Package 'BayesNSGP'

September 10, 2026

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) <doi:10.48550/arXiv.1702.00434>). 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. Also provided are nearest-neighbor Gaussian process components for use directly in user-written model code, where the spatial process is retained as a latent field: neighbor-structure construction, a latent-field density and matching simulation function, a purpose-built Metropolis-Hastings sampler that updates the field one node at a time, and posterior prediction at unobserved locations.

Version 0.3.1

Date 2026-09-10

Maintainer Daniel Turek <danielturek@gmail.com>

Depends R (>= 3.4.0),nimble

Imports FNN,Matrix,methods,StatMatch,sf,ggplot2

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

License GPL-3

Encoding UTF-8

Config/roxygen2/version 8.1.0

RoxygenNote 7.3.2

NeedsCompilation no

Author Daniel Turek [aut, cre],
Mark Risser [aut],
Fabian Ketwaroo [aut]

Repository CRAN

Date/Publication 2026-09-10 11:00:02 UTC

Contents

calcQF . . . 3
calculateAD_ns . . . 3
calculateU_ns . . . 5
computeAD . . . 6
computeC . . . 8
computeNeighbors . . . 9
computeQF . . . 10
conditionLatentObs . . . 11
crossCy_sm . . . 12
Cy_sm . . . 14
determineNeighbors . . . 15
displayNeighbors . . . 16
dmnorm_gp2Scale . . . 17
dmnorm_nngp . . . 18
dmnorm_NN_GP . . . 19
dmnorm_sgv . . . 20
expcov . . . 21
get_single_reverse_neighbors . . . 22
inverseEigen . . . 23
matern_corr . . . 24
nimble_sparse_chol . . . 25
nimble_sparse_cholesky . . . 25
nimble_sparse_crossprod . . . 26
nimble_sparse_solve . . . 26
nimble_sparse_solveMat . . . 27
nimble_sparse_tcrossprod . . . 27
NNGP.pred . . . 28
nsCorr . . . 29
nsCrosscorr . . . 31
nsCrossdist . . . 32
nsCrossdist3d . . . 33
nsDist . . . 34
nsDist3d . . . 35
nsgpModel . . . 36
nsgpPredict . . . 38
orderCoordinatesMMD . . . 40
rmnorm_gp2Scale . . . 40
rmnorm_nngp . . . 41
rmnorm_NN_GP . . . 42
rmnorm_sgv . . . 43
RWNNGP_setup . . . 44
R_sparse_chol . . . 45
R_sparse_cholesky . . . 46
R_sparse_crossprod . . . 46
R_sparse_solve . . . 47
R_sparse_solveMat . . . 47

R_sparse_tcrossprod 48
sampler_RW_NN_GP 48
sgvSetup 50

Index **51**

`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.

`computeAD` *Compute NNGP regression coefficients and conditional variances*

Description

Computes the Nearest-Neighbor Gaussian Process (NNGP) factorization components for each location: the regression coefficients (A) and conditional variances (D) following the algorithms described by Finley et al. (2019)

Usage

```
computeAD(edist, nid.dist, neighbors.id, rho, sigma2, k)
```

Arguments

edist	Numeric matrix ($M \times M$). Pairwise Euclidean distances between spatial locations.
nid.dist	Numeric matrix ($M \times (k+1)$). Distances from each location to its ordered neighbor set, including self-distance in the last column.
neighbors.id	Integer matrix ($M \times k$). Indices of neighbor locations for each spatial location, based on the chosen ordering.
rho	Positive numeric scalar. Length-scale parameter ρ.
sigma2	Positive numeric scalar. Marginal variance σ^2.
k	Positive integer. Maximum number of neighbors.

Details

This function computes the parameters of the NNGP factorization by expressing the joint distribution as a product of conditional Gaussian densities:

$$p(\mathbf{x}) = \prod_{i=1}^{M} p(x_i \mid x_{N(i)})$$

where $N(i)$ denotes the set of neighbors of location i.

For each location i, the conditional distribution is:

$$x_i \mid x_{N(i)} \sim \mathcal{N}\left(A_i x_{N(i)},\ D_i\right)$$

where:

- A_i is a vector of regression coefficients
- D_i is the conditional variance

These quantities are obtained from the covariance structure:

- $A_i = C_{i,N(i)} C^{-1}_{N(i),N(i)}$
- $D_i = C_{i,i} - C_{i,N(i)} C^{-1}_{N(i),N(i)} C_{N(i),i}$

where:

- $C_{N(i),N(i)}$ is the covariance matrix among neighbors
- $C_{i,N(i)}$ is the covariance between location i and its neighbors

The covariance structure is defined using an exponential covariance function.

These parameters are used to efficiently evaluate the NNGP likelihood and simulate from the process.

Value

A numeric matrix ($M \times (k+1)$) where:

- First k columns: regression coefficients (A matrix)
- Last column: conditional variances (D diagonal entries)

Author(s)

Fabian Ketwaroo

References

Finley, A. O., Datta, A., Cook, B. D., Morton, D. C., Andersen, H. E., & Banerjee, S. (2019). Efficient algorithms for Bayesian nearest neighbor Gaussian processes. *Journal of Computational and Graphical Statistics*, 28(2), 401–414. doi:10.1080/10618600.2018.1537924

Examples

```
coords <- matrix(runif(40), ncol = 2)
Nk = 5
res <- computeNeighbors(coords, k = Nk)
AD <- computeAD(edist = res$edist_sorted,
                nid.dist = res$neighbors_dist,
                neighbors.id = res$neighbor_idx,
                rho = 0.1,
                sigma2 = 0.3,
                k = Nk)
```

computeC *Construct local covariance matrix for NNGP conditional distributions*

Description

Constructs the local covariance matrix associated with a focal location and its neighbor set under the Nearest-Neighbor Gaussian Process (NNGP).

Usage

```
computeC(fdist, ndist, rho, sigma2)
```

Arguments

fdist	Numeric matrix ($k \times k$) containing pairwise Euclidean distances among the neighbors of the focal location.
ndist	Numeric vector (length $k + 1$) containing distances between the focal location and its neighbors, followed by the self-distance (0) of the focal location.
rho	Positive numeric scalar. Length-scale parameter ρ controlling the decay of spatial correlation.
sigma2	Positive numeric scalar. Marginal variance parameter σ^2.

Details

This function constructs the local covariance matrix used in the NNGP approximation for conditional Gaussian distributions. The covariance structure is defined using an exponential covariance function.

The structure of the matrix is:

- Top-left block: covariance among neighbors
- Last row/column: covariance between focal location and neighbors
- Bottom-right: marginal variance of the focal location

A small nugget term (1e-6) is added to the diagonal for numerical stability.

Value

A numeric matrix of dimension $(k + 1) \times (k + 1)$ representing the local covariance matrix for the focal location and its neighbors.

Author(s)

Fabian Ketwaroo

`computeNeighbors` *Compute neighbor sets for a Nearest-Neighbor Gaussian Process (NNGP)*

Description

Computes ordered neighbor sets based on Vecchia's approximation for a set of spatial coordinates. Locations are first ordered by their x-coordinate, and neighbors are selected from previously ordered locations.

Usage

```
computeNeighbors(coords, k, proj)
```

Arguments

`coords`	Numeric matrix ($M \times 2$) of spatial coordinates, where each row corresponds to a location.
`k`	Positive integer. Number of neighbors for each location.
`proj`	Coordinate reference system (CRS) used to compute distances. Can be an EPSG code (e.g., 4326) or a CRS object compatible with `sf`.

Details

Neighbor selection follows a Vecchia-type ordering. For each location i, neighbors are chosen from the set of previously ordered locations $1, \ldots, i-1$.

Distances are computed using `sf::st_distance` and depend on the CRS.

The first location has no neighbors. For the first $k+1$ locations, all previous locations are used as neighbors. For subsequent locations, the k nearest neighbors among previously ordered points are selected.

Value

A list with components:

coords_sorted Matrix of coordinates sorted by x-coordinate

edist_sorted Matrix ($M \times M$) of pairwise distances between sorted coordinates

neighbors Binary matrix ($M \times M$) indicating neighbor relationships

neighbor_idx Matrix ($M \times k$) of neighbor indices for each location

neighbors_dist Matrix ($M \times k$) of distances to neighbors

Author(s)

Fabian Ketwaroo

See Also

st_distance

Examples

```
coords <- matrix(runif(40), ncol = 2)
res <- computeNeighbors(coords, k = 5, proj = 4326)
```

---

computeQF *Compute quadratic form for NNGP log-likelihood*

Description

Evaluates the quadratic form appearing in the Gaussian log-likelihood under the Nearest-Neighbor Gaussian Process (NNGP).

Usage

```
computeQF(u, v, AD, neighbors.id)
```

Arguments

u	Numeric vector (length M). Typically centered data ($x - \mu$).
v	Numeric vector (length M). Typically identical to u in symmetric cases.
AD	Numeric matrix ($M \times (k+1)$). Output from computeAD, where: • Columns $1:k$ contain regression coefficients A_i • Column $k+1$ contains conditional variances D_i
neighbors.id	Integer matrix ($M \times k$). Neighbor indices for each location.

Details

This function computes the quadratic form associated with the NNGP precision matrix without explicitly constructing the full covariance or precision matrix.

The expression is obtained by decomposing the joint Gaussian density into a product of conditional densities:

$$p(\mathbf{x}) = \prod_{i=1}^{M} p(x_i \mid x_{N(i)})$$

where $N(i)$ denotes the set of neighbors of location i.

Each term corresponds to a squared, standardized conditional residual:

$$r_i = u_i - A_i u_{N(i)}$$

and the quadratic form is:

$$\sum_i \frac{r_i^2}{D_i}$$

This provides an efficient way to evaluate the Gaussian log-likelihood under the NNGP approximation.

Value

A scalar value representing the quadratic form:

$$\sum_{i=1}^{M} \frac{\left(u_i - A_i u_{N(i)}\right)\left(v_i - A_i v_{N(i)}\right)}{D_i}$$

Author(s)

Fabian Ketwaroo

See Also

computeAD

`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`).

crossCy_sm *Calculate sparse kernel, core kernel, and determine nonzero entries*

Description

Cy_sm calculates the normalized sparse kernel for a fixed set of bump function hyperparameters and returns the nonzero entries. Note that the matrix is calculated and returned in dense format.

Usage

```
crossCy_sm(
  Xdists,
  coords,
  Pcoords,
  d,
  n1,
  n2,
  r0,
  s0,
  cstat_opt,
  normalize,
  bumpLocs,
  rads,
  ampls,
  shps,
  Xdist1_sq,
  Xdist2_sq,
  Xdist12,
  Sigma11,
  Sigma22,
  Sigma12,
  PSigma11,
  PSigma22,
  PSigma12,
  nu,
  log_sigma_vec,
  Plog_sigma_vec
)
```

Arguments

Xdists	N x N matrix of Euclidean distances
coords	N x d matrix of coordinate/input locations
Pcoords	N x d matrix of coordinate/input locations
d	Scalar; dimension of the spatial domain.
n1	Scalar; number of outer products.

n2	Scalar; number of bump functions in each outer product.
r0	Scalar; length-scale of sparse stationary kernel.
s0	Scalar; signal-variance of sparse stationary kernel.
cstat_opt	Scalar; determines the compactly supported kernel. See Details.
normalize	Logical; should C_sparse have 1's along the diagonal (1 = TRUE)
bumpLocs	Array of bump function locations (n2*d x n1)
rads	Matrix of bump function radii (n1 x n2; denoted r_{ij})
ampls	Matrix of bump function amplitudes (n1 x n2; denoted a_{ij})
shps	Matrix of bump function shape parameters (n1 x n2; denoted b_{ij})
Xdist1_sq	N x N matrix; contains values of pairwise squared distances in the x-coordinate.
Xdist2_sq	N x N matrix; contains values of pairwise squared distances in the y-coordinate.
Xdist12	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.
PSigma11	Vector of length N; contains the 1-1 element of the anisotropy process for each station.
PSigma22	Vector of length N; contains the 2-2 element of the anisotropy process for each station.
PSigma12	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.
log_sigma_vec	Vector of length N; log of the signal standard deviation.
Plog_sigma_vec	Vector of length N; log of the signal standard deviation.

Value

Returns a sparse matrix (N x 3) of the nonzero elements of the product between the core and sparse kernel.

Cy_sm *Calculate sparse kernel, core kernel, and determine nonzero entries*

Description

Cy_sm calculates the normalized sparse kernel for a fixed set of bump function hyperparameters and returns the nonzero entries. Note that the matrix is calculated and returned in dense format.

Usage

```
Cy_sm(
  dists,
  coords,
  N,
  d,
  n1,
  n2,
  r0,
  s0,
  cstat_opt,
  normalize,
  bumpLocs,
  rads,
  ampls,
  shps,
  dist1_sq,
  dist2_sq,
  dist12,
  Sigma11,
  Sigma22,
  Sigma12,
  nu,
  log_sigma_vec,
  lognuggetSD
)
```

Arguments

dists	N x N matrix of Euclidean distances
coords	N x d matrix of coordinate/input locations
N	Scalar; number of data measurements.
d	Scalar; dimension of the spatial domain.
n1	Scalar; number of outer products.
n2	Scalar; number of bump functions in each outer product.
r0	Scalar; length-scale of sparse stationary kernel.

s0	Scalar; signal-variance of sparse stationary kernel.
cstat_opt	Scalar; determines the compactly supported kernel. See Details.
normalize	Logical; should C_sparse have 1's along the diagonal
bumpLocs	Array of bump function locations (n2*d x n1)
rads	Matrix of bump function radii (n1 x n2; denoted r_{ij})
ampls	Matrix of bump function amplitudes (n1 x n2; denoted a_{ij})
shps	Matrix of bump function shape parameters (n1 x n2; denoted b_{ij})
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.
log_sigma_vec	Vector of length N; log of the signal standard deviation.
lognuggetSD	Vector of length N; log of the error standard deviation.

Value

Returns a sparse matrix (N x 3) of the nonzero elements of the product between the core and sparse kernel.

---

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)
```

---

displayNeighbors *Plot neighbor sets for a Nearest-Neighbor Gaussian Process (NNGP)*

Description

Visualizes the neighbor set for a given location based on an ordered Nearest-Neighbor Gaussian Process (NNGP) constructed using Vecchia's approximation.

Usage

```
displayNeighbors(ind, coords.sorted, neighbor_idx, proj)
```

Arguments

ind	Integer index of the focal location (in the sorted order).
coords.sorted	Numeric matrix ($M \times 2$) of spatial coordinates sorted according to the ordering used in the NNGP construction. Output from computeNeighbors.
neighbor_idx	Integer matrix ($M \times k$) where each row contains the indices of the neighbors for a given location. Output from computeNeighbors.
proj	Coordinate reference system (CRS) used for plotting. Can be an EPSG code or a CRS object compatible with sf.

Details

The function assumes that coordinates and neighbor indices are based on an ordered representation of the data (e.g., sorted by x-coordinate or another ordering used in Vecchia's approximation).

The neighbor set for location i consists of previously ordered locations, as defined by the NNGP construction.

Value

A ggplot2 object showing:

- All spatial locations (black markers)
- The focal location (red point)
- Its neighbors (blue points)
- A vertical reference line at the focal location's x-coordinate

Author(s)

Fabian Ketwaroo

See Also

computeNeighbors

Examples

```
coords <- matrix(runif(40), ncol = 2)
res <- computeNeighbors(coords, k = 5, proj = 4326)
p <- displayNeighbors(
  ind = 10,
  coords.sorted = res$coords_sorted,
  neighbor_idx = res$neighbor_idx,
  proj = 4326
)
print(p)
```

---

dmnorm_gp2Scale *Function for the evaluating the Gaussian likelihood with gp2Scale sparse covariance.*

Description

dmnorm_gp2Scale (and rmnorm_gp2Scale) calculate the usual Gaussian likelihood for a fixed set of parameters (but with sparse matrices). Finally, the distributions must be registered within nimble.

Usage

```
dmnorm_gp2Scale(x, mean, Cov, N, Nnz, log = 1)
```

Arguments

x	Vector of measurements
mean	Vector of mean values
Cov	Matrix of size N x N; sparse kernel
N	Number of measurements in x
Nnz	Number of measurements in x
log	Logical; should the density be evaluated on the log scale.

Value

Returns the Gaussian likelihood using the gp2Scale sparse covariance.

---

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_NN_GP *Compute Nearest-Neighbor Gaussian Process (NNGP) log-density*

Description

Evaluates the log-density of a multivariate normal distribution under the Nearest-Neighbor Gaussian Process (NNGP) approximation as defined by Data et al 2016.

Usage

```
dmnorm_NN_GP(x, mu, AD, neighbors.id, log = 1)
```

Arguments

x	Numeric vector (length M). Observed spatial process values.
mu	Numeric vector (length M). Mean vector.
AD	Numeric matrix ($M \times (k+1)$). Output from computeAD, where: • Columns $1:k$ contain regression coefficients A_i • Column $k+1$ contains conditional variances D_i
neighbors.id	Integer matrix ($M \times k$). Neighbor indices defining the conditioning sets $N(i)$. Output from computeNeighbors
log	Logical/Integer. If TRUE (or 1), returns the log-density; otherwise returns the density.

Details

The NNGP approximation factorizes the joint density as:

$$p(\mathbf{x}) \approx \prod_{i=1}^{M} p(x_i \mid x_{N(i)})$$

leading to the log-density:

$$-\frac{1}{2}\left(N\log(2\pi) + \sum_i \log D_i + Q\right)$$

where:

- D_i are conditional variances
- Q is the quadratic form computed by computeQF

The quadratic form corresponds to:

$$Q = \sum_{i=1}^{M} \frac{(x_i - \mu_i - A_i(x_{N(i)} - \mu_{N(i)}))^2}{D_i}$$

This formulation avoids constructing the sparse covariance matrix, enabling scalable likelihood evaluation for large spatial datasets.

Value

A scalar value corresponding to the (log-)density.

Author(s)

Fabian Ketwaroo

References

Datta, A., Banerjee, S., Finley, A.O. and Gelfand, A.E., 2016. Hierarchical nearest-neighbor Gaussian process models for large geostatistical datasets. *Journal of the American Statistical Association*, 111(514), 800-812. doi:10.1080/01621459.2015.1044091

Examples

```
M <- 2000      # number of spatial locations
coords <- matrix(runif(2*M), ncol = 2)
Nk <- 15       # number of neighbors
res <- computeNeighbors(coords, k = Nk)
AD <- computeAD(edist = res$edist_sorted,
                nid.dist = res$neighbors_dist,
                neighbors.id = res$neighbor_idx,
                rho = 0.1,
                sigma2 = 0.3,
                k = Nk)
w <- rmnorm_NN_GP(n = 1, mu = rep(0, M),
                  AD = AD[1:M, 1:(Nk+1)],
                  neighbors.id = res$neighbor_idx[1:M, 1:Nk])
loglike <- dmnorm_NN_GP(w, mu = rep(0, M),
                        AD = AD[1:M, 1:(Nk+1)],
                        neighbors.id = res$neighbor_idx[1:M, 1:Nk],
                        log = TRUE)
```

`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.

---

expcov *Exponential covariance function for Gaussian processes*

Description

Computes the exponential covariance matrix.

Usage

```
expcov(edists, rho, sigma2)
```

Arguments

edists	Numeric matrix ($M \times M$) of pairwise Euclidean distances between spatial locations.
rho	Positive numeric scalar. Length-scale parameter ρ controlling the decay of correlation with distance.
sigma2	Positive numeric scalar. Signal variance parameter σ^2 controlling the overall variance.

Details

The covariance function is defined as:

$$C_{i,j} = \sigma^2 \exp(-d_{i,j}/\rho)$$

where d_{ij} is the distance between locations i and j.

Larger values of ρ imply slower decay of correlation with distance (i.e., stronger spatial autocorrelation). Larger values of σ^2 increase the marginal variance of the process.

A small nugget term (1e-6) is added to the diagonal for numerical stability.

Value

A numeric matrix ($M \times M$) representing the covariance between spatial locations.

Author(s)

Fabian Ketwaroo

Examples

```
coords <- matrix(runif(10), ncol = 2)
d <- as.matrix(dist(coords))
expcov(edists = d, rho = 0.5, sigma2 = 1)
```

---

get_single_reverse_neighbors

Find reverse neighbors for a single NNGP node

Description

Identifies all "children" (reverse neighbors) for a specific target location j in a Nearest-Neighbor Gaussian Process (NNGP). It searches the neighbor matrix to find all locations k that condition upon location j.

Usage

```
get_single_reverse_neighbors(target_node, neighbor_idx)
```

Arguments

`target_node`	Integer. The index of the location j whose reverse neighbors you want to find.
`neighbor_idx`	Integer matrix ($M \times k$). Neighbor indices for all locations, output from `computeNeighbors`.

Details

This function optimizes the reverse lookup for a single node by avoiding the full $O(M)$ list construction loop. It uses vectorization to scan rows of the neighbor matrix where the row index k is greater than the target index j, enforcing the Directed Acyclic Graph (DAG) structure of the NNGP.

Value

An integer vector containing the indices of all locations k that have `target_node` as a neighbor, maintaining the NNGP ordering constraint ($j < k$).

Author(s)

Fabian Ketwaroo

Examples

```
nn_matrix <- matrix(c(0, 0,
                      1, 0,
                      1, 2,
                      1, 3), nrow = 4, byrow = TRUE)

# Find which nodes have node 1 as a neighbor
get_single_reverse_neighbors(target_node = 1, neighbor_idx = nn_matrix)
```

`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_cholesky *nimble_sparse_chol*

Description

nimble_sparse_chol

Usage

```
nimble_sparse_cholesky(i, j, x)
```

Arguments

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

`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_solveMat *nimble_sparse_crossprod*

Description

nimble_sparse_crossprod

Usage

```
nimble_sparse_solveMat(i, j, x, z, 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.
transp	Optional indicator of using the transpose

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.

`NNGP.pred` *Posterior predictive sampling for NNGP*

Description

Generates a posterior predictive draw at a new location x_0 using the Nearest-Neighbor Gaussian Process (NNGP) approximation. This function automatically handles coordinate sorting, neighbor selection, and distance calculations internally following Algorithm 2 of Finley et al. (2019).

Usage

```
NNGP.pred(x0, coords_sorted, rho, sigma2, w.x0.all, k)
```

Arguments

`x0`	Numeric matrix (1×2) or vector of length 2 representing the coordinates of the new prediction location.
`coords_sorted`	Numeric matrix ($M \times 2$) of the observed training locations. Output from `computeNeighbors`.
`rho`	Positive numeric scalar. Length-scale parameter ρ.
`sigma2`	Positive numeric scalar. Marginal variance parameter σ^2.
`w.x0.all`	Numeric vector of length M. The realizations of the spatial process (random effects) at all observed locations, ordered to correspond with `coords_sorted`.
`k`	Integer. The number of nearest neighbors to consider for the prediction approximation.

Details

The function merges the prediction point x_0 into the training coordinates and identifies the k nearest neighbors from the set of points that precede it in the x-axis ordering.

The predictive distribution is Gaussian:

$$w(x_0) \mid \mathbf{w}_{N_0} \sim \mathcal{N}(m, v)$$

where m is the **Kriging mean** and v is the **Kriging variance** defined as:

- $m = c^\top C_{N_0}^{-1} \mathbf{w}_{N_0}$
- $v = C(x_0, x_0) - c^\top C_{N_0}^{-1} c$

Here:

- N_0 is the set of indices of the k nearest neighbors of x_0 among the observed locations.
- $\mathbf{w}_{N_0}$ is the vector of realizations of the spatial process at those neighbor locations (passed as `w.s.x0`).
- c is the $k \times 1$ covariance vector between x_0 and its neighbors in N_0.
- C_{N_0} is the $k \times k$ covariance matrix among the neighbors in N_0.

The exponential covariance function is used: $C(d) = \sigma^2 \exp(-d/\rho)$. A small nugget ($10^{-6}$) is added to the diagonal of C_{N_0} for numerical stability.

The internal subsetting of `w.x0.all` ensures that $\mathbf{w}_{N_0}$ correctly represents the process realizations at the identified neighbor locations.

Value

A numeric scalar representing a posterior predictive draw at location x_0.

Author(s)

Fabian Ketwaroo

References

Finley, A. O., Datta, A., Cook, B. D., Morton, D. C., Andersen, H. E., & Banerjee, S. (2019). Efficient algorithms for Bayesian nearest neighbor Gaussian processes. *Journal of Computational and Graphical Statistics*, 28(2), 401–414. doi:10.1080/10618600.2018.1537924

Examples

```
# Setup training data
M <- 50
coords <- matrix(runif(M * 2), ncol = 2)
w_all <- rnorm(M) # Mock spatial realizations

# coords_sorted and w.x0.all must be aligned to the same x-sorted order
# that computeNeighbors() would produce, since NNGP.pred() selects
# neighbors by position in that order.
ord <- order(coords[, 1])
coords_sorted <- coords[ord, ]
w_sorted <- w_all[ord]

# Predict at a new location using k = 5 neighbors
new_loc <- c(0.5, 0.5)
pred_draw <- NNGP.pred(x0 = new_loc,
                       coords_sorted = coords_sorted,
                       rho = 0.2,
                       sigma2 = 1.0,
                       w.x0.all = w_sorted,
                       k = 5)
```

`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",
  sparse_model = "none",
  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)
sparse_model	Character; specifies the model to be used for the sparse kernel when using the gp2Scale likelihood. Defaults to "none".
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 = list(),
  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

```
## Not run:
# 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 )

## End(Not run)
```

`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_gp2Scale` *Function for the evaluating the SGV approximate density.*

Description

`dmnorm_gp2Scale` (and `rmnorm_gp2Scale`) calculate the usual Gaussian likelihood for a fixed set of parameters (but with sparse matrices). Finally, the distributions must be registered within `nimble`.

Usage

```
rmnorm_gp2Scale(n, mean, Cov, N, Nnz)
```

Arguments

n	Number of realizations to generate
mean	Vector of mean values
Cov	Matrix of size N x N; sparse kernel
N	Number of measurements in x
Nnz	Number of measurements in x

Value

Not applicable.

---

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_NN_GP` *Simulate from a Nearest-Neighbor Gaussian Process (NNGP)*

Description

Efficiently simulates a realization from a Gaussian process using the Nearest-Neighbor Gaussian Process (NNGP) approximation as described by Datta (2022).

Usage

```
rmnorm_NN_GP(n, mu, AD, neighbors.id)
```

Arguments

`n`	Integer. Number of samples to generate. Currently only supports `n = 1`.
`mu`	Numeric vector (length M). Mean vector of the process.
`AD`	Numeric matrix ($M \times (k+1)$). Output from `computeAD`, where: • Columns $1:k$ contain regression coefficients (A matrix) • Column $k+1$ contains conditional variances (D)
`neighbors.id`	Integer matrix ($M \times k$). Neighbor indices for each location, defining the conditioning set $N(i)$. Output from `computeNeighbors`.

Details

The simulation follows the recursive form derived from the sparse Cholesky factorization of the precision matrix (Σ^{-1}). Let $z = x - \mu$ be the zero-mean spatial residuals. The NNGP simulates these residuals sequentially:

$$z_i = \sum_{j \in N(i)} A_{ij} z_j + \epsilon_i$$

where:

- $N(i)$ is the neighbor set of location i, such that $j < i$.
- A_{ij} are the kriging weights computed from the local covariance.
- $\epsilon_i \sim \mathcal{N}(0, D_i)$ are independent innovations.

This sequential approach exploits the sparsity of the Cholesky factor L, where $\Sigma^{-1} \approx (I - A)^\top D^{-1}(I - A)$ and allows for $O(Mk^3)$ simulation, which is significantly faster than the standard $O(M^3)$ Cholesky decomposition for large M.

Value

A numeric vector (length M) representing a simulated realization from the NNGP.

Author(s)

Fabian Ketwaroo

References

Datta, A. (2022). Nearest-neighbor sparse Cholesky matrices in spatial statistics. *Wiley Interdisciplinary Reviews: Computational Statistics*, 14(5), e1574. doi:10.1002/wics.1574

Examples

```
M <- 2000      # number of spatial locations
coords <- matrix(runif(2*M), ncol = 2)
Nk <- 15       # number of neighbors
res <- computeNeighbors(coords, k = Nk)
AD <- computeAD(edist = res$edist_sorted,
                nid.dist = res$neighbors_dist,
                neighbors.id = res$neighbor_idx,
                rho = 0.1,
                sigma2 = 0.3,
                k = Nk)
w <- rmnorm_NN_GP(n = 1, mu = rep(0, M),
                  AD = AD[1:M, 1:(Nk+1)],
                  neighbors.id = res$neighbor_idx[1:M, 1:Nk])
```

`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.

RWNNGP_setup *Helper to extract NNGP local neighborhood nodes for custom MCMC sampling*

Description

Pre-calculates the local graph structure, target indices, and specific coordinate strings within the AD matrix for a single target node. This forms the control list payload needed by sampler_RW_NN_GP.

Usage

```
RWNNGP_setup(node_id, AD, neighbors.id, Rneighbors.id, N.neighbors, k = k)
```

Arguments

node_id	Integer. The index of the specific spatial location currently being set up for targeted random-walk sampling (ranges from 1 to M).
AD	Character. The name of the matrix containing the NNGP coefficients within the NIMBLE model object (typically passed as a string like "AD").
neighbors.id	Integer matrix ($M \times k$). The forward neighbor index tracking matrix where rows indicate the target location.
Rneighbors.id	Integer vector. The reverse neighbor indices (children) for node_id, computed by get_single_reverse_neighbors.
N.neighbors	Integer vector. A vector of length M storing the exact number of active forward neighbors assigned to each location.
k	Integer. The maximum number of neighbors specified in the NNGP configuration.

Details

This utility is designed to run in an R loop when configuring an MCMC specification prior to model compilation. By converting matrix indexing operations into explicit character node paths (e.g., "AD[5,1]"), it shifts the burden of matrix searching from the runtime execution loop of the C++ compiled sampler into a one-time R setup cost.

It maps out how a change in node_id will ripple through its forward neighbors and back through the reverse neighbors whose conditional distributions depend directly on the target node's value.

Value

A named list containing specific structural elements for the target node:

- `update_id`: Integer vector combining the `node_id` and its reverse neighbors.
- `Fneighbors.id`: Integer vector containing non-zero forward neighbor indices.
- `AFnodes`: Character vector of parsed string addresses pointing to forward coefficients.
- `ARnodes`: Character vector of parsed string addresses pointing to reverse coefficients.
- `A.neighbors`: Flattened character vector of matrix addresses representing all relevant neighborhood coefficients needed to re-evaluate residuals during the loop.

Author(s)

Fabian Ketwaroo

`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_cholesky *R_sparse_chol*

Description

R_sparse_chol

Usage

```
R_sparse_cholesky(i, j, x)
```

Arguments

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

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_solveMat *nimble_sparse_crossprod*

Description

nimble_sparse_crossprod

Usage

```
R_sparse_solveMat(i, j, x, z, 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.
transp	Optional indicator of using the transpose

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.

sampler_RW_NN_GP *NNGP-Specific Random Walk Metropolis-Hastings Sampler*

Description

A custom NIMBLE sampler for updating individual spatial random effects under a Nearest-Neighbor Gaussian Process (NNGP) approximation. This sampler uses a factorized likelihood approach to perform highly efficient local Metropolis-Hastings sampling with a normal proposal distribution (Metropolis, 1953), implementing the adaptation routine given in Shaby and Wells (2011).

Usage

```
sampler_RW_NN_GP(model, mvSaved, target, control)
```

Arguments

model	(uncompiled) model on which the MCMC is to be run
mvSaved	modelValues object to be used to store MCMC samples
target	The node to be sampled (a single spatial random effect scalar).
control	A list of control parameters:

- AD: Character (default "AD"). The name of the matrix containing the NNGP coefficients within the NIMBLE model object.
- neighbors.id: Character (default "neighbors.id"). The name of the forward neighbor matrix within the NIMBLE model object.

- `adaptive`: Logical (default `TRUE`). Whether to use an adaptive step-size procedure.
- `adaptInterval`: Integer (default `200`). Number of iterations between adaptive adjustments.
- `scale`: Numeric (default `1`). Initial scale/standard deviation for the random walk proposal.

Details

Instead of calculating the full NNGP log-likelihood, which is $O(M)$ (where M is the total number of spatial locations), this sampler exploits the Directed Acyclic Graph (DAG) structure of the NNGP to reduce the complexity of a single-node update to $O(k^2)$, where k is the number of neighbors.

During the MCMC configuration stage, the sampler automatically queries the local graph architecture using `get_single_reverse_neighbors` and `RWNNGP_setup` to resolve dependencies internally. When a single node w_i is updated, only its own conditional density and the conditional densities of its "children" (the reverse neighbors that depend on it) are affected.

The local log-Metropolis-Hastings ratio ($\log MHR$) is calculated as:

$$\log MHR = -\frac{(r_i^{*2} - r_i^2)}{2D_i} + \sum_{j \in \mathcal{R}(i)} -\frac{(r_j^{*2} - r_j^2)}{2D_j}$$

where:

- r_i and r_i^* are the current and proposed conditional residuals for the target node.
- $\mathcal{R}(i)$ is the set of reverse neighbors (indices j such that $i \in N(j)$).
- r_j and r_j^* are the current and proposed residuals for those reverse neighbors.
- D represents the conditional variances.

The residuals are updated efficiently using the difference $\delta = w_i^* - w_i$, such that $r_j^* = r_j - A_{ji}\delta$, where A_{ji} is the NNGP regression coefficient.

Author(s)

Fabian Ketwaroo

References

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6), 1087-1092.

Shaby, B. A. and Wells, M. T. (2011). Exploring an adaptive Metropolis-Hastings algorithm. *Department of Statistical Science, Cornell University Tech Report*.

`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,
  order_coords_pred = 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.
`order_coords_pred`	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

calcQF, 3
calculateAD_ns, 3
calculateU_ns, 5
computeAD, 6
computeC, 8
computeNeighbors, 9
computeQF, 10
conditionLatentObs, 11
crossCy_sm, 12
Cy_sm, 14

determineNeighbors, 15
displayNeighbors, 16
dmnorm_gp2Scale, 17
dmnorm_NN_GP, 19
dmnorm_nngp, 18
dmnorm_sgv, 20

expcov, 21

get_single_reverse_neighbors, 22

inverseEigen, 23

matern_corr, 24

nimble_sparse_chol, 25
nimble_sparse_cholesky, 25
nimble_sparse_crossprod, 26
nimble_sparse_solve, 26
nimble_sparse_solveMat, 27
nimble_sparse_tcrossprod, 27
NNGP.pred, 28
nsCorr, 29
nsCrosscorr, 31
nsCrossdist, 32
nsCrossdist3d, 33
nsDist, 34
nsDist3d, 35
nsgpModel, 36
nsgpPredict, 38

orderCoordinatesMMD, 40

R_sparse_chol, 45
R_sparse_cholesky, 46
R_sparse_crossprod, 46
R_sparse_solve, 47
R_sparse_solveMat, 47
R_sparse_tcrossprod, 48
rmnorm_gp2Scale, 40
rmnorm_NN_GP, 42
rmnorm_nngp, 41
rmnorm_sgv, 43
RWNNGP_setup, 44

sampler_RW_NN_GP, 48
sgvSetup, 50
st_distance, *10*