# Package ‘noisyCE2’

October 13, 2022

**Type** Package

**Title** Cross-Entropy Optimisation of Noisy Functions

**Version** 1.1.0

**Author** Flavio Santi [cre, aut] (<https://orcid.org/0000-0002-2014-1981>)

**Maintainer** Flavio Santi <flavio.santi@univr.it>

**URL** https://www.flaviosanti.it/software/noisyCE2

**BugReports** https://github.com/f-santi/noisyCE2/issues

**Description** Cross-Entropy optimisation of unconstrained deterministic and noisy functions illustrated in Rubinstein and Kroese (2004, ISBN: 978-1-4419-1940-3) through a highly flexible and customisable function which allows user to define custom variable domains, sampling distributions, updating and smoothing rules, and stopping criteria. Several built-in methods and settings make the package very easy-to-use under standard optimisation problems.

**Imports** magrittr

**Suggests** coda, testthat

**License** GPL (>= 2)

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.1

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2020-11-09 13:10:10 UTC

## R topics documented:

---

noisyCE2-package *Cross-Entropy Optimisation of Noisy Functions*

---

## Description

The package `noisyCE2` implements the cross-entropy algorithm (Rubinstein and Kroese, 2004) for the optimisation of unconstrained deterministic and noisy functions through a highly flexible and customisable function which allows user to define custom variable domains, sampling distributions, updating and smoothing rules, and stopping criteria. Several built-in methods and settings make the package very easy-to-use under standard optimisation problems.

## Details

The package permits a noisy function to be maximised by means of the cross-entropy algorithm. Formally, problems in the form

$$\max_{x \in \Theta} \mathbf{E}(f(x))$$

are tackled for a noisy function $f \colon \Theta \subseteq \mathbf{R}^m \to \mathbf{R}$.

## Author(s)

**Maintainer**: Flavio Santi <flavio.santi@univr.it> (ORCID)

## References

Bee M., G. Espa, D. Giuliani, F. Santi (2017) "A cross-entropy approach to the estimation of generalised linear multilevel models", *Journal of Computational and Graphical Statistics*, **26** (3), pp. 695-708. https://doi.org/10.1080/10618600.2016.1278003

Rubinstein, R. Y., and Kroese, D. P. (2004), *The Cross-Entropy Method*, Springer, New York. ISBN: 978-1-4419-1940-3

## See Also

Useful links:

- https://www.flaviosanti.it/software/noisyCE2
- Report bugs at https://github.com/f-santi/noisyCE2/issues

## Examples

```
# EXAMPLE 1
# The negative 4-dimensional paraboloid can be maximised as follows:
negparaboloid <- function(x) { -sum((x - (1:4))^2) }
sol <- noisyCE2(negparaboloid, domain = rep('real', 4))

# EXAMPLE 2
# The 10-dimensional Rosenbrock's function can be minimised as follows:
rosenbrock <- function(x) {
  sum(100 * (tail(x, -1) - head(x, -1)^2)^2 + (head(x, -1) - 1)^2)
}

newvar <- type_real(
  init = c(0, 2),
  smooth = list(
    quote(smooth_lin(x, xt, 1)),
    quote(smooth_dec(x, xt, 0.7, 5))
  )
)

sol <- noisyCE2(
  rosenbrock, domain = rep(list(newvar), 10),
  maximise = FALSE, N = 2000, maxiter = 10000
)

# EXAMPLE 3
# The negative 4-dimensional paraboloid with additive Gaussian noise can be
# maximised as follows:
noisyparaboloid <- function(x) { -sum((x - (1:4))^2) + rnorm(1) }
sol <- noisyCE2(noisyparaboloid, domain = rep('real', 4), stoprule = geweke(x))
# where the stopping criterion based on the Geweke's test has been adopted
# according to Bee et al. (2017).
```

---

`geweke` *Geweke's test stopping rule*

---

## Description

`geweke` tests the convergence of `x` through the Geweke's test.

## Usage

```
geweke(x, frac1 = 0.3, frac2 = 0.4, pvalue = 0.05)
```

### Arguments

| | |
|---|---|
| x | numeric vector of last $\gamma_n$ values, as selected by the function passed to noisyCE2() through the argument stopwindow. |
| frac1, frac2 | fraction arguments of the Geweke's test according to coda::geweke.diag(). |
| pvalue | threshold of the $p$-value which triggers the stop of the algorithm. |

### Value

A numeric indicating whether the algorithm has converged:

| | |
|---|---|
| 0 | the algorithm has converged. |
| 1 | the algorithm has not converged. |

### See Also

Other stopping rules: ts_change()

---

noisyCE2 *Cross-Entropy Optimisation of Noisy Functions*

---

### Description

Unconstraint optimisation of noisy functions through the cross-entropy algorithm.

### Usage

```
noisyCE2(
  f,
  domain,
  ...,
  rho = 0.05,
  N = 1000,
  smooth = NULL,
  stopwindow = tail(gam, (n > 20) * n/2),
  stoprule = ts_change(x),
  maxiter = 1000,
  maximise = TRUE,
  verbose = "v"
)

## S3 method for class 'noisyCE2'
print(x, ...)

## S3 method for class 'noisyCE2'
summary(object, ...)
```

```
## S3 method for class 'noisyCE2'
plot(x, what = c("x", "gam", "param"), start = NULL, end = NULL, ...)

## S3 method for class 'noisyCE2'
coef(object, ...)
```

### Arguments

| | |
|---|---|
| `f` | objective function which takes the vector of optimisation variables as first argument. |
| `domain` | a `list` (or other coercible objects) where each component specifies the domain of each variable of the objective function `f`. The components of the list may be either objects of `typevar` class (see type_variable) or strings identifying one of type_variable functions (for example `"real"` for function `type_real()`). See § Examples. |
| `...` | other arguments to be passed to `f` or to other methods (for `print` and `plot`). |
| `rho` | parameter $\rho$ of the Cross-Entropy algorithm. This argument may be passed either as a numeric value in $(0, 1)$ or as an unevaluated expression which may include the number of current iteration `n`, or the argument `N`. |
| `N` | parameter $N$ of the Cross-Entropy algorithm. This argument may be passed either as a positive integer or as an unevaluated expression which may include the number of current iteration `n`. |
| `smooth` | list of unevaluated expressions to be used as smoothing rules for the parameters of the sampling probability distributions of **all variables**. If not `NULL`, all default or set smoothing rules of all variables will be overwritten. See type_variable for details and examples. |
| `stopwindow` | unevaluated expression returning the object to be passed to the stopping rule. Symbol `gam` permits the time series $\gamma_t$ to be used (as a `numeric` vector). |
| `stoprule` | stopping rule passed as an unevaluated expression including `x` as the object returned by evaluation of argument `stopwindow`. The algorithm is stopped when zero is returned by the evaluation of `stoprule`. If returned object has attribute `mess`, this is used as a message. Currently, built-in stopping rules are `ts_change()` and `geweke()`, others may be defined by user. |
| `maxiter` | maximum number of iteration. When it is reached, algorithm is stopped whether or not the stopping criterion is satisfied. If the maximum number of iteration is reached, the `code` and the `message` components of `noisyCE` object are overwritten. |
| `maximise` | if `TRUE` (default) `f` is maximised, otherwise a minimisation of `f` is performed. |
| `verbose` | algorithm verbosity (values `v`, `vv` and `vvv` are admitted). |
| `x, object` | object of class `noisyCE2`, as returned by `noisyCE2`. |
| `what` | type of plot should be drawn. If `what = "x"` (default), values of the variables are plotted as time series; if `what = "gam"`, time series of statistics $\gamma$ is plotted; if `what = "param"`, time series of parameters of the sampling distributions are plotted. |
| `start, end` | first and last value to be plotted. If `NULL`, all values are plotted. |

## Value

An object of class `noisyCE2` structured as a list with the following components:

- `f` argument `f`.
- `fobj` objective function `f` where possible arguments passed through argument `...` have been substituted. Thus, the value of the objective function maximised by `noisyCE` in `x0` can be computed as `fobj(x0)`. If a minimisation has been performed, `fobj` returns `f` with sign inverted.
- `xopt` `numeric` vector with solution.
- `hxopt` matrix of `niter` rows and `length(xopt)` columns with values of variables generated by the optimisation algorithm.
- `param` `list` of `length(xopt)` components where time series of parameters (vectors $v_t$) are stored for each variable as `data.frame` objects with `niter+1` rows (the first rows are the starting values set through function `noisyCEcontrol`).
- `gam` vector of values $\gamma_t$.
- `niter` number of iterations.
- `code` convergence code of the algorithm. Value `0` means that algorithm has converged; other values are defined according to the stopping rule.
- `convMess` textual message associated to the convergence code (if any).
- `compTimes` named vector computation times of each phase.

## Methods (by generic)

- `print`: display synthetic information about a `noisyCE2` object
- `summary`: display summary information about a `noisyCE2` object
- `plot`: plot various components of a `noisyCE2` object
- `coef`: get the solution of the optimisation

## Examples

```
library(magrittr)
# Optimisation of the 4-dimensional function:
# f(x1,x2,x3,x4)=-(x1-1)^2-(x2-2)^2-(x3-3)^2-(x4-4)^2
sol <- noisyCE2(function(x) -sum((x - (1:4))^2), domain = rep('real', 4))
# Representation of the convergence process:
plot(sol, what = 'x')
plot(sol, what = 'gam')
```

---

`smooth_dec` *Decreasing first-order smoothing rule*

---

## Description

Decreasing smoothing rule

$$x_{t+1} := a_t\, x_t + (1 - a_t)\, x_{t-1}$$

where

$$a_t := b\left(1 - \left(1 - \frac{1}{t}\right)^q\right)$$

for some $0.7 \leq b \leq 1$ and some $5 \leq q \leq 10$.

## Usage

```
smooth_dec(x, xt, b, qu)
```

## Arguments

| | |
|---|---|
| `x` | `numeric` value of the last value of the parameter. |
| `xt` | `numeric` vector of past values of the parameter (time series). |
| `b` | smoothing parameter $b$. |
| `qu` | smoothing parameter $q$. |

## Value

A `numeric` vector of updated parameters.

## See Also

Other smoothing rules: `smooth_lin()`

---

`smooth_lin` *Linear first-order smoothing rule*

---

## Description

Linear smoothing rule

$$x_{t+1} := a\, x_t + (1 - a)\, x_{t-1}$$

for some $a \in [0, 1]$.

## Usage

```
smooth_lin(x, xt, a)
```

### Arguments

| | |
|---|---|
| `x` | `numeric` value of the last value of the parameter. |
| `xt` | `numeric` vector of past values of the parameter (time series). |
| `a` | smoothing parameter $a$. |

### Value

A `numeric` vector of updated parameters.

### See Also

Other smoothing rules: `smooth_dec()`

---

`ts_change` *Time series change stopping rule*

---

### Description

Deterministic stopping rule based on the last change in the value of $\gamma_n$. Changes smaller than `tol`, or relative changes smaller than `reltol` stop the algorithm. This criterion is suitable only in case of deterministic objective functions.

### Usage

```
ts_change(x, reltol = 1e-04, tol = 1e-12)
```

### Arguments

| | |
|---|---|
| `x` | `numeric` vector of last $\gamma_n$ values, as selected by the function passed to `noisyCE2()` through the argument `stopwindow`. |
| `reltol` | relative changes smaller than `tol` stop the algorithm. |
| `tol` | changes smaller than `tol` stop the algorithm. |

### Value

A `numeric` indicating whether the algorithm has converged:

| | |
|---|---|
| `0` | the algorithm has converged. |
| `1` | the algorithm has not converged. |

### See Also

Other stopping rules: `geweke()`

`type_variable` *Functions for defining the types of variables*

## Description

All functions permit fully-customised types of variable to be defined. Functions other than `type_custom` already include standard default values which make the definition of standard variable types easier and quicker.

## Usage

```
type_custom(
  type = "custom",
  init = c(0, 10),
  randomXj = function(n, v) {     rnorm(n, v[1], v[2]) },
  x2v = function(x) {     c(mean(x), sd(x)) },
  v2x = function(v) {     v[1] },
  smooth = list(quote(smooth_lin(x, xt, 1)), quote(smooth_dec(x, xt, 0.9, 10))),
  ...
)

type_real(...)

type_positive(...)

type_negative(...)
```

## Arguments

| | |
|---|---|
| `type` | label for identifying the type of variable. The name is not used internally in any case. |
| `init` | `numeric` vector of starting values of parameters of the sampling distribution. |
| `randomXj` | function for randomly generating variable values according to the sampling distribution. The function should take the number of observations to be generated as a first argument, and the vector of parameters as a second argument; a vector of random values should be returned. |
| `x2v` | function for updating the parameters of the sampling distribution. *No smoothing is needed.* The function should take a single argument to be used for updating the parameters. |
| `v2x` | function for obtaining point values of variable from the parameters of the sampling distribution. |
| `smooth` | list of unevaluated expressions of smoothing functions for each parameter of the sampling distribution. |
| `...` | further arguments to be included into the `typevar` object. In case of function for predefined types, it is possible to use ellipsis for overwriting default values (see § Examples). |

## Value

An object of class `type` and `typevar`, where `type` is the value of the argument `type` passed to `type_custom`, or predefined lables (if not overwritten) in case of other functions.

## Examples

```
# Define a new type of real variable where the first parameter of the
# sampling distribution is updated through the median (instead of the
# mean):
type_real(
  type = 'real2',
  x2v = function(x) { c(median(x), sd(x)) }
)

# Define a new type of real variable whith different smoothing
# parameters:
type_real(
  type = 'real3',
  smooth = list(
    quote(smooth_lin(x, xt, 0.8)),
    quote(smooth_dec(x, xt, 0.99, 15))
  )
)
```

# Index