Package: ODEsensitivity (via r-universe)

August 23, 2024

Version 1.1.2

Title Sensitivity Analysis of Ordinary Differential Equations

Description Performs sensitivity analysis in ordinary differential equation (ode) models. The package utilize the ode interface from 'deSolve' and connects it with the sensitivity analysis from 'sensitivity'. Additionally we add a method to run the sensitivity analysis on variables with class 'ODEnetwork'. A detailed plotting function provides outputs on the calculations. The method is described by Weber, Theers, Surmann, Ligges, and Weihs (2018) <doi:10.17877/DE290R-18874>.

URL `https://github.com/surmann/ODEsensitivity`

BugReports `https://github.com/surmann/ODEsensitivity/issues`

License LGPL-3

Encoding UTF-8

Depends R (>= 3.1.1), checkmate, deSolve, ODEnetwork (>= 1.3.0), sensitivity (>= 1.12.1)

Suggests covr, knitr, parallel, rmarkdown, testthat

RoxygenNote 6.0.1

VignetteBuilder knitr

Language en-US

Repository https://surmann.r-universe.dev

RemoteUrl https://github.com/surmann/odesensitivity

RemoteRef HEAD

RemoteSha d005b31a55d3e438aa5dc848bcba5edb0e4de929

Contents

`ODEmorris` *Morris Screening for ODE Models*

Description

`ODEmorris` is the generic function for performing a sensitivity analysis of ODE models using Morris's elementary effects screening method.

Usage

```
ODEmorris(mod, ...)
```

Arguments

`mod`	either a model function supplied in the manner as needed for `ode` (for `ODEmorris.default`) or an object of class `ODEnetwork` (for `ODEmorris.ODEnetwork`).
`...`	further arguments passed to methods, see `ODEmorris.default` and `ODEmorris.ODEnetwork`.

Details

There are two methods for this generic function: `ODEmorris.default` (for general ODE models) and `ODEmorris.ODEnetwork` (for objects of class `ODEnetwork`, see package `ODEnetwork`).

Author(s)

Frank Weber

See Also

`ODEmorris.default`, `ODEmorris.ODEnetwork`

ODEmorris.default *Morris Screening for General ODE Models*

Description

ODEmorris.default is the default method of ODEmorris. It performs a sensitivity analysis for general ODE models using the Morris screening method.

Usage

```
## Default S3 method:
ODEmorris(mod, pars, state_init, times, binf = 0,
  bsup = 1, r = 500, design = list(type = "oat", levels = 10, grid.jump =
  1), scale = TRUE, ode_method = "lsoda", parallel_eval = FALSE,
  parallel_eval_ncores = NA, ...)
```

Arguments

mod	[function(Time, State, Pars)] model to examine, supplied in the manner as needed for ode (see example below).
pars	[character(k)] names of the parameters to be included as input variables in Morris screening.
state_init	[numeric(z)] vector of z initial values. Must be named (with unique names).
times	[numeric] points of time at which the sensitivity analysis should be executed (vector of arbitrary length). The first point of time must be greater than zero.
binf	[character(1 or k)] vector of lower borders of possible input parameter values. If they are all equal, a single value can be set.
bsup	[character(1 or k)] vector of upper borders of possible input parameter values. If they are all equal, a single value can be set.
r	[integer(1 or 2)] if of length 1, the number of repetitions of the design. If of length 2, a space-filling optimization of the sampling design is used, see morris. However, this space-filling optimization might lead to long runtimes, so length 1 is recommended for r. Defaults to 500.
design	[list] a list specifying the design type and its parameters, cf. morris.
scale	[logical(1)] if TRUE, scaling is done for the input design of experiments after building the design and before calculating the elementary effects, cf. morris. Defaults to TRUE, which is highly recommended if the factors have different orders of magnitude, see morris.

`ode_method`	`[character(1)]` method to be used for solving the differential equations, see `ode`. Defaults to `"lsoda"`.
`parallel_eval`	`[logical(1)]` logical indicating if the evaluation of the ODE model shall be performed parallelized.
`parallel_eval_ncores`	`[integer(1)]` number of processor cores to be used for parallelization. Only applies if `parallel_eval = TRUE`. If set to `NA` (as per default) and `parallel_eval = TRUE`, 1 processor core is used.
`...`	further arguments passed to or from other methods.

Details

Function `ode` from `deSolve` is used to solve the ODE system.

The sensitivity analysis is done for all state variables and all timepoints simultaneously using `morris` from the package `sensitivity`.

For non-ODE models, values for `r` are typically between 10 and 50. However, much higher values are recommended for ODE models (the default is `r = 500`).

Value

List of class `ODEmorris` of length `length(state_init)` containing in each element a matrix for one state variable. The matrices themselves contain the Morris screening results for all timepoints (rows: `mu`, `mu.star` and `sigma` for every parameter; columns: timepoints).

Note

If the evaluation of the model function takes too long, it might be helpful to try another ODE-solver (argument `ode_method`). The `ode_methods` `"vode"`, `"bdf"`, `"bdf_d"`, `"adams"`, `"impAdams"` and `"impAdams_d"` might be faster than the default `"lsoda"`.

If `morris` throws a warning message stating "In ... keeping ... repetitions out of ...", try using a bigger number of `levels` in the `design` argument (only possible for OAT design).

Author(s)

Stefan Theers, Frank Weber

References

J. O. Ramsay, G. Hooker, D. Campbell and J. Cao, 2007, *Parameter estimation for differential equations: a generalized smoothing approach*, Journal of the Royal Statistical Society, Series B, 69, Part 5, 741–796.

See Also

`morris`, `plot.ODEmorris`

Examples

```
##### Lotka-Volterra equations #####
# The model function:
LVmod <- function(Time, State, Pars) {
  with(as.list(c(State, Pars)), {
    Ingestion    <- rIng  * Prey * Predator
    GrowthPrey   <- rGrow * Prey * (1 - Prey/K)
    MortPredator <- rMort * Predator

    dPrey        <- GrowthPrey - Ingestion
    dPredator    <- Ingestion * assEff - MortPredator

    return(list(c(dPrey, dPredator)))
  })
}
# The parameters to be included in the sensitivity analysis and their lower
# and upper boundaries:
LVpars  <- c("rIng", "rGrow", "rMort", "assEff", "K")
LVbinf <- c(0.05, 0.05, 0.05, 0.05, 1)
LVbsup <- c(1.00, 3.00, 0.95, 0.95, 20)
# The initial values of the state variables:
LVinit  <- c(Prey = 1, Predator = 2)
# The timepoints of interest:
LVtimes <- c(0.01, seq(1, 50, by = 1))
# Morris screening:
set.seed(7292)
# Warning: The following code might take very long!

LVres_morris <- ODEmorris(mod = LVmod,
                          pars = LVpars,
                          state_init = LVinit,
                          times = LVtimes,
                          binf = LVbinf,
                          bsup = LVbsup,
                          r = 500,
                          design = list(type = "oat",
                                        levels = 10, grid.jump = 1),
                          scale = TRUE,
                          ode_method = "lsoda",
                          parallel_eval = TRUE,
                          parallel_eval_ncores = 2)


##### FitzHugh-Nagumo equations (Ramsay et al., 2007) #####
FHNmod <- function(Time, State, Pars) {
  with(as.list(c(State, Pars)), {

    dVoltage <- s * (Voltage - Voltage^3 / 3 + Current)
    dCurrent <- - 1 / s *(Voltage - a + b * Current)

    return(list(c(dVoltage, dCurrent)))
  })
}
# Warning: The following code might take very long!

FHNres_morris <- ODEmorris(mod = FHNmod,
                           pars = c("a", "b", "s"),
                           state_init = c(Voltage = -1, Current = 1),
                           times = seq(0.1, 50, by = 5),
                           binf = c(0.18, 0.18, 2.8),
                           bsup = c(0.22, 0.22, 3.2),
                           r = 500,
                           design = list(type = "oat",
                                         levels = 50, grid.jump = 1),
                           scale = TRUE,
                           ode_method = "adams",
                           parallel_eval = TRUE,
                           parallel_eval_ncores = 2)
```

ODEmorris.ODEnetwork *Morris Screening for Objects of Class* ODEnetwork

Description

ODEmorris.ODEnetwork performs a sensitivity analysis for objects of class ODEnetwork using the Morris screening method. Package ODEnetwork is required for this function to work.

Usage

```
## S3 method for class 'ODEnetwork'
ODEmorris(mod, pars, times, binf = 0, bsup = 1,
  r = 500, design = list(type = "oat", levels = 10, grid.jump = 1),
  scale = TRUE, ode_method = "lsoda", parallel_eval = FALSE,
  parallel_eval_ncores = NA, ...)
```

Arguments

mod	[ODEnetwork] list of class ODEnetwork.
pars	[character(k)] names of the parameters to be included as input variables in Morris screening. All parameter names must be contained in names(ODEnetwork::createParamVec(mod)) and must not be derivable from other parameters supplied (e.g. "k.2.1" can be derived from "k.1.2", so supplying "k.1.2" suffices).
times	[numeric] points of time at which the sensitivity analysis should be executed (vector of arbitrary length). The first point of time must be greater than zero.

`binf`	`[character(1` or `k)]` vector of lower borders of possible values for the `k` input parameters. If they are all equal, a single value can be set.
`bsup`	`[character(1` or `k)]` vector of upper borders of possible values for the `k` input parameters. If they are all equal, a single value can be set.
`r`	`[integer(1)]` if of length 1, the number of repetitions of the `design`. If of length 2, a space-filling optimization of the sampling design is used, see `morris`. However, this space-filling optimization might lead to long runtimes, so length 1 is recommended for `r`. Defaults to 500.
`design`	`[list]` a list specifying the design type and its parameters, cf. `morris`.
`scale`	`[logical(1)]` if `TRUE`, scaling is done for the input design of experiments after building the design and before calculating the elementary effects, cf. `morris`. Defaults to `TRUE`, which is highly recommended if the factors have different orders of magnitude, see `morris`.
`ode_method`	`[character(1)]` method to be used for solving the ODEs in situations where the solution has to be determined numerically, see `ode` for details. Defaults to `"lsoda"`.
`parallel_eval`	`[logical(1)]` logical indicating if the evaluation of the ODE model shall be performed parallelized.
`parallel_eval_ncores`	`[integer(1)]` number of processor cores to be used for parallelization. Only applies if `parallel_eval = TRUE`. If set to `NA` (as per default) and `parallel_eval = TRUE`, 1 processor core is used.
`...`	further arguments passed to or from other methods.

Details

If the object of class `ODEnetwork` supplied for `mod` doesn't include any events, the solution of the ODE network is determined analytically using `simuNetwork`. In the presence of events, `simuNetwork` uses `ode` to solve the ODE network numerically.

The sensitivity analysis is done for all state variables and all timepoints simultaneously using `morris` from the package `sensitivity`.

For non-ODE models, values for `r` are typically between 10 and 50. However, much higher values are recommended for ODE models (the default is `r = 500`).

Value

List of class `ODEmorris` of length `2 * nrow(mod$state)` containing in each element a matrix for one state variable (all components of the 2 state variables are analyzed independently). The matrices themselves contain the Morris screening results for all timepoints (rows: `mu`, `mu.star` and `sigma` for every parameter; columns: timepoints).

Note

In situations where the solution of the ODE model has to be determined numerically, it might be helpful to try another ODE-solver if the evaluation of the model function takes too long, (argument `ode_method`). The `ode_methods` `"vode"`, `"bdf"`, `"bdf_d"`, `"adams"`, `"impAdams"` and `"impAdams_d"` might be faster than the default `"lsoda"`.

If `morris` throws a warning message stating "In ... keeping ... repetitions out of ...", try using a bigger number of `levels` in the `design` argument (only possible for OAT design).

Author(s)

Frank Weber

See Also

`morris`, `plot.ODEmorris`

Examples

```
##### A network of 4 mechanical oscillators connected in a circle #####
# Definition of the network using the package "ODEnetwork":
M_mat <- rep(2, 4)
K_mat <- diag(rep(2 * (2*pi*0.17)^2, 4))
K_mat[1, 2] <- K_mat[2, 3] <-
  K_mat[3, 4] <- K_mat[1, 4] <- 2 * (2*pi*0.17)^2 / 10
D_mat <- diag(rep(0.05, 4))
library("ODEnetwork")
lfonet <- ODEnetwork(masses = M_mat, dampers = D_mat, springs = K_mat)
# The parameters to be included in the sensitivity analysis and their lower
# and upper boundaries:
LFOpars <- c("k.1", "k.2", "k.3", "k.4",
             "d.1", "d.2", "d.3", "d.4")
LFObinf <- c(rep(0.2, 4), rep(0.01, 4))
LFObsup <- c(rep(20, 4), rep(0.1, 4))
# Setting of the initial values of the state variables:
lfonet <- setState(lfonet, state1 = rep(2, 4), state2 = rep(0, 4))
# The timepoints of interest:
LFOtimes <- seq(25, 150, by = 2.5)
# Morris screening:
set.seed(283)
# Warning: The following code might take very long!

LFOres_morris <- ODEmorris(mod = lfonet,
                           pars = LFOpars,
                           times = LFOtimes,
                           binf = LFObinf,
                           bsup = LFObsup,
                           r = 500,
                           design = list(type = "oat",
                                         levels = 10, grid.jump = 1),
                           scale = TRUE,
                           parallel_eval = TRUE,
                         parallel_eval_ncores = 2)
```

`ODEsensitivity` *Performing Sensitivity Analysis in ODE Models*

Description

`ODEsensitivity` provides methods to perform sensitivity analysis (SA) in ordinary differential equation (ODE) models. Its functions are based on the implementations of Morris and Sobol' SA in the `sensitivity` package (Pujol et al., 2015). However, a modified version of the `sensitivity`-package is required that enables `morris`, `soboljansen` and `sobolmartinez` to handle three-dimensional arrays as model outputs. Each element of the third dimension of the output array is then used to contain the results for one state variable of the ODE model. Each element of the second dimension of the output array is used for one timepoint.

Details

The main functions are `ODEmorris` and `ODEsobol`, which are generic functions. They have default methods for general ODE models (`ODEmorris.default`, `ODEsobol.default`) as well as methods for objects of class `ODEnetwork` (`ODEmorris.ODEnetwork`,`ODEsobol.ODEnetwork`). For the latter two methods, the package `ODEnetwork` is required.

See the `sensitivity` package and its `morris`, `soboljansen` and `sobolmartinez` implementations for further information on sensitivity analysis in `R`.

`ODEsobol` *Sobol' Sensitivity Analysis for ODE Models*

Description

`ODEsobol` is the generic function for performing a Sobol' sensitivity analysis of ODE models.

Usage

```
ODEsobol(mod, ...)
```

Arguments

`mod`	either a model function supplied in the manner as needed for `ode` (for `ODEsobol.default`) or an object of class `ODEnetwork` (for `ODEsobol.ODEnetwork`).
`...`	further arguments passed to methods, see `ODEsobol.default` and `ODEsobol.ODEnetwork`.

Details

There are two methods for this generic function: `ODEsobol.default` (for general ODE models) and `ODEsobol.ODEnetwork` (for objects of class `ODEnetwork`, see package `ODEnetwork`).

Author(s)

Frank Weber

See Also

`ODEsobol.default`, `ODEsobol.ODEnetwork`

`ODEsobol.default` *Sobol' Sensitivity Analysis for General ODE Models*

Description

`ODEsobol.default` is the default method of `ODEsobol`. It performs the variance-based Sobol' sensitivity analysis for general ODE models.

Usage

```
## Default S3 method:
ODEsobol(mod, pars, state_init, times, n = 1000,
  rfuncs = "runif", rargs = "min = 0, max = 1", sobol_method = "Martinez",
  ode_method = "lsoda", parallel_eval = FALSE, parallel_eval_ncores = NA,
  ...)
```

Arguments

`mod`	`[function(Time, State, Pars)]` model to examine, supplied in the manner as needed for `ode` (see example below).
`pars`	`[character(k)]` names of the parameters to be included as input variables in the Sobol' sensitivity analysis.
`state_init`	`[numeric(z)]` vector of `z` initial values. Must be named (with unique names).
`times`	`[numeric]` points of time at which the sensitivity analysis should be executed (vector of arbitrary length). The first point of time must be greater than zero.
`n`	`[integer(1)]` number of random parameter values used to estimate the Sobol' sensitivity indices by Monte Carlo simulation. Defaults to 1000.

`rfuncs`	`[character(1 or k)]` names of the functions used to generate the `n` random values for the `k` parameters. Can be of length 1 or `k`. If of length 1, the same function is used for all parameters. Defaults to `"runif"`, so a uniform distribution is assumed for all parameters.
`rargs`	`[character(1 or k)]` arguments to be passed to the functions in `rfuncs`. Can be of length 1 or `k`. If of length 1, the same arguments are used for all parameters. Each element of `rargs` has to be a string of the form `"tag1 = value1, tag2 = value2, ..."`, see example below. Default is `"min = 0, max = 1"`, so (together with the default value of `rfuncs`) a uniform distribution on [0, 1] is assumed for all parameters.
`sobol_method`	`[character(1)]` either `"Jansen"` or `"Martinez"`, specifying which modification of the variance-based Sobol' method shall be used. Defaults to `"Martinez"`.
`ode_method`	`[character(1)]` method to be used for solving the differential equations, see `ode`. Defaults to `"lsoda"`.
`parallel_eval`	`[logical(1)]` logical indicating if the evaluation of the ODE model shall be performed parallelized.
`parallel_eval_ncores`	`[integer(1)]` number of processor cores to be used for parallelization. Only applies if `parallel_eval = TRUE`. If set to `NA` (as per default) and `parallel_eval = TRUE`, 1 processor core is used.
`...`	further arguments passed to or from other methods.

Details

Function `ode` from `deSolve` is used to solve the ODE system.

The sensitivity analysis is done for all state variables and all timepoints simultaneously. If `sobol_method = "Jansen"`, `soboljansen` from the package `sensitivity` is used to estimate the Sobol' sensitivity indices and if `sobol_method = "Martinez"`, `sobolmartinez` is used (also from the package `sensitivity`).

Value

List of length `length(state_init)` and of class `ODEsobol` containing in each element a list of the Sobol' sensitivity analysis results for the corresponding `state_init`-variable (i.e. first order sensitivity indices `S` and total sensitivity indices `T`) for every point of time in the `times` vector. This list has an extra attribute `"sobol_method"` where the value of argument `sobol_method` is stored (either `"Jansen"` or `"Martinez"`).

Note

If the evaluation of the model function takes too long, it might be helpful to try a different type of ODE-solver (argument `ode_method`). The `ode_methods` `"vode"`, `"bdf"`, `"bdf_d"`, `"adams"`, `"impAdams"` and `"impAdams_d"` might be faster than the standard `ode_method` `"lsoda"`.

If n is too low, the Monte Carlo estimation of the sensitivity indices might be very bad and even produce first order indices < 0 or total indices > 1. First order indices in the interval [-0.05, 0) and total indices in (1, 1.05] are considered as minor deviations and set to 0 resp. 1 without a warning. First order indices < -0.05 or total indices > 1.05 are considered as major deviations. They remain unchanged and a warning is thrown. Up to now, first order indices > 1 or total indices < 0 haven't occured yet. If this should be the case, please contact the package author.

Author(s)

Stefan Theers, Frank Weber

References

J. O. Ramsay, G. Hooker, D. Campbell and J. Cao, 2007, *Parameter estimation for differential equations: a generalized smoothing approach*, Journal of the Royal Statistical Society, Series B, 69, Part 5, 741–796.

See Also

soboljansen, sobolmartinez, plot.ODEsobol

Examples

```
##### Lotka-Volterra equations #####
# The model function:
LVmod <- function(Time, State, Pars) {
  with(as.list(c(State, Pars)), {
    Ingestion    <- rIng  * Prey * Predator
    GrowthPrey   <- rGrow * Prey * (1 - Prey/K)
    MortPredator <- rMort * Predator

    dPrey        <- GrowthPrey - Ingestion
    dPredator    <- Ingestion * assEff - MortPredator

    return(list(c(dPrey, dPredator)))
  })
}
# The parameters to be included in the sensitivity analysis and their lower
# and upper boundaries:
LVpars  <- c("rIng", "rGrow", "rMort", "assEff", "K")
LVbinf <- c(0.05, 0.05, 0.05, 0.05, 1)
LVbsup <- c(1.00, 3.00, 0.95, 0.95, 20)
# The initial values of the state variables:
LVinit  <- c(Prey = 1, Predator = 2)
# The timepoints of interest:
LVtimes <- c(0.01, seq(1, 50, by = 1))
set.seed(59281)
# Sobol' sensitivity analysis (here only with n = 500, but n = 1000 is
# recommended):
# Warning: The following code might take very long!

LVres_sobol <- ODEsobol(mod = LVmod,
                        pars = LVpars,
                        state_init = LVinit,
                        times = LVtimes,
                        n = 500,
                        rfuncs = "runif",
                        rargs = paste0("min = ", LVbinf,
                                       ", max = ", LVbsup),
                        sobol_method = "Martinez",
                        ode_method = "lsoda",
                        parallel_eval = TRUE,
                        parallel_eval_ncores = 2)


##### FitzHugh-Nagumo equations (Ramsay et al., 2007) #####
FHNmod <- function(Time, State, Pars) {
  with(as.list(c(State, Pars)), {

    dVoltage <- s * (Voltage - Voltage^3 / 3 + Current)
    dCurrent <- - 1 / s *(Voltage - a + b * Current)

    return(list(c(dVoltage, dCurrent)))
  })
}
# Warning: The following code might take very long!

FHNres_sobol <- ODEsobol(mod = FHNmod,
                         pars = c("a", "b", "s"),
                         state_init = c(Voltage = -1, Current = 1),
                         times = seq(0.1, 50, by = 5),
                         n = 500,
                         rfuncs = "runif",
                         rargs = c(rep("min = 0.18, max = 0.22", 2),
                                   "min = 2.8, max = 3.2"),
                         sobol_method = "Martinez",
                         ode_method = "adams",
                         parallel_eval = TRUE,
                         parallel_eval_ncores = 2)

# Just for demonstration purposes: The use of different distributions for the
# parameters (here, the distributions and their arguments are chosen
# completely arbitrarily):
# Warning: The following code might take very long!

demo_dists <- ODEsobol(mod = FHNmod,
                       pars = c("a", "b", "s"),
                       state_init = c(Voltage = -1, Current = 1),
                       times = seq(0.1, 50, by = 5),
                       n = 500,
                       rfuncs = c("runif", "rnorm", "rexp"),
                       rargs = c("min = 0.18, max = 0.22",
                                 "mean = 0.2, sd = 0.2 / 3",
                                 "rate = 1 / 3"),
                       sobol_method = "Martinez",
                       ode_method = "adams",
                       parallel_eval = TRUE,
                       parallel_eval_ncores = 2)
```

ODEsobol.ODEnetwork *Sobol' Sensitivity Analysis for Objects of Class* ODEnetwork

---

Description

ODEsobol.ODEnetwork performs the variance-based Sobol' sensitivity analysis for objects of class ODEnetwork. Package ODEnetwork is required for this function to work.

Usage

```
## S3 method for class 'ODEnetwork'
ODEsobol(mod, pars, times, n = 1000, rfuncs = "runif",
  rargs = "min = 0, max = 1", sobol_method = "Martinez",
  ode_method = "lsoda", parallel_eval = FALSE, parallel_eval_ncores = NA,
  ...)
```

Arguments

mod	[ODEnetwork] list of class ODEnetwork.
pars	[character(k)] names of the parameters to be included as input variables in the Sobol' sensitivity analysis. All parameters must be contained in names(ODEnetwork::createParamVec(mod)) and must not be derivable from other parameters supplied (e.g. "k.2.1" can be derived from "k.1.2", so supplying "k.1.2" suffices).
times	[numeric] points of time at which the sensitivity analysis should be executed (vector of arbitrary length). The first point of time must be greater than zero.
n	[integer(1)] number of random parameter values used to estimate the Sobol' sensitivity indices by Monte Carlo simulation. Defaults to 1000.
rfuncs	[character(1 or k)] names of the functions used to generate the n random values for the k parameters. Can be of length 1 or k. If of length 1, the same function is used for all parameters. Defaults to "runif", so a uniform distribution is assumed for all parameters.
rargs	[character(1 or k)] arguments to be passed to the functions in rfuncs. Can be of length 1 or k. If of length 1, the same arguments are used for all parameters. Each element of rargs has to be a string of the form "tag1 = value1, tag2 = value2, ...",

	see example below. Default is "min = 0, max = 1", so (together with the default value of rfuncs) a uniform distribution on [0, 1] is assumed for all parameters.
sobol_method	[character(1)] either "Jansen" or "Martinez", specifying which modification of the variance-based Sobol' method shall be used. Defaults to "Martinez".
ode_method	[character(1)] method to be used for solving the ODEs in situations where the solution has to be determined numerically, see ode for details. Defaults to "lsoda".
parallel_eval	[logical(1)] logical indicating if the evaluation of the ODE model shall be performed parallelized.
parallel_eval_ncores	[integer(1)] number of processor cores to be used for parallelization. Only applies if parallel_eval = TRUE. If set to NA (as per default) and parallel_eval = TRUE, 1 processor core is used.
...	further arguments passed to or from other methods.

Details

If the object of class ODEnetwork supplied for mod doesn't include any events, the solution of the ODE network is determined analytically using simuNetwork. In the presence of events, simuNetwork uses ode to solve the ODE network numerically.

The sensitivity analysis is done for all state variables and all timepoints simultaneously. If sobol_method = "Jansen", soboljansen from the package sensitivity is used to estimate the Sobol' sensitivity indices and if sobol_method = "Martinez", sobolmartinez is used (also from the package sensitivity).

Value

List of length 2 * nrow(mod$state) and of class ODEsobol containing in each element a list of the Sobol' sensitivity analysis results for the corresponding state variable (i.e. first order sensitivity indices S and total sensitivity indices T) for every point of time in the times vector. This list has an extra attribute "sobol_method" where the value of argument sobol_method is stored (either "Jansen" or "Martinez").

Note

In situations where the solution of the ODE model has to be determined numerically, it might be helpful to try a different type of ODE-solver (argument ode_method) if the simulation of the model takes too long. The ode_methods "vode", "bdf", "bdf_d", "adams", "impAdams" and "impAdams_d" might be faster than the standard ode_method "lsoda".

If n is too low, the Monte Carlo estimation of the sensitivity indices might be very bad and even produce first order indices < 0 or total indices > 1. First order indices in the interval [-0.05, 0) and total indices in (1, 1.05] are considered as minor deviations and set to 0 resp. 1 without a warning. First order indices < -0.05 or total indices > 1.05 are considered as major deviations. They remain unchanged and a warning is thrown. Up to now, first order indices > 1 or total indices < 0 haven't occured yet. If this should be the case, please contact the package author.

Author(s)

Frank Weber

See Also

soboljansen, sobolmartinez, plot.ODEsobol

Examples

```
##### A network of 4 mechanical oscillators connected in a circle #####
# Definition of the network using the package "ODEnetwork":
M_mat <- rep(2, 4)
K_mat <- diag(rep(2 * (2*pi*0.17)^2, 4))
K_mat[1, 2] <- K_mat[2, 3] <-
  K_mat[3, 4] <- K_mat[1, 4] <- 2 * (2*pi*0.17)^2 / 10
D_mat <- diag(rep(0.05, 4))
library("ODEnetwork")
lfonet <- ODEnetwork(masses = M_mat, dampers = D_mat, springs = K_mat)
# The parameters to be included in the sensitivity analysis and their lower
# and upper boundaries:
LFOpars <- c("k.1", "k.2", "k.3", "k.4",
             "d.1", "d.2", "d.3", "d.4")
LFObinf <- c(rep(0.2, 4), rep(0.01, 4))
LFObsup <- c(rep(20, 4), rep(0.1, 4))
# Setting of the initial values of the state variables:
lfonet <- setState(lfonet, state1 = rep(2, 4), state2 = rep(0, 4))
# The timepoints of interest:
LFOtimes <- seq(25, 150, by = 2.5)
# Sobol' sensitivity analysis (here only with n = 500, but n = 1000 is
# recommended):
set.seed(1739)
# Warning: The following code might take very long! There are warnings
# occurring which might be due to "n" being too low.

suppressWarnings(
  LFOres_sobol <- ODEsobol(mod = lfonet,
                           pars = LFOpars,
                           times = LFOtimes,
                           n = 500,
                           rfuncs = "runif",
                           rargs = paste0("min = ", LFObinf,
                                          ", max = ", LFObsup),
                           sobol_method = "Martinez",
                           parallel_eval = TRUE,
                           parallel_eval_ncores = 2)
)
```

plot.ODEmorris — *Plot of the Results of Morris Screening for Objects of Class* ODEmorris

Description

plot.ODEmorris plots the results of Morris screening for objects of class ODEmorris.

Usage

```
## S3 method for class 'ODEmorris'
plot(x, pars_plot = NULL, state_plot = names(x)[1],
  kind = "sep", colors_pars = NULL, main_title = NULL,
  legendPos = "outside", type = "l", ...)
```

Arguments

x	[ODEmorris] output of ODEmorris (of class ODEmorris).
pars_plot	[character(k)] names of the k parameters to be plotted. If NULL (the default), all parameters are plotted.
state_plot	[character(1)] name of the state variable to be plotted. Defaults to the name of the first state variable.
kind	[character(1)] kind of the plot, choose between "sep" and "trajec" (see details).
colors_pars	[character(>= k)] vector of the colors to be used for the k different parameters. Must be at least of length k (only the first k elements will be used, though). If NULL (the default), rainbow(k) is used.
main_title	[character(1)] title for the plot. If kind = "sep", this is the overall title for the two separate plots. If NULL (the default), a standard title is generated.
legendPos	[character(1)] keyword for the legend position, either one of those specified in legend or "outside" (the default), which means the legend is placed under the plot (useful, if there are many parameters in the model).
type	[character(1)] plot type, i.e. "p", "l", "b", "c", "o", "s", "h" or "n". Defaults to "l".
...	additional arguments passed to plot.default.

Details

Morris sensitivity indices are plotted for one state variable (chosen by argument `state_plot`) and the parameters named in `pars_plot`. If no parameters are named in `pars_plot`, the sensitivity indices for all parameters are plotted. There are two kinds of plots:

- `kind = "sep"`: separate plots of the Morris sensitivity indices μ^* and σ against time
- `kind = "trajec"`: plot of μ^* against σ

Value

`TRUE` (invisible; for testing purposes).

Note

Not all plotting arguments can be passed by `...`, for example `xlab` and `ylab` are fixed.

Author(s)

Stefan Theers, Frank Weber

See Also

`ODEmorris`, `morris`

Examples

```
##### Lotka-Volterra equations #####
LVmod <- function(Time, State, Pars) {
  with(as.list(c(State, Pars)), {
    Ingestion    <- rIng  * Prey * Predator
    GrowthPrey   <- rGrow * Prey * (1 - Prey/K)
    MortPredator <- rMort * Predator

    dPrey        <- GrowthPrey - Ingestion
    dPredator    <- Ingestion * assEff - MortPredator

    return(list(c(dPrey, dPredator)))
  })
}
LVpars  <- c("rIng", "rGrow", "rMort", "assEff", "K")
LVbinf <- c(0.05, 0.05, 0.05, 0.05, 1)
LVbsup <- c(1.00, 3.00, 0.95, 0.95, 20)
LVinit  <- c(Prey = 1, Predator = 2)
LVtimes <- c(0.01, seq(1, 50, by = 1))
set.seed(7292)
# Warning: The following code might take very long!

LVres_morris <- ODEmorris(mod = LVmod,
                          pars = LVpars,
                          state_init = LVinit,
                          times = LVtimes,
                          binf = LVbinf,
                          bsup = LVbsup,
                          r = 500,
                          design = list(type = "oat",
                                        levels = 10, grid.jump = 1),
                          scale = TRUE,
                          ode_method = "lsoda",
                          parallel_eval = TRUE,
                          parallel_eval_ncores = 2)
my_cols <- c("firebrick", "orange2", "dodgerblue",
             "forestgreen", "black")
plot(LVres_morris, kind = "sep", colors_pars = my_cols)
plot(LVres_morris, pars_plot = c("rGrow", "rMort"), state_plot = "Predator",
     kind = "trajec", colors_pars = my_cols[2:3])


##### A network of 4 mechanical oscillators connected in a circle #####
M_mat <- rep(2, 4)
K_mat <- diag(rep(2 * (2*pi*0.17)^2, 4))
K_mat[1, 2] <- K_mat[2, 3] <-
  K_mat[3, 4] <- K_mat[1, 4] <- 2 * (2*pi*0.17)^2 / 10
D_mat <- diag(rep(0.05, 4))
library("ODEnetwork")
lfonet <- ODEnetwork(masses = M_mat, dampers = D_mat, springs = K_mat)
LFOpars <- c("k.1", "k.2", "k.3", "k.4",
             "d.1", "d.2", "d.3", "d.4")
LFObinf <- c(rep(0.2, 4), rep(0.01, 4))
LFObsup <- c(rep(20, 4), rep(0.1, 4))
lfonet <- setState(lfonet, state1 = rep(2, 4), state2 = rep(0, 4))
LFOtimes <- seq(25, 150, by = 2.5)
set.seed(283)
# Warning: The following code might take very long!

LFOres_morris <- ODEmorris(mod = lfonet,
                           pars = LFOpars,
                           times = LFOtimes,
                           binf = LFObinf,
                           bsup = LFObsup,
                           r = 500,
                           design = list(type = "oat",
                                         levels = 10, grid.jump = 1),
                           scale = TRUE,
                           parallel_eval = TRUE,
                           parallel_eval_ncores = 2)
plot(LFOres_morris, pars_plot = paste0("k.", 1:4), state_plot = "x.2",
     kind = "sep", colors_pars = my_cols)
```

plot.ODEsobol *Plot of the Results of Sobol' Sensitivity Analysis for Objects of Class* ODEsobol

Description

`plot.ODEsobol` plots the results of Sobol' SA for objects of class `ODEsobol`.

Usage

```
## S3 method for class 'ODEsobol'
plot(x, pars_plot = NULL, state_plot = names(x)[1],
  colors_pars = NULL, main_title = NULL, legendPos = "outside",
  type = "l", ...)
```

Arguments

`x`	`[ODEsobol]` output of `ODEsobol` (of class `ODEsobol`).
`pars_plot`	`[character(k)]` names of the `k` parameters to be plotted. If `NULL` (the default), all parameters are plotted.
`state_plot`	`[character(1)]` name of the state variable to be plotted. Defaults to the name of the first state variable.
`colors_pars`	`[character(>= k)]` vector of the colors to be used for the `k` different parameters. Must be at least of length `k` (only the first `k` elements will be used, though). If `NULL` (the default), `rainbow(k)` is used.
`main_title`	`[character(1)]` common title for the two graphics. Default is `NULL`, which means an automatic title is generated.
`legendPos`	`[character(1)]` keyword for the legend position, either one of those specified in `legend` or `"outside"` (the default), which means the legend is placed under the plot (useful, if there are many parameters in the model).
`type`	`[character(1)]` plot type, i.e. `"p"`, `"l"`, `"b"`, `"c"`, `"o"`, `"s"`, `"h"` or `"n"`. Defaults to `"l"`.
`...`	additional arguments passed to `plot.default`.

Details

First order and total Sobol' sensitivity indices are plotted for one state variable (chosen by argument `state_plot`) and the parameters named in `pars_plot` against time. If no parameters are named in `pars_plot`, the sensitivity indices for all parameters are plotted.

Value

TRUE (invisible; for testing purposes).

Note

Not all arguments of `plot.default` can be passed by `...`, for example `xlab` and `ylab` are fixed.

Author(s)

Stefan Theers, Frank Weber

See Also

`ODEsobol`, `soboljansen`,`sobolmartinez`

Examples

```
##### Lotka-Volterra equations #####
LVmod <- function(Time, State, Pars) {
  with(as.list(c(State, Pars)), {
    Ingestion    <- rIng  * Prey * Predator
    GrowthPrey   <- rGrow * Prey * (1 - Prey/K)
    MortPredator <- rMort * Predator

    dPrey        <- GrowthPrey - Ingestion
    dPredator    <- Ingestion * assEff - MortPredator

    return(list(c(dPrey, dPredator)))
  })
}
LVpars  <- c("rIng", "rGrow", "rMort", "assEff", "K")
LVbinf <- c(0.05, 0.05, 0.05, 0.05, 1)
LVbsup <- c(1.00, 3.00, 0.95, 0.95, 20)
LVinit  <- c(Prey = 1, Predator = 2)
LVtimes <- c(0.01, seq(1, 50, by = 1))
set.seed(59281)
# Warning: The following code might take very long!

LVres_sobol <- ODEsobol(mod = LVmod,
                        pars = LVpars,
                        state_init = LVinit,
                        times = LVtimes,
                        n = 500,
                        rfuncs = "runif",
                        rargs = paste0("min = ", LVbinf,
                                       ", max = ", LVbsup),
                        sobol_method = "Martinez",
                        ode_method = "lsoda",
                        parallel_eval = TRUE,
                        parallel_eval_ncores = 2)
my_cols <- c("firebrick", "orange2", "dodgerblue",
             "forestgreen", "black")
plot(LVres_sobol, colors_pars = my_cols)
plot(LVres_sobol, pars_plot = c("rGrow", "rMort"), state_plot = "Predator",
     colors_pars = my_cols[2:3])


##### A network of 4 mechanical oscillators connected in a circle #####
M_mat <- rep(2, 4)
K_mat <- diag(rep(2 * (2*pi*0.17)^2, 4))
```

```
K_mat[1, 2] <- K_mat[2, 3] <-
  K_mat[3, 4] <- K_mat[1, 4] <- 2 * (2*pi*0.17)^2 / 10
D_mat <- diag(rep(0.05, 4))
library("ODEnetwork")
lfonet <- ODEnetwork(masses = M_mat, dampers = D_mat, springs = K_mat)
LFOpars <- c("k.1", "k.2", "k.3", "k.4",
             "d.1", "d.2", "d.3", "d.4")
LFObinf <- c(rep(0.2, 4), rep(0.01, 4))
LFObsup <- c(rep(20, 4), rep(0.1, 4))
lfonet <- setState(lfonet, state1 = rep(2, 4), state2 = rep(0, 4))
LFOtimes <- seq(25, 150, by = 2.5)
set.seed(1739)
# Warning: The following code might take very long!

suppressWarnings(
  LFOres_sobol <- ODEsobol(mod = lfonet,
                           pars = LFOpars,
                           times = LFOtimes,
                           n = 500,
                           rfuncs = "runif",
                           rargs = paste0("min = ", LFObinf,
                                          ", max = ", LFObsup),
                           sobol_method = "Martinez",
                           parallel_eval = TRUE,
                           parallel_eval_ncores = 2)
)
plot(LFOres_sobol, pars_plot = paste0("k.", 1:4), state_plot = "x.2",
     colors_pars = my_cols)
```

tdcc *A Measure of Top-Down Correlation*

Description

With the use of Savage scores, the Top-Down Correlation Coefficient TDCC compares b rankings.

Usage

```
tdcc(ranks, pearson = FALSE, plot = FALSE)
```

Arguments

ranks	[matrix(nrow = b, ncol = k)] (bxk)-matrix of the ranks of the k variables for each of the b sensitivity analyses, ties are neglected, must be integers.
pearson	[logical(1)] Should the ordinary Pearson coefficient with Savage scores be computed (b = 2)? Default is FALSE.

`plot`	`[logical(1)]` Should scatter plots showing rankings and Savage scores be created (`b = 2`)? Default is `FALSE`.

Details

NOTE: As the implementation of the coefficient of concordance is still defective, please use the Pearson coefficient!

Value

A named vector with components:

- `kendall`: Coefficient of concordance.
- `pearson`: Pearson coefficient (only if `pearson = TRUE`).

Author(s)

Stefan Theers

References

R. L. Iman and W. J. Conover, *A Measure of Top-Down Correlation*, Technometrics, Vol. 29, No. 3 (Aug., 1987), pp. 351–357.

Examples

```
# b=2 sensitivity analysis techniques A and B that rate the influence of
# k=20 variables/ input parameters (example taken from Iman and Conover, 1987):
ranking <- rbind(A = 1:20,
                 B = c(1,3,2,4,16,10,19,12,18,17,
                       20,5,14,7,8,11,6,15,9,13))
tdcc(ranking, pearson = TRUE, plot = TRUE)
```

Index