Package ‘SPOT’

February 19, 2015

Maintainer Martin Zaefferer <martin.zaefferer@gmx.de>
License GPL (>= 2)
Title Sequential Parameter Optimization
Type Package
LazyLoad yes
Author T. Bartz-Beielstein with contributions from: J. Ziegenhirt, W. Konen, O. Flasch, M. Friese, P. Koch, M. Zaefferer, B. Naujoks
Description R-Package for Sequential Parameter Optimization Toolbox
Version 1.0.4184
URL http://www.gm.fh-koeln.de/campus/personen/lehrende/thomas.bartz-beielstein/00489/
Date 26.06.2013
Depends R (>= 2.14.0), rpart, emoa
Suggests AlgDesign, BB, cmaes, DEoptim, dfoptim, DiceKriging, DoE.wrapper, earth, fields, FrF2, GenSA, ggplot2, gtools, lattice, lhs, lm4, MASS, mco, minqa, mlegp, monmlp, nloptr, kernlab, penalizedSVM, pso, qrnn, randomForest, rgenoud, rgl, rgp (>= 0.3-4), rsm, tgp, twiddler
Collate 'sms_emoa.r' 'spot.R' 'spotCreateDesignBasicDoe.R'
'spotCreateDesignFrF2.R' 'spotCreateDesignLhs.R'
'spotCreateDesignLhsOpt.R' 'spotGenerateSequentialDesign.R' 'spotGetOptions.R'
'spotMcoSort.R' 'spotMeta.R' 'spotOcba.R' 'spotOptim.R'
'spotPrepareData.R' 'spotRepeats.R' 'spotAlgStartEs.R'
'spotAlgStartRgp.R' 'spotAlgStartSann.R' 'spotComparison.R'
'spotEnsembleSingleBLAbern.R' 'spotEnsembleSingleBLAnorm.R'
'spotEnsembleSingleEpGreedy.R' 'spotEnsembleSinglePOKER.R'
'spotEnsembleSingleRoundSearch.R' 'spotEnsembleSingleSoftMax.R'
'spotEnsembleSingleUCB1.R' 'spotExplainInterfaces.R'
'spotFuncStartBranin.R' 'spotFuncStartMexicanHat.R'
'spotFuncStartRastrigin.R' 'spotFuncStartRosenbrock.R'
'spotFuncStartSixHump.R' 'spotFuncStartSphere.R'
R topics documented:

'spotFuncTools.R' 'spotGui.R' 'spotHelpFunctions.R'
'spotOptimizationFunctions.R' 'spotPredictCoForrester.R'
'spotPredictDace.R' 'spotPredictDice.R' 'spotPredictEarth.R'
'spotPredictEnsembleMultiAlternate.R'
'spotPredictEnsembleMultiAverage.R'
'spotPredictEnsembleMultiChoose.R'
'spotPredictEnsembleMultiRank.R'
'spotPredictEnsembleMultiRankWeighted.R' 'spotPredictEsvm.R'
'spotPredictForrester.R' 'spotPredictGausspr.R'
'spotPredictKrig.R' 'spotPredictKsvm.R' 'spotPredictLm.R'
'spotPredictLmFactor.R' 'spotPredictMlegp.R' 'spotPredictMLP.R'
'spotPredictQrnn.R' 'spotPredictRandomForest.R'
'spotPredictTgp.R' 'spotPredictTree.R'
'spotSeqDesignIncreaseFunctions.R' 'spotStatistics.R'
'spotTargetFunctions.R' 'spotPlot.R' 'spotReport3d.R'
'spotEnsembleFunctions.R' 'spotExpectedImprovement.R'
'spotModelSelection.R' 'spotPredictHelpFunctions.R'
'spotPredictRandomForestMlegp.R' 'spotPredictTest.R'
'spotPredictMCO.R' 'spotReportMAMP.R' 'spotReportSAMP.R'
'spotAlgStartEsGlg.R' 'spotModelParetoOptim.R'
'spotModelOptim.R' 'spotCreateDesignFactors.R'
'spotGaussianLandscapeGenerator.R' 'spotAlgStartSmsEmoaGlg.R'
'spotOptimEs.R'

NeedsCompilation no

Repository CRAN

Date/Publication 2013-06-26 14:52:50

R topics documented:

SPOT-package .................................................. 5
daceBuilder .................................................. 6
dacePredictor .................................................. 7
forrBuilder .................................................... 9
forrCoBuilder ................................................ 11
forrCoRegPredictor ......................................... 13
forrRegPredictor ........................................... 14
forrReintPredictor ......................................... 15
spot .......................................................... 16
spotAlgEs ...................................................... 17
spotAlgStartEs .............................................. 18
spotAlgStartEsGlg .......................................... 19
spotAlgStartEsVar .......................................... 20
spotAlgStartRgp ............................................ 21
spotAlgStartSann .......................................... 22
spotAlgStartSannVar ....................................... 23
R topics documented:

spotAlgStartSmsEmoaGlg ........................................... 24
spotCreateDesign.................................................. 25
spotCreateDesignBasicDoe .................................. 25
spotCreateDesignFactors .................................. 26
spotCreateDesignFrF2 .................................... 27
spotCreateDesignLhd ..................................... 27
spotCreateDesignLhs .................................... 28
spotCreateDesignLhsOpt .................................. 29
spotEnsembleMultiAlternate .................................. 30
spotEnsembleMultiAverage .................................. 31
spotEnsembleMultiChoose .................................. 32
spotEnsembleMultiRank ..................................... 33
spotEnsembleMultiRankWeighted ......................... 34
spotEnsembleSingleBLAbern ................................ 36
spotEnsembleSingleBLAnorm ................................ 37
spotEnsembleSingleEpsGreedy ................................ 38
spotEnsembleSinglePOKER .................................. 39
spotEnsembleSingleRoundSearch ......................... 41
spotEnsembleSingleSoftMax ................................ 42
spotEnsembleSingleUCB1 .................................. 43
spotFeedback .................................................. 44
spotGetOptions .................................................. 45
spotGlgCreate .................................................... 49
spotGlgCreateN ................................................... 50
spotGlgCreateRot ............................................ 51
spotGlgCreateRotSearched .................................. 52
spotGlgEval ..................................................... 53
spotGlgEvalN ..................................................... 54
spotGlgEvalRot .................................................. 55
spotGlgInit ....................................................... 56
spotGlgInitN ....................................................... 57
spotGui ......................................................... 58
spotHelpInterfaces.............................................. 58
spotInfillExpImp ............................................... 59
spotInfillHyperVolume ....................................... 59
spotInfillLcbHyperVolume ................................... 60
spotInfillLcbMulti ............................................... 61
spotInfillLcbSingle ............................................. 61
spotInfillProbImp ............................................... 62
spotInfillSD ...................................................... 62
spotInfillSExI2d ............................................... 63
spotModel.func .................................................. 64
spotModel.predict ............................................... 65
spotModel.train .................................................. 66
spotModelDescentLm ......................................... 67
spotOcba .......................................................... 67
spotModelOptim ............................................... 69
spotModelParetoOptim ...................................... 69
spotOcba .......................................................... 70
R topics documented:

- spotOptim
- spotOptimEs
- spotOptimInterface
- spotOptimizationInterface
- spotOptimizationInterfaceMco
- spotOptimLHS
- spotPredict...
- spotPredictCoForrester
- spotPredictDace
- spotPredictDice
- spotPredictEarth
- spotPredictEsvm
- spotPredictForrester
- spotPredictGausspr
- spotPredictKrig
- spotPredictKsvm
- spotPredictLm
- spotPredictLmFactor
- spotPredictMCO
- spotPredictMlegp
- spotPredictMLP
- spotPredictQrnn
- spotPredictRandomForest
- spotPredictRandomForestMlegp
- spotPredictTgp
- spotPredictTree
- spotReadBstFile
- spotReadRoi
- spotReport3d
- spotReportContour
- spotReportDefault
- spotReportEarth
- spotReportMAMP
- spotReportMetaDefault
- spotReportSAMP
- spotReportSens
- spotROI
- spotSelectionCriteria
- spotSExI2d
- spotSmsEmoa
- spotSmsEmoaKriging
- spotStepAutoOpt
- spotStepInitial
- spotStepMetaOpt
- spotStepReport
- spotStepRunAlg
- spotStepSequential
- spotSurf3d
Description

Sequential Parameter Optimization Toolbox in R

Details

SPOT is a package for R, using statistic models to find optimal parameters for optimization algorithms. SPOT is a very flexible and user oriented tool box for parameter optimization. The flexibility has its price: to fully use all the possibilities of flexibility the user is requested to look at a large number of spot-parameters to change. The good news is, that some defaults are given that already work perfectly well for 90 percent of the users.

<table>
<thead>
<tr>
<th>Package</th>
<th>SPOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Package</td>
</tr>
<tr>
<td>Version</td>
<td>1.0.4184</td>
</tr>
<tr>
<td>Date</td>
<td>26.06.2013</td>
</tr>
<tr>
<td>License</td>
<td>GPL (&gt;= 2)</td>
</tr>
<tr>
<td>LazyLoad</td>
<td>yes</td>
</tr>
</tbody>
</table>

Author(s)

Thomas Bartz-Beielstein <thomas.bartz-beielstein@fh-koeln.de> with contributions from: J. Ziegenhirt, W. Konen, O. Flasch, M. Friese, P. Koch, M. Zaefferer, B. Naujoks

References

http://www.gm.fh-koeln.de/campus/personen/lehrende/thomas.bartz-beielstein/00489/
http://www.springer.com/3-540-32026-1

See Also

Main interface functions are spot and spotOptim. Also, a graphical interface can be used with spotGui
Description

This Kriging meta model is based on DACE (Design and Analysis of Computer Experiments). It allows to choose different regression and correlation models. The optimization of model parameters is by default done with a bounded simplex method from the nloptr package.

Usage

daceBuilder(parameters, objectives, startTheta, tol = 1e-06, budget = 100, regr = regpoly0, corr = corrnoisykriging, nugget = -1, algtheta = "optim-L-BFGS-B")

Arguments

parameters known design points. That is, a matrix with \( n \) rows (for each point) and \( \text{dim} \) columns (for each dimension).
objectives vector of observations at known design points of length \( n \).
startTheta vector which contains initial guess for model parameters \( \theta \). Initial guess will be set depending on correlation function if this vector is missing.
tol Tolerance stopping criterion for the simplex MLE. Default is \( 1e^{-6} \).
budget Number of Likelihood evaluations, as a stopping criterion for the simplex MLE. Default is Inf. The value will be multiplied with the length of the model parameter vector to be optimized.
regr Regression function to be used: \( \text{regpoly0} \) (default), \( \text{regpoly1} \), \( \text{regpoly2} \). Can be a custom user function.
corr Correlation function to be used: \( \text{corrnoisykriging} \) (default), \( \text{corrkriing} \), \( \text{corrnoisygauss} \), \( \text{corrgauss} \), \( \text{correxpg} \), \( \text{correxpg} \), \( \text{corrlin} \), \( \text{corrcubic} \), \( \text{carrspherical} \), \( \text{corrspline} \). Can also be user supplied (if in the right form).
nugget Value for nugget. Default is -1, which means the nugget will be optimized during MLE. Else it can be fixed in a range between 0 and 1.
algtheta algorithm used to find \( \theta \), default is "optim-L-BFGS-B". Else, any from the list of possible method values in \text{spotOptimizationInterface} can be chosen.

Value

returns a list with the following elements:

model Again a list, containing model parameters
like Likelihood value
theta activity parameters \( \theta \) (vector)
dacePredictor

\[ p \] exponents \( p \) (vector)
\[ \lambda \] nugget value (numeric)
\[ \text{nevals} \] Number of iterations during MLE

Author(s)

The authors of the original DACE Matlab toolbox [http://www2.imm.dtu.dk/~hbn/dace/](http://www2.imm.dtu.dk/~hbn/dace/) are Hans Bruun Nielsen <hbn@imm.dtu.dk>, Soren Nymand Lophaven and Jacob Sondergaard. Extension of the Matlab code by Tobias Wagner <wagner@isf.de>. Porting and adaptation to R and further extensions by Martin Zaefferer <martin.zaefferer@fh-koeln.de>.

References


See Also

spotPredictDace dacePredictor

Examples

```r
## Create design points
x = cbind(runif(20)*15-5,runif(20)*15)
## Compute observations at design points (for Branin function)
y = apply(x,1,spotBraninFunction)
## Create model with default settings
fit = daceBuilder(x,y)
## Print model parameters
print(fit)
## Create with different regression and correlation functions
fit = daceBuilder(x,y,regr=regpoly2,corr=corrspline)
## Print model parameters
print(fit)
```

---

dacePredictor  \hspace{1cm} DACE predictor

Description

Predicts \( y(x) \) for a given DACE model (i.e. as created by \texttt{daceBuilder}). The user can choose whether to predict only mean or if he is also interested in gradient, mean squared error MSE, or the MSE gradient.

Usage

```r
dacePredictor(x, fit, GRAD = FALSE, MSE = FALSE, GRADMSE = FALSE)
```
Arguments

x mx candidate points of dimension n to be predicted. That is, a matrix with mx rows and n columns.

fit the model fit, as returned by daceBuilder

GRAD specify whether gradient of response should be computed (default is FALSE, which means no). Even if GRAD is TRUE, the gradient will only be computed in case of a single design point mx==1.

MSE specify whether estimated MSE of response should be computed (default is FALSE, which means no)

GRADMSE specify whether gradient of MSE should be computed (default is FALSE, which means no). Even if GRADMSE is TRUE, the gradient will only be computed in case of a single design point mx==1.

Value

returns a list with the following elements:

f Predicted response y at design points x (always)

df Gradient of response y at design points x (only if: GRAD==TRUE and mx==1)

s Estimated MSE (only if: MSE==TRUE)

ds Gradient of MSE (only if: GRADMSE==TRUE and mx==1)

Author(s)

The authors of the original DACE Matlab toolbox http://www.imm.dtu.dk/~hbn/dace/ are Hans Bruun Nielsen <hbn@imm.dtu.dk>, Soren Nymand Lophaven and Jacob Sondergaard. Additional code for generalization to different models by Tobias Wagner <wagner@isf.de>. Porting and adaptation to R and further extensions by Martin Zaefferer <martin.zaefferer@fh-koeln.de>.

References


See Also

spotPredictDace daceBuilder

Examples

```r
## Create design points
x = cbind(runif(20)*15-5,runif(20)*15)
## Compute observations at design points (for Branin function)
y = apply(x,1,spotBraninFunction)
## Create model
fit = daceBuilder(x,y)
## Create candidate design points
```
forrBuilder

Description

This function builds a Kriging model based on code by Forrester et al. By default exponents (p) are fixed at a value of two, and a nugget (or regularization constant) is used.

Usage

```r
forrBuilder(X, y, loval = 0.001, upval = 100,
            algtheta = "optim-L-BFGS-B", budgetalgtheta = 100,
            lb = NULL, ub = NULL, opt.p = FALSE, lambda.loval = -6,
            lambda.upval = 0)
```

Arguments

- **X**: design matrix (sample locations)
- **y**: vector of observations at X
- **loval**: lower boundary for theta, default is 1e-3
- **upval**: upper boundary for theta, default is 100
- **algtheta**: algorithm used to find theta, default is "optim-L-BFGS-B". Else, any from the list of possible method values in `spotOptimizationInterface` can be chosen.
- **budgetalgtheta**: budget for the above mentioned algorithm, default is 100. The value will be multiplied with the length of the model parameter vector to be optimized.
- **lb**: lower boundary of the design space. Will be extracted from the matrix X if not given.
- **ub**: upper boundary of the design space. Will be extracted from the matrix X if not given.
- **opt.p**: boolean that specifies whether the exponents (p) should be optimized. Else they will be set to two. Default value is FALSE. Default is highly recommended as the implementation of this feature is not yet well tested and might be faulty.
- **lambda.loval**: lower boundary for regularization constant (nugget), default is -6. (lambda=10^\lambda, e.g. 10^{-6})
- **lambda.upval**: upper boundary for regularization constant (nugget), default is 0. (lambda=10^\lambda, e.g. 1)
Value

a fit (list), with the options and found parameters for the model which has to be passed to the
predictor function:
X sample locations (scaled to values between 0 and 1)
y observations at sample locations (see parameters)
loval lower boundary for theta (see parameters)
upval upper boundary for theta (see parameters)
algteta algorithm to find theta (see parameters)
budgetalgtheta budget for the above mentioned algorithm (see parameters)
optrp boolean that specifies whether the exponents (p) were optimized (see parameters)
normalizeymin minimum in normalized space
normalizeymax maximum in normalized space
normalizexmin minimum in input space
normalizexmax maximum in input space
dmodeltheta vector of activity parameters
Theta log_10 vector of activity parameters (i.e. log10(dmodeltheta))
dmodellambda regularization constant (nugget)
Lambda log_10 of regularization constant (nugget) (i.e. log10(dmodellambda))
yonemu Ay-ones*mu
ssq sigma square
mu mean mu
Psi matrix large Psi
Psinv inverse of Psi
nevals number of Likelihood evaluations during MLE

References

Forrester, Alexander I.J.; Sobester, Andras; Keane, Andy J. (2008). Engineering Design via Surro-

See Also

spotPredictForrester forrRegPredictor forrReintPredictor

Examples

```r
## Create design points
x = cbind(runif(20)*15-5,runif(20)*15)
## Compute observations at design points (for Branin function)
y = as.matrix(apply(x,1,spotBraninFunction))
## Create model with default settings
fit = forrBuilder(x,y)
## Print model parameters
print(fit)
```
forrCoBuilder

Build Forrester Co-Kriging

Description

This function builds a Co-Kriging model based on code by Forrester et al. Please note that the expensive sample locations should be contained in the cheap sample locations. Furthermore, it has to be made sure that the correlated functions do not yield identical values. That is, ye and yc should have common sample locations, but different values. The sample locations only evaluated on the cheap function can be chosen arbitrarily.

Usage

```r
forrCoBuilder(Xe, ye, Xc, yc, fitC, loval = 0.001,
            upval = 100, algtheta = "optim-L-BFGS-B",
            budgetAlgtheta = 100, lb = NULL, ub = NULL,
            opt.p = FALSE, lambda.loval = -6, lambda.upval = 0,
            rho.loval = -5, rho.upval = 5)
```

Arguments

- **Xe**: design matrix (expensive sample locations)
- **ye**: 1-row matrix of expensive observations at Xe
- **Xc**: design matrix (cheap sample locations). The bottom of this matrix should contain expensive samples.
- **yc**: 1-row matrix of cheap observations at Xc.
- **fitC**: object of class forr, containing a Kriging model build through the cheap observations
- **loval**: lower boundary for theta, default is 1e-3
- **upval**: upper boundary for theta, default is 100
- **algtheta**: algorithm used to find theta, default is "optim-L-BFGS-B". Else, any from the list of possible method values in `spOptimizationInterface` can be chosen.
- **budgetAlgtheta**: budget for the above mentioned algorithm, default is 100. The value will be multiplied with the length of the model parameter vector to be optimized.
- **lb**: lower boundary of the design space. Will be extracted from the matrix Xe if not given.
- **ub**: upper boundary of the design space. Will be extracted from the matrix Xe if not given.
- **opt.p**: boolean that specifies whether the exponents (p) should be optimized. Else they will be set to two. Default value is FALSE. Default is highly recommended as the implementation of this feature is not yet well tested and might be faulty.
- **lambda.loval**: lower boundary for regularization constant (nugget), default is -6. (lambda=10^lambda, e.g. 10^-6)
lambda.upval  upper boundary for regularization constant (nugget), default is 0. (lambda=10^lambda, e.g. 1)
rho.loval   lower boundary for rho, default is -5.
rho.upval   upper boundary for rho, default is 5.

Value

a fit (list) of class coforr. This contains Co-Kriging specific parameters, as well as two fits of class forr which represent the cheap and expensive models.

References


See Also

spotPredictCoForrester forrCoRegPredictor forrBuilder

Examples

## This is the one-variable example described by Forrester et al.
## The "expensive" function to be modeled is
ovar <- function(x){(x*6-2)^2*sin((x*6-2)*2)}
## The "cheap" function to be modeled is
covar <- function(x){A=0.5;B=10;C=-5;D=0;
A*(((x+D)*6-2)^2)*sin(((x+D)*6-2)^2)+((x+D)-0.5)*B+C}
## construct cheap and expensive sample locations
xe <- rbind(0,0.4,0.6,1)
xc <- rbind(0.1,0.2,0.3,0.5,0.7,0.8,0.9,0,0.4,0.6,1)
## get observations of samples
ye <- rbind(ovar(xe))
yc <- rbind(covar(xc))
## build the Co-Kriging model, with cheap and expensive observations
set.seed(2)
fitC <- forrBuilder(xc, yc, 1e-3, 1e2, "optim-L-BFGS-B", 100,0,1,TRUE);
fit <- forrCoBuilder(xe, ye, xc, yc, fitC, 1e-3, 1e2, "optim-L-BFGS-B", 100,0,1,TRUE)
## build the ordinary Kriging model with expensive observations only
fit1 <- forrBuilder(xe, ye, 1e-3, 1e2, "optim-L-BFGS-B", 100,0,1,TRUE)
## Predict and plot over whole design space
x=seq(from=0,to=1,by=0.01)
yco <- forrCoRegPredictor(as.matrix(x),fit,TRUE)
ypc <- forrRegPredictor(as.matrix(x),fitC,TRUE)
ype <- forrRegPredictor(as.matrix(x),fit,TRUE)
yy <- forrRegPredictor(as.matrix(x),fit1,TRUE)
plot(x,ovar(x),type="l",ylim=c(-15,20),lwd=3)
Predict new samples on a Forrester Co-Kriging model.

**Usage**

```r
forrcoregpredictor(x, fit, pred.all = FALSE)
```

**Arguments**

- `x`: design matrix to be predicted
- `fit`: fit of the Co-Kriging model (settings and parameters), as created by `forrcobuilder`
- `pred.all`: if TRUE return all (RMSE and prediction, in a dataframe), else return only prediction

**Value**

Returned value is dependent on the setting of `pred.all`
- TRUE: data.frame with columns `f` (function values) and `s` (RMSE)
- FALSE: vector of function values only

**References**


**See Also**

`forrcobuilder`
Predict Forrester Model

Description

Predict samples on a Forrester Kriging model.

Usage

```r
forrRegPredictor(x, ModelInfo, pred.all = FALSE)
```

Arguments

- `x`: design matrix to be predicted
- `ModelInfo`: fit of the Kriging model (settings and parameters)
- `pred.all`: if TRUE return all (RMSE and prediction, in a dataframe), else return only prediction

Value

Returned value is dependent on the setting of `pred.all`
- TRUE: data.frame with columns f (function values) and s (RMSE)
- FALSE: vector of function values only

See Also

`forrBuilder` `forrReintPredictor`

Examples

```r
## Create design points
x = cbind(runif(20)*15-5,runif(20)*15)
## Compute observations at design points (for Branin function)
y = as.matrix(apply(x,1,spotBraninFunction))
## Create model
fit = forrBuilder(x,y)
## Create candidate design points
xx = cbind(runif(20)*15-5,runif(20)*15)
## Predict candidates
y1 = forrRegPredictor(xx,fit)$f
## Plot model (in comments, due to time consumption)
#fn <- function(x)(forrRegPredictor(as.matrix(x),fit)$f)
#spotSurf3d(fn,c(-5,0),c(10,15))
## Plot real function
#spotSurf3d(function(x){apply(x,1,spotBraninFunction)},c(-5,0),c(10,15))
```
forrReintPredictor  
*Predict Forrester Model (Re-interpolating)*

**Description**

Kriging predictor with re-interpolation to avoid stalling the optimization process which employs this model as a surrogate. This is supposed to be used with deterministic experiments, which do need a non-interpolating model that avoids predicting non-zero error at sample locations. This can be useful when the model is deterministic (i.e. repeated evaluations of one parameter vector do not yield different values) but does have a "noisy" structure (e.g. due to computational inaccuracies, systematical error).

**Usage**

```r
forrReintPredictor(x, ModelInfo, pred.all = FALSE)
```

**Arguments**

- `x` design matrix to be predicted
- `ModelInfo` fit of the Kriging model (settings and parameters)
- `pred.all` if TRUE return all (RMSE and prediction, in a dataframe), else return only prediction

**Details**

Please note that this re-interpolation implementation will not necessarily yield values of exactly zero at the sample locations used for model building. Slight deviations can occur.

**Value**

Returned value is dependent on the setting of `pred.all`
- TRUE: data.frame with columns `f` (function values) and `s` (RMSE)
- FALSE: vector of function values only

**See Also**

`forrBuilder`, `forrCoBuilder`, `forrRegPredictor`

**Examples**

```r
## Create design points
dx = cbind(runif(20)*15-5,runif(20)*15)
## Compute observations at design points (for Branin function)
y = as.matrix(apply(x,1,spotBraninFunction))
## Create model
fit = forrBuilder(x,y)
## first estimate error with regressive predictor
sreg = forrRegPredictor(x,fit,TRUE)$s
```
## now estimate error with re-interpolating predictor
sreint = forReintPredictor(x, fit, TRUE)$
print(sreg)
print(sreint)
## sreint should be close to zero, significantly smaller than sreg

---

**spot**

*Main function for the use of SPOT*

### Description

Sequential Parameter Optimization Toolbox (SPOT) provides a toolbox for the sequential optimization of parameter driven tasks. Use `spotOptim` for a `optim` like interface.

### Usage

```r
spot(configFile = "NULL", spotTask = "auto",
     srcPath = NA, spotConfig = NA, ...)
```

### Arguments

- **configFile**: the absolute path including file-specifier, there is no default, this value should always be given.
- **spotTask**: [init|seq|run|auto|rep] the switch for the tool used, default is "auto".
- **srcPath**: the absolute path to user written sources that extend SPOT, the default(NA) will search for sources in the path `<.libPath()>/SPOT/R`.
- **spotConfig**: a list of parameters used to configure spot, default is spotConfig=NA, which means the configuration will only be read from the configFile, not given by manual user input. Notice that parameters given in spotConfig will overwrite both default values assigned by SPOT, AND values defined in the Config file. However, values not passed by spotConfig will still be used as defaults. If you want to see those defaults, look at `spotGetOptions`.
- **...**: additional parameters to be passed on to target function which is called inside `alg.func`. Only relevant for `spotTask "auto"` and "run".

### Details

The path given with the `userConfigFile` also fixes the working directory used throughout the run of all SPOT functions. All files that are needed for input/output can and will be given relative to the path of the `userConfigFile` (this also holds for the binary of the algorithm). This refers to files that are specified in the `configFile` by the user.

It is of major importance to understand that `spot` by default expects to optimize noisy functions. That means, the default settings of `spot`, which are also used in `spotOptim`, include repeats of the initial and sequentially created design points. Also, as a default OCBA is used to spread the design points for optimal usage of the function evaluation budget. OCBA will not work when there is no variance in the data. So if the user wants to optimize non-noisy functions, the following settings
should be used:
spotConfig$spot.ocba <- FALSE
spotConfig$seq.design.maxRepeats <- 1
spotConfig$init.design.repeats <- 1

Note

spot() expects char vectors as input, e.g. spot("c:/configfile.conf","auto")

See Also

SPOT spotOptim spotStepAutoOpt spotStepInitial spotStepSequential spotStepRunAlg
spotStepReport spotPrepare spotPrepareSystem

spotAlgEs

Evolution Strategy Implementation

Description

This function is used by spotAlgStartEs as a main loop for running the Evolution Strategy with
the given parameter set specified by SPOT.

Usage

spotAlgEs(mue = 10, nu = 10, dimension = 2, mutation = 2,
  sigmaInit = 1, nSigma = 1, tau0 = 0, tau = 1,
  rho = "bi", sel = -1, stratReco = 1, objReco = 2,
  maxGen = Inf, maxIter = 100, seed = 1, noise = 0,
  fName = spotBraninFunction, lowerLimit = -1,
  upperLimit = 1, verbosity = 0, plotResult = FALSE,
  logPlotResult = FALSE, term = "iter",
  sigmaRestart = 0.1, preScanMult = 1,
  globalOpt = rep(0, dimension), ...)

Arguments

mue number of parents, default is 10
nu number, default is 10
dimension dimension number of the target function, default is 2
mutation mutation type, either 1 or 2, default is 1
sigmaInit initial sigma value (standard deviation), default is 1.0
nSigma number of standard deviations, default is 1
tau0 number, default is 0.0. tau0 is the general multiplier.
tau number, learning parameter for self adaption, default is 1.0. tau is the local
multiplier for step sizes (for each dimension).
rho  
  number of parents involved in the procreation of an offspring (mixing number), default is "bi"

sel  
  number of selected individuals, default is 1

stratReco  
  value, Recombination operator for strategy variables, default is 1

objReco  
  value, Recombination operator for object variables, default is 2

maxGen  
  number of generations, stopping criterion, default is Inf

maxIter  
  number of iterations, stopping criterion, default is 100

seed  
  number, random seed, default is 1

noise  
  number, value of noise added to fitness values, default is 0.0

fName  
  function, fitness function, default is spotBraninFunction

lowerLimit  
  number, lower limit for search space, default is -1.0

upperLimit  
  number, upper limit for search space, default is 1.0

verbosity  
  defines output verbosity of the ES, default is 0

plotResult  
  boolean, asks if results are plotted, default is FALSE

logPlotResult  
  boolean, asks if plot results should be logarithmic, default is FALSE

term  
  string, which termination criterion should be used, default is "iter"

sigmaRestart  
  number, value of sigma on restart, default is 0.1

preScanMult  
  initial population size is multiplied by this number for a pre-scan, default is 1

globalOpt  
  termination criterion on reaching a desired optimum value, default is rep(0, dimension)

...  
  additional parameters to be passed on to fName

See Also

SPOT spotAlgStartEs

spotAlgStartEs  
  Interface for an Evolution Strategy to be tuned by SPOT...

Description

This Evolution Strategy implementation can be tuned by SPOT. The ES can use different fitness functions. The results are written to the res file. This function is needed as an interface, to ensure the right information are passed from SPOT to the target algorithm (e.g. the ES) and vice versa.

Usage

spotAlgStartEs(spotConfig)
Arguments

spotConfig

Contains the list of spot configurations, results of the algorithm can be passed to this list instead of the .res file. spotConfig defaults to "NA", and will only be passed to the Algorithm if spotConfig$spot.fileMode=FALSE. See also: spotGetOptions

Items used are:

- alg.currentDesign: data frame holding the design points that will be evaluated
- io.apdFileName: name of the apd file
- io.desFileName: name of the des file
- io.resFileName: name of the res file, for logging results (if spotConfig$spot.fileMode==TRUE)
- spot.fileMode: boolean, if selected with true the results will also be written to the res file, otherwise it will only be saved in the spotConfig returned by this function

Value

this function returns the spotConfig list with the results in spotConfig$alg.currentResult

See Also

SPOT spotAlgEs spotAlgStartEsVar

Algorithm Interface to ES + GLG

Description

Interface for mixed model tuning of an Evolution Strategy ES optimizing Gaussian Landscapes. For each repeat of an ES parameter setting, a different seed for the ES is used. At the same time, this seed is used to create different Gaussian Landscapes with the Gaussian Landscape Generator GLG, see spotGlgCreate.

Usage

spotAlgStartEsGlg(spotConfig)

Arguments

spotConfig

Contains the list of spot configurations, results of the algorithm can be passed to this list instead of the .res file. spotConfig defaults to "NA", and will only be passed to the Algorithm if spotConfig$spot.fileMode=FALSE. See also: spotGetOptions

Items used are:

- alg.currentDesign: data frame holding the design points that will be evaluated
- io.apdFileName: name of the apd file
- io.desFileName: name of the des file
spotalgstartesvar

io.resFileName: name of the res file, for logging results (if spotConfig$spot.fileMode==TRUE)
spot.fileMode: boolean, if selected with true the results will also be written to
the res file, otherwise it will only be saved in the spotConfig returned by this
function

Value
this function returns the spotConfig list with the results in spotConfig$alg.currentResult

See Also
SPOT spot demo optim spotFuncStartBranin spotalgstartesvar

---

Description
This Evolution Strategy implementation can be tuned by SPOT. The ES can use different fitness
functions. The results are written to the res file. This function is needed as an interface, to en-
sure the right information are passed from SPOT to the target algorithm (e.g. the ES) and vice
versa. In contrast to spotalgstartes it is an interface for Pareto optimization, to optimize both the
performance as well as the variance of the ES algorithm, to reach more robust results.

Usage

spotalgstartesvar(spotConfig)

Arguments

spotalgstartesvar

Contains the list of spot configurations, results of the algorithm can be passed
to this list instead of the .res file. spotConfig defaults to "NA", and will only be
passed to the Algorithm if spotConfig$spot.fileMode=FALSE. See also: spotgetoptions
Items used are:

alg.currentDesign: data frame holding the design points that will be evaluated
io.apdFileName: name of the apd file
io.desFileName: name of the des file
io.resFileName: name of the res file, for logging results (if spotConfig$spot.fileMode==TRUE)
spot.fileMode: boolean, if selected with true the results will also be written to
the res file, otherwise it will only be saved in the spotConfig returned by this
function

Value
this function returns the spotConfig list with the results in spotConfig$alg.currentResult
**spotAlgStartRgp**

**Description**

SPOT uses this function for some demos to call the `symbolicRegression` function from the `rgp` package. This function is needed as an interface, to ensure the right information are passed from SPOT to the target algorithm (i.e. RGP) and vice versa.

**Usage**

```r
spotAlgStartRgp(spotConfig)
```

**Arguments**

- `spotConfig` contains the list of spot configurations, results of the algorithm can be passed to this list instead of the `.res` file. `spotConfig` defaults to "NA", and will only be passed to the Algorithm if `spotConfig$spot.fileMode=FALSE`. See also: `spotGetOptions`

Items used are:

- `alg.currentDesign`: data frame holding the design points that will be evaluated
- `io.apdFileName`: name of the apd file
- `io.desFileName`: name of the des file
- `io.resFileName`: name of the res file, for logging results (if `spotConfig$spot.fileMode==TRUE`)
- `spot.fileMode`: boolean, if selected with true the results will also be written to the res file, otherwise it will only be saved in the `spotConfig` returned by this function

**Value**

This function returns the `spotConfig` list with the results in `spotConfig$alg.currentResult`

**See Also**

- SPOT `spotAlgEs` `spotAlgStartEs`

---

**spotAlgStartRgp**  
**Interface for RGP to be tuned by SPOT**

---

**Description**

SPOT uses this function for some demos to call the `symbolicRegression` function from the `rgp` package. This function is needed as an interface, to ensure the right information are passed from SPOT to the target algorithm (i.e. RGP) and vice versa.

**Usage**

```r
spotAlgStartRgp(spotConfig)
```

**Arguments**

- `spotConfig` contains the list of spot configurations, results of the algorithm can be passed to this list instead of the `.res` file. `spotConfig` defaults to "NA", and will only be passed to the Algorithm if `spotConfig$spot.fileMode=FALSE`. See also: `spotGetOptions`

Items used are:

- `alg.currentDesign`: data frame holding the design points that will be evaluated
- `io.apdFileName`: name of the apd file
- `io.desFileName`: name of the des file
- `io.resFileName`: name of the res file, for logging results (if `spotConfig$spot.fileMode==TRUE`)
- `spot.fileMode`: boolean, if selected with true the results will also be written to the res file, otherwise it will only be saved in the `spotConfig` returned by this function

**Value**

This function returns the `spotConfig` list with the results in `spotConfig$alg.currentResult`

**See Also**

- SPOT `spot demo`
spotAlgStartSann

Interface for SANN to be tuned by SPOT

Description

SPOT uses this function for some demos to call the `optim` function with the SANN method, which means Simulated Annealing. The SANN uses `spotFuncStartBranin` as a target function. This function is needed as an interface, to ensure the right information are passed from SPOT to the target algorithm (e.g. the SANN) and vice versa.

Usage

`spotAlgStartSann(spotConfig)`

Arguments

`spotConfig` Contains the list of spot configurations, results of the algorithm can be passed to this list instead of the .res file. `spotConfig` defaults to "NA", and will only be passed to the Algorithm if `spotConfig$spot.fileMode=FALSE`. See also: `spotGetOptions`

Items used are:

- `alg.currentDesign`: data frame holding the design points that will be evaluated
- `io.apdFileName`: name of the apd file
- `io.desFileName`: name of the des file
- `io.resFileName`: name of the res file, for logging results (if `spotConfig$spot.fileMode==TRUE`)
- `spot.fileMode`: boolean, if selected with true the results will also be written to the res file, otherwise it will only be saved in the `spotConfig` returned by this function

Value

this function returns the `spotConfig` list with the results in `spotConfig$alg.currentResult`

See Also

`SPOT` `spot demo optim spotFuncStartBranin spotAlgStartSannVar`
Interface for SANN to be tuned robustly by SPOT

Description

SPOT uses this function for some demos to call the `optim` function with the SANN method, which means Simulated Annealing. The SANN uses `spotFuncStartBranin` as a target function. This function is needed as an interface, to ensure the right information are passed from SPOT to the target algorithm (e.g., the SANN) and vice versa. In contrast to `spotalgStartSann`, it is an interface for Pareto optimization, to optimize both the performance as well as the variance of the SANN algorithm, to reach more robust results.

Usage

```
spotalgStartSannVar(spotConfig)
```

Arguments

- `spotConfig`: Contains the list of spot configurations, results of the algorithm can be passed to this list instead of the .res file. `spotConfig` defaults to "NA", and will only be passed to the Algorithm if `spotConfig$spot.fileMode=FALSE`. See also: `spotGetOptions`.

Items used are:

- `alg.currentDesign`: data frame holding the design points that will be evaluated
- `io.apdFileName`: name of the apd file
- `io.desFileName`: name of the des file
- `io.resFileName`: name of the res file, for logging results (if `spotConfig$spot.fileMode==TRUE`)
- `spot.fileMode`: boolean, if selected with true the results will also be written to the res file, otherwise it will only be saved in the `spotConfig` returned by this function

Value

- this function returns the `spotConfig` list with the results in `spotConfig$alg.currentResult`

See Also

- SPOT demo optim spotFuncStartBranin spotalgStartSann
spotAlgStartSmsEmoaGlg

Algorithm Interface to ES + GLG

Description

Interface for mixed model tuning of an Evolution Strategy ES optimizing Gaussian Landscapes. For each repeat of an ES parameter setting, a different seed for the ES is used. At the same time, this seed is used to create different Gaussian Landscapes with the Gaussian Landscape Generator GLG, see spotGlgCreate.

Usage

spotAlgStartSmsEmoaGlg(spotConfig)

Arguments

spotConfig Contains the list of spot configurations, results of the algorithm can be passed to this list instead of the .res file. spotConfig defaults to "NA", and will only be passed to the Algorithm if spotConfig$spot.fileMode=FALSE. See also: spotGetOptions

Items used are:

- alg.currentDesign: data frame holding the design points that will be evaluated
- io.apdFileName: name of the apd file
- io.desFileName: name of the des file
- io.resFileName: name of the res file, for logging results (if spotConfig$spot.fileMode==TRUE)
- spot.fileMode: boolean, if selected with true the results will also be written to the res file, otherwise it will only be saved in the spotConfig returned by this function

Value

this function returns the spotConfig list with the results in spotConfig$alg.currentResult

See Also

SPOT spot demo optim spotFuncStartBranin spotAlgStartSannVar
Description

SPOT provides some functions for the creation of a set of design points. Design of Experiment (DoE) and latin hypercube designs (Lhd) are the best known and may be used even to create designs. Two different steps in SPOT, the initial step and all the sequential steps may use different design creating functions. To provide and include the users own design creating function, the user must follow these instructions carefully:

1) The function the user wants to include must be an R-function
2) use the function `spotInstAndLoadPackages` to add the packages that are required for your function, just make it the first line in your function.
3) adapt the configuration file (.conf), up to four parameters are to be included/changed:
   - `init.design.func="myCreateDesign1"
   - `seq.design.func="myCreateDesign2"

Usage

```r
spotCreateDesign() 
```

See Also

Design creating functions that are shipped with SPOT are: `spotCreateDesignLhs, spotCreateDesignFrF2, spotCreateDesignBasicDoe`, please check these examples for the correct input parameters and the structure of the return value before you include your own design creating function.
The Options of the configuration file (.conf) are described in `spotGetOptions`.

Description

Create a full factorial design based on the number of dimensions and the number of design points. Uses the function `gen.factorial` from the AlgDesign package. Two levels are used for each variable.

Usage

```r
spotCreateDesignBasicDoe(spotConfig, noDesPoints = 100, repeats = 1) 
```
**spotCreateDesignFactors**

**Arguments**

- `spotConfig`: list of `spotConfiguration`, where only the table `alg.ROI` (.roi-file) is used.
- `nodespoints`: optional parameter gives the number of design points. If `nodespoints` is larger than the size of full factorial design, an error is reported.
- `repeats`: is obsolete for this type of design, it has no influence.

**Details**

The dimension is driven from the number of rows of the .roi - file (each row in the roi file defines a variable. In case of a missing number of design points a value is calculated from the dimension

**Value**

- Matrix M
  - M has dimension columns and `nodespoints` rows

**See Also**

`spotCreateDesignLhd, spotCreateDesignFrF2, spotCreateDesignLhs, spotCreateDesignLhsOpt`

---

**spotCreateDesignFactors**

**Description**

Design function for designs consisting of factors only. Factorial designs for all factors. Assume that factors are encoded in the ROI, with integer notation.

**Usage**

```r
spotCreateDesignFactors(spotConfig, nodesPoints = NaN,
  repeats = NaN)
```

**Arguments**

- `spotConfig`: list of spot settings
- `nodesPoints`: is obsolete for this type of design, it has no influence. The number of points is fixed by the number of factors and their levels.
- `repeats`: is obsolete for this type of design, it has no influence.

**Value**

- matrix M
  - M has dimension columns and `nodesPoints` rows
**Description**

Uses the function FrF2 from the FrF2-package to generate a (fractional) factorial design. The factorial design is augmented with the augment.ccd function from DoE.wrapper package. The dimension is determined from the number of rows of the .roi - file (each row in the roi file defines a variable).

**Usage**

```r
spotCreateDesignFrF2(spotConfig, noDesPoints = NaN, repeats = NaN)
```

**Arguments**

- `spotConfig` list of spot settings
- `noDesPoints` is obsolete for this type of design, it has no influence. The number of points is fixed.
- `repeats` is obsolete for this type of design, it has no influence.

**Value**

matrix $M$
- $M$ has dimension columns and `noDesPoints` rows

**See Also**

- `spotCreateDesignBasicDoe`, `spotCreateDesignLhd`, `spotCreateDesignLhs`, `spotCreateDesignLhsOpt`

---

**Description**

Create a design based on the number of dimensions and the number of design points. The dimension is driven from the number of rows of the .roi - file (each row in the roi file defines a variable. In case of a missing number of design points a value is calculated from the dimension.
**Usage**

```r
spotCreateDesignLhs(spotConfig, noDesPoints = NaN, repeats = 1)
```

**Arguments**

- `spotConfig`: list of `spotConfiguration`
- `noDesPoints`: number of design points, default is NaN
- `retries`: number of retries, which is the number of trials to find a design with the lowest distance, default is 1

**Value**

- matrix `design`
- design has dimension columns and `noDesPoints` rows with entries corresponding to the region of interest.

**Author(s)**

Christian Lasarczyk

**See Also**

- `spotCreateDesignBasicDoe`, `spotCreateDesignFrF2`, `spotCreateDesignLhs`, `spotCreateDesignLhsOpt`

---

**Description**

Uses the `improvedLHS` function from the `lhs` package to create a Latin Hypercube Design. `improvedLHS` optimizes the euclidean distance between points in the sample.

**Usage**

```r
spotCreateDesignLhs(spotConfig, noDesPoints = NaN, repeats = NaN)
```

**Arguments**

- `spotConfig`: list of `spotConfiguration`
- `noDesPoints`: number of design points, default is NaN
- `repeats`: number of repeats, default is NaN
Value

matrix M
- M has dimension columns and noDesPoints rows with entries corresponding to the region of interest.

See Also

spotCreateDesignBasicDoe, spotCreateDesignFrF2, spotCreateDesignLhd, spotCreateDesignLhsOpt

Description

Uses the optimumLHS function from the lhs package to create a Latin Hypercube Design. optimumLHS uses the S optimality criterion.

Usage

spotCreateDesignLhsOpt(spotConfig, noDesPoints = NaN, repeats = NaN)

Arguments

spotConfig list of spotConfiguration
noDesPoints number of design points, default is NaN
repeats number of repeats, default is NaN

Value

matrix M
- M has dimension columns and noDesPoints rows with entries corresponding to the region of interest.

See Also

spotCreateDesignBasicDoe, spotCreateDesignFrF2, spotCreateDesignLhd, spotCreateDesignLhsOpt
Description

Alternating Recommendation of Candidate Points

This "multi ensemble" (see details) is comparable to the "single ensemble" `spotEnsembleSingleRoundSearch`. In contrast to round search, models are not alternated from one sequential SPOT step to another. Instead, all models are used to alternatingly to suggest multiple points for each step. This of course means that SPOT needs to be configured to use several new candidates in each sequential step (i.e. `spotConfig$seq.design.new.size` should be larger than one, ideally a multiple of the number of models in the ensemble). To determine the order of models, the feedback (e.g. error) of the model is used.

Relevant configuration parameters of this ensemble are:

- The function to calculate feedback (e.g. the error) of the models, default is `spotconfig$seq.ensemble.feed.func<-spotFeedback$error$full`
- The number of designs cut off for leave-one-out validation should always be zero for this ensemble, i.e. `spotConfig$seq.ensemble.cut.num <- 0`

Usage

```r
spotEnsembleMultiAlternate(rawB, mergedB, design, 
spotConfig, fit = NULL)
```

Arguments

- `rawB`: unmerged data
- `mergedB`: merged data
- `design`: new design points which should be predicted
- `spotConfig`: global list of all options, needed to provide data for calling functions
- `fit`: should always be NULL. Evaluation of existing fits is not implemented for this ensemble.

Details

This is a "multi ensemble", meaning that in every sequential step all models in the ensemble are trained and evaluated. The target is to actively combine all models responses, to get the best estimate on which candidate points are optimal.

The models used are specified in the `spotConfig` list, for instance:

```r
spotConfig$seq.ensemble.predictors = c(spotPredictRandomForest, spotPredictEarth, spotPredictForest)
```

To specify the settings of each individual model, set:

```r
seq.ensemble.settings = list(list(setting=1),list(setting=2),list(setting=3),list(setting=4))
```

Any parameters set in each of the corresponding lists (here: 4 individual lists) will overwrite settings in the main `spotConfig` list, when the concerned model function is called.
spotEnsembleMultiAverage

Value

returns the list spotConfig

References


Description

Weighted Average of Model Prediction
This "multi ensemble" (see details) combines the response of the models in the ensemble using a weighted average. The predicted values of each model are weighted by the models feedback (e.g. error), and then combined by averaging.

Relevant configuration parameters of this ensemble are:
- The function to calculate feedback (e.g. the error) of the models, default is spotConfig$seq.ensemble.feed.func<-spotFe
- The function average model predictions, default is spotConfig$seq.ensemble.average.func<-mean
- The number of designs created for validation, which can be 0 or larger integers, e.g spotConfig$seq.ensemble.cut.num <-
- The the cut-off type used for validation, which can be 0 or larger integers, e.g spotConfig$seq.ensemble.cut.num <- "perc"
- The size of the point-set cut off for validation, which can be 0 or larger integers, e.g spotConfig$seq.ensemble.cut.num <-

Usage

spotEnsembleMultiAverage(rawB, mergedB, design, spotConfig, fit = NULL)

Arguments

rawB unmerged data
mergedB merged data
design new design points which should be predicted
spotConfig global list of all options, needed to provide data for calling functions
fit if an existing model ensemble fit is supplied, the models will not be build based on data, but only evaluated with the existing fits (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.
**Details**

This is a "multi ensemble", meaning that in every sequential step all models in the ensemble are trained and evaluated. The target is to actively combine all models responses, to get the best estimate on which candidate points are optimal.

The models used are specified in the `spotconfig` list, for instance:

```r
spotConfig$seq.ensemble.predictors = c(spotPredictRandomForest, spotPredictEarth, spotPredictForest)
```

To specify the settings of each individual model, set:

```r
seq.ensemble.settings = list(list(setting=1), list(setting=2), list(setting=3), list(setting=4))
```

Any parameters set in each of the corresponding lists (here: 4 individual lists) will overwrite settings in the main `spotConfig` list, when the concerned model function is called.

**Value**

returns the list `spotConfig`

**References**


---

**spotEnsembleMultiChoose**

**Description**

Choose a model based on feedback

This "multi ensemble" (see details) evaluates each models feedback (e.g. error), and the best one is chosen to be used for the determination of new optimal candidate points. In order to allow for randomization, the epsilon parameter can be set to values between 0 and 1, where 1 means completely random (feedback is ignored) and 0 means completely deterministic (choice only dependent on feedback, not random).

Relevant configuration parameters of this ensemble are:

- The function to calculate feedback (e.g. the error) of the models, default is `spotConfig$seq.ensemble.feed.func<-spotFeedbackError`
- The function average model predictions, default is `spotConfig$seq.ensemble.average.func<-mean`
- The number of designs created for validation, which should be zero, i.e. `spotConfig$seq.ensemble.cut.num <- 0`
- The epsilon parameter, default is `spotConfig$seq.ensemble.epsilon <- 0`

**Usage**

```r
spotEnsembleMultiChoose(rawB, mergedB, design, spotConfig, fit = NULL)
```
spotEnsembleMultiRank

Arguments

- rawB: unmerged data
- mergedB: merged data
- design: new design points which should be predicted
- spotConfig: global list of all options, needed to provide data for calling functions
- fit: if an existing model ensemble fit is supplied, the models will not be build based on data, but only evaluated with the existing fits (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.

Details

This is a "multi ensemble", meaning that in every sequential step all models in the ensemble are trained and evaluated. The target is to actively combine all models responses, to get the best estimate on which candidate points are optimal.

The models used are specified in the spotConfig list, for instance:

```r
spotConfig$seq.ensemble.predictors = c(spotPredictRandomForest, spotPredictEarth, spotPredictForest)
```

To specify the settings of each individual model, set:

```r
seq.ensemble.settings = list(list(setting=1),list(setting=2),list(setting=3),list(setting=4))
```

Any parameters set in each of the corresponding lists (here: 4 individual lists) will overwrite settings in the main spotConfig list, when the concerned model function is called.

Value

returns the list spotConfig

References


Description

Alternating Recommendation of Candidate Points

This "multi ensemble" approach tries to combine the predictions of all models in the ensemble. In order to do so, all meta models are evaluated and their predictions are ranked. The overall response is then built as the sum of the exponentiated ranks.

Relevant configuration parameters of this ensemble are:

The function to calculate feedback (e.g. the error) of the models, default is spotConfig$seq.ensemble.feed.func <- spotFeedbackError

The number of designs cut off for leave-one-out validation should always be zero for this ensemble, i.e. spotConfig$seq.ensemble.cut.num <- 0
Usage

`spotEnsembleMultiRankWeighted(rawB, mergedB, design, spotConfig, fit = NULL)`

Arguments

- `rawB`: unmerged data
- `mergedB`: merged data
- `design`: new design points which should be predicted
- `spotConfig`: global list of all options, needed to provide data for calling functions
- `fit`: should always be `NULL`. Evaluation of existing fits is not implemented for this ensemble.

Details

This is a "multi ensemble", meaning that in every sequential step all models in the ensemble are trained and evaluated. The target is to actively combine all models responses, to get the best estimate on which candidate points are optimal.

The models used are specified in the `spotConfig` list, for instance:

```r
spotConfig$seq.ensemble.predictors = c(spotPredictRandomForest, spotPredictEarth, spotPredictForest)
```

To specify the settings of each individual model, set:

```r
seq.ensemble.settings = list(list(setting=1), list(setting=2), list(setting=3), list(setting=4))
```

Any parameters set in each of the corresponding lists (here: 4 individual lists) will overwrite settings in the main `spotConfig` list, when the concerned model function is called.

Value

returns the list `spotConfig`

References

Description

Alternating Recommendation of Candidate Points

This "multi ensemble" uses the same ranking approach as spotEnsembleMultiRank. Additionally, the ranks are weighted. To determine an appropriate weight, the models feedback (e.g. error measure) is employed.

Relevant configuration parameters of this ensemble are:

The function to calculate feedback (e.g. the error) of the models, default is spotConfig$seq.ensemble.feed.func<-spotFeedFunc

The number of designs cut off for leave-one-out validation should always be zero for this ensemble, i.e spotConfig$seq.ensemble.cut.num <- 0

Usage

spotEnsembleMultiRankWeighted(rawB, mergedB, design, 
spotConfig, fit = NULL)

Arguments

rawB unmerged data
mergedB merged data
design new design points which should be predicted
spotConfig global list of all options, needed to provide data for calling functions
fit should always be NULL. Evaluation of existing fits is not implemented for this ensemble.

Details

This is a "multi ensemble", meaning that in every sequential step all models in the ensemble are trained and evaluated. The target is to actively combine all models responses, to get the best estimate on which candidate points are optimal.

The models used are specified in the spotConfig list, for instance:

spotConfig$seq.ensemble.predictors = c(spotPredRandomForest, spotPredEarth, spotPredForrest)

To specify the settings of each individual model, set:

seq.ensemble.settings = list(list(setting=1),list(setting=2),list(setting=3),list(setting=4))

Any parameters set in each of the corresponding lists (here: 4 individual lists) will overwrite settings in the main spotConfig list, when the concerned model function is called.

Value

returns the list spotConfig

References

Description

BLA Bernoulli - Bayesian Learning Automaton for Bernoulli Distributed Feedback [Braadland_Norheim]

An advantage of this is, that the only relevant parameters are the initial values of the beta distribution. A disadvantage is the stationarity of the algorithm. The ensemble problem in SPOT is of dynamic nature.

The default reward function is `

Usage

`spotEnsembleSingleBLAbern(rawB, mergedB, design, spotConfig, fit = NULL)

Arguments

- `rawB` unmerged data
- `mergedB` merged data
- `design` new design points which should be predicted
- `spotConfig` global list of all options, needed to provide data for calling functions
- `fit` if an existing model ensemble fit is supplied, the models will not be build based on data, but only evaluated with the existing fits (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.

Details

This is a "single ensemble", meaning that in every sequential step only one model in the ensemble is trained and evaluated. The target is to actively "learn" which of the models are most suitable, based on their individual success.

The models used are specified in the `spotConfig` list, for instance:

```
spotConfig$seq.ensemble.predictors = c(spotPredictRandomForest, spotPredictEarth, spotPredictForest)
```

To specify the settings of each individual model, set:

```
seq.ensemble.settings = list(list(setting=1),list(setting=2),list(setting=3),list(setting=4))
```

Any parameters set in each of the corresponding lists (here: 4 individual lists) will overwrite settings in the main `spotConfig` list, when the concerned model function is called.

Value

returns the list `spotConfig`
References

Single Ensemble: BLAnorm

Description
BLA normal - Bayesian Learning Automaton for Normally Distributed Feedback[Braadland_Norheim]
An advantage of this approach is, that the only relevant parameters are the initial values which here remain at defaults. A disadvantage is the stationarity of the algorithm. The ensemble problem in SPOT is of dynamic nature.
The default reward function is `spotConfig$seq.ensemble.feed.func <- spotFeedback.reward.norm`.

Usage
`spotEnsembleSingleBLAnorm(rawB, mergedB, design, spotConfig, fit = NULL)`

Arguments
- `rawB`: unmerged data
- `mergedB`: merged data
- `design`: new design points which should be predicted
- `spotConfig`: global list of all options, needed to provide data for calling functions
- `fit`: if an existing model ensemble fit is supplied, the models will not be build based on data, but only evaluated with the existing fits (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.

Details
This is a "single ensemble", meaning that in every sequential step only one model in the ensemble is trained and evaluated. The target is to actively "learn" which of the models are most suitable, based on their individual success.
The models used are specified in the `spotConfig` list, for instance:
`spotConfig$seq.ensemble.predictors = c(spotPredictRandomForest, spotPredictEarth, spotPredictForest)`
To specify the settings of each individual model, set:
`spotConfig$seq.ensemble.settings = list(list(setting=1),list(setting=2),list(setting=3),list(setting=4))`
Any parameters set in each of the corresponding lists (here: 4 individual lists) will overwrite settings in the main `spotConfig` list, when the concerned model function is called.
Value

returns the list spotConfig

References


spotEnsembleSingleEpsGreedy

Single Ensemble: Epsilon Greedy

Description

Eps. Greedy - Select the model with the greatest success with probability 1-epsilon (greedy decision), select a random model with probability epsilon (explorative decision). That means, the algorithm is completely greedy with epsilon=0, and completely explorative with epsilon=1. The default reward function is spotConfig$seq.ensemble.feed.func <- spotFeedback.reward.bern. The value of epsilon spotConfig$seq.greed.epsilon <- 0.5.

Usage

spotEnsembleSingleEpsGreedy(rawB, mergedB, design, spotConfig, fit = NULL)

Arguments

rawB unmerged data
mergedB merged data
design new design points which should be predicted
spotConfig global list of all options, needed to provide data for calling functions
fit if an existing model ensemble fit is supplied, the models will not be build based on data, but only evaluated with the existing fits (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.
Details

This is a "single ensemble", meaning that in every sequential step only one model in the ensemble is trained and evaluated. The target is to actively "learn" which of the models are most suitable, based on their individual success.

The models used are specified in the spotConfig list, for instance:

```r
spotConfig$seq.ensemble.predictors = c(spotPredictRandomForest, spotPredictEarth, spotPredictForest)
```

To specify the settings of each individual model, set:

```r
seq.ensemble.settings = list(list(setting=1),list(setting=2),list(setting=3),list(setting=4))
```

Any parameters set in each of the corresponding lists (here: 4 individual lists) will overwrite settings in the main spotConfig list, when the concerned model function is called.

Value

returns the list spotConfig

References

Arguments

rawB       unmerged data
mergedB    merged data
design     new design points which should be predicted
spotConfig global list of all options, needed to provide data for calling functions
fit        if an existing model fit is supplied, the model will not be build based on data, but only evaluated with the model fit (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.

Details

This is a "single ensemble", meaning that in every sequential step only one model in the ensemble is trained and evaluated. The target is to actively "learn" which of the models are most suitable, based on their individual success.

The models used are specified in the spotConfig list, for instance:

```r
spotConfig$seq.ensemble.predictors = c(spotPredictRandomForest, spotPredictEarth, spotPredictForest)
```

To specify the settings of each individual models, set:

```r
seq.ensemble.settings = list(list(setting=1),list(setting=2),list(setting=3),list(setting=4))
```

Any parameters set in each of the corresponding lists (here: 4 individual lists) will overwrite settings in the main spotConfig list, when the concerned model function is called.

Value

returns the list spotConfig

Note

Implementation might still be faulty, not suggested for serious experiments. This is work in progress.

References

**Description**

Round Search - This is a very simple approach to employing a set of models (ensemble). Each of the chosen models is used in turn. That means:

At time \( t \), select model number \(((t-1) \mod k)+1\).

In contrast to other single ensembles (i.e. `spotEnsembleSingleSoftMax`) this approach does not use any kind of reward function to rate the success of the models. It has also no additional parameters to set, but will strongly depend on the ordering of the specified models.

**Usage**

```r
spotEnsembleSingleRoundSearch(rawB, mergedB, design, spotConfig, fit = NULL)
```

**Arguments**

- `rawB`: matrix of raw x and y values
- `mergedB`: matrix of merged x and y values, does not have replicate entries
- `design`: design points to be evaluated by the meta model
- `spotConfig`: the list of all parameters. Information about probability and success are stored in the sublist `spotConfig$seq$predictDual`
- `fit`: if an existing model ensemble fit is supplied, the models will not be build based on data, but only evaluated with the existing fits (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters `mergedB` and `rawB` will not be used at all in the function.

**Details**

This is a "single ensemble", meaning that in every sequential step only one model in the ensemble is trained and evaluated. The target is to actively "learn" which of the models are most suitable, based on their individual success.

The models used are specified in the `spotConfig` list, for instance:

```r
spotConfig$seq.ensemble.predictors = c(spotPredictRandomForest, spotPredictEarth, spotPredictForest)
```

To specify the settings of each individual model, set:

```r
seq.ensemble.settings = list(list(setting=1),list(setting=2),list(setting=3),list(setting=4))
```

Any parameters set in each of the corresponding lists (here: 4 individual lists) will overwrite settings in the main `spotConfig` list, when the concerned model function is called.

**Value**

returns the list `spotConfig`
References


spotEnsembleSingleSoftMax

Single Ensemble: SoftMax

Description

SoftMax Algorithm - Select a model based on a probability proportional to its success (i.e. reward)
The default reward function is spotConfig$seq.ensemble.feed.func<~ spotFeedback.reward.bern.
Two parameters have to be set. The parameter alpha is used as a multiplier when a models reward is updated:
The default value is spotConfig$seq.softmax.alpha <- 0.5.
The parameter tau is used to control exploration/exploitation trade-off. SoftMax is completely greedy with tau=0, and completely random with tau going to infinity:
The default value is spotConfig$seq.softmax.tau <- 1.

Usage

spotEnsembleSingleSoftMax(rawB, mergedB, design, 
spotConfig, fit = NULL)

Arguments

rawB unmerged data
mergedB merged data
design new design points which should be predicted
spotConfig global list of all options, needed to provide data for calling functions
fit if an existing model ensemble fit is supplied, the models will not be build based on data, but only evaluated with the existing fits (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.

Details

This is a "single ensemble", meaning that in every sequential step only one model in the ensemble is trained and evaluated. The target is to actively "learn" which of the models are most suitable, based on their individual success.
The models used are specified in the spotConfig list, for instance:

```
spotConfig$seq.ensemble.predictors = c(spotPredictRandomForest, spotPredictEarth, spotPredictForest)
```

To specify the settings of each individual model, set:
seq.ensemble.settings = list(list(setting=1), list(setting=2), list(setting=3), list(setting=4))

Any parameters set in each of the corresponding lists (here: 4 individual lists) will overwrite settings in the main spotConfig list, when the concerned model function is called.

**Value**

returns the list spotConfig

**References**


---

**Description**

UCB1 - Upper Confidence Bound 1

Use the model that maximizes $x_i + \sqrt{\frac{2 \ln(n)}{n_i}}$, where $x_i$ is the average reward of the model $i$, $n$ is the number of sequential steps (i.e. number of times any model was used), and $n_i$ is the number of times the model $i$ was chosen. The default reward function is spotConfig$seq.ensemble.feed.func$<spotFeedback.reward.bern.

**Usage**

```
spotEnsembleSingleUCB1(rawB, mergedB, design, spotConfig, fit = NULL)
```

**Arguments**

- **rawB** unmerged data
- **mergedB** merged data
- **design** new design points which should be predicted
- **spotConfig** global list of all options, needed to provide data for calling functions
fit if an existing model ensemble fit is supplied, the models will not be build based on data, but only evaluated with the existing fits (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.

Details
This is a "single ensemble", meaning that in every sequential step only one model in the ensemble is trained and evaluated. The target is to actively "learn" which of the models are most suitable, based on their individual success.
The models used are specified in the spotConfig list, for instance:
spotConfig$seq.ensemble.predictors = c(spotPredictRandomForest, spotPredictEarth, spotPredictForest)

To specify the settings of each individual model, set:
seq.ensemble.settings = list(list(setting=1),list(setting=2),list(setting=3),list(setting=4))

Any parameters set in each of the corresponding lists (here: 4 individual lists) will overwrite settings in the main spotConfig list, when the concerned model function is called.

Value
returns the list spotConfig

References

spotFeedback

---

**Ensemble Feedback Functions**

**Description**
This describes the feedback functions which can be employed in the spotEnsembleSingle* and spotEnsembleMulti* functions.

**Usage**

```r
spotFeedback.y(spotConfig,mergedB,rawB);
spotFeedback.sd.interSubModelsFull(spotConfig,mergedB,rawB);
spotFeedback.deviation(spotConfig,mergedB,rawB);
spotFeedback.error.full(spotConfig,mergedB,rawB);
spotFeedback.error.last(spotConfig,mergedB,rawB);
spotFeedback.error.order(spotConfig,mergedB,rawB);
```
spotFeedback.error.combo(spotConfig, mergedB, rawB);
spotFeedback.reward.bern(spotConfig, mergedB, rawB);
spotFeedback.reward.norm(spotConfig, mergedB, rawB);

Arguments

spotConfig parameter list, as created by the calling functions
mergedB merged list of design points as evaluated on the target function of SPOT
rawB raw list of design points as evaluated on the target function of SPOT

Details

If spotFeedback.error.full or spotFeedback.error.last are used, the user can choose an error function of type $f(a,b)$ where $a$ and $b$ are vectors whose differences are the deviations between model and real function. By default this function is chosen with

```r
spotConfig$seq.ensemble.error.func <- spotSelectionRMSE
```

See `spotSelectionCriteria` for some other options. Note that ".full" indicates that the modelling error is calculated based on all known design points, whereas ".last" indicates that the modelling error is only calculated based on the last evaluated design points, which can be considered unseen data for the model. Therefore, ".full" might be biased, whereas ".last" might be more random due to the small number of samples. spotFeedback.error.full and spotFeedback.error.last return scalar values for each model, i.e. a vector.

spotFeedback.y simply returns predictions from previous steps. It therefore returns a vector for each model, i.e. a matrix of num-model columns, and npoints rows.

spotFeedback.sd.interSubModelsFull when cut off sub-designs are available for validation, this function can be used to calculate the standard deviation at each point. It therefore returns a vector for each model, i.e. a matrix of num-model columns, and npoints rows.

spotFeedback.reward* reward based on success of the model used in last sequential step. Scalar value returned.

Value

spotFeedback.y, spotFeedback.sd.interSubModelsFull and spotFeedback.deviation return a matrix with as many columns as models in the ensemble and as many rows as points in mergedB.

spotFeedback.reward.bern and spotFeedback.reward.norm return single values, which indicate the Bernoulli or normal distributed reward for the model used in the last sequential step.

all other functions return a vector with an error measure for each model in the ensemble.

---

spotGetOptions Set all options by conf-file or by default
Description

spotGetOptions
1.) sets default values
2.) overwrites all default values by the settings the user provides with the config file (.conf-file)

All options described here, that are not marked as "internal variable" may be changed by the user. This will be done by reading the ".conf"-file that the user has specified as the first (and maybe sole) parameter to the function spot(). To change this default value of a variable, simply write a line into the ".conf"-file following this syntax:
<variable>=<value> e.g.: spot.seed=54321

This function will do even more: the user may define his own variables in the .conf-file and may use them in user written plugins. All plugins will get the whole list of options with the parameter "spotConfig". As a result a variable given in the .conf file as my.var=37
may be referred to by spotConfig$my.var and can be used in all functions - especially in the functions that are designed to be open to adaptions where ever necessary.

Usage

spotGetOptions(srcPath = ".", configFileName)

Arguments

srcPath the absolute path to the SPOT sources
configFileName users config file (.conf) the absolute path including file-specifier of the user config File

Value

spotGetOptions returns the list of all SPOT options created by this function:

auto.loop.steps [Inf] number of iterations the loop over all SPOT-steps should be repeated
auto.loop.nevals [200] budget of algorithm/simulator runs - most important parameter for runtime of the algorithm in case the spot-function is called with the "auto"-task
spot.continue [FALSE] boolean, SPOT will try to continue based on existing results in spotConfig or .res file if this value is TRUE
spot.fileMode [TRUE] boolean, that defines if files are used to read and write results (which is the "classic" spot procedure) or if SPOT will only use the workspace to store variables.
spot.seed [123] global seed setting for all random generator dependent calls within SPOT. same seed shall repeat same results, BUT: please note: this is NOT the seed for the algorithm! see alg.seed
alg.func ["spotFuncStartBranin"] target function to be optimized by SPOT.
SPOT searches for the given function name in workspace
If alg.func is a string, SPOT expects an interface like spotFuncStartBranin.
If alg.func is a function, SPOT expects a function of type y=f(x) (see: spotOptimInterface).
alg.resultColumn
"\["\] string to indicate the name of the result column. This might be automatically reset if `spotOptimInterface` is used, i.e. if a function is passed to `alg.func`.

It can be a vector of strings, if multi objective optimization is done.

alg.seed
[1234] seed for random generator to be used by the user defined algorithm. This is needed to reproduce the results. Set to NA if seed should not be reset.

alg.roi
internal parameter for the initial region of interest (do not try to set this one, it will be overwritten with default values). It is used to provide an easy to use matrix with the data from the ".roi"-file (= Region Of Interest)

The user can create an ROI matrix with the `spotROI` constructor.

alg.aroi
internal parameter for the actual region of interest (do not try to set this one, it will be overwritten with default values). It is used to provide an easy to use matrix with the data from the ".aroi"-file (= Actual Region Of Interest)

alg.currentDesign
[usually not changed by user] data frame of the design that will be evaluated by the next call to `spotStepRunAlg`.

alg.currentResult
[usually not changed by user] data frame that contains the results of the target algorithm runs.

alg.currentBest
[usually not changed by user] data frame that contains the best results of each step conducted by spot.

io.columnSep
["\["\] column separator for the input/output files, default means: arbitrary whitespace sequence, should be set by the value you want to have between your columns

io.apdFileName
[depends: <configFileName>.apd] name of the .apd -file (Algorithm Problem Definition file, holding all specification the user written algorithm needs to perform a complete optimization)

io.roiFileName
[depends: <configFileName>.roi] name of the .roi -file (Region Of Interest File, holding all varying parameters and constraints)

io.desFileName
[depends: <configFileName>.des] name of the .des -file (DESign file, the file the user written algorithm uses as input to the parameters it should change)

io.resFileName
[depends: <configFileName>.res] name of the .res -file (RESULT file) the user written algorithm has to write its results into this file

io.bstFileName
[depends: <configFileName>.bst] name of the .bst -file (Best file) the result file will be condensed to this file

io.pdfFileName
[depends: <configFileName>.pdf] name of the .pdf -file the default report will write its summary of results in this pdf file

io.fbsFileName
[depends: <configFileName>.bst] name of the .fbs -file (Final BestSolution file) collects all final best values of all .bst files during a .meta-run

io.verbosity
[3] level of verbosity of the program, 0 should be silent and 3 should produce all output- sometimes just interesting for the developer...
init.design.func
["spotCreateDesignLhd"] name of the function to create an initial design. Please also see the notes SPOT - extensions

init.design.size
[10] number of initial design points to be created. Required by some space filling design generators. Will be used in the <init.design.func>.R-file. If =NA a value is calculated by formula.

init.design.retries
[100] number of retries the initial designs should be retried to find randomly a design with maximum distance between the points. This parameter will be ignored if the function is deterministic (like doe)

init.design.repeats
[2] number of repeats for each design point to be called with the <alg.func>

init.delete.previous.files
[TRUE] delete an existing res or bst file. Should be set to FALSE if a SPOT run is resumed, after crash or user triggered stop.

seq.design.size
[10000] number of sequential design points to be created

seq.design.retries
[10] number of retries the initial designs should be retried to find randomly a design with maximum distance between the points. This parameter will be ignored if the function is deterministic (like doe)

seq.design.oldBest.size
[1] number of the best already evaluated design points that should be taken into consideration for the next sequential designs (e.g. for to have an appropriate number of repeats

seq.design.new.size
[2] according to the predictor the new design points during the seq step are ordered by their expected values. This parameter states how many new design points should be evaluated

seq.design.maxRepeats
[NA] each design point is to be evaluated several times for statistically sound results. The number of "repeats" will increase, but will not exceed this seq.design.maxRepeats - value

seq.design.increase.func
["spotSeqDesignIncreasePlusOne"] functional description of how the repeats are increased (until the seq.design.maxRepeats are reached). Default increases the number of repeats by adding one.

seq.design.func
["spotCreateDesignLhd"] name of the function to create sequential design. Please also see the notes SPOT - extensions

seq.mco.selection
["hypervol"] selection scheme for new design candidates in case of multi objective optimization. "hypervol" considers contribution of each point, "tournament2" is a tournament selection. "tournament1" is not yet recommended for use.
seq.predictionModel.func
   ["spotPredictRandomForest"] name of the function calling a predictor. Default uses a Random Forest.
seq.predictionOpt.func
   [NA] If not NA this string will be interpreted as a function name. The function is expected to add a new setting to the sequential design. See spotPredictOptMulti
seq.merge.func [mean] defines the function that merges the results from the different repeat-runs for a design. Default is to calculate the mean value.
seq.transformation.func
   [I] function for transformation of "Y" before new model is created, default: Identity function
seq.useAdaptiveRoi
   [FALSE] use region of interest adaptation
report.func
   ["spotReportDefault"] name of the function providing the report ("spotReportSens","spotReport3d","spotReportContour")
report.io.screen
   [FALSE] report graphics will be printed to screen (FALSE=no, TRUE=yes)
report.io.pdf
   [TRUE] report graphics will be printed to pdf (FALSE=no, TRUE=yes)

Note
Additional settings can and will be written to the spotConfig list by other optional functions. See the documentation of these to functions for details.

See Also
SPOT spotPrepare

spotGlgCreate Create Gaussian Landscape

Description
This function is based on the Gaussian Landscape Generator by Bo Yuan and Marcus Gallagher. It creates a new Gaussian Landscape every time it is called. This Landscape can be evaluated like a function.

Usage
spotGlgCreate(dimension = 2, nGaussian = 10,
   lower = rep(0, dimension), upper = rep(1, dimension),
   globalvalue = 1, ratio = 0.8, seed = 1)
Arguments

dimension       dimensionality of the landscapes input space. Default is 2.
nGaussian       number of Gaussian components in the landscape. Default is 10.
lower           lower boundary of the landscape, defaults to \( \text{rep}(0, \text{dimension}) \).
upper           upper boundary of the landscape, defaults to \( \text{rep}(1, \text{dimension}) \).
globalvalue     the global maximum value, i.e. the maximum of the Gaussian component with
                the largest value. Default is 1.
ratio           maximum ratio of the local maxima, local optima are randomly generated within
                \([0, \text{globalvalue} \times \text{ratio}] \). Has to be larger than 0 and smaller than 1. Defaults
to 0.8.
seed            seed for the random number generator used before creation of the landscape.
                Generator status will be saved and reset afterwards.

Value

returns a function. The function takes a point (vector) as input, with as many values as specified in
dimension. The function returns a single scalar value, which is the Landscape value at the current
point. The function has several attributes which reflect the items returned by \texttt{spotGlgInit}. reflect
the values returned by

References

B. Yuan and M. Gallagher (2003) "On Building a Principled Framework for Evaluating and Test-
ing Evolutionary Algorithms: A Continuous Landscape Generator". In Proceedings of the 2003
Congress on Evolutionary Computation, IEEE, pp. 451-458, Canberra, Australia.

See Also

\texttt{spotGlgCreateRot} \texttt{spotGlgCreateN} \texttt{spotGlgCreateRotSearched}

Examples

```r
## Create a landscape function with default settings:
landscapeFun <- spotGlgCreate()
## Plot the landscape (uncomment before running this example)
#spotSurf3d(landscapeFun)
```

\texttt{spotGlgCreateN} \textit{Create Gaussian Landscape (multiple)}

Description

This function is based on the Gaussian Landscape Generator by Bo Yuan and Marcus Gallagher.
The difference to \texttt{spotGlgCreate} is, that this function creates multiple landscapes at once which
can be used as a multi objective test problem. They are simply created sequentially after the random
number generator seed is set. Any differences are due to their non-deterministic nature.
Usage

```
spotGlgCreateRot(odimension = 2, idimension = 2,
nGaussian = rep(10, odimension),
lower = rep(0, idimension), upper = rep(1, idimension),
globalvalue = rep(1, odimension),
    ratio = rep(0.8, odimension), seed = 1)
```

Arguments

- `odimension`: dimensionality of output, i.e. number of landscapes. Default is 2.
- `idimension`: dimensionality of the landscapes input space. Default is 2.
- `nGaussian`: number of Gaussian components in the landscape. Default is 10.
- `lower`: lower boundary of the landscape, defaults to `rep(0, dimension)`. 
- `upper`: upper boundary of the landscape, defaults to `rep(1, dimension)`. 
- `globalvalue`: the global maximum value, i.e. the maximum of the Gaussian component with the largest value. Default is 1.
- `ratio`: maximum ratio of the local maxima, local optima are randomly generated within `c(0, globalvalue*ratio]`. Has to be larger than 0 and smaller than 1. Defaults to 0.8.
- `seed`: seed for the random number generator used before creation of the landscape. Generator status will be saved and reset afterwards.

Value

returns a function. The function takes a point (vector) as input, with as many values as specified in `dimension`. The function returns a vector value, representing the output from each landscape.

References


See Also

- `spotGlgCreate`, `spotGlgCreateRot`, `spotGlgCreateRotSearched`

---

**spotGlgCreateRot**

*Create Gaussian Landscape (rotated)*

**Description**

This function is based on the Gaussian Landscape Generator by Bo Yuan and Marcus Gallagher. The difference to `spotGlgCreate` is, that this function creates one landscape and a second one is created by rotating the first. This can be used as a simple two-objective test function.
Usage

spotGlgCreateRotSearched(idimension = 2, alpha = pi/2, nGaussian = 10, lower = rep(0, idimension), upper = rep(1, idimension), globalvalue = 1, ratio = 0.8, seed = 1)

Arguments

- idimension: dimensionality of the landscapes input space. Default is 2.
- alpha: rotation angle. Default is pi/2.
- nGaussian: number of Gaussian components in the landscape. Default is 10.
- lower: lower boundary of the landscape, defaults to rep(0, idimension).
- upper: upper boundary of the landscape, defaults to rep(1, idimension).
- globalvalue: the global maximum value, i.e. the maximum of the Gaussian component with the largest value. Default is 1.
- ratio: maximum ratio of the local maxima, local optima are randomly generated within [0, globalvalue*ratio]. Has to be larger than 0 and smaller than 1. Defaults to 0.8.
- seed: seed for the random number generator used before creation of the landscape. Generator status will be saved and reset afterwards.

Value

returns a function. The function takes a point (vector) as input, with as many values as specified in dimension. The function returns a vector value, representing the output from the two landscapes landscape.

References


See Also

spotGlgCreate, spotGlgCreateN, spotGlgCreateRotSearched

Description

This function is very similar to spotGlgCreateRot. However, to allow for meaningful results when comparing hypervolumes, this function supports random-searching the created landscapes. The resulting mean hypervolume of the random-search can be used for comparisons.
Usage

spotGlgCreateRotSearchedly(dimension = 2, alpha = pi/2,
    nGaussian = 10, lower = rep(0, dimension),
    upper = rep(1, dimension), globalvalue = 1,
    ratio = 0.8, seed = 1, repeats = 100, evals = 1000)

Arguments

idimension  dimensionality of the landscapes input space. Default is 2.
alpha       rotation angle. Default is pi/2.
nGaussian   number of Gaussian components in the landscape. Default is 10.
lower       lower boundary of the landscape, defaults to \text{rep}(0, \text{dimension}).
upper       upper boundary of the landscape, defaults to \text{rep}(1, \text{dimension}).
globalvalue the global maximum value, i.e. the maximum of the Gaussian component with
             the largest value. Default is 1.
ratio       maximum ratio of the local maxima, local optima are randomly generated within
             \text{[0, globalvalue*ratio]}. Has to be larger than 0 and smaller than 1. Defaults to 0.8.
seed        seed for the random number generator used before creation of the landscape.
             Generator status will be saved and reset afterwards.
repeats     number of random-search runs performed on the landscape.
evals       evaluations of the landscapes in each run of the random-search.

Value

returns a function. The function takes a point (vector) as input, with as many values as specified in

dimension. The function returns a vector value, representing the output from the two landscapes

landscape. The mean hypervolume is the “meanhvol” attribute of said function.

See Also

    spotGlgCreate  spotGlgCreateN  spotGlgCreateRot

Description

This function is based on the Gaussian Landscape Generator by Bo Yuan and Marcus Gallagher. It
randomly evaluates one or several points in a Gaussian Landscape created by spotGlgInit.

Usage

spotGlgEval(x, glg)
 Arguments

x  matrix of sample sites, containing one point in each row.

glg  list of values defining the Gaussian Landscape, created by spotGlgInit.

Value

returns a list, with the following items:
value  value of the combined landscape components
value of each component

Author(s)

Original Matlab code by Bo Yuan, ported to R by Martin Zaefferer

References


See Also

spotGlgCreate, spotGlgInit, spotGlgEvalRot, spotGlgEvalN

Examples

## Create a landscape with default settings:
landscape <- spotGlgInit()

## Create a landscape with larger boundaries and more Gaussian components
value <- spotGlgEval(c(0.5,0.5),landscape)

---

spotGlgEvalN  Gaussian Landscape Evaluation (multiple)

Description

This function is based on the Gaussian Landscape Generator by Bo Yuan and Marcus Gallagher. It randomly evaluates one or several points in several Gaussian Landscapes at once, as created by spotGlgInitN.

Usage

spotGlgEvalN(x, glglist)

Arguments

x  matrix of sample sites, containing one point in each row.

glglist  list of lists (one for each landscape) of values defining the Gaussian Landscape, created by spotGlgInit.
Value

returns a list, with the following items:
value vector of values from each landscape. components value of each component (will be NA)

References


See Also

spotGlgCreateN, spotGlgInitN, spotGlgEval, spotGlgEvalRot

Description

This function is based on the Gaussian Landscape Generator by Bo Yuan and Marcus Gallagher. It randomly evaluates one or several points in a combination of two landscapes, where the second is a rotation of the first.

Usage

spotGlgEvalRot(x, glg)

Arguments

x matrix of sample sites, containing one point in each row.
glg list of values defining the Gaussian Landscape, created by spotGlgInit.

Value

returns a list, with the following items:
value vector of 2 values, first belongs to non-rotated landscape. components value of each component (will be NA)

References


See Also

spotGlgCreateRot, spotGlgInit, spotGlgEval, spotGlgEvalN
Description
This function is based on the Gaussian Landscape Generator by Bo Yuan and Marcus Gallagher. It randomly initializes a Gaussian Landscape with the specified parameters.

Usage
```
spotGlgInit(dimension = 2, nGaussian = 10,
          lower = rep(0, dimension), upper = rep(1, dimension),
          globalvalue = 1, ratio = 0.8, seed = 1)
```

Arguments
- **dimension**: dimensionality of the landscapes input space. Default is 2.
- **nGaussian**: number of Gaussian components in the landscape. Default is 10.
- **lower**: lower boundary of the landscape, defaults to `rep(0, dimension)`.
- **upper**: upper boundary of the landscape, defaults to `rep(1, dimension)`.
- **globalvalue**: the global maximum value, i.e. the maximum of the Gaussian component with the largest value. Default is 1.
- **ratio**: maximum ratio of the local maxima, local optima are randomly generated within `[0, globalvalue*ratio]`. Has to be larger than 0 and smaller than 1. Defaults to 0.8.
- **seed**: seed for the random number generator used before creation of the landscape. Generator status will be saved and reset afterwards.

Value
returns a list, with the following items:
- **mean**: Matrix containing the mean vectors of the Gaussian components in the landscape, i.e. the locations of the local maxima of the functions. First vector (i.e. first row) will be the global maximum.
- **covinv**: Inverse of covariance matrix of each Gaussian component, stored as 3-dimensional array.
- **opt**: optimal values, i.e. maxima of the Gaussian components. `ngauss` number of Gaussian components `d` is the dimension

Author(s)
Original Matlab code by Bo Yuan, ported to R by Martin Zaefferer

References
spotGlgInitN

See Also

spotGlgCreate, spotGlgEval, spotGlgInitN

Examples

## Create a landscape with default settings:
landscape <- spotGlgInit()

## Create a landscape with larger boundaries and more Gaussian components
landscape <- spotGlgInit(2, 100, -5, 5, 10, 0.8)

---

spotGlgInitN: Initialize Gaussian Landscape (multiple)

Description

This function is based on the Gaussian Landscape Generator by Bo Yuan and Marcus Gallagher. It randomly initializes multiple Gaussian Landscapes with the specified parameters. Does not have to be called by user, if spotGlgCreateN is used.

Usage

spotGlgInitN(oDimension = 2, iDimension = 2,
             nGaussian = rep(10, oDimension),
             lower = rep(0, iDimension), upper = rep(1, iDimension),
             globalvalue = rep(1, oDimension),
             ratio = rep(0.8, oDimension), seed = 1)

Arguments

- `oDimension`: dimensionality of output, i.e. number of landscapes. Default is 2.
- `iDimension`: dimensionality of the landscapes input space. Default is 2.
- `nGaussian`: number of Gaussian components in the landscape. Default is 10.
- `lower`: lower boundary of the landscape, defaults to `rep(0, dimension)`. 
- `upper`: upper boundary of the landscape, defaults to `rep(1, dimension)`. 
- `globalvalue`: the global maximum value, i.e. the maximum of the Gaussian component with the largest value. Default is 1.
- `ratio`: maximum ratio of the local maxima, local optima are randomly generated within `[0, globalvalue*ratio]`. Has to be larger than 0 and smaller than 1. Defaults to 0.8.
- `seed`: seed for the random number generator used before creation of the landscape. Generator status will be saved and reset afterwards.
Value

returns a list containing again lists for each landscape, containing with the following items:
mean Matrix containing the mean vectors of the Gaussian components in the landscape, i.e. the
locations of the local maxima of the functions. First vector (i.e. first row) will be the global maxi-
mum. covinv Inverse of covariance matrix of each Gaussian component, stored as 3-dimensional
array. opt optimal values, i.e. maxima of the Gaussian components ngauss number of Gaussian
components d is the dimension

References

B. Yuan and M. Gallagher (2003) "On Building a Principled Framework for Evaluating and Test-
ing Evolutionary Algorithms: A Continuous Landscape Generator". In Proceedings of the 2003
Congress on Evolutionary Computation, IEEE, pp. 451-458, Canberra, Australia.

See Also

spotGlgCreateN, spotGlgEvalN, spotGlgInit

spotGui

Start the SPOT GUI

Description

This function starts the graphical user interface for SPOT, which is based on java. The .jar File
started with this function is in the directory where SPOT is installed.

Usage

spotGui()

Details

Java runtime environment needs to be installed to use the GUI.

spotHelpInterfaces...

General Help on SPOT...

Description

SPOT is a system open and designed to plug in
- your own algorithm see spotAlgStart...
- your own predictors see spotPredict...
- your own design creating functions spotCreateDesign...
- your own increase functions to increase the repeats for each sequential step

Usage

spotHelpInterfaces...()
spotInfillExpImp  

Neg. Log. of Expected Improvement Infill Criterion

Description

This is the single objective infill criterion "Expected Improvement" as introduced by Forrester (2008).

Usage

spotInfillExpImp(mean, sd, min)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>predicted mean values</td>
</tr>
<tr>
<td>sd</td>
<td>predicted st. deviation</td>
</tr>
<tr>
<td>min</td>
<td>the currently known optimum value</td>
</tr>
</tbody>
</table>

Value

Returns the negative log. of the Expected Improvement.

References


spotInfillHyperVolume  

MCO Infill Criterion

Description

Same as spotInfillLcbHyperVolume, but without using lower confidence bound. Simply aggregates contributed hypervolumes based on predicted means, penalizing dominated points.

Usage

spotInfillHyperVolume(resy, resvar, y, ref = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resy</td>
<td>predicted objective values</td>
</tr>
<tr>
<td>resvar</td>
<td>not used</td>
</tr>
<tr>
<td>y</td>
<td>the current Pareto front</td>
</tr>
<tr>
<td>ref</td>
<td>reference point, if not given will be chosen as maximum of observed values plus one</td>
</tr>
</tbody>
</table>
Value
returns the contribution (or penalty) for each row in resy

Description
This multi objective infill criterion is similar to the SMS-EGO infill criterion by Ponweiser (2008). It aggregates the objective values for each point by calculating the hypervolume contribution. As a first step the lower confidence bound is calculated, decreasing the predicted objective values by their predicted variance. Unlike SMS-EGO, epsilon dominance is not employed here. Also, the penalties for dominated points are calculated differently: The hypervolume between the dominated points and the current true Pareto front is used.

Usage
spotInfillLcbHyperVolume(resy, resvar, y, ref = NULL)

Arguments
- resy: predicted objective values
- resvar: predicted variance
- y: the current Pareto front
- ref: reference point, if not given will be chosen as maximum of observed values plus one

Value
returns the contribution (or penalty) for each row in resy

Note
An optimizer like pso will work signif. better than cmaes with this infill criterion.

References
**spotInfillLcbMulti**  
*Lower Confidence Bound Infill Criterion*

**Description**
Calculate lower confidence bound for all objectives and all observations. Also used by `spotInfillLcbHyperVolume`.

**Usage**
```
spotInfillLcbMulti(resy, resvar, y, ref = NULL)
```

**Arguments**
- `resy`: predicted objective values
- `resvar`: predicted variance
- `y`: the current Pareto front
- `ref`: unused

**Value**
returns the LCB for each row and column in resy.

---

**spotInfillLcbSingle**  
*Single objective lower confidence bound*

**Description**
LCB=mean-sd

**Usage**
```
spotInfillLcbSingle(mean, sd, min)
```

**Arguments**
- `mean`: predicted mean values
- `sd`: predicted st. deviation
- `min`: the currently known optimum value

**Value**
Returns the difference of mean and sd
**spotInfillProbImp**  
*Probability of Improvement Infill Criterion*

**Description**

This is the single objective infill criterion "Probability of Improvement" as introduced by Forrester (2008).

**Usage**

`spotInfillProbImp(mean, sd, min)`

**Arguments**

- **mean**: predicted mean values
- **sd**: predicted st. deviation
- **min**: the currently known optimum value

**Value**

Returns the negative of the Probability of Improvement.

**References**


---

**spotInfillSD**  
*Neg. SD Infill Criterion*

**Description**

Infill criterion: negative standard deviation only. This type of infill criterion is useful when pure exploration is desired.

**Usage**

`spotInfillSD(mean, sd, min)`

**Arguments**

- **mean**: predicted mean values
- **sd**: predicted st. deviation
- **min**: the currently known optimum value
spotInfillSExI2d

Value

Returns the of sd.

Description

This two-objective infill criterion is the Expected Improvement of the S-metric. That is, it aggregates the predicted objective values by an exact calculation of the Expected Improvement EI in hypervolume. As this gets more complex and time-consuming for higher dimensional objective spaces, this is only implemented for the two-objective case. An approximation approach for higher dimensional problems exists, but is not yet implemented in SPOT.

Usage

spotInfillSExI2d(resy, resvar, y, ref = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resy</td>
<td>predicted objective values</td>
</tr>
<tr>
<td>resvar</td>
<td>predicted variance</td>
</tr>
<tr>
<td>y</td>
<td>the current Pareto front</td>
</tr>
<tr>
<td>ref</td>
<td>reference point, if not given will be chosen as maximum of observed values plus one</td>
</tr>
</tbody>
</table>

Value

returns the EI for each row in resy

References

M. Emmerich, A.H. Deutz, J.W. Klinkenberg: The computation of the expected improvement in dominated hypervolume of Pareto front approximations, LIACS TR-4-2008, Leiden University, The Netherlands

See Also

spotSExI2d
spotModel.func  

**Model Prediction Interface**

**Description**

spotModel.func creates a function which represents the chosen surrogate model (see method parameter). The created function will yield values predicted at given new locations.

**Usage**

```r
spotModel.func(X, y, method = "spotPredictTree", settings = NULL)
```

**Arguments**

- **X**  
  Matrix of sample locations. Rows for points, columns for variables.
- **y**  
  Observations at sample locations (vector)
- **method**  
  which model function should be used specified as a string, e.g., "spotPredictRandomForest", "spotPredictLm", "spotPredictForrester". Most other "spotPredict*" functions provided by SPOT should work.
- **settings**  
  additional settings, as used by the corresponding spotPredict* function. NULL means default model settings are used.

**Value**

a function of type \( y = f(X) \), where \( X \) is new data, \( f \) is the surrogate model, and \( y \) the predicted values.

**See Also**

Alternatively, `spotModel.train` and `spotModel.predict` can be used to do a similar job.

**Examples**

```r
## simple test function
sphereFunction <- function(X) rowSums(X^2)
## sample locations
Xtrain <- matrix(runif(20),,2)
## evaluate on test function
y <- sphereFunction(Xtrain)
## train model
model <- spotModel.func(Xtrain,y,"spotPredictLm")
## evaluate model at a new sample location
newy <- model(c(0.5,0.5))
## plot model (based on 10*10 point matrix)
spotSurf3d(model,s=10)
```
Description

This function is used to interface the spotPredict* (e.g.: spotPredictRandomForest) functions. It serves as a simple interface to predict new data with on a surrogate models.

Usage

spotModel.predict(X, fit)

Arguments

X  
Matrix of locations to be predicted at. Rows for points, columns for variables.

fit  
a list of settings, including the model fit. This can either be a fit created by spotModel.train or a spotConfig list as returned by spot.

Value

the list of settings, including the model fit. This list can be employed as the "fit" parameter in spotModel.predict.

See Also

spotModel.train, spotModel.func

Examples

## simple test function
sphereFunction <- function(X) rowSums(X^2)
## sample locations
Xtrain <- matrix(runif(20),2)
## evaluate on test function
y <- sphereFunction(Xtrain)
## train model
model <- spotModel.train(Xtrain,y,"spotPredictLm")
## evaluate model at a new sample location
newy <- spotModel.predict(c(0.5,0.5),model)
spotModel.train  Model Training Interface

Description

This function is used to interface the spotPredict* (e.g.: spotPredictRandomForest) functions. It serves as a simple interface to train surrogate models, as need by the user.

Usage

spotModel.train(X, y, method = "spotPredictTree",
               settings = NULL)

Arguments

X Matrix of sample locations. Rows for points, columns for variables.
y Observations at sample locations (vector)
method which model function should be used specified as a string, e.g., "spotPredictRandomForest", "spotPredictLm", "spotPredictForrester". Most other "spotPredict*" functions provided by SPOT should work.
settings additional settings, as used by the corresponding spotPredict* function. NULL means default model settings are used.

Value

the list of settings, including the model fit. This list can be employed as the "fit" parameter in spotModel.predict.

See Also

instead of building a fit to be evaluated by spotModel.predict the spotModel.func function directly produces functions, which can be evaluated with new data.

Examples

```r
## simple test function
sphereFunction <- function(X) rowSums(X^2)
## sample locations
Xtrain <- matrix(runif(20),,2)
## evaluate on test function
y <- sphereFunction(Xtrain)
## train model
model <- spotModel.train(Xtrain,y,"spotPredictLm")
## print model fit
print(model$seq.modelFit)
```
spotModelDescentLm

**Steepest Descent on RSM (linear model)**

**Description**

Optimizes an existing fit of a linear model created by the rsm function. Uses steepest descent method and adaption of ROI alternatingly.

**Usage**

```r
spotModelDescentLm(startPoint, spotConfig)
```

**Arguments**

- `startPoint`: not used here
- `spotConfig`: list of all options, needed to provide data for calling functions.

**Value**

returns the list `spotConfig` with two new entries:
- `spotConfig$optDesign` are the parameters of the new minimal design point
- `spotConfig$optDesignY` is the associated value of the objective function

**See Also**

`spotModelParetoOptim`, `spotModelOptim`

---

spotModelOptim

**Optimize predicted meta model**

**Description**

Optimizes an existing fit of a model to get an optimal new design point. Executed after building the prediction model in the sequential SPOT step.

**Usage**

```r
spotModelOptim(startPoint, spotConfig)
```
Arguments

**startPoint**
initial points for the optimization

**spotConfig**
list of all options, needed to provide data for calling functions.

This function uses the parameter `spotConfig$seq$model_fit`. This is supposed to be a fit, that can be evaluated by the associated `spotConfig$seq$prediction_model$func` function. The parameter `spotConfig$seq$prediction_opt$method` will be used to choose the optimization method to be used to find the minimum of the fitted model:

- "optim-L-BFGS-B" - BFGS quasi-Newton: stats Package
- "pso" - Particle Swarm Optimization: pso Package
- "cmaes" - Covariance Matrix Adaptation Evolution Strategy: cmaes Package
- "genoud" - Combines evolutionary search algorithms with derivative-based (Newton or quasi-Newton) methods: rgenoud Package
- "DEoptim" - Differential Evolution implementation: DEoptim Package
- "bobyqa" - Trust region method that forms quadratic models by interpolation: minqa Package
- "BBoptim" - Strategy using different Barzilai-Borwein step-lengths: BB Package
- "GenSA" - Generalized simulated annealing which for global minimization of a very complex non-linear objective function with a very large number of optima: GenSA Package
- "hjkb" - Bounded Hooke-Jeeves algorithm for derivative-free optimization: dfoptim Package

Additionally to the above methods, several methods from the package nloptr can be chosen. For instance:

- "NLOPT_LN_NELDERMEAD" - Nelder-Mead Simplex
- "NLOPT_LN_SBPLX" - Nelder-Mead Simplex on sequence of subspaces
- "NLOPT_GN_DIRECT" - Direct Search
- "NLOPT_GN_DIRECT_L" - Direct Search, locally biased

The complete list of suitable nlopt methods (non-gradient, bound constraints) is:

- "NLOPT_GN_DIRECT","NLOPT_GN_DIRECT_L","NLOPT_GN_DIRECT_L_RAND","NLOPT_GN_DIRECT_NOSCAL","NLOPT_GN_DIRECT_L_NOSCAL","NLOPT_GN_DIRECT_L_RAND_NOSCAL","NLOPT_GN_ORIG_DIRECT","NLOPT_GN_ORIG_DIRECT_L","NLOPT_LN_PRAXIS","NLOPT_GN_CRS2_LM","NLOPT_LN_COBYLA","NLOPT_LN_NEWUOA_BOUND","NLOPT_LN_NELDERMEAD","NLOPT_LN_SBPLX","NLOPT_LN_BOBYQA","NLOPT_GN_ISRES"

All of the above methods use bound constraints, which will be chosen with the limits specified in `spotConfig$alg.roi` (or the .roi file). For references and details on the specific methods, please check the documentation of the packages that provide them.

Note that some methods may require additional parameterization. For this purpose, it would be recommended to use `spotPredictOptMulti` as a template to write custom functions. `spotPredictOptMulti` itself is limited to calling mostly default settings of a large number of different optimizers available in R.
Value

returns the list spotConfig with two new entries:
spotConfig$optDesign are the parameters of the new minimal design point
spotConfig$optDesignY is the associated value of the objective function

See Also

spotModelParetoOptim solves the same task for multi-criteria optimization (i.e. more than just one surrogate model)

spotModelParetoOptim  Multi criteria optimization of predicted surrogate models

Description

Uses by default the number of design points expected as population size for multi criteria optimization of the models build in the current sequential step. Executed after building the prediction models.

Usage

spotModelParetoOptim(startPoint, spotConfig)

Arguments

  startPoint    initial information, not yet used
  spotConfig    list of all options, needed to provide data for calling functions.

This function uses the parameter spotConfig$seq.modelFit. This is supposed to be a list of fits (i.e. one fit for each objective), that can be evaluated by calling the original model interface with that fit list. The parameter spotConfig$seq.predictionOpt.method will be used to choose the optimization method to be used to find the minimum of the fitted model:
"nsga2"
"sms-emoa"

Value

returns the list spotConfig with two new entries:
spotConfig$optDesign are the parameters of the new Pareto optimal design points
spotConfig$optDesignY are the associated values of the objective functions (e.g. meta model values)
See Also

*spotModelOptim* solves the same task for single objective optimization (i.e. just one surrogate model)

See *spotSmsEmoa* for the used SMS-EMOA implementation

---

**spotOcba**  
*Optimal Computing Budget Allocation OCBA for SPOT*

**Description**

Spreads the budget in an optimal way for the different design points, considering a minimization problem

**Usage**

```r
spotOcba(samp.mean, samp.var, samp.count, budget.add, 
  iz = NA, verbose = 0)
```

**Arguments**

- `samp.mean`: vector of mean values, length `nd`
- `samp.var`: vector of variances, length `nd`
- `samp.count`: vector of repeats performed already, length `nd`
- `budget.add`: additional number of repeats, distributed among the `nd` design points
- `iz`: indifference zone
- `verbose`: verbosity 0|1|2|3 0 is no printing

See Also

*spotGenerateSequentialDesign* *spotRepeatsOcba*

---

**spotOptim**  
*spotOptim: optim-like spot interface*

**Description**

Besides *spot* this is one of the main interfaces for using the SPOT package. It is build like the *optim* interface.

**Usage**

```r
spotOptim(par = NULL, fn, gr = NULL, ..., lower = -Inf, 
  upper = Inf, method, control = list())
```
Arguments

par is a point in search interval (defines dimension)

fn is the target function (it can also be a string with the name of a spot interface function, like "spotFuncStartBranin")

g is a gradient function, not implemented yet

... additional parameters to be passed on to fn

lower is a vector that defines the lower boundary of search space

upper is a vector that defines the upper boundary of search space

method is a string that describes which method is to be used.

control is a list of additional settings. maxit is the number of function evaluations, all the other settings will simply be passed to SPOT (see spotGetOptions for details)

Details

It is of important to note that spot by default expects to optimize noisy functions. That means, the default settings of spot, which are also used in spotOptim, include repeats of the initial and sequentially created design points. Also, as a default OCBA is used to spread the design points for optimal usage of the function evaluation budget. OCBA will not work when there is no variance in the data. So if the user wants to optimize non-noisy functions, the following settings should be used:

c\$spot.ocba <- FALSE
c\$seq.design.maxRepeats <- 1
c\$init.design.repeats <- 1

A call to a noisy function could look like this:

objFunction <- function(x){y=(x[1]+2)^2+(x[2]-4)^2+runif(1)}
spotOptim(par=c(1,1), fn=objFunction, lower=c(-10,-10), upper=c(10,10), method="spotPredictRandomForest"

A call to a non-noisy function could look like this:

objFunction <- function(x){y=(x[1]+2)^2+(x[2]-4)^2}
spotOptim(par=c(1,1), fn=objFunction, lower=c(-10,-10), upper=c(10,10), method="spotPredictRandomForest"

Value

This function returns a list with:

par parameters of the found solution

value target function value of the found solution

See Also

spot spotOptimInterface spotOptimizationInterface spotOptimizationInterfaceMco
Description

This is an interface to the Evolution Strategy used as a target algorithm by some SPOT demos. It is build like the optim interface.

Usage

spotOptimEs(par, fn, gr = NULL, ..., lower, upper,
method = NULL, control = list())

Arguments

par is a point in search interval (defines dimension)
fn is the target function
gr gradient function, not used by this function
... additional parameters to be passed on to fn
lower is a vector that defines the lower boundary of search space
upper is a vector that defines the upper boundary of search space
method this parameter is not used in the current version.
control is a list of additional settings.

The control list can contain the following settings:

maxit number of iterations, stopping criterion, default is 100
mue number of parents, default is 10
nu number, default is 10
dimension dimension number of the target function, default is 2
mutation string of mutation type, default is 1
sigmaInit initial sigma value (standard deviation), default is 1.0
nSigma number of standard deviations, default is 1
tau0 number, default is 0.0
tau number, learning parameter for self adaption, default is 1.0
rho number of parents involved in the procreation of an offspring (mixing number), default is "bi"
 sel number of selected individuals, default is 1
stratReco value, Recombination operator for strategy variables, default is 1
objReco value, Recombination operator for object variables, default is 2
maxGen number of generations, stopping criterion, default is Inf
seed number, random seed, default is 1
noise number, value of noise added to fitness values, default is 0.0
lowerLimit number, lower limit for search space, default is -1.0
spotOptimInterface

- **upperLimit** number, upper limit for search space, default is 1.0
- **verbosity** defines output verbosity of the ES, default is 0
- **plotResult** boolean, specifies if results are plotted, default is FALSE
- **logPlotResult** boolean, defines if plot results should be logarithmic, default is FALSE
- **term** a string, defines which termination criterion should be used, default is "iter"
- **sigmaRestart** number, value of sigma on restart, default is 0.1
- **preScanMult** initial population size is multiplied by this number for a pre-scan, default is 1
- **globalOpt** termination criterion on reaching a desired optimum value, default is rep(0, dimension)

**Value**
This function returns a list with:
- **par** parameters of the found solution
- **value** target function value of the found solution

**See Also**
- optim
- spotOptim

---

**Description**
SPOT uses this function to call functions passed to spotOptim or spot like they would be passed to optim(). That means, it will be used whenever an actual function is passed instead of a string. When a string is passed the string itself will contain the interface to use. This function is needed as an interface, to ensure the right information are passed from SPOT to the target function. It can handle single and multi criteria target functions, e.g. functions that return numerics or vectors of numerics.

**Usage**

spotOptimInterface(spotConfig, ...)

**Arguments**
- **spotConfig** Contains the list of spot configurations, results of the algorithm can be passed to this list instead of the .res file. spotConfig defaults to "NA", and will only be passed to the Algorithm if spotConfig$spot.fileMode=FALSE. See also: spotGetOptions

Items used are:
alg.currentDesign: data frame holding the design points that will be evaluated
io.apdFileName: name of the apd file
io.desFileName: name of the des file
io.resFileName: name of the res file, for logging results (if spotConfig$spot.fileMode==TRUE)
spot.fileMode: boolean, if selected with true the results will also be written to the res file, otherwise it will only be saved in the spotConfig returned by this function
spotConfig$alg.tar.func target function of type y=f(x,...)
... additional parameters to be passed on to target function: spotConfig$alg.tar.func

Value

this function returns the spotConfig list with the results in spotConfig$alg.currentResult

See Also

SPOT spot demo optim spotOptim

Description

This function is an interface fashioned like the optim function. It is used in SPOT to access several different optimization methods.

Usage

spotOptimizationInterface(par, fn, gr = NULL, lower, upper, method, control, ...)

Arguments

par is a point (vector) in the decision space of fn
fn is the target function of type y = f(x, ...)
gr gradient function, not implemented yet
lower is a vector that defines the lower boundary of search space
upper is a vector that defines the upper boundary of search space
method is a string that describes which method is to be used, as implemented in this function. Else it can be a function, which is a custom optimization function created by the user. See details.
control is a list of additional settings. See details.
... additional parameters to be passed on to fn
Details

The control list contains:
- `fevals` stopping criterion, number of evaluations allowed for fn (defaults to 100)
- `reltol` stopping criterion, relative tolerance (defaults to 1e-6)
- `abstol` stopping criterion, absolute tolerance (defaults to 1e-6)
- `popsize` population size or number of particles (default depends on method)
- `restarts` whether or not to do restarts, default is FALSE
- `vectorized` whether or not fn can evaluate multiple points at once, defaults to FALSE (only relevant for cmaes and pso methods)

Please note that all settings will have to be passed to the actual optimization method. Not all of those make use of the items listed above.

Also note that the parameter `method` will be used to choose the optimization method from the following list:
- "lhs" - Latin Hypercube Sampling
- "optim-L-BFGS-B" - BFGS quasi-Newton: stats Package
- "pso" - Particle Swarm Optimization: pso Package
- "cmaes" - Covariance Matrix Adaptation Evolution Strategy: cmaes Package
- "genoud" - Combines evolutionary search algorithms with derivative-based (Newton or quasi-Newton) methods: rgenoud Package
- "DEoptim" - Differential Evolution implementation: DEoptim Package
- "bb" - Trust region method that forms quadratic models by interpolation: minqa Package
- "BBoptim" - Strategy using different Barzilai-Borwein step-lengths: BB Package
- "GenSA" - Generalized simulated annealing which for global minimization of a very complex non-linear objective function with a very large number of optima: GenSA Package
- "hjkb" - Bounded Hooke-Jeeves algorithm for derivative-free optimization: dfoptim Package

Additionally to the above methods, several methods from the package nloptr can be chosen. For instance:
- "NLOPT_LN_NELDERMEAD" - Nelder-Mead Simplex
- "NLOPT_LN_SBPLX" - Nelder-Mead Simplex on sequence of subspaces
- "NLOPT_GN_DIRECT" - Direct Search
- "NLOPT_GN_DIRECT_L" - Direct Search, locally biased

The complete list of suitable nlopt methods (non-gradient, bound constraints) is:
- "NLOPT_GN_DIRECT","NLOPT_GN_DIRECT_L","NLOPT_GN_DIRECT_L_RAND","NLOPT_GN_DIRECT_NOSCAL","NLOPT_GN_ORIG_DIRECT","NLOPT_GN_ORIG_DIRECT_L","NLOPT_LN_PRAXIS","NLOPT_GN_CRS2_LM","NLOPT_LN_NELDERMEAD","NLOPT_LN_SBPLX","NLOPT_LN_BOBYQA","NLOPT_GN_ISRES"

All of the above methods use bound constraints. For references and details on the specific methods, please check the documentation of the packages that provide them.

Furthermore, the user can choose to use a function instead of a string for the `method`. The used function should have the same parameters and arguments as documented for this very function, i.e. `spotOptimizationInterface`. 
Value

This function returns a list with:
par parameters of the found solution
value target function value of the found solution
counts number of evaluations of fn

See Also

spotOptim spotOptimizationInterfaceMco

Description

This function is an interface fashioned like the optim function. It is used in SPOT to access several different multi-criteria optimization methods.

Usage

spotOptimizationInterfaceMco(par, fn, gr = NULL, lower,
upper, method, control, ref, ...)

Arguments

par is a point (vector) in the decision space of fn, par is not used (yet)
fn is the target function of type $y = f(x, \ldots)$
gr gradient function, gr is not used (yet)
lower is a vector that defines the lower boundary of search space
upper is a vector that defines the upper boundary of search space
method is a string that describes which method is to be used, as implemented in this function. Else it can be a function, which is a custom optimization function created by the user. See details.
control is a list of additional settings. See details.
ref reference point. Please provide this even with methods that do not use it (e.g. "nsga2"), to specify the dimension of the objective space.
... additional parameters to be passed on to fn
**Details**

The control list contains:

- **fevals**: stopping criterion, number of evaluations allowed for fn (defaults to 100)
- **popsize**: population size or number of particles (default depends on method)
- **restarts**: whether or not to do restarts, default is FALSE

Also note that the parameter **method** will be used to choose the optimization method from the following list:

- "nsga2": the nsga2 function from the mco Package
- "sms-emoa": The basic sms-emoa in the SPOT package

All of the above methods use bound constraints. For references and details on the specific methods, please check the documentation of the packages that provide them.

Furthermore, the user can choose to use a function instead of a string for the **method**. The used function should have the same parameters and arguments as documented for this very function, i.e. **spotOptimizationInterfaceMco**.

**Value**

This function returns a list with:

- **par**: parameters of the found solutions, e.g. the Pareto set
- **value**: target function values of the found solutions, e.g. the Pareto front
- **counts**: number of evaluations of fn

**See Also**

- **spotOptimizationInterface**
- **spotOptim**

---

**Description**

This function is an interface to Latin Hypercube Sampling (LHS) fashioned like the **optim** function. That means, LHS is performed to optimize a target function, i.e. returning the sample with the lowest function value.

**Usage**

```
spotOptimLHS(par, fn, gr = NULL, lower, upper, control, ...)
```

**Arguments**

- **par**: is a point (vector) in the decision space of fn. Points in par will be added to the design created by LHS.
- **fn**: is the target function of type \( y = f(x, \ldots) \)
gradient function, gr is not used (yet)

is a vector that defines the lower boundary of search space

is a vector that defines the upper boundary of search space

is a list of additional settings. See details.

additional parameters to be passed on to fn

The control list contains:

number of design points created

number of designs created during creation of a well spread design

whether or not fn can evaluate multiple points at once, defaults to FALSE

This function returns a list with:

parameters of the found solutions, e.g. the Pareto set

target function values of the found solutions, e.g. the Pareto front

number of evaluations of fn

SPOT provides some predictors to create a meta-model of the function or algorithm to be analyzed. Nevertheless the user may provide and include his own predictors. To solve this task the user must follow these instructions carefully:

1) The function the user wants to include must be an R-function provided in the R workspace
2) use the function spotInstandLoadPackages to add the packages that are required for your function, just make it the first line in your function.
3) adapt the configuration file, two parameters are to be included/changed:

this example assumes the existence of a function with name "myPredictLm" in the workspace

spotPredict...
See Also

Predictors that are shipped with SPOT are: spotPredictLm, spotPredictMlegp spotPredictRandomForest, spotPredictTree, spotPredictTgp, please check these examples for the correct input parameters and the structure of the return value before you include your own predictors.

The Options of the configuration file (.conf) are described in spotGetOptions **************************************************

---

spotPredictCoForrester

Meta Model Interface: Forrester’s Co-Kriging

Description

Interface to the Co-Kriging model based on Matlab code by Forrester et al. 2008. Make sure to define a cheap, correlated function to be used here in spotConfig$seq.forr.co.fn. This should be in form of \( y = f(x) \). This function is not allowed to yield the same values as the actual (expensive) target function.

Usage

```r
spotPredictCoForrester(rawB, mergedB, design, spotConfig, 
fit = NULL)
```

Arguments

- `rawB` unmerged data
- `mergedB` merged data
- `design` new design points which should be predicted
- `spotConfig` global list of all options, needed to provide data for calling functions. The model specific settings in this list are:
  - `spotConfig$seq.forr.loval` lower boundary for theta, default is \( 1 \times 10^{-3} \)
  - `spotConfig$seq.forr.upval` upper boundary for theta, default is 100
  - `spotConfig$seq.forr.opt.p` boolean that specifies whether the exponents \( (p) \) should be optimized. Else they will be set to two. Default value is FALSE. Default is highly recommended as the implementation of this feature is not yet well tested and might be faulty.
  - `spotConfig$seq.forr.algtheta` algorithm used to find theta, default is "optim-L-BFGS-B". Else, any from the list of possible method values in spotOptimizationInterface can be chosen.
  - `spotConfig$seq.forr.budgetalgtheta` budget for the above mentioned algorithm, default is 100. The value will be multiplied with the length of the model parameter vector to be optimized. `spotConfig$seq.forr.reinterpolate` boolean that specifies whether re-interpolation should be used during the prediction process. Default value is FALSE. Setting this to TRUE is recommended, when an error estimate of nearly zero is desired at sample locations, regardless of chosen regularization constant (nugget). Please note that prediction with interpolation will take longer than without.
spotConfig$seq.forr.savetheta boolean that specifies whether the exponents (p) should be optimized. Else they will be set to two. Default value is \texttt{FALSE}. Default is recommended since this feature not yet well tested, and might lead to a preference of local optima.

\textbf{fit}

if an existing model fit is supplied, the model will not be build based on data, but only evaluated with the model fit (on the design data). To build the model, this parameter has to be \texttt{NULL}. If it is not \texttt{NULL} the parameters \texttt{mergedB} and \texttt{rawB} will not be used at all in the function.

\textbf{Value}

returns the list \texttt{spotConfig} with two new entries:

- \texttt{spotConfig$seq.modelFit} fit of the model used with the predictor functions
- \texttt{spotConfig$seq.largeDesignY} the y values of the design, evaluated with the fit

\textbf{References}


\textbf{See Also}

\texttt{forrCoBuilder} \texttt{forrCoRegPredictor}

\begin{tabular}{ll}
\textbf{spotPredictDace} & \textit{Meta Model Interface: DACE Kriging} \\
\end{tabular}

\textbf{Description}

This Kriging meta model is based on DACE (Design and Analysis of Computer Experiments). It allows to choose different regression and correlation models. If multiple response variables are present, a DACE model for each will be created for the purpose of multi objective optimization.

\textbf{Usage}

\begin{verbatim}
spotPredictDace(rawB, mergedB, design, spotConfig, fit = NULL)
\end{verbatim}
Arguments

rawB                  unmerged data
mergedB               merged data (aggregation of repeated design points)
design                new design points which should be predicted
spotConfig            global list of all options, needed to provide data for calling functions. This also
                       contains a list, with settings for Forrester:
                       spotConfig$seq.dace.budget Budget for MLE of parameters, default is 100.
                       The value will be multiplied with the length of the model parameter vector to
                       be optimized. (which means 100*i evaluations, where i depends on the problem
                       dimension as well as the correlation function and whether a nugget value is to
                       be determined)
                       spotConfig$seq.dace.tol Tolerance stopping criterion for MLE, default is
                       1e-6
                       spotConfig$seq.dace.regr Regression function to be used: regpoly0 (de-
                       fault), regpoly1, regpoly2
                       spotConfig$seq.dace.corr Correlation function to be used: corrnoisykriging
                       (default), cokriging, corrnoisygauss, corrgauss, correxp, correxp2,
                       corrlin, corrcubic, corr spherical, corr spline
                       spotConfig$seq.dace.nugget Value for nugget. Default is -1, which means
                       the nugget will be optimized during MLE. Else it can be fixed in a range between
                       0 and 1.
fit                    if an existing model fit is supplied, the model will not be build based on data,
                       but only evaluated with the model fit (on the design data). To build the model,
                       this parameter has to be NULL. If it is not NULL the parameters mergedB and
                       rawB will not be used at all in the function.

Value

returns the list spotConfig with two new entries:
spotConfig$seq.modelFit fit of the model used with dacePredictor
spotConfig$seq.largeDesignY the y values of the design, evaluated with the fit

Author(s)

The authors of the original DACE Matlab toolbox http://www2.imm.dtu.dk/~hbn/dace/ are
Hans Bruun Nielsen <hbn@imm.dtu.dk>, Soren Nymand Lophaven and Jacob Sondergaard.
Extension of the Matlab code by Tobias Wagner <wagner@isf.de>.
Porting and adaptation to R and further extensions by Martin Zaefferer <martin.zaefferer@fh-koeln.de>.

References

S.~Lophaven, H.~Nielsen, and J.~Sondergaard. DACE—A Matlab Kriging Toolbox. Technical Re-
port IMM-REP-2002-12, Informatics and Mathematical Modelling, Technical University of Den-
mark, Copenhagen, Denmark, 2002.

See Also
dacePredictor daceBuilder
spotPredictDice  

**Meta Model Interface: Dice Kriging**

**Description**

Kriging meta model based on the DiceKriging package. It usually provides good prediction performance, but is rather unstable.

**Usage**

```r
spotPredictDice(rawB, mergedB, design, spotConfig, 
fit = NULL)
```

**Arguments**

- `rawB`: matrix of raw x and y values
- `mergedB`: matrix of merged x and y values, does not have replicate entries
- `design`: design points to be evaluated by the meta model
- `spotConfig`: the list of all parameters is given.
- `fit`: if an existing model fit is supplied, the model will not be build based on data, but only evaluated with the model fit (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.

**Value**

returns the list `spotConfig` with two new entries:

- `spotConfig$seq.modelFit`: fit of the model used with `predict()`
- `spotConfig$seq.largeDesignY`: the y values of the design, evaluated with the fit

**See Also**

- **SPOT**

spotPredictEarth  

**Meta Model Interface: Multivariate Adaptive Regression Spline**

**Description**

Prediction based on earth package, using Multivariate Adaptive Regression Spline models. Can be used both for single and multi objective SPOT.
Usage

```r
spotPredictEarth(rawB, mergedB, design, spotConfig,
  fit = NULL)
```

Arguments

- `rawB`: unmerged data
- `mergedB`: merged data
- `design`: new design points which should be predicted
- `spotConfig`: global list of all options, needed to provide data for calling functions
- `fit`: if an existing model fit is supplied, the model will not be build based on data, but only evaluated with the model fit (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.

Details

This is a model that can incorporate parameters which are marked as FACTORS (i.e. categorical parameters) in the region of interest, see `spotROI`. Please note that the design used to train the MARS model should contain all levels of the factor variable. FACTORS are not ordered, and therefore are impossible to extrapolate on. If new data is given in the design variable which contains unseen FACTOR levels, please note that this will probably create NA values in the prediction. (With the exception that no NA values are created if the concerned FACTOR is not selected as a predictor by `earth`) NA values might yield errors in your SPOT run, ending it prematurely. It is therefore recommended to build a initial design which contains at least one example of each FACTOR level.

Value

returns the list `spotConfig` with two new entries:

- `spotConfig$seq.modelFit` fit of the earth model used with `predict()`
- `spotConfig$seq.largeDesignY` the y values of the design, evaluated with the fit

See Also

- `spotPredictLmFactor`

---

Meta Model Interface: Support Vector Machine

Description

Meta model based on svm function in the e1071-package, which builds a support vector machine for regression.
spotPredictForrester

Usage

spotPredictEsvm(rawB, mergedB, design, spotConfig, 
    fit = NULL)

Arguments

rawB unmerged data
mergedB merged data
design new design points which should be predicted
spotConfig global list of all options, needed to provide data for calling functions
fit if an existing model fit is supplied, the model will not be build based on data, 
    but only evaluated with the model fit (on the design data). To build the model, 
    this parameter has to be NULL. If it is not NULL the parameters mergedB and 
    rawB will not be used at all in the function.

Value

returns the list spotConfig with two new entries:
    spotConfig$seq.modelFit fit of the model used with predict()
    spotConfig$seq.largeDesignY the y values of the design, evaluated with the fit

spotPredictForrester Meta Model Interface: Forrester's Kriging

Description

Interface to the Kriging model based on Matlab code by Forrester et al. 2008. Can be used both for 
    single and multi objective SPOT.

Usage

spotPredictForrester(rawB, mergedB, design, spotConfig, 
    fit = NULL)

Arguments

rawB unmerged data
mergedB merged data
design new design points which should be predicted
spotConfig global list of all options, needed to provide data for calling functions. The model 
    specific settings in this list are
    spotConfig$seq.forr.loval lower boundary for theta, default is 1e-3
    spotConfig$seq.forr.upval upper boundary for theta, default is 100
    spotConfig$seq.forr.opt.p boolean that specifies whether the exponents (p)
should be optimized. Else they will be set to two. Default value is FALSE. Default is highly recommended as the implementation of this feature is not yet well tested and might be faulty.

spotConfig$seq.forr.algtheta algorithm used to find theta, default is "optim-L-BFGS-B". Else, any from the list of possible method values in spotOptimizationInterface can be chosen.

spotConfig$seq.forr.budgetalgtheta budget for the above mentioned algorithm, default is 100. The value will be multiplied with the length of the model parameter vector to be optimized. spotConfig$seq.forr.reinterpolate boolean that specifies whether re-interpolation should be used during the prediction process. Default value is FALSE. Setting this to TRUE is recommended, when an error estimate of nearly zero is desired at sample locations, regardless of chosen regularization constant (nugget). Please note that prediction with interpolation will take longer than without.

spotConfig$seq.forr.savetheta boolean that specifies whether the exponents (p) should be optimized. Else they will be set to two. Default value is FALSE. Default is recommended since this feature not yet well tested, and might lead to a preference of local optima.

fit if an existing model fit is supplied, the model will not be build based on data, but only evaluated with the model fit (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.

Value

returns the list spotConfig with two new entries:

spotConfig$seq.modelFit fit of the model used with the predictor functions
spotConfig$seq.largeDesignY the y values of the design, evaluated with the fit

References


See Also

forrBuilder forrRegPredictor forrReintPredictor

spotPredictGausspr Meta Model Interface: Gaussian Processes

Description

Gaussian processes predictor based on gauspr function in kernlab package.
Usage

```r
spotPredictGausspr(rawB, mergedB, design, spotConfig, 
fit = NULL)
```

Arguments

- `rawB` unmerged data
- `mergedB` merged data
- `design` new design points which should be predicted
- `spotConfig` global list of all options, needed to provide data for calling functions
- `fit` if an existing model fit is supplied, the model will not be build based on data, but only evaluated with the model fit (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.

Value

returns the list `spotConfig` with two new entries:

- `spotConfig$seq.modelFit` fit of the model used with `predict()`
- `spotConfig$seq.largeDesignY` the y values of the design, evaluated with the fit

---

### Meta Model Interface: Fields Kriging

**Description**

Kriging meta model based on fields package.

**Usage**

```r
spotPredictKrig(rawB, mergedB, design, spotConfig, 
fit = NULL)
```

**Arguments**

- `rawB` unmerged data
- `mergedB` merged data
- `design` new design points which should be predicted
- `spotConfig` global list of all options, needed to provide data for calling functions
- `fit` if an existing model fit is supplied, the model will not be build based on data, but only evaluated with the model fit (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.
spotPredictKsvm

Value
returns the list spotConfig with two new entries:
spotConfig$seq.modelFit fit of the Krig model used with predict()
spotConfig$seq.largeDesignY the y values of the design, evaluated with the fit

Description
Meta model based on ksvm function in kernlab package, which builds a support vector machine for regression.

Usage
spotPredictKsvm(rawB, mergedB, design, spotConfig, fit = NULL)

Arguments
rawB       unmerged data
mergedB    merged data
design     new design points which should be predicted
spotConfig global list of all options, needed to provide data for calling functions
fit        if an existing model fit is supplied, the model will not be build based on data, but only evaluated with the model fit (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.

Value
returns the list spotConfig with two new entries:
spotConfig$seq.modelFit fit of the model used with predict()
spotConfig$seq.largeDesignY the y values of the design, evaluated with the fit
spotPredictLm

Meta Model Interface: Linear Model

Description
A linear prediction model, which will use higher order interactions if data is sufficient. Can be used both for single and multi objective SPOT.

Usage

spotPredictLm(rawB, mergedB, design, spotConfig, fit = NULL)

Arguments
rawB: unmerged data
mergedB: merged data
design: new design points which should be predicted
spotConfig: global list of all options, needed to provide data for calling functions
fit: if an existing model fit is supplied, the model will not be build based on data, but only evaluated with the model fit (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.

Details
This function implements a linear model for prediction. Depending on the numbers of variables either no interactions, interaction between the variables may be used or a full quadratic model is provided.

Value
returns the list spotConfig with two new entries:
spotConfig$seq.modelFit fit of the model used with predict()
spotConfig$seq.largeDesignY the y values of the design, evaluated with the fit

See Also
SPOT
spotPredictLmFactor

Meta Model Interface: Linear model with factors for SPOT

Description

This function uses the lm function shipped with R, while spotPredictLm uses the rsm package.

Usage

spotPredictLmFactor(rawB, mergedB, design, spotConfig, 
fit = NULL)

Arguments

rawB unmerged data
mergedB merged data
design new design points which should be predicted
spotConfig global list of all options, needed to provide data for calling functions
fit if an existing model fit is supplied, the model will not be build based on data, 
but only evaluated with the model fit (on the design data). To build the model, 
this parameter has to be NULL. If it is not NULL the parameters mergedB and 
rawB will not be used at all in the function.

Details

The linear model approach described here can incorporate parameters which are marked as FACTORS (i.e. categorical parameters) in the region of interest, see spotROI. Please note that the design used to train the linear model should contain all levels of the factor variable. FACTORS are not ordered, and therefore are impossible to extrapolate on. If new data is given in the design variable which contains unseen FACTOR levels, please note that this will probably create NA values in the prediction. NA values might yield errors in your SPOT run, ending it prematurely. It is therefore recommended to build a initial design which contains at least one example of each FACTOR level.

Value

returns the list spotConfig with two new entries:
spotConfig$seq.modelFit fit of the earth model used with predict()
spotConfig$seq.largeDesignY the y values of the design, evaluated with the fit

See Also

spotPredictLm
spotPredictMCO  

Meta Model Interface: Multi Criteria Modeling

Description

This interface function is supposed to be used for Multi Criteria Problems. It can be employed when the user wants to specify different models for each of the objectives, instead of modeling all the objectives with the same technique. The user has therefore to specify a list of configurations, where the different models and their settings are specified.

Usage

```r
spotPredictMCO(rawB, mergedB, design, spotConfig, fit = NULL)
```

Arguments

- `rawB`: unmerged data
- `mergedB`: merged data
- `design`: new design points which should be predicted
- `spotConfig`: global list of all options, needed to provide data for calling functions. The most important elements in this list are here:
  - `spotConfig$mco.configs` list of model configurations, e.g., `list(seq.predictionModel.func="spotpredictforrester", list(seq.predictionModel.func="spotpredictlm", lambdaD=upvalD))`
  - In this example, two Kriging models are specified for each of two objectives, but with different settings for the lower boundary of lambda. Else, different models could be specified, e.g., `list(seq.predictionModel.func="spotpredictforrester"), list(seq.predictionModel.func="spotpredictlm", lambdaD=upvalD))`
- `fit`: if an existing model fit is supplied, the model will not be build based on data, but only evaluated with the model fit (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.

Value

returns the list `spotConfig` with two new entries:
- `spotConfig$seq.modelFit` fit of the model used with the predictor functions
- `spotConfig$seq.largeDesignY` the y values of the design, evaluated with the fit

References


See Also

- `forrBuilder`
- `forrRegPredictor`
- `forrReintPredictor`
Meta Model Interface: Maximum Likelihood Estimation for Gaussian Processes, Kriging

Description

Kriging model based on mlegp package. This function uses two settings, which are stored in the spotConfig parameter:

- spotConfig$seq.mlegp.constantMean
  - Use constant mean (mu) in mlegp (=1) or linear model (=0); 1 by default
- spotConfig$seq.mlegp.min.nugget
  - minimum value of nugget term; 0 by default

If those settings are not in spotConfig their mentioned defaults will be used.

If the numeric value of spotConfig$mlegp.reduce is smaller than the observations in mergedB, spotConfig$mlegp.reduce will specify how many samples should be drawn without replacement from mergedB. This can prevent explosion of time consumption in this function. Mlegp can be used both for single and multi objective SPOT.

Usage

spotPredictMlegp(rawB, mergedB, design, spotConfig, fit = NULL)

Arguments

- rawB
  - matrix of raw x and y values
- mergedB
  - matrix of merged x and y values, does not have replicate entries
- design
  - design points to be evaluated by the meta model
- spotConfig
  - the list of all parameters is given.
- fit
  - if an existing model fit is supplied, the model will not be build based on data, but only evaluated with the model fit (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.

Value

returns the list spotConfig with two new entries:

- spotConfig$seq.modelFit fit of the model used with predict()
- spotConfig$seq.largeDesignY the y values of the design, evaluated with the fit

See Also

SPOT
**spotPredictMLP**  
*Meta Model Interface: Multi-layer Perceptron*

**Description**
Meta model based on monmlp package for multi layer perceptrons

**Usage**
```r
spotPredictMLP(rawB, mergedB, design, spotConfig,  
fit = NULL)
```

**Arguments**
- `rawB` unmerged data
- `mergedB` merged data
- `design` new design points which should be predicted
- `spotConfig` global list of all options, needed to provide data for calling functions
- `fit` if an existing model fit is supplied, the model will not be build based on data, but only evaluated with the model fit (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.

**Value**
returns the list `spotConfig` with two new entries:
- `spotConfig$seq.modelFit` fit of the earth model used with predict()
- `spotConfig$seq.largeDesignY` the y values of the design, evaluated with the fit

**spotPredictQrnn**  
*Meta Model Interface: Quantile Regression Neural Network*

**Description**
This meta model uses the "qrnn" package to build a quantile regression neural network.

**Usage**
```r
spotPredictQrnn(rawB, mergedB, design, spotConfig,  
fit = NULL)
```
Arguments

rawB unmerged data
mergedB merged data
design new design points which should be predicted
spotConfig global list of all options, needed to provide data for calling functions
fit if an existing model fit is supplied, the model will not be build based on data, but only evaluated with the model fit (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.

Value

returns the list spotConfig with two new entries:
  spotConfig$seq.modelFit fit of the earth model used with predict()
  spotConfig$seq.largeDesignY the y values of the design, evaluated with the fit

Description

A prediction model interface based on randomForest package, using a random forest for regression. Can be used both for single and multi objective SPOT.

Usage

spotPredictRandomForest(rawB, mergedB, design, spotConfig, fit = NULL)

Arguments

rawB unmerged data
mergedB merged data
design new design points which should be predicted
spotConfig global list of all options, needed to provide data for calling functions
fit if an existing model fit is supplied, the model will not be build based on data, but only evaluated with the model fit (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.
Details

This is a model that can incorporate parameters which are marked as FACTORS (i.e. categorical parameters) in the region of interest, see spotROI. Please note that the design used to train the RF model should contain all levels of the factor variable. FACTORS are not ordered, and therefore are impossible to extrapolate on. If new data is given in the design variable which contains unseen FACTOR levels, please note that this will probably create NA values in the prediction. NA values might yield errors in your SPOT run, ending it prematurely. It is therefore recommended to build a initial design which contains at least one example of each FACTOR level.

Value

returns the list spotConfig with two new entries:
- spotConfig$seq.modelFit fit of the model used with predict()
- spotConfig$seq.largeDesignY the y values of the design, evaluated with the fit

See Also

SPOT
Value

returns the list `spotConfig` with two new entries:
- `spotConfig$seq.modelFit` fit of the model used with `predict()`
- `spotConfig$seq.largeDesignY` the y values of the large design, evaluated with the fit

See Also

SPOT

---

`spotPredictTgp`  
*Meta Model Interface: Treed Gaussian Processes*

**Description**

This function implements a model for prediction, based on Mat’s tgp

**Usage**

```r
spotPredictTgp(rawB, mergedB, design, spotConfig, fit = NULL)
```

**Arguments**

- `rawB`  
  unmerged data
- `mergedB`  
  merged data
- `design`  
  new design points which should be predicted
- `spotConfig`  
  global list of all options, needed to provide data for calling functions
- `fit`  
  if an existing model fit is supplied, the model will not be build based on data, but only evaluated with the model fit (on the design data). To build the model, this parameter has to be NULL. If it is not NULL the parameters `mergedB` and `rawB` will not be used at all in the function.

**Value**

returns the list `spotConfig` with a new entry:
- `spotConfig$seq.modelFit` fit of the model used with `predict()`
- `spotConfig$seq.largeDesignY` the y values of the design, evaluated with the fit

See Also

SPOT
spotPredictTree  

Meta Model Interface: Tree

Description

A prediction model based on rpart, using a single tree model.

Usage

spotPredictTree(rawB, mergedB, design, spotConfig, fit = NULL)

Arguments

rawB  
unmerged data

mergedB  
merged data

design  
new design points which should be predicted

spotConfig  
global list of all options, needed to provide data for calling functions

fit  
if an existing model fit is supplied, the model will not be build based on data, but only evaluated with the model fit (on the largeDesign data). To build the model, this parameter has to be NULL. If it is not NULL the parameters mergedB and rawB will not be used at all in the function.

Value

returns the list spotConfig with two new entries:

spotConfig$seq.modelFit fit of the model used with predict()

spotConfig$seq.largeDesignY the y values of the design, evaluated with the fit

See Also

SPOT

spotReadBstFile  

Read .bst File

Description

This function reads the .bst file of the current project. The data is used for plots and reports. Will usually not be used if no result/best files are written.

Usage

spotReadBstFile(spotConfig)
spotReadRoi

Arguments

- **spotConfig**: the list of all parameters is given, used here `spotReadBstFile`: the name of the .bst file to be read. `spotConfig$sio.columnSep`: Separator for the columns.

Value

- **data.frame bstData**: `bstData` holds a column "y" with the results and all columns with column-names derived from the .roi file (should be the parameters of the algorithm).

See Also

- SPOT spot spotPlotBst

---

**spotReadRoi**

**Spot Read ROI**

Description

Help function `spotReadRoi` reads region of interest from the .roi-file.

Usage

```r
spotReadRoi(roiFile, sep, verbosity = 0)
```

Arguments

- **roiFile**: this file contains the roi.
- **sep**: this is used as a column separator.
- **verbosity**: io.verbosity for this specified output string.

Value

- **data.frame aroi**: `aro` contains the data from the roi file.
spotReport3d 3d Plot of Meta Model - Report Function

Description

Function to generate a 3d surface plot of the predicted meta model.

Usage

spotReport3d(spotConfig)

Arguments

spotConfig  the configuration list of all spot parameters.

The parameter spotConfig$report.interactive=TRUE will be set as default if not contained in the list. That means, by default the user will be asked to specify which parameters will be varied when the report is started. This is done in a small twiddler GUI. If the user wants to specify which parameters should be plotted against each other, before starting the report, he can set the parameter spotConfig$report.aIndex and spotConfig$report.bIndex. They should be two different integer numbers. They will only be used if spotConfig$report.interactive is FALSE. By default they will be set to 1 and 2, so the first two parameters in the ROI will be plotted. In case of a multi objective problem, spotConfig$report.cIndex will determine which point from the Pareto front will be used to determine values of parameters not on the axis of the plot.

Details

This report function uses the parameter spotConfig$seq.modelFit to plot the predicted model. If spotConfig$seq.modelFit is NULL, the model is generated, based on spotConfig$seq.predictionModel.func. It is not recommended to use this function at the end of an "auto" run of SPOT, make sure to save results first. By default, twiddler will be used to let the user specify which of the parameters should be varied in the plot. Values not varied in the graph are fixed to their "best" value according to the current Results.

See Also

spotSurfContour
Description

Function to generate a contour plot of the predicted meta model.

Usage

spotReportContour(spotConfig)

Arguments

spotConfig the configuration list of all spot parameters.
The parameter spotConfig$report.interactive=TRUE will be set as default if not
contained in the list. That means, by default the user will be asked to specify
which parameters will be varied when the report is started. This is done in a
small twiddler GUI. If the user wants to specify which parameters should be
plotted against each other, before starting the report, he can set the parameter
spotConfig$report.aIndex and spotConfig$report.bIndex. They should be two
different integer numbers. They will only be used if spotConfig$report.interactive
is FALSE. By default they will be set to 1 and 2, so the first two parameters
in the ROI will be plotted. In case of a multi objective problem, spotCon-
fig$report.cIndex will determine which point from the Pareto front will be used
to determine values of parameters not on the axis of the plot.

Details

This report function uses the parameter spotConfig$seq.modelFit to plot the predicted model. If
spotConfig$seq.modelFit is NULL, the model is generated, based on spotConfig$seq.predictionModel.func.
It is not recommended to use this function at the end of an "auto" run of SPOT, make sure to save
results first. By default, twiddler will be used to let the user specify which of the parameters should
be varied in the plot. Values not varied in the graph are fixed to their "best" value according to the
current Results.

See Also

spotSurfContour
spotReportDefault       Default Report

Description

Function generates a simple report.

Usage

spotReportDefault(spotConfig)

Arguments

spotConfig       the configuration list of all spot parameters

Details

This function is used when no report function is requested by the user. It creates some text output and also draws a tree, printing it to screen or pdf. If the report.io.pdf setting is TRUE the graphic is printed to a pdf file (usually named like your .conf file, and placed in the same folder) if report.io.screen is set TRUE the graphic is printed to the screen. Both can be FALSE or TRUE at the same time. If the user does not specify those values, the defaults will be used as shown in spotGetOptions, which means there will be only screen output, and no pdf.

In case of multi objective optimization this function will report the hyper volume indicator, and write the Pareto front and the Pareto set to spotConfig$mco.val and spotConfig$mco.par. spotReportDefault is currently the only recommendable report function for multi objective SPOT, besides custom functions created by users.

Value

list spotConfig with changed values

See Also

spot, spotStepReport

spotReportEarth       Earth Report

Description

Generates a report for results of a SPOT run with spotPredictEarth

Usage

spotReportEarth(spotConfig)
Arguments

spotConfig the configuration list of all spot parameters

Value

list spotConfig with changed values

See Also

spot spotStepReport spotPredictEarth

Description

This function does a random effects model analysis of a SPOT run, using the lme4 package. It is supposed to be used for the case of Multiple Algorithms Multiple Problems. The SPOT demo 22 provides a simple example for mixed model analysis. Call for the demo is: demo(spotDemo22MixedModelMAMP, ask=F).

Usage

spotReportMAMP(spotConfig)

Arguments

spotConfig the configuration list of all spot parameters

Value

list spotConfig with changed values

See Also

spotReportSAMP spotStepReport
spotReportMetaDefault  Default Report for Meta Runs

Description
Function to generate a simple report for meta runs.

Usage
spotReportMetaDefault(spotConfig)

Arguments
spotConfig  the configuration list of all spot parameters

Details
This function draws a scatterplot matrix (based on car), printing it to screen or pdf. If the report.io.pdf setting is TRUE the graphic is printed to a pdf file (usually named like your .conf file, and placed in the same folder) if report.io.screen is set TRUE the graphic is printed to the screen. Both can be FALSE or TRUE at the same time. If the user does not specify those values, the defaults will be used as shown in spotGetOptions, which means there will be only screen output, and no pdf.

See Also
SPOT spot spotStepReport

spotReportSAMP  SAMP Model Report

Description
This function does a random effects model analysis of a SPOT run, using the lme4 package. It is supposed to be used for the case of Single Algorithm Multiple Problems. The SPOT demo 21 provides a simple example for mixed model analysis. Call for the demo is: demo(spotDemo21MixedModelSAMP, ask=F).

Usage
spotReportSAMP(spotConfig)

Arguments
spotConfig  the configuration list of all spot parameters

Value
list spotConfig with changed values
spotReportSens

See Also

spotReportMAMP spotStepReport

spotReportSens | Sensitivity Report

Description

Function to generate a report with sensitivity plot.

Usage

spotReportSens(spotConfig)

Arguments

spotConfig | the configuration list of all spot parameters

Details

The sensitivity curves are based on a metamodel which is a random forest with 100 trees fitted to the result points from RES-file. The plot contains: x-axis: ROI for each parameter normalized to [-1,1] y-axis:

See Also

SPOT spot spotStepReport

spotROI | Region Of Interest Constructor

Description

This function can be used to construct a region of interest (ROI), to be passed on to spot(). Note, upper == lower is allowed, but no element in upper should be smaller than the corresponding element in lower.

Usage

spotROI(lower, upper, type = "FLOAT", varnames = NULL, dimROI = NULL)
**Arguments**

- **lower**: vector or a single number, specifying lower boundary of ROI variables
- **upper**: vector or a single number, specifying upper boundary of ROI variables
- **type**: vector of strings or single string, specifying the data type of the variables. Can be: "FLOAT", "INT", "FACTOR"
- **varnames**: vector or NULL, telling the name of each variable. Can be NULL, so that default variable names will be used.
- **dimROI**: defines the number of variables. If dimROI is set (not NULL), the other vectors should have length=dimROI, or length=1.

**Value**

returns a data frame containing the ROI ubfirmatuib

**Examples**

```
## without varnames or dimROI
alg.roi <- spotROI(c(0,0),c(1,1),c("FLOAT","FLOAT"))
## with varnames
alg.roi <- spotROI(c(0,0),c(1,1),c("FLOAT","FLOAT"),c("VARX1","VARX3"))
## lower and upper only
alg.roi <- spotROI(c(0,1,-2),10)
alg.roi <- spotROI(c(0,1,-2),c(2,3,100))
## with dimROI
alg.roi <- spotROI(-10,10,"FLOAT",dimROI=4)
## with varnames and dimROI
alg.roi <- spotROI(-10,10,"FLOAT",c("x1","x2","x3","x4"),dimROI=4)
```

---

**spotSelectionCriteria**  
*Model selection and error estimation*

**Description**

This is a number of functions for model selection and error estimation:
- **rsq**: R-Squared, can be adjusted with a complexity measure
- **sse**: Sum of Squared Errors
- **sae**: Sum of Absolute Errors
- **mse**: Mean Squared Errors
- **rmse**: Root Mean Squared Errors
- **aic**: Akaike Information Criterion, with bias adjustment for small sample sizes
- **scaled**: this refers to an approach that scales the data before error calculation, see Keijzer (2004).
**Usage**

```r
spotSelectionRsq(yi, fi);
spotSelectionAdjustedRsq(yi, fi, p);
spotSelectionAic(yi, fi, p); spotSelectionSse(yi, fi);
spotSelectionSae(yi, fi); spotSelectionMse(yi, fi);
spotSelectionMae(yi, fi); spotSelectionRmse(yi, fi);
spotSelectionScaledSse(yi, fi);
spotSelectionScaledMse(yi, fi);
spotSelectionScaledRmse(yi, fi);
```

**Arguments**

- `yi` sampled values vector
- `fi` predicted values vector
- `p` complexity measure or number of regressors

**Value**

a scalar value is returned

**References**


---

**Description**

This two-objective infill criterion is the Expected Improvement of the S-metric. That is, it aggregates the predicted objective values by an exact calculation of the Expected Improvement EI in hypervolume. As this gets more complex and time-consuming for higher dimensional objective spaces, this is only implemented for the two-objective case. An approximation approach for higher dimensional problems exists, but is not yet implemented in SPOT.

**Usage**

```r
spotSExI2d(P, r, mu, s)
```
**Arguments**

- **P**
  Approximation set: provide f1,f2 coordinates of current Pareto front approximation

- **r**
  Reference point: used for computing the hypervolume

- **mu**
  Mean vector: mean value of predictive distribution (e.g. from Gaussian process), f1, f2

- **s**
  Standard deviations of predictive distribution

**Value**

returns the EI for each row in resy

**Author(s)**

(c) Michael Emmerich and Andre Deutz, LIACS, Leiden University, 2010
<emmerich@liacs.nl>, <deutz@liacs.nl>
R port by Patrick Koch, Cologne University of Applied Sciences
<patrick.koch@fh-koeln.de>

**References**

M. Emmerich, A.H. Deutz, J.W. Klinkenberg: The computation of the expected improvement in dominated hypervolume of Pareto front approximations, LIACS TR-4-2008, Leiden University, The Netherlands

**See Also**

spotInfillSExI2d

**Examples**

```r
print(spotSExI2d(data.frame(x1=c(0,1,2),x2=c(2,1,0)),c(3,3),c(0,0),c(0.1,0.1)))
# should be approx. 3.08
print(spotSExI2d(data.frame(x1=c(1,2),x2=c(2,1)),c(11,11),c(10,10),c(4,4)))
# should be approx. 0.0726
```

---

**Description**

Straight forward SMS-EMOA implementation. This function is used to optimize several surrogate models when doing multi objective optimization with SPOT. See: spotParetoOptMulti.
**Usage**

```r
spotSmsEmoa(f, lower, upper, ..., 
control = list(mu = 100L, sbx.n = 15, sbx.p = 0.7, pm.n = 25, pm.p = 0.3))
```

**Arguments**

- `f`: target function to be optimized of type \( f(x) = y \) where both \( x \) and \( y \) are vectors. The target function should return a vector of \( \text{length}(y) \) containing NAs if the input vector \( x \) contains NA values.
- `lower`: the lower boundary vector of the decision space
- `upper`: the upper boundary vector of the decision space
- `...`: further settings relayed to `f`
- `control`: list of parameters (defaults are: `mu`=100L, `sbx.n`=15, `sbx.p`=0.7, `pm.n`=25, `pm.p`=0.3)

**Value**

list with archive of solutions, active Pareto front and others

**Author(s)**

O. Mersmann

**References**


See the following link for up to date version of this implementation: [https://git.p-value.net/emoa.git/plain/examples/sms_emoa.r](https://git.p-value.net/emoa.git/plain/examples/sms_emoa.r)

---

**Description**

Straight forward SMS-EMOA implementation, but supported by a Kriging model when selecting new individuals.

**Usage**

```r
spotSmsEmoaKriging(f, lower, upper, ..., 
control = list(mu = 100L, sbx.n = 15, sbx.p = 0.7, pm.n = 25, pm.p = 0.3))
```
**Arguments**

- **f**
  
  target function to be optimized of type \( f(x) = y \) where both \( x \) and \( y \) are vectors. The target function should return a vector of length(\( y \)) containing NAs if the input vector \( x \) contains NA values.

- **lower**
  
  the lower boundary vector of the decision space

- **upper**
  
  the upper boundary vector of the decision space

- **...**
  
  further settings relayed to \( f \)

- **control**
  
  list of parameters (defaults are: \( \text{mu}=100L, \text{sbx.n}=15, \text{sbx.p}=0.7, \text{pm.n}=25, \text{pm.p}=0.3 \))

**Value**

list with archive of solutions, active Pareto front and others

**See Also**


See the following link for up to date version of this implementation: [https://git.p-value.net/emoa.git/plain/examples/sms_emoa.r](https://git.p-value.net/emoa.git/plain/examples/sms_emoa.r)
spotStepInitial

See Also

SPOT spot spotStepInitial spotStepSequential spotStepRunAlg spotStepReport spotGetOptions

spotStepInitial  SPOT Step: Initialize (First SPOT-Step)

Description

Creates a sequential design based on the results derived so far. Therefore it is essential to have another design evaluated before and have a.res file to use. Afterwards the design is extended by 4 columns: CONFIG, REPEATS, STEP, SEED

Usage

spotStepInitial(spotConfig)

Arguments

spotConfig  the list of all parameters is given, but the used ones are:
spotConfig$init.design.func holds the spotCreateDesign<XXX> function to be used for building an initial design.
spotConfig$init.design.size number of points that should be created for the initial design.
spotConfig$init.design.retries gives the number of trials to find a design with the greatest distance between points, (default is 1)
spotConfig$init.design.repeats number of repeats for one initial design-point
spotConfig$alg.seed seed value for reproducible runs
spotConfig$srcPath source path as given when spot() is called (or uses default)
spotConfig$io.verbosity verbosity for command window output, which is passed to the output function

Details

uses the functions spotConfig$init.design.func and link{spotWriteDes} that writes a design to the file <xxx>.des
spotStepMetaOpt \hspace{1em} \textit{SPOT Step Meta} \hspace{1em}

\textbf{Description}

\begin{quote}
Attention: This feature is work in progress, documentation is not up to date.
\end{quote}

\textbf{Usage}

\begin{verbatim}
spotStepMetaOpt(spotConfig)
\end{verbatim}

\textbf{Arguments}

\begin{verbatim}
spotConfig \hspace{1em} the list of all parameters is given
\end{verbatim}

\textbf{Details}

The \textit{meta} task calls \texttt{spotStepMetaOpt} which itself calls \texttt{spot} with several different fixed parameters to provide a mixed optimization mechanism: analyse a fully qualified test of some parameters and the intelligent optimization of other parameters. e.g. the number of the dimension of a problem etc.

To start this step you could for example do this:
\begin{verbatim}
spot("configFileName.conf","meta")
\end{verbatim}

\textbf{See Also}

\begin{verbatim}
spotGetOptions
\end{verbatim}

spotStepReport \hspace{1em} \textit{SPOT Step Report} \hspace{1em}

\textbf{Description}

Forth and last step for SPOT, that is by default a call of \texttt{spotReportDefault}

\textbf{Usage}

\begin{verbatim}
spotStepReport(spotConfig)
\end{verbatim}

\textbf{Arguments}

\begin{verbatim}
spotConfig \hspace{1em} the list of all parameters is given, it is forwarded to the call of the report-function
\end{verbatim}
Details

This step provides a very basic report about the .res-file, based on settings in the spotConfig.
The mainly used parameters of spotConfig is spotConfig$report.func, specifying which report shall be called. The user can specify his own report and should set the value report.func in the configuration file according to the specification rules given. If nothing is set, the default report is used.

See Also

SPOT spot spotReportDefault spotGetOptions

---

**spotStepRunAlg**  
*SPOT Step Algorithm Call*

Description

This is the second SPOT Step after step "initial" - but also needed after each step "sequential", and is a call frame for the algorithm-call.

Usage

spotStepRunAlg(spotConfig, ...)

Arguments

- **spotConfig** the list of all configuration parameters, but most important ones are:
  - spotConfig$alg.func the name of the R target function
  - spotConfig$io.apdFileName filename for the problem definition of the algorithm, first parameter of the generically defined R-function spotConfig$alg.func
  - spotConfig$io.desFileName filename for the input of the algorithm, second parameter of the generically defined R-function spotConfig$alg.func
  - spotConfig$io.resFileName filename for the output of the algorithm third parameter of the generically defined R-function spotConfig$alg.func
  - spotConfig$io.verbosity verbosity for command window output, which is passed to the output function
  - ... additional parameters to be passed on to target function which is called inside alg.func

Details

The algorithm is the heart of what the user must provide, but SPOT should be able to handle them in the most flexible manner. This function is an interface to the algorithm, given as a R-function.

See Also

SPOT spot spotStepInitial spotStepSequential
spotStepSequential  

**SPOT Step Sequential**

**Description**

Third SPOT Step to generate a sequential new design, this is mainly a call of `spotGenerateSequentialDesign`.

**Usage**

```r
spotStepSequential(spotConfig)
```

**Arguments**

- `spotConfig`: the list of all parameters is given, but the used ones are:
  - `spotConfig$io.resFileName` is checked for existence, if not, function fails with error
  - `spotConfig$algSourceSrcPath` needed for the error message
  - `spotConfig$userConfFileName` needed for the error message

**Details**

Creates a sequential design based on the results derived so far. Therefore, it is essential to have another design evaluated before and have a `.res` file to use. It uses the functions `spotGenerateSequentialDesign` and `spotWriteDes` to write a sequential design to the file `<xxx>.des`.

---

**spotSurf3d**  

**spotSurf3d**

**Description**

Simple surface plot in three dimensions, using the rgl package with `persp3` to plot.

**Usage**

```r
spotSurf3d(f = function(x) { sum(x^2) },
  lo = c(0, 0), up = c(1, 1), s = 100, clip = c(NA, NA),
  points, ...)
```
Arguments

- **f**: function to be plotted. The function should either be able to take two vectors or one matrix specifying sample locations, i.e., $z = f(x)$ or $z = f(x_2, x_1)$ where $Z$ is a two column matrix containing the sample locations $x_1$ and $x_2$.

- **lo**: lower boundary for $x_1$ and $x_2$ (defaults to $c(0, 0)$).

- **up**: upper boundary (defaults to $c(1, 1)$).

- **s**: number of samples along each dimension. E.g., $f$ will be evaluated $s^2$ times.

- **clip**: $z$ values smaller than clip[1] and larger than clip[2] are set to NA, to prevent them being visible in the plot. May result in ragged plots, but controls scaling.

- **points**: can be omitted, but if given the points in this matrix are added to the plot.

- **...**: additional parameters passed to $f$.

See Also

- **spotSurfContour**

Examples

```r
spotSurf3d(function(x){apply(x,1,spotBraninFunction)},c(-5,0),c(10,15))
```

Description

Simple surface plot, using the filled.contour function.

Usage

```r
spotSurfContour(f = function(x) { sum(x^2) },
    lo = c(0, 0), up = c(1, 1), s = 100, xlab = "x1",
    ylab = "x2", title = "", levels = NULL, points1,
    points2, ...)
```

Arguments

- **f**: function to be plotted. The function should either be able to take two vectors or one matrix specifying sample locations, i.e., $y = f(x)$ or $y = f(x_2, x_1)$ where $Z$ is a two column matrix containing the sample locations $x_1$ and $x_2$.

- **lo**: lower boundary for $x_1$ and $x_2$ (defaults to $c(0, 0)$).

- **up**: upper boundary (defaults to $c(1, 1)$).

- **s**: number of samples along each dimension. E.g., $f$ will be evaluated $s^2$ times.

- **xlab**: lable of first axis

- **ylab**: lable of second axis
Title

levels number of levels for the plotted function value. Will be set automatically with default NULL.

points1 can be omitted, but if given the points in this matrix are added to the plot in form of dots.

points2 can be omitted, but if given the points in this matrix are added to the plot in form of crosses.

... additional parameters passed to f.

See Also

spotSurf3d

Examples

spotSurfContour(function(x){apply(x,1,spotBraninFunction)},c(-5,0),c(10,15))

Description

help function spotWriteAroi writes actual region of interest to the .aroi-file.

Usage

spotWriteAroi(aroi, verbosity, sep, filename)

Arguments

aroi data frame.

verbosity for values greater than two, a message is given.

sep the column separator used when writing the table to .aroi-file.

filename the filename the aroi should be written to.

Details

Result/Effects: rewrites the actual region of interest-file.
Testfunctions

| Testfunctions | Single objective test functions for SPOT |

### Description

The following target functions are available:

- **spotSphereFunction:**
  multi-dimensional sphere function, one global optimum: Sum[x^2]

- **spotSphere1Function:**
  multi-dimensional sphere function, one global optimum: Sum[i^2*(x[i]-i)^2, i, 1, ndim]

- **spotSixHumpFunction:**
  Two dimensional target function, two global optima the 6-hump camel back function, see [http://www.it.lut.fi/ip/evo/functions/node26.html](http://www.it.lut.fi/ip/evo/functions/node26.html)

- **spotRosenbrockFunction:**
  Two dimensional Rosenbrocks function with one global optimum, see: [http://en.wikipedia.org/wiki/Rosenbrock_function](http://en.wikipedia.org/wiki/Rosenbrock_function). Maple: f := (1-x)^2+100*(y-x^2)^2 plot3d(f, x = -1.5 .. 1.5, y = -.5 .. 2)

- **spotRosenbrockGradientFunction:**
  Gradient of Rosenbrocks function.

- **spotRastriginFunction:**
  Multi-dimensional rastrigin function, one global optimum see [http://en.wikipedia.org/wiki/Rastrigin_function](http://en.wikipedia.org/wiki/Rastrigin_function)

- **spotMexicanHatFunction:**
  Two dimensional MexicanHat function, with a circular valley of global optima

- **spotBraninFunction:**
  Two dimensional Branin function implementation, 3 global optima, see also: [http://www.it.lut.fi/ip/evo/functions/node27.html](http://www.it.lut.fi/ip/evo/functions/node27.html)

- **spotWildFunction:**
  Another test function, y=10*sin(0.3*x)*sin(1.3*x^2) + 0.00001*x^4 + 0.2*x+80

### Usage

```plaintext
spotSphereFunction(x); spotSphere1Function(x);
spotSixHumpFunction(x); spotRosenbrockFunction(x);
spotRosenbrockGradientFunction(x);
spotRastriginFunction(x); spotMexicanHatFunction(x);
spotBraninFunction(x); spotWildFunction(x);
```
Arguments

- `x` vector that will be evaluated by the test-function

Value

- `y` number
  - `y` is the response value of the corresponding `x` vector

See Also

- SPOT spot demo
Index

*Topic **package**
  SPOT-package, 5

corrcubic, 6, 81
correxpg, 6, 81
corrrkrging, 6, 81
corrlin, 6, 81
corrnoisygauss, 6, 81
corrnoisykrging, 6, 81
corrspherical, 6, 81
corrspline, 6, 81
daceBuilder, 6, 7, 8, 81
dacePredictor, 7, 7, 81
demo, 20–24, 74, 116

forrbuilder, 9, 12, 14, 15, 85, 90
forrcobuilder, 11, 13, 15, 80
forrcoregpredictor, 12, 13, 80
forrreintpredictor, 10, 14, 15, 85, 90
optim, 16, 20, 22–24, 70, 72–74, 76, 77

regpoly0, 6, 81
regpoly1, 6, 81
regpoly2, 6, 81

SPOT, 17–24, 74, 82, 88, 91, 94–97, 101–103, 109, 111, 116
SPOT (SPOT=package), 5
spot, 5, 16, 20–24, 70, 71, 73, 74, 97, 100–103, 109–111, 116
SPOT=package, 5
spotAlgEs, 17, 19, 21
spotAlgStart..., 58
spotAlgStartEs, 17, 18, 19, 20, 21
spotAlgStartEsGlg, 19
spotAlgStartEsVar, 19, 20

spotAlgStartRgp, 21
spotAlgStartSann, 22, 23
spotAlgStartSannVar, 20, 22, 23, 24
spotAlgStartSmemoaGlg, 24
spotBrainFunction, 18
spotBrainFunction (Testfunctions), 115
spotCreateDesign..., 25, 58
spotCreateDesignBasicDo, 25, 25, 27–29
spotCreateDesignFactors, 26
spotCreateDesignFrF2, 25, 26, 27, 28, 29
spotCreateDesignLhd, 26, 27, 27, 29
spotCreateDesignLhs, 25–28, 28
spotCreateDesignLhsOpt, 26–29, 29
spotEnsembleMultiAlternate, 30
spotEnsembleMultiAverage, 31
spotEnsembleMultiChoose, 32
spotEnsembleMultiRank, 33, 35
spotEnsembleMultiRankWeighted, 34
spotEnsembleSingleBLAs, 36
spotEnsembleSingleBLAnorm, 37
spotEnsembleSingleEpsGreedy, 38
spotEnsembleSinglePOKER, 39
spotEnsembleSingleRoundSearch, 30, 41
spotEnsembleSingleSoftMax, 41, 42
spotEnsembleSingleUCB1, 43
spotFeedback, 44
spotFeedback.reward.bern, 36, 38, 42, 43
spotFeedback.reward.norm, 37, 39
spotFuncStartBranin, 20, 22–24, 46, 71
spotGenerateSequentialDesign, 70, 112
spotGetOptions, 16, 19–25, 45, 71, 73, 79, 100, 102, 109–111
spotGlgCreate, 19, 24, 49, 50–54, 57
spotGlgCreateN, 50, 50, 52, 53, 55, 57, 58
spotGlgCreateRot, 50, 51, 51, 52, 53, 55
spotGlgCreateRotSearched, 50–52, 52
spotGlgEval, 53, 55, 57
spotGlgEvalN, 54, 54, 55, 58
spotGlgEvalRot, 54, 55, 55
INDEX

spotGlginIt, 50, 54, 55, 56, 58
spotGlginItN, 55, 57, 57
spotGui, 5, 58
spotHelpInterfaces..., 58
spotInfillExpImp, 59
spotInfillHyperVolume, 59
spotInfillLbcbHyperVolume, 59, 60, 61
spotInfillLbcbMulti, 61
spotInfillLbcbSingle, 61
spotInfillProbImp, 62
spotInfillSD, 62
spotInfillSEXI2d, 63, 106
spotInstantAndLoadPackages, 25, 78
spotMexicanHatFunction (Testfunctions), 115
spotModel.func, 64, 65, 66
spotModel.predict, 64, 65, 66
spotModel.train, 64, 65, 66
spotModelDescentLm, 67
spotModelOptim, 67, 67, 70
spotModelParetoOptim, 67, 69, 69
spotOcb, 70
spotOptim, 5, 16, 17, 70, 73, 74, 76–78
spotOptimEs, 72
spotOptimInterface, 46, 47, 71, 73
spotOptimizationInterface, 6, 9, 11, 71, 74, 77–79, 85
spotOptimizationInterfaceMco, 71, 76, 76
spotOptimLHS, 77
spotParetoOptMulti, 106
spotPlotBst, 97
spotPredict..., 58, 78
spotPredictCoForrester, 12, 79
spotPredictDace, 7, 8, 80
spotPredictDice, 82
spotPredictEarth, 82, 101
spotPredictEsvm, 83
spotPredictForrester, 10, 84
spotPredictGausspr, 85
spotPredictKrig, 86
spotPredictKsvm, 87
spotPredictLm, 79, 88, 89
spotPredictLmFactor, 83, 89
spotPredictMCO, 90
spotPredictMlegp, 79, 91
spotPredictMLP, 92
spotPredictOptMulti, 49
spotPredictQRnn, 92
spotPredictRandomForest, 65, 66, 79, 93
spotPredictRandomForestMlegp, 94
spotPredictTgp, 79, 95
spotPredictTree, 79, 96
spotPrepare, 17
spotPrepareSystem, 17
spotRastriginFunction (Testfunctions), 115
spotReadBstFile, 96
spotReadRoi, 97
spotRepeatsOcb, 70
spotReport3d, 98
spotReportContour, 99
spotReportDefault, 100, 110, 111
spotReportEarth, 100
spotReportMAMP, 101, 103
spotReportMetaDefault, 102
spotReportSAMP, 101, 102
spotReportSens, 103
spotROI, 47, 83, 89, 94, 103
spotRosenbrockFunction (Testfunctions), 115
spotRosenbrockGradientFunction (Testfunctions), 115
spotSelectionAdjustedRsq (spotSelectionCriteria), 104
spotSelectionAic (spotSelectionCriteria), 104
spotSelectionCriteria, 45, 104
spotSelectionMae (spotSelectionCriteria), 104
spotSelectionMse (spotSelectionCriteria), 104
spotSelectionRsme (spotSelectionCriteria), 104
spotSelectionRsq (spotSelectionCriteria), 104
spotSelectionSae (spotSelectionCriteria), 104
spotSelectionScaledMse (spotSelectionCriteria), 104
spotSelectionScaledRsme (spotSelectionCriteria), 104
spotSelectionScaledSse (spotSelectionCriteria), 104
spotSelectionSse (spotSelectionCriteria), 104
spotSEXI2d, 63, 105
spotSixHumpFunction (Testfunctions), 115
spotSmsEmoa, 70, 106
spotSmsEmoaKriging, 107
spotSphere1Function (Testfunctions), 115
spotSphereFunction (Testfunctions), 115
spotStepAutoOpt, 17, 108
spotStepInitial, 17, 109, 109, 111
spotStepMetaOpt, 110
spotStepReport, 17, 100–103, 109, 110
spotStepRunAlg, 17, 47, 109, 111
spotStepSequential, 17, 109, 111, 112
spotSurf3d, 112, 114
spotSurfContour, 98, 99, 113, 113
spotWildFunction (Testfunctions), 115
spotWriteAroi, 114
spotWriteDes, 112

Testfunctions, 115