Package ‘rCMA’

October 14, 2022

Type Package

Title R-to-Java Interface for 'CMA-ES'

Version 1.1.1

Date 2015-04-30

Author Wolfgang Konen <`wolfgang.konen@fh-koeln.de`>, Nikolaus Hansen <hansen .AT. lri.fr>

Maintainer Wolfgang Konen <`wolfgang.konen@fh-koeln.de`>

Description Tool for providing access to the Java version 'CMAEvolutionStrategy' of Nikolaus Hansen. 'CMA-ES' is the Covariance Matrix Adaptation Evolution Strategy, see <`https://www.lri.fr/~hansen/cmaes_inmatlab.html#java`>.

License GPL (>= 3)

Depends R (>= 2.14.0),

Suggests rJava

Collate 'rCMA.R' 'cmaGetters.R' 'cmaEvalMeanX.R'

NeedsCompilation no

Repository CRAN

Date/Publication 2022-06-24 11:41:52 UTC

R topics documented:

`rCMA-package` *R interface to the Java CMA-ES of Niko Hansen*

Description

CMA-ES R-to-Java interface

Details

Package:	rCMA
Type:	Package
Version:	1.1
Date:	2015-04-30
License:	GPL (>= 3)
LazyLoad:	yes

rCMA is a package to perform CMA-ES optimization, using the *Java* implementation by Niko Hansen [Hansen2009].
CMA-ES [HansOst96, Hansen13] is the Covariance Matrix Adapting Evolutionary Strategy for numeric black box optimization.
The main features of rCMA are:

1. Abiltiy to start the Java CMA-ES optimization with fitness functions defined in R.
2. Constraint handling: Arbitrary constraints can be incorporated, see function parameter `isFeasible` in `cmaOptimDP`.
3. Extensibility: Full access to all methods of the Java class `CMAEvolutionStrategy` through package `rJava`. New methods can be added easily. See the documentation of `cmaEvalMeanX` for further details, explanation of JNI types and a full example.
4. Test and Debug: The access of Java methods from R allows for easy debugging and test of programs using `CMAEvolutionStrategy` through R scripts without the necessity to change the underlying JAR file.

The main entry point functions are `cmaNew`, `cmaInit` and `cmaOptimDP`.

Note: To install `rJava` properly on some Unix systmes, it might be necessary to issue as root the command `R CMD javareconf` once, or, as normal user to issue the command `R CMD javareconf -e` prior to installing package `rJava` or prior to loading library `rJava`.

Author(s)

Wolfgang Konen (<`wolfgang.konen@fh-koeln.de`>)

References

[HansOst96] Hansen, N. and Ostermeier, A.: Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, pp. 312-317, 1996. PDF.

[Hansen09] https://www.lri.fr/~hansen/javadoc Nikolaus Hansen: Javadoc for CMA-ES Java package fr.inria.optimization.cmaes, 2009.
[Hansen13] https://www.lri.fr/~hansen/cmaesintro.html Nikolaus Hansen: The CMA Evolution Strategy Web Page, 2013.
[Urbanek13] http://cran.r-project.org/web/packages/rJava Urbanek, S.: rJava: Low-level R to Java interface, 2013.
[Oracle14] http://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.html Oracle: The Java Native Interface. Programmer's Guide and Specification, 2014.

`cmaCalcFitness` *Calculate the fitness of a population.*

Description

The population is usually obtained by `cmaSamplePopulation`.

Usage

```
cmaCalcFitness(cma, popR, fitFunc)
```

Arguments

`cma`	CMA-ES Java object, already initialized with `cmaInit`
`popR`	a (`dimension` x `popSize`) matrix from `cmaSamplePopulation`
`fitFunc`	a function to be minimized. Signature: accepts a vector `x`, returns a `double`.

Value

`fitness`, a vector of length `cmaGetPopulationSize(cma)` with the fitness of each individuum

Author(s)

Wolfgang Konen, FHK, 2013

See Also

`cmaSamplePopulation`, `cmaUpdateDistribution`, `cmaNew`

Examples

```
cma <- cmaNew();
   cmaInit(cma,dimension=2,initialX=1.5);
   popR <- cmaSamplePopulation(cma);
   fitFunc <- function(x) {sum(x*x)};
   fitness <- cmaCalcFitness(cma,popR,fitFunc);
   cmaUpdateDistribution(cma,fitness);
```

`cmaEvalMeanX` *Evaluate the meanX of the current population.*

Description

After executing `cmaOptimDP`, there is a current population and a best-ever solution. Evaluate for the mean of the current population whether it is feasible and whether the mean is an even better solution. If so, update the best-ever solution.

Usage

```
cmaEvalMeanX(cma, fitFunc, isFeasible = function(x) TRUE)
```

Arguments

`cma`	CMA-ES Java object, already initialized with `cmaInit`
`fitFunc`	a function to be minimized. Signature: accepts a vector x, returns a double.
`isFeasible`	`[function(x){TRUE}]` a Boolean function checking the feasibility of the vector `x`. The default is to return always `TRUE`.

Details

The code of this function is also instructive as a full example for the extensibility of the `rJava` interface to CMA-ES. See the full code in `demo/demoEvalMeanX`. Some example `rJava`-calls are:

```
rJava::.jcall(cma,"[D","getMeanX");
bestSolutionObj =
 rJava::.jcall(cma,"Lfr/inria/optimization/cmaes/CMASolution;","setFitnessOfMeanX",fitFunc(meanX)
rJava::.jcall(bestSolutionObj,"J","getEvaluationNumber");
```

Every direct method of classes in the CMA-ES Java package `cmaes` (see [Hansen09] for the complete Javadoc and [Hansen13] for an overview on CMA-ES in total) can be accessed with the `.jcall`-mechanism of the `rJava` R package:

```
rJava::.jcall(obj,returnType,method,...)
```

where `...` stands for the calling parameter(s) of `method`.
`returnType` is a string following the JNI type convention (see, e.g. [Oracle14])

Field Descriptor	Java Language Type
Z	boolean
C	char
I	int
J	long
F	float
D	double
[I	int[]

[[D	double[][]
Ljava/langString;	java.lang.String
S	java.lang.String
T	short

(Note: (a) the terminating `";"` in `"Ljava/langString;"` (!) and (b) `"S"` is a short hand for `"Ljava/langString;"` and `"T"` is the re-mapped code for `short`.)

The calling parameters in `...` have to be matched exactly. In R, numeric vectors are stored as `doubles`, so the calling syntax

```
bestSolutionObj = .jcall(cma,rType,"setFitnessOfMeanX",fitFunc(meanX));
```

is just right for the Java method `setFitnessOfMeanX(double[])` . In other cases, the calling R variable `x` has to be cast explicitly:

Cast	Java Language Type
.jbyte(x)	byte
.jchar(x)	char
as.integer(x)	int
.jlong(x)	long
.jfloat(x)	float

Value

`bestSolution`, a list with entries:

`bestX`	a vector of length `dimension` containing the best-ever solution, including `meanX`
`meanX`	a vector of length `dimension` containing the mean of the current (last) population in `cma`
`bestFitness`	the best-ever fitness value, including the evaluation of meanX
`bestEvalNum`	the function evaluation count where `bestFitness` occured
`lastEvalNum`	the total function evaluation count. If `bestEvalNum==lastEvalNum` then the best-ever fitness occured in the evaluation of `meanX`.

Author(s)

Wolfgang Konen, FHK, 2013-2015

References

[Hansen09] https://www.lri.fr/~hansen/javadoc Nikolaus Hansen: Javadoc for CMA-ES Java package fr.inria.optimization.cmaes, 2009.
[Hansen13] https://www.lri.fr/~hansen/cmaesintro.html Nikolaus Hansen: The CMA Evolution Strategy, 2013.
[Oracle14] http://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.

`html` Oracle: The Java Native Interface. Programmer's Guide and Specification. Chapter 3 (JNI types), Sec. 'Type Signatures', 2014.

See Also

`cmaInit`, `cmaOptimDP`

Examples

```
## Not run:
   ## just to show the syntax, without calling cmaOptimDP
   fitFunc <- function(x) {  sum(x*x); }
   isFeasible <- function(x) {  TRUE;  }
   cma <- cmaNew(propFile="CMAEvolutionStrategy.properties");
   cmaInit(cma,dimension=2,initialX=1.5);
   bestSolution=cmaEvalMeanX(cma,fitFunc,isFeasible);
   str(bestSolution);

## End(Not run)
```

`cmaInit` *Initialize a CMA-ES Java object.*

Description

Initialize a CMA-ES Java object.

Usage

```
cmaInit(cma, seed = NULL, dimension = NULL, initialX = NULL,
  initialStandardDeviations = NULL)
```

Arguments

`cma`	CMA-ES Java object, as created by `cmaNew`
`seed`	[NULL] if not NULL, set the seed to the given value
`dimension`	[NULL] if not NULL, overwrite the dimension setting from `propFile` (`cmaNew`)
`initialX`	[NULL] if not NULL, overwrite the initialX setting from `propFile` (`cmaNew`). initialX can be a double or a double vector of length `dimension`.
`initialStandardDeviations`	[NULL] if not NULL, overwrite the initialStandardDeviations setting from `propFile` `cmaNew`. initialStandardDeviations can be a double or a double vector of length `dimension`.

Value

`fitness`, a vector of 0's with the length of the intended population.

Note

As a side effect, the CMA-ES Java object `cma` of class `CMAEvolutionStrategy` is transferred into an augmented state. As a second side effect, the population size is set to

$$\lambda = 4 + 3floor(ln(n))$$

where $n =$`dimension`.

Author(s)

Wolfgang Konen, FHK, 2013

See Also

`cmaNew`, `cmaOptimDP`

Examples

```
cma <- cmaNew();
   cmaInit(cma,seed=42,dimension=2,initialX=1.5);
```

`cmaNew` *Create a new CMA-ES Java object.*

Description

Create a new CMA-ES Java object.

Usage

```
cmaNew(propFile = NULL)
```

Arguments

`propFile`	[NULL] filename of a file with property settings. If NULL, read file `CMAEvolutionStrategy.properties` from the package directory (`find.package("rCMA")`)

Value

the new CMA-ES Java object of class `CMAEvolutionStrategy`, which has as additional attribute `props`, the Java `Properties` object as read from `propFile`.

Note

The default properties file can be found in `CMAEvolutionStrategy.properties`. A read-only copy can be inspected by browsing to "Index" (of package rCMA), then "Overview of user guides ...".

It allows to set more parameter, especially more stop conditions.

Author(s)

Wolfgang Konen, FHK, 2013

See Also

`cmaInit`

Examples

```
## show how element initialX can be inferred from attribute props:
   ## (see  cmaEvalMeanX-documentation for further details on .jcall and its argument "S")
   cma <- cmaNew();
   props <- attr(cma,"props");
   initialX = rJava::.jcall(props,"S","getProperty","initialX");
   print(initialX);
```

`cmaOptimDP` *Perform a CMA-ES optimization with constraints (DP).*

Description

The optimization uses DP (death penalty) for handling constraint violations: Each time an infeasible individual is encountered, it is thrown away and a new individual is resampled from the CMA distribution.

Usage

```
cmaOptimDP(cma, fitFunc, isFeasible = function(x) {     TRUE },
  maxDimPrint = 5, iterPrint = 10, verbose = 2)
```

Arguments

`cma`	CMA-ES Java object, already initialized with `cmaInit`
`fitFunc`	a function to be minimized. Signature: accepts a vector `x`, returns a `double`.
`isFeasible`	[`function(x){TRUE}`] a Boolean function checking the feasibility of the vector `x`. The default is to return always `TRUE`.
`maxDimPrint`	[5] how many dimensions of vector `x` to print in diagnostic output
`iterPrint`	[10] after how many iterations should diagnostic output be printed?
`verbose`	[2] possible values are 0 (no output), 1, 2 (much output)

Details

This functions loops through iterations (generations) until a stop condition is met: In each iteration, a population is sampled (infeasible individuals are replaced via Java function `resampleSingle`), its fitness vector is evaluated and the CMA distribution is updated according to this fitness vector.

Every `iterPrint` generations a one-line diagnostic output of the form

```
iter  fitness | x1 x2 ... xp
```

is printed where `fitness` is the current best value of the fitness function to be minimized and `x1 x2 ... xp` are the first `maxDimPrint` dimensions of the corresponding best point in input space.

Value

`res`, a list with diagnostic output from the optimization run:

`sMsg`	a string vector with all console output from the optimization run. To print it, use: `cat(sMsg)` or `for (x in sMsg) cat(x)`
`bestX`	vector of length `dimension` with the best-ever solution X
`bestFitness`	the corresponding best-ever fitness
`bestEvalNum`	the fitness function evaluation number which gave this best-ever result
`nIter`	number of iterations
`fitnessVec`	vector of length `nIter`: the best fitness after each iteration
`xMat`	(`nIter` x `dimension`)-matrix: `xMat[i,]` is the best solution X after iteration `i`
`cfe`	number of constraint function evaluations (`isFeasible`)
`ffe`	number of fitness function evaluations (`fitFunc`)

Note

If your fitness function depends on other parameters besides `x`, then encapsulate it in a new function `fitFunc` at a place where the other parameters are accessible and rely on R's mechanism to locate the other parameters in the environment surrounding `fitFunc`:

```
par1 <- someObject;


fitFunc <- function(x) {  myFuncWithOtherPars(x,par1); }
```

Author(s)

Wolfgang Konen, FHK, 2013-2015

See Also

`cmaNew`, `cmaInit`

Examples

```
## demo/demoCMA2.R: demo usage of package rCMA.
##
## After doing the unconstrained sphere (as in demoCMA1.r, for later reference in plot),
## the constrained sphere problem TR2 is solved.
## The problem TR2 has in addition to the fitness function 'sphere' the constraint function
## 'above the hyperplane sum_i(x_i) = n', where n is the input space dimension.
## The constrained optimum is at (1,...,1) and it has the value fTarget2=n.
##
## Note that in this second case, the optimimum lies exactly at the boundary
## of the feasible region: res2$bestX=c(1,...,1).
##
## This script does exactly the same as class CMAExampleConstr in cma_jAll.jar,
## but it allows to define the functions fitFunc and isFeasible on the R side.
## They can be replaced by arbitrary other R functions, which may depend on other
## R variables as well.
##
## The constraint handling approach is a very simple one: death penalty, i.e. if we get an
## infeasible individual, it is immediately discarded and a new one is drawn from the distribution.
## (This approach will run into trouble if the current distribution does not allow to reach any
## feasible solutions.)
##
library(rCMA)
fitFunc <- function(x) {  sum(x*x); }
isFeasible <- function(x) {  (sum(x) - length(x)) >= 0;  }
n = 2;

cma <- cmaNew(propFile="CMAEvolutionStrategy.properties");
cmaInit(cma,seed=42,dimension=n,initialX=1.5, initialStandardDeviations=0.2);
res1 = cmaOptimDP(cma,fitFunc,iterPrint=30);

cma <- cmaNew(propFile="CMAEvolutionStrategy.properties");
cmaInit(cma,seed=42,dimension=n,initialX=1.5, initialStandardDeviations=0.2);
res2 = cmaOptimDP(cma,fitFunc,isFeasible,iterPrint=30);

fTarget =c(0,n);
plot(res1$fitnessVec-fTarget[1],type="l",log="y",xlim=c(1,max(res1$nIter,res2$nIter))
    ,xlab="Iteration",ylab="Distance to target fitness");
lines(res2$fitnessVec-fTarget[2],col="red");
legend("topright",legend=c("TR2","sphere"),lwd=rep(1,2),col=c("red","black"))
str(res2);

bestSolution=rCMA::cmaEvalMeanX(cma,fitFunc,isFeasible);
str(bestSolution);
```

`cmaSamplePopulation` *Sample a population from the current CMA-ES distribution.*

Description

The population size is given by `cmaGetPopulationSize(cma)`. It can be either set manually with `cmaSetPopulationSize(cma,p)`, prior to `cmaInit(cma)`, or CMA-ES will use the default population size
`popSize = 4 + 3*log(dimension)`.

Usage

```
cmaSamplePopulation(cma)
```

Arguments

`cma`	CMA-ES Java object, already initialized with `cmaInit`

Value

`popR`, a (`dimension` x `popSize`) matrix with `popR[,1]` being the first individuum in the population.
`dimension` = `cmaGetDimension(cma)`
`popSize` = `cmaGetPopulationSize(cma)`

Author(s)

Wolfgang Konen, FHK, 2013

See Also

`cmaUpdateDistribution`, `cmaNew`

Examples

```
cma <- cmaNew();
   cmaInit(cma,dimension=2,initialX=1.5);
   popR <- cmaSamplePopulation(cma);
```

`cmaSetDimension` *rCMA Getters and Setters.*

Description

Get or set various elements of CMA-ES Java object `cma`.

`cmaSetDimension` sets the problem dimension (only prior to `cmaInit`)
`cmaGetDimension` returns the problem dimension
`cmaSetPopulationSize` sets the population size (only prior to `cmaInit`)
`cmaGetPopulationSize` returns the population size
`cmaSetInitialX` set the mean vector for the initial population (only prior to `cmaInit`)
`cmaGetInitialX` returns the mean vector for the initial population

`cmaSetCountEval` sets the counter for fitness function evaluations (only prior to `cmaInit`)
`cmaGetCountEval` returns the counter for fitness function evaluations

Usage

```
cmaSetDimension(cma, i)

cmaGetDimension(cma)

cmaSetPopulationSize(cma, i)

cmaGetPopulationSize(cma)

cmaSetInitialX(cma, initialX)

cmaGetInitialX(cma)

cmaSetCountEval(cma, p)

cmaGetCountEval(cma)
```

Arguments

`cma`	CMA-ES Java object, created with `cmaNew`
`i`	a parameter of type integer
`initialX`	either a double or a double vector of length `cmaGetDimension`
`p`	a parameter of type long

Value

none for the setters, the requested element(s) for the getters

See Also

`cmaSetStopFitness`, `cmaNew`, `cmaInit`

`cmaSetStopFitness` *rCMA Stop Conditions.*

Description

Set various stop conditions of CMA-ES Java object `cma` (only prior to `cmaInit`).

`cmaSetStopFitness` sets the stop condition: fitness function below `d` (default: DOUBLE.MinValue)
`cmaSetStopMaxFunEvals` sets the stop condition: max number of fitness function evaluations
`cmaSetStopTolFun` sets the stop condition: delta of fitness function below `d` (default: 1e-12)

Usage

```
cmaSetStopFitness(cma, d)

cmaSetStopMaxFunEvals(cma, p)

cmaSetStopTolFun(cma, d)
```

Arguments

`cma`	CMA-ES Java object, created with `cmaNew`
`d`	a parameter of type double
`p`	a parameter of type long

Note

If your fitness can become negative, you need to set `cmaSetStopFitness` to a value different from the default to prevent premature stopping.

The properties file (read by `cmaNew`) can be used to set further stop conditions. If they are not set, the following defaults are active:

name	default setting	meaning
stopTolFunHist	1e-13	similar to stopTolFun, see CMA-ES Javadoc for details
stopTolX	0.0	stop if seacrh steps become smaller than stopTolX
stopTolXfactor	0.0	stop if search steps become smaller than stopTolXFactor * initial step size
stopMaxIter	+Inf	stop if number of iterations (generations) are greater

See Also

`cmaSetDimension`, `cmaNew`, `cmaInit`

`cmaUpdateDistribution` *Update CMA-ES distribution with the fitness vector of the last population.*

Description

Update CMA-ES distribution with the fitness vector of the last population.

Usage

```
cmaUpdateDistribution(cma, fitness)
```

Arguments

`cma`	CMA-ES Java object, already initialized with `cmaInit`
`fitness`	vector of length `cmaGetPopulationSize(cma)` with the fitness of each individuum

Note

As a side effect, the CMA-ES Java object cma of class `CMAEvolutionStrategy` is augmented.

Author(s)

Wolfgang Konen, FHK, 2013

See Also

`cmaSamplePopulation`, `cmaNew`, `cmaOptimDP`

Index