



genVecQMC . . . . .	9
mvtNormQuasiMonteCarlo . . . . .	10
mvTProbQuasiMonteCarlo . . . . .	11
rExtremalStudentParetoProcess . . . . .	13
scoreEstimation . . . . .	14
simulBrownResnick . . . . .	16
simulPareto . . . . .	17
spectralLikelihood . . . . .	18

<b>Index</b>	<b>20</b>
--------------	-----------

---

mvPot-package	<i>Multivariate Peaks-over-Threshold Modelling for Extreme Events Analysis</i>
---------------	--

---

## Description

The mvPot package provides functions to perform high-dimensional peaks-over-threshold inference of spatial processes such as the Brown–Resnick. Parallel implementation for censored likelihood allows up to 500 locations, whereas the gradient score can handle thousands of locations. The package also includes simulations algorithms for the Brown-Resnick max-stable process as well as its associated Pareto process. A tutorial describing a complete case study of Red Sea temperature anomalies extremes can be found at [https://github.com/r-fndv/mvPot\\_tutorial](https://github.com/r-fndv/mvPot_tutorial).

## Details

The mvPot package provides functions to perform high-dimensional peaks-over-threshold inference of spatial processes such as the Brown–Resnick.

spectralLikelihood relies on the spectral likelihood as developed by Engelke et al. (2015). This methods is fast to compute, however it is not robust with regard to non-extreme components.

censoredLikelihoodBR (Wadsworth and Tawn, 2013) is a likelihood function for exceedances with at least one component exceeding a threshold and where low components, i.e., components under their threshold,. This approach is robust and performs best but requires heavy computations. The implementation in this package makes use of quasi-Monte Carlo estimation and thus can handle 100 locations in a reasonable time and up to 500 when parallelized. The analog function for extremal Student processes is censoredLikelihoodXS.

scoreEstimation is a faster alternative to the censoredLikelihood, which is more robust than spectralLikelihood. This method can also be used with any kind of differentiable risk functional (Fondeville and Davison, 2016). Here the algorithm is limited only by matrix inversion and thus thousands of locations can be used.

simulBrownResnick is an exact algorithm for simulation of Brown-Resnick max-stable processes as described in Dombry et al. (2015).

simulPareto allows for simulation of Pareto processes associated to log-Gaussian random functions.

rExtremalStudentParetoProcess allows for simulation of Pareto processes associated to Student random functions, using the accept-reject algorithm of Thibaud and Opitz (2015).

`mvNormQuasiMonteCarlo` and `mvTProbQuasiMonteCarlo` are Cpp functions to evaluate the distribution function of Gaussian and t integrals, using a quasi-Monte Carlo algorithm based on randomly shifted lattice rules.

### Author(s)

Raphael de Fondeville

Maintainer: Raphael de Fondeville <raphael.de-fondeville@epfl.ch>

### References

de Fondeville, R. and Davison A. (2018). High-dimensional peaks-over-threshold inference. *Biometrika*, 105(3), 575-592.

Engelke, S. et al. (2015). Estimation of Huesler-Reiss Distributions and Brown-Resnick Processes. *Journal of the Royal Statistical Society: Series B*, 77(1), 239-265

Wadsworth, J.L. and Tawn, J.A. (2013). Efficient inference for spatial extreme value processes associated to log-Gaussian random functions. *Biometrika*, 101(1), 1-15.

Thibaud, E. and T. Opitz (2015). Efficient inference and simulation for elliptical Pareto processes. *Biometrika*, 102(4), 855-870.

Dombry, C., Engelke, S. and Oesting, M. (2016). Exact simulation of max-stable processes. *Biometrika*, 103(2), 303-317.

Genz, A. and Bretz, F. (2009). *Computations of Multivariate Normal and t Probabilities*, volume 105. Springer: Dordrecht.

### Examples

```
#Define semi-variogram function
vario <- function(h, alpha = 1.5){
  norm(h,type = "2")^alpha
}

#Define locations
loc <- expand.grid(1:4, 1:4)

#Simulate data
obs <- simulPareto(1000, loc, vario)

#Evaluate risk functional
sums <- sapply(obs, sum)

#Define weighting function
weighFun <- function(x, u){
  x * (1 - exp(-(sum(x) / u - 1)))
}

#Define partial derivative of weighting function
dWeighFun <- function(x, u){
  (1 - exp(-(sum(x) / u - 1))) + (x / u) * exp( - (sum(x) / u - 1))
}
```

```

#Select exceedances
threshold <- quantile(sums, 0.9)
exceedances <- obs[sums > threshold]

#Define objective function
objectiveFunction = function(parameter, exceedances, loc, vario, weighthFun, dWeigthFun, threshold){

  #Define semi-variogram for the corresponding parameters
  varioModel <- function(h){
    vario(h, parameter[1])
  }

  #Compute score
  scoreEstimation(exceedances, loc, varioModel, weighthFun, dWeigthFun, u = threshold)
}

#Estimate the parameter by optimization of the objective function
est <- optim(par = c(1.5),
             fn = objectiveFunction,
             exceedances = exceedances,
             loc = loc,
             vario = vario,
             weighthFun = weighthFun,
             dWeigthFun = dWeigthFun,
             threshold = threshold,
             control = list(maxit = 100, trace = 1),
             lower = c(0.01),
             upper = c(1.99),
             method = "L-BFGS-B")

```

---

censoredLikelihoodBR *Censored log-likelihood function for the Brown–Resnick model.*

---

## Description

Compute the peaks-over-threshold censored negative log-likelihood function for the Brown–Resnick model.

## Usage

```

censoredLikelihoodBR(
  obs,
  loc,
  vario,
  u,
  p = 499L,
  vec = NULL,

```

```

    nCores = 1L,
    cl = NULL,
    likelihood = "mgp",
    ntot = NULL,
    ...
)

censoredLikelihood(
  obs,
  loc,
  vario,
  u,
  p = 499L,
  vec = NULL,
  nCores = 1L,
  cl = NULL
)

```

### Arguments

obs	List of vectors for which at least one component exceeds a high threshold.
loc	Matrix of coordinates as given by <code>expand.grid()</code> .
vario	Semi-variogram function taking a vector of coordinates as input.
u	Vector of threshold under which to censor components.
p	Number of samples used for quasi-Monte Carlo estimation. Must be a prime number.
vec	Generating vector for the quasi-Monte Carlo procedure. For a given prime p and dimension, can be computed using <code>genVecQMC</code> .
nCores	Number of cores used for the computation
cl	Cluster instance as created by <code>makeCluster</code> of the <code>parallel</code> package.
likelihood	vector of strings specifying the contribution. Either "mgp" for multivariate generalized Pareto, "poisson" for a Poisson contribution for the observations falling below or "binom" for a binomial contribution.
ntot	integer number of observations below and above the threshold, to be used with Poisson or binomial likelihood
...	Additional arguments passed to Cpp routine.

### Details

The function computes the censored negative log-likelihood function based on the representation developed by Wadsworth et al. (2014) and Engelke et al. (2015). Margins must have been standardized first, for instance to the unit Frechet scale.

### Value

Negative censored log-likelihood for the set of observations `obs` and semi-variogram `vario` with attributes `exponentMeasure` for all of the likelihood type selected, in the order "mgp", "poisson", "binom".

**Author(s)**

Raphael de Fondeville

**References**

Wadsworth, J. L. and J. A. Tawn (2014). Efficient inference for spatial extreme value processes associated to log-Gaussian random functions. *Biometrika*, 101(1), 1-15.

Asadi, P., Davison A. C. and S. Engelke (2015). Extremes on River Networks. *Annals of Applied Statistics*, 9(4), 2023-2050.

**Examples**

```
#Define semi-variogram function
vario <- function(h){
  0.5 * norm(h, type = "2")^1.5
}

#Define locations
loc <- expand.grid(1:4, 1:4)

#Simulate data
obs <- simulPareto(1000, loc, vario)

#Evaluate risk functional
maxima <- sapply(obs, max)
thres <- quantile(maxima, 0.9)

#Select exceedances
exceedances <- obs[maxima > thres]

#Compute generating vector
p <- 499
latticeRule <- genVecQMC(p, (nrow(loc) - 1))
primeP <- latticeRule$primeP
vec <- latticeRule$genVec

#Compute log-likelihood function
censoredLikelihoodBR(obs = exceedances, loc = loc, vario = vario,
  u = thres, p = primeP, vec = vec, ntot = 1000)
```

---

censoredLikelihoodXS    *Censored log-likelihood function of the extremal Student model*

---

**Description**

Compute the peaks-over-threshold censored negative log-likelihood function for the extremal Student model.

**Usage**

```

censoredLikelihoodXS(
  obs,
  loc,
  corrFun,
  nu,
  u,
  p = 499L,
  vec = NULL,
  nCores = 1L,
  cl = NULL,
  likelihood = "mgp",
  ntot = NULL,
  std = FALSE,
  ...
)

```

**Arguments**

<code>obs</code>	List of vectors for which at least one component exceeds a high threshold.
<code>loc</code>	Matrix of coordinates as given by <code>expand.grid()</code> .
<code>corrFun</code>	correlation function taking a vector of coordinates as input.
<code>nu</code>	degrees of freedom of the Student process
<code>u</code>	Vector of thresholds under which to censor components.
<code>p</code>	Number of samples used for quasi-Monte Carlo estimation. Must be a prime number.
<code>vec</code>	Generating vector for the quasi-Monte Carlo procedure. For a given <code>p</code> and dimensionality, can be computed using <code>genVecQMC</code> .
<code>nCores</code>	Number of cores used for the computation
<code>cl</code>	Cluster instance as created by <code>makeCluster</code> of the <code>parallel</code> package.
<code>likelihood</code>	vector of string specifying the contribution. Either <code>"mgp"</code> for multivariate generalized Pareto, <code>"poisson"</code> for a Poisson contribution for the observations falling below or <code>"binom"</code> for a binomial contribution.
<code>ntot</code>	integer number of observations below and above the threshold, to be used with Poisson or binomial likelihood
<code>std</code>	logical; if <code>std = TRUE</code> , consider <code>obs/u</code> for scalar <code>u</code> and exceedances over 1 rather than <code>obs &gt; u</code> for potentially vector <code>u</code> . This affects the value of the log-likelihood function. Default to <code>FALSE</code> .
<code>...</code>	Additional arguments passed to Cpp routine.

**Details**

The function computes the censored log-likelihood function based on the representation developed by Ribatet (2013); see also Thibaud and Opitz (2015). Margins must have been standardized, for instance to unit Fréchet.

**Value**

Negative censored log-likelihood function for the set of observations `obs` and correlation function `corrFun`, with attributes `exponentMeasure` for all of the likelihood type selected, in the order "mgp", "poisson", "binom"..

**Author(s)**

Leo Belzile

**References**

Thibaud, E. and T. Opitz (2015). Efficient inference and simulation for elliptical Pareto processes. *Biometrika*, 102(4), 855-870.

Ribatet, M. (2013). Spatial extremes: max-stable processes at work. *JSFS*, 154(2), 156-177.

**Examples**

```
#Define correlation function
corrFun <- function(h, alpha = 1, lambda = 1){
  exp(-norm(h, type = "2")^alpha/lambda)
}

#Define locations
loc <- expand.grid(1:4, 1:4)

#Compute generating vector
p <- 499L
latticeRule <- genVecQMC(p, (nrow(loc) - 1))
primeP <- latticeRule$primeP
vec <- latticeRule$genVec

#Simulate data
Sigma <- exp(-as.matrix(dist(loc))^0.8)
obs <- rExtremalStudentParetoProcess(n = 1000, nu = 5, Sigma = Sigma)
obs <- split(obs, row(obs))

#Evaluate risk functional
maxima <- sapply(obs, max)
thresh <- quantile(maxima, 0.9)

#Select exceedances
exceedances <- obs[maxima > thresh]

#Compute log-likelihood function
eval <- censoredLikelihoodXS(exceedances, loc, corrFun, nu = 5, u = thresh, primeP, vec)
```



---

`genVecQMC`*Generating vectors for lattice rules*

---

**Description**

Compute an efficient generating vector for quasi-Monte Carlo estimation.

**Usage**

```
genVecQMC(p, d, bt = rep(1, d), gm = c(1, (4/5)^(0:(d - 2))))
```

**Arguments**

<code>p</code>	number of samples to use in the quasi-Monte Carlo procedure.
<code>d</code>	Dimension of the multivariate integral to estimate.
<code>bt</code>	Tuning parameter for finding the vector. See D. Nuyens and R. Cools (2004) for more details.
<code>gm</code>	Tuning parameter for finding the vector. See D. Nuyens and R. Cools (2004) for more details.

**Details**

The function computes a generating vector for efficient multivariate integral estimation based on D. Nuyens and R. Cools (2004). If `p` is not a prime, the nearest smaller prime is used instead.

**Value**

`primeP`, the highest prime number smaller than `p` and `genVec`, a `d`-dimensional generating vector defining an efficient lattice rule for `primeP` samples.

**References**

Nuyens, D. and R. Cools (2004). Fast component-by-component construction, a reprise for different kernels. In Monte Carlo and Quasi-Monte Carlo Methods 2004, H. Niederreiter and D. Talay, eds. Springer: Berlin, 373-87.

**Examples**

```
#Define the number of sample.
p <- 500

#Choose a dimension
d <- 300

#Compute the generating vector
latticeRule <- genVecQMC(p,d)

print(latticeRule$primeP)
```

```
print(latticeRule$genVec)
```

---

```
mvtNormQuasiMonteCarlo
```

*Multivariate normal distribution function*

---

### Description

Estimate the multivariate distribution function with quasi-Monte Carlo method.

### Usage

```
mvtNormQuasiMonteCarlo(p, upperBound, cov, genVec, ...)
```

### Arguments

p	Number of samples used for quasi-Monte Carlo estimation. Must be a prime number.
upperBound	Vector of probabilities, i.e., the upper bound of the integral.
cov	Covariance matrix of the multivariate normal distribution. Must be positive semi-definite. <b>WARNING:</b> for performance in high-dimensions, no check is performed on the matrix. It is the user responsibility to ensure that the matrix is positive semi-definite.
genVec	Generating vector for the quasi-Monte Carlo procedure. Can be computed using <code>genVecQMC</code> .
...	Additional arguments passed to Cpp routine.

### Details

The function uses a quasi-Monte Carlo procedure based on randomly shifted lattice rules to estimate the distribution function a multivariate normal distribution as described in Genz and Bretz (2009) on page 50.

### Value

A named vector with components `estimate` estimate of the distribution function along error, 3 times the empirical Monte Carlo standard error over the `nrep` replications.

### Source

Adapted from the QSILATMVNV by A. Genz (2013)

### References

Genz, A. and Bretz, F. (2009). Computations of Multivariate Normal and t Probabilities, volume 105. Springer: Dordrecht.

**Examples**

```

#Define locations
loc <- expand.grid(1:4, 1:4)
ref <- sample.int(16, 1)

#Compute variogram matrix
variogramMatrix <- ((sqrt((outer(loc[,1],loc[,1],"-")^2 +
(outer(loc[,2],loc[,2],"-")^2)) / 2)^(1.5)

#Define an upper bound
upperBound <- variogramMatrix[-ref,ref]

#Compute covariance matrix
cov <- (variogramMatrix[-ref,ref]%*%t(matrix(1, (nrow(loc) - 1), 1)) +
t(variogramMatrix[ref,-ref]%*%t(matrix(1, (nrow(loc) - 1), 1))) -
variogramMatrix[-ref,-ref])

#Compute generating vector
p <- 499
latticeRule <- genVecQMC(p, (nrow(loc) - 1))

#Estimate the multivariate distribution function
mvtNormQuasiMonteCarlo(latticeRule$primeP, upperBound, cov, latticeRule$genVec)

```

---

mvTProbQuasiMonteCarlo

*Multivariate t distribution function*


---

**Description**

Estimate the multivariate t distribution function with quasi-Monte Carlo method.

**Usage**

```
mvtProbQuasiMonteCarlo(p, upperBound, cov, nu, genVec, ...)
```

**Arguments**

p	Number of samples used for quasi-Monte Carlo estimation. Must be a prime number.
upperBound	Vector of probabilities, i.e., the upper bound of the integral.
cov	Covariance matrix of the multivariate normal distribution. Must be positive semi-definite. <b>WARNING:</b> for performance in high-dimensions, no check is done to ensure positive-definiteness of the covariance matrix. It is the user responsibility to ensure that this property is verified.
nu	Degrees of freedom of the t distribution.
genVec	Generating vector for the quasi-Monte Carlo procedure. Can be computed using genVecQMC.
...	Additional arguments passed to Cpp routine.

**Details**

The function uses a quasi-Monte Carlo procedure based on randomly shifted lattice rules to estimate the distribution function a multivariate normal distribution as described in Genz and Bretz (2009) on page 50.

For compatibility reasons, the function handles the univariate case, which is passed on to pt.

**Value**

A named vector with components estimate estimate of the distribution function along error, 3 times the empirical Monte Carlo standard error over the nrep replications.

**Author(s)**

Raphael de Fondeville

**Source**

Adapted from the QSILATMTV Matlab routine by A. Genz (2013)

**References**

Genz, A. and Bretz, F. (2009). Computations of Multivariate Normal and t Probabilities, volume 105. Springer: Dordrecht.

**Examples**

```
#Define locations
loc <- expand.grid(1:4, 1:4)
ref <- sample.int(16, 1)

#Define degrees of freedom
nu <- 3

#Compute variogram matrix
variogramMatrix <- ((sqrt((outer(loc[,1],loc[,1],"-")^2 +
(outer(loc[,2],loc[,2],"-")^2)) / 2)^(1.5))

#Define an upper bound
upperBound <- variogramMatrix[-ref,ref]

#Compute covariance matrix
cov <- (variogramMatrix[-ref,ref]%*%t(matrix(1, (nrow(loc) - 1), 1)) +
t(variogramMatrix[ref,-ref]%*%t(matrix(1, (nrow(loc) - 1), 1))) -
variogramMatrix[-ref,-ref])

#Compute generating vector
p <- 499
latticeRule <- genVecQMC(p, (nrow(loc) - 1))

#Estimate the multivariate distribution function
mvTProbQuasiMonteCarlo(latticeRule$primeP, upperBound, cov, nu, latticeRule$genVec)
```

---

rExtremalStudentParetoProcess

*Simulation of extremal Student generalized Pareto vectors*


---

## Description

Simulation of Pareto processes associated to the max functional. The algorithm is described in section 4 of Thibaud and Opitz (2015). The Cholesky decomposition of the matrix Sigma leads to samples on the unit sphere with respect to the Mahalanobis distance. An accept-reject algorithm is then used to simulate samples from the Pareto process. If `normalize = TRUE`, the vector is scaled by the exponent measure  $\kappa$  so that the maximum of the sample is greater than  $\kappa$ .

## Usage

```
rExtremalStudentParetoProcess(
  n,
  Sigma,
  nu,
  normalize = FALSE,
  matchol = NULL,
  trunc = TRUE
)
```

## Arguments

n	sample size
Sigma	a d by d correlation matrix
nu	degrees of freedom parameter
normalize	logical; should unit Pareto samples above $\kappa$ be returned?
matchol	Cholesky matrix $\mathbf{A}$ such that $\mathbf{A}\mathbf{A}^\top = \mathbf{\Sigma}$ . Corresponds to <code>t(chol(Sigma))</code> . Default to NULL, in which case the Cholesky root is computed within the function.
trunc	logical; should negative components be truncated at zero? Default to TRUE.

## Value

an n by d matrix of samples, with attributes "accept.rate" indicating the fraction of samples accepted.

## Note

If  $\nu > 2$ , an accept-reject algorithm using simulations from the angular measure on the  $l_1$  is at least twice as efficient. The relative efficiency of the latter is much larger for larger  $\nu$ . This algorithm should therefore not be used in high dimensions as its acceptance rate is several orders of magnitude smaller than that implemented in [rparp](#).

**Author(s)**

Emeric Thibaud, Leo Belzile

**References**

Thibaud, E. and T. Opitz (2015). Efficient inference and simulation for elliptical Pareto processes. *Biometrika*, 102(4), 855-870.

**See Also**

[rparp](#)

**Examples**

```
loc <- expand.grid(1:4, 1:4)
Sigma <- exp(-as.matrix(dist(loc))^1.5)
rExtremalStudentParetoProcess(100, Sigma, nu = 2)
```

---

scoreEstimation

*Gradient score function for the Brown–Resnick model.*

---

**Description**

Compute the peaks-over-threshold gradient score function for the Brown–Resnick model.

**Usage**

```
scoreEstimation(
  obs,
  loc,
  vario,
  weightFun = NULL,
  dWeightFun = NULL,
  nCores = 1L,
  cl = NULL,
  ...
)
```

**Arguments**

obs	List of vectors exceeding an R-threshold, see de Fondeville and Davison (2018) for more details.
loc	Matrix of coordinates as given by <code>expand.grid()</code> .
vario	Semi-variogram function taking a vector of coordinates as input.
weightFun	Function of weights.
dWeightFun	Partial derivative function of <code>weightFun</code> .

nCores	Number of cores used for the computation
cl	Cluster instance as created by makeCluster of the parallel package.
...	Parameters for weightFun and dWeightFun.

### Details

The function computes the gradient score based on the representation developed by Wadsworth et al. (2014). Margins must have been standardized. The weighting function must be differentiable and verify some properties for consistency, see de Fondeville and Davison (2018) for more details.

### Value

Evaluation of the gradient score function for the set of observations `obs` and semi-variogram `vario`.

### Author(s)

Raphael de Fondeville

### References

de Fondeville, R. and Davison A. (2018). High-dimensional peaks-over-threshold inference. *Biometrika*, 105(3), 575-592.

Wadsworth, J. L. and J. A. Tawn (2014). Efficient inference for spatial extreme value processes associated to log-Gaussian random functions. *Biometrika*, 101(1), 1-15.

### Examples

```
#Define variogram function
vario <- function(h){
  1 / 2 * norm(h,type = "2")^1.5
}

#Define locations
loc <- expand.grid(1:4, 1:4)

#Simulate data
obs <- simulPareto(1000, loc, vario)

#Evaluate risk functional
sums <- sapply(obs, sum)

#Define weighting function
weightFun <- function(x, u){
  x * (1 - exp(-(sum(x / u) - 1)))
}

#Define partial derivative of weighting function
dWeightFun <- function(x, u){
  (1 - exp(-(sum(x / u) - 1))) + (x / u) * exp(- (sum(x / u) - 1))
}
```

```
#Select exceedances
threshold <- quantile(sums, 0.9)
exceedances <- obs[sums > threshold]

#Evaluate gradient score function
scoreEstimation(exceedances, loc, vario, weightFun = weightFun, dWeightFun, u = threshold)
```

---

simulBrownResnick      *Simulation of Brown–Resnick max-stable random vectors*

---

### Description

simulBrownResnick provides  $n$  replicates of a Brown–Resnick max-stable process with semi-variogram `vario` at locations `loc`.

### Usage

```
simulBrownResnick(n, loc, vario, nCores = 1, cl = NULL)
```

### Arguments

<code>n</code>	Number of replicates desired.
<code>loc</code>	Matrix of coordinates as given by <code>expand.grid()</code> .
<code>vario</code>	Semi-variogram function.
<code>nCores</code>	Number of cores needed for the computation
<code>cl</code>	Cluster instance as created by <code>makeCluster</code> of the <code>parallel</code> package. Make sure the random number generator has been properly initialized with <code>clusterSetRNGStream()</code> .

### Details

The algorithm used here is based on the spectral representation of the Brown–Resnick model as described in Dombry et al. (2015). It provides  $n$  exact simulations on the unit Fréchet scale and requires, in average, for each max-stable vector, the simulation of  $d$  Pareto processes, where  $d$  is the number of locations.

### Value

List of  $n$  random vectors drawn from a max-stable Brown–Resnick process with semi-variogram `vario` at location `loc`.

### References

Dombry, C., Engelke, S. and M. Oesting. Exact simulation of max-stable processes. *Biometrika*, 103(2), 303-317.



**Examples**

```
#Define semi-variogram function
vario <- function(h){
  1 / 2 * norm(h,type = "2")^1.5
}

#Define locations
loc <- expand.grid(1:4, 1:4)

#Simulate data
obs <- simulBrownResnick(10, loc, vario)
```

---

simulPareto

*Simulate log-Gaussian Pareto random vectors*


---

**Description**

simulPareto provides  $n$  replicates of the multivariate Pareto distribution associated to log-Gaussian random function with semi-variogram `vario`.

**Usage**

```
simulPareto(n, loc, vario, nCores = 1, cl = NULL)
```

**Arguments**

<code>n</code>	Number of replicates desired.
<code>loc</code>	Matrix of coordinates as given by <code>expand.grid()</code> .
<code>vario</code>	Semi-variogram function.
<code>nCores</code>	Number of cores used for the computation
<code>cl</code>	Cluster instance as created by <code>makeCluster</code> of the <code>parallel</code> package. Make sure the random number generator has been properly initialized with <code>clusterSetRNGStream()</code> .

**Details**

The algorithm used here is based on the spectral representation of the Brown–Resnick model as described in Dombry et al. (2015). It provides  $n$  replicates conditioned that  $\text{mean}(x) > 1$  on the unit Fréchet scale.

**Value**

List of  $n$  random vectors drawn from a multivariate Pareto distribution with semi-variogram `vario`.

**References**

Dombry, C., Engelke, S. and M. Oesting. Exact simulation of max-stable processes. *Biometrika*, 103(2), 303-317.

**Examples**

```
#Define variogram function
vario <- function(h){
  1 / 2 * norm(h,type = "2")^1.5
}

#Define locations
loc <- expand.grid(1:4, 1:4)

#Simulate data
obs <- simulPareto(100, loc, vario)
```

---

spectralLikelihood      *Spectral log-likelihood function*

---

**Description**

Compute the negative spectral log-likelihood function for Brown–Resnick model with peaks-over-threshold.

**Usage**

```
spectralLikelihood(obs, loc, vario, nCores = 1L, cl = NULL)
```

**Arguments**

obs	List of observations vectors for which $\text{sum}(x)$ exceeds a high threshold.
loc	Matrix of coordinates as given by <code>expand.grid()</code> .
vario	Semi-variogram function taking a vector of coordinates as input.
nCores	Number of cores used for the computation
cl	Cluster instance as created by <code>makeCluster</code> of the <code>parallel</code> package.

**Details**

The function compute the negative log-likelihood function based on the spectral representation developed by Engelke et al. (2015). This simplified expression is obtained by conditioning on the event ‘ $\text{sum}(x)$  exceeds a high threshold  $u > 1$ ’. Margins must have been standardized.

**Value**

Negative spectral log-likelihood function evaluated at the set of observations `obs` with semi-variogram `vario`.

**References**

Engelke, S. et al. (2015). Estimation of Huesler-Reiss distributions and Brown-Resnick processes. *Journal of the Royal Statistical Society: Series B*, 77(1), 239-265

**Examples**

```
#Define semi-variogram function
vario <- function(h){
  1 / 2 * norm(h,type = "2")^1.5
}

#Define locations
loc <- expand.grid(1:4, 1:4)

#Simulate data
obs <- simulPareto(1000, loc, vario)

#Evaluate risk functional
sums <- sapply(obs, sum)

#Select exceedances
exceedances <- obs[sums > quantile(sums, 0.9)]

#Evaluate the spectral function
spectralLikelihood(exceedances, loc, vario)
```

# Index

censoredLikelihood  
    (censoredLikelihoodBR), 4  
censoredLikelihoodBR, 4  
censoredLikelihoodXS, 6  
  
genVecQMC, 9  
  
mvPot (mvPot-package), 2  
mvPot-package, 2  
mvtNormQuasiMonteCarlo, 10  
mvTProbQuasiMonteCarlo, 11  
  
rExtremalStudentParetoProcess, 13  
rparp, 13, 14  
  
scoreEstimation, 14  
simulBrownResnick, 16  
simulPareto, 17  
spectralLikelihood, 18