Package: ZVCV (via r-universe)

September 8, 2024

Type Package

Title Zero-Variance Control Variates

Version 2.1.2

Date 2022-11-02

Description Stein control variates can be used to improve Monte Carlo estimates of expectations when the derivatives of the log target are available. This package implements a variety of such methods, including zero-variance control variates (ZV-CV, Mira et al. (2013) <doi:10.1007/s11222-012-9344-6>), regularised ZV-CV (South et al., 2018 <arXiv:1811.05073>), control functionals (CF, Oates et al. (2017) <doi:10.1111/rssb.12185>) and semi-exact control functionals (SECF, South et al., 2020 <arXiv:2002.00033>). ZV-CV is a parametric approach that is exact for (low order) polynomial integrands with Gaussian targets. CF is a non-parametric alternative that offers better than the standard Monte Carlo convergence rates. SECF has both a parametric and a non-parametric component and it offers the advantages of both for an additional computational cost. Functions for applying ZV-CV and CF to two estimators for the normalising constant of the posterior distribution in Bayesian statistics are also supplied in this package. The basic requirements for using the package are a set of samples, derivatives and function evaluations.

BugReports https://github.com/LeahPrice/ZVCV/issues

License GPL (>= 2)

LazyLoad yes

Imports Rcpp (>= 0.11.0), glmnet, abind, mvtnorm, stats, Rlinsolve, magrittr, dplyr

Suggests partitions, ggplot2, ggthemes

LinkingTo Rcpp, RcppArmadillo, BH

LazyData true

Encoding UTF-8

RoxygenNote 7.1.1

Repository https://leahprice.r-universe.dev

RemoteUrl https://github.com/leahprice/zvcv

RemoteRef HEAD

RemoteSha fac5e870d889bbcd35ee8a6e82578c009d9355fe

Contents

aSECF 2
aSECF_crossval 6
CF 9
CF_crossval 12
evidence 15
Expand_Temperatures 20
getX 21
K0_fn 22
logsumexp 24
medianTune 24
nearPD 25
Phi_fn 26
SECF 27
SECF_crossval 30
squareNorm 33
VDP 34
zvcv 36
ZVCV_package 40

Index **45**

`aSECF` *Approximate semi-exact control functionals (aSECF)*

Description

This function performs approximate semi-exact control functionals as described in South et al (2020). It uses a nystrom approximation and conjugate gradient to speed up SECF. This is faster than `SECF` for large N. If you would like to choose between different kernels using cross-validation, then you can use `aSECF_crossval`.

Usage

```
aSECF(
  integrands,
  samples,
  derivatives,
  polyorder = NULL,
```

```
  steinOrder = NULL,
  kernel_function = NULL,
  sigma = NULL,
  K0 = NULL,
  nystrom_inds = NULL,
  est_inds = NULL,
  apriori = NULL,
  conjugate_gradient = TRUE,
  reltol = 0.01,
  diagnostics = FALSE
)
```

Arguments

integrands	An N by k matrix of integrands (evaluations of the function of interest)
samples	An N by d matrix of samples from the target
derivatives	An N by d matrix of derivatives of the log target with respect to the parameters
polyorder	(optional) The order of the polynomial to be used in the parametric component, with a default of 1. We recommend keeping this value low (e.g. only 1-2).
steinOrder	(optional) This is the order of the Stein operator. The default is 1 in the control functionals paper (Oates et al, 2017) and 2 in the semi-exact control functionals paper (South et al, 2020). The following values are currently available: 1 for all kernels and 2 for "gaussian", "matern" and "RQ". See below for further details.
kernel_function	(optional) Choose between "gaussian", "matern", "RQ", "product" or "prodsim". See below for further details.
sigma	(optional) The tuning parameters of the specified kernel. This involves a single length-scale parameter in "gaussian" and "RQ", a length-scale and a smoothness parameter in "matern" and two parameters in "product" and "prodsim". See below for further details.
K0	(optional) The kernel matrix. One can specify either this or all of sigma, steinOrder and kernel_function. The former involves pre-computing the kernel matrix using K0_fn and is more efficient when using multiple estimators out of CF, SECF and aSECF or when using the cross-validation functions.
nystrom_inds	(optional) The sample indices to be used in the Nystrom approximation.
est_inds	(optional) A vector of indices for the estimation-only samples. The default when est_inds is missing or NULL is to perform both estimation of the control variates and evaluation of the integral using all samples. Otherwise, the samples from est_inds are used in estimating the control variates and the remainder are used in evaluating the integral. Splitting the indices in this way can be used to reduce bias from adaption and to make computation feasible for very large sample sizes (small est_inds is faster), but in general in will increase the variance of the estimator.
apriori	(optional) A vector containing the subset of parameter indices to use in the polynomial. Typically this argument would only be used if the dimension of the problem is very large or if prior information about parameter dependencies is

known. The default is to use all parameters $1:d$ where d is the dimension of the target.

`conjugate_gradient`
: (optional) A flag for whether to perform conjugate gradient to further speed up the nystrom approximation (the default is true).

`reltol`
: (optional) The relative tolerance for choosing when the stop conjugate gradient iterations (the default is 1e-02). using `squareNorm`, as long as the `nystrom_inds` are `NULL`.

`diagnostics`
: (optional) A flag for whether to return the necessary outputs for plotting or estimating using the fitted model. The default is `false` since this requires some additional computation when `est_inds` is `NULL`.

Value

A list with the following elements:

- `expectation`: The estimate(s) of the (k) expectations(s).
- `cond_no`: (Only if `conjugate_gradient = TRUE`) The condition number of the matrix being solved using conjugate gradient.
- `iter`: (Only if `conjugate_gradient = TRUE`) The number of conjugate gradient iterations
- `f_true`: (Only if `est_inds` is not `NULL`) The integrands for the evaluation set. This should be the same as integrands[setdiff(1:N,est_inds),].
- `f_hat`: (Only if `est_inds` is not `NULL`) The fitted values for the integrands in the evaluation set. This can be used to help assess the performance of the Gaussian process model.
- `a`: (Only if `diagnostics = TRUE`) The value of a as described in South et al (2020), where predictions are of the form $f_hat = K0 * a + Phi * b$ for heldout K0 and Phi matrices and estimators using heldout samples are of the form $mean(f - f_hat) + b[1]$.
- `b`: (Only if `diagnostics = TRUE`) The value of b as described in South et al (2020), where predictions are of the form $f_hat = K0 * a + Phi * b$ for heldout K0 and Phi matrices and estimators using heldout samples are of the form $mean(f - f_hat) + b[1]$.
- `ny_inds`: (Only if `diagnostics = TRUE`) The indices of the samples used in the nystrom approximation (this will match nystrom_inds if this argument was not `NULL`).

On the choice of σ, the kernel and the Stein order

The kernel in Stein-based kernel methods is $L_x L_y k(x, y)$ where L_x is a first or second order Stein operator in x and $k(x, y)$ is some generic kernel to be specified.

The Stein operators for distribution $p(x)$ are defined as:

- `steinOrder=1`: $L_x g(x) = \nabla_x^T g(x) + \nabla_x \log p(x)^T g(x)$ (see e.g. Oates el al (2017))
- `steinOrder=2`: $L_x g(x) = \Delta_x g(x) + \nabla_x logp(x)^T \nabla_x g(x)$ (see e.g. South el al (2020))

Here ∇_x is the first order derivative wrt x and $\Delta_x = \nabla_x^T \nabla_x$ is the Laplacian operator.

The generic kernels which are implemented in this package are listed below. Note that the input parameter `sigma` defines the kernel parameters σ.

- "gaussian": A Gaussian kernel,

$$k(x,y) = exp(-z(x,y)/\sigma^2)$$

- "matern": A Matern kernel with $\sigma = (\lambda, \nu)$,

$$k(x,y) = bc^{\nu} z(x,y)^{\nu/2} K_{\nu}(cz(x,y)^{0.5})$$

where $b = 2^{1-\nu}(\Gamma(\nu))^{-1}$, $c = (2\nu)^{0.5}\lambda^{-1}$ and $K_{\nu}(x)$ is the modified Bessel function of the second kind. Note that λ is the length-scale parameter and ν is the smoothness parameter (which defaults to 2.5 for $steinOrder = 1$ and 4.5 for $steinOrder = 2$).

- "RQ": A rational quadratic kernel,

$$k(x,y) = (1 + \sigma^{-2} z(x,y))^{-1}$$

- "product": The product kernel that appears in Oates et al (2017) with $\sigma = (a, b)$

$$k(x,y) = (1 + az(x) + az(y))^{-1} exp(-0.5b^{-2} z(x,y))$$

- "prodsim": A slightly different product kernel with $\sigma = (a, b)$ (see e.g. https://www.imperial.ac.uk/inference-group/projects/monte-carlo-methods/control-functionals/),

$$k(x,y) = (1 + az(x))^{-1}(1 + az(y))^{-1} exp(-0.5b^{-2} z(x,y))$$

In the above equations, $z(x) = \sum_j x[j]^2$ and $z(x,y) = \sum_j (x[j] - y[j])^2$. For the last two kernels, the code only has implementations for steinOrder=1. Each combination of steinOrder and kernel_function above is currently hard-coded but it may be possible to extend this to other kernels in future versions using autodiff. The calculations for the first three kernels above are detailed in South et al (2020).

Author(s)

Leah F. South

References

South, L. F., Karvonen, T., Nemeth, C., Girolami, M. and Oates, C. J. (2020). Semi-Exact Control Functionals From Sard's Method. https://arxiv.org/abs/2002.00033

See Also

See ZVCV for examples and related functions. See aSECF_crossval for a function to choose between different kernels for this estimator.

aSECF_crossval *Approximate semi-exact control functionals (aSECF) with cross-validation*

Description

This function chooses between a list of kernel tuning parameters (sigma_list) or a list of K0 matrices (K0_list) for the approximate semi-exact control functionals method described in South et al (2020). The latter requires calculating and storing kernel matrices using K0_fn but it is more flexible because it can be used to choose the Stein operator order and the kernel function, in addition to its parameters. It is also faster to pre-specify K0_fn. For estimation with fixed kernel parameters, use aSECF.

Usage

```
aSECF_crossval(
  integrands,
  samples,
  derivatives,
  polyorder = NULL,
  steinOrder = NULL,
  kernel_function = NULL,
  sigma_list = NULL,
  est_inds = NULL,
  apriori = NULL,
  num_nystrom = NULL,
  conjugate_gradient = TRUE,
  reltol = 0.01,
  folds = NULL,
  diagnostics = FALSE
)
```

Arguments

integrands	An N by k matrix of integrands (evaluations of the function of interest)
samples	An N by d matrix of samples from the target
derivatives	An N by d matrix of derivatives of the log target with respect to the parameters
polyorder	(optional) The order of the polynomial to be used in the parametric component, with a default of 1. We recommend keeping this value low (e.g. only 1-2).
steinOrder	(optional) This is the order of the Stein operator. The default is 1 in the control functionals paper (Oates et al, 2017) and 2 in the semi-exact control functionals paper (South et al, 2020). The following values are currently available: 1 for all kernels and 2 for "gaussian", "matern" and "RQ". See below for further details.
kernel_function	(optional) Choose between "gaussian", "matern", "RQ", "product" or "prodsim". See below for further details.

`sigma_list` (optional between this and `K0_list`) A list of tuning parameters for the specified kernel. This involves a list of single length-scale parameter in "gaussian" and "RQ", a list of vectors containing length-scale and smoothness parameters in "matern" and a list of vectors of the two parameters in "product" and "prodsim". See below for further details. When `sigma_list` is specified and not `K0_list`, the $K0$ matrix is computed twice for each selected tuning parameter.

`est_inds` (optional) A vector of indices for the estimation-only samples. The default when `est_inds` is missing or `NULL` is to perform both estimation of the control variates and evaluation of the integral using all samples. Otherwise, the samples from `est_inds` are used in estimating the control variates and the remainder are used in evaluating the integral. Splitting the indices in this way can be used to reduce bias from adaption and to make computation feasible for very large sample sizes (small `est_inds` is faster), but in general in will increase the variance of the estimator.

`apriori` (optional) A vector containing the subset of parameter indices to use in the polynomial. Typically this argument would only be used if the dimension of the problem is very large or if prior information about parameter dependencies is known. The default is to use all parameters $1 : d$ where d is the dimension of the target.

`num_nystrom` (optional) The number of samples to use in the Nystrom approximation, with a default of ceiling(sqrt(N)). The nystrom indices cannot be passed in here because of the way the cross-validation has been set up.

`conjugate_gradient` (optional) A flag for whether to perform conjugate gradient to further speed up the nystrom approximation (the default is true).

`reltol` (optional) The relative tolerance for choosing when the stop conjugate gradient iterations (the default is 1e-02). using `squareNorm`, as long as the `nystrom_inds` are `NULL`.

`folds` (optional) The number of folds for cross-validation. The default is five.

`diagnostics` (optional) A flag for whether to return the necessary outputs for plotting or estimating using the fitted model. The default is `false` since this requires some additional computation when `est_inds` is `NULL`.

Value

A list with the following elements:

- `expectation`: The estimate(s) of the (k) expectations(s).
- `mse`: A matrix of the cross-validation mean square prediction errors. The number of columns is the number of tuning options given and the number of rows is k, the number of integrands of interest.
- `optinds`: The optimal indices from the list for each expectation.
- `cond_no`: (Only if `conjugate_gradient = TRUE`) The condition number of the matrix being solved using conjugate gradient.
- `iter`: (Only if `conjugate_gradient = TRUE`) The number of conjugate gradient iterations

- `f_true`: (Only if `est_inds` is not `NULL`) The integrands for the evaluation set. This should be the same as integrands[setdiff(1:N,est_inds),].
- `f_hat`: (Only if `est_inds` is not `NULL`) The fitted values for the integrands in the evaluation set. This can be used to help assess the performance of the Gaussian process model.
- `a`: (Only if `diagnostics = TRUE`) The value of a as described in South et al (2020), where predictions are of the form $f_hat = K0 * a + Phi * b$ for heldout K0 and Phi matrices and estimators using heldout samples are of the form $mean(f - f_hat) + b[1]$.
- `b`: (Only if `diagnostics = TRUE`) The value of b as described in South et al (2020), where predictions are of the form $f_hat = K0 * a + Phi * b$ for heldout K0 and Phi matrices and estimators using heldout samples are of the form $mean(f - f_hat) + b[1]$.
- `ny_inds`: (Only if `diagnostics = TRUE`) The indices of the samples used in the nystrom approximation (this will match nystrom_inds if this argument was not `NULL`).

On the choice of σ, the kernel and the Stein order

The kernel in Stein-based kernel methods is $L_x L_y k(x, y)$ where L_x is a first or second order Stein operator in x and $k(x, y)$ is some generic kernel to be specified.

The Stein operators for distribution $p(x)$ are defined as:

- `steinOrder=1`: $L_x g(x) = \nabla_x^T g(x) + \nabla_x \log p(x)^T g(x)$ (see e.g. Oates el al (2017))
- `steinOrder=2`: $L_x g(x) = \Delta_x g(x) + \nabla_x logp(x)^T \nabla_x g(x)$ (see e.g. South el al (2020))

Here ∇_x is the first order derivative wrt x and $\Delta_x = \nabla_x^T \nabla_x$ is the Laplacian operator.

The generic kernels which are implemented in this package are listed below. Note that the input parameter `sigma` defines the kernel parameters σ.

- `"gaussian"`: A Gaussian kernel,

$$k(x, y) = exp(-z(x, y)/\sigma^2)$$

- `"matern"`: A Matern kernel with $\sigma = (\lambda, \nu)$,

$$k(x, y) = bc^\nu z(x, y)^{\nu/2} K_\nu(cz(x, y)^{0.5})$$

  where $b = 2^{1-\nu}(\Gamma(\nu))^{-1}$, $c = (2\nu)^{0.5}\lambda^{-1}$ and $K_\nu(x)$ is the modified Bessel function of the second kind. Note that λ is the length-scale parameter and ν is the smoothness parameter (which defaults to 2.5 for $steinOrder = 1$ and 4.5 for $steinOrder = 2$).
- `"RQ"`: A rational quadratic kernel,

$$k(x, y) = (1 + \sigma^{-2} z(x, y))^{-1}$$

- `"product"`: The product kernel that appears in Oates et al (2017) with $\sigma = (a, b)$

$$k(x, y) = (1 + az(x) + az(y))^{-1} exp(-0.5b^{-2} z(x, y))$$

- `"prodsim"`: A slightly different product kernel with $\sigma = (a, b)$ (see e.g. https://www.imperial.ac.uk/inference-group/projects/monte-carlo-methods/control-functionals/),

$$k(x, y) = (1 + az(x))^{-1}(1 + az(y))^{-1} exp(-0.5b^{-2} z(x, y))$$

In the above equations, $z(x) = \sum_j x[j]^2$ and $z(x,y) = \sum_j (x[j] - y[j])^2$. For the last two kernels, the code only has implementations for `steinOrder=1`. Each combination of `steinOrder` and `kernel_function` above is currently hard-coded but it may be possible to extend this to other kernels in future versions using autodiff. The calculations for the first three kernels above are detailed in South et al (2020).

Author(s)

Leah F. South

References

South, L. F., Karvonen, T., Nemeth, C., Girolami, M. and Oates, C. J. (2020). Semi-Exact Control Functionals From Sard's Method. https://arxiv.org/abs/2002.00033

See Also

See ZVCV for examples and related functions. See `aSECF_crossval` for a function to choose between different kernels for this estimator.

`CF` *Control functionals (CF)*

Description

This function performs control functionals as described in Oates et al (2017). To choose between different kernels using cross-validation, use `CF_crossval`.

Usage

```
CF(
  integrands,
  samples,
  derivatives,
  steinOrder = NULL,
  kernel_function = NULL,
  sigma = NULL,
  K0 = NULL,
  est_inds = NULL,
  one_in_denom = FALSE,
  diagnostics = FALSE
)
```

Arguments

integrands	An N by k matrix of integrands (evaluations of the function of interest)
samples	An N by d matrix of samples from the target
derivatives	An N by d matrix of derivatives of the log target with respect to the parameters
steinOrder	(optional) This is the order of the Stein operator. The default is 1 in the control functionals paper (Oates et al, 2017) and 2 in the semi-exact control functionals paper (South et al, 2020). The following values are currently available: 1 for all kernels and 2 for "gaussian", "matern" and "RQ". See below for further details.
kernel_function	(optional) Choose between "gaussian", "matern", "RQ", "product" or "prodsim". See below for further details.
sigma	(optional) The tuning parameters of the specified kernel. This involves a single length-scale parameter in "gaussian" and "RQ", a length-scale and a smoothness parameter in "matern" and two parameters in "product" and "prodsim". See below for further details.
K0	(optional) The kernel matrix. One can specify either this or all of sigma, steinOrder and kernel_function. The former involves pre-computing the kernel matrix using K0_fn and is more efficient when using multiple estimators out of CF, SECF and aSECF or when using the cross-validation functions.
est_inds	(optional) A vector of indices for the estimation-only samples. The default when est_inds is missing or NULL is to perform both estimation of the control variates and evaluation of the integral using all samples. Otherwise, the samples from est_inds are used in estimating the control variates and the remainder are used in evaluating the integral. Splitting the indices in this way can be used to reduce bias from adaption and to make computation feasible for very large sample sizes (small est_inds is faster), but in general in will increase the variance of the estimator.
one_in_denom	(optional) Whether or not to include a $1+$ in the denominator of the control functionals estimator, as in equation 2 on p703 of Oates et al (2017). The $1+$ in the denominator is an arbitrary choice so we set it to zero by default.
diagnostics	(optional) A flag for whether to return the necessary outputs for plotting or estimating using the fitted model. The default is false since this requires some additional computation when est_inds is NULL.

Value

A list with the following elements:

- expectation: The estimate(s) of the (k) expectation(s).
- f_true: (Only if est_inds is not NULL) The integrands for the evaluation set. This should be the same as integrands[setdiff(1:N,est_inds),].
- f_hat: (Only if est_inds is not NULL) The fitted values for the integrands in the evaluation set. This can be used to help assess the performance of the Gaussian process model.
- a: (Only if diagnostics = TRUE) The value of a as described in South et al (2020), where predictions are of the form $f_hat = K0 * a + 1 * b$ for heldout K0 and estimators using heldout samples are of the form $mean(f - f_hat) + b$.

- b: (Only if `diagnostics = TRUE`) The value of b as described in South et al (2020), where predictions are of the form $f_hat = K0 * a + 1 * b$ for heldout K0 and estimators using heldout samples are of the form $mean(f - f_hat) + b$.
- ksd: (Only if `diagnostics = TRUE`) An estimated kernel Stein discrepancy based on the fitted model that can be used for diagnostic purposes. See South et al (2020) for further details.
- bound_const: (Only if `diagnostics = TRUE` and `est_inds=NULL`) This is such that the absolute error for the estimator should be less than $ksd \times bound_const$.

Warning

Solving the linear system in CF has $O(N^3)$ complexity and is therefore not suited to large N. Using est_inds will instead have an $O(N_0^3)$ cost in solving the linear system and an $O((N - N_0)^2)$ cost in handling the remaining samples, where N_0 is the length of est_inds. This can be much cheaper for large N.

On the choice of σ, the kernel and the Stein order

The kernel in Stein-based kernel methods is $L_x L_y k(x, y)$ where L_x is a first or second order Stein operator in x and $k(x, y)$ is some generic kernel to be specified.

The Stein operators for distribution $p(x)$ are defined as:

- `steinOrder=1`: $L_x g(x) = \nabla_x^T g(x) + \nabla_x \log p(x)^T g(x)$ (see e.g. Oates el al (2017))
- `steinOrder=2`: $L_x g(x) = \Delta_x g(x) + \nabla_x logp(x)^T \nabla_x g(x)$ (see e.g. South el al (2020))

Here ∇_x is the first order derivative wrt x and $\Delta_x = \nabla_x^T \nabla_x$ is the Laplacian operator.

The generic kernels which are implemented in this package are listed below. Note that the input parameter `sigma` defines the kernel parameters σ.

- `"gaussian"`: A Gaussian kernel,

$$k(x, y) = exp(-z(x, y)/\sigma^2)$$

- `"matern"`: A Matern kernel with $\sigma = (\lambda, \nu)$,

$$k(x, y) = bc^\nu z(x, y)^{\nu/2} K_\nu(cz(x, y)^{0.5})$$

  where $b = 2^{1-\nu}(\Gamma(\nu))^{-1}$, $c = (2\nu)^{0.5}\lambda^{-1}$ and $K_\nu(x)$ is the modified Bessel function of the second kind. Note that λ is the length-scale parameter and ν is the smoothness parameter (which defaults to 2.5 for $steinOrder = 1$ and 4.5 for $steinOrder = 2$).
- `"RQ"`: A rational quadratic kernel,

$$k(x, y) = (1 + \sigma^{-2} z(x, y))^{-1}$$

- `"product"`: The product kernel that appears in Oates et al (2017) with $\sigma = (a, b)$

$$k(x, y) = (1 + az(x) + az(y))^{-1} exp(-0.5 b^{-2} z(x, y))$$

- `"prodsim"`: A slightly different product kernel with $\sigma = (a, b)$ (see e.g. https://www.imperial.ac.uk/inference-group/projects/monte-carlo-methods/control-functionals/),

$$k(x, y) = (1 + az(x))^{-1}(1 + az(y))^{-1} exp(-0.5 b^{-2} z(x, y))$$

In the above equations, $z(x) = \sum_j x[j]^2$ and $z(x,y) = \sum_j (x[j] - y[j])^2$. For the last two kernels, the code only has implementations for `steinOrder=1`. Each combination of `steinOrder` and `kernel_function` above is currently hard-coded but it may be possible to extend this to other kernels in future versions using autodiff. The calculations for the first three kernels above are detailed in South et al (2020).

Author(s)

Leah F. South

References

Oates, C. J., Girolami, M. & Chopin, N. (2017). Control functionals for Monte Carlo integration. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 79(3), 695-718.

South, L. F., Karvonen, T., Nemeth, C., Girolami, M. and Oates, C. J. (2020). Semi-Exact Control Functionals From Sard's Method. https://arxiv.org/abs/2002.00033

See Also

See ZVCV for examples and related functions. See `CF_crossval` for a function to choose between different kernels for this estimator.

`CF_crossval` *Control functionals (CF) with cross-validation*

Description

This function chooses between a list of kernel tuning parameters (`sigma_list`) or a list of K0 matrices (`K0_list`) for the control functionals method described in Oates et al (2017). The latter requires calculating and storing kernel matrices using `K0_fn` but it is more flexible because it can be used to choose the Stein operator order and the kernel function, in addition to its parameters. It is also faster to pre-specify `K0_fn`. For estimation with fixed kernel parameters, use `CF`.

Usage

```
CF_crossval(
  integrands,
  samples,
  derivatives,
  steinOrder = NULL,
  kernel_function = NULL,
  sigma_list = NULL,
  K0_list = NULL,
  est_inds = NULL,
  log_weights = NULL,
  one_in_denom = FALSE,
  folds = NULL,
  diagnostics = FALSE
)
```

Arguments

integrands	An N by k matrix of integrands (evaluations of the function of interest)
samples	An N by d matrix of samples from the target
derivatives	An N by d matrix of derivatives of the log target with respect to the parameters
steinOrder	(optional) This is the order of the Stein operator. The default is 1 in the control functionals paper (Oates et al, 2017) and 2 in the semi-exact control functionals paper (South et al, 2020). The following values are currently available: 1 for all kernels and 2 for "gaussian", "matern" and "RQ". See below for further details.
kernel_function	(optional) Choose between "gaussian", "matern", "RQ", "product" or "prodsim". See below for further details.
sigma_list	(optional between this and K0_list) A list of tuning parameters for the specified kernel. This involves a list of single length-scale parameter in "gaussian" and "RQ", a list of vectors containing length-scale and smoothness parameters in "matern" and a list of vectors of the two parameters in "product" and "prodsim". See below for further details. When sigma_list is specified and not K0_list, the $K0$ matrix is computed twice for each selected tuning parameter.
K0_list	(optional between this and sigma_list) A list of kernel matrices, which can be calculated using K0_fn.
est_inds	(optional) A vector of indices for the estimation-only samples. The default when est_inds is missing or NULL is to perform both estimation of the control variates and evaluation of the integral using all samples. Otherwise, the samples from est_inds are used in estimating the control variates and the remainder are used in evaluating the integral. Splitting the indices in this way can be used to reduce bias from adaption and to make computation feasible for very large sample sizes (small est_inds is faster), but in general in will increase the variance of the estimator.
log_weights	(optional) A vector of length N containing the logged weights of the samples. The default is equal weights. The weights are only used in estimating the cross-validation error. This method is not implemented for the case where est_inds is specified becausing specifying est_inds typically indicates a desire for an unbiased estimator and using self-normalised importance weights introduces bias.
one_in_denom	(optional) Whether or not to include a $1+$ in the denominator of the control functionals estimator, as in equation 2 on p703 of Oates et al (2017). The $1+$ in the denominator is an arbitrary choice so we set it to zero by default.
folds	(optional) The number of folds for cross-validation. The default is five.
diagnostics	(optional) A flag for whether to return the necessary outputs for plotting or estimating using the fitted model. The default is false since this requires some additional computation when est_inds is NULL.

Value

A list with the following elements:

- expectation: The estimate(s) of the (k) expectation(s).

- `mse`: A matrix of the cross-validation mean square prediction errors. The number of columns is the number of tuning options given and the number of rows is k, the number of integrands of interest.
- `optinds`: The optimal indices from the list for each expectation.
- `f_true`: (Only if `est_inds` is not `NULL`) The integrands for the evaluation set. This should be the same as integrands[setdiff(1:N,est_inds),].
- `f_hat`: (Only if `est_inds` is not `NULL`) The fitted values for the integrands in the evaluation set. This can be used to help assess the performance of the Gaussian process model.
- `a`: (Only if `diagnostics = TRUE`) The value of a as described in South et al (2020), where predictions are of the form $f_hat = K0 * a + 1 * b$ for heldout K0 and estimators using heldout samples are of the form $mean(f - f_hat) + b$.
- `b`: (Only if `diagnostics = TRUE`) The value of b as described in South et al (2020), where predictions are of the form $f_hat = K0 * a + 1 * b$ for heldout K0 and estimators using heldout samples are of the form $mean(f - f_hat) + b$.
- `ksd`: (Only if `diagnostics = TRUE`) An estimated kernel Stein discrepancy based on the fitted model that can be used for diagnostic purposes. See South et al (2020) for further details.
- `bound_const`: (Only if `diagnostics = TRUE` and `est_inds=NULL`) This is such that the absolute error for the estimator should be less than $ksd \times bound_const$.

Warning

Solving the linear system in CF has $O(N^3)$ complexity and is therefore not suited to large N. Using est_inds will instead have an $O(N_0^3)$ cost in solving the linear system and an $O((N - N_0)^2)$ cost in handling the remaining samples, where N_0 is the length of est_inds. This can be much cheaper for large N.

On the choice of σ, the kernel and the Stein order

The kernel in Stein-based kernel methods is $L_x L_y k(x, y)$ where L_x is a first or second order Stein operator in x and $k(x, y)$ is some generic kernel to be specified.

The Stein operators for distribution $p(x)$ are defined as:

- `steinOrder=1`: $L_x g(x) = \nabla_x^T g(x) + \nabla_x \log p(x)^T g(x)$ (see e.g. Oates el al (2017))
- `steinOrder=2`: $L_x g(x) = \Delta_x g(x) + \nabla_x logp(x)^T \nabla_x g(x)$ (see e.g. South el al (2020))

Here ∇_x is the first order derivative wrt x and $\Delta_x = \nabla_x^T \nabla_x$ is the Laplacian operator.

The generic kernels which are implemented in this package are listed below. Note that the input parameter `sigma` defines the kernel parameters σ.

- `"gaussian"`: A Gaussian kernel,

$$k(x, y) = exp(-z(x, y)/\sigma^2)$$

- `"matern"`: A Matern kernel with $\sigma = (\lambda, \nu)$,

$$k(x, y) = bc^\nu z(x, y)^{\nu/2} K_\nu(cz(x, y)^{0.5})$$

where $b = 2^{1-\nu}(\Gamma(\nu))^{-1}$, $c = (2\nu)^{0.5}\lambda^{-1}$ and $K_\nu(x)$ is the modified Bessel function of the second kind. Note that λ is the length-scale parameter and ν is the smoothness parameter (which defaults to 2.5 for $steinOrder = 1$ and 4.5 for $steinOrder = 2$).

- "RQ": A rational quadratic kernel,

$$k(x,y) = (1 + \sigma^{-2} z(x,y))^{-1}$$

- "product": The product kernel that appears in Oates et al (2017) with $\sigma = (a, b)$

$$k(x,y) = (1 + az(x) + az(y))^{-1} exp(-0.5 b^{-2} z(x,y))$$

- "prodsim": A slightly different product kernel with $\sigma = (a, b)$ (see e.g. https://www.imperial.ac.uk/inference-group/projects/monte-carlo-methods/control-functionals/),

$$k(x,y) = (1 + az(x))^{-1} (1 + az(y))^{-1} exp(-0.5 b^{-2} z(x,y))$$

In the above equations, $z(x) = \sum_j x[j]^2$ and $z(x,y) = \sum_j (x[j] - y[j])^2$. For the last two kernels, the code only has implementations for steinOrder=1. Each combination of steinOrder and kernel_function above is currently hard-coded but it may be possible to extend this to other kernels in future versions using autodiff. The calculations for the first three kernels above are detailed in South et al (2020).

Author(s)

Leah F. South

References

Oates, C. J., Girolami, M. & Chopin, N. (2017). Control functionals for Monte Carlo integration. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 79(3), 695-718.

South, L. F., Karvonen, T., Nemeth, C., Girolami, M. and Oates, C. J. (2020). Semi-Exact Control Functionals From Sard's Method. https://arxiv.org/abs/2002.00033

See Also

See ZVCV for examples and related functions. See CF for a function to perform control functionals with fixed kernel specifications.

---

evidence *Evidence estimation with ZV-CV*

Description

The functions evidence_CTI and evidence_CTI_CF can be used to improve upon the thermodynamic integration (TI) estimate of the normalising constant with ZV-CV and CF, respectively. The functions evidence_SMC and evidence_SMC_CF do the same thing for the sequential Monte Carlo (SMC) normalising constant identity.

Usage

```
evidence_CTI(
  samples,
  loglike,
  der_loglike,
  der_logprior,
  temperatures,
  temperatures_all,
  most_recent,
  est_inds,
  options,
  folds = 5
)

evidence_CTI_CF(
  samples,
  loglike,
  der_loglike,
  der_logprior,
  temperatures,
  temperatures_all,
  most_recent,
  est_inds,
  steinOrder,
  kernel_function,
  sigma_list,
  folds = 5
)

evidence_SMC(
  samples,
  loglike,
  der_loglike,
  der_logprior,
  temperatures,
  temperatures_all,
  most_recent,
  est_inds,
  options,
  folds = 5
)

evidence_SMC_CF(
  samples,
  loglike,
  der_loglike,
  der_logprior,
  temperatures,
```

```
  temperatures_all,
  most_recent,
  est_inds,
  steinOrder,
  kernel_function,
  sigma_list,
  folds = 5
)
```

Arguments

samples	An N by d by T matrix of samples from the T power posteriors, where N is the number of samples and d is the dimension of the target distribution
loglike	An N by T matrix of log likelihood values corresponding to samples
der_loglike	An N by d by T matrix of the derivatives of the log likelihood with respect to the parameters, with parameter values corresponding to samples
der_logprior	An N by d by T matrix of the derivatives of the log prior with respect to the parameters, with parameter values corresponding to samples
temperatures	A vector of length T of temperatures for the power posterior temperatures
temperatures_all	An adjusted vector of length tau of temperatures. Better performance should be obtained with a more conservative temperature schedule. See Expand_Temperatures for a function to adjust the temperatures.
most_recent	A vector of length tau which gives the indices in the original temperatures that the new temperatures correspond to.
est_inds	(optional) A vector of indices for the estimation-only samples. The default when est_inds is missing or NULL is to perform both estimation of the control variates and evaluation of the integral using all samples. Otherwise, the samples from est_inds are used in estimating the control variates and the remainder are used in evaluating the integral. Splitting the indices in this way can be used to reduce bias from adaption and to make computation feasible for very large sample sizes (small est_inds is faster), but in general in will increase the variance of the estimator.
options	A list of control variate specifications for ZV-CV. This can be a single list containing the elements below (the defaults are used for elements which are not specified). Alternatively, it can be a list of lists containing any or all of the elements below. Where the latter is used, the function zvcv automatically selects the best performing option based on cross-validation.
folds	The number of folds used in k-fold cross-validation for selecting the optimal control variate. For ZV-CV, this may include selection of the optimal polynomial order, regression type and subset of parameters depending on options. For CF, this includes the selection of the optimal tuning parameters in sigma_list. The default is five.
steinOrder	(optional) This is the order of the Stein operator. The default is 1 in the control functionals paper (Oates et al, 2017) and 2 in the semi-exact control functionals paper (South et al, 2020). The following values are currently available: 1 for all kernels and 2 for "gaussian", "matern" and "RQ". See below for further details.

`kernel_function`
: (optional) Choose between "gaussian", "matern", "RQ", "product" or "prodsim". See below for further details.

`sigma_list`
: (optional between this and `K0_list`) A list of tuning parameters for the specified kernel. This involves a list of single length-scale parameter in "gaussian" and "RQ", a list of vectors containing length-scale and smoothness parameters in "matern" and a list of vectors of the two parameters in "product" and "prodsim". See below for further details. When `sigma_list` is specified and not `K0_list`, the $K0$ matrix is computed twice for each selected tuning parameter.

Value

The function `evidence_CTI` returns a list, containing the following components:

- `log_evidence_PS1`: The 1st order quadrature estimate for the log normalising constant
- `log_evidence_PS2`: The 2nd order quadrature estimate for the log normalising constant
- `regression_LL`: The set of tau `zvcv` type returns for the 1st order quadrature expectations
- `regression_vLL`: The set of tau `zvcv` type returns for the 2nd order quadrature expectations

The function `evidence_CTI_CF` returns a list, containing the following components:

- `log_evidence_PS1`: The 1st order quadrature estimate for the log normalising constant
- `log_evidence_PS2`: The 2nd order quadrature estimate for the log normalising constant
- `regression_LL`: The set of tau `CF_crossval` type returns for the 1st order quadrature expectations
- `regression_vLL`: The set of tau `CF_crossval` type returns for the 2nd order quadrature expectations
- `selected_LL_CF`: The set of tau selected tuning parameters from `sigma_list` for the 1st order quadrature expectations.
- `selected_vLL_CF`: The set of tau selected tuning parameters from `sigma_list` for the 2nd order quadrature expectations.

The function `evidence_SMC` returns a list, containing the following components:

- `log_evidence`: The logged SMC estimate for the normalising constant
- `regression_SMC`: The set of tau `zvcv` type returns for the expectations

The function `evidence_SMC_CF` returns a list, containing the following components:

- `log_evidence`: The logged SMC estimate for the normalising constant
- `regression_SMC`: The set of tau `CF_crossval` type returns for the expectations
- `selected_CF`: The set of tau selected tuning parameters from `sigma_list` for the expectations

On the choice of σ, the kernel and the Stein order

The kernel in Stein-based kernel methods is $L_x L_y k(x,y)$ where L_x is a first or second order Stein operator in x and $k(x,y)$ is some generic kernel to be specified.

The Stein operators for distribution $p(x)$ are defined as:

- `steinOrder=1`: $L_x g(x) = \nabla_x^T g(x) + \nabla_x \log p(x)^T g(x)$ (see e.g. Oates el al (2017))
- `steinOrder=2`: $L_x g(x) = \Delta_x g(x) + \nabla_x logp(x)^T \nabla_x g(x)$ (see e.g. South el al (2020))

Here ∇_x is the first order derivative wrt x and $\Delta_x = \nabla_x^T \nabla_x$ is the Laplacian operator.

The generic kernels which are implemented in this package are listed below. Note that the input parameter `sigma` defines the kernel parameters σ.

- `"gaussian"`: A Gaussian kernel,

$$k(x,y) = exp(-z(x,y)/\sigma^2)$$

- `"matern"`: A Matern kernel with $\sigma = (\lambda, \nu)$,

$$k(x,y) = bc^\nu z(x,y)^{\nu/2} K_\nu(cz(x,y)^{0.5})$$

  where $b = 2^{1-\nu}(\Gamma(\nu))^{-1}$, $c = (2\nu)^{0.5}\lambda^{-1}$ and $K_\nu(x)$ is the modified Bessel function of the second kind. Note that λ is the length-scale parameter and ν is the smoothness parameter (which defaults to 2.5 for $steinOrder = 1$ and 4.5 for $steinOrder = 2$).
- `"RQ"`: A rational quadratic kernel,

$$k(x,y) = (1 + \sigma^{-2} z(x,y))^{-1}$$

- `"product"`: The product kernel that appears in Oates et al (2017) with $\sigma = (a, b)$

$$k(x,y) = (1 + az(x) + az(y))^{-1} exp(-0.5 b^{-2} z(x,y))$$

- `"prodsim"`: A slightly different product kernel with $\sigma = (a, b)$ (see e.g. https://www.imperial.ac.uk/inference-group/projects/monte-carlo-methods/control-functionals/),

$$k(x,y) = (1 + az(x))^{-1}(1 + az(y))^{-1} exp(-0.5 b^{-2} z(x,y))$$

In the above equations, $z(x) = \sum_j x[j]^2$ and $z(x,y) = \sum_j (x[j] - y[j])^2$. For the last two kernels, the code only has implementations for `steinOrder=1`. Each combination of `steinOrder` and `kernel_function` above is currently hard-coded but it may be possible to extend this to other kernels in future versions using autodiff. The calculations for the first three kernels above are detailed in South et al (2020).

Author(s)

Leah F. South

References

Mira, A., Solgi, R., & Imparato, D. (2013). Zero variance Markov chain Monte Carlo for Bayesian estimators. Statistics and Computing, 23(5), 653-662.

South, L. F., Oates, C. J., Mira, A., & Drovandi, C. (2019). Regularised zero variance control variates for high-dimensional variance reduction. https://arxiv.org/abs/1811.05073

See Also

See an example at VDP and see ZVCV for more package details. See Expand_Temperatures for a function that can be used to find stricter (or less stricter) temperature schedules based on the conditional effective sample size.

Expand_Temperatures *Adjusting the temperature schedule*

Description

This function is used to adjust the temperature schedule so that it is more (or less) strict than the original.

Usage

```
Expand_Temperatures(
  temperatures,
  loglike,
  rho,
  bisec_tol = .Machine$double.eps^0.25
)
```

Arguments

temperatures	A vector of length T temperatures for the power posterior temperatures.
loglike	An N by T matrix of log likelihood values corresponding to the samples.
rho	The tolerance for the new temperatures. Temperatures are selected so that the conditional effective sample size (CESS) at each temperature is $\rho * N$ where N is the population size.
bisec_tol	The tolerance for the bisection method used in selecting temperatures. The default is .Machine$double.eps^0.25

Value

A list is returned, containing the following components:

- temperatures_all: The new set of temperatures of length tau.
- relevant_samples: A vector of length tau containing indices to show which particle sets the new temperatures are based on.
- logw: An N by tau matrix of log normalised weights of the particles

Author(s)

Leah F. South

References

South, L. F., Oates, C. J., Mira, A., & Drovandi, C. (2019). Regularised zero variance control variates for high-dimensional variance reduction. https://arxiv.org/abs/1811.05073

See Also

See `evidence` for functions to estimate the evidence, `VDP` for an example and ZVCV for more package details.

`getX` *ZV-CV design matrix*

Description

The function `getX` is used to get the matrix of covariates for the regression based on a specified polynomial order.

Usage

```
getX(samples, derivatives, polyorder)
```

Arguments

`samples`	An N by d matrix of samples from the target
`derivatives`	An N by d matrix of derivatives of the log target with respect to the parameters
`polyorder`	The order of the polynomial.

Value

The design matrix for the regression (except for the column of 1's for the intercept).

See Also

`Phi_fn` for a very similar function for use in semi-exact control functionals. The function `Phi_fn` essentially gets the same matrix but with a column of ones added.

K0_fn *Kernel matrix calculation*

Description

This function calculates the full K_0 matrix, which is a first or second order Stein operator applied to a standard kernel. The output of this function can be used as an argument to CF, CF_crossval, SECF, SECF_crossval, aSECF and aSECF_crossval. The kernel matrix is automatically computed in all of the above methods, but it is faster to calculate in advance when using more than one of the above functions and when using any of the crossval functions.

Usage

```
K0_fn(
  samples,
  derivatives,
  sigma,
  steinOrder,
  kernel_function,
  Z = NULL,
  nystrom_inds = NULL
)
```

Arguments

samples	An N by d matrix of samples from the target
derivatives	An N by d matrix of derivatives of the log target with respect to the parameters
sigma	The tuning parameters of the specified kernel. This involves a single length-scale parameter in "gaussian" and "RQ", a length-scale and a smoothness parameter in "matern" and two parameters in "product" and "prodsim". See below for further details.
steinOrder	This is the order of the Stein operator. The default is 1 in the control functionals paper (Oates et al, 2017) and 2 in the semi-exact control functionals paper (South et al, 2020). The following values are currently available: 1 for all kernels and 2 for "gaussian", "matern" and "RQ". See below for further details.
kernel_function	Choose between "gaussian", "matern", "RQ", "product" or "prodsim". See below for further details.
Z	(optional) An N by N (or N by m where m is the length of nystrom_inds). This can be calculated using squareNorm.
nystrom_inds	(optional) The sample indices to be used in the Nystrom approximation (for when using aSECF).

Value

An N by N kernel matrix (or N by m where m is the length of nystrom_inds).

On the choice of σ, the kernel and the Stein order

The kernel in Stein-based kernel methods is $L_x L_y k(x,y)$ where L_x is a first or second order Stein operator in x and $k(x,y)$ is some generic kernel to be specified.

The Stein operators for distribution $p(x)$ are defined as:

- `steinOrder=1`: $L_x g(x) = \nabla_x^T g(x) + \nabla_x \log p(x)^T g(x)$ (see e.g. Oates el al (2017))
- `steinOrder=2`: $L_x g(x) = \Delta_x g(x) + \nabla_x logp(x)^T \nabla_x g(x)$ (see e.g. South el al (2020))

Here ∇_x is the first order derivative wrt x and $\Delta_x = \nabla_x^T \nabla_x$ is the Laplacian operator.

The generic kernels which are implemented in this package are listed below. Note that the input parameter `sigma` defines the kernel parameters σ.

- `"gaussian"`: A Gaussian kernel,

$$k(x,y) = exp(-z(x,y)/\sigma^2)$$

- `"matern"`: A Matern kernel with $\sigma = (\lambda, \nu)$,

$$k(x,y) = bc^\nu z(x,y)^{\nu/2} K_\nu(cz(x,y)^{0.5})$$

  where $b = 2^{1-\nu}(\Gamma(\nu))^{-1}$, $c = (2\nu)^{0.5}\lambda^{-1}$ and $K_\nu(x)$ is the modified Bessel function of the second kind. Note that λ is the length-scale parameter and ν is the smoothness parameter (which defaults to 2.5 for $steinOrder = 1$ and 4.5 for $steinOrder = 2$).
- `"RQ"`: A rational quadratic kernel,

$$k(x,y) = (1 + \sigma^{-2} z(x,y))^{-1}$$

- `"product"`: The product kernel that appears in Oates et al (2017) with $\sigma = (a, b)$

$$k(x,y) = (1 + az(x) + az(y))^{-1} exp(-0.5 b^{-2} z(x,y))$$

- `"prodsim"`: A slightly different product kernel with $\sigma = (a, b)$ (see e.g. https://www.imperial.ac.uk/inference-group/projects/monte-carlo-methods/control-functionals/),

$$k(x,y) = (1 + az(x))^{-1}(1 + az(y))^{-1} exp(-0.5 b^{-2} z(x,y))$$

In the above equations, $z(x) = \sum_j x[j]^2$ and $z(x,y) = \sum_j (x[j] - y[j])^2$. For the last two kernels, the code only has implementations for `steinOrder=1`. Each combination of `steinOrder` and `kernel_function` above is currently hard-coded but it may be possible to extend this to other kernels in future versions using autodiff. The calculations for the first three kernels above are detailed in South et al (2020).

Author(s)

Leah F. South

References

Oates, C. J., Girolami, M. & Chopin, N. (2017). Control functionals for Monte Carlo integration. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 79(3), 695-718.

South, L. F., Karvonen, T., Nemeth, C., Girolami, M. and Oates, C. J. (2020). Semi-Exact Control Functionals From Sard's Method. https://arxiv.org/abs/2002.00033

logsumexp *Stable log sum of exponential calculations*

Description

The function logsumexp is used for stable computation of log(sum(exp(x))), which is useful when summing weights for example.

Usage

```
logsumexp(x)
```

Arguments

x	The values for which you want to compute log(sum(exp(x)))

Value

The stable result of log(sum(exp(x)))

See Also

See ZVCV for more package details.

medianTune *Median heuristic*

Description

This function calculates the median heuristic for use in e.g. the Gaussian, Matern and rational quadratic kernels.

Usage

```
medianTune(samples, Z = NULL)
```

Arguments

samples	An N by d matrix of samples from the target
Z	(optional) An NxN matrix of square norms, which can be calculated using squareNorm, as long as the nystrom_inds are NULL.

Value

The median heuristic, which can then be used as the length-scale parameter in the Gaussian, Matern and rational quadratic kernels

Author(s)

Leah F. South

References

Garreau, D., Jitkrittum, W. and Kanagawa, M. (2017). Large sample analysis of the median heuristic. https://arxiv.org/abs/1707.07269

See Also

See medianTune and K0_fn for functions which use this.

---

nearPD *Nearest symmetric positive definite matrix*

Description

This function finds the nearest symmetric positive definite matrix to the given matrix. It is used throughout the package to handle numerical issues in matrix inverses and cholesky decompositions.

Usage

```
nearPD(K0)
```

Arguments

K0	A square matrix

Value

The closest symmetric positive definite matrix to K0.

Author(s)

Adapted from Matlab code by John D'Errico

References

Higham, N. J. (1988). Computing a nearest symmetric positive semidefinite matrix. Linear Algebra and its Applications, 103, 103-118.

D'Errico, J. (2013). nearestSPD Matlab function. https://uk.mathworks.com/matlabcentral/fileexchange/42885-nearestspd.

`Phi_fn` *Phi matrix calculation*

Description

This function calculates the Φ matrix, which is a second order Stein operator applied to a polynomial. See South et al (2020) for further details. This function is not required for estimation but may be useful when evaluation samples are not initially available since estimators using heldout samples are of the form $mean(f - f_hat) + b[1]$ where $f_hat = K0 * a + Phi * b$ for heldout K0 and Phi matrices.

Usage

```
Phi_fn(samples, derivatives, polyorder = NULL, apriori = NULL)
```

Arguments

`samples`	An N by d matrix of samples from the target
`derivatives`	An N by d matrix of derivatives of the log target with respect to the parameters
`polyorder`	(optional) The order of the polynomial to be used in the parametric component, with a default of 1. We recommend keeping this value low (e.g. only 1-2).
`apriori`	(optional) A vector containing the subset of parameter indices to use in the polynomial. Typically this argument would only be used if the dimension of the problem is very large or if prior information about parameter dependencies is known. The default is to use all parameters $1 : d$ where d is the dimension of the target.

Value

An N by Q matrix (where Q is determined by the polynomial order and the apriori).

Author(s)

Leah F. South

References

South, L. F., Karvonen, T., Nemeth, C., Girolami, M. and Oates, C. J. (2020). Semi-Exact Control Functionals From Sard's Method. https://arxiv.org/abs/2002.00033

SECF *Semi-exact control functionals (SECF)*

Description

This function performs semi-exact control functionals as described in South et al (2020). To choose between different kernels using cross-validation, use `SECF_crossval`.

Usage

```
SECF(
  integrands,
  samples,
  derivatives,
  polyorder = NULL,
  steinOrder = NULL,
  kernel_function = NULL,
  sigma = NULL,
  K0 = NULL,
  est_inds = NULL,
  apriori = NULL,
  diagnostics = FALSE
)
```

Arguments

`integrands`	An N by k matrix of integrands (evaluations of the function of interest)
`samples`	An N by d matrix of samples from the target
`derivatives`	An N by d matrix of derivatives of the log target with respect to the parameters
`polyorder`	(optional) The order of the polynomial to be used in the parametric component, with a default of 1. We recommend keeping this value low (e.g. only 1-2).
`steinOrder`	(optional) This is the order of the Stein operator. The default is 1 in the control functionals paper (Oates et al, 2017) and 2 in the semi-exact control functionals paper (South et al, 2020). The following values are currently available: 1 for all kernels and 2 for "gaussian", "matern" and "RQ". See below for further details.
`kernel_function`	(optional) Choose between "gaussian", "matern", "RQ", "product" or "prodsim". See below for further details.
`sigma`	(optional) The tuning parameters of the specified kernel. This involves a single length-scale parameter in "gaussian" and "RQ", a length-scale and a smoothness parameter in "matern" and two parameters in "product" and "prodsim". See below for further details.
`K0`	(optional) The kernel matrix. One can specify either this or all of `sigma`, `steinOrder` and `kernel_function`. The former involves pre-computing the kernel matrix using `K0_fn` and is more efficient when using multiple estimators out of `CF`, `SECF` and `aSECF` or when using the cross-validation functions.

`est_inds` (optional) A vector of indices for the estimation-only samples. The default when `est_inds` is missing or `NULL` is to perform both estimation of the control variates and evaluation of the integral using all samples. Otherwise, the samples from `est_inds` are used in estimating the control variates and the remainder are used in evaluating the integral. Splitting the indices in this way can be used to reduce bias from adaption and to make computation feasible for very large sample sizes (small `est_inds` is faster), but in general in will increase the variance of the estimator.

`apriori` (optional) A vector containing the subset of parameter indices to use in the polynomial. Typically this argument would only be used if the dimension of the problem is very large or if prior information about parameter dependencies is known. The default is to use all parameters $1:d$ where d is the dimension of the target.

`diagnostics` (optional) A flag for whether to return the necessary outputs for plotting or estimating using the fitted model. The default is `false` since this requires some additional computation when `est_inds` is `NULL`.

Value

A list with the following elements:

- `expectation`: The estimate(s) of the (k) expectation(s).
- `f_true`: (Only if `est_inds` is not `NULL`) The integrands for the evaluation set. This should be the same as integrands[setdiff(1:N,est_inds),].
- `f_hat`: (Only if `est_inds` is not `NULL`) The fitted values for the integrands in the evaluation set. This can be used to help assess the performance of the Gaussian process model.
- `a`: (Only if `diagnostics = TRUE`) The value of a as described in South et al (2020), where predictions are of the form $f_hat = K0 * a + Phi * b$ for heldout K0 and Phi matrices and estimators using heldout samples are of the form $mean(f - f_hat) + b[1]$.
- `b`: (Only if `diagnostics = TRUE`) The value of b as described in South et al (2020), where predictions are of the form $f_hat = K0 * a + Phi * b$ for heldout K0 and Phi matrices and estimators using heldout samples are of the form $mean(f - f_hat) + b[1]$.
- `ksd`: (Only if `diagnostics = TRUE`) An estimated kernel Stein discrepancy based on the fitted model that can be used for diagnostic purposes. See South et al (2020) for further details.
- `bound_const`: (Only if `diagnostics = TRUE` and `est_inds=NULL`) This is such that the absolute error for the estimator should be less than $ksd \times bound_const$.

Warning

Solving the linear system in SECF has $O(N^3 + Q^3)$ complexity where N is the sample size and Q is the number of terms in the polynomial. Standard SECF is therefore not suited to large N. The method aSECF is designed for larger N and details can be found at `aSECF` and in South et al (2020). An alternative would be to use est_inds which has $O(N_0^3 + Q^3)$ complexity in solving the linear system and $O((N - N_0)^2)$ complexity in handling the remaining samples, where N_0 is the length of est_inds. This can be much cheaper for small N_0 but the estimation of the Gaussian process model is only done using N_0 samples and the evaluation of the integral only uses $N - N_0$ samples.

On the choice of σ, the kernel and the Stein order

The kernel in Stein-based kernel methods is $L_x L_y k(x,y)$ where L_x is a first or second order Stein operator in x and $k(x,y)$ is some generic kernel to be specified.

The Stein operators for distribution $p(x)$ are defined as:

- `steinOrder=1`: $L_x g(x) = \nabla_x^T g(x) + \nabla_x \log p(x)^T g(x)$ (see e.g. Oates el al (2017))
- `steinOrder=2`: $L_x g(x) = \Delta_x g(x) + \nabla_x logp(x)^T \nabla_x g(x)$ (see e.g. South el al (2020))

Here ∇_x is the first order derivative wrt x and $\Delta_x = \nabla_x^T \nabla_x$ is the Laplacian operator.

The generic kernels which are implemented in this package are listed below. Note that the input parameter `sigma` defines the kernel parameters σ.

- `"gaussian"`: A Gaussian kernel,

$$k(x,y) = exp(-z(x,y)/\sigma^2)$$

- `"matern"`: A Matern kernel with $\sigma = (\lambda, \nu)$,

$$k(x,y) = bc^\nu z(x,y)^{\nu/2} K_\nu(cz(x,y)^{0.5})$$

  where $b = 2^{1-\nu}(\Gamma(\nu))^{-1}$, $c = (2\nu)^{0.5}\lambda^{-1}$ and $K_\nu(x)$ is the modified Bessel function of the second kind. Note that λ is the length-scale parameter and ν is the smoothness parameter (which defaults to 2.5 for $steinOrder = 1$ and 4.5 for $steinOrder = 2$).
- `"RQ"`: A rational quadratic kernel,

$$k(x,y) = (1 + \sigma^{-2} z(x,y))^{-1}$$

- `"product"`: The product kernel that appears in Oates et al (2017) with $\sigma = (a, b)$

$$k(x,y) = (1 + az(x) + az(y))^{-1} exp(-0.5 b^{-2} z(x,y))$$

- `"prodsim"`: A slightly different product kernel with $\sigma = (a, b)$ (see e.g. https://www.imperial.ac.uk/inference-group/projects/monte-carlo-methods/control-functionals/),

$$k(x,y) = (1 + az(x))^{-1}(1 + az(y))^{-1} exp(-0.5 b^{-2} z(x,y))$$

In the above equations, $z(x) = \sum_j x[j]^2$ and $z(x,y) = \sum_j (x[j] - y[j])^2$. For the last two kernels, the code only has implementations for `steinOrder=1`. Each combination of `steinOrder` and `kernel_function` above is currently hard-coded but it may be possible to extend this to other kernels in future versions using autodiff. The calculations for the first three kernels above are detailed in South et al (2020).

Author(s)

Leah F. South

References

Oates, C. J., Girolami, M. & Chopin, N. (2017). Control functionals for Monte Carlo integration. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 79(3), 695-718.

South, L. F., Karvonen, T., Nemeth, C., Girolami, M. and Oates, C. J. (2020). Semi-Exact Control Functionals From Sard's Method. https://arxiv.org/abs/2002.00033

See Also

See ZVCV for examples and related functions. See SECF_crossval for a function to choose between different kernels for this estimator.

SECF_crossval *Semi-exact control functionals (SECF) with cross-validation*

Description

This function chooses between a list of kernel tuning parameters (sigma_list) or a list of K0 matrices (K0_list) for the semi-exact control functionals method described in South et al (2020). The latter requires calculating and storing kernel matrices using K0_fn but it is more flexible because it can be used to choose the Stein operator order and the kernel function, in addition to its parameters. It is also faster to pre-specify K0_fn. For estimation with fixed kernel parameters, use SECF.

Usage

```
SECF_crossval(
  integrands,
  samples,
  derivatives,
  polyorder = NULL,
  steinOrder = NULL,
  kernel_function = NULL,
  sigma_list = NULL,
  K0_list = NULL,
  est_inds = NULL,
  apriori = NULL,
  folds = NULL,
  diagnostics = FALSE
)
```

Arguments

integrands	An N by k matrix of integrands (evaluations of the function of interest)
samples	An N by d matrix of samples from the target
derivatives	An N by d matrix of derivatives of the log target with respect to the parameters
polyorder	(optional) The order of the polynomial to be used in the parametric component, with a default of 1. We recommend keeping this value low (e.g. only 1-2).
steinOrder	(optional) This is the order of the Stein operator. The default is 1 in the control functionals paper (Oates et al, 2017) and 2 in the semi-exact control functionals paper (South et al, 2020). The following values are currently available: 1 for all kernels and 2 for "gaussian", "matern" and "RQ". See below for further details.
kernel_function	(optional) Choose between "gaussian", "matern", "RQ", "product" or "prodsim". See below for further details.

`sigma_list` (optional between this and `K0_list`) A list of tuning parameters for the specified kernel. This involves a list of single length-scale parameter in "gaussian" and "RQ", a list of vectors containing length-scale and smoothness parameters in "matern" and a list of vectors of the two parameters in "product" and "prodsim". See below for further details. When `sigma_list` is specified and not `K0_list`, the $K0$ matrix is computed twice for each selected tuning parameter.

`K0_list` (optional between this and `sigma_list`) A list of kernel matrices, which can be calculated using `K0_fn`.

`est_inds` (optional) A vector of indices for the estimation-only samples. The default when `est_inds` is missing or `NULL` is to perform both estimation of the control variates and evaluation of the integral using all samples. Otherwise, the samples from `est_inds` are used in estimating the control variates and the remainder are used in evaluating the integral. Splitting the indices in this way can be used to reduce bias from adaption and to make computation feasible for very large sample sizes (small `est_inds` is faster), but in general in will increase the variance of the estimator.

`apriori` (optional) A vector containing the subset of parameter indices to use in the polynomial. Typically this argument would only be used if the dimension of the problem is very large or if prior information about parameter dependencies is known. The default is to use all parameters $1:d$ where d is the dimension of the target.

`folds` (optional) The number of folds for cross-validation. The default is five.

`diagnostics` (optional) A flag for whether to return the necessary outputs for plotting or estimating using the fitted model. The default is `false` since this requires some additional computation when `est_inds` is `NULL`.

Value

A list with the following elements:

- `expectation`: The estimate(s) of the (k) expectation(s).
- `mse`: A matrix of the cross-validation mean square prediction errors. The number of columns is the number of tuning options given and the number of rows is k, the number of integrands of interest.
- `optinds`: The optimal indices from the list for each expectation.
- `f_true`: (Only if `est_inds` is not `NULL`) The integrands for the evaluation set. This should be the same as integrands[setdiff(1:N,est_inds),].
- `f_hat`: (Only if `est_inds` is not `NULL`) The fitted values for the integrands in the evaluation set. This can be used to help assess the performance of the Gaussian process model.
- `a`: (Only if `diagnostics = TRUE`) The value of a as described in South et al (2020), where predictions are of the form $f_hat = K0 * a + Phi * b$ for heldout K0 and Phi matrices and estimators using heldout samples are of the form $mean(f - f_hat) + b[1]$.
- `b`: (Only if `diagnostics = TRUE`) The value of b as described in South et al (2020), where predictions are of the form $f_hat = K0 * a + Phi * b$ for heldout K0 and Phi matrices and estimators using heldout samples are of the form $mean(f - f_hat) + b[1]$.

- ksd: (Only if diagnostics = TRUE) An estimated kernel Stein discrepancy based on the fitted model that can be used for diagnostic purposes. See South et al (2020) for further details.
- bound_const: (Only if diagnostics = TRUE and est_inds=NULL) This is such that the absolute error for the estimator should be less than $ksd \times bound_const$.

Warning

Solving the linear system in SECF has $O(N^3 + Q^3)$ complexity where N is the sample size and Q is the number of terms in the polynomial. Standard SECF is therefore not suited to large N. The method aSECF is designed for larger N and details can be found at aSECF and in South et al (2020). An alternative would be to use est_inds which has $O(N_0^3 + Q^3)$ complexity in solving the linear system and $O((N - N_0)^2)$ complexity in handling the remaining samples, where N_0 is the length of est_inds. This can be much cheaper for large N but the estimation of the Gaussian process model is only done using N_0 samples and the evaluation of the integral only uses $N - N_0$ samples.

On the choice of σ, the kernel and the Stein order

The kernel in Stein-based kernel methods is $L_x L_y k(x, y)$ where L_x is a first or second order Stein operator in x and $k(x, y)$ is some generic kernel to be specified.

The Stein operators for distribution $p(x)$ are defined as:

- steinOrder=1: $L_x g(x) = \nabla_x^T g(x) + \nabla_x \log p(x)^T g(x)$ (see e.g. Oates el al (2017))
- steinOrder=2: $L_x g(x) = \Delta_x g(x) + \nabla_x logp(x)^T \nabla_x g(x)$ (see e.g. South el al (2020))

Here ∇_x is the first order derivative wrt x and $\Delta_x = \nabla_x^T \nabla_x$ is the Laplacian operator.

The generic kernels which are implemented in this package are listed below. Note that the input parameter sigma defines the kernel parameters σ.

- "gaussian": A Gaussian kernel,

$$k(x, y) = exp(-z(x, y)/\sigma^2)$$

- "matern": A Matern kernel with $\sigma = (\lambda, \nu)$,

$$k(x, y) = bc^\nu z(x, y)^{\nu/2} K_\nu(cz(x, y)^{0.5})$$

  where $b = 2^{1-\nu}(\Gamma(\nu))^{-1}$, $c = (2\nu)^{0.5}\lambda^{-1}$ and $K_\nu(x)$ is the modified Bessel function of the second kind. Note that λ is the length-scale parameter and ν is the smoothness parameter (which defaults to 2.5 for $steinOrder = 1$ and 4.5 for $steinOrder = 2$).
- "RQ": A rational quadratic kernel,

$$k(x, y) = (1 + \sigma^{-2} z(x, y))^{-1}$$

- "product": The product kernel that appears in Oates et al (2017) with $\sigma = (a, b)$

$$k(x, y) = (1 + az(x) + az(y))^{-1} exp(-0.5b^{-2} z(x, y))$$

- "prodsim": A slightly different product kernel with $\sigma = (a, b)$ (see e.g. https://www.imperial.ac.uk/inference-group/projects/monte-carlo-methods/control-functionals/),

$$k(x, y) = (1 + az(x))^{-1}(1 + az(y))^{-1} exp(-0.5b^{-2} z(x, y))$$

In the above equations, $z(x) = \sum_j x[j]^2$ and $z(x,y) = \sum_j (x[j] - y[j])^2$. For the last two kernels, the code only has implementations for `steinOrder=1`. Each combination of `steinOrder` and `kernel_function` above is currently hard-coded but it may be possible to extend this to other kernels in future versions using autodiff. The calculations for the first three kernels above are detailed in South et al (2020).

Author(s)

Leah F. South

References

Oates, C. J., Girolami, M. & Chopin, N. (2017). Control functionals for Monte Carlo integration. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 79(3), 695-718.

South, L. F., Karvonen, T., Nemeth, C., Girolami, M. and Oates, C. J. (2020). Semi-Exact Control Functionals From Sard's Method. https://arxiv.org/abs/2002.00033

See Also

See ZVCV for examples and related functions. See `SECF` for a function to perform semi-exact control functionals with fixed kernel specifications.

`squareNorm` *Squared norm matrix calculation*

Description

This function gets the matrix of square norms which is needed for all kernels. Calculating this can help to save time if you are also interested in calculating the median heuristic, handling multiple tuning parameters or trying other kernels.

Usage

```
squareNorm(samples, nystrom_inds = NULL)
```

Arguments

`samples`	An N by d matrix of samples from the target
`nystrom_inds`	The (optional) sample indices to be used in the Nystrom approximation (for when using aSECF).

Value

An N by N matrix of squared norms between samples (or N by m where m is the length of `nystrom_inds`).

Author(s)

Leah F. South

See Also

See medianTune and K0_fn for functions which use this.

VDP *Example of estimation using SMC*

Description

This example illustrates how ZV-CV can be used for post-processing of results from likelihood-annealing SMC. In particular, we use ZV-CV to estimate posterior expectations and the evidence for a single SMC run of this example based on the Van der Pol oscillatory differential equations (Van der Pol, 1926). Further details about this example and applications to ZV-CV can be found in Oates et al. (2017) and South et al. (2019).

Given that the focus of this R package is on ZV-CV, we assume that samples have already been obtained from SMC and put into the correct format. One could use the R package RcppSMC or implement their own sampler in order to obtain results like this. The key is to make sure the derivatives of the log likelihood and log prior are stored, along with the inverse temperatures.

Usage

```
data(VDP)
```

Format

A list containing the following :

N The size of the SMC population

rho The tolerance for the new temperatures, which are selected so that the CESS at each temperature is $\rho * N$ where N is the population size.

temperatures A vector of length T of inverse power posterior temperatures

samples An N by d by T matrix of samples from the T power posteriors, where d is the dimension of the target distribution. The samples are transformed to be on the log scale and all derivatives are with respect to log samples.

loglike An N by T matrix of log likelihood values corresponding to samples

logprior An N by T matrix of log prior values corresponding to samples

der_loglike An N by d by T matrix of the derivatives of the log likelihood with respect to the parameters, with parameter values corresponding to samples

der_logprior An N by d by T matrix of the derivatives of the log prior with respect to the parameters, with parameter values corresponding to samples

References

Oates, C. J., Girolami, M. & Chopin, N. (2017). Control functionals for Monte Carlo integration. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 79(3), 695-718.

South, L. F., Oates, C. J., Mira, A., & Drovandi, C. (2019). Regularised zero-variance control variates for high-dimensional variance reduction.

Van der Pol, B. (1926). On relaxation-oscillations. The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science, 2(11), 978-992.

See Also

See ZVCV for more package details.

Examples

```
set.seed(1)

# Load the SMC results
data(VDP)

# Set up the list of control variates to choose from
options <- list()
# Vanilla Monte Carlo
options[[1]] <- list(polyorder = 0)
# Standard ZV-CV with polynomial order selected through cross-validation
options[[2]] <- list(polyorder = Inf, regul_reg = FALSE)

##############################
# Posterior expectation - The true expectation is 0.9852 to 4 decimal places
##############################

# Note the exp() because samples and derivatives were stored on the log scale
# but we are interested in the expectation on the original scale
posterior <- zvcv(exp(VDP$samples[,,8]), VDP$samples[,,8],
VDP$der_loglike[,,8] + VDP$der_logprior[,,8], options = options)
posterior$expectation # The posterior expectation estimate
posterior$polyorder # The selected polynomial order

##############################
# Evidence estimation - The true logged evidence is 10.36 to 2 decimal places
##############################

# Getting additional temperatures based on maintaing a CESS of 0.91N rather than 0.9N.
# The value 0.91 is used for speed but South et al. (2019) use 0.99.
temp <- Expand_Temperatures(VDP$temperatures, VDP$loglike, 0.91)
VDP$temperatures_new <- temp$temperatures_all # the new temperatures
VDP$most_recent <- temp$relevant_samples # the samples associated with the new temperatures

n_sigma <- 3 # For speed, South et al. (2019) uses 15
sigma_list <- as.list( 10^(0.5*seq(-3,4,length.out=n_sigma)) )

# Evidence estimation using the SMC identity
```

```
Z_SMC <- evidence_SMC(VDP$samples, VDP$loglike, VDP$der_loglike, VDP$der_logprior,
VDP$temperatures, VDP$temperatures_new, VDP$most_recent, options = options)
Z_SMC$log_evidence

# Evidence estimation using the SMC identity
Z_SMC_CF <- evidence_SMC_CF(VDP$samples, VDP$loglike, VDP$der_loglike, VDP$der_logprior,
VDP$temperatures, VDP$temperatures_new, VDP$most_recent, steinOrder = 2,
kernel_function = "gaussian", sigma_list = sigma_list, folds = 2)
Z_SMC_CF$log_evidence

# Evidence estimation using the CTI identity
Z_CTI <- evidence_CTI(VDP$samples, VDP$loglike, VDP$der_loglike, VDP$der_logprior,
VDP$temperatures, VDP$temperatures_new, VDP$most_recent, options = options)
Z_CTI$log_evidence_PS2

# Evidence estimation using the CTI identity
Z_CTI_CF <- evidence_CTI_CF(VDP$samples, VDP$loglike, VDP$der_loglike, VDP$der_logprior,
VDP$temperatures, VDP$temperatures_new, VDP$most_recent, steinOrder = 2,
kernel_function = "gaussian", sigma_list = sigma_list, folds = 2)
Z_CTI_CF$log_evidence_PS2
```

---

zvcv *ZV-CV for general expectations*

Description

The function zvcv is used to perform (regularised) ZV-CV given a set of samples, derivatives and function evaluations.

Usage

```
zvcv(
  integrand,
  samples,
  derivatives,
  log_weights,
  integrand_logged = FALSE,
  est_inds,
 options = list(polyorder = 2, regul_reg = TRUE, alpha_elnet = 1, nfolds = 10, apriori
    = (1:NCOL(samples)), intercept = TRUE, polyorder_max = Inf),
  folds = 5
)
```

Arguments

integrand	An N by k matrix of integrands (evaluations of the functions of interest)
samples	An N by d matrix of samples from the target
derivatives	An N by d matrix of derivatives of the log target with respect to the parameters

`log_weights` (optional) A vector of length N containing log weights of the samples. The default is equal weights.

`integrand_logged`
(optional) Sometimes it is better to input the integrand on the logged scale for stability. If the actual integrand is the exponential of `integrand`, then `integrand_logged = TRUE`. Otherwise, the default of `integrand_logged = FALSE` should be used.

`est_inds` (optional) A vector of indices for the estimation-only samples. The default when `est_inds` is missing or `NULL` is to perform both estimation of the control variates and evaluation of the integral using all samples. Otherwise, the samples from `est_inds` are used in estimating the control variates and the remainder are used in evaluating the integral. Splitting the indices in this way can be used to reduce bias from adaption and to make computation feasible for very large sample sizes (small `est_inds` is faster), but in general in will increase the variance of the estimator.

`options` A list of control variate specifications. This can be a single list containing the elements below (the defaults are used for elements which are not specified). Alternatively, it can be a list of lists containing any or all of the elements below. Where the latter is used, the function `zvcv` automatically selects the best performing option based on cross-validation.

- `polyorder`: The order of the polynomial, with a default of 2. A value of `Inf` will get the cross-validation method to choose between orders.
- `regul_reg`: A flag for whether regularised regression is to be used. The default is TRUE, i.e. regularised regression is used.
- `alpha_elnet`: The alpha parameter for elastic net. The default is 1, which correponds to LASSO. A value of 0 would correspond to ridge regression.
- `nfolds`: The number of folds used in cross-validation to select lambda for LASSO or elastic net. The default is 10.
- `apriori`: A vector containing the subset of parameter indices to use in the polynomial. Typically this argument would only be used if the dimension of the problem is very large or if prior information about parameter dependencies is known. The default is to use all parameters $1 : d$ where d is the dimension of the target. In `zvcv`, this is equivalent to using only the relevant columns in `samples` and `derivatives`).
- `intercept`: A flag for whether the intercept should be estimated or fixed to the empirical mean of the integrand in the estimation set. The default is to include an intercept (`intercept = TRUE`) as this tends to lead to better variance reductions. Note that an `intercept = TRUE` flag may be changed to `intercept = FALSE` within the function if `integrand_logged = TRUE` and a `NaN` is encountered. See South et al. (2018) for further details.
- `polyorder_max`: The maximum allowable polynomial order. This may be used to prevent memory issues in the case that the polynomial order is selected automatically. A default maximum polynomial order based on the regression design matrix having no more than ten million elements will be selected if the `polyorder` is infinite and in this case a warning will be given. Recall that setting your default R settings to `options(warn=1)` will ensure that you receive these warnings in real time. Optimal polynomial order selection may go to at most this maximum value, or it may stop earlier.

`folds` The number of folds used in k-fold cross-validation for selecting the optimal control variate. Depending on the `options`, this may include selection of the optimal polynomial order, regression type and subset of parameters in the polynomial. The default is five.

Value

A list is returned, containing the following components:

- `expectation`: The estimates of the expectations.
- `num_select`: The number of non-zero coefficients in the polynomial.
- `mse`: The mean square error for the evaluation set.
- `coefs`: The estimated coefficients for the regression (columns are for the different integrands).
- `integrand_logged`: The `integrand_logged` input stored for reference.
- `est_inds`: The `est_inds` input stored for reference.
- `polyorder`: The `polyorder` value used in the final estimate.
- `regul_reg`: The `regul_reg` flag used in the final estimate.
- `alpha_elnet`: The `alpha_elnet` value used in the final estimate.
- `nfolds`: The `nfolds` value used in the final estimate.
- `apriori`: The `apriori` vector used in the final estimate.
- `intercept`: The `intercept` flag used in the final estimate.
- `polyorder_max`: The `polyorder_max` flag used in the final estimate, if multiple `options` are specified.

Author(s)

Leah F. South

References

Mira, A., Solgi, R., & Imparato, D. (2013). Zero variance Markov chain Monte Carlo for Bayesian estimators. Statistics and Computing, 23(5), 653-662.

South, L. F., Oates, C. J., Mira, A., & Drovandi, C. (2019). Regularised zero variance control variates for high-dimensional variance reduction. https://arxiv.org/abs/1811.05073

See Also

See ZVCV and `VDP` for additional examples. See `evidence` for functions which use `zvcv` to estimate the normalising constant of the posterior.

Examples

```
# An example where ZV-CV can result in zero-variance estimators

# Estimating some expectations when theta is bivariate normally distributed with:
mymean <- c(-1.5,1.5)
mycov <- matrix(c(1,0.5,0.5,2),nrow=2)

# Perfect draws from the target distribution (could be replaced with
# approximate draws from e.g. MCMC or SMC)
N <- 30
require(mvtnorm)
set.seed(1)
samples <- rmvnorm(N, mean = mymean, sigma = mycov)
# derivatives of Gaussian wrt x
derivatives <- t( apply(samples,1,function(x) -solve(mycov)%*%(x - mymean)) )

# The integrands are the marginal posterior means of theta, the variances and the
# covariance (true values are c(-1.5,1.5,1,2,0.5))
integrand <- cbind(samples[,1],samples[,2],(samples[,1] - mymean[1])^2,
    (samples[,2] - mymean[2])^2, (samples[,1] - mymean[1])*(samples[,2] - mymean[2]))

# Estimates without ZV-CV (i.e. vanilla Monte Carlo integration)
# Vanilla Monte Carlo
sprintf("%.15f",colMeans(integrand))

# ZV-CV with fixed specifications
# For this example, polyorder = 1 with OLS is exact for the first two integrands and
# polyorder = 2 with OLS is exact for the last three integrands

# ZV-CV with 2nd order polynomial, OLS and a polynomial based on only x_1.
# For diagonal mycov, this would be exact for the first and third expectations.
sprintf("%.15f",zvcv(integrand, samples, derivatives,
    options = list(polyorder = 2, regul_reg = FALSE, apriori = 1))$expectation)

# ZV-CV with 1st order polynomial and OLS (exact for the first two integrands)
sprintf("%.15f",zvcv(integrand, samples, derivatives,
    options = list(polyorder = 1, regul_reg = FALSE))$expectation)

# ZV-CV with 2nd order polynomial and OLS (exact for all)
sprintf("%.15f",zvcv(integrand, samples, derivatives,
    options = list(polyorder = 2, regul_reg = FALSE))$expectation)

# ZV-CV with cross validation
myopts <- list(list(polyorder = Inf, regul_reg = FALSE),list(polyorder = Inf, nfolds = 4))
temp <- zvcv(integrand,samples,derivatives,options = myopts, folds = 2)
temp$polyorder # The chosen control variate order
temp$regul_reg # Flag for if the chosen control variate uses regularisation
# Cross-val ZV-CV to choose the polynomial order and whether to perform OLS or LASSO
sprintf("%.15f",temp$expectation) # Estimate based on the chosen control variate
```

ZVCV_package *Zero-Variance Control Variates*

Description

This package can be used to perform post-hoc variance reduction of Monte Carlo estimators when the derivatives of the log target are available. The main functionality is available through the following functions. All of these use a set of N d-dimensional samples along with the associated derivatives of the log target. You can evaluate posterior expectations of k functions.

- zvcv: For estimating expectations using (regularised) zero-variance control variates (ZV-CV, Mira et al, 2013; South et al, 2018). This function can also be used to choose between various versions of ZV-CV using cross-validation.
- CF: For estimating expectations using control functionals (CF, Oates et al, 2017).
- SECF: For estimating expectations using semi-exact control functionals (SECF, South et al, 2020).
- aSECF: For estimating expectations using approximate semi-exact control functionals (aSECF, South et al, 2020).
- CF_crossval: CF with cross-validation tuning.
- SECF_crossval: SECF with cross-validation tuning.
- aSECF_crossval: aSECF with cross-validation tuning.

ZV-CV is exact for polynomials of order at most polyorder under Gaussian targets and is fast for large N (although setting a limit on polyorder through polyorder_max is recommended for large N). CF is a non-parametric approach that offers better than the standard Monte Carlo convergence rates. SECF has both a parametric and a non-parametric component and it offers the advantages of both for an additional computational cost. The cost of SECF is reduced in aSECF using nystrom approximations and conjugate gradient.

Helper functions

- getX: Calculates the design matrix for ZV-CV (without the column of 1's for the intercept)
- medianTune: Calculates the median heuristic for use in e.g. the Gaussian, Matern and rational quadratic kernels. Using the median heuristic is an alternative to cross-validation.
- K0_fn: Calculates the K_0 matrix. The output of this function can be used as an argument to CF, CF_crossval, SECF, SECF_crossval, aSECF and aSECF_crossval. The kernel matrix is automatically computed in all of the above methods, but it is faster to calculate in advance when using more than one of the above functions and when using any of the crossval functions.
- Phi_fn: Calculates the Phi matrix for SECF and aSECF (similar to getX but with different arguments and it includes the column of 1's)
- squareNorm: Gets the matrix of square norms which is needed for all kernels. Calculating this can help to save time if you are also interested in calculating the median heuristic, handling multiple tuning parameters or trying other kernels.

- nearPD: Finds the nearest symmetric positive definite matrix to the given matrix, for handling numerical issues.
- logsumexp: Performs stable computation of the log sum of exponential (useful when handling the sum of weights)

Evidence estimation

The following functions are used to estimate the evidence (the normalisiing constant of the posterior) as described in South et al (2018). They are relevant when sequential Monte Carlo with an annealing schedule has been used to collect the samples, and therefore are not of interest to those who are interested in variance reduction based on vanilla MCMC.

- evidence_CTI and evidence_CTI_CF: Functions to estimate the evidence using thermodynamic integration (TI) with ZV-CV and CF, respectively
- evidence_SMC and evidence_SMC_CF: Function to estimate the evidence using the SMC evidence identity with ZV-CV and CF, respectively.

The function Expand_Temperatures can be used to adjust the temperature schedule so that it is more (or less) strict than the original schedule of T temperatures.

Author(s)

Leah F. South

References

Mira, A., Solgi, R., & Imparato, D. (2013). Zero variance Markov chain Monte Carlo for Bayesian estimators. Statistics and Computing, 23(5), 653-662.

South, L. F., Karvonen, T., Nemeth, C., Girolami, M. and Oates, C. J. (2020). Semi-Exact Control Functionals From Sard's Method. https://arxiv.org/abs/2002.00033

South, L. F., Oates, C. J., Mira, A., & Drovandi, C. (2018). Regularised zero-variance control variates for high-dimensional variance reduction. https://arxiv.org/abs/1811.05073

See Also

Useful links:

- Report bugs at https://github.com/LeahPrice/ZVCV/issues

Examples

```
# A real data example using ZV-CV is available at \link{VDP}.
# This involves estimating posterior expectations and the evidence from SMC samples.

# The remainder of this section is duplicating (albeit with a different random
# seed) Figure 2a of South et al. (2020).

N_repeats <- 2 # For speed, the actual code uses 100
N_all <- 25 # For speed, the actual code uses c(10,25,50,100,250,500,1000)
sigma_list <- list(10^(-1.5),10^(-1),10^(-0.5),1,10^(0.5),10)
```

```
nfolds <- 4 # For speed, the actual code uses 10
folds <- 2 # For speed, the actual code uses 5
d <- 4

integrand_fn <- function(x){
  return (1 + x[,2] + 0.1*x[,1]*x[,2]*x[,3] + sin(x[,1])*exp(-(x[,2]*x[,3])^2))
}

results <- data.frame()
for (N in N_all){

 # identify the largest polynomial order that can be fit without regularisation for auto ZV-CV
  max_r <- 0
  while (choose(d + max_r + 1,d)<((folds-1)/folds*N)){
   max_r <- max_r + 1
  }

  MC <- ZV1 <- ZV2 <- ZVchoose <- rep(NaN,N_repeats)
  CF <- SECF1 <- aSECF1 <- SECF2 <- aSECF2 <- rep(NaN,N_repeats)
  CF_medHeur <- SECF1_medHeur <- aSECF1_medHeur <- rep(NaN,N_repeats)
  SECF2_medHeur <- aSECF2_medHeur <- rep(NaN,N_repeats)
  for (i in 1:N_repeats){
    x <- matrix(rnorm(N*d),ncol=d)
    u <- -x
    f <- integrand_fn(x)

    MC[i] <- mean(f)
    ZV1[i] <- zvcv(f,x,u,options=list(polyorder=1,regul_reg=FALSE))$expectation
   # Checking if the sample size is large enough to accommodation a second order polynomial
    if (N > choose(d+2,d)){
      ZV2[i] <- zvcv(f,x,u,options=list(polyorder=2,regul_reg=FALSE))$expectation
    }
    myopts <- list(list(polyorder=Inf,regul_reg=FALSE,polyorder_max=max_r),
        list(polyorder=Inf,nfolds=nfolds))
    ZVchoose[i] <- zvcv(f,x,u,options=myopts,folds = folds)$expectation

    # Calculating the kernel matrix in advance for CF and SECF
    K0_list <- list()
    for (j in 1:length(sigma_list)){
      K0_list[[j]] <- K0_fn(x,u,sigma_list[[j]],steinOrder=2,kernel_function="RQ")
    }

    CF[i] <- CF_crossval(f,x,u,K0_list=K0_list,folds = folds)$expectation
    SECF1[i] <- SECF_crossval(f,x,u,K0_list=K0_list,folds = folds)$expectation
    aSECF1[i] <- aSECF_crossval(f,x,u,steinOrder=2,kernel_function="RQ",
        sigma_list=sigma_list,reltol=1e-05,folds = folds)$expectation
    if (max_r>=2){
    SECF2[i] <- SECF_crossval(f,x,u,polyorder=2,K0_list=K0_list,folds = folds)$expectation
      aSECF2[i] <- aSECF_crossval(f,x,u,polyorder=2,steinOrder=2,kernel_function="RQ",
          sigma_list=sigma_list,reltol=1e-05,folds = folds)$expectation
    }

    medHeur <- medianTune(x)
```

```
    K0_medHeur <- K0_fn(x,u,medHeur,steinOrder=2,kernel_function="RQ")
    CF_medHeur[i] <- CF(f,x,u,K0=K0_medHeur)$expectation
    SECF1_medHeur[i] <- SECF(f,x,u,K0=K0_medHeur)$expectation
    aSECF1_medHeur[i] <- aSECF(f,x,u,steinOrder=2,kernel_function="RQ",
          sigma=medHeur,reltol=1e-05)$expectation
    if (max_r>=2){
      SECF2_medHeur[i] <- SECF(f,x,u,polyorder=2,K0=K0_medHeur)$expectation
      aSECF2_medHeur[i] <- aSECF(f,x,u,polyorder=2,steinOrder=2,kernel_function="RQ",
          sigma=medHeur,reltol=1e-05)$expectation
    }

    # print(sprintf("--%d",i))
  }
  # Adding the results to a data frame
  MSE_crude <- mean((MC - 1)^2)
  results <- rbind(results,data.frame(N=N, order = "1 or NA",
      efficiency = 1, type = "MC"))
  results <- rbind(results,data.frame(N=N, order = "1 or NA",
      efficiency = MSE_crude/mean((ZV1 - 1)^2), type = "ZV"))
  results <- rbind(results,data.frame(N=N, order = "2",
      efficiency = MSE_crude/mean((ZV2 - 1)^2), type = "ZV"))
  results <- rbind(results,data.frame(N=N, order = "1 or NA",
      efficiency = MSE_crude/mean((ZVchoose - 1)^2), type = "ZVchoose"))
  results <- rbind(results,data.frame(N=N, order = "1 or NA",
      efficiency = MSE_crude/mean((CF - 1)^2), type = "CF"))
  results <- rbind(results,data.frame(N=N, order = "1 or NA",
      efficiency = MSE_crude/mean((SECF1 - 1)^2), type = "SECF"))
  results <- rbind(results,data.frame(N=N, order = "1 or NA",
      efficiency = MSE_crude/mean((aSECF1 - 1)^2), type = "aSECF"))
  if (((folds-1)/folds*N) > choose(d+2,d)){
    results <- rbind(results,data.frame(N=N, order = "2",
      efficiency = MSE_crude/mean((SECF2 - 1)^2), type = "SECF"))
    results <- rbind(results,data.frame(N=N, order = "2",
      efficiency = MSE_crude/mean((aSECF2 - 1)^2), type = "aSECF"))
  }

  results <- rbind(results,data.frame(N=N, order = "1 or NA",
      efficiency = MSE_crude/mean((CF_medHeur - 1)^2), type = "CF_medHeur"))
  results <- rbind(results,data.frame(N=N, order = "1 or NA",
      efficiency = MSE_crude/mean((SECF1_medHeur - 1)^2), type = "SECF_medHeur"))
  results <- rbind(results,data.frame(N=N, order = "1 or NA",
      efficiency = MSE_crude/mean((aSECF1_medHeur - 1)^2), type = "aSECF_medHeur"))
  if (((folds-1)/folds*N) > choose(d+2,d)){
    results <- rbind(results,data.frame(N=N, order = "2",
      efficiency = MSE_crude/mean((SECF2_medHeur - 1)^2), type = "SECF_medHeur"))
    results <- rbind(results,data.frame(N=N, order = "2",
      efficiency = MSE_crude/mean((aSECF2_medHeur - 1)^2), type = "aSECF_medHeur"))
  }
  # print(N)
}


## Not run:
```

```
# Plotting results where cross-validation is used for kernel methods
require(ggplot2)
require(ggthemes)
a <- ggplot(data=subset(results,!(type %in% c("CF_medHeur","SECF_medHeur",
  "aSECF_medHeur","SECF_medHeur","aSECF_medHeur"))),
  aes(x=N,y=efficiency,col=type,linetype=order)) + scale_color_pander() +
  ggtitle("") + geom_line(size=1.5) + scale_x_log10() + scale_y_log10() +
  annotation_logticks(base=10) + labs(x="N",y="Efficiency",color="Method",
  linetype="Polynomial Order") + theme_minimal(base_size = 15) +
  theme(legend.key.size = unit(0.5, "cm"),legend.key.width =  unit(1, "cm")) +
  guides(linetype = guide_legend(override.aes = list(size=1),title.position = "top"),
  color = guide_legend(override.aes = list(size=1),title.position = "top"))
print(a)


# Plotting results where the median heuristic is used for kernel methods
b <- ggplot(data=subset(results,!(type %in% c("CF","SECF","aSECF","SECF","aSECF"))),
            aes(x=N,y=efficiency,col=type,linetype=order)) + scale_color_pander() +
  ggtitle("") + geom_line(size=1.5) + scale_x_log10() + scale_y_log10() +
  annotation_logticks(base=10) + labs(x="N",y="Efficiency",color="Method",
  linetype="Polynomial Order") + theme_minimal(base_size = 15) +
  theme(legend.key.size = unit(0.5, "cm"),legend.key.width =  unit(1, "cm")) +
  guides(linetype = guide_legend(override.aes = list(size=1),title.position = "top"),
  color = guide_legend(override.aes = list(size=1),title.position = "top"))
print(b)

## End(Not run)
```

Index

aSECF, 2, *3*, *6*, *10*, *22*, *27, 28*, *32*, *40*
aSECF_crossval, *2*, *5*, 6, *9*, *22*, *40*

CF, *3*, 9, *10*, *12*, *15*, *22*, *27*, *40*
CF_crossval, *9*, *12*, 12, *22*, *40*

evidence, 15, *21*, *38*
evidence_CTI, *41*
evidence_CTI *(*evidence*)*, 15
evidence_CTI_CF, *41*
evidence_CTI_CF *(*evidence*)*, 15
evidence_SMC, *41*
evidence_SMC *(*evidence*)*, 15
evidence_SMC_CF, *41*
evidence_SMC_CF *(*evidence*)*, 15
Expand_Temperatures, *17*, *20*, 20, *41*

getX, 21, *40*

K0_fn, *3*, *6*, *10*, *12, 13*, 22, *25*, *27*, *30, 31*, *34*, *40*

logsumexp, 24, *41*

medianTune, 24, *25*, *34*, *40*

nearPD, 25, *41*

Phi_fn, *21*, 26, *40*

SECF, *2, 3*, *10*, *22*, *27*, 27, *30*, *33*, *40*
SECF_crossval, *22*, *27*, *30*, 30, *40*
squareNorm, *4*, *7*, *22*, *24*, 33, *40*

VDP, *20, 21*, 34, *38*

ZVCV, *5*, *9*, *12*, *15*, *20, 21*, *24*, *30*, *33*, *35*, *38*
ZVCV *(*ZVCV_package*)*, 40
zvcv, 36, *40*
ZVCV-package *(*ZVCV_package*)*, 40
ZVCV_package, 40