Package ‘ODEnetwork’

April 3, 2020

Version 1.3.2

Title Network of Differential Equations

Description Simulates a network of ordinary differential equations of order two. The package provides an easy interface to construct networks. In addition you are able to define different external triggers to manipulate the trajectory. The method is described by Surmann, Ligges, and Weihs (2014) <doi:10.1109/ENERGYCON.2014.6850482>.

URL https://github.com/surmann/ODEnetwork

BugReports https://github.com/surmann/ODEnetwork/issues

License LGPL-3

Encoding UTF-8

Depends R (>= 3.0.2), checkmate (>= 1.5), deSolve

Suggests testthat, knitr, rmarkdown

LazyData yes

ByteCompile yes

RoxygenNote 7.1.0

VignetteBuilder knitr

Language en-GB

NeedsCompilation no

Author Dirk Surmann [aut, cre] (<https://orcid.org/0000-0003-0873-137X>)

Maintainer Dirk Surmann <surmann@statistik.tu-dortmund.de>

Repository CRAN

Date/Publication 2020-04-03 12:50:02 UTC

R topics documented:

`calcResonances` *Calculate Resonance Frequencies*

Description

Calculates the resonance frequencies of a given `ODEnetwork`. The resonance frequencies are calculated without respect to the dampers and neighbourhood structure.

Usage

```
calcResonances(odenet)
```

Arguments

`odenet`	Object of class `ODEnetwork`.

Value

a data frame with a vector of resonance frequencies.

Examples

```
masses <- 1
dampers <- as.matrix(0.1)
springs <- as.matrix(4)
odenet <- ODEnetwork(masses, dampers, springs)
calcResonances(odenet)
```

convertCoordinates *Converts coordinates between cartesian and polar*

Description

Converts a given matrix with two rows from polar to cartesian coordinates and vice versa.

Usage

```
convertCoordinates(coords, convertto = "cartesian")
```

Arguments

coords	[matrix] Matrix with two columns. Each row contains the pair (x, y) in cartesian coordinates or (radius, angle) in polar coordinates. The angle is given in radian [0, 2*pi]
convertto	[character] Defines the target coordinate system for conversion. Options are "cartesian" and "polar". Default: "cartesian"

Value

a matrix with converted coordinates

Examples

```
if (interactive()) {
 coordsK <- rbind(c(3, 0), c(1, 3), c(0, 2), c(-3, 1), c(-1, 0), c(-1, -3), c(0, -2), c(2, -3))
  coordsP <- convertCoordinates(coordsK, "polar")
}
```

createEvents *Creates Events*

Description

Creates functions for constant and linear events of the given ODEnetwork to know when events have to be replaced or forced.

Usage

```
createEvents(odenet)
```

Arguments

odenet [ODEnetwork]
List of class ODEnetwork.

Value

an extended list of class [ODEnetwork].

Examples

```
if (interactive()) {
  masses <- 1
  dampers <- as.matrix(1.5)
  springs <- as.matrix(4)
  odenet <- ODEnetwork(masses, dampers, springs)
  eventdat <- data.frame(  var = c("x.1", "x.1")
                         , time = c(1, 3)
                         , value = c(1, 3)
                         , stringsAsFactors = TRUE
                         )
  odenet <- setState(odenet, 0, 0)
  odenet <- setEvents(odenet, eventdat)
  createEvents(odenet)
}
```

---

createJacobian *Create Jacobian matrix for the parameters of the ODE.*

Description

Creates the Jacobian matrix for a special set of parameters of the ODE. The first n columns contain the derivatives with respect to d_ii, followed by the derivatives with respect to k_ii. The last 2*n columns include the derivatives with respect to the states.

Usage

```
createJacobian(odenet, ParamVec = NA)
```

Arguments

odenet [ODEnetwork]
List of class ODEnetwork.

ParamVec [vector] of length n
Named vector to overwrite corresponding parameters (see updateOscillators). Masses start with "m." followed by a number (e.g.: "m.12"). Dampers start with "d." followed by one or two numbers separated by a dot (e.g.: "d.2", "d.5.6"). Springs start with "k.", like dampers (e.g.: "k.4", "k.3.9"). The triangle elements of the dampers and springs are characterised by increasing numbers. A name

"d.3.5" is correct, in contrast to "d.5.3" which is ignored. This is done to speed up, because the matrices are symmetric. State1 and state2 start with "st1." or "st2." respectively, followed by a number (e.g.: "st1.15", "st2.8"). If the vector is set, the following parameters are ignored.

Value

the Jacobian matrix of size 2n*4n.

Examples

```
masses <- 4:6
dampers <- diag(1:3)
springs <- diag(7:9)
odenet <- ODEnetwork(masses, dampers, springs)
position <- rep(10, 3)
velocity <- rep(0, 3)
odenet <- setState(odenet, position, velocity)
jac <- createJacobian(odenet)
```

---

createOscillators *Creates Set of Differential Equations*

Description

Creates the set of differential equations of order one from the ODEnetwork.

Usage

```
createOscillators(odenet)
```

Arguments

odenet	[ODEnetwork] List of class ODEnetwork.

Value

a function with a set of differential equations of order one to use in a numerical step

Examples

```
if (interactive()) {
  masses <- c(1, 2)
  dampers <- diag(c(0.1, 0.5))
  dampers[1, 2] <- 0.05
  springs <- diag(c(4, 10))
  springs[1, 2] <- 6
  odenet <- ODEnetwork(masses, dampers, springs)
  createOscillators(odenet)
}
```

createParamVec *Creates Parameter Vector*

Description

Creates a vector with assigned parameters of the given ODEnetwork.

Usage

```
createParamVec(odenet)
```

Arguments

odenet	[ODEnetwork] List of class ODEnetwork.

Value

a named vector with assigned values

Examples

```
if (interactive()) {
  masses <- c(1, 2)
  dampers <- diag(c(0.1, 0.5))
  dampers[1, 2] <- 0.05
  springs <- diag(c(4, 10))
  springs[1, 2] <- 6
  odenet <- ODEnetwork(masses, dampers, springs)
  createParamVec(odenet)
}
```

createState *Creates starting State Vector*

Description

Creates a vector with the starting state of the given ODEnetwork.

Usage

```
createState(odenet)
```

Arguments

odenet	[ODEnetwork] List of class ODEnetwork.

Value

a named vector with assigned starting state

Examples

```
if (interactive()) {
  masses <- c(1, 2)
  dampers <- diag(c(0.1, 0.5))
  dampers[1, 2] <- 0.05
  springs <- diag(c(4, 10))
  springs[1, 2] <- 6
  odenet <- ODEnetwork(masses, dampers, springs)
  createState(odenet)
}
```

---

estimateDistances *Estimate distances between oscillators*

Description

Estimates the distances between the oscillators of a `ODEnetwork` from an equilibrium state.

Usage

```
estimateDistances(
  odenet,
  equilibrium,
  distGround = c("combined", "individual", "fixed", c("A", "B", "123", "A")),
  optim.control = list()
)
```

Arguments

`odenet` [ODEnetwork]
List of class `ODEnetwork`.

`equilibrium` [numeric(n)]
The desired equilibrium positions of the oscillators.

`distGround` [character(1)] or [character(n)]
`"combined"` estimates one value for all distances of the oscillators to the ground. Optimisation starts from `median(equilibrium)`.
`"individual"` estimates individual distance values for every oscillator. Optimisation starts from `equilibrium`.
`"fixed"` no estimation of the distances to the ground. Set to diagonal of distances matrix in `ODEnetwork`.
`character(n)` specifies groups of oscillators which distances to the ground are estimated by the same value. Optimisation starts from `median(equilibrium)` of the specified groups.
Default is `"combined"`

`optim.control` [list()]
A list of control parameters for optim. See `optim`.

Value

an extended list of class `ODEnetwork`.
Matrix of distances is added or overwritten.

Examples

```
masses <- c(1, 1)
dampers <- diag(c(1, 1))
springs <- diag(c(1, 1))
springs[1, 2] <- 1
equilibrium <- c(1/3, 5/3)
odenet <- ODEnetwork(masses, dampers, springs)
estimateDistances(odenet, equilibrium)$distances
estimateDistances(odenet, equilibrium, distGround="individual")$distances
```

---

getResult *Get Result*

Description

Getting result from numerical solving algorithm of given `ODEnetwork`.

Usage

```
getResult(odenet)
```

Arguments

`odenet` [ODEnetwork]
Object of class `ODEnetwork`.

Value

a matrix with columns time and states 1 and 2 of class `deSolve`.

Examples

```
masses <- 1
dampers <- as.matrix(0.1)
springs <- as.matrix(4)
odenet <- ODEnetwork(masses, dampers, springs)
getResult(odenet)
```

ODEnetwork *Constructor of the class ODEnetwork*

Description

Creates a list of class ODEnetwork. The coordinate type can be set to cartesian (position and velocity) or to polar coordinates (angle and magnitude).

Usage

```
ODEnetwork(masses, dampers, springs, cartesian = TRUE, distances = NA)
```

Arguments

masses — [vector] of length n
The masses of the mechanical oscillators.

dampers — [matrix] quadratic of size n
The dampers of the mechanical oscillators on the main diagonal. Connecting dampers between oscillators on the upper triangle. (Will be copied automatically to create a symmetric matrix.)

springs — [matrix] quadratic of size n
The springs are defined in the network like matrix of dampers.

cartesian — [boolean(1)]
If TRUE, state1 and state2 are position and velocity, otherwise angle and magnitude. Default is TRUE.

distances — [matrix] quadratic of size n
Describes the length of each spring. Elements on the main diagonal describe spring length connecting the masses to the ground. All upper triangle elements describe spring distance between two masses $i < j$. Default is NA, which is equivalent to a zero matrix. (Value will be copied automatically to lower triangle creating a symmetric matrix.)

Value

a list of class [ODEnetwork].

Examples

```
mM <- c(40, 10, 10)
mD <- diag(c(1, 5, 0))
mD[1, 2] <- 1
mD[2, 3] <- 1
mK <- diag(c(50, 50, 0))
mK[1, 2] <- 10
mK[2, 3] <- 10
odenet <- ODEnetwork(mM, mD, mK)
```

plot.ODEnetwork *Plots Results of ODEnetwork*

Description

Plots the results of simuNetwork of the given ODEnetwork in different ways.

Usage

```
## S3 method for class 'ODEnetwork'
plot(x, ..., state = "12", var = NULL)
```

Arguments

x	[ODEnetwork] List of class ODEnetwork.
...	Additional arguments.
state	[character] The type of result, that is plotted. If 1, only state1 (position or angle) is plotted over time. If 2, only state2 (velocity or magnitude) is plotted over time. If 12, state1 and state2 are plotted over time. If 1vs2, state2 is plotted over state1. Default is state12
var	[numeric(n)] Subset of variables to plot. Default is NULL, which plots all variables.

Examples

```
masses <- c(1, 2)
dampers <- diag(c(0.1, 0.5))
dampers[1, 2] <- 0.05
springs <- diag(c(4, 10))
springs[1, 2] <- 6
odenet <- ODEnetwork(masses, dampers, springs)
odenet <- setState(odenet, c(1, 3), c(0, 0))
odenet <- simuNetwork(odenet, seq(0, 10, by = 0.05))
plot(odenet)
plot(odenet, var = 2L)
plot(odenet, state = "1")
plot(odenet, state = "2")
plot(odenet, state = "1vs2")
```

setEvents *Setting Events*

Description

Sets different types of events for the given ODEnetwork.

Usage

```
setEvents(odenet, events, type = "dirac")
```

Arguments

odenet	[ODEnetwork] List of class ODEnetwork.
events	[data.frame] Data frame with a time base and named column per variable. See events for detailed definition of events.
type	[character] The type of the events to use. Possible values are dirac, constant or linear. Type dirac sets the current state at a given time point to a new value. Constant sets the state to the given value and the state does not change until setting new value or the end of events. Linear interpolates linear between events and sets the state variables to this value. Default is dirac.

Value

an extended list of class [ODEnetwork].

Examples

```
masses <- 1
dampers <- as.matrix(1.5)
springs <- as.matrix(4)
odenet <- ODEnetwork(masses, dampers, springs)
eventdat <- data.frame(  var = c("x.1", "x.1")
                       , time = c(1, 3)
                       , value = c(1, 3)
                       , stringsAsFactors = TRUE
                       )
odenet <- setState(odenet, 0, 0)
odenet <- setEvents(odenet, eventdat)
odenet <- simuNetwork(odenet, seq(0, 10, by = 0.1))
plot(odenet)
```

`setState` *Set starting State*

Description

Sets the starting State for the given ODEnetwork.

Usage

```
setState(odenet, state1, state2)
```

Arguments

odenet	[ODEnetwork] List of class ODEnetwork.
state1	[numeric(n)] Numeric vector of length n (same as in ODEnetwork) with position or angle.
state2	[numeric(n)] Numeric vector of length n (same as in ODEnetwork) with velocity or magnitude.

Value

an extended list of class [ODEnetwork].

Examples

```
masses <- 4:6
dampers <- diag(1:3)
springs <- diag(7:9)
odenet <- ODEnetwork(masses, dampers, springs)
position <- rep(10, 3)
velocity <- rep(0, 3)
odenet <- setState(odenet, position, velocity)
```

`simuNetwork` *Simulation of the Differential Equations*

Description

Simulates the given ODEnetwork over a time range.

Usage

```
simuNetwork(odenet, times, origin.min.time = FALSE, ...)
```

Arguments

`odenet`	`[ODEnetwork]` List of class `ODEnetwork`.
`times`	`[numeric]` Time sequence to calculate or simulate the ode network.
`origin.min.time`	`[logical(1L)]` Define origin of ode time. `FALSE` sets it to 0, `TRUE` to the minimum of `times`. Default is `FALSE`.
`...`	Additional arguments.

Value

an extended list of class [`ODEnetwork`].

Examples

```
masses <- 4:6
dampers <- diag(1:3)
springs <- diag(7:9)
odenet <- ODEnetwork(masses, dampers, springs)
position <- rep(10, 3)
velocity <- rep(0, 3)
odenet <- setState(odenet, position, velocity)
odenet <- simuNetwork(odenet = odenet, times = seq(0, 20))
```

`updateOscillators` *Update oscillator parameter of an existing ODE network.*

Description

Updates the parameters of an existing ODE network. Possible parameters are the oscillator configuration and the starting values. The function overwrites the parameters in the vector and matrices. It will not change the size of the network. It is possible to set a new vector and new matrices, or single parameters in a names vector.

Usage

```
updateOscillators(
  odenet,
  ParamVec = NULL,
  masses = NULL,
  dampers = NULL,
  springs = NULL,
  distances = NULL,
  state1 = NULL,
  state2 = NULL
)
```

Arguments

odenet
: [ODEnetwork]
List of class ODEnetwork.

ParamVec
: [vector] of length n
Named vector to overwrite corresponding parameters. Masses start with "m." followed by a number (e.g.: "m.12"). Dampers start with "d." followed by one or two numbers separated by a dot (e.g.: "d.2", "d.5.6"). Springs start with "k.", like dampers (e.g.: "k.4", "k.3.9") Distances start with "r.", like dampers (e.g.: "r.7", "r.1.9") The triangle elements of the dampers, springs, and distances are characterised by increasing numbers. A name "d.3.5" is correct, in contrast to "d.5.3" which is ignored. This is done to speed up, because the matrices are symmetric. State1 and state2 start with "st1." or "st2." respectively, followed by a number (e.g.: "st1.15", "st2.8"). If the vector is set, the following parameters are ignored.

masses
: [vector] of length n
The masses of the mechanical oscillators.

dampers
: [matrix] quadratic of size n
The dampers of the mechanical oscillators on the main diagonal. Connecting dampers between oscillators on the upper triangle. (Will be copied automatically to create a symmetric matrix.)

springs
: [matrix] quadratic of size n
The springs are defined in the network like matrix of dampers.

distances
: [matrix] quadratic of size n
Describe spring distance between two masses $i < j$. Negative value will be copied automatically to lower triangle.

state1
: [vector] of length n
Starting values of state 1 (position or angle).

state2
: [vector] of length n
Starting values of state 2 (velocity or magnitude).

Value

an extended list of class [ODEnetwork].

Examples

```
masses <- c(1:5)
dampers <- diag(11:15)
springs <- diag(21:25)
odenet <- ODEnetwork(masses, dampers, springs)
odenet <- updateOscillators(odenet, masses = c(3:7))
odenet <- updateOscillators(odenet, c(k.1.2 = 201, k.3.5 = 202, r.1 = 2))
# Warning: Following value is ignored, because it is on the lower triangle
odenet <- updateOscillators(odenet, c(d.2.1 = 101))
```

Index