Package ‘sns’

July 23, 2025

Type Package

Title Stochastic Newton Sampler (SNS)

Version 1.2.2

Date 2022-11-01

Author Alireza S. Mahani, Asad Hasan, Marshall Jiang, Mansour T.A. Sharabiani

Maintainer Alireza Mahani <alireza.s.mahani@gmail.com>

Description Stochastic Newton Sampler (SNS) is a Metropolis-Hastings-based, Markov Chain Monte Carlo sampler for twice differentiable, log-concave probability density functions (PDFs) where the proposal density function is a multivariate Gaussian resulting from a second-order Taylor-series expansion of log-density around the current point. The mean of the Gaussian proposal is the full Newton-Raphson step from the current point. A Boolean flag allows for switching from SNS to Newton-Raphson optimization (by choosing the mean of proposal function as next point). This can be used during burn-in to get close to the mode of the PDF (which is unique due to concavity). For high-dimensional densities, mixing can be improved via 'state space partitioning' strategy, in which SNS is applied to disjoint subsets of state space, wrapped in a Gibbs cycle. Numerical differentiation is available when analytical expressions for gradient and Hessian are not available. Facilities for validation and numerical differentiation of log-density are provided. Note: Formerly available versions of the MfUSampler can be obtained from the archive <https://cran.r-project.org/src/contrib/Archive/MfUSampler/>.

License GPL (>= 2)

Imports mvtnorm, coda, numDeriv

Suggests RegressionFactory, MfUSampler

NeedsCompilation no

Repository CRAN

Date/Publication 2022-11-02 11:02:22 UTC

Contents

`ess` *Effective Sample Size Calculator*

Description

Computes the effective sample size of MCMC chains, using the algorithm in Section 2.3 of the paper by Madeline Thompson. The algorithm is taken from earlier work on 'Initial Sequence Estimators' by multiple authors.

Usage

```
ess(x, method = c("coda", "ise"))
```

Arguments

`x`	Matrix object with each sample (possibly multivariate) as a row. Effective sample size calculation is done independently for each column of `x`.
`method`	Method of calculating effective size. Current options are "coda" which calls `effectiveSize` function in `coda` package, and "ise" which uses the 'Initial Sequence Estimators' method described in Section 2.3 of Thompson (2010).

Value

Vector with effective sample sizes for the time series in each column of `x`.

Author(s)

Alireza S. Mahani, Asad Hasan, Marshall Jiang, Mansour T.A. Sharabiani

References

Thompson, Madeleine (2010) *A Comparison of Methods for Computing Autocorrelation Time* `https://arxiv.org/pdf/1011.0175v1.pdf`

plot.sns *Plotting "sns" Objects*

Description

Method for visualizing the output of `sns.run`.

Usage

```
## S3 method for class 'sns'
plot(x, nburnin = max(nrow(x)/2, attr(x, "nnr"))
  , select = if (length(x) <= 10) 1:5 else 1, ...)
```

Arguments

`x`	Object of class "sns", typically the output of `sns.run`.
`nburnin`	Number of burn-in iterations to discard before generating effective sample size, histograms, and autocorrelation plots.
`select`	Which plot types must be generated. See below for description.
`...`	Arguments passed to/from other functions.

Value

`plot.sns` produces the following types of plots: 1) log-probability trace plot (vertical line, if present, indicates transition from nr to mcmc mode), 2) trace plot of state variables (one per coordinate; vertical line has same meaning as 1), 3) effective sample size by coordinate (horizontal line indicates maximum effective size possible, equal to number of samples after discarding nburnin initial iterations), 4) post-burnin state vector histograms (one per coordinate, vertical line indicates post-burnin average, 5) autocorrelation plots, one per coordinate.

Author(s)

Alireza S. Mahani, Asad Hasan, Marshall Jiang, Mansour T.A. Sharabiani

References

Mahani A.S., Hasan A., Jiang M. & Sharabiani M.T.A. (2016). Stochastic Newton Sampler: The R Package sns. Journal of Statistical Software, Code Snippets, 74(2), 1-33. doi:10.18637/jss.v074.c02

See Also

`sns.run`

predict.sns *Sample-based prediction using "sns" Objects*

Description

Method for sample-based prediction using the output of `sns.run`.

Usage

```
## S3 method for class 'sns'
predict(object, fpred
  , nburnin = max(nrow(object)/2, attr(object, "nnr"))
  , end = nrow(object), thin = 1, ...)
## S3 method for class 'predict.sns'
summary(object
  , quantiles = c(0.025, 0.5, 0.975)
  , ess.method = c("coda", "ise"), ...)
## S3 method for class 'summary.predict.sns'
print(x, ...)
```

Arguments

`object`	Object of class "sns" (output of `sns.run`) or "predict.sns" (output of `predict.sns`).
`fpred`	Prediction function, accepting a single value for the state vector and producing a vector of outputs.
`nburnin`	Number of burn-in iterations discarded for sample-based prediction.
`end`	Last iteration used in sample-based prediction.
`thin`	One out of `thin` iterations within the specified range are used for sample-based prediction.
`quantiles`	Values for which sample-based quantiles are calculated.
`ess.method`	Method used for calculating effective sample size. Default is to call `effectiveSize` from package `coda`.
`x`	An object of class "summary.predict.sns".
`...`	Arguments passed to/from other functions.

Value

`predict.sns` produces a matrix with number of rows equal to the length of prediction vector produces by `fpred`. Its numnber of columns is equal to the number of samples used within the user-specified range, and after thinning (if any). `summary.predict.sns` produces sample-based prediction mean, standard deviation, quantiles, and effective sample size.

Note

See package vignette for more details on SNS theory, software, examples, and performance.

Author(s)

Alireza S. Mahani, Asad Hasan, Marshall Jiang, Mansour T.A. Sharabiani

References

Mahani A.S., Hasan A., Jiang M. & Sharabiani M.T.A. (2016). Stochastic Newton Sampler: The R Package sns. Journal of Statistical Software, Code Snippets, 74(2), 1-33. doi:10.18637/jss.v074.c02

See Also

`sns.run`

Examples

```
## Not run:

# using RegressionFactory for generating log-likelihood and derivatives
library("RegressionFactory")

loglike.poisson <- function(beta, X, y) {
  regfac.expand.1par(beta, X = X, y = y,
    fbase1 = fbase1.poisson.log)
}

# simulating data
K <- 5
N <- 1000
X <- matrix(runif(N * K, -0.5, +0.5), ncol = K)
beta <- runif(K, -0.5, +0.5)
y <- rpois(N, exp(X %*% beta))

beta.init <- rep(0.0, K)
beta.smp <- sns.run(beta.init, loglike.poisson,
  niter = 1000, nnr = 20, mh.diag = TRUE, X = X, y = y)

# prediction function for mean response
predmean.poisson <- function(beta, Xnew) exp(Xnew %*% beta)
ymean.new <- predict(beta.smp, predmean.poisson,
                     nburnin = 100, Xnew = X)
summary(ymean.new)

# (stochastic) prediction function for response
predsmp.poisson <- function(beta, Xnew)
  rpois(nrow(Xnew), exp(Xnew %*% beta))
ysmp.new <- predict(beta.smp, predsmp.poisson
                    , nburnin = 100, Xnew = X)
summary(ysmp.new)


## End(Not run)
```

sns *Stochastic Newton Sampler (SNS)*

Description

SNS is a Metropolis-Hastings MCMC sampler with a multivariate Gaussian proposal function resulting from a local, second-order Taylor series expansion of log-density. The mean of the Gaussian proposal is identical to the full Newton-Raphson step from the current point. During burn-in, Newton-Raphson optimization can be performed to get close to the mode of the pdf which is unique due to convexity, resulting in faster convergence. For high dimensional densities, state space partitioning can be used to improve mixing. Support for numerical differentiation is provided using **numDeriv** package. `sns` is the low-level function for drawing one sample from the distribution. For drawing multiple samples from a (fixed) distribution, consider using `sns.run`.

Usage

```
sns(x, fghEval, rnd = TRUE, gfit = NULL, mh.diag = FALSE
  , part = NULL, numderiv = 0
  , numderiv.method = c("Richardson", "simple")
  , numderiv.args = list(), ...)
```

Arguments

`x`	Current state vector.
`fghEval`	Log-density to be sampled from. A valid log-density can have one of 3 forms: 1) return log-density, but no gradient or Hessian, 2) return a list of `f` and `g` for log-density and its gradient vector, respectively, 3) return a list of `f`, `g`, and `h` for log-density, gradient vector, and Hessian matrix. Missing derivatives are computed numerically.
`rnd`	Runs 1 iteration of Newton-Raphson optimization method (non-stochastic or 'nr' mode) when `FALSE`. Runs Metropolis-Hastings (stochastic or 'mcmc' mode) for drawing a sample when `TRUE`.
`gfit`	Gaussian fit at point `init`. If `NULL` then `sns` will compute a Gaussian fit at `x`.
`mh.diag`	Boolean flag, indicating whether detailed MH diagnostics such as components of acceptance test must be returned or not.
`part`	List describing partitioning of state space into subsets. Each element of the list must be an integer vector containing a set of indexes (between `1` and `length(x)` or `length(init)`) indicating which subset of all dimensions to jointly sample. These integer vectors must be mutually exclusive and collectively exhaustive, i.e. cover the entire state space and have no duplicates, in order for the partitioning to represent a valid Gibbs sampling approach. See `sns.make.part` and `sns.check.part`.
`numderiv`	Integer with value from the set `0,1,2`. If `0`, no numerical differentiation is performed, and thus `fghEval` is expected to supply `f`, `g` and `h`. If `1`, we expect `fghEval` to provide `f` amd `g`, and Hessian will be calculated numerically. If

	2, `fghEval` only returns log-density, and numerical differentiation is needed to calculate gradient and Hessian.
`numderiv.method`	Method used for numeric differentiation. This is passed to the `grad` and `hessian` functions in **numDeriv** package. See the package documentation for details.
`numderiv.args`	Arguments to the numeric differentiation method chosen in `numderiv.method`, passed to `grad` and `hessian` functions in **numDeriv**. See package documentation for details.
`...`	Other arguments to be passed to `fghEval`.

Value

`sns` returns the sample drawn as a vector, with attributes:

`accept`	A boolean indicating whether the proposed point was accepted.
`ll`	Value of the log-density at the sampled point.
`gfit`	List containing Gaussian fit to pdf at the sampled point.

Note

1. Since SNS makes local Gaussian approximations to the density with the covariance matrix of the Gaussian proposal being the log-density Hessian, there is a strict requirement for the log-density to be concave.

2. Proving log-concavity for arbitrary probability distributions is non-trvial. However, distributions *generated* by replacing parameters of a concave distribution with linear expressions are known to be log-concave. This negative-definiteness invariance as well as expressions for full gradient and Hessian in terms of derivatives of low-dimensional base distributions are discussed in the vignette. The GLM expansion framework is available in the R package **RegressionFactory**.

3. See package vignette for more details on SNS theory, software, examples, and performance.

Author(s)

Alireza S. Mahani, Asad Hasan, Marshall Jiang, Mansour T.A. Sharabiani

References

Mahani A.S., Hasan A., Jiang M. & Sharabiani M.T.A. (2016). Stochastic Newton Sampler: The R Package sns. Journal of Statistical Software, Code Snippets, 74(2), 1-33. doi:10.18637/jss.v074.c02

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. Biometrika, 57(1), 97-109.

Qi, Y., & Minka, T. P. (2002). Hessian-based markov chain monte-carlo algorithms. 1st Cape Cod Workshop on Monte Carlo Methods.

See Also

`sns.run`, `sns.fghEval.numaug`

Examples

```
## Not run:

# using RegressionFactory for generating log-likelihood and its derivatives
library(RegressionFactory)

loglike.poisson <- function(beta, X, y) {
  regfac.expand.1par(beta, X = X, y = y,
                     fbase1 = fbase1.poisson.log)
}

# simulating data
K <- 5
N <- 1000
X <- matrix(runif(N * K, -0.5, +0.5), ncol = K)
beta <- runif(K, -0.5, +0.5)
y <- rpois(N, exp(X %*% beta))

beta.init <- rep(0.0, K)

# glm estimate, for reference
beta.glm <- glm(y ~ X - 1, family = "poisson",
                start = beta.init)$coefficients

# running SNS in non-stochastic mode
# this should produce results very close to glm
beta.sns <- beta.init
for (i in 1:20)
  beta.sns <- sns(beta.sns, loglike.poisson, X = X, y = y, rnd = F)

# comparison
all.equal(as.numeric(beta.glm), as.numeric(beta.sns))

# trying numerical differentiation
loglike.poisson.fonly <- function(beta, X, y) {
  regfac.expand.1par(beta, X = X, y = y, fgh = 0,
                     fbase1 = fbase1.poisson.log)
}

beta.sns.numderiv <- beta.init
for (i in 1:20)
  beta.sns.numderiv <- sns(beta.sns.numderiv, loglike.poisson.fonly
                  , X = X, y = y, rnd = F, numderiv = 2)
all.equal(as.numeric(beta.glm), as.numeric(beta.sns.numderiv))

# add numerical derivatives to fghEval outside sns
loglike.poisson.numaug <- sns.fghEval.numaug(loglike.poisson.fonly
  , numderiv = 2)

beta.sns.numaug <- beta.init
for (i in 1:20)
  # set numderiv to 0 to avoid repeating
  # numerical augmentation inside sns
  beta.sns.numaug <- sns(beta.sns.numaug, loglike.poisson.numaug
                         , X = X, y = y, rnd = F, numderiv = 0)
all.equal(as.numeric(beta.glm), as.numeric(beta.sns.numaug))


## End(Not run)
```

`sns.check.logdensity` *Utility function for validating log-density*

Description

Utility function for validating log-density: 1) dimensional consistency of function argument, gradient and Hessian, 2) finiteness of function, gradient and Hessian, 3) closeness of analytical and numerical derivatives, and 4) negative definiteness of Hessian.

Usage

```
sns.check.logdensity(x, fghEval
  , numderiv.method = c("Richardson", "complex")
  , numderiv.args = list()
  , blocks = append(list(1:length(x)), as.list(1:length(x)))
  , dx = rep(1, length(x)), nevals = 100, negdef.tol = 1e-08, ...)
## S3 method for class 'sns.check.logdensity'
print(x, ...)
```

Arguments

`x`	For `sns.check.logdensity`, initial point, around which a random collection of points are generated to perform validation tests. For `print.sns.check.logdensity`, an object of class `sns.check.logdensity`, typically the output of `sns.check.logdensity` function.
`fghEval`	Log-density to be validated. A valid log-density can have one of 3 forms: 1) return log-density, but no gradient or Hessian, 2) return a list of `f` and `g` for log-density and its gradient vector, respectively, 3) return a list of `f`, `g`, and `h` for log-density, gradient vector, and Hessian matrix.
`numderiv.method`	Method used for numeric differentiation. This is passed to the `grad` and `hessian` functions in **numDeriv** package. See the package documentation for details.
`numderiv.args`	Arguments to the numeric differentiation method chosen in `numderiv.method`, passed to `grad` and `hessian` functions in **numDeriv**. See package documentation for details.
`blocks`	A list of state space subsets (identified by their positional indexes), for which negative-definiteness of Hessian blocks are to be tested. The default is to test for 1) entire state space, and 2) each dimension individually.

`dx`	A vector of same length as `x`. For `i`'th dimension, `nevals` values are sampled from a uniform distribution with min/max values equal to `x[i]-0.5*dx[i]` and `x[i]+0.5*dx[i]`, respectively. Vectors smaller than `length(x)` are extended as needed by recycling the provided values for `dx`.
`nevals`	Number of points in state space, for which validation tests will be performed.
`negdef.tol`	Lower bound for absolute value of (negative) eigenvalues of Hessian, evaluated at each of the `nevals` points in the state space. If one or more eigenvalues have absolute values smaller than `ngdef.tol`, log-density is declared non-log-concave at that point.
`...`	Other arguments to be passed to `fghEval`.

Value

`sns.check.logdensity` returns a list of class `sns.check.logdensity`, with the following elements:

`check.ld.struct`	Boolean flag, indicating whether log-density `fghEval` has one of the 3 forms of output, described above.
`numderiv`	Integer with values of `0,1,2`. A value of `0` means analytical gradient and Hessian have been provided, and thus there is no need for numerical differentiation. `1` means analytical gradient is provided, but Hessian must be calculated numerically. `2` means both gradient and Hessian must be numerically calculated. Users can pass this value to subsequent `sns` or `sns.run` calls.
`check.length.g`	Boolean flag, indicating whether length of gradient vector (element `g`) returned by `fghEval` equals `length(x)`.
`check.dim.h`	Boolean flag, indicating whether number of rows and columns of the Hessian matrix (element `h`) returned by `fghEval` equal `length(x)`.
`x.mat`	Collection of state space vectors (one per row), for which validation tests are performed. It has `nevals` rows and `length(x)` columns.
`t.evals`	Time spent on evaluating `fghEval` on `nevals` points chosen randomly in the neighborhood of `x`, as specified by `dx`. This includes log-density and, if provided, analytical evaluations of gradient and Hessian.
`t.num.evals`	Time spent on evaluating the numeric version of `fghEval`, in which gradient and Hessian are computed numerically, using `grad` and `hessian` functions in the **numDeriv** package. Comparison of this number with `t.evals` provides the user with insight into the relative speed of numerical differentiation compared to analytical versions.
`f.vec`	Vector of log-density values for state space vectors listed in `x.mat`.
`g.mat.num`	Collection of numerically-computed gradient vectors for state space values listed in `x.mat`, with the same dimension conventions.
`is.g.num.finite`	Boolean flag, indicating whether all numerically-computed gradient vectors have finite values.

`h.array.num`	Collection of numerically-computed Hessian matrices at points listed in `x.mat`. First dimension is of length `nevals`, and the remaining two dimensions equal `length(x)`.
`is.h.num.finite`	Boolean flag, indicating whether all numerically-computed Hessian matrices have finite values.
`g.mat`	Collection of analytically-computed gradient vectors for state space values listed in `x.mat`, with the same dimension conventions. This is only available if `fghEval` has a `g` field; otherwise `NA`.
`is.g.finite`	Boolean flag (if available), indicating whether all analytically-computed gradient vectors have finite values (if available).
`g.diff.max`	If available, maximum relative difference between analytical and numerical gradient vectors, over all `nevals` points in `x.mat`. Relative diference is defined as L2 norm of difference between the two gradient vectors, divided by the L2 norm of the analytical gradient vector.
`h.array`	If available, collection of analytically-computed Hessian matrices at points listed in `x.mat`. Dimensional conventions are the same as `h.array.num`.
`is.h.finite`	Boolean flag (if available), indicating whether all analytically-computed Hessian matrices have finite values.
`h.diff.max`	If available, maximum relative difference between analytical and numerical Hessian matrices, over all `nevals` points in `x.mat`. Relative difference is defined as the Frobenius norm of difference of analytical and numerical Hessian matrices, divided by the Frobenius norm of analytical Hessian.
`is.negdef.num`	Boolean flag, indicating whether numerical Hessian is negative-definite at all state space points indicated in `x.mat`.
`is.negdef`	Boolean flag, indicating whether analytical Hessian is negative-definite at all state space points indicated in `x.mat`.

Note

1. Validation tests performed in `sns.check.logdensity` cannot prove that a log-density is twice-differentiable, or globally concave. However, when e.g. log-density Hessian is seen to be non-negative-definite at one of the points tested, we can definitively say that the Hessian is not globally negative-definite, and therefore `sns` should not be used for sampling from this distribution. Users must generally consider this function as a supplement to analytical work.

2. See package vignette for more details on SNS theory, software, examples, and performance.

Author(s)

Alireza S. Mahani, Asad Hasan, Marshall Jiang, Mansour T.A. Sharabiani

References

Mahani A.S., Hasan A., Jiang M. & Sharabiani M.T.A. (2016). Stochastic Newton Sampler: The R Package sns. Journal of Statistical Software, Code Snippets, 74(2), 1-33. doi:10.18637/jss.v074.c02

Examples

```
## Not run:

# using RegressionFactory for generating log-likelihood and its derivatives
library(RegressionFactory)

loglike.poisson <- function(beta, X, y) {
  regfac.expand.1par(beta, X = X, y = y,
                     fbase1 = fbase1.poisson.log)
}

# simulating data
K <- 5
N <- 1000
X <- matrix(runif(N * K, -0.5, +0.5), ncol = K)
beta <- runif(K, -0.5, +0.5)
y <- rpois(N, exp(X %*% beta))

beta.init <- rep(0.0, K)

my.check <- sns.check.logdensity(beta.init, loglike.poisson
  , X = X, y = y, blocks = list(1:K))
my.check

# mistake in log-likelihood gradient
loglike.poisson.wrong <- function(beta, X, y) {
  ret <- regfac.expand.1par(beta, X = X, y = y,
                            fbase1 = fbase1.poisson.log)
  ret$g <- 1.2 * ret$g
  return (ret)
}
# maximum relative diff in gradient is now much larger
my.check.wrong <- sns.check.logdensity(beta.init
  , loglike.poisson.wrong, X = X, y = y, blocks = list(1:K))
my.check.wrong

# mistake in log-likelihood Hessian
loglike.poisson.wrong.2 <- function(beta, X, y) {
  ret <- regfac.expand.1par(beta, X = X, y = y,
                            fbase1 = fbase1.poisson.log)
  ret$h <- 1.2 * ret$h
  return (ret)
}
# maximum relative diff in Hessian is now much larger
my.check.wrong.2 <- sns.check.logdensity(beta.init
  , loglike.poisson.wrong.2, X = X, y = y, blocks = list(1:K))
my.check.wrong.2


## End(Not run)
```

sns.fghEval.numaug — *Utility function for augmentation of a log-density function with numerical gradient and Hessian as needed*

Description

Augmenting a log-density with numerical gradient and Hessian, so it can be used by `sns` or `sns.run`. This augmentation will also be done inside the function, if the value of `numderiv` parameter passed to `sns` and `sns.run` is `1` or `2`. The advantage of using `sns.fghEval.numaug` outside these functions is efficiency, since the agumentation code will not have to be executed in every function call. Users must set `numderiv` to `0` when calling `sns` or `sns.run` if calling `sns.fghEval.numaug` first. See example.

Usage

```
sns.fghEval.numaug(fghEval, numderiv = 0
  , numderiv.method = c("Richardson", "simple")
  , numderiv.args = list())
```

Arguments

`fghEval`	Log-density to be sampled from. A valid log-density can have one of 3 forms: 1) return log-density, but no gradient or Hessian, 2) return a list of `f` and `g` for log-density and its gradient vector, respectively, 3) return a list of `f`, `g`, and `h` for log-density, gradient vector, and Hessian matrix. Missing derivatives are computed numerically.
`numderiv`	This must be matched with `fghEval`: Integer with value from the set `0,1,2`. If `0`, no numerical differentiation is performed, and thus `fghEval` is expected to supply `f`, `g` and `h`. If `1`, we expect `fghEval` to provide `f` amd `g`, and Hessian will be calculated numerically. If `2`, `fghEval` only returns log-density, and numerical differentiation is needed to calculate gradient and Hessian.
`numderiv.method`	Method used for numeric differentiation. This is passed to the `grad` and `hessian` functions in **numDeriv** package. See the package documentation for details.
`numderiv.args`	Arguments to the numeric differentiation method chosen in `numderiv.method`, passed to `grad` and `hessian` functions in **numDeriv**. See package documentation for details.

Value

A function, accepting same arguments as `fghEval`, but guaranteed to return the original log-density, plus gradient and Hessian (both of which could possibly by numerically calculated). If `numderiv=0`, `fghEval` is returned without change. The function will return log-density, gradient and Hessian as elements `f`, `g` and `h` of a list.

Note

See package vignette for more details on SNS theory, software, examples, and performance.

Author(s)

Alireza S. Mahani, Asad Hasan, Marshall Jiang, Mansour T.A. Sharabiani

References

Mahani A.S., Hasan A., Jiang M. & Sharabiani M.T.A. (2016). Stochastic Newton Sampler: The R Package sns. Journal of Statistical Software, Code Snippets, 74(2), 1-33. doi:10.18637/jss.v074.c02

See Also

`sns`, `sns.run`

`sns.make.part` *Utility Functions for Creating and Validating State Space Partitions*

Description

Utility functions for creating and validating state space partitions, to be used in SNS for improving the mixing of sampled chains for high-dimensional posteriors.

Usage

```
sns.make.part(K, nsubset, method = "naive")
sns.check.part(part, K)
```

Arguments

`K`	Dimensionality of state space.
`nsubset`	Number of subsets to partition the state space dimensions into.
`method`	Method used for state space partitioning. Currently, only `naive` method is implemented, where coordinates are distributed evenly (or as evenly as possible) across subsets.
`part`	A list of length `nsubset`, with each element a vector of integer values, representing the coordinates belonging to a subset. This list is the output of `sns.make.part`.

Value

`sns.make.part` produces a list of integer vectors, each containing coordinates belonging to the same subset. `sns.check.part` produces a boolean flag, indicating whether or not the partition list is valid or not. The subset members must constitute a mutually-exclusive, collectively-exhaustive set relative to `1:K`.

Author(s)

Alireza S. Mahani, Asad Hasan, Marshall Jiang, Mansour T.A. Sharabiani

References

Mahani A.S., Hasan A., Jiang M. & Sharabiani M.T.A. (2016). Stochastic Newton Sampler: The R Package sns. Journal of Statistical Software, Code Snippets, 74(2), 1-33. doi:10.18637/jss.v074.c02

See Also

`sns`, `sns.run`

Examples

```
# creating a valid partition of a 6-dimensional state space
my.part.valid <- list(c(1,2,3), c(4,5,6))
is.valid.1 <- sns.check.part(my.part.valid, 6)
cat("is partition valid: ", is.valid.1, "\n")

# creating an invalid partition of a 6-dimensional state space
# (coordinate 4 is missing)
my.part.invalid <- list(c(1,2,3), c(5,6))
is.valid.2 <- sns.check.part(my.part.invalid, 6)
cat("is partition valid: ", is.valid.2, "\n")
```

`sns.run` *Drawing multiple samples using Stochastic Newton Sampler*

Description

This is a wrapper around `sns`, allowing one to draw multiple samples from a distribution while collecting diagnostic information.

Usage

```
sns.run(init, fghEval, niter = 100, nnr = min(10, round(niter/4))
  , mh.diag = FALSE, part = NULL, print.level = 0
  , report.progress = ceiling(niter/10)
  , numderiv = 0, numderiv.method = c("Richardson", "simple")
  , numderiv.args = list()
  , ...)
```

Arguments

`init`	Initial value for the MCMC chain.
`fghEval`	Log-density to be sampled from. A valid log-density can have one of 3 forms: 1) return log-density, but no gradient or Hessian, 2) return a list of `f` and `g` for log-density and its gradient vector, respectively, 3) return a list of `f`, `g`, and `h` for log-density, gradient vector, and Hessian matrix. Missing derivatives are computed numerically.
`niter`	Number of iterations to perform (in 'nr' and 'mcmc' mode combined).
`nnr`	Number of initial iterations to spend in 'nr' mode.
`mh.diag`	Boolean flag, indicating whether detailed MH diagnostics such as components of acceptance test must be returned or not.
`part`	List describing partitioning of state space into subsets. Each element of the list must be an integer vector containing a set of indexes (between `1` and `length(x)` or `length(init)`) indicating which subset of all dimensions to jointly sample. These integer vectors must be mutually exclusive and collectively exhaustive, i.e. cover the entire state space and have no duplicates, in order for the partitioning to represent a valid Gibbs sampling approach. See `sns.make.part` and `sns.check.part`.
`print.level`	If greater than 0, print sampling progress report.
`report.progress`	Number of sampling iterations to wait before printing progress reports.
`numderiv`	Integer with value from the set `0,1,2`. If `0`, no numerical differentiation is performed, and thus `fghEval` is expected to supply `f`, `g` and `h`. If `1`, we expect `fghEval` to provide `f` amd `g`, and Hessian will be calculated numerically. If `2`, `fghEval` only returns log-density, and numerical differentiation is needed to calculate gradient and Hessian.
`numderiv.method`	Method used for numeric differentiation. This is passed to the `grad` and `hessian` functions in **numDeriv** package. See the package documentation for details.
`numderiv.args`	Arguments to the numeric differentiation method chosen in `numderiv.method`, passed to `grad` and `hessian` functions in **numDeriv**. See package documentation for details.
`...`	Other parameters to be passed to `fghEval`.

Value

`sns.run` returns an object of class `sns` with elements:

`samplesMat`	A matrix object with `nsample` rows and `K` cols.
`acceptance`	Metropolis proposal percentage acceptance.
`burn.iters`	Number of burn-in ierations.
`sample.time`	Time in seconds spent in sampling.
`burnin.time`	Time in seconds spent in burn-in.

Note

1. `sns.run` cannot be used if SNS is being run as part of a Gibbs cycle, such that the conditional distribution being sampled by SNS changes from one iteration to next. In such cases, `sns` must be used instead, inside an explicit Gibbs-cycle `for` loop.

2. See package vignette for more details on SNS theory, software, examples, and performance.

Author(s)

Alireza S. Mahani, Asad Hasan, Marshall Jiang, Mansour T.A. Sharabiani

References

Mahani A.S., Hasan A., Jiang M. & Sharabiani M.T.A. (2016). Stochastic Newton Sampler: The R Package sns. Journal of Statistical Software, Code Snippets, 74(2), 1-33. doi:10.18637/jss.v074.c02

See Also

`sns`, `summary.sns`, `plot.sns`, `predict.sns`

Examples

```
## Not run:

# using RegressionFactory for generating log-likelihood and its derivatives
library(RegressionFactory)

loglike.poisson <- function(beta, X, y) {
  regfac.expand.1par(beta, X = X, y = y,
    fbase1 = fbase1.poisson.log)
}

# simulating data
K <- 5
N <- 1000
X <- matrix(runif(N * K, -0.5, +0.5), ncol = K)
beta <- runif(K, -0.5, +0.5)
y <- rpois(N, exp(X

beta.init <- rep(0.0, K)

# glm estimate (ML), for reference
beta.glm <- glm(y ~ X - 1, family = "poisson",
                start = beta.init)$coefficients

# sampling of likelihood
beta.smp <- sns.run(init = beta.init
  , fghEval = loglike.poisson, niter = 1000
  , nnr = 20, X = X, y = y)
smp.summ <- summary(beta.smp)

# compare mean of samples against ML estimate (from glm)
```

```
cbind(beta.glm, smp.summ$smp$mean)

# trying numerical differentiation
loglike.poisson.fonly <- function(beta, X, y) {
  regfac.expand.1par(beta, X = X, y = y, fgh = 0,
                     fbase1 = fbase1.poisson.log)
}
beta.smp <- sns.run(init = beta.init
  , fghEval = loglike.poisson.fonly, niter = 1000, nnr = 20
  , X = X, y = y, numderiv = 2)
smp.summ <- summary(beta.smp)
cbind(beta.glm, smp.summ$smp$mean)


## End(Not run)
```

`summary.sns` *Summarizing "sns" Objects*

Description

Methods for summarizing the output of `sns.run`, and for printing the summary.

Usage

```
## S3 method for class 'sns'
summary(object, quantiles = c(0.025, 0.5, 0.975)
  , pval.ref = 0.0, nburnin = max(nrow(object)/2, attr(object, "nnr"))
  , end = nrow(object), thin = 1, ess.method = c("coda", "ise"), ...)
## S3 method for class 'summary.sns'
print(x, ...)
```

Arguments

`object`	An object of class "sns", typically the output of `sns.run`.
`quantiles`	Values for which sample-based quantiles are calculated.
`pval.ref`	Reference value for state space variables, used for calculating sample-based p-values.
`nburnin`	Number of initial iterations to discard before calculating the sample statistics. A warning is issued if this number is smaller than the initial iterations run in NR mode.
`end`	Last iteration to use for calculating sample statistics. Defaults to last iteration.
`thin`	One out of `thin` samples are kept for calculating sample statistics. Default is 1, using all samples within specified range.
`ess.method`	Method used for calculating effective sample size. Default is to call `effectiveSize` from package coda.
`x`	An object of class "summary.sns", typically the output of `summary.sns`.
`...`	Arguments passed to/from other functions.

Value

`summary.sns` returns a list with these elements:

`K`	Dimensionality of state space.
`nnr`	Number of NR (Newton-Raphson) iterations performed at the beginning.
`nburnin`	Number of burn-in iterations. These are discarded before calculating sample statistics.
`end`	Last iteration to use for calculating sample statistics.
`thin`	One out of every `thin` iterations within the specified range is used for calculating sample statistics.
`niter`	Total iterations, including NR and MCMC modes.
`nsmp`	Number of samples within specified range (before applying thinning).
`nseq`	Number of samples used for calculating sample statistics (after applying thinning).
`npart`	Number of subsets used in state space partitioning. If no partitioning is done, the value is `1`.
`accept.rate`	Acceptance rate for the MH transition proposals, calculated over `nsmp` iterations.
`reldev.mean`	Mean relative deviation from quadratic approximation, defined as difference between actual log-density change and the value predicted from quadratic fit at density maximum, divided by the actual change. The location of density maximum is assumed to be the value at the end of the last NR iteration. Therefore, for this measure to be accurate, users must ensure `nnr` is sufficiently large to allow for convegrence of the optimization phase.
`pval.ref`	Same as input.
`ess.method`	Same as input.
`smp`	A list with elements `mean`, `sd`, `ess`, `quantiles`, `pval` representing sample-based mean, standard deviation, effective size, quantiles and sample-based p-values, based on specified range and using thinning (if specified).

Author(s)

Alireza S. Mahani, Asad Hasan, Marshall Jiang, Mansour T.A. Sharabiani

References

Mahani A.S., Hasan A., Jiang M. & Sharabiani M.T.A. (2016). Stochastic Newton Sampler: The R Package sns. Journal of Statistical Software, Code Snippets, 74(2), 1-33. doi:10.18637/jss.v074.c02

See Also

`sns.run`

Index