Package 'SACOBRA'

August 25, 2015

Type Package

Title Self-Adjusting COBRA

Version 0.7

Date 2015-08-24

Author Wolfgang Konen <wolfgang.konen@fh-koeln.de> [aut], Samineh Bagheri [cre, aut], Patrick Koch [aut]

Maintainer Samineh Bagheri <samineh.bagheri@fh-koeln.de>

Description Performs constrained optimization for expensive black-box problems.

License GPL (>= 2)

Depends R (>= 2.14.0),

Suggests nloptr, FNN, MASS, dfoptim, DEoptim, lhs, rgl, grDevices, scales

Imports testit, methods

Collate 'cobraInit.R' 'cobraPhaseI.R' 'cobraPhaseII.R' 'defaultRI.R' 'defaultSAC.R' 'defaultTR.R' 'drawSurrogate3d.R' 'fnArchive.R' 'initialHjkb.R' 'isres2.R' 'multiRunPlot.R' 'multiCOBRA.R' 'nmkb2.R' 'RbfInter.R' 'repairChootinan.R' 'repairInfeasRI2.R' 'SACOBRA.R' 'startCobra.R' 'trustRegion.R'

NeedsCompilation no

Repository CRAN

Date/Publication 2015-08-25 19:20:50

R topics documented:

`SACOBRA-package` *Self-adjusting Constrained Optimization with RBF*

Description

Self-adjusting Constrained Optimization with RBF

Details

Package:	SACOBRA
Type:	Package
Version:	0.7
Date:	30.08.2015
License:	GPL (>= 2)
LazyLoad:	yes

SACOBRA is a package for numeric constrained optimization of expensive black-box functions under severely limited budgets. It is an extension of the COBRA algorithm by Regis (R. Regis: "Constrained optimization by radial basis function interpolation for high-dimensional expensive black-box problems with infeasible initial points", Engineering Optimization, Taylor & Francis, 46, p. 218-243, 2013)

These extensions include:
1) A repair algorithm for infeasible solutions,
2) several internal optimizers and several initial design generation methods,
3) self-adjusting random restart algorithm,
4) self-adjusting logarithmic transform for objective functions with large output ranges,

5) range normalization of constraint functions,
6) self-adjusting DRC selection.

SACOBRA performs optimization with a minimum of true function evaluations. It has proven to work well on probelms with high dimensions (e.g. d=124) and many constraints (e.g. 60). It is usable for all kind of numeric optimization, but not for combinatorial optimization.

For more details see:
Koch, P.; Bagheri, S.; Konen, W. et al.: "A New Repair Method For Constrained Optimization". In: Proceedings of the 17th Genetic and Evolutionary Computation Conference, 2015, http://www.gm.fh-koeln.de/~konen/Publikationen/Koch2015a-GECCO.pdf
and
Koch, P.; Bagheri, S. et al.: "Constrained Optimization with a Limited Number of Function Evaluations" In: W. Hoffmann, F. & Huellermeier, E. (Eds.), Proceedings 24. Workshop Computational Intelligence, Universitaetsverlag Karlsruhe, 2014, 119-134, http://www.gm.fh-koeln.de/~konen/Publikationen/Koch2014a-GMA-CI.pdf.

The main entry point functions are `cobraInit` and `startCobra`. See `startCobra` for an example and `cobraInit` for an overview of adjustable SACOBRA-parameters. Another example is in `multiCOBRA`.

Author(s)

Samineh Bagheri (<Samineh.Bagheri@fh-koeln.de>), Wolfgang Konen (<Wolfgang.Konen@fh-koeln.de>), Patrick Koch

References

http://lwibs01.gm.fh-koeln.de/blogs/ciop/research/monrep/

`cobraInit` *Initial phase for COBRA optimizer*

Description

Constraint-based optimization initialization

Usage

```
cobraInit(xStart, fn, fName, lower, upper, nConstraints, feval,
  initDesign = "RANDOM", initDesPoints = 2 * length(xStart) + 1,
  initDesOptP = NULL, initBias = 0.005, seqOptimizer = "COBYLA",
  seqFeval = 1000, seqTol = 1e-06, penaF = c(3, 1.7, 3e+05),
  squaresF = TRUE, squaresC = TRUE, conTol = 0,
  constraintHandling = "DEFAULT", sigmaD = c(3, 2, 100),
  repairInfeas = FALSE, repairMargin = NULL, ri = defaultRI(),
  DOSAC = 1, sac = defaultSAC(DOSAC), epsilonInit = NULL,
  epsilonMax = NULL, solu = NULL, saveIntermediate = FALSE,
```

```
saveSurrogates = FALSE, RBFmodel = "cubic", RBFwidth = -1,
GaussRule = "One", RBFrho = 0, skipPhaseI = TRUE,
trueFuncForSurrogates = FALSE, drFactor = 1, XI = DRCL,
rescale = TRUE, newlower = -1, newupper = 1, DEBUG_XI = FALSE,
SKIP_HIGH = FALSE, DEBUG_RBF = FALSE, TrustRegion = FALSE,
TRlist = defaultTR(), verbose = 1, verboseIter = 10, cobraSeed)
```

Arguments

xStart	a vector containing the starting point for the optimization problem
fn	objective function that is to be minimized, should return a vector of the objective function value and the constraint values
fName	file name .Rdata where the results of cobraPhaseII are saved
lower	lower bound of search space, same dimension as xStart
upper	upper bound of search space, same dimension as xStart
nConstraints	number of constraints
feval	maximum number of function evaluations
initDesign	["RANDOM"] one out of ["RANDOM","LHS","BIASED","OPTIMIZED","OPTBIASED"]
initDesPoints	number of initial points, must be smaller than feval
initDesOptP	[NULL] only for initDesign=="OPTBIASED": number of points for the "OPT" phase. If NULL, take initDesPoints.
initBias	[0.005] bias for normal distribution in "OPTBIASED" and "BIASED"
seqOptimizer	["COBYLA"] string defining the optimization method for COBRA phases I and II, one out of ["COBYLA","ISRES","HJKB","NMKB","ISRESCOBY"]
seqFeval	maximum number of function evaluations on the surrogate model
seqTol	[1e-6] Convergence tolerance, see param tol in nmkb
penaF	[c(3,1.7,3e5)] parameters for dynamic penalty factor (fct subProb in cobraPhaseII): c(start,augment,max)
squaresF	[FALSE] set to TRUE for fitnessSurrogate <- trainCubicRBF(..., squares=T)
squaresC	[FALSE] set to TRUE for constraintSurrogates <- trainCubicRBF(..., squares=T)
conTol	[0.0] constraint violation tolerance
constraintHandling	["DEFAULT"] (other choices: "JOINESHOUCK", "SMITHTATE", "COIT", "BAECKKHURI"; experimental, only for penalty-based internal optimizers NMKB or HJKB, see the code in function subProb in cobraPhaseII)
sigmaD	[c(3,2.0,100)] parameters for dynamic distance factor (fct subProb in cobraPhaseII): c(start,augment,max)
repairInfeas	[FALSE] if TRUE, try to repair infeasible solutions
repairMargin	– deprecated –
ri	[defaultRI()] list with other parameters for repairInfeasRI2

DOSAC	[0|1|2] if >0, any elements of sac not set by the user are set to defaultSAC(DOSAC). 0: COBRA-R settings, 1: SACOBRA settings, 2: SACOBRA settings with fewer parameters and more online adujustement (aFF and aCF are done parameter free).
sac	[defaultSAC(DOSAC)] list with other parameters for SACOBRA
epsilonInit	initial constant added to each constraint to maintain a certain margin to boundary
epsilonMax	maximum for constant added to each constraint
solu	[NULL] the best-known solution (only for diagnostics). This is normally a vector of length d. If there are multiple solutions, it is a matrix with d columns (each row is a solution). If NULL, then the current best point will be used in cobraPhaseII. solu is given in original input space.
saveIntermediate	[FALSE] if TRUE, then cobraPhaseI + II save intermediate results in dir 'results/' (create it, if necessary)
saveSurrogates	[FALSE] if TRUE, then cobraPhaseII returns the last surrogate models in cobra$fitnessSurrogate and cobra$constraintSurrogates
RBFmodel	["cubic"] a string assigning type of the RBF model, "cubic" or "Gaussian"
RBFwidth	[-1] only relevant for Gaussian RBF model, see trainGaussRBF
GaussRule	["One"] only relevant for Gaussian RBF model, see trainGaussRBF
RBFrho	[0.0] experimental: 0: interpolating, >0, approximating (spline-like) Gaussian RBFs
skipPhaseI	[FALSE] if TRUE, then skip cobraPhaseI
trueFuncForSurrogates	[FALSE] if TRUE, use the true (constraint & fitness) functions instead of surrogates (only for debug analysis)
drFactor	[1.0] factor multiplied to the DR-constraint (gCOBRA)
XI	magic parameters for the distance requirement (DR)
rescale	[TRUE] if TRUE change [lower,upper] to hypercube [newlower,newupper]^d
newlower	[-1] lower bound of each rescaled input space dimension, if rescale==TRUE
newupper	[+1] upper bound of each rescaled input space dimension, if rescale==TRUE
DEBUG_XI	[FALSE] if TRUE, then print in cobraPhaseII extra debug information: xStart in every iteration to console and add some extra debug columns to cobra$df
SKIP_HIGH	[FALSE] (deprecated) if TRUE, then build the surrogate models in cobraPhaseII by skipping the data points having Fres (objective function) in the highest decile.
DEBUG_RBF	[FALSE] visualize RBF (only for dimension==2)
TrustRegion	[FALSE] if TRUE, an embedded trust region algorithm trustRegion is performed.
TRlist	[defaultTR()] a list of parameters, needed only in case TrustRegion==TRUE.
verbose	[1] one out of [0|1|2], how much output to print
verboseIter	[10]
cobraSeed	seed for random number generator

Details

If epsilonInit or epsilonMax are NULL on input, then cobra$epsilonInit and cobra$epsilonMax, resp., are set to 0.005*l where l is the smallest side of the search box.
Note that the parameters penaF, sigmaD, constraintHandling are only relevant for penalty-based internal optimizers NMKB or HJKB.

Value

cobra, an object of class COBRA, this is a (long) list containing most of the argument settings (see above) and in addition (among others)

A	(feval x dim)-matrix containing the initial design points in input . space. If rescale==TRUE, all points are in **rescaled** input space.
Fres	a vector of the objective values of the initial design points
Gres	a matrix of the constraint values of the initial design points
Tfeas	the threshhold parameter for the number of consecutive iterations that yield feasible solutions before margin epsilon is reduced
Tinfeas	the threshhold parameter for the number of consecutive iterations that yield infeasible solutions before margin epsilon is increased
numViol	number of constraint violations
maxViol	maximum constraint violation
refinedX	A vector of all refined solutions generated by trust region algorithm (see trustRegion)

Note that cobra$Fres, cobra$fbest, cobra$fbestArray and similar contain always the objective values of the orignial function cobra$fn[1]. (The surrogate models may be trained on a plog-transformed version of this function.)

Author(s)

Wolfgang Konen, Samineh Bagheri, Patrick Koch, Cologne Univeristy of Applied Sciences

See Also

startCobra, cobraPhaseI, cobraPhaseII

---

cobraPhaseI *Find a feasible solution.*

Description

Find a feasible solution using the COBRA optimizer phase I by searching new infill points with the help of RBF surrogate models

Usage

```
cobraPhaseI(cobra)
```

Arguments

cobra	an object of class COBRA, this is a (long) list containing all settings from cobraInit

Value

cobra, an object of class COBRA

Author(s)

Wolfgang Konen, Samineh Bagheri, Patrick Koch, Cologne Univeristy of Applied Sciences

See Also

cobraPhaseII, cobraInit

---

cobraPhaseII *Improve the feasible solution by searching new infill points*

Description

Improve the feasible solution using the COBRA optimizer phase II by searching new infill points with the help of RBF surrogate models. May be even called if no feasible solution is found yet, then phase II will try to find feasible solutions.

Usage

```
cobraPhaseII(cobra)
```

Arguments

cobra	an object of class COBRA, this is a (long) list containing all settings from cobraInit

Value

cobra, an object of class COBRA from cobraInit, enhanced here by the following elements (among others):

fn	function returning an (m+1)-vector c(objective,g1,...,gm). This function may be a rescaled and plog-transformed version of the original fn passed into cobraInit. The original fn is in cobra$originalFn.
df	data frame with summary of the optimization run (see below)
df2	data frame with additional summary information (see below)
A	(feval x dim)-matrix containing all evaluated points in input space. If rescale==TRUE, all points are in **rescaled** input space.
Fres	a vector of the objective values of all evaluated points

Gres	a matrix of the constraint values of all evaluated points
fbest	the best feasible objective value found
xbest	the point in input space yielding the best feasible objective value
ibest	the corresponding iteration number (row of cobra$df, of cobra$A)
PLOG	If TRUE, then the objective surrogate model is trained on the plog-transformed objective function.

Note that cobra$Fres, cobra$fbest, cobra$fbestArray and similar contain always the objective values of the orignial function cobra$fn[1]. (The surrogate models may be trained on a plog-transformed version of this function.)

The data frame cobra$df contains one row per iteration with columns

iter

y true objective value Fres

predY surrogate objective value. Note: The surrogate may be trained on plog-transformed training data, but predY is transformed back to the original objective range. NA for the initial design points.

predSolu surrogate objective value at best-known solution cobra$solu, if given. If cobra$solu is NULL, take the current point instead. Note: The surrogate may be trained on plog-transformed training data, but predSolu is transformed back to the original objective range. NA for the initial design points.

feasible

feasPred

nViolations

maxViolation

FEval number of function evaluations in sequential optimizer. NA if it was a repair step

Best ever-best feasible objective value fbest. As long as there is no feasible point, take among those with minimum number of violated constraints the one with minimum Fres.

optimizer e.g. "COBYLA"

optimizationTime in sec

conv

seed

The data frame cobra$df2 contains one row per phase-II-iteration with columns

iter

predY surrogate objective value. Note: The surrogate may be trained on plog-transformed training data, but predY is transformed back to the original objective range. NA for the initial design points.

predVal surrogate objective value + penalty

predSolu surrogate objective value at true solution (see cobradfpredSolu)

predSoluPenal surrogate objective value + penalty at true solution (only diagnostics)

sigmaD

penaF

XI the DRC element used in the current iteration

EPS

Author(s)

Wolfgang Konen, Samineh Bagheri, Patrick Koch, Cologne Univeristy of Applied Sciences

See Also

cobraPhaseI, cobraInit

---

defaultRI *Default settings for* repairInfeasRI2 *and* repairChootinan.

Description

Sets suitable defaults for the repair-infeasible part of COBRA.
With the call setOpts(myRI,defaultRI()) it is possible to extend a partial list myRI to a list containing all ri-elements (the missing ones are taken from defaultRI())

Usage

```
defaultRI(repairMargin = 0.01)
```

Arguments

repairMargin [1e-2] repair only solutions whose infeasibility is less than this margin

Details

A solution x is said to be ϵ**-feasible** for constraint function f, if

$$f(x) + \epsilon \leq 0$$

The **infeasibility** of a solution is its maximum constraint violation (0 for a feasible solution).

Value

a list with the following elements:

RIMODE [2] one out of {0,1,2,3 } with 0,1: deprecated older versions of RI2, 2: the recommended RI2-case, see repairInfeasRI2, 3: Chootinan's method, see repairChootinan

eps1 [1e-4] include all constraints not eps1-feasible into the repair mechanism

eps2 [1e-4] selects the solution with the shortest shift among all random realizations which are eps2-feasible

q [3.0] draw coefficients α_k from uniform distribution $U[0, q]$

mmax [1000] draw mmax random realizations

repairMargin repair only solutions whose infeasibility is less than this margin.

repairOnlyFresBetter [FALSE] if TRUE, then repair only iterates with fitness < so-far-best-fitness + marFres

marFres [0.0] only relevant if repairOnlyFresBetter==TRUE

Author(s)

Wolfgang Konen, Cologne Univeristy of Applied Sciences

See Also

repairInfeasRI2, repairChootinan

---

defaultSAC *Default settings for the SACOBRA part of COBRA.*

Description

Sets suitable defaults for the SACOBRA part of COBRA.
With the call setOpts(mySAC,defaultSAC()) it is possible to extend a partial list mySAC to a list containing all sac-elements (the missing ones are taken from defaultSAC()).

Usage

```
defaultSAC(DOSAC = 1)
```

Arguments

DOSAC	[0|1|2] 0: COBRA-R settings (turn off SACOBRA), 1: SACOBRA settings, 2: SACOBRA settings with fewer parameters and more online adujustement (aFF and aCF are done parameter free).

Details

For backward compatibility, a logical DOSAC (deprecated) is mapped from FALSE to 0 and from TRUE to 1.

Value

a list with the following elements (the values in parantheses [|] are the values for DOSAC=[0|1|2]):

RS flag for random start algorithm [FALSE|TRUE|TRUE]

RStype type of the function to calculate probability to start the internal optimizer with a random starting point[NA|SIGMOID|CONSTANT] (see function RandomStart in SACOBRA.R)

RSmax maximum probability of a random start when RStype is selected as SIGMOID (see RandomStart in SACOBRA.R) if RStype is CONSTANT then random start is done with a constant probability determined from mean(c(RSmax,RSmin)) [NA|0.3|0.3]

RSmin manimum probability of a random start when RStype is selected as SIGMOID (see RandomStart in SACOBRA.R)[NA|0.05|0.05]

aDRC flag for automatic DRC adjustment [FALSE|TRUE|TRUE]

aFF flag for automatic objective function transformation [FALSE|TRUE|TRUE]

aCF flag for automatic constraint function transformation [FALSE|TRUE|TRUE]

TFRange threshold, if FRange is larger, then apply automatic objective function transformation (see plog). [Inf|1e+05|-1]

TGR threshold, if GRatio is larger, then apply automatic constraint function transformation. GRatio is the ratio "largest GRange / smallest GRange" where GRange is the min-max range of a specific constraint. If TGR < 1, then the transformation is always performed. [Inf|1e+03|-1].

CS threshold unsuccessful iterations for random start algorithm. If CS iterations in a row do not improve the ever-best feasible solution, then perform a restart. [10|10|10]

adaptivePLOG (experimental) flag for objective function transformation with plog, where the parameter pShift is adapted during iterations. [FALSE|FALSE|FALSE]

onlinePLOG flag for online decision making wether use plog or not according to p-effect plog. [FALSE|FALSE|TRUE]

pEffectInit Initial pEffect value when using onlinePLOG. If pEffectInit >= 2 then the initial model is built after plog transformation. [NA|NA|2]

Author(s)

Samineh Bagheri, Cologne Univeristy of Applied Sciences

See Also

cobraInit, cobraPhaseII

---

defaultTR *Default settings for the trust-region part of COBRA.*

Description

Sets default values for the trust-region part cobra$TRlist of COBRA. With the call setOpts(myTR,defaultTR()) it is possible to extend a partial list myTR to a list containing all TR-elements (the missing ones are taken from defaultTR()).

Usage

```
defaultTR()
```

Value

a list with the following elements

radiMin Minimum fraction of the width of the search space to be used as radius of the trust region [c(0:1)]

radiMax Maximum fraction of the width of the search space to be used as radius of the trust region [c(0:1)]

radiInit Initial radius of trust region

forwardRescale *rescaling*

Description

Scale vector x in original space forward to rescaled space (usually $[-1, 1]^d$)

Usage

```
forwardRescale(x, cobra)
```

Arguments

x	a vector in the original input space
cobra	list from cobraInit, we need here
	originalL a vector with lower bounds in original input space
	originalU a vector with upper bounds in original input space
	newlower a number, the rescaled lower bound for all dimensions
	newupper a number, the rescaled upper bound for all dimensions

Value

z, scaled version of vector x

See Also

inverseRescale

getFbest *Return best objective function value*

Description

Return the original objective function value at the best feasible solution

Usage

```
getFbest(cobra)
```

Arguments

cobra	an object of class COBRA (see cobraInit)

Details

Note: We cannot take the best function value via `cobra$fn`, because this may be modified by `plog()` or others)

Value

the original objective function value at the best feasible solution

See Also

`getXbest`

`getXbest` *Return best feasible solution in original space*

Description

Return best feasible solution in original space

Usage

```
getXbest(cobra)
```

Arguments

`cobra`	an object of class COBRA (see `cobraInit`)

Value

the best feasible solution in original space

See Also

`getFbest`

`interpRBF` *Apply cubic or Gaussian RBF interpolation to new data for d>1.*

Description

Apply cubic or Gaussian RBF interpolation to new data for d>1.

Usage

```
interpRBF(x, rbf.model)
```

Arguments

`x`	vector holding a point of dimension d
`rbf.model`	trained RBF model (or set of models), see `trainCubicRBF` or `trainGaussRBF`

Value

value $s(x)$ of the trained model at x
- or -
vector $s_j(x)$ with values for all trained models $j = 1, ..., m$ at x

Author(s)

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

See Also

`trainCubicRBF`, `predict.RBFinter`

`inverseRescale` *inverse rescaling*

Description

Scale vector x in rescaled space back to original space

Usage

```
inverseRescale(x, cobra)
```

Arguments

x	a vector in the rescaled input space (usually $[-1, 1]^d$)
cobra	list from cobraInit, we need here
	originalL a vector with lower bounds in original input space
	originalU a vector with upper bounds in original input space
	newlower a number, the rescaled lower bound for all dimensions
	newupper a number, the rescaled upper bound for all dimensions

Value

z, inverse rescaling of vector x

See Also

forwardRescale

---

multiCOBRA *Perform multiple COBRA runs*

Description

Perform multiple COBRA runs. Each run starts with a different seed so that a different start point, a different initial design and different random restarts are choosen.

Usage

```
multiCOBRA(fn, lower, upper, nrun = 10, feval = 200, funcName = "GXX",
  fName = paste0("mult-", funcName, ".Rdata"), path = NULL, cobra = NULL,
  optim = NULL, target = 0.05, saveRdata = FALSE, ylim = c(1e-05,
  10000), plotPDF = FALSE, startSeed = 41)
```

Arguments

fn	objective function that is to be minimized, should return a vector of the objective function value and the constraint values
lower	lower bound of search space
upper	upper bound of search space
nrun	[10] number of runs
feval	[200] function evaluations per run
funcName	["GXX"] name of the problem
fName	file name .Rdata where the results (dfAll and others) are saved (only if saveRdata==TRUE)
path	[NULL] optional path

cobra	[NULL] list with COBRA settings. If NULL and for elements not present in this list, the defaults from cobraInit are used.
optim	[NULL] the true optimum (or best known value) of the problem
target	[0.05] a single run meets the target, if the final error is smaller than target
saveRdata	[FALSE] if TRUE, save results (dfAll,optim,target,fName,funcName) on fName
ylim	the y limits
plotPDF	[FALSE] if TRUE, plot not only to current graphics device but to <fName>.pdf as well
startSeed	[41]

Details

Side effect: An error plot showing each run and the mean and median of all runs (see multiRunPlot). The results (dfAll and others) are saved to <fName>.Rdata.

Value

mres, a list containing

cobra	the settings and results from **last** run
dfAll	a data frame with a result summary for **all** runs (see below)
z	a vector containing for each run the ever-best feasible objective value
z2	a data frame containing for each run the minimum error (if optim is available)

The data frame dfAll contains one row per iteration with columns (among others)

ffc fitness function calls (i.e. the iterations cobra$iter)

fitVal true fitness function value

fitSur surrogate fitness function value

feas is current iterate feasible on the true constraints?

feval number of evaluations of the internal optimizer on the surrogate functions (NA if it is a repairInfeasible-step)

XI the DRC element used in the current iteration

everBestFeas the ever-best feasible fitness function value

run the number of the current run

X1,X2,... the solution in (original) input space

Author(s)

Wolfgang Konen, Samineh Bagheri, Cologne Univeristy of Applied Sciences

See Also

multiRunPlot, cobraPhaseII

Examples

```
## solve G11 problem nrun times  and plot the results of all nrun runs
nrun=4
feval=30

## Defining the constraint problem: G11
fn <- function(x) {
  y<-x[1]*x[1]+((x[2]-1)^2)
  y<-as.numeric(y)

  g1 <- as.numeric(+(x[2] - x[1]^2))

  return(c(objective=y, g1=g1))
}
funcName="G11"
d=2
nConstraints<-1
lower<-c(-1,-1)
upper<-c(+1,+1)

## Initializing and running cobra
cobra <- cobraInit(xStart=c(0,0), fn=fn, fName=funcName, lower=lower, upper=upper,
                   nConstraints=nConstraints,  feval=feval,
                   initDesPoints=3*d, DOSAC=1,cobraSeed=1)

mres <- multiCOBRA(fn,lower,upper,nrun=nrun,feval=feval,optim=0.75
                  ,cobra=cobra,funcName=funcName
                  ,ylim=c(1e-12,1e-0),plotPDF=FALSE,startSeed=42)

#Two true solutions at x1* = c(-sqrt(0.5),0.5) and x2* = c(+sqrt(0.5),0.5)
#where the true optimum is f(x1*) = f(x2*) = -0.75
print(getXbest(mres$cobra))
print(getFbest(mres$cobra))
print(mres$z2)
```

`multiRunPlot` *Plot the results from multipe COBRA runs.*

Description

Plot for each run one black curve 'error vs. iterations' and aggregate the mean curve (red) and the median curve (green) of all runs. 'error' is the distance between the ever-best feasible value and `optim`.

Usage

```
multiRunPlot(dfAll, optim = NULL, fName = "", main = "", xlim = NULL,
  ylim = c(1e-05, 10000), ylog = TRUE, xlog = FALSE, target = 0.05,
  plotPDF = FALSE, subPDF = NULL, legendWhere = "topright")
```

Arguments

dfAll	the data frame of all runs, obtained with multiCOBRA or loaded from .Rdata file
optim	[NULL] the true optimum (or best known value) of the problem (only for diagnostics). If optim==NULL, we plot instead of errors the ever-best feasible values.
fName	[""] the name of the .Rdata file, printed as subtitle
main	[""] the name of the problem (e.g. "G01 problem"), printed as title
xlim	the x limits
ylim	the y limits
ylog	[TRUE] logarithmic y-axis
xlog	[FALSE] logarithmic x-axis
target	[0.05] a single run meets the target, if the final error is smaller than target
plotPDF	[FALSE] if TRUE, plot to 'fName'.pdf
subPDF	[NULL] optional subdirectory where .pdf should go
legendWhere	["topright"]

Details

Print some diagnostic information: final median & mean error, percentage of runs which meet the target (only if optim is available)).

Value

z3, a vector containing for each run the ever-best feasible objective value

Author(s)

Wolfgang Konen, Samineh Bagheri, Cologne Univeristy of Applied Sciences

See Also

multiCOBRA, cobraPhaseII

---

plog *Monotonic transform*

Description

The function is introduced in [Regis 2014] and extended here by a parameter p_{shift}.
Let $y' = (y - p_{shift})$:

$$plog(y) = \ln(1 + y'), \quad \text{if} \quad y' \geq 0$$

$$plog(y) = -\ln(1 - y'), \quad \text{if} \quad y' < 0$$

Usage

```
plog(y, pShift = 0)
```

Arguments

y	function argument
pShift	shift

Value

$plog(y)$

See Also

plogReverse

---

plogReverse *Inverse of* plog

Description

Inverse of plog

Usage

```
plogReverse(y, pShift = 0)
```

Arguments

y	function argument
pShift	shift

Value

$plog^{-1}(y)$

See Also

plog

predict.RBFinter *Apply cubic or Gaussian RBF interpolation*

Description

Apply cubic or Gaussian RBF interpolation to a set of new data points for d>1.

Usage

```
## S3 method for class 'RBFinter'
predict(rbf.model, newdata, ...)
```

Arguments

rbf.model	trained RBF model (or set of models), see trainCubicRBF or trainGaussRBF
newdata	matrix or data frame with d columns. Each row contains a data point x_i, $i = 1, \ldots, n$
...	(not used)

Value

vector of model responses $s(x_i)$, one element for each data point x_i
- or -
if rbf.model is a set of m models, a (n x m)-matrix containing in each row the response $s_j(x_i)$ of all models $j = 1, \ldots, m$ to x_i

Author(s)

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

See Also

trainCubicRBF, trainGaussRBF, interpRBF

repairChootinan *Repair an infeasible solution with the method of Chootinan.*

Description

Implements the method of [Choo2006] Chootinan & Chen "Constraint handling in genetic algorithms using a gradient-based repair method", Computers & Operations Research 33 (2006), p. 2263.

Usage

```
repairChootinan(x, fReal, rbf.model, cobra, checkIt = FALSE, conFunc = NULL)
```

Arguments

x	an infeasible solution (vector of length dimension)
fReal	a vector of length nconstraint holding the real constraint values at x
rbf.model	the constraint surrogate models
cobra	parameter list, we need here
	lower lower bounds of search region
	upper upper bounds of search region
	ri a list with all parameters for repairChootinan
checkIt	[FALSE] if TRUE, perform a check whether the returned solution is really feasible. Needs access to the true constraint function conFunc
conFunc	[NULL] function returning real constraint vector (needed if checkIt==T)

Value

z, a vector with a repaired (hopefully feasible) solution

Author(s)

Wolfgang Konen, Cologne Univeristy of Applied Sciences

See Also

repairInfeasRI2, cobraPhaseII

---

repairInfeasRI2 *Repair an infeasible solution with the method RI2*

Description

If the solution x is infeasible, i.e. if any(fReal>0)==TRUE:

1. Estimate the gradient of the constraint surrogate function(s) (go a tiny step in each dimension in the direction of constraint increase).
2. Take cobrarimmax random realizations in the 'feasible parallelepiped' and select among them the best feasible solution, based on the surrogates,
3. Check whether the new solution is for every dimension in the bounds [cobra$lower,cobra$upper] of the search region. If not, set the gradient to 0 in these dimensions and re-iterate from step 2.

There is no guarantee but a good chance, that the returned solution z will be feasible.

Usage

```
repairInfeasRI2(x, fReal, rbf.model, cobra, checkIt = FALSE, conFunc = NULL)
```

Arguments

x	an infeasible solution (vector of length dimension)
fReal	a vector of length nconstraint holding the real constraint values at x
rbf.model	the constraint surrogate models
cobra	parameter list, we need here
	lower lower bounds of search region
	upper upper bounds of search region
	ri a list with all parameters for repairInfeasRI2
	trueFuncForSurrogate if TRUE (only for diagnostics), use the true constraint functions conFunc instead of the constraint surrogate models rbf.model
checkIt	[FALSE] if TRUE, perform a check whether the returned solution is really feasible. Needs access to the true constraint function conFunc
conFunc	[NULL] function returning real constraint vector (needed only if checkIt==T or if cobra$trueFuncForSurrogate==T)

Details

For further details see [Koch15a] Koch, P.; Bagheri, S.; Konen, W. et al. "A New Repair Method For Constrained Optimization". Proc. 17th Genetic and Evolutionary Computation Conference (GECCO), 2015.

Value

z, a vector with a repaired (hopefully feasible) solution

Author(s)

Wolfgang Konen, Cologne Univeristy of Applied Sciences

See Also

repairChootinan, cobraPhaseII

---

setOpts *Merge the options from a partial list and the default list*

Description

Merge the options from a partial list and the default list

Usage

```
setOpts(opts, defaultOpt)
```

Arguments

opts	a partial list of options
defaultOpt	a list with default values for every element

Value

a list combined from opts and defaultOpt where every available element in opts overrides the default. For the rest of the elements the value from defaultOpt is taken.
A warning is issued for every element appearing in opts but not in defaultOpt

Author(s)

Samineh Bagheri, Wolfgang Konen, Cologne Univeristy of Applied Sciences

See Also

defaultRI, defaultSAC

---

startCobra *Start COBRA (constraint-based optimization) phase I and/or phase II*

Description

Start COBRA (constraint-based optimization) phase I and/or phase II for object cobra

Usage

```
startCobra(cobra)
```

Arguments

cobra	initialized COBRA object, i.e. the return value from cobraInit

Value

cobra, an object of class COBRA

See Also

cobraInit, cobraPhaseI, cobraPhaseII

Examples

```
## solve G01 problem

## defining the constraint problem: G01
fn<-function(x){
obj<- sum(5*x[1:4])-(5*sum(x[1:4]*x[1:4]))-(sum(x[5:13]))
g1<- (2*x[1]+2*x[2]+x[10]+x[11] - 10)
g2<- (2*x[1]+2*x[3]+x[10]+x[12] - 10)
g3<- (2*x[2]+2*x[3]+x[11]+x[12] - 10)

g4<- -8*x[1]+x[10]
g5<- -8*x[2]+x[11]
g6<- -8*x[3]+x[12]

g7<- -2*x[4]-x[5]+x[10]
g8<- -2*x[6]-x[7]+x[11]
g9<- -2*x[8]-x[9]+x[12]

res<-c(obj, g1 ,g2 , g3
       , g4 , g5 , g6
       , g7 , g8 , g9)
return(res)
}
fName="G01"
d=13
lower=rep(0,d)
upper=c(rep(1,9),rep(100,3),1)
nConstraints=9
set.seed(1)
xStart<-runif(d,min=lower,max=upper)

## Initializing cobra
 cobra <- cobraInit(xStart=xStart, fn=fn, fName=fName, lower=lower, upper=upper,
                    nConstraints=nConstraints, feval=60, seqFeval=500,
                    initDesPoints=3*d, DOSAC=1, cobraSeed=1)

cobra <- startCobra(cobra)
#The solution is at x* = c(rep(1,9),rep(3,3),1)
#where the optimum is f(x*) = -15
print(getXbest(cobra))
print(getFbest(cobra))

## Plot the resulting error (best-so-far feasible optimizer result - true optimum)
## on a logarithmic scale:
fb=cobra$df$Best
fb[1:(which(cobra$df$feasible)[1]-1)] <- NA  # invalidate iterates before the 1st feasible point
plot(fb-(-15),log="y",type="l",ylab="error",xlab="iteration")
```

`trainCubicRBF` *Fit cubic RBF interpolation to training data for d>1.*

Description

The model for a point $z = (z_1, ..., z_d)$ is fitted using n sample points $x_1, ..., x_n$

$$s(z) = \lambda_1 * \Phi(||z - x_1||) + ... + \lambda_n * \Phi(||z - x_n||) + c_0 + c_1 * z_1 + ... + c_d * z_d$$

where $\Phi(r) = r^3$ denotes the cubic radial basis function. The coefficients $\lambda_1, ..., \lambda_n, c_0, c_1, ..., c_d$ are determined by this training procedure. This is for the default case `squares==FALSE`. In case `squares==TRUE` there are d additional pure square terms and the model is

$$s_{sq}(z) = s(z) + c_{d+1} * z_1^2 + ... + c_{d+d} * z_d^2$$

Usage

```
trainCubicRBF(xp, U, squares = FALSE, rho = 0)
```

Arguments

`xp`	n points x_i of dimension d are arranged in (n x d) matrix `xp`
`U`	vector of length n, containing samples $u(x_i)$ of the scalar function u to be fitted - or - (n x m) matrix, where each column 1,...,m contains one vector of samples $u_j(x_i)$ for the m'th model, j=1,...,m
`squares`	[FALSE] flag, see description
`rho`	[0.0] experimental: 0: interpolating, >0, approximating (spline-like) Gaussian RBFs

Details

The linear equation system is solved via SVD inversion. Near-zero elements in the diagonal matrix D are set to zero in D^{-1}. This is numerically stable for rank-deficient systems.

Value

`rbf.model`, an object of class `RBFinter`, which is basically a list with elements:

`coef`	(n+d+1 x m) matrix holding in column m the coefficients for the m'th model: $\lambda_1, ..., \lambda_n, c_0, c_1, ..., c_d$. In case `squares==TRUE` it is an (n+2d+1 x m) matrix holding additionally the coefficients $c_{d+1}, ..., c_{d+d}$.
`xp`	matrix xp
`d`	dimension d
`npts`	number n of points x_i
`squares`	TRUE or FALSE
`type`	"CUBIC"

Author(s)

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

See Also

trainGaussRBF, predict.RBFinter, interpRBF

`trainGaussRBF` *Fit Gaussian RBF model to training data for d>1.*

Description

The model for a point $z = (z_1, ..., z_d)$ is fitted using n sample points $x_1, ..., x_n$

$$s(z) = \lambda_1 * \Phi(||z - x_1||) + ... + \lambda_n * \Phi(||z - x_n||) + c_0 + c_1 * z_1 + ... + c_d * z_d$$

where $\Phi(r) = \exp(-r^2/(2 * \sigma^2))$ denotes the Gaussian radial basis function with width σ. The coefficients $\lambda_1, ..., \lambda_n, c_0, c_1, ..., c_d$ are determined by this training procedure. This is for the default case `squares==FALSE`. In case `squares==TRUE` there are d additional pure square terms and the model is

$$s_{sq}(z) = s(z) + c_{d+1} * z_1^2 + ... + c_{d+d} * z_d^2$$

Usage

```
trainGaussRBF(xp, U, width, squares = FALSE, RULE = "One",
  widthFactor = 1, rho = 0)
```

Arguments

`xp`	n points x_i of dimension d are arranged in (n x d) matrix `xp`
`U`	vector of length n, containing samples $u(x_i)$ of the scalar function u to be fitted - or - (n x m) matrix, where each column 1,...,m contains one vector of samples $u_j(x_i)$ for the m'th model, j=1,...,m
`width`	[-1] either a positive real value which is the constant width σ for all Gaussians in all iterations, or -1. Then the appropriate width σ is calculated anew in each iteration with one of the rules `RULE`, based on the distribution of data points `xp`
`squares`	[FALSE] flag, see description
`RULE`	["One"] one out of ["One" | "Two" | "Three"], different rules for automatic width estimation.
`widthFactor`	[1.0] additional constant factor applied to each width σ
`rho`	[0.0] experimental: 0: interpolating, >0, approximating (spline-like) Gaussian RBFs

Details

The linear equation system is solved via SVD inversion. Near-zero elements in the diagonal matrix D are set to zero in D^{-1}. This is numerically stable for rank-deficient systems.

Value

`rbf.model`, an object of class `RBFinter`, which is basically a list with elements:

`coef`	(n+d+1 x m) matrix holding in column m the coefficients for the m'th model: $\lambda_1, ..., \lambda_n, c_0, c_1, ..., c_d$. In case `squares==TRUE` it is an (n+2d+1 x m) matrix holding additionally the coefficients $c_{d+1}, ..., c_{d+d}$.
`xp`	matrix xp
`d`	dimension d
`npts`	number n of points x_i
`squares`	TRUE or FALSE
`width`	
`type`	"GAUSS"

Author(s)

Wolfgang Konen, Samineh Bagheri

See Also

`trainCubicRBF`, `predict.RBFinter`, `interpRBF`

`trustRegion` *Perform trust region refinement*

Description

If `cobra$TrustRegion==TRUE` (see `cobraInit`), then `trustRegion` is called after every iteration in order to refine the best solution so far. This function builds a local model around the best solution and runs a local search in the trust region to refine the best solution and find a better solution in the neighborhood.

Usage

```
trustRegion(cobra)
```

Arguments

`cobra`	an object of class `cobra`, which is basically a list (see `cobraInit`)

Value

the modified cobra with new/updated elements

TRDONE logical, is TRUE if there are more than d+1 points in the trusted region and thus surrogates can be trained. Otherwise FALSE.

refinedX if TRDONE==TRUE the refined solution from the trust-region run, otherwise NA

If TRDONE==TRUE the relevant lists and counters (A,Fres,df,...) of cobra will be updated in cobraPhaseII as well.

Author(s)

Samineh Bagheri (<samineh.bagheri@fh-koeln.de>)

Index