Package: deBif (via r-universe)

August 26, 2024

Type Package

Title Bifurcation Analysis of Ordinary Differential Equation Systems

Version 0.1.9

Description Shiny application that performs bifurcation and phaseplane analysis of systems of ordinary differential equations. The package allows for computation of equilibrium curves as a function of a single free parameter, detection of transcritical, saddle-node and hopf bifurcation points along these curves, and computation of curves representing these transcritical, saddle-node and hopf bifurcation points as a function of two free parameters. The shiny-based GUI allows visualization of the results in both 2D- and 3D-plots. The implemented methods for solution localisation and curve continuation are based on the book ``Elements of applied bifurcation theory" (Kuznetsov, Y. A., 1995; ISBN: 0-387-94418-4).

License GPL-3

Encoding UTF-8

NeedsCompilation yes

Imports graphics, deSolve (>= 1.3), rootSolve (>= 1.8), rstudioapi (>= 0.13), shiny (>= 1.7), shinyjs (>= 2.1), shinydashboard (>= 0.7), shinydashboardPlus (>= 2.0)

Suggests knitr, R.rsp, rmarkdown

RoxygenNote 7.2.3

VignetteBuilder R.rsp

Depends R (>= 4.2)

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

RemoteUrl https://github.com/amderoos/debif

RemoteRef HEAD

RemoteSha a4e17d6278ec385ab87c0ed3efd0ff3f8591065c

Contents

`bifurcation` *Phaseplane analysis of a system of ODEs*

Description

`bifurcation`

Usage

```
bifurcation(model, state, parms, resume = TRUE, ...)
```

Arguments

`model`	(function, required)
	An R-function that computes the values of the derivatives in the ODE system (the model definition) at time t. The model must be defined as: model <- function(t, state, parms), where t is the current time point in the integration, state is the current value of the variables in the ODE #' system and parms is a vector or list of parameters. The return value of func should be a list, whose first and single element is a vector containing the derivatives of y with respect to time. The derivatives must be specified in the same order as the state variables state. The vector state and parms should both have name attributes for all their elements
`state`	(numeric vector, required)
	The initial (state) values for the ODE system. This vector should have name attributes for all its elements
`parms`	(numeric vector, required)
	The values of the parameters in the ODE system. This vector should have name attributes for all its elements
`resume`	(boolean, optional)

If TRUE the program will try to load the curves computed during the last session from the global variable '<model>BifCurves' and try to restore the numerical and plot settings by importing them from the global variable '<model>BifSettings', where the substring '<model>' is the name of the function describing the dynamics, which is passed as first argument to 'bifurcation()'. The program saves the curves computed during a session and the numerical and plot settings of this last session in these global variables '<model>BifCurves' and '<model>BifSettings'.

... (optional arguments)

Additional arguments that can be included at the command line to tweak graphical default values used by the application. Valid arguments are:

`lwd`: Line width (default 3)

`cex`: Base font size (default 1.2)

`tcl.len`: Length of axes ticks (default 0.03)

`bifsym`: Symbol used to mark a bifurcation point in an equilibrium curve (default: 8)

`biflblpos`: Location of label of a bifurcation point. Values of 1, 2, 3 and 4, respectively, indicate positions below, to the left of, above and to the right of the symbol marking the bifurcation point (default: 3)

`unstablelty`: Line style of curve section representing unstable equilibrium points (default: 3 (refers to dotted lines))

`saveplotas`: Possible values: "pdf" or "png" (default). Save plot to PDF or PNG file.

Details

bifurcation(model, state, parms, resume = TRUE, ...)

Value

None.

Examples

```
if(interactive()){
# The initial state of the system has to be specified as a named vector of state values.
state <- c(R=1, N=0.01)

# Parameters has to be specified as a named vector of parameters.
parms <- c(r=1, K=1, a=1, c=1, delta=0.5)

# The model has to be specified as a function that returns
# the derivatives as a list.
model <- function(t, state, parms) {
  with(as.list(c(state,parms)), {

    dR <- r*R*(1 - R/K) - a*R*N
    dN <- c*a*R*N - delta*N

   # The order of the derivatives in the returned list has to be
   # identical to the order of the state variables contained in
   # the argument "state"
    return(list(c(dR, dN)))
  })
}

bifurcation(model, state, parms)
}
```

`deBifExample` *Examples of phaseplane analysis of a system of ODEs*

Description

`deBifExample`

Usage

```
deBifExample(example)
```

Arguments

`example`	(string, optional)	
		Name of the example. If not provided a list of examples is returned

Details

deBifExample(example)

Function runs one of the examples provided with the deBif package

Value

None.

`deBifHelp` *Opens the deBif manual*

Description

`deBifHelp` opens the manual of the the deBif package in html format.

Usage

```
deBifHelp()
```

Value

None.

Examples

```
if(interactive()){
deBifHelp()
}
```

`deBifReset` *Reloads the deBif package*

Description

`deBifReset` unloads and reloads the deBif package.

Usage

```
deBifReset()
```

Value

None.

Examples

```
if(interactive()){
deBifReset()
}
```

phaseplane *Phaseplane analysis of a system of ODEs*

Description

phaseplane

Usage

```
phaseplane(model, state, parms, resume = TRUE, ...)
```

Arguments

model	(function, required) An R-function that computes the values of the derivatives in the ODE system (the model definition) at time t. The model must be defined as: model <- function(t, state, parms), where t is the current time point in the integration, state is the current value of the variables in the ODE #' system and parms is a vector or list of parameters. The return value of func should be a list, whose first and single element is a vector containing the derivatives of y with respect to time. The derivatives must be specified in the same order as the state variables state. The vector state and parms should both have name attributes for all their elements
state	(numeric vector, required) The initial (state) values for the ODE system. This vector should have name attributes for all its elements
parms	(numeric vector, required) The values of the parameters in the ODE system. This vector should have name attributes for all its elements
resume	(boolean, optional) If TRUE the program will try to load the curves computed during the last session from the global variable '<model>PhaseCurves' and try to restore the numerical and plot settings by importing them from the global variable '<model>PhaseSettings', where the substring '<model>' is the name of the function describing the dynamics, which is passed as first argument to 'bifurcation()'. The program saves the curves computed during a session and the numerical and plot settings of this last session in these global variables '<model>PhaseCurves' and '<model>PhaseSettings'.
...	(optional arguments)

Additional arguments that can be included at the command line to tweak graphical default values used by the application. Valid arguments are:

`lwd`: Line width (default 2)

`cex`: Base font size (default 1.2)

`tcl.len`: Length of axes ticks (default 0.03)

`saveplotas`: Possible values: "pdf" or "png" (default). Save plot to PDF or PNG file.

Details

phaseplane(model, state, parms, resume = TRUE, ...)

Value

None.

Examples

```
if(interactive()){
# The initial state of the system has to be specified as a named vector of state values.
state <- c(R=1, N=0.01)

# Parameters has to be specified as a named vector of parameters.
parms <- c(r=1, K=1, a=1, c=1, delta=0.5)

# The model has to be specified as a function that returns
# the derivatives as a list.
model <- function(t, state, parms) {
  with(as.list(c(state,parms)), {

    dR <- r*R*(1 - R/K) - a*R*N
    dN <- c*a*R*N - delta*N

   # The order of the derivatives in the returned list has to be
   # identical to the order of the state variables contained in
   # the argument "state"
    return(list(c(dR, dN)))
  })
}

phaseplane(model, state, parms)
}
```

Index