

# Package ‘SPOT’

March 8, 2012

**Maintainer** Martin Zaefferer <martin.zaefferer@smail.fh-koeln.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** 1.0.2257

**URL** <http://www.gm.fh-koeln.de/campus/personen/lehrende/thomas.bartz-beielstein/00489/>

**Date** 08.03.2012

**Depends** R (>= 2.14.0), rpart, maptree, emoa

**Suggests** rsm, earth, DiceKriging, kernlab, penalizedSVM, qrnn, mlegp, tgp, randomForest, fields, FrF2, DoE.wrapper, AlgDesign, lhs, cmaes, pso, rgenoud, DEoptim, subplex, minqa, rgl, rgp, twiddler, mco, MASS

**Collate**

'sms\_emoa.r' 'spot.R' 'spotAlgStartEs.R' 'spotAlgStartRgp.R' 'spotAlgStartSann.R' 'spotComparison.R' 'spotCreateDesign.R'

**Repository** CRAN

**Date/Publication** 2012-03-08 15:09:14

**R topics documented:**

|  |    |
|--|----|
| SPOT-package . . . . .                     | 3  |
| spot . . . . .                             | 4  |
| spotAlgEs . . . . .                        | 5  |
| spotAlgStartEs . . . . .                   | 7  |
| spotAlgStartEsVar . . . . .                | 7  |
| spotAlgStartRgp . . . . .                  | 8  |
| spotAlgStartSann . . . . .                 | 9  |
| spotAlgStartSannVar . . . . .              | 10 |
| spotCalcNoise . . . . .                    | 11 |
| spotCreate... . . . .                      | 11 |
| spotCreateDesignBasicDoe . . . . .         | 12 |
| spotCreateDesignDoeR3 . . . . .            | 13 |
| spotCreateDesignFrF2 . . . . .             | 13 |
| spotCreateDesignLhd . . . . .              | 14 |
| spotCreateDesignLhs . . . . .              | 15 |
| spotGenerateSequentialDesign . . . . .     | 15 |
| spotGenerateSequentialDesignOcba . . . . . | 16 |
| spotGetMergedDataMatrixB . . . . .         | 17 |
| spotGetOptions . . . . .                   | 18 |
| spotGetRawDataMatrixB . . . . .            | 21 |
| spotGetRawResData . . . . .                | 22 |
| spotGui . . . . .                          | 23 |
| spotInterface... . . . .                   | 23 |
| spotMcoSort . . . . .                      | 24 |
| spotNormDesign . . . . .                   | 24 |
| spotOcba . . . . .                         | 25 |
| spotOptim . . . . .                        | 25 |
| spotOptimInterface . . . . .               | 27 |
| spotParetoOptMulti . . . . .               | 28 |
| spotPlotBst . . . . .                      | 29 |
| spotPredict... . . . .                     | 29 |
| spotPredictDice . . . . .                  | 30 |
| spotPredictEarth . . . . .                 | 31 |
| spotPredictForrester . . . . .             | 31 |
| spotPredictGausspr . . . . .               | 32 |
| spotPredictKrig . . . . .                  | 33 |
| spotPredictKsvm . . . . .                  | 34 |
| spotPredictLm . . . . .                    | 34 |
| spotPredictLmOptim . . . . .               | 35 |
| spotPredictMlegp . . . . .                 | 36 |
| spotPredictOptMulti . . . . .              | 37 |
| spotPredictPOKER . . . . .                 | 38 |
| spotPredictPsvm . . . . .                  | 39 |
| spotPredictQrnn . . . . .                  | 39 |
| spotPredictRandomForest . . . . .          | 40 |
| spotPredictRandomForestMlegp . . . . .     | 41 |

|                                 |    |
|---------------------------------|----|
| spotPredictTgp . . . . .        | 42 |
| spotPredictTree . . . . .       | 42 |
| spotPrepareData . . . . .       | 43 |
| spotReadBstFile . . . . .       | 44 |
| spotReadRoi . . . . .           | 45 |
| spotReport3d . . . . .          | 45 |
| spotReportContour . . . . .     | 46 |
| spotReportDefault . . . . .     | 47 |
| spotReportMetaDefault . . . . . | 47 |
| spotReportSens . . . . .        | 48 |
| spotRgpTargetFunction . . . . . | 49 |
| spotROI . . . . .               | 49 |
| spotRsm . . . . .               | 50 |
| spotSmsEmoa . . . . .           | 51 |
| spotStepAutoOpt . . . . .       | 52 |
| spotStepInitial . . . . .       | 52 |
| spotStepMetaOpt . . . . .       | 53 |
| spotStepReport . . . . .        | 54 |
| spotStepRunAlg . . . . .        | 54 |
| spotStepSequential . . . . .    | 55 |
| spotWriteAroi . . . . .         | 56 |
| spotWriteBest . . . . .         | 56 |
| spotWriteDes . . . . .          | 57 |
| Testfunctions . . . . .         | 58 |

|              |           |
|--------------|-----------|
| <b>Index</b> | <b>60</b> |
|--------------|-----------|

---

|              |   |
|--------------|---|
| SPOT-package | <i>Sequential Parameter Optimization Toolbox in R</i> |
|--------------|---|

---

## Description

Sequential Parameter Optimization Toolbox in R

## Details

|           |            |
|-----------|------------|
| Package:  | SPOT       |
| Type:     | Package    |
| Version:  | 1.0.2257   |
| Date:     | 08.03.2012 |
| License:  | GPL (>= 3) |
| LazyLoad: | yes        |

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.

### 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](#)

---

|      |  |
|------|--|
| spot | <i>Main function for the use of SPOT</i> |
|------|--|

---

### 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

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

### Arguments

|            |   |
|------------|---|
| configFile | the absolute path including filespecifier, there is no default, this value should always be given   |
| spotTask   | [initlseq run autolrep] 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 <a href="#">spotGetOptions</a> |
| ...        | 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 = "selfA", sigmaInit = 1, nSigma = 1,
  tau0 = 0, tau = 1, rho = "bi", sel = -1, stratReco = 1,
  objReco = 2, maxGen = Inf, maxIter = 100, seed = 1,
  noise = 0, thrs = "no", thrsConstant = 0,
  fName = spotBraninFunction, lowerLimit = -1,
  upperLimit = 1, verbosity = 0, plotResult = FALSE,
  logPlotResult = FALSE, term = "iter",
  sigmaRestart = 0.1, preScanMult = 1,
  globalOpt = rep(0, dimension), conf = -1)
```

**Arguments**

|               |  |
|---------------|--|
| 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 "selfA"  |
| 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  |
| 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                                       |
| thrs          | threshold string, default is "no"  |
| thrsConstant  | number, default is 0.0   |
| fName         | function, fitness function, default is <a href="#">spotBraninFunction</a>                            |
| 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 <code>rep(0, dimension)</code> |
| conf          | config number passed to the result file, default is -1   |

**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](#)

---

spotAlgStartEsVar      *Interface for an Evolution Strategy to be robustly 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. 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 proposedly reach more robust results.

**Usage**

```
spotAlgStartEsVar(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**

```
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](#)

---

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 proposedly 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: <a href="#">spotGetOptions</a><br>Items used are:<br><br>alg.currentDesign: data frame holding the design points that will be evaluated<br>io.apdFileName: name of the apd file<br>io.desFileName: name of the des file<br>io.resFileName: name of the res file, for logging results (if spotConfig\$spot.fileMode==TRUE)<br>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](#) [spotAlgStartSann](#)

---

|               |                        |
|---------------|------------------------|
| spotCalcNoise | <i>Calculate Noise</i> |
|---------------|------------------------|

---

**Description**

Calculates Noise for a given Function-Value (y) dependant on the noise intensity (noise) and the way of calculating noise (noise.type)

**Usage**

```
spotCalcNoise(y, noise = 0, noise.type = "weighted",
              spot.noise.minimum.at.value = 0)
```

**Arguments**

|                             |  |
|-----------------------------|--|
| y                           | the function value where noise will be added to      |
| noise                       | noise magnitude (absolute or weighted)               |
| noise.type                  | type of noise can either be "weighted" or "constant" |
| spot.noise.minimum.at.value | Voice magnitude at minimum                           |

**Value**

numeric  
holding the noise Value

---

|               |  |
|---------------|--|
| spotCreate... | <i>General Help on Design Creation in SPOT..</i> |
|---------------|--|

---

**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 provided in a separated file where the function name **MUST** be the same as the filename (just without the file extension .R)
- 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"
```

this example assumes the existence of the files "myCreateDesign1.R" and "myCreateDesign2.R" in the directory "/home/theUser/mySpotExtensions/" holding a function "myCreateDesign1" or "myCreateDesign2" respectively

**Usage**

```
spotCreate...()
```

**See Also**

Design creating functions that are shipped with SPOT are: [spotCreateDesignDoeR3](#), [spotCreateDesignLhs](#), [spotCreateD](#) 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](#)

---

```
spotCreateDesignBasicDoe
```

*Create a full factorial design based on the number of dimensions and the number of design points*

---

**Description**

Uses the function `gen.factorial` from the `AlgDesign` package.

**Usage**

```
spotCreateDesignBasicDoe(spotConfig, noDesPoints = 100,
  repeats = 1)
```

**Arguments**

|                          |  |
|--------------------------|--|
| <code>spotConfig</code>  | list of <code>spotConfiguration</code> , where only the table <code>\$alg.roi</code> (.roi-file) is used,  |
| <code>noDesPoints</code> | optional parameter gives the number of design points. If <code>noDesPoints</code> is larger than the size of full factorial design, an error is reported.                        |
| <code>repeats</code>     | is obsolete for "doe" for <code>dow</code> gives a deterministic return value. The parameter is given to meet the standard interface to design creating functions (default is 1) |

**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

---

spotCreateDesignDoeR3 *Create a factorial design with resolution III*

---

**Description**

Uses the function FrF2 from the FrF2-package to generate a (fractional) factorial design

**Usage**

```
spotCreateDesignDoeR3(spotConfig, noDesPoints = 100,  
  repeats = 1)
```

**Arguments**

|             |  |
|-------------|--|
| spotConfig  | list of spotConfiguration, where only the table \$alg.roi (.roi-file) is used,   |
| noDesPoints | optional parameter gives the number of design points   |
| repeats     | is obsolete for "doe" for dow gives a deterministic reurn value. The parameter is given to meet the standard interface to design creating functions (default is 1) |

**Details**

The dimension is determined from the number of rows of the .roi - file (each row in the roi file defines a variable).

**Value**

matrix M  
- M has dimension columns and noDesPoints rows

---

spotCreateDesignFrF2 *Create a resolution III design with center point based on the number of dimensions and the number of design points*

---

**Description**

Uses the function FrF2 from the FrF2-package to generate a (fractional) factorial design

**Usage**

```
spotCreateDesignFrF2(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       |

**Details**

The dimension is determined from the number of rows of the .roi - file (each row in the roi file defines a variable).

**Value**

matrix M  
- M has dimension columns and noDesPoints rows

---

spotCreateDesignLhd     *Create a design based on the number of dimensions and the number of design points*

---

**Description**

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**

```
spotCreateDesignLhd(spotConfig, noDesPoints = NaN,
  retries = NaN)
```

**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 NaN |

**Value**

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

---

spotCreateDesignLhs     *Create a resolution III design with center point based on the number of dimensions and the number of design points*

---

### Description

Uses the function maxminLHS from the lhs to generate a Latin Hypercube Design. This function attempts to optimize the sample by maximizing the minimum distance between design points (maximin criteria).

### Usage

```
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       |

### Details

The dimension is determined from the number of rows of the .roi - file (each row in the roi file defines a variable).

### Value

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

---

spotGenerateSequentialDesign     *Generate Design for next sequential evaluation*

---

### Description

Creates a new design. Design points are determined with respect to the current result file.

### Usage

```
spotGenerateSequentialDesign(spotConfig)
```

**Arguments**

spotConfig      the list of all parameters is given, but the used ones are:  
 spotConfig\$io.resFileName: is checked for existence. If not found, function fails with error  
 spotConfig\$algSourceSrcPath: needed for the error message  
 spotConfig\$userConfFileName: needed for the error message  
 spotConfig\$io.verbosity: needed for command window output

**Details**

Uses the functions [spotPrepareData](#), [spotGetRawDataMatrixB](#), [spotGetMergedDataMatrixB](#), [spotWriteLines](#), [spotWriteBest](#), [spotPlotBst](#) returns a sequential design to be written to the file <xxx>.des (will be determined from calling function)

**Value**

data.frame design  
 - design contains one or more new design points to be calculated

---

spotGenerateSequentialDesignOcba

*Generate Design for next sequential evaluation with OCBA*

---

**Description**

Creates a new design. Design points are determined with respect to the current result file. Number of repeats are adapted according to the OCBA approach. Uses the functions [spotPrepareData](#), [spotGetRawDataMatrixB](#), [spotGetMergedDataMatrixB](#), [spotWriteLines](#), [spotWriteBest](#), [spotPlotBst](#), [spotOcba](#) returns a sequential design to be written to the file <xxx>.des (will be determined from calling function)

**Usage**

```
spotGenerateSequentialDesignOcba(spotConfig)
```

**Arguments**

spotConfig      the list of all parameters is given, but the used ones are:  
 spotConfig\$io.resFileName: is checked for existence. If not found, function fails with error  
 spotConfig\$algSourceSrcPath: needed for the error message  
 spotConfig\$userConfFileName: needed for the error message  
 spotConfig\$io.verbosity: needed for command window output

**Value**

data.frame design  
- design contains one or more new design points to be calculated

---

spotGetMergedDataMatrixB

*Get Merged Data Matrix B*

---

**Description**

The merged data that are the result of [spotPrepareData](#) must be the first input parameter. These data prepared and results in "Matrix B" that consists of the y-vector of the results and the x-Matrix of inputs that are bound together to a matrix B.

**Usage**

```
spotGetMergedDataMatrixB(mergedData, spotConfig)
```

**Arguments**

|            |   |
|------------|---|
| mergedData | the Data prepared as done with <a href="#">spotPrepareData</a>  |
| spotConfig | the list of all parameters is given, but used only as input parameter to the function call of <a href="#">spotPrepareData</a> |

**Value**

data.frame B  
- B holds a column "y" with the results and all columns with column-names derived from .roi file (should be the parameters of the algorithm). Values are sorted with respect to the y values (increasing)

**See Also**

[spotPrepareData](#)

---

|                |   |
|----------------|---|
| spotGetOptions | <i>Set all options by conf-file or by default</i> |
|----------------|---|

---

### 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 adaptations 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 filespecifier 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.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`).

|                                |   |
|--------------------------------|---|
| <code>alg.resultColumn</code>  | ["Y"] string to indicate the name of the result column. This might be automatically reset if <code>spotOptimInterface</code> is used, i.e. if a function is passed to <code>alg.func</code><br><br>It can be a vector of strings, if multi objective optimization is done.  |
| <code>alg.seed</code>          | [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.  |
| <code>alg.roi</code>           | 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)<br><br>The user can create an ROI matrix with the <code>spotROI</code> constructor. |
| <code>alg.ROI</code>           | 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 ".ROI"-file (= Actual Region Of Interest)   |
| <code>alg.currentDesign</code> | [usually not changed by user] data frame of the design that will be evaluated by the next call to <code>spotStepRunAlg</code>   |
| <code>alg.currentResult</code> | [usually not changed by user] data frame that contains the results of the target algorithm runs   |
| <code>alg.currentBest</code>   | [usually not changed by user] data frame that contains the best results of each step conducted by spot  |
| <code>io.columnSep</code>      | [" "] 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   |
| <code>io.apdFileName</code>    | [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)   |
| <code>io.roiFileName</code>    | [depends: <configFileName>.roi] name of the .roi -file (Region Of Interest - File, holding all varying parameters and constraints)  |
| <code>io.desFileName</code>    | [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)  |
| <code>io.resFileName</code>    | [depends: <configFileName>.res] name of the .res -file (RESult file) the user written algorithm has to write its results into this file   |
| <code>io.bstFileName</code>    | [depends: <configFileName>.bst] name of the .bst -file (BeST file) the result-file will be condensed to this file   |
| <code>io.pdfFileName</code>    | [depends: <configFileName>.pdf] name of the .pdf -file the default report will write its summary of results in this pdf file  |
| <code>io.fbsFileName</code>    | [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 programm, 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 desing 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.predictionModel.func ["spotPredictLm"] name of the function calling a predictor. Default uses a Linear Model. Please also see the notes SPOT - extensions

|                         |   |
|-------------------------|---|
| 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 <a href="#">spotPredictOptMulti</a> |
| 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](#)

---

spotGetRawDataMatrixB *Get Raw Data Matrix B*

---

**Description**

The result (.res) file is read and the data are prepared for further processing results in "Matrix B" that consists of the y-vector of the results and the x-Matrix of inputs that are bound together to a matrix B

**Usage**

```
spotGetRawDataMatrixB(spotConfig)
```

**Arguments**

|            |   |
|------------|---|
| spotConfig | the list of all parameters is given, also used as input parameter to the function call <code>spotGetRawResData(spotConfig)</code> the other ones in use are: <code>spotConfig\$alg.roi</code> : the roi data provided as matrix <code>spotConfig\$alg.resultColumn</code> : name of the result column |
|------------|---|

**Value**

data.frame B  
- B holds a column "y" with the results and all columns with column-names derived from .roi file (should be the parameters of the algorithm. Values are sorted with respect to the y values (increasing)

**See Also**

[spotPrepareData](#)

---

|                   |                            |
|-------------------|----------------------------|
| spotGetRawResData | <i>Get Raw Result Data</i> |
|-------------------|----------------------------|

---

**Description**

spotGetRawResData is based on [spotPrepareData](#)

**Usage**

```
spotGetRawResData(spotConfig)
```

**Arguments**

|            |   |
|------------|---|
| spotConfig | the list of all parameters is given, used ones are: spotConfig\$io.resFileName: result file where data are read from spotConfig\$io.columnSep: column separator as determined by getConfig parameter to the function call <a href="#">spotPrepareData</a> |
|------------|---|

**Details**

The result (.res) file is read and the data is returned as is

**Value**

data.frame rawResData  
- rawResData contains values from the result file

**See Also**

[spotPrepareData](#)

---

`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.

**See Also**[SPOT](#)

---

`spotInterface...`*General Help on SPOT...*

---

**Description**

SPOT is a system open and designed to plug in

- your own algorithm see [spotAlgStartAlg...](#)
- your own predictors see [spotPredict...](#)
- your own design creating functions [spotCreate...](#)
- your own increase functions to increase the repeats for each sequential step

**Usage**`spotInterface...()`

---

|             |   |
|-------------|---|
| spotMcoSort | <i>Sorting by NDS-rank and Hypervolume Contribution</i> |
|-------------|---|

---

**Description**

Sorts the large design for the purpose of multi objective optimization with SPOT. First non dominated sorting rank is used. If the choice of points for the next sequential step is not clear by nds rank, the hypervolume contribution of the competing points is recalculated sequentially to remove those with the smallest contribution.

**Usage**

```
spotMcoSort(largeDesign, designY, newsize)
```

**Arguments**

|             |  |
|-------------|--|
| largeDesign | the design matrix in the parameter space, to be sorted by the associated y-values for each objective |
| designY     | objective value matrix. Contains objective values associated to largeDesign                          |
| newsize     | this is the number of points that need to be selected, i.e. the seq.design.new.size                  |

**Value**

largeDesign  
- The sorted large design

---

|                |                         |
|----------------|-------------------------|
| spotNormDesign | <i>spotNormDesign()</i> |
|----------------|-------------------------|

---

**Description**

Produces a normalized design and calculates the minimal distance if required A design is a matrix with dim columns and size rows

**Usage**

```
spotNormDesign(dim, size, calcMinDistance = FALSE)
```

**Arguments**

|                 |  |
|-----------------|--|
| dim             | number, dimension of the problem (will be no. of columns of the result matrix)   |
| size            | number of points with that dimension needed. (will be no. of rows of the result matrix)  |
| calcMinDistance | Boolean to indicate whether a minimal distance should be calculated<br>this function is only the basis with a normalized design, that is used as a basis function to spotDesignLhd() |

**Value**

list L  
 - L consists of a matrix and nd (if required) a minimal distance

---

 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**

```
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**

[spotGenerateSequentialDesignOcba](#)

---

 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**

```
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 " <a href="#">spotFuncStartBranin</a> ")   |
| gr      | 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. <code>maxit</code> is the number of function evaluations, all the other settings will simply be passed to SPOT (see <a href="#">spotGetOptions</a> for details) |

**Details**

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:

```
control$spot.ocba <- FALSE
control$seq.design.maxRepeats <- 1
control$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="spotPredictRandomFores
```

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="spotPredictRandomFores
```

**Value**

This function returns a list with:

```
par parameters of the found solution
value target function value of the found solution
```

**See Also**

[SPOT spot spotOptimInterface](#)

---

spotOptimInterface      *Interface for Target Functions*

---

### 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 <code>spotConfig\$spot.fileMode=FALSE</code> . See also: <a href="#">spotGetOptions</a><br>Items used are:<br><br>alg.currentDesign: data frame holding the design points that will be evaluated<br>io.apdFileName: name of the apd file<br>io.desFileName: name of the des file<br>io.resFileName: name of the res file, for logging results (if <code>spotConfig\$spot.fileMode==TRUE</code> )<br>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<br>spotConfig\$alg.tar.func target function of type <code>y=f(x,...)</code> |
| ...        | additional parameters to be passed on to target function: <code>spotConfig\$alg.tar.func</code>  |

### Value

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

### See Also

[SPOT](#) [spot](#) [demo](#) [optim](#) [spotOptim](#)

---

spotParetoOptMulti      *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

```
spotParetoOptMulti(startPoint, spotConfig)
```

### Arguments

|            |   |
|------------|---|
| startPoint | initial information, not yet used   |
| spotConfig | list of all options, needed to provide data for calling functions.<br>This function uses the parameter <code>spotConfig\$seq.modelFit</code> . 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 <code>spotConfig\$seq.predictionOpt.method</code> will be used to choose the optimization method to be used to find the minimum of the fitted model:<br>"nsga2"<br>"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

[spotPredictOptMulti](#) solves the same task for single objective optimization (i.e. just one surrogate model)  
 See [spotSmsEmoa](#) for the used SMS-EMOA implementation

---

`spotPlotBst`*Plot Best Solution Found so far*

---

### Description

Function used to continuously plot the actually retrieved best value throughout a SPOT run. The number of variables shown is limited to twelve - make sure the relevant variables belong to the first lines of your .roi-file.

### Usage

```
spotPlotBst(spotConfig)
```

### Arguments

`spotConfig` the list of all parameters is given, used here `alg.roi`: the region of interest reduced to a matrix

### See Also

[SPOT](#) [spot](#) [spotReadBstFile](#)

---

`spotPredict...`*General Help on Prediction models in SPOT...*

---

### Description

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 a separated file where the function name **MUST** be the same as the filename (just without the file extension .R)
- 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:

```
seq.predictionModel.func="myPredictLm"
```

this example assumes the existence of a file "myPredictLm.R" in the directory "/home/theUser/mySpotExtensions/" holding a function "myPredictLm"

### Usage

```
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](#) #####

---

 spotPredictDice

*Meta Model Interface: Dice Kriging*


---

**Description**

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

**Usage**

```
spotPredictDice(rawB, mergedB, largeDesign, spotConfig,
  externalFit = NULL)
```

**Arguments**

|             |  |
|-------------|--|
| rawB        | matrix of raw x and y values   |
| mergedB     | matrix of merged x and y values, does not have replicate entries   |
| largeDesign | design points to be evaluated by the meta model  |
| spotConfig  | the list of all parameters is given.   |
| externalFit | 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 paramters 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 Krig model used with predict()  
 spotConfig\$seq.largeDesignY the y values of the large 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

```
spotPredictEarth(rawB, mergedB, largeDesign, spotConfig,
  externalFit = NULL)
```

### Arguments

|             |  |
|-------------|--|
| rawB        | unmerged data  |
| mergedB     | merged data  |
| largeDesign | new design points which should be predicted  |
| spotConfig  | global list of all options, needed to provide data for calling functions   |
| externalFit | 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 paramters 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 large design, evaluated with the fit

---

spotPredictForrester      *Meta Model Interface: Forresters Kriging*

---

### Description

Kriging predictor based on matlab code by Forrester et.al. Can be used both for single and multi objective SPOT.

### Usage

```
spotPredictForrester(rawB, mergedB, largeDesign,
  spotConfig, externalFit = NULL)
```

**Arguments**

|             |   |
|-------------|---|
| rawB        | unmerged data   |
| mergedB     | merged data   |
| largeDesign | 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:<br>spotConfig\$seq.forrester\$Option What to predict, can be: "Pred"(default), "NegLogExpImp", "NegProbImp", "RMSE"<br>spotConfig\$seq.forrester\$loval Lower boundary for theta, default is 1e-3<br>spotConfig\$seq.forrester\$upval Upper boundary for theta, default is 100<br>spotConfig\$seq.forrester\$algheta algorithm used to find theta, default is "cmaes"<br>spotConfig\$seq.forrester\$budgetalgheta Budget for the above mentioend algorithm, default is 1000 |
| externalFit | 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 paramters 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 large design, evaluated with the fit

**References**

Forrester, Alexander I.J.; Sobester, Andras; Keane, Andy J. (2008). Engineering Design via Surrogate Modelling - A Practical Guide. John Wiley & Sons.

**See Also**

[spotDetermineTheta](#) [spotNormalizeMatrix2](#) [spotNormalizeMatrix](#) [spotReverseNormalizeMatrix](#)  
[spotRegPredictor](#) [spotRegLikelihood](#)

---

spotPredictGausspr      *Meta Model Interface: Gaussian Processes*

---

**Description**

Gaussian processes predictor based on gausspr function in kernlab package.

**Usage**

```
spotPredictGausspr(rawB, mergedB, largeDesign,
  spotConfig, externalFit = NULL)
```

**Arguments**

|             |  |
|-------------|--|
| rawB        | unmerged data  |
| mergedB     | merged data  |
| largeDesign | new design points which should be predicted  |
| spotConfig  | global list of all options, needed to provide data for calling functions   |
| externalFit | 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 paramters 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 large design, evaluated with the fit

---

spotPredictKrig      *Meta Model Interface: Fields Kriging*

---

**Description**

Kriging meta model based on fields package.

**Usage**

```
spotPredictKrig(rawB, mergedB, largeDesign, spotConfig,
  externalFit = NULL)
```

**Arguments**

|             |  |
|-------------|--|
| rawB        | unmerged data  |
| mergedB     | merged data  |
| largeDesign | new design points which should be predicted  |
| spotConfig  | global list of all options, needed to provide data for calling functions   |
| externalFit | 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 paramters 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 Krig model used with predict()  
 spotConfig\$seq.largeDesignY the y values of the large design, evaluated with the fit

---

spotPredictKsvm      *Meta Model Interface: Support Vector Machine*

---

### Description

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

### Usage

```
spotPredictKsvm(rawB, mergedB, largeDesign, spotConfig,
  externalFit = NULL)
```

### Arguments

|             |  |
|-------------|--|
| rawB        | unmerged data  |
| mergedB     | merged data  |
| largeDesign | new design points which should be predicted  |
| spotConfig  | global list of all options, needed to provide data for calling functions   |
| externalFit | 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 paramters 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 large 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, lhd, spotConfig,
  externalFit = NULL)
```

**Arguments**

|             |  |
|-------------|--|
| rawB        | unmerged data  |
| mergedB     | merged data  |
| lhd         | new design points which should be predicted  |
| spotConfig  | global list of all options, needed to provide data for calling functions   |
| externalFit | 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 lhd data). To build the model, this parameter has to be NULL. If it is not NULL the paramters 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 Krig model used with predict()  
 spotConfig\$seq.largeDesignY the y values of the large design, evaluated with the fit

**See Also**

[SPOT](#)

---

spotPredictLmOptim     *Meta Model Interface: Spot Predictor Linear Model Optimized*

---

**Description**

A linear prediction model with optimization on the response surface and automated adaptation of the region of interest is used to predict improved design points

**Usage**

```
spotPredictLmOptim(rawB, mergedB, lhd, spotConfig,
  externalFit = NULL)
```

**Arguments**

|            |  |
|------------|--|
| rawB       | matrix of raw x and y values                                     |
| mergedB    | matrix of merged x and y values, does not have replicate entries |
| lhd        | design points to be evaluated by the meta model                  |
| spotConfig | the list of all parameters is given                              |

`externalFit` 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 lhd data). To build the model, this parameter has to be NULL. If it is not NULL the paramters `mergedB` and `rawB` will not be used at all in the function.

If `spotConfig` does not contain the list element `seq.useCanonicalPath` (boolean), it will be created with the default value FALSE. This setting tells if to start at saddle point in both directions (canonical path analysis). If `spotConfig` does not contain the list element `seq.useGradient` (boolean), it will be created with the default value FALSE. This setting tells if gradient information is used or not.

### 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 large design, evaluated with the fit

### See Also

[SPOT](#)

---

|                               |  |
|-------------------------------|--|
| <code>spotPredictMlegp</code> | <i>Meta Model Interface: Maximum Likelihood Estimation for Gaussian Processes, Kriging</i> |
|-------------------------------|--|

---

### 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, largeDesign, spotConfig,
  externalFit = NULL)
```

**Arguments**

|             |  |
|-------------|--|
| rawB        | matrix of raw x and y values   |
| mergedB     | matrix of merged x and y values, does not have replicate entries   |
| largeDesign | design points to be evaluated by the meta model  |
| spotConfig  | the list of all parameters is given.   |
| externalFit | 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 paramters 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 Krig model used with predict()  
 spotConfig\$seq.largeDesignY the y values of the large design, evaluated with the fit

**See Also**

[SPOT](#)

---

spotPredictOptMulti    *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**

```
spotPredictOptMulti(startPoint, spotConfig)
```

**Arguments**

|            |  |
|------------|--|
| startPoint | initial points for the optimization  |
| spotConfig | list of all options, needed to provide data for calling functions.<br>This function uses the parameter spotConfig\$seq.modelFit. This is supposed to be a fit, that can be evaluated by the predict() interface. 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:<br>"optim-BFGS"<br>"optim-L-BFGS-B"<br>"optim-CG"<br>"optim-NM"<br>"optim-SANN" |

```
"pso"
"cmaes"
"genoud"
"DEoptim"
"subplex"
"bobyqa"
"newuoa"
"uobyqa"
```

### 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

---

|                  |                         |
|------------------|-------------------------|
| spotPredictPOKER | <i>spotPredictPOKER</i> |
|------------------|-------------------------|

---

### Description

Price Of Knowledge and Estimated Reward - ensemble approach

### Usage

```
spotPredictPOKER(rawB, mergedB, largeDesign, spotConfig,
  externalFit = NULL)
```

### Arguments

|                          |   |
|--------------------------|---|
| <code>rawB</code>        | unmerged data   |
| <code>mergedB</code>     | merged data   |
| <code>largeDesign</code> | new design points which should be predicted   |
| <code>spotConfig</code>  | global list of all options, needed to provide data for calling functions  |
| <code>externalFit</code> | 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 <code>largeDesign</code> data). To build the model, this parameter has to be NULL. If it is not NULL the paramters <code>mergedB</code> and <code>rawB</code> will not be used at all in the function. |

### 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 large design, evaluated with the fit

---

spotPredictPsvm      *Meta Model Interface: Penalized Support Vector Machine*

---

### Description

This meta model for SPOT is based on the penalizedSVM package, using svm() to build a support vector machine model.

### Usage

```
spotPredictPsvm(rawB, mergedB, largeDesign, spotConfig,
  externalFit = NULL)
```

### Arguments

|             |  |
|-------------|--|
| rawB        | unmerged data  |
| mergedB     | merged data  |
| largeDesign | new design points which should be predicted  |
| spotConfig  | global list of all options, needed to provide data for calling functions   |
| externalFit | 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 paramters 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 large 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

```
spotPredictQrnn(rawB, mergedB, largeDesign, spotConfig,
  externalFit = NULL)
```

**Arguments**

|             |  |
|-------------|--|
| rawB        | unmerged data  |
| mergedB     | merged data  |
| largeDesign | new design points which should be predicted  |
| spotConfig  | global list of all options, needed to provide data for calling functions   |
| externalFit | 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 paramters 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 large design, evaluated with the fit

---

 spotPredictRandomForest

*Meta Model Interface: Random Forest*

---

**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, largeDesign,
  spotConfig, externalFit = NULL)
```

**Arguments**

|             |  |
|-------------|--|
| rawB        | unmerged data  |
| mergedB     | merged data  |
| largeDesign | new design points which should be predicted  |
| spotConfig  | global list of all options, needed to provide data for calling functions   |
| externalFit | 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 paramters 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 Krig model used with predict()  
 spotConfig\$seq.largeDesignY the y values of the large design, evaluated with the fit

**See Also**[SPOT](#)

---

`spotPredictRandomForestMlegp`*Meta Model Interface: Random Forest combined with Mlegp*

---

**Description**

A prediction model based on `rpart`, using a random forest and `mlegp`.

**Usage**

```
spotPredictRandomForestMlegp(rawB, mergedB, largeDesign,  
  spotConfig, externalFit = NULL)
```

**Arguments**

|                          |  |
|--------------------------|--|
| <code>rawB</code>        | unmerged data  |
| <code>mergedB</code>     | merged data  |
| <code>largeDesign</code> | new design points which should be predicted  |
| <code>spotConfig</code>  | global list of all options, needed to provide data for calling functions   |
| <code>externalFit</code> | 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 <code>largeDesign</code> data). To build the model, this parameter has to be <code>NULL</code> . If it is not <code>NULL</code> the paramters <code>mergedB</code> and <code>rawB</code> will not be used at all in the function. |

**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 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

```
spotPredictTgp(rawB, mergedB, lhd, spotConfig,
  externalFit = NULL)
```

### Arguments

|             |  |
|-------------|--|
| rawB        | unmerged data  |
| mergedB     | merged data  |
| lhd         | new design points which should be predicted  |
| spotConfig  | global list of all options, needed to provide data for calling functions   |
| externalFit | 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 lhd data). To build the model, this parameter has to be NULL. If it is not NULL the paramters 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 Krig model used with predict()  
 spotConfig\$seq.largeDesignY the y values of the large 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, lhd, spotConfig,
  externalFit = NULL)
```

**Arguments**

|             |  |
|-------------|--|
| rawB        | unmerged data  |
| mergedB     | merged data  |
| lhd         | new design points which should be predicted  |
| spotConfig  | global list of all options, needed to provide data for calling functions   |
| externalFit | 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 paramters 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 Krig model used with predict()  
 spotConfig\$seq.largeDesignY the y values of the large design, evaluated with the fit

**See Also**

[SPOT](#)

---

|                 |                              |
|-----------------|------------------------------|
| spotPrepareData | <i>Prepare Data for SPOT</i> |
|-----------------|------------------------------|

---

**Description**

spotPrepareData prepares the data from .res-file

**Usage**

```
spotPrepareData(spotConfig)
```

**Arguments**

|            |   |
|------------|---|
| spotConfig | the list of all parameters is given, used parameters are: spotConfig\$io.resFileName<br>spotConfig\$io.columnSep spotConfig\$alg.roi spotConfig\$alg.resultColumn<br>spotConfig\$seq.transformation.func() spotConfig\$seq.merge.func() |
|------------|---|

**Details**

The result (.res) file is read and the data are prepared for further processing results in an unsorted list of several values

**Value**

list resultList  
 - resultList is a list of values (not sorted in increasing order!) x: vector of vector (=matrix) of input values, the "points" mergedY: vector of merged fitness values, the result column (each row of x has a corresponding Y-value) count: the number of repeats CONFIG: the unique CONFIG number for each unique design point (sdev=NA if count=1) pNames: names of the parameters as derived from roi-file step.last: the maximum number of the step-column of the result-file STEP: SPOT STEP when design point was created

**See Also**

[SPOT spot](#)

---

|                 |                       |
|-----------------|-----------------------|
| spotReadBstFile | <i>Read .bst File</i> |
|-----------------|-----------------------|

---

**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)
```

**Arguments**

spotConfig      the list of all parameters is given, used here spotReadBstFile: the name pf the .bst file to be read spotConfig\$io.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 .roi file (should be the parameters of the algorithm)

**See Also**

[SPOT spot spotPlotBst](#)

---

|             |                      |
|-------------|----------------------|
| spotReadRoi | <i>Spot Read ROI</i> |
|-------------|----------------------|

---

**Description**

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

**Usage**

```
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  
- roi contains the data from the roi file

---

|              |  |
|--------------|--|
| spotReport3d | <i>3d Plot of Meta Model - Report Function</i> |
|--------------|--|

---

**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.<br>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. |
|------------|--|

## 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.

---

spotReportContour      *Model Contour Plot - Report Function*

---

## 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.

## 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.

---

spotReportDefault      *Default Report*

---

### Description

Function to generate 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 doesn't 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 spot 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 doesn't 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](#)

---

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](#)

---

spotRgpTargetFunction *Target function for RGP / SPOT tuning*

---

**Description**

This target function is used by [spotAlgStartRgp](#) to start rgp with different population or tournament sizes.

**Usage**

```
spotRgpTargetFunction(populationSize = 100,  
  tournamentSize = 10, time = 120)
```

**Arguments**

populationSize the populationSize parameter, as used by `symbolicRegression()` in `rgp`  
tournamentSize the tournamentSize parameter, as used by `makeTournamentSelection()` in `rgp`  
time the time for the stopping criterion of the rgp run, in seconds

**Value**

this function returns the RMSE (fitness) of the best individual in the population

**See Also**

[spotAlgStartRgp](#)

---

spotROI *Region Of Interest Constructor*

---

**Description**

This function can be used to construct a region of interest, to be passed on to `spot()`.

**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. |

**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)
```

spotRsm

*rsm Model***Description**

Help function creating a response-surface Regression this function will be replaced when rsm-package has fixed a bug

**Usage**

```
spotRsm(formula, data)
```

**Arguments**

|         |  |
|---------|--|
| formula | must be a formula                          |
| data    | parameter holding the data for the formula |

**Value**

model used by predict  
fix of rsm - Model

---

|             |  |
|-------------|--|
| spotSmsEmoa | <i>SMS-EMOA: S-Metric-Selection Evolutionary Multiobjective Optimization Algorithm</i> |
|-------------|--|

---

### 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

```
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 length( $y$ ) containing NA's 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

### See Also

N. Beume, B. Naujoks, and M.Emmerich. *SMS-EMOA: Multiobjective selection based on dominated hypervolume*. European Journal of Operational Research, 181(3):1653–1669, 2007.

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)

---

spotStepAutoOpt      *SPOT Step Auto Opt*

---

### Description

spotStepAutoOpt is the default task called, when spot is started.

### Usage

```
spotStepAutoOpt(spotConfig, ...)
```

### Arguments

|            |  |
|------------|--|
| spotConfig | the list of all parameters is given, it is forwarded to the call of the reportfunction the used parameters of spotConfig are just spotConfig\$auto.loop.steps specifying the number of meta models that should be calculated |
| ...        | additional parameters to be passed on to target function which is called inside alg.func   |

### Details

The auto task calls the tasks `init` and `run` once and loops `auto.loop.steps` times over the `steps seq` and `run` finalising the function with a call of the `report` function. Instead of `auto.loop.steps` also `auto.loop.nevals` can be used as a stopping criterion.

### 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. Therefor 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 minimal distance, (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 | <i>SPOT Step Meta</i> |
|-----------------|-----------------------|

---

**Description**

Attention: This feature is work in progress, documentation is not up to date.

**Usage**

```
spotStepMetaOpt(spotConfig)
```

**Arguments**

spotConfig      the list of all parameters is given

**Details**

The meta task calls spotStepMetaOpt which itself calls [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:

```
spot("configFileName.conf", "meta")
```

**See Also**

[spotGetOptions](#)

---

|                |                         |
|----------------|-------------------------|
| spotStepReport | <i>SPOT Step Report</i> |
|----------------|-------------------------|

---

### Description

Forth and last step for SPOT, that is by default a call of [spotReportDefault](#)

### Usage

```
spotStepReport(spotConfig)
```

### Arguments

spotConfig      the list of all parameters is given, it is forwarded to the call of the reportfunction

### 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 | <i>SPOT Step Algorithm Call</i> |
|----------------|---------------------------------|

---

### 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:<br>spotConfig\$alg.func the name of the R target function<br>spotConfig\$io.apdFileName filename for the problem definition of the algorithm, first parameter of the generically defined R-function spotConfig\$alg.func<br>spotConfig\$io.desFileName filename for the input of the algorithm, second parameter of the generically defined R-function spotConfig\$alg.func<br>spotConfig\$io.resFileName filename for the output of the algorithm third parameter of the generically defined R-function spotConfig\$alg.func<br>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. So this is the interface providing several possibilities to include an algorithm as (unix)-shell or as R-function (the latter could also provide a system call, so it should be able to integrate user written algorithms as flexible as possible. Result/Effects: external, user written function is called and expected to write the results into the file: spotConfig\$io.resFileName

**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 [spotGenerateSequentialDesign0cba](#)

**Usage**

```
spotStepSequential(spotConfig)
```

**Arguments**

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

**Details**

Creates a sequential design based on the results derived so far. Therefor it is essential to have another design evaluated before and have a .res file to use. It uses the functions [spotGenerateSequentialDesign](#) and [spotWriteDes](#) writes a sequential design to the file <xxx>.des

---

|               |                        |
|---------------|------------------------|
| spotWriteAroi | <i>Spot Write Aroi</i> |
|---------------|------------------------|

---

**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

---

|               |                        |
|---------------|------------------------|
| spotWriteBest | <i>Spot Write Best</i> |
|---------------|------------------------|

---

**Description**

Helper function that simply writes data to the .bst-file (appending or creating - depends on the existance of the .bst-file)

**Usage**

```
spotWriteBest(B, spotConfig)
```

**Arguments**

|            |  |
|------------|--|
| B          | matrix containing the merged result data of the current SPOT run   |
| spotConfig | all parameters, the ones of interest are:<br>the name of the file for the best-data to be stored in: spotConfig\$io.bestFileName<br>the name string of the result column: spotConfig\$alg.resultColumn<br>the boolean that specifies if files are actually used or not: spot.fileMode<br>the data frame that will be extended with the current best: alg.currentBest |

**Details**

Result/Effects: adds one row to the best file and the alg.currentBest data frame.

---

|              |                          |
|--------------|--------------------------|
| spotWriteDes | <i>Spot Write Design</i> |
|--------------|--------------------------|

---

**Description**

help function that simply writes Data to the .des-file

**Usage**

```
spotWriteDes(des, verbosity, sep, filename)
```

**Arguments**

|           |   |
|-----------|---|
| des       | design provided by any spotCreateDesignXXX()-function                     |
| verbosity | for values greater than two, a message is given                           |
| sep       | the column separator (should not be empty for writing table to .des-file) |
| filename  | the filename the design should be written to                              |

**Details**

Result/Effects: rewrites the design-file for the next call of the [spotStepRunAlg](#)

**See Also**

[spotStepRunAlg](#)

**Description**

The following target functions are available:

spotSphereFunction:

multi-dimensional sphere function, one global optimum:  $\text{Sum}[x^2]$

spotSphere1Function:

multi-dimensional sphere function, one global optimum:  $\text{Sum}[i^2*(x[[i]]-i)^2, i, 1, \text{ndim}]$

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>

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>

spotWildFunction:

Another test functions,  $y=10*\sin(0.3*x)*\sin(1.3*x^2) + 0.00001*x^4 + 0.2*x+80$

**Usage**

```
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 testfunction

**Value**

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

**See Also**

[SPOT spot demo](#)

# Index

- \*Topic **package**
  - SPOT-package, 3
- demo, 9, 10, 27, 59
- optim, 4, 9, 10, 25, 27
- SPOT, 5–10, 23, 26, 27, 29, 30, 35–37, 41–44, 47, 48, 52, 54, 55, 59
- SPOT (SPOT-package), 3
- spot, 4, 4, 9, 10, 25–27, 29, 44, 47, 48, 52–55, 59
- SPOT-package, 3
- spotAlgEs, 5, 7, 8
- spotAlgStartAlg..., 23
- spotAlgStartEs, 5, 6, 7, 7, 8
- spotAlgStartEsVar, 7, 7
- spotAlgStartRgp, 8, 49
- spotAlgStartSann, 9, 10
- spotAlgStartSannVar, 10, 10
- spotBraninFunction, 6
- spotBraninFunction (Testfunctions), 58
- spotCalcNoise, 11
- spotCreate..., 11, 23
- spotCreateDesignBasicDoe, 12, 12
- spotCreateDesignDoeR3, 12, 13
- spotCreateDesignFrF2, 12, 13
- spotCreateDesignLhd, 14
- spotCreateDesignLhs, 12, 15
- spotDetermineTheta, 32
- spotFuncStartBranin, 9, 10, 19, 26
- spotGenerateSequentialDesign, 15, 56
- spotGenerateSequentialDesignOcba, 16, 25, 55
- spotGetMergedDataMatrixB, 16, 17
- spotGetOptions, 4, 7–10, 12, 18, 26, 27, 30, 47, 48, 52–54
- spotGetRawDataMatrixB, 16, 21
- spotGetRawResData, 22
- spotGui, 4, 23
- spotInstAndLoadPackages, 11, 29
- spotInterface..., 23
- spotMcoSort, 24
- spotMexicanHatFunction (Testfunctions), 58
- spotNormalizeMatrix, 32
- spotNormalizeMatrix2, 32
- spotNormDesign, 24
- spotOcba, 16, 25
- spotOptim, 4, 5, 25, 27
- spotOptimInterface, 19, 26, 27
- spotParetoOptMulti, 28, 51
- spotPlotBst, 16, 29, 44
- spotPredict..., 23, 29
- spotPredictDice, 30
- spotPredictEarth, 31
- spotPredictForrester, 31
- spotPredictGausspr, 32
- spotPredictKrig, 33
- spotPredictKsvm, 34
- spotPredictLm, 30, 34
- spotPredictLmOptim, 35
- spotPredictMlegp, 30, 36
- spotPredictOptMulti, 21, 28, 37
- spotPredictPOKER, 38
- spotPredictPsvm, 39
- spotPredictQrnn, 39
- spotPredictRandomForest, 30, 40
- spotPredictRandomForestMlegp, 41
- spotPredictTgp, 30, 42
- spotPredictTree, 30, 42
- spotPrepare, 5
- spotPrepareData, 16, 17, 22, 43
- spotPrepareSystem, 5
- spotRastriginFunction (Testfunctions), 58
- spotReadBstFile, 29, 44
- spotReadRoi, 45
- spotRegLikelihood, 32

spotRegPredictor, 32  
spotReport3d, 45  
spotReportContour, 46  
spotReportDefault, 47, 54  
spotReportMetaDefault, 47  
spotReportSens, 48  
spotReverseNormalizeMatrix, 32  
spotRgpTargetFunction, 49  
spotROI, 19, 49  
spotRosenbrockFunction (Testfunctions),  
58  
spotRosenbrockGradientFunction  
(Testfunctions), 58  
spotRsm, 50  
spotSixHumpFunction (Testfunctions), 58  
spotSmsEma, 28, 51  
spotSphere1Function (Testfunctions), 58  
spotSphereFunction (Testfunctions), 58  
spotStepAutoOpt, 5, 52  
spotStepInitial, 5, 52, 52, 55  
spotStepMetaOpt, 53  
spotStepReport, 5, 47, 48, 52, 54  
spotStepRunAlg, 5, 19, 52, 54, 57  
spotStepSequential, 5, 52, 55, 55  
spotWildFunction (Testfunctions), 58  
spotWriteAroi, 56  
spotWriteBest, 16, 56  
spotWriteDes, 56, 57  
spotWriteLines, 16  
Testfunctions, 58