossdist` *Calculate coordinate-specific cross-distance matrices*

Description

`nsCrossdist` calculates coordinate-specific cross distances in x, y, and x-y for use in the nonstationary cross-correlation calculation. This function is useful for calculating posterior predictions.

Usage

```
nsCrossdist(coords, Pcoords, scale_factor = NULL, isotropic = FALSE)
```

Arguments

`coords`	N x 2 matrix; contains x-y coordinates of station (observed) locations.
`Pcoords`	M x 2 matrix; contains x-y coordinates of prediction locations.
`scale_factor`	Scalar; optional argument for re-scaling the distances.
`isotropic`	Logical; indicates whether distances should be calculated using Euclidean distance (`isotropic = TRUE`) or using the anisotropic formulation (`isotropic = FALSE`).

Value

A list of distances matrices, with the following components:

`dist1_sq`	M x N matrix; contains values of pairwise squared cross- distances in the x-coordinate.
`dist2_sq`	M x N matrix; contains values of pairwise squared cross- distances in the y-coordinate.
`dist12`	M x N matrix; contains values of pairwise signed cross- distances between the x- and y-coordinates.
`scale_factor`	Value of the scale factor used to rescale distances.

Examples

```
# Generate some coordinates
coords <- cbind(runif(100),runif(100))
Pcoords <- cbind(runif(200),runif(200))
# Calculate distances
Xdist_list <- nsCrossdist(coords, Pcoords)
```

`nsCrossdist3d` *Calculate coordinate-specific cross-distance matrices, only for nearest neighbors and store in an array*

Description

`nsCrossdist3d` generates and returns new 3-dimensional arrays containing the former dist1_sq, dist2_s1, and dist12 matrices, but only as needed for the k nearest-neighbors of each location. these 3D matrices (dist1_3d, dist2_3d, and dist12_3d) are used in the new implementation of calculateAD_ns().

Usage

```
nsCrossdist3d(
  coords,
  predCoords,
  P_nID,
  scale_factor = NULL,
  isotropic = FALSE
)
```

Arguments

`coords`	N x d matrix; contains the x-y coordinates of stations.
`predCoords`	M x d matrix
`P_nID`	N x k matrix; contains indices of nearest neighbors.
`scale_factor`	Scalar; optional argument for re-scaling the distances.
`isotropic`	Logical; indicates whether distances should be calculated separately for each coordinate dimension (FALSE) or simultaneously for all coordinate dimensions (TRUE). `isotropic = FALSE` can only be used for two-dimensional coordinate systems.

Value

Arrays with nearest neighbor distances in each coordinate direction. When the spatial dimension d > 2, dist1_3d contains squared Euclidean distances, and dist2_3d and dist12_3d are empty.

Examples

```
# Generate some coordinates and neighbors
coords <- cbind(runif(100),runif(100))
predCoords <- cbind(runif(200),runif(200))
P_nID <- FNN::get.knnx(coords, predCoords, k = 10)$nn.index # Prediction NN
# Calculate distances
Pdist <- nsCrossdist3d(coords, predCoords, P_nID)
```

---

nsDist *Calculate coordinate-specific distance matrices*

Description

`nsDist` calculates x, y, and x-y distances for use in the nonstationary correlation calculation. The sign of the cross-distance is important. The function contains an optional argument for re-scaling the distances such that the coordinates lie in a square.

Usage

```
nsDist(coords, scale_factor = NULL, isotropic = FALSE)
```

Arguments

`coords`	N x 2 matrix; contains the x-y coordinates of stations
`scale_factor`	Scalar; optional argument for re-scaling the distances.
`isotropic`	Logical; indicates whether distances should be calculated separately for each coordinate dimension (FALSE) or simultaneously for all coordinate dimensions (TRUE). `isotropic = TRUE` can only be used for two-dimensional coordinate systems.

Value

A list of distances matrices, with the following components:

`dist1_sq`	N x N matrix; contains values of pairwise squared distances in the x-coordinate.
`dist2_sq`	N x N matrix; contains values of pairwise squared distances in the y-coordinate.
`dist12`	N x N matrix; contains values of pairwise signed cross- distances between the x- and y-coordinates.
`scale_factor`	Value of the scale factor used to rescale distances.

Examples

```
# Generate some coordinates
coords <- cbind(runif(100),runif(100))
# Calculate distances
dist_list <- nsDist(coords)
# Use nsDist to calculate Euclidean distances
dist_Euclidean <- sqrt(nsDist(coords, isotropic = TRUE)$dist1_sq)
```

---

nsDist3d *Calculate coordinate-specific distance matrices, only for nearest neighbors and store in an array*

Description

nsDist3d generates and returns new 3-dimensional arrays containing the former dist1_sq, dist2_sq, and dist12 matrices, but only as needed for the k nearest-neighbors of each location. these 3D matrices (dist1_3d, dist2_3d, and dist12_3d) are used in the new implementation of calculateAD_ns().

Usage

```
nsDist3d(coords, nID, scale_factor = NULL, isotropic = FALSE)
```

Arguments

coords	N x 2 matrix; contains the x-y coordinates of stations.
nID	N x k matrix; contains indices of nearest neighbors.
scale_factor	Scalar; optional argument for re-scaling the distances.
isotropic	Logical; indicates whether distances should be calculated separately for each coordinate dimension (FALSE) or simultaneously for all coordinate dimensions (TRUE). isotropic = TRUE can only be used for two-dimensional coordinate systems.

Value

Arrays with nearest neighbor distances in each coordinate direction.

Examples

```
# Generate some coordinates and neighbors
coords <- cbind(runif(100),runif(100))
nID <- determineNeighbors(coords, 10)
# Calculate distances
nsDist3d(coords, nID)
```

nsgpModel *NIMBLE code for a generic nonstationary GP model*

Description

This function sets up and compiles a nimble model for a general nonstationary Gaussian process.

Usage

```
nsgpModel(
  tau_model = "constant",
  sigma_model = "constant",
  Sigma_model = "constant",
  mu_model = "constant",
  likelihood = "fullGP",
  coords,
  data,
  constants = list(),
  monitorAllSampledNodes = TRUE,
  ...
)
```

Arguments

tau_model	Character; specifies the model to be used for the log(tau) process. Options are "constant" (spatially-constant), "logLinReg" (log-linear regression), and "approxGP" (approximation to a Gaussian process).
sigma_model	Character; specifies the model to be used for the log(sigma) process. See tau_model for options.
Sigma_model	Character; specifies the model to be used for the Sigma anisotropy process. Options are "constant" (spatially-constant), "constantIso" (spatially-constant and isotropic), "covReg" (covariance regression), "compReg" (componentwise regression), "compRegIso" (isotropic componentwise regression), "npApproxGP" (nonparameteric regression via an approximation to a stationary Gaussian process), and "npApproxGPIso" (isotropic nonparameteric regression via an approximation to a stationary Gaussian process)
mu_model	Character; specifies the model to be used for the mu mean process. Options are "constant" (spatially-constant), "linReg" (linear regression), and "zero" (a fixed zero-mean).
likelihood	Character; specifies the likelihood model. Options are "fullGP" (the exact Gaussian process likelihood), "NNGP" (the nearest-neighbor GP for the response approximate likelihood), and "SGV" (the sparse general Vecchia approximate likelihood).
coords	N x d matrix of spatial coordinates.
data	N-vector; observed vector of the spatial process of interest

`constants` A list of constants required to build the model; depends on the specific parameter process models chosen.

`monitorAllSampledNodes` Logical; indicates whether all sampled nodes should be stored (`TRUE`) or not (`FALSE`).

`...` Additional arguments can be passed to the function; for example, as an alternative to the `constants` list, items can be passed directly via this argument.

Value

A `nimbleCode` object.

Examples

```
# Generate some data: stationary/isotropic
N <- 100
coords <- matrix(runif(2*N), ncol = 2)
alpha_vec <- rep(log(sqrt(1)), N) # Log process SD
delta_vec <- rep(log(sqrt(0.05)), N) # Log nugget SD
Sigma11_vec <- rep(0.4, N) # Kernel matrix element 1,1
Sigma22_vec <- rep(0.4, N) # Kernel matrix element 2,2
Sigma12_vec <- rep(0, N) # Kernel matrix element 1,2
mu_vec <- rep(0, N) # Mean
nu <- 0.5 # Smoothness
dist_list <- nsDist(coords)
Cor_mat <- nsCorr( dist1_sq = dist_list$dist1_sq, dist2_sq = dist_list$dist2_sq,
                   dist12 = dist_list$dist12, Sigma11 = Sigma11_vec,
                   Sigma22 = Sigma22_vec, Sigma12 = Sigma12_vec, nu = nu )
Cov_mat <- diag(exp(alpha_vec)) %*% Cor_mat %*% diag(exp(alpha_vec))
D_mat <- diag(exp(delta_vec)^2)
set.seed(110)
data <- as.numeric(mu_vec + t(chol(Cov_mat + D_mat)) %*% rnorm(N))
# Set up constants
constants <- list( nu = 0.5, Sigma_HP1 = 2 )
# Defaults: tau_model = "constant", sigma_model = "constant", mu_model = "constant",
# and Sigma_model = "constant"
Rmodel <- nsgpModel(likelihood = "fullGP", constants = constants, coords = coords, data = data )
```

`nsgpPredict` *Posterior prediction for the NSGP*

Description

`nsgpPredict` conducts posterior prediction for MCMC samples generated using nimble and nsgpModel.

Usage

```
nsgpPredict(
  model,
  samples,
  coords.predict,
  predict.process = TRUE,
  constants,
  seed = 0,
  ...
)
```

Arguments

model	A NSGP nimble object; the output of nsgpModel.
samples	A matrix of J rows, each is an MCMC sample of the parameters corresponding to the specification in nsgpModel.
coords.predict	M x d matrix of prediction coordinates.
predict.process	Logical; determines whether the prediction corresponds to the y(·) process (TRUE) or z(·) (FALSE; this would likely only be used for, e.g., cross-validation).
constants	An optional list of contants to use for prediction; alternatively, additional arguments can be passed to the function via the ... argument.
seed	An optional random seed argument for reproducibility.
...	Additional arguments can be passed to the function; for example, as an alternative to the constants list, items can be passed directly via this argument.

Value

The output of the function is a list with two elements: obs, a matrix of J posterior predictive samples for the N observed locations (only for likelihood = "SGV", which produces predictions for the observed locations by default; this element is NULL otherwise); and pred, a corresponding matrix of posterior predictive samples for the prediction locations. Ordering and neighbor selection for the prediction coordinates in the SGV likelihood are conducted internally, as with nsgpModel.

Examples

```
# Generate some data: stationary/isotropic
N <- 100
coords <- matrix(runif(2*N), ncol = 2)
alpha_vec <- rep(log(sqrt(1)), N) # Log process SD
delta_vec <- rep(log(sqrt(0.05)), N) # Log nugget SD
Sigma11_vec <- rep(0.4, N) # Kernel matrix element 1,1
Sigma22_vec <- rep(0.4, N) # Kernel matrix element 2,2
Sigma12_vec <- rep(0, N) # Kernel matrix element 1,2
mu_vec <- rep(0, N) # Mean
nu <- 0.5 # Smoothness
dist_list <- nsDist(coords)
```

```
Cor_mat <- nsCorr( dist1_sq = dist_list$dist1_sq, dist2_sq = dist_list$dist2_sq,
                   dist12 = dist_list$dist12, Sigma11 = Sigma11_vec,
                   Sigma22 = Sigma22_vec, Sigma12 = Sigma12_vec, nu = nu )
Cov_mat <- diag(exp(alpha_vec)) %*% Cor_mat %*% diag(exp(alpha_vec))
D_mat <- diag(exp(delta_vec)^2)
set.seed(110)
data <- as.numeric(mu_vec + t(chol(Cov_mat + D_mat)) %*% rnorm(N))
# Set up constants
constants <- list( nu = 0.5, Sigma_HP1 = 2 )
# Defaults: tau_model = "constant", sigma_model = "constant", mu_model = "constant",
# and Sigma_model = "constant"
Rmodel <- nsgpModel(likelihood = "fullGP", constants = constants, coords = coords, data = data )
conf <- configureMCMC(Rmodel)
Rmcmc <- buildMCMC(conf)
Cmodel <- compileNimble(Rmodel)
Cmcmc <- compileNimble(Rmcmc, project = Rmodel)
samples <- runMCMC(Cmcmc, niter = 200, nburnin = 100)
# Prediction
predCoords <- as.matrix(expand.grid(seq(0,1,l=10),seq(0,1,l=10)))
postpred <- nsgpPredict( model = Rmodel, samples = samples, coords.predict = predCoords )
```

`orderCoordinatesMMD` *Order coordinates according to a maximum-minimum distance criterion.*

Description

`orderCoordinatesMMD` orders an array of (x,y) spatial coordinates according to the "maximum minimum distance" (MMD), as described in Guinness, 2018. (Points are selected to maximize their minimum distance to already- selected points).

Usage

```
orderCoordinatesMMD(coords, exact = FALSE)
```

Arguments

`coords`	N x 2 array of N 2-dimensional (x,y) spatial coordinates.
`exact`	Logical; `FALSE` uses a fast approximation to MMD ordering (and is almost always recommended), while `TRUE` uses exact MMD ordering but is infeasible for large number of locations.

Value

A list of distances matrices, with the following components:

`orderedCoords`	N x 2 matrix; contains the ordered spatial coordinates as `coords`.
`orderedIndicesNoNA`	N-vector; contains the ordered indices with any NA values removed.

Examples

```
coords <- cbind(runif(100), runif(100))
orderCoordinatesMMD(coords)
```

`rmnorm_nngp` *Function for the evaluating the NNGP approximate density.*

Description

`dmnorm_nngp` (and `rmnorm_nngp`) calculate the approximate NNGP likelihood for a fixed set of parameters (i.e., A and D matrices). Finally, the distributions must be registered within `nimble`.

Usage

```
rmnorm_nngp(n, mean, AD, nID, N, k)
```

Arguments

`n`	N-vector of data.
`mean`	N-vector with current values of the mean
`AD`	N x (k+1) matrix; the first k columns are the 'A' matrix, and the last column is the 'D' vector.
`nID`	N x k matrix of neighbor indices.
`N`	Scalar; number of data measurements.
`k`	Scalar; number of nearest neighbors.

Value

The NNGP approximate density.

`rmnorm_sgv` *Function for the evaluating the SGV approximate density.*

Description

`dmnorm_sgv` (and `rmnorm_sgv`) calculate the approximate SGV likelihood for a fixed set of parameters (i.e., the U matrix). Finally, the distributions must be registered within `nimble`.

Usage

```
rmnorm_sgv(n, mean, U, N, k)
```

Arguments

n	Vector of measurements
mean	Vector of mean valiues
U	Matrix of size N x 3; representation of a sparse N x N Cholesky of the precision matrix. The first two columns contain row and column indices, respectively, and the last column is the nonzero elements of the matrix.
N	Number of measurements in x
k	Number of neighbors for the SGV approximation.

Value

Not applicable.

R_sparse_chol *R_sparse_chol*

Description

R_sparse_chol

Usage

```
R_sparse_chol(i, j, x, n)
```

Arguments

i	Vector of row indices.
j	Vector of column indices.
x	Vector of values in the matrix.
n	Length of the vector

R_sparse_crossprod *nimble_sparse_crossprod*

Description

nimble_sparse_crossprod

Usage

```
R_sparse_crossprod(i, j, x, z, n, subset = -1, transp = 1)
```

Arguments

i	Vector of row indices.
j	Vector of column indices.
x	Vector of values in the matrix.
z	Vector to calculate the cross-product with.
n	Length of the vector
subset	Optional vector of rows to include in the calculation.
transp	Optional indicator of using the transpose

R_sparse_solve *nimble_sparse_solve*

Description

nimble_sparse_solve

Usage

```
R_sparse_solve(i, j, x, z)
```

Arguments

i	Vector of row indices.
j	Vector of column indices.
x	Vector of values in the matrix.
z	Vector to calculate the cross-product with.

R_sparse_tcrossprod *nimble_sparse_tcrossprod*

Description

nimble_sparse_tcrossprod

Usage

```
R_sparse_tcrossprod(i, j, x, subset = -1)
```

Arguments

i	Vector of row indices.
j	Vector of column indices.
x	Vector of values in the matrix.
subset	Optional vector of rows to include in the calculation.

sgvSetup *One-time setup wrapper function for the SGV approximation*

Description

sgvSetup is a wrapper function that sets up the SGV approximation. Three objects are required: (1) ordering the locations, (2) identify nearest neighbors, and (3) determine the conditioning set. This function only needs to be run once per SGV analysis.

Usage

```
sgvSetup(
  coords,
  coords_pred = NULL,
  k = 15,
  seed = NULL,
  pred.seed = NULL,
  order_coords = TRUE
)
```

Arguments

coords	Matrix of observed locations.
coords_pred	Optional matrix of prediction locations.
k	Number of neighbors.
seed	Setting the seed for reproducibility of the observed location ordering
pred.seed	Setting the seed for reproducibility of the prediction ordering.
order_coords	Logical; should the coordinates be ordered.

Value

A list with the following components:

ord	A vector of ordering position for the observed locations.
ord_pred	A vector of ordering position for the prediction locations (if coords_pred is provided).
ord_all	A concatenated vector of ord and ord_pred.
coords_ord	A matrix of ordered locations (observed and prediction), included for convenience.
nID_ord	A matrix of (ordered) neighbor indices.
condition_on_y_ord	A matrix indicating whether the conditioning set for each (ordered) location is on the latent process (y, 1) or the observed values (z, 0).

Index