Package 'tsf'

March 13, 2024

Type Package

Title Fitting algebraic systems describing thermodynamic binding isotherms

Version 0.1

Date 2024-02-20

Author Krämer Konrad [aut, cre],

Maintainer Krämer Konrad <Konrad_kraemer@yahoo.de>

Description Optimizing of unknown paramaters in algebraic systems which describe thermodynamic bindings of dyes and guest to a host.
A script interface as well as a shiny app are included in the package.

License GPL-3

Imports shiny,
DT,
shinydashboard,
shinyWidgets,
shinyjs,
rootSolve,
ggplot2,
patchwork,
R6,
sensitivity,
openxlsx,
future,
promises

Suggests knitr, rmarkdown, tinytest

VignetteBuilder knitr

RoxygenNote 7.2.3

Encoding UTF-8

Roxygen list(markdown = TRUE)

R topics documented:

`Communicator` *Communicator class*

Description

a class for communicating via a temporary file

Public fields

`file` is a file which contains the current status

`result` is a file in which data can be written or read information.

Methods

Public methods:

- `Communicator$new()`
- `Communicator$getStatus()`
- `Communicator$setStatus()`
- `Communicator$setData()`
- `Communicator$getData()`
- `Communicator$interrupt()`
- `Communicator$ready()`
- `Communicator$running()`
- `Communicator$isInterrupted()`
- `Communicator$destroy()`
- `Communicator$clone()`

Method `new()`**:** create a new Communicator Object

Usage:

```
Communicator$new()
```

Method `getStatus()`**:** get the current status

Usage:

```
Communicator$getStatus()
```

Method `setStatus()`**:** write a status to the status file

Usage:

```
Communicator$setStatus(msg)
```

Arguments:

`msg` is the message which should be set in the file

Method `setData()`**:** write data to the result file

Usage:

```
Communicator$setData(data)
```

Arguments:

`data` is a string which should be written to the result file

Method `getData()`**:** get the current data from the result file

Usage:

```
Communicator$getData()
```

Method `interrupt()`**:** set the status to "interrupt"

Usage:

```
Communicator$interrupt()
```

Method `ready()`**:** set the status to "ready"

Usage:

```
Communicator$ready()
```

Method `running()`**:** write a status to the status file.

Usage:

```
Communicator$running(percComplete)
```

Arguments:

`percComplete` is the message which should be set in the file. If percComplete is not passed than the message is set to "Running..."

Method `isInterrupted()`**:** Checks if the current status is "interrupt"

Usage:

```
Communicator$isInterrupted()
```

Method `destroy()`**:** removes the temporary files. **This method has to be called at the end of the lifetime of the object!**

Usage:

```
Communicator$destroy()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Communicator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

convertToNum *Conversion of expression to numbers*

Description

converts an expression to a number

Usage

```
convertToNum(l)
```

Arguments

l	is an expression which should be evaluated. If the expression can be evaluated to a number the number is returned. Otherwise an error is returned.

createPolynom *Converts an algebraic system to a polynome by eliminating variables*

Description

Converts an algebraic system to a polynome by eliminating variables

Usage

```
createPolynom(f, elimVars)
```

Arguments

f	is a function defining the algebraic system
elimVars	is a character vector defining the variables which have to be eliminated

Value

the resulting polynome of the elimination

Examples

```
f <- function() {
  h + hd + -h0 = 0
  d + hd -d0 = 0
  hd / (h*d) -kd = 0
}
elimVars <- c("h", "d")
createPolynom(f, elimVars)
```

`ErrorClass` *ErrorClass class for handling errors*

Description

a class for handling error messages

Public fields

`message` the error message

`object` an R object which can be stored in the class instance if an error and a result should be returned together.

Methods

Public methods:

- `ErrorClass$new()`
- `ErrorClass$clone()`

Method `new()`**:** create a new ErrorClass Object

Usage:

```
ErrorClass$new(message, object = NULL)
```

Arguments:

`message` the string describing the error

`object` is optional if something besides the message should be stored

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ErrorClass$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`opti` *Optimize algebraic systems which describe thermodynamic binding systems*

Description

Optimize algebraic systems which describe thermodynamic binding systems

Usage

```
opti(
  case,
  lowerBounds,
  upperBounds,
  path,
  additionalParameters,
  npop = 40,
  ngen = 200,
  Topology = "random",
  errorThreshold = -Inf,
  runAsShiny = FALSE
)
```

Arguments

`case`	is a character describing which system should be investigated. Either: "hg", "ida" or "gda".
`lowerBounds`	is a numeric vector defining the lower boundaries of the parameter. In case of *hg* the order of the parameters is: *khd*, *I0*, *IHD* and *ID* In case of *ida* and *ga* the order of the parameters is: *kg*, *I0*, *IHD* and *ID*.
`upperBounds`	is a numeric vector defining the upper boundaries of the parameter. The order is the same as for the lower boundaries.
`path`	is a filepath which contains tabular x-y data. The concentraion of dye or guest respectivly is assumed to be in the first column. Furthermore, should the corresponding signal be stored in the second column. As an alternative an already loaded data.frame can be passed to the function.
`additionalParameters`	are required parameters which are specific for each case. In case of *hg* a numeric vector of length 1 is expected which contains the concentration of the host. In case of *ida* a numeric vector of length 3 is expected which contains the concentration of the host, dye and the *khd* parameter. In case of *gda* a numeric vector of length 3 is expected which contains the concentration of the host, guest and the *khd* parameter.
`npop`	is an optional integer argument defining the number of particles during optimization. The default value is set to 40.
`ngen`	is an optional integer argument defining the number of generations of the particle swarm optimization. The default value is set to 200.
`Topology`	is an optional character argument defining which topology should be used by the particle swarm algorithm. The options are "star" and "random". The default topology is the "random" topology.
`errorThreshold`	is an optional numeric argument defining a sufficient small error which acts as a stop signal for the particle swarm algorithm. The default value is set to -Inf.
`runAsShiny`	is internally used when running the algorithm from shiny.

Value

either an instance of ErrorClass if something went wrong. Otherwise the optimized parameter and the *insilico* signal values are returned.

Examples

```
path <- paste0(system.file("examples", package = "tsf"), "/IDA.txt")
opti("ida", c(1, 0, 0, 0), c(10^9, 10^6, 10^6, 10^6), path, c(4.3, 6.0, 7079458))
```

`pso` *Particle swarm optimization*

Description

Interface to the particle swarm optimization (pso) algorithm.

Usage

```
pso(
  env,
  lb,
  ub,
  loss,
  ngen,
  npop,
  error_threshold,
  global = FALSE,
  saveSwarm = FALSE,
  runAsShiny = FALSE
)
```

Arguments

`env`	is something that is passed to the loss function in addition to the parameters which get optimized
`lb`	is a numeric vector defining the lower boundaries of the parameter
`ub`	is a numeric vector defining the upper boundaries of the parameter
`loss`	is the loss function for which the optimal parameter set should be found
`ngen`	is the number of generations the pso should run
`npop`	is the number of particles which should be used during optimization
`error_threshold`	is a number defining a sufficient small error at which the optimization is stopped

global	is a logical parameter. If set to TRUE a global star topology is used. Thus, each particle compares itself with the global best particle of the entire swarm. In contrast, if global is set to FALSE a random arbitrary neighborhood is used instead. Thus, each particle has a neighberhood which contains K neighbours where K is between 0 and 3. From the swarm K neighbours are drawn randomly. From the neighberhood the best particle is used for comparison. The neighberhood is calculated for each generation.
saveSwarm	is a logical value defining whether the entire optimization should be saved.
runAsShiny	is an internal parameter which is used when running the shiny app interface.

Examples

```
rosenbrock <- function(parameter, env, Ignore) {
 value <- 0
 for (i in 1:(length(parameter) - 1)) {
   value <- value +
     100*(parameter[i + 1] - parameter[i]^2)^2 +
     (1 - parameter[i])^2
 }
return(value)
}

set.seed(1234)
res <- tsf::pso(new.env(), rep(-10, 3), rep(10, 3), rosenbrock, 1000, 40,
                0.00001, TRUE, FALSE)
```

---

runApp *Offers a GUI for the optimization of algebraic systems describing thermodynamic binding systems*

Description

Offers a GUI for the optimization of algebraic systems describing thermodynamic binding systems

Usage

```
runApp(port)
```

Arguments

port	is a number defining the port to use.

Examples

```
tsf::runApp()
```

sensitivity *Optimize algebraic systems which describe thermodynamic binding systems*

Description

Optimize algebraic systems which describe thermodynamic binding systems

Usage

```
sensitivity(
  case,
  parameters,
  path,
  additionalParameters,
  percentage = NULL,
  OffsetBoundaries = NULL,
  runAsShiny = FALSE
)
```

Arguments

case	is a character describing which system should be investigated. Either: "hg", "ida" or "gda".
parameters	is a numeric vector containing already optimized parameter. In case of *hg* the order of the parameters is: *khd*, *I0*, *IHD* and *ID* In case of *ida* and *ga* the order of the parameters is: *kg*, *I0*, *IHD* and *ID*.
path	is a filepath which contains tabular x-y data. The concentraion of dye or guest respectivly is assumed to be in the first column. Furthermore, should the corresponding signal be stored in the second column. As an alternative an already loaded data.frame can be passed to the function.
additionalParameters	are required parameters which are specific for each case. In case of *hg* a numeric vector of length 1 is expected which contains the concentration of the host. In case of *ida* a numeric vector of length 3 is expected which contains the concentration of the host, dye and the *khd* parameter. In case of *gda* a numeric vector of length 3 is expected which contains the concentration of the host, guest and the *khd* parameter.
percentage	is the percentage +/- from parameters in which the sensitivity should be analysed.
OffsetBoundaries	in case percentage is not suitable a numeric vector (equivalent to parameters) can be used which is added/substracted from parameters. It is only possible to set either percentage or OffsetBoundaries.
runAsShiny	is internally used when running the algorithm from shiny.

Value

either an instance of ErrorClass if something went wrong. Otherwise plots showing the sensitivity are returned.

Examples

```
path <- paste0(system.file("examples", package = "tsf"), "/IDA.txt")
res <- opti("ida", c(1, 0, 0, 0), c(10^9, 10^6, 10^6, 10^6), path, c(4.3, 6.0, 7079458))
sensitivity("ida", res[[2]], path, c(4.3, 6.0, 7079458), 20)
```

Index