Package ‘cmaesr’

December 4, 2016

Type Package
Title Covariance Matrix Adaptation Evolution Strategy
Description Pure R implementation of the Covariance Matrix Adaptation - Evolution Strategy (CMA-ES) with optional restarts (IPOP-CMA-ES).
Version 1.0.3
Date 2016-12-04
Maintainer Jakob Bossek <j.bossek@gmail.com>
URL https://github.com/jakobbossek/cmaesr
BugReports https://github.com/jakobbossek/cmaesr/issues
License BSD_2_clause + file LICENSE
Depends ParamHelpers (>= 1.8), BBmisc (>= 1.6), checkmate (>= 1.1), smoof (>= 1.4)
Imports ggplot2
Suggests testthat
ByteCompile yes
RoxygenNote 5.0.1
NeedsCompilation no
Author Jakob Bossek [aut, cre]
Repository CRAN
Date/Publication 2016-12-04 17:21:54

R topics documented:
callMonitor ................................................................. 2
 cmaes ................................................................. 2
getDefaultStoppingConditions ........................................ 4
makeMonitor ............................................................. 5
makeSimpleMonitor .................................................... 6
makeStoppingCondition .............................................. 6

1
callMonitor: Helper to call certain step function of a monitor.

Description

This function serves to call a specific monitor step.

Usage

```r
callMonitor(monitor, step, envir = parent.frame())
```

Arguments

- `step`: [character(1)] One of before, step, after.
- `envir`: [environment] The environment to pass.

CMAES: Covariance-Matrix-Adaptation

Description

Performs non-linear, non-convex optimization by means of the Covariance Matrix Adaptation - Evolution Strategy (CMA-ES).

Usage

```r
cmaes(objective.fun, start.point = NULL, monitor = makeSimpleMonitor(),
      control = list(stop.ons = c(getDefaultStoppingConditions())))
```
Arguments

- **objective.fun** [smoof_function]
  Continuous objective function of type smoof_function. The function must expect a vector of numerical values and return a scalar numerical value.

- **start.point** [numeric]
  Initial solution vector. If NULL, one is generated randomly within the box constraints offered by the parameter set of the objective function. Default is NULL.

- **monitor** [cma_monitor]
  Monitoring object. Default is makeSimpleMonitor, which produces a console output.

- **control** [list]
  Further parameters for the CMA-ES. See the details section for more in-depth information. Stopping conditions are also defined here. By default only some stopping conditions are passed. See get_default_stopping_conditions.

Details

This is a pure R implementation of the popular CMA-ES optimizer for continuous black box optimization [2, 3]. It features a flexible system of stopping conditions and enables restarts [1], which can be triggered by arbitrary stopping conditions and can lead to superior performance on multimodal problems.

You may pass additional parameters to the CMA-ES via the `control` argument. This argument must be a named list. The following control elements will be considered by the CMA-ES implementation:

- **lambda** [integer(1)] Number of offspring generated in each generation.
- **mu** [integer(1)] Number of individuals in each population. Defaults to \[\lambda/2\].
- **weights** [numeric] Numeric vector of positive weights.
- **sigma** [numeric(1)] Initial step-size. Default is 0.5.
- **restart.triggers** [character] List of stopping condition codes / short names (see make_stopping_condition). All stopping conditions which are placed in this vector do trigger a restart instead of leaving the main loop. Default is the empty character vector, i.e., restart is not triggered.
- **max.restarts** [integer(1)] Maximal number of restarts. Default is 0. If set to >= 1, the CMA-ES is restarted with a higher population size if at least one of the stopping conditions is defined as a restart trigger restart.triggers.
- **restart.multiplier** [numeric(1)] Factor which is used to increase the population size after restart. Default is 2.
- **stop.ons** [list] List of stopping conditions. The default is to stop after 10 iterations or after a kind of a stagnation (see get_default_stopping_conditions).
- **log.population** [logical(1L)] Should each population be stored? Default is FALSE.

Value

cma_result Result object. Internally a list with the following components:

- **par.set** [ParamSet] Parameter set of the objective function.
**getDefaultStoppingConditions**

- **best.param** [numeric] Final best parameter setting.
- **best.fitness** [numeric(1L)] Fitness value of the best.param.
- **n.evals** [integer(1L)] Number of function evaluations performed.
- **past.time** [integer(1L)] Running time of the optimization in seconds.
- **n.restarts** [integer(1L)] Number of restarts.
- **population.trace** [list] Trace of population.
- **message** [character(1L)] Message generated by stopping condition.

**Note**

Internally a check for an indefinite covariance matrix is always performed, i.e., this stopping condition is always prepended internally to the list of stopping conditions.

**References**


**Examples**

```r
# generate objective function from smoof package
fn = makeRosenbrockFunction(dimensions = 2L)
res = cmaes(
  fn,
  monitor = NULL,
  control = list(
    sigma = 1.5,
    lambda = 40,
    stop.ons = c(list(stopOnMaxIters(100L)), getDefaultStoppingConditions())
  )
)
print(res)
```

---

`getDefaultStoppingConditions`

*Return list of default stopping conditions.*

**Description**

Default stopping conditions which are active in the reference implementation by Nico Hansen in Python.
**makeMonitor**

**Usage**

getDefaultStoppingConditions()

**Value**

list

---

**makeMonitor**

*Factory method for monitor objects.*

**Description**

Monitors can be plugged in the main `cmaes` function. They have full access to the environment of the optimization routine and can be used to write/log/visualize relevant data in each iteration.

**Usage**

makeMonitor(before = NULL, step = NULL, after = NULL, ...)

**Arguments**

- **before** [function]
  Function called one time after initialization of the EA.
- **step** [function]
  Function applied after each iteration of the algorithm.
- **after** [function]
  Function applied after the EA terminated.
- **...** [any]
  Not used.

**Value**

cma_monitor Monitor object.

**See Also**

`makeSimpleMonitor`, `makeVisualizingMonitor`
**makeSimpleMonitor**  
*Generator for simple monitor.*

**Description**

The simple monitor prints the iteration, current best parameter values and best fitness to the standard output.

**Usage**

```r
makeSimpleMonitor(max.params = 4L)
```

**Arguments**

- `max.params`  
  *integer(1)*  
  Maximal number of parameters to show in output.

**Value**

`cma_monitor`

---

**makeStoppingCondition**  
*Generate a stopping condition object.*

**Description**

A list of stopping conditions can be passed to the `cmaes` function. Instead of hardcoding the stopping criteria into the main function they exist as stand-alone functions for maximal flexibility and extendability.

**Usage**

```r
makeStoppingCondition(name, message, stop.fun, code = name, control = list())
```

**Arguments**

- `name`  
  *character(1)*  
  Name of the stopping condition.

- `message`  
  *character(1)*  
  Message returned if the stopping conditions is active.

- `stop.fun`  
  *function*  
  Function which expects an environment `envir` as its only argument and returns a single logical value.
makeVisualizingMonitor

**code**

[character(1)]

Internal code, i.e., short name used to potentially trigger restarts. Default is name.

**control**

[list]

Control params.

**Value**

cma_stopping_condition Stopping condition object.

---

**Description**

This generator visualizes the optimization process for two-dimensional functions by means of ggplot2.

**Usage**

```r
makeVisualizingMonitor(show.last = FALSE, show.distribution = TRUE, 
xlim = NULL, ylim = NULL)
```

**Arguments**

- **show.last** [logical(1)]
  Should the last population be visualized as well? Default is FALSE.

- **show.distribution** [logical(1)]
  Should an ellipsis of the normal distribution be plotted? Default is TRUE.

- **xlim** [numeric(2) || NULL]
  Limits for the first axis. Default is NULL, i.e., the bounds are determined automatically.

- **ylim** [numeric(2) || NULL]
  Limits for the second axis. Default is NULL, i.e., the bounds are determined automatically.

**Details**

The plot contains points representing the current population, the center of mass or mean value of the population respectively. Optionally an ellipsis representing the normal distribution of the points can be depicted.

**Value**

cma_monitor
stopOnCondCov    \hspace{1cm} \textit{Stopping condition: high condition number.}

\underline{Description}
Stop if condition number of covariance matrix exceeds tolerance value.

\underline{Usage}
\texttt{stopOnCondCov(tol = 1e+14)}

\underline{Arguments}
tol [numeric(1)]
Tolerance value. Default is 1e14.

\underline{Value}
cma_stopping_condition

\underline{See Also}
Other stopping conditions: \texttt{stopOnMaxIters, stopOnNoEffectAxis, stopOnNoEffectCoord, stopOnOptParam, stopOnOptValue, stopOnTimeBudget}

stopOnMaxEvals    \hspace{1cm} \textit{Stopping condition: maximal function evaluations.}

\underline{Description}
Stop if maximal number of function evaluations is reached.

\underline{Usage}
\texttt{stopOnMaxEvals(max.evals)}

\underline{Arguments}
max.evals [integer(1)]
Maximal number of allowed function evaluations.

\underline{Value}
cma_stopping_condition
stopOnMaxIters

**stopOnMaxIters**  
*Stopping condition: maximal iterations.*

### Description
Stop on maximal number of iterations.

### Usage
```
stopOnMaxIters(max.iter = 100L)
```

### Arguments
- **max.iter**  
  [integer(1)]
  Maximal number of iterations. Default is 100.

### Value
```
cma_stopping_condition
```

### See Also
Other stopping conditions: `stopOnCondCov`, `stopOnNoEffectAxis`, `stopOnNoEffectCoord`, `stopOnOptParam`, `stopOnOptValue`, `stopOnTimeBudget`

---

stopOnNoEffectAxis

**stopOnNoEffectAxis**  
*Stopping condition: principal axis.*

### Description
Stop if addition of 0.1 * sigma in a principal axis direction does not change mean value.

### Usage
```
stopOnNoEffectAxis()
```

### Value
```
cma_stopping_condition
```

### See Also
Other stopping conditions: `stopOnCondCov`, `stopOnMaxIters`, `stopOnNoEffectCoord`, `stopOnOptParam`, `stopOnOptValue`, `stopOnTimeBudget`
stopOnNoEffectCoord  Stopping condition: standard deviation in coordinates.

Description
Stop if addition of 0.2 * standard deviations in any coordinate does not change mean value.

Usage
stopOnNoEffectCoord()

Value
cma_stopping_condition

See Also
Other stopping conditions: stopOnCondCov, stopOnMaxIters, stopOnNoEffectAxis, stopOnOptParam, stopOnOptValue, stopOnTimeBudget

stopOnOptParam  Stopping condition: optimal params.

Description
Stop if euclidean distance of parameter is below some tolerance value.

Usage
stopOnOptParam(opt.param, tol = 1e-08)

Arguments
opt.param [numeric]
Known optimal parameter settings.
tol [numeric(1)]
Tolerance value. Default is $1e^{-8}$.

Value
cma_stopping_condition

See Also
Other stopping conditions: stopOnCondCov, stopOnMaxIters, stopOnNoEffectAxis, stopOnNoEffectCoord, stopOnOptValue, stopOnTimeBudget
stopOnOptValue

**Stopping condition: optimal objective value.**

**Description**

Stop if best solution is close to optimal objective value.

**Usage**

```r
stopOnOptValue(opt.value, tol = 1e-08)
```

**Arguments**

- `opt.value` [numeric(1)]
  Known optimal objective function value.
- `tol` [numeric(1)]
  Tolerance value. Default is $1e^{-8}$.

**Value**
cma_stopping_condition

**See Also**

Other stopping conditions: `stopOnCondCov, stopOnMaxIters, stopOnNoEffectAxis, stopOnNoEffectCoord, stopOnOptParam, stopOnTimeBudget`

---

stopOnTimeBudget

**Stopping condition: maximal time.**

**Description**

Stop if maximal running time budget is reached.

**Usage**

```r
stopOnTimeBudget(budget)
```

**Arguments**

- `budget` [integer(1)]
  Time budget in seconds.

**Value**
cma_stopping_condition
**stopOnTolX**

**Description**
Stop if the standard deviation falls below a tolerance value in all coordinates?

**Usage**
```
stopOnTolX(tol = 1e-12)
```

**Arguments**
- `tol`: [integer(1)]
  - Tolerance value.

**Value**
cma_stopping_condition

---

**See Also**
Other stopping conditions: `stopOnCondCov, stopOnMaxIters, stopOnNoEffectAxis, stopOnNoEffectCoord, stopOnOptParam, stopOnOptValue`

---

**Stoping condition: low standard deviation.**
Index

*Topic optimize
  cmaes, 2

callMonitor, 2
cmaes, 2, 5, 6

getDefaultStoppingConditions, 3, 4

makeMonitor, 5
makeSimpleMonitor, 3, 5, 6
makeStoppingCondition, 3, 6
makeVisualizingMonitor, 5, 7

ParamSet, 3

stopOnCondCov, 8, 9–12
stopOnMaxEvals, 8
stopOnMaxIters, 8, 9, 10–12
stopOnNoEffectAxis, 8, 9, 10–12
stopOnNoEffectCoord, 8–10, 10, 11, 12
stopOnOptParam, 8–10, 10, 11, 12
stopOnOptValue, 8–10, 11, 12
stopOnTimeBudget, 8–11, 11
stopOnTo1X, 12