Package ‘phaseR’

October 12, 2019

Type Package
Title Phase Plane Analysis of One- And Two-Dimensional Autonomous ODE Systems
Version 2.1.3
Imports deSolve, graphics, grDevices, utils
Description Performs a qualitative analysis of one- and two-dimensional autonomous ordinary differential equation systems, using phase plane methods. Programs are available to identify and classify equilibrium points, plot the direction field, and plot trajectories for multiple initial conditions. In the one-dimensional case, a program is also available to plot the phase portrait. Whilst in the two-dimensional case, programs are additionally available to plot nullclines and stable/unstable manifolds of saddle points. Many example systems are provided for the user. For further details can be found in Grayling (2014) <doi:10.32614/RJ-2014-023>.
License MIT + file LICENSE
LazyData TRUE
Suggests knitr, rmarkdown, testthat
Date 2019-09-10
URL https://github.com/mjg211/phaseR
BugReports https://github.com/mjg211/phaseR/issues
RoxygenNote 6.1.1
Encoding UTF-8
VignetteBuilder knitr
NeedsCompilation no
Author Michael J Grayling [aut, cre] (<https://orcid.org/0000-0002-0680-6668>), Gerhard Burger [ctb] (<https://orcid.org/0000-0003-1062-5576>), Stephen P Ellner [ctb], John M Guckenheimer [ctb]
Maintainer Michael J Grayling <michael.grayling@newcastle.ac.uk>
Repository CRAN
Date/Publication 2019-10-12 07:30:02 UTC
R topics documented:

phaseR-package .......................................................... 3
.paramDummy ............................................................ 4
competition .............................................................. 4
drawManifolds .......................................................... 5
eexample ................................................................. 7
eexample10 ............................................................... 8
eexample11 ............................................................... 9
eexample12 ............................................................. 10
eexample13 ............................................................. 11
eexample14 ............................................................. 12
eexample15 ............................................................. 13
eexample2 ............................................................... 14
eexample3 ............................................................... 15
eexample4 ............................................................... 16
eexample5 ............................................................... 17
eexample6 ............................................................... 19
eexample7 ............................................................... 20
eexample8 ............................................................... 21
eexample9 ............................................................... 22
exponential .............................................................. 23
findEquilibrium ......................................................... 24
flowField ................................................................. 26
lindemannMechanism .................................................. 29
logistic ................................................................. 30
lotkaVolterra ........................................................... 31
monomolecular .......................................................... 32
morrisLecar ............................................................ 33
nullclines ............................................................... 34
numericalSolution ...................................................... 37
phasePlaneAnalysis .................................................... 38
phasePortrait .......................................................... 40
simplePendulum ....................................................... 42
SIR ................................................................. 43
stability ............................................................... 44
toggle ................................................................. 46
trajectory .............................................................. 47
vanDerPol ............................................................... 49
vonBertalanffy ........................................................ 51

Index ................................................................. 53
phaseR-package

**Phase plane analysis of one- and two-dimensional autonomous ODE systems**

**Description**

phaseR is an R package for the qualitative analysis of one- and two-dimensional autonomous ODE systems, using phase plane methods. Programs are available to identify and classify equilibrium points, plot the direction field, and plot trajectories for multiple initial conditions. In the one-dimensional case, a program is also available to plot the phase portrait. Whilst in the two-dimensional case, additionally programs are available to plot nullclines and stable/unstable manifolds of saddle points. Many example systems are provided for the user.

**Details**

- **Package:** phaseR
- **Type:** Package
- **Version:** 2.1
- **Date:** 2019-31-05
- **License:** GNU GPLv3

The package contains nine main functions for performing phase plane analyses:

- **drawManifolds:** Draws the stable and unstable manifolds of a saddle point in a two dimensional autonomous ODE system.
- **findEquilibrium:** Identifies a nearby equilibrium point of an autonomous ODE system based on a specified starting point.
- **flowField:** Plots the flow or velocity field of a one- or two-dimensional autonomous ODE system.
- **nullclines:** Plots the nullclines of a two-dimensional autonomous ODE system.
- **numericalSolution:** Numerically solves a two-dimensional autonomous ODE system in order to plot the two dependent variables against the independent variable.
- **phasePlaneAnalysis:** Provides a simple means of performing a phase plane analysis by typing only numbers in to the command line.
- **phasePortrait:** Plots the phase portrait of a one-dimensional autonomous ODE system, for use in classifying equilibria.
- **stability:** Performs stability, or perturbation, analysis in order to classify equilibria.
- **trajectory:** Numerically solves a one- or two-dimensional ODE system to plot trajectories in the phase plane.

In addition, the package contains over 25 derivative functions for example systems. Links to these can be found in the package index.
competition

An accompanying vignette containing further information, examples, and exercises, can also be accessed with vignette("phaseR").
This package makes use of the ode function in the package deSolve.

Author(s)

Michael J Grayling (michael.grayling@ncl.ac.uk)
Contributor(s): Gerhard Burger

.paramDummy

A function such that we can apply DRY in param documentation

Description

A function such that we can apply DRY in param documentation

Usage

.paramDummy(state.names)

Arguments

state.names The state names for ode functions that do not use positional states.

competition

The species competition model

Description

The derivative function of the species competition model, an example of a two-dimensional autonomous ODE system.

Usage

competition(t, y, parameters)

Arguments

t The value of t, the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.
y The values of x and y, the dependent variables, to evaluate the derivative at. Should be a numeric vector of length one.
parameters The values of the parameters of the system. Should be a numeric vector with parameters specified in the following order: r_1, K_1, \alpha_{12}, r_2, K_2, \alpha_{21}.
Details

competition evaluates the derivative of the following coupled ODE system at the point \((t, x, y)\):

\[
\frac{dx}{dt} = r_1 x (K_1 - x - \alpha_{12} y) / K_1, \quad \frac{dy}{dt} = r_2 y (K_2 - y - \alpha_{21} x) / K_2.
\]

Its format is designed to be compatible with \texttt{ode} from the \texttt{deSolve} package.

Value

Returns a \texttt{list} containing the values of the two derivatives at \((t, x, y)\).

Author(s)

Michael J Grayling

See Also

\texttt{ode}

drawManifolds  Stable and unstable manifolds

Description

Plots the stable and unstable manifolds of a saddle point. A search procedure is utilised to identify an equilibrium point, and if it is a saddle then its manifolds are added to the plot.

Usage

\[
\text{drawManifolds}(\text{deriv, } y0 = \text{NULL, parameters = NULL, tstep = 0.1, tend = 100, col = c("green", "red"), add.legend = TRUE, state.names = c("x", "y"), ...})
\]

Arguments

\begin{itemize}
\item \texttt{deriv} \quad A function computing the derivative at a point for the ODE system to be analysed. Discussion of the required structure of these functions can be found in the package vignette, or in the help file for the function \texttt{ode}.
\item \texttt{y0} \quad The initial point from which a saddle will be searched for. This can either be a \texttt{numeric vector} of \texttt{length} two, reflecting the location of the two dependent variables, or alternatively this can be specified as \texttt{NULL}, and then \texttt{locator} can be used to specify the initial point on a plot. Defaults to \texttt{NULL}.
\item \texttt{parameters} \quad Parameters of the ODE system, to be passed to \texttt{deriv}. Supplied as a \texttt{numeric vector}; the order of the parameters can be found from the \texttt{deriv} file. Defaults to \texttt{NULL}.
\end{itemize}
tstep

The step length of the independent variable, used in numerical integration. Decreasing the absolute magnitude of tstep theoretically makes the numerical integration more accurate, but increases computation time. Defaults to 0.01.

tend

The final time of the numerical integration performed to identify the manifolds.

col

Sets the colours used for the stable and unstable manifolds. Should be a character vector of length two. Will be reset accordingly if it is of the wrong length. Defaults to c("green","red").

add.legend

Logical. If TRUE, a legend is added to the plots. Defaults to TRUE.

state.names

The state names for ode functions that do not use positional states.

... Additional arguments to be passed to plot.

Value

Returns a list with the following components:

add.legend As per input.
col As per input, but with possible editing if a character vector of the wrong length was supplied.
deriv As per input.
parameters As per input.
stable.1 A numeric matrix whose columns are the numerically computed values of the dependent variables for part of the stable manifold.
stable.2 A numeric matrix whose columns are the numerically computed values of the dependent variables for part of the stable manifold.
tend As per input.
unstable.1 A numeric matrix whose columns are the numerically computed values of the dependent variables for part of the unstable manifold.
unstable.2 A numeric matrix whose columns are the numerically computed values of the dependent variables for part of the unstable manifold.
y0 As per input.
ystar Location of the identified equilibrium point.

Author(s)

Michael J Grayling, Stephen P Ellner, John M Guckenheimer
example1  

Example ODE system 1

Description

The derivative function of an example one-dimensional autonomous ODE system.

Usage

example1(t, y, parameters)

Arguments

t  The value of $t$, the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.

y  The value of $y$, the dependent variable, to evaluate the derivative at. Should be a numeric vector of length one.

parameters  The values of the parameters of the system. Not used here.

Details

example1 evaluates the derivative of the following ODE at the point $(t, y)$:

\[
\frac{dy}{dt} = 4 - y^2.
\]

Its format is designed to be compatible with ode from the deSolve package.

Value

Returns a list containing the value of the derivative at $(t, y)$.

Author(s)

Michael J Grayling

See Also

ode
Example ODE system 10

Description

The derivative function of an example two-dimensional autonomous ODE system.

Usage

example10(t, y, parameters)

Arguments

t
The value of \( t \), the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.

y
The values of \( x \) and \( y \), the dependent variables, to evaluate the derivative at. Should be a numeric vector of length two.

parameters
The values of the parameters of the system. Not used here.

Details

example10 evaluates the derivatives of the following coupled ODE system at the point \((t, x, y)\):

\[
\frac{dx}{dt} = -x + x^3, \quad \frac{dy}{dt} = -2y.
\]

Its format is designed to be compatible with ode from the deSolve package.

Value

Returns a list containing the values of the two derivatives at \((t, x, y)\).

Author(s)

Michael J Grayling

See Also

ode
Description

The derivative function of an example two-dimensional autonomous ODE system.

Usage

example11(t, y, parameters)

Arguments

- **t**
  - The value of \( t \), the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.

- **y**
  - The values of \( x \) and \( y \), the dependent variables, to evaluate the derivative at. Should be a numeric vector of length two.

- **parameters**
  - The values of the parameters of the system. Not used here.

Details

example11 evaluates the derivatives of the following coupled ODE system at the point \((t, x, y)\):

\[
\frac{dx}{dt} = x(3 - x - 2y), \quad \frac{dy}{dt} = -y(2 - x - y).
\]

Its format is designed to be compatible with ode from the deSolve package.

Value

Returns a list containing the values of the two derivatives at \((t, x, y)\).

Author(s)

Michael J Grayling

See Also

ode
Examples

# Plot the velocity field, nullclines and several trajectories
example11_flowField <- flowField(example11,
xlim = c(-5, 5),
ylim = c(-5, 5),
points = 21,
add = FALSE)
y0 <- matrix(c(4, 4, -1, -1,
-2, 1, 1, -1), 4, 2,
byrow = TRUE)
example11_nullclines <- nullclines(example11,
xlim = c(-5, 5),
ylim = c(-5, 5),
points = 200)
example11_trajectory <- trajectory(example11,
y0 = y0,
tlim = c(0, 10))

# Determine the stability of the equilibrium points
example11_stability_1 <- stability(example11, ystar = c(0, 0))
example11_stability_2 <- stability(example11, ystar = c(0, 2))
example11_stability_3 <- stability(example11, ystar = c(1, 1))
example11_stability_4 <- stability(example11, ystar = c(3, 0))

example12

Example ODE system 12

Description

The derivative function of an example two-dimensional autonomous ODE system.

Usage

example12(t, y, parameters)

Arguments

t The value of \( t \), the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.
y The values of \( x \) and \( y \), the dependent variables, to evaluate the derivative at. Should be a numeric vector of length two.
parameters The values of the parameters of the system. Not used here.

details

example12 evaluates the derivatives of the following coupled ODE system at the point \((t, x, y)\):

\[
\frac{dx}{dt} = x - y, \quad \frac{dy}{dt} = x^2 + y^2 - 2.
\]

Its format is designed to be compatible with \texttt{ode} from the \texttt{deSolve} package.
**Value**

Returns a list containing the values of the two derivatives at \((t, x, y)\).

**Author(s)**

Michael J Grayling

**See Also**

ode

**Examples**

```r
# Plot the velocity field, nullclines and several trajectories
example12_flowField <- flowField(example12,
xlim = c(-4, 4),
ylim = c(-4, 4),
points = 17,
add = FALSE)
y0 <- matrix(c(2, 2, -3, 0,
  0, 2, 0, -3), 4, 2,
byrow = TRUE)
example12_nullclines <- nullclines(example12,
xlim = c(-4, 4),
ylim = c(-4, 4),
points = 200)
example12_trajectory <- trajectory(example12,
y0 = y0,
tlim = c(0, 10))
# Determine the stability of the equilibrium points
example12_stability_1 <- stability(example12,
ystar = c(1, 1))
example12_stability_2 <- stability(example12,
ystar = c(-1, -1))
```

---

**Example ODE system 13**

**Description**

The derivative function of an example two-dimensional autonomous ODE system.

**Usage**

```r
example13(t, y, parameters)
```
Arguments

- **t**: The value of $t$, the independent variable, to evaluate the derivative at. Should be a **numeric vector** of **length** one.

- **y**: The values of $x$ and $y$, the dependent variables, to evaluate the derivative at. Should be a **numeric vector** of **length** two.

- **parameters**: The values of the parameters of the system. Not used here.

Details

`example13` evaluates the derivatives of the following coupled ODE system at the point $(t, x, y)$:

\[
\frac{dx}{dt} = 2 - x^2 - y^2, \quad \frac{dy}{dt} = x^2 - y^2.
\]

Its format is designed to be compatible with `ode` from the deSolve package.

Value

Returns a **list** containing the values of the two derivatives at $(t, x, y)$.

Author(s)

Michael J Grayling

See Also

- `ode`

Description

The derivative function of an example two-dimensional autonomous ODE system.

Usage

`example14(t, y, parameters)`

Arguments

- **t**: The value of $t$, the independent variable, to evaluate the derivative at. Should be a **numeric vector** of **length** one.

- **y**: The values of $x$ and $y$, the dependent variables, to evaluate the derivative at. Should be a **numeric vector** of **length** two.

- **parameters**: The values of the parameters of the system. Not used here.
Details

example15 evaluates the derivatives of the following coupled ODE system at the point \((t, x, y)\):

\[
\frac{dx}{dt} = x^2 - y - 10, \quad \frac{dy}{dt} = -3x^2 + xy.
\]

Its format is designed to be compatible with ode from the deSolve package.

Value

Returns a list containing the values of the two derivatives at \((t, x, y)\).

Author(s)

Michael J Grayling

See Also

ode

Example ODE system 15

Description

The derivative function of an example two-dimensional autonomous ODE system.

Usage

example15(t, y, parameters)

Arguments

\(t\) The value of \(t\), the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.

\(y\) The values of \(x\) and \(y\), the dependent variables, to evaluate the derivative at. Should be a numeric vector of length two.

parameters The values of the parameters of the system. Not used here.

Details

example15 evaluates the derivatives of the following coupled ODE system at the point \((t, x, y)\):

\[
\frac{dx}{dt} = x^2 - 3xy + 2x, \quad \frac{dy}{dt} = x + y - 1.
\]

Its format is designed to be compatible with ode from the deSolve package.
Value

Returns a list containing the values of the two derivatives at \((t, x, y)\).

Author(s)

Michael J Grayling

See Also

ode

Example ODE system 2

Description

The derivative function of an example one-dimensional autonomous ODE system.

Usage

```r
example2(t, y, parameters)
```

Arguments

t The value of \(t\), the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.
y The value of \(y\), the dependent variable, to evaluate the derivative at. Should be a numeric vector of length one.
parameters The values of the parameters of the system. Not used here.

Details

```latex
\frac{dy}{dt} = y(1 - y)(2 - y).
```

Its format is designed to be compatible with \texttt{ode} from the \texttt{deSolve} package.

Value

Returns a list containing the value of the derivative at \((t, y)\).

Author(s)

Michael J Grayling
See Also

ode

Examples

# Plot the flow field and several trajectories
example2_flowField <- flowField(example2,
    xlim = c(0, 4),
    ylim = c(-1, 3),
    system = "one.dim",
    add = FALSE,
    xlab = "t")

example2_trajectory <- trajectory(example2,
    y0 = c(-0.5, 0.5, 1.5, 2.5),
    tlim = c(0, 4),
    system = "one.dim")

# Plot the phase portrait
example2_phasePortrait <- phasePortrait(example2,
    ylim = c(-0.5, 2.5),
    frac = 0.5)

# Determine the stability of the equilibrium points
example2_stability_1 <- stability(example2,
    ystar = 0,
    system = "one.dim")
example2_stability_2 <- stability(example2,
    ystar = 1,
    system = "one.dim")
example2_stability_3 <- stability(example2,
    ystar = 2,
    system = "one.dim")

Example ODE system 3

Description

The derivative function of an example two-dimensional autonomous ODE system.

Usage

d3.example3(t, y, parameters)

Arguments

t The value of \( t \), the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.

y The values of \( x \) and \( y \), the dependent variables, to evaluate the derivative at. Should be a numeric vector of length two.

parameters The values of the parameters of the system. Not used here.
Details

example4 evaluates the derivatives of the following coupled ODE system at the point \((t, x, y)\):

\[
\frac{dx}{dt} = -x, \quad \frac{dy}{dt} = -4x.
\]

Its format is designed to be compatible with \texttt{ode} from the \texttt{deSolve} package.

Value

Returns a \texttt{list} containing the values of the two derivatives at \((t, x, y)\).

Author(s)

Michael J Grayling

See Also

\texttt{ode}

---

\texttt{example4} \hspace{1cm} \textit{Example ODE system 4}

Description

The derivative function of an example two-dimensional autonomous ODE system.

Usage

\texttt{example4(t, y, parameters)}

Arguments

t \hspace{1cm} The value of \(t\), the independent variable, to evaluate the derivative at. Should be a \texttt{numeric vector} of \texttt{length} one.

y \hspace{1cm} The values of \(x\) and \(y\), the dependent variables, to evaluate the derivative at. Should be a \texttt{numeric vector} of \texttt{length} two.

parameters \hspace{1cm} The values of the parameters of the system. Not used here.

Details

example4 evaluates the derivatives of the following coupled ODE system at the point \((t, x, y)\):

\[
\frac{dx}{dt} = -x, \quad \frac{dy}{dt} = 4x.
\]

Its format is designed to be compatible with \texttt{ode} from the \texttt{deSolve} package.
example5

Value
Returns a list containing the values of the two derivatives at \((t, x, y)\).

Author(s)
Michael J Grayling

See Also
ode

Examples

# Plot the velocity field, nullclines and several trajectories
diff <- flowField(diff,
exlim = c(-3, 3),
ylim = c(-5, 5),
points = 19,
add = FALSE)
y0 <- matrix(c(1, 0, -1, 0, 2, 2,
-2, 2, -3, -4), 5, 2,
byrow = TRUE)
diff_nullclines <- nullclines(diff,
exlim = c(-3, 3),
ylim = c(-5, 5))
diff_trajectory <- trajectory(diff,
y0 = y0,
tlim = c(0, 10))

Example ODE system 5

Description
The derivative function of an example two-dimensional autonomous ODE system.

Usage
evaluation(t, y, parameters)

Arguments
t       The value of \(t\), the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.
y       The values of \(x\) and \(y\), the dependent variables, to evaluate the derivative at. Should be a numeric vector of length two.
parameters  The values of the parameters of the system. Not used here.
Details

dexample5 evaluates the derivatives of the following coupled ODE system at the point \((t, x, y)\):

\[
\frac{dx}{dt} = 2x + y, \quad \frac{dy}{dt} = 2x - y.
\]

Its format is designed to be compatible with \texttt{ode} from the \texttt{deSolve} package.

Value

Returns a \texttt{list} containing the values of the two derivatives at \((t, x, y)\).

Author(s)

Michael J Grayling

See Also

\texttt{ode}

Examples

# Plot the velocity field, nullclines, manifolds and several trajectories
example5_flowField <- flowField(example5,
   xlim = c(-3, 3),
   ylim = c(-3, 3),
   points = 19,
   add = FALSE)
y0 <- matrix(c(1, 0, -1, 0, 2, 2,
   -2, 2, 0, 3, 0, -3), 6, 2,
   byrow = TRUE)
example5_nullclines <- nullclines(example5,
   xlim = c(-3, 3),
   ylim = c(-3, 3))
example5_trajectory <- trajectory(example5,
   y0 = y0,
   tlim = c(0, 10))

# Plot x and y against t
example5_numericalSolution <- numericalSolution(example5,
   y0 = c(0, 3),
   tlim = c(0, 3))

# Determine the stability of the equilibrium point
example5_stability <- stability(example5,
   ystar = c(0, 0))
example6

Example ODE System 6

Description

The derivative function of an example two-dimensional autonomous ODE system.

Usage

example6(t, y, parameters)

Arguments

t The value of \( t \), the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.
y The values of \( x \) and \( y \), the dependent variables, to evaluate the derivative at. Should be a numeric vector of length two.
parameters The values of the parameters of the system. Not used here.

Details

example6 evaluates the derivatives of the following coupled ODE system at the point \((t, x, y)\):

\[
\frac{dx}{dt} = x + 2y, \quad \frac{dy}{dt} = -2x + y.
\]

Its format is designed to be compatible with ode from the deSolve package.

Value

Returns a list containing the values of the two derivatives at \((t, x, y)\).

Author(s)

Michael J Grayling

See Also

ode
example7  

Example ODE system 7

Description

The derivative function of an example two-dimensional autonomous ODE system.

Usage

example7(t, y, parameters)

Arguments

t  The value of \( t \), the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.

y  The values of \( x \) and \( y \), the dependent variables, to evaluate the derivative at. Should be a numeric vector of length two.

parameters  The values of the parameters of the system. Not used here.

Details

example7 evaluates the derivatives of the following coupled ODE system at the point \(( t, x, y )\):

\[
\frac{dx}{dt} = -x - y, \quad \frac{dy}{dt} = 4x + y.
\]

Its format is designed to be compatible with ode from the deSolve package.

Value

Returns a list containing the values of the two derivatives at \(( t, x, y )\).

Author(s)

Michael J Grayling

See Also

ode
example8

Example ODE system 8

Description

The derivative function of an example two-dimensional autonomous ODE system.

Usage

gexample8(t, y, parameters)

Arguments

t
The value of t, the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.

y
The values of x and y, the dependent variables, to evaluate the derivative at. Should be a numeric vector of length two.

parameters
The values of the parameters of the system. Not used here.

Details

gexample8 evaluates the derivatives of the following coupled ODE system at the point (t, x, y):

\[
\frac{dx}{dt} = y, \quad \frac{dy}{dt} = -x - y.
\]

Its format is designed to be compatible with ode from the deSolve package.

Value

Returns a list containing the values of the two derivatives at (t, x, y).

Author(s)

Michael J Grayling

See Also

ode
example9

Example ODE system 9

Description

The derivative function of an example two-dimensional autonomous ODE system.

Usage

example9(t, y, parameters)

Arguments

- **t**: The value of \( t \), the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.
- **y**: The values of \( x \) and \( y \), the dependent variables, to evaluate the derivative at. Should be a numeric vector of length two.
- **parameters**: The values of the parameters of the system. Not used here.

Details

example9 evaluates the derivatives of the following coupled ODE system at the point \((t, x, y)\):

\[
\frac{dx}{dt} = -2x + 3y, \quad \frac{dy}{dt} = 7x + 6y.
\]

Its format is designed to be compatible with ode from the deSolve package.

Value

Returns a list containing the values of the two derivatives at \((t, x, y)\).

Author(s)

Michael J Grayling

See Also

ode
Examples

# Plot the velocity field, nullclines and several trajectories
example9_flowField <- flowField(example9,
    xlim = c(-3, 3),
    ylim = c(-3, 3),
    points = 19,
    add = FALSE)
y0 <- matrix(c(1, 0, -3, 2,
    2, -2, -2, -2), 4, 2,
    byrow = TRUE)
example9_nullclines <- nullclines(example9,
    xlim = c(-3, 3),
    ylim = c(-3, 3))
example9_trajectory <- trajectory(example9,
    y0 = y0,
    tlim = c(0, 10))
# Determine the stability of the equilibrium point
example9_stability <- stability(example9,
    ystar = c(0, 0))

exponential

The exponential growth model

Description

The derivative function of the exponential growth model, an example of a one-dimensional autonomous ODE system.

Usage

exponential(t, y, parameters)

Arguments

t
    The value of \( t \), the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.

y
    The value of \( y \), the dependent variable, to evaluate the derivative at. Should be a numeric vector of length one.

parameters
    The values of the parameters of the system. Should be a numeric vector prescribing the value of \( \beta \).

Details

exponential evaluates the derivative of the following ODE at the point \((t, y)\):

\[
\frac{dy}{dt} = \beta y.
\]

Its format is designed to be compatible with \texttt{ode} from the \texttt{deSolve} package.
Value

Returns a list containing the value of the derivative at \((t, y)\).

Author(s)

Michael J Grayling

See Also

ode

Description

Searches for an equilibrium point of a system, taking the starting point of the search as a user specified location. On identifying such a point, a classification is performed, and an informatively shaped point can be added to the plot.

Usage

```r
findEquilibrium(deriv, y0 = NULL, parameters = NULL, 
    system = "two.dim", tol = 1e-16, max.iter = 50, h = 1e-06, 
    plot.it = FALSE, summary = TRUE, state.names = if (system == 
    "two.dim") c("x", "y") else "y")
```

Arguments

- **deriv**: A function computing the derivative at a point for the ODE system to be analysed. Discussion of the required structure of these functions can be found in the package vignette, or in the help file for the function `ode`.
- **y0**: The starting point of the search. In the case of a one-dimensional system, this should be a numeric vector of length one indicating the location of the dependent variable initially. In the case of a two-dimensional system, this should be a numeric vector of length two reflecting the location of the two dependent variables initially. Alternatively this can be specified as NULL, and then locator can be used to specify the initial point on a plot. Defaults to NULL.
- **parameters**: Parameters of the ODE system, to be passed to deriv. Supplied as a numeric vector; the order of the parameters can be found from the deriv file. Defaults to NULL.
- **system**: Set to either "one.dim" or "two.dim" to indicate the type of system being analysed. Defaults to "two.dim".
- **tol**: The tolerance for the convergence of the search algorithm. Defaults to 1e-16.
- **max.iter**: The maximum allowed number of iterations of the search algorithm. Defaults to 50.
**findEquilibrium**

- **h**
  - Step length used to approximate the derivative(s). Defaults to 1e-6.

- **plot.it**
  - Logical. If TRUE, a point is plotted at the identified equilibrium point, with shape corresponding to its classification.

- **summary**
  - Set to either TRUE or FALSE to determine whether a summary of the progress of the search procedure is returned. Defaults to TRUE.

- **state.names**
  - The state names for ode functions that do not use positional states.

**Value**

Returns a list with the following components (the exact make up is dependent on the value of system):

- **classification**
  - The classification of the identified equilibrium point.

- **Delta**
  - In the two-dimensional system case, value of the Jacobian’s determinant at the equilibrium point.

- **deriv**
  - As per input.

- **discriminant**
  - In the one-dimensional system case, the value of the discriminant used in perturbation analysis to assess stability. In the two-dimensional system case, the value of \( tr^2 - 4 \times Delta \).

- **eigenvalues**
  - In the two-dimensional system case, the value of the Jacobian’s eigenvalues at the equilibrium point.

- **eigenvectors**
  - In the two-dimensional system case, the value of the Jacobian’s eigenvectors at the equilibrium point.

- **jacobian**
  - In the two-dimensional system case, the Jacobian at the equilibrium point.

- **h**
  - As per input.

- **max.iter**
  - As per input.

- **parameters**
  - As per input.

- **plot.it**
  - As per input.

- **summary**
  - As per input.

- **system**
  - As per input.

- **tr**
  - In the two-dimensional system case, the value of the Jacobian’s trace at the equilibrium point.

- **tol**
  - As per input.

- **y0**
  - As per input.

- **ystar**
  - The location of the identified equilibrium point.

**Author(s)**

Michael J Grayling, Stephen P Ellner, John M Guckenheimer
flowField

Description

Plots the flow or velocity field for a one- or two-dimensional autonomous ODE system.

Usage

flowField(deriv, xlim, ylim, parameters = NULL, system = "two.dim", points = 21, col = "gray", arrow.type = "equal", arrow.head = 0.05, frac = 1, add = TRUE, state.names = if (system == "two.dim") c("x", "y") else "y", xlab = if (system == "two.dim") state.names[1] else "t", ylab = if (system == "two.dim") state.names[2] else state.names[1], ...)  

Arguments

- **deriv**  
  A function computing the derivative at a point for the ODE system to be analysed. Discussion of the required format of these functions can be found in the package vignette, or in the help file for the function `ode`.

- **xlim**  
  In the case of a two-dimensional system, this sets the limits of the first dependent variable in which gradient reflecting line segments should be plotted. In the case of a one-dimensional system, this sets the limits of the independent variable in which these line segments should be plotted. Should be a numeric vector of length two.

- **ylim**  
  In the case of a two-dimensional system this sets the limits of the second dependent variable in which gradient reflecting line segments should be plotted. In the case of a one-dimensional system, this sets the limits of the dependent variable in which these line segments should be plotted. Should be a numeric vector of length two.

- **parameters**  
  Parameters of the ODE system, to be passed to deriv. Supplied as a numeric vector; the order of the parameters can be found from the deriv file. Defaults to NULL.

- **system**  
  Set to either "one.dim" or "two.dim" to indicate the type of system being analysed. Defaults to "two.dim".

- **points**  
  Sets the density of the line segments to be plotted; points segments will be plotted in the x and y directions. Fine tuning here, by shifting points up and down, allows for the creation of more aesthetically pleasing plots. Defaults to 11.

- **col**  
  Sets the colour of the plotted line segments. Should be a character vector of length one. Will be reset accordingly if it is of the wrong length. Defaults to "gray".
**flowField**

- **arrow.type**: Sets the type of line segments plotted. If set to "proportional" the **length** of the line segments reflects the magnitude of the derivative. If set to "equal" the line segments take equal lengths, simply reflecting the gradient of the derivative(s). Defaults to "equal".

- **arrow.head**: Sets the length of the arrow heads. Passed to **arrows**. Defaults to 0.05.

- **frac**: Sets the fraction of the theoretical maximum length line segments can take without overlapping, that they can actually attain. In practice, **frac** can be set to greater than 1 without line segments overlapping. Fine tuning here assists the creation of aesthetically pleasing plots. Defaults to 1.

- **add**: Logical. If **TRUE**, the flow field is added to an existing plot. If **FALSE**, a new plot is created. Defaults to **TRUE**.

- **state.names**: The state names for **ode** functions that do not use positional states.

- **xlab**: Label for the x-axis of the resulting plot.

- **ylab**: Label for the y-axis of the resulting plot.

- **...**: Additional arguments to be passed to either **plot** or **arrows**.

**Value**

Returns a **list** with the following components (the exact make up is dependent on the value of **system**):

- **add**: As per input.

- **arrow.head**: As per input.

- **arrow.type**: As per input.

- **col**: As per input, but with possible editing if a **character vector** of the wrong **length** was supplied.

- **deriv**: As per input.

- **dx**: A **numeric matrix**. In the case of a two-dimensional system, the values of the derivative of the first dependent derivative at all evaluated points.

- **dy**: A **numeric matrix**. In the case of a two-dimensional system, the values of the derivative of the second dependent variable at all evaluated points. In the case of a one-dimensional system, the values of the derivative of the dependent variable at all evaluated points.

- **frac**: As per input.

- **parameters**: As per input.

- **points**: As per input.

- **system**: As per input.

- **x**: A **numeric vector**. In the case of a two-dimensional system, the values of the first dependent variable at which the derivatives were computed. In the case of a one-dimensional system, the values of the independent variable at which the derivatives were computed.

- **xlab**: As per input.

- **xlim**: As per input.
flowField

y

A numeric vector. In the case of a two-dimensional system, the values of the second dependent variable at which the derivatives were computed. In the case of a one-dimensional system, the values of the dependent variable at which the derivatives were computed.

ylab

As per input.

ylim

As per input.

Author(s)

Michael J Grayling

See Also

arrows, plot

Examples

# Plot the flow field, nullclines and several trajectories for the # one-dimensional autonomous ODE system logistic
logistic_flowField <- flowField(logistic,
    xlim = c(0, 5),
    ylim = c(-1, 3),
    parameters = c(1, 2),
    points = 21,
    system = "one.dim",
    add = FALSE)

logistic_nullclines <- nullclines(logistic,
    xlim = c(0, 5),
    ylim = c(-1, 3),
    parameters = c(1, 2),
    system = "one.dim")

logistic_trajectory <- trajectory(logistic,
    y0 = c(-0.5, 0.5, 1.5, 2.5),
    tlim = c(0, 5),
    parameters = c(1, 2),
    system = "one.dim")

# Plot the velocity field, nullclines and several trajectories for the # two-dimensional autonomous ODE system simplePendulum
simplePendulum_flowField <- flowField(simplePendulum,
    xlim = c(-7, 7),
    ylim = c(-7, 7),
    parameters = 5,
    points = 19,
    add = FALSE)

y0 <- matrix(c(0, 1, 0, 4, -6, 1, 5, 0.5, 0, -3),
    5, 2, byrow = TRUE)

simplePendulum_nullclines <- nullclines(simplePendulum,
    xlim = c(-7, 7),
    ylim = c(-7, 7),
lindemannMechanism

parameters = 5,
points = 500)

simplePendulum_trajectory <- trajectory(simplePendulum,
y0 = y0,
tlim = c(0, 10),
parameters = 5)

---

lindemannMechanism  The Lindemann mechanism

Description
The derivative function of the non-dimensional version of the Lindemann mechanism, an example of a two-dimensional autonomous ODE system.

Usage
lindemannMechanism(t, y, parameters)

Arguments
t  The value of t, the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.
y  The values of x and y, the dependent variables, to evaluate the derivative at. Should be a numeric vector of length two.
parameters  The values of the parameters of the system. Should be a numeric vector prescribing the value of α.

Details
lindemannMechanism evaluates the derivative of the following ODE at the point (t, x, y):

\[
\frac{dx}{dt} = -x^2 + \alpha xy, \quad \frac{dy}{dt} = x^2 - \alpha xy - y.
\]

Its format is designed to be compatible with ode from the deSolve package.

Value
Returns a list containing the values of the two derivatives at (t, x, y).

Author(s)
Michael J Grayling

See Also
ode
logistic

The logistic growth model

Description

The derivative function of the logistic growth model, an example of a two-dimensional autonomous ODE system.

Usage

logistic(t, y, parameters)

Arguments

t
The value of t, the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.

y
The value of y, the dependent variable, to evaluate the derivative at. Should be a numeric vector of length one.

parameters
The values of the parameters of the system. Should be a numeric vector with parameters specified in the following order: β, K.

Details

logistic evaluates the derivative of the following ODE at the point (t, y):

$$ \frac{dy}{dt} = \beta y(1 - y/K). $$

Its format is designed to be compatible with ode from the deSolve package.

Value

Returns a list containing the value of the derivative at (t, y).

Author(s)

Michael J Grayling

See Also

ode
Examples

# Plot the velocity field, nullclines and several trajectories
logistic_flowField <- flowField(logistic,
                               xlim = c(0, 5),
                               ylim = c(-1, 3),
                               parameters = c(1, 2),
                               points = 21,
                               system = "one.dim",
                               add = FALSE)

logistic_nullclines <- nullclines(logistic,
                                   xlim = c(0, 5),
                                   ylim = c(-1, 3),
                                   parameters = c(1, 2),
                                   system = "one.dim")

logistic_trajectory <- trajectory(logistic,
                                    y0 = c(-0.5, 0.5, 1.5, 2.5),
                                    tlim = c(0, 5),
                                    parameters = c(1, 2),
                                    system = "one.dim")

# Plot the phase portrait
logistic_phasePortrait <- phasePortrait(logistic,
                                       ylim = c(-0.5, 2.5),
                                       parameters = c(1, 2),
                                       points = 10,
                                       frac = 0.5)

# Determine the stability of the equilibrium points
logistic_stability_1 <- stability(logistic,
                                 ystar = 0,
                                 parameters = c(1, 2),
                                 system = "one.dim")

logistic_stability_2 <- stability(logistic,
                                 ystar = 2,
                                 parameters = c(1, 2),
                                 system = "one.dim")

lotkaVolterra

The Lotka-Volterra model

Description

The derivative function of the Lotka-Volterra model, an example of a two-dimensional autonomous
ODE system.

Usage

lotkaVolterra(t, y, parameters)
monomolecular

Arguments

\( t \)

The value of \( t \), the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.

\( y \)

The values of \( x \) and \( y \), the dependent variables, to evaluate the derivative at. Should be a numeric vector of length two.

parameters

The values of the parameters of the system. Should be a numeric vector with parameters specified in the following order: \( \lambda, \epsilon, \eta, \delta \).

Details

\texttt{lotkaVolterra} evaluates the derivative of the following ODE at the point \((t,x,y)\):

\[
\frac{dx}{dt} = \lambda x - \epsilon xy, \quad \frac{dy}{dt} = \eta xy - \delta y.
\]

Its format is designed to be compatible with \texttt{ode} from the \texttt{deSolve} package.

Value

Returns a \texttt{list} containing the values of the two derivatives at \((t,x,y)\).

Author(s)

Michael J Grayling

See Also

\texttt{ode}

---

monomolecular

The monomolecular growth model

Description

The derivative function of the monomolecular growth model, an example of a one-dimensional autonomous ODE system.

Usage

\texttt{monomolecular}(t, y, parameters)

Arguments

\( t \)

The value of \( t \), the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.

\( y \)

The value of \( y \), the dependent variable, to evaluate the derivative at. Should be a numeric vector of length one.

parameters

The values of the parameters of the system. Should be a numeric vector with parameters specified in the following order: \( \beta, K \).
Details

monomolecular evaluates the derivative of the following ODE at the point \((t, y)\):

\[
\frac{dy}{dt} = \beta(K - y).
\]

Its format is designed to be compatible with \texttt{ode} from the \texttt{deSolve} package.

Value

Returns a \texttt{list} containing the value of the derivative at \((t, y)\).

Author(s)

Michael J Grayling

See Also

\texttt{ode}

---

\texttt{morrisLecar}  \hspace{1cm} \textit{The Morris-Lecar model}

Description

The derivative function of the Morris-Lecar model, an example of a two-dimensional autonomous ODE system.

Usage

\texttt{morrisLecar(t, y, parameters)}

Arguments

\begin{itemize}
  \item \texttt{t} \hspace{1cm} The value of \(t\), the independent variable, to evaluate the derivative at. Should be a \texttt{numeric vector} of \texttt{length} one.
  \item \texttt{y} \hspace{1cm} The values of \(x\) and \(y\), the dependent variables, to evaluate the derivative at. Should be a \texttt{numeric vector} of \texttt{length} two.
  \item \texttt{parameters} \hspace{1cm} The values of the parameters of the system. Should be a \texttt{numeric vector} with parameters specified in the following order: \(g_{Ca}, \phi\).
\end{itemize}

Details

\texttt{morrisLecar} evaluates the derivative of the following ODE at the point \((t, x, y)\):

\[
\frac{dx}{dt} = 0.05(90 - 0.5g_{Ca}(1 + \tanh(x + 1.2)/18))(x - 120) - 8y(x + 84) - 2(x + 60),
\]

\[
\frac{dy}{dt} = \phi(0.5 \left[ 1 + \tanh \left( \frac{x - 2}{30} \right) \right] - y) \cosh \left( \frac{x - 2}{60} \right).
\]

Its format is designed to be compatible with \texttt{ode} from the \texttt{deSolve} package.
nullclines

Value
Returns a list containing the values of the two derivatives at \((t, x, y)\).

Author(s)
Michael J Grayling

See Also
ode

nullclines

Description
Plots nullclines for two-dimensional autonomous ODE systems. Can also be used to plot horizontal lines at equilibrium points for one-dimensional autonomous ODE systems.

Usage
```
nullclines(deriv, xlim, ylim, parameters = NULL, system = "two.dim",
            points = 101, col = c("blue", "cyan"), add = TRUE,
            add.legend = TRUE, state.names = if (system == "two.dim") c("x", "y")
                               else "y", ...)
```

Arguments
- `deriv`: A function computing the derivative at a point for the ODE system to be analysed. Discussion of the required structure of these functions can be found in the package vignette, or in the help file for the function `ode`.
- `xlim`: In the case of a two-dimensional system, this sets the limits of the first dependent variable in which gradient reflecting line segments should be plotted. In the case of a one-dimensional system, this sets the limits of the independent variable in which these line segments should be plotted. Should be a numeric vector of length two.
- `ylim`: In the case of a two-dimensional system this sets the limits of the second dependent variable in which gradient reflecting line segments should be plotted. In the case of a one-dimensional system, this sets the limits of the dependent variable in which these line segments should be plotted. Should be a numeric vector of length two.
- `parameters`: Parameters of the ODE system, to be passed to `deriv`. Supplied as a numeric vector; the order of the parameters can be found from the `deriv` file. Defaults to `NULL`.
- `system`: Set to either "one.dim" or "two.dim" to indicate the type of system being analysed. Defaults to "two.dim".
nullclines

points  Sets the density at which derivatives are computed; points x points derivatives will be computed. Levels of zero gradient are identified using these computations and the function contour. Increasing the value of points improves identification of nullclines, but increases computation time. Defaults to 101.

col  In the case of a two-dimensional system, sets the colours used for the x- and y-nullclines. In the case of a one-dimensional system, sets the colour of the lines plotted horizontally along the equilibria. Should be a character vector of length two. Will be reset accordingly if it is of the wrong length. Defaults to c(“blue”, “cyan”).

add  Logical. If TRUE, the nullclines are added to an existing plot. If FALSE, a new plot is created. Defaults to TRUE.

add.legend  Logical. If TRUE, a legend is added to the plots. Defaults to TRUE.

state.names  The state names for ode functions that do not use positional states.

...  Additional arguments to be passed to either plot or contour.

Value

Returns a list with the following components (the exact make up is dependent on the value of system):

add  As per input.

add.legend  As per input.

col  As per input, but with possible editing if a character vector of the wrong length was supplied.

deriv  As per input.

dx  A numeric matrix. In the case of a two-dimensional system, the values of the derivative of the first dependent derivative at all evaluated points.

dy  A numeric matrix. In the case of a two-dimensional system, the values of the derivative of the second dependent variable at all evaluated points. In the case of a one-dimensional system, the values of the derivative of the dependent variable at all evaluated points.

parameters  As per input.

points  As per input.

system  As per input.

x  A numeric vector. In the case of a two-dimensional system, the values of the first dependent variable at which the derivatives were computed. In the case of a one-dimensional system, the values of the independent variable at which the derivatives were computed.

xlim  As per input.

y  A numeric vector. In the case of a two-dimensional system, the values of the second dependent variable at which the derivatives were computed. In the case of a one-dimensional system, the values of the dependent variable at which the derivatives were computed.

ylim  As per input.
Note

In order to ensure a nullcline is plotted, set xlim and ylim strictly enclosing its location. For example, to ensure a nullcline is plotted along x = 0, set ylim to, e.g., begin at -1.

Author(s)

Michael J Grayling

See Also

contour, plot

Examples

# Plot the flow field, nullclines and several trajectories for the
# one-dimensional autonomous ODE system logistic.
logistic_flowField <- flowField(logistic,
    xlim = c(0, 5),
    ylim = c(-1, 3),
    parameters = c(1, 2),
    points = 21,
    system = "one.dim",
    add = FALSE)

logistic_nullclines <- nullclines(logistic,
    xlim = c(0, 5),
    ylim = c(-1, 3),
    parameters = c(1, 2),
    system = "one.dim")

logistic_trajectory <- trajectory(logistic,
    y0 = c(-0.5, 0.5, 1.5, 2.5),
    tlim = c(0, 5),
    parameters = c(1, 2),
    system = "one.dim")

# Plot the velocity field, nullclines and several trajectories for the
# two-dimensional autonomous ODE system simplePendulum.
simplePendulum_flowField <- flowField(simplePendulum,
    xlim = c(-7, 7),
    ylim = c(-7, 7),
    parameters = 5,
    points = 19,
    add = FALSE)

y0 <- matrix(c(0, 1, 0, 4, -6, 1, 5, 0.5, 0, -3),
    5, 2, byrow = TRUE)
simplePendulum_nullclines <- nullclines(simplePendulum,
    xlim = c(-7, 7),
    ylim = c(-7, 7),
    parameters = 5,
    points = 500)
simplePendulum_trajectory <- trajectory(simplePendulum,
Numerically solves a two-dimensional autonomous ODE system for a given initial condition, using `ode` from the package `deSolve`. It then plots the dependent variables against the independent variable.

### Usage

```r
numericalSolution(deriv, y0 = NULL, tlim, tstep = 0.01, parameters = NULL, type = "one", col = c("red", "blue"), add.grid = TRUE, add.legend = TRUE, state.names = c("x", "y"), xlab = "t", ylab = state.names, ...)
```

### Arguments

- **deriv**: A function computing the derivative at a point for the ODE system to be analysed. Discussion of the required structure of these functions can be found in the package vignette, or in the help file for the function `ode`.

- **y0**: The initial condition. Should be a numeric vector of length two reflecting the location of the two dependent variables initially.

- **tlim**: Sets the limits of the independent variable for which the solution should be plotted. Should be a numeric vector of length two. If `tlim[2] > tlim[1]`, then `tstep` should be negative to indicate a backwards trajectory.

- **tstep**: The step length of the independent variable, used in numerical integration. Decreasing the absolute magnitude of `tstep` theoretically makes the numerical integration more accurate, but increases computation time. Defaults to `0.01`.

- **parameters**: Parameters of the ODE system, to be passed to `deriv`. Supplied as a numeric vector; the order of the parameters can be found from the `deriv` file. Defaults to `NULL`.

- **type**: If set to "one" the trajectories are plotted on the same graph. If set to "two" they are plotted on separate graphs. Defaults to "one".

- **col**: Sets the colours of the trajectories of the two dependent variables. Should be a character vector of length two. Will be reset accordingly if it is of the wrong length. Defaults to c("red","blue").

- **add.grid**: Logical. If TRUE, grids are added to the plots. Defaults to TRUE.

- **add.legend**: Logical. If TRUE, a legend is added to the plots. Defaults to TRUE.

- **state.names**: The state names for `ode` functions that do not use positional states.

- **xlab**: Label for the x-axis of the resulting plot.

- **ylab**: Label for the y-axis of the resulting plot.

- **...**: Additional arguments to be passed to `plot`.

---

**Description**

Numerically solves a two-dimensional autonomous ODE system for a given initial condition, using `ode` from the package `deSolve`. It then plots the dependent variables against the independent variable.
phasePlaneAnalysis

Value

Returns a list with the following components:

- add.grid: As per input.
- add.legend: As per input.
- col: As per input, but with possible editing if a character vector of the wrong length was supplied.
- deriv: As per input.
- parameters: As per input.
- t: A numeric vector containing the values of the independent variable at each integration step.
- tlim: As per input.
- tstep: As per input.
- x: A numeric vector containing the numerically computed values of the first dependent variable at each integration step.
- y: A numeric vector containing the numerically computed values of the second dependent variable at each integration step.
- y0: As per input.

Author(s)

Michael J Grayling

See Also

ode, plot

Examples

# A two-dimensional autonomous ODE system, vanDerPol.
vanDerPol_numericalSolution <- numericalSolution(vanDerPol, 
  y0 = c(4, 2),
  tlim = c(0, 100),
  parameters = 3)

phasePlaneAnalysis  Phase plane analysis

Description

Allows the user to perform a basic phase plane analysis and produce a simple plot without the need to use the other functions directly. Specifically, a range of options are provided and the user inputs a value to the console to decide what is added to the plot.
phasePlaneAnalysis(deriv, xlim, ylim, tend = 100, parameters = NULL,
    system = "two.dim", add = FALSE, state.names = if (system ==
    "two.dim") c("x", "y") else "y")

Arguments

deriv       A function computing the derivative at a point for the ODE system to be anal-
yzed. Discussion of the required structure of these functions can be found in the
package vignette, or in the help file for the function ode.

xlim        In the case of a two-dimensional system, this sets the limits of the first dependent
variable in any subsequent plot. In the case of a one-dimensional system, this
sets the limits of the independent variable. Should be a numeric vector of
length two.

ylim        In the case of a two-dimensional system this sets the limits of the second depen-
dent variable in any subsequent plot. In the case of a one-dimensional system,
this sets the limits of the dependent variable. Should be a numeric vector of
length two.

tend        The value of the independent variable to end any subsequent numerical integra-
tions at.

parameters  Parameters of the ODE system, to be passed to deriv. Supplied as a numeric
vector; the order of the parameters can be found from the deriv file. Defaults
to NULL.

system      Set to either "one.dim" or "two.dim" to indicate the type of system being anal-
yzed. Defaults to "two.dim".

add         Logical. If TRUE, the chosen features are added to an existing plot. If FALSE, a
new plot is created. Defaults to FALSE.

state.names The state names for ode functions that do not use positional states.

Details

The user designates the derivative file and other arguments as per the above. Then the following ten
options are available for execution:

- 1. Flow field: Plots the flow field of the system. See flowField.
- 2. Nullclines: Plots the nullclines of the system. See nullclines.
- 3. Find fixed point (click on plot): Searches for an equilibrium point of the system, taking the
starting point of the search as where the user clicks on the plot. See findEquilibrium.
- 4. Start forward trajectory (click on plot): Plots a trajectory, i.e., a solution, forward in time
with the starting point taken as where the user clicks on the plot. See trajectory.
- 5. Start backward trajectory (click on plot): Plots a trajectory, i.e., a solution, backward in
time with the starting point taken as where the user clicks on the plot. See trajectory.
- 6. Extend Current trajectory (a trajectory must already be plotted): Extends already plotted
trajectories further on in time. See trajectory.
7. Local stable/unstable manifolds of a saddle (two-dimensional systems only) (click on plot): Plots the stable and unstable manifolds of a saddle point. The user clicks on the plot and an equilibrium point is identified see (3) above, if this point is a saddle then the manifolds are plotted. See drawManifolds.

8. Grid of trajectories: Plots a set of trajectories, with the starting points defined on an equally spaced grid over the designated plotting range for the dependent variable(s). See trajectory.

9. Exit: Exits the current call to phasePlaneAnalysis().

10. Save plot as PDF: Saves the produced plot as "phasePlaneAnalysis.pdf" in the current working directory.

Author(s)
Michael J Grayling, Stephen P Ellner, John M Guckenheimer

---

phasePortrait  

Phase portrait plot

---

Description
For a one-dimensional autonomous ODE, it plots the phase portrait, i.e., the derivative against the dependent variable. In addition, along the dependent variable axis it plots arrows pointing in the direction of dependent variable change with increasing value of the independent variable. From this stability of equilibrium points (i.e., locations where the horizontal axis is crossed) can be determined.

Usage

```r
phasePortrait(deriv, ylim, ystep = 0.01, parameters = NULL,
             points = 10, frac = 0.75, arrow.head = 0.075, col = "black",
             add.grid = TRUE, state.names = "y", xlab = state.names,
             ylab = paste0("d", state.names), 
             ...
```

Arguments

- `deriv` A function computing the derivative at a point for the ODE system to be analysed. Discussion of the required structure of these functions can be found in the package vignette, or in the help file for the function `ode`.
- `ylim` Sets the limits of the dependent variable for which the derivative should be computed and plotted. Should be a numeric vector of length two.
- `ystep` Sets the step length of the dependent variable vector for which derivatives are computed and plotted. Decreasing ystep makes the resulting plot more accurate, but comes at a small cost to computation time. Defaults to 0.01.
- `parameters` Parameters of the ODE system, to be passed to deriv. Supplied as a numeric vector; the order of the parameters can be found from the deriv file. Defaults to NULL.
points  Sets the density at which arrows are plotted along the horizontal axis; points arrows will be plotted. Fine tuning here, by shifting points up and down, allows for the creation of more aesthetically pleasing plots. Defaults to 10.

frac  Sets the fraction of the theoretical maximum length line segments can take without overlapping, that they actually attain. Fine tuning here assists the creation of aesthetically pleasing plots. Defaults to 0.75.

arrow.head  Sets the length of the arrow heads. Passed to arrows. Defaults to 0.075.

col  Sets the colour of the line in the plot, as well as the arrows. Should be a character vector of length one. Will be reset accordingly if it is of the wrong length. Defaults to "black".

add.grid  Logical. If TRUE, a grid is added to the plot. Defaults to TRUE.

state.names  The state names for ode functions that do not use positional states.

xlab  Label for the x-axis of the resulting plot.

ylab  Label for the y-axis of the resulting plot.

...  Additional arguments to be passed to either plot or arrows.

Value

Returns a list with the following components:

add.grid  As per input.

arrow.head  As per input.

col  As per input, but with possible editing if a character vector of the wrong length was supplied.

deriv  As per input.

dy  A numeric vector containing the value of the derivative at each evaluated point.

frac  As per input.

parameters  As per input.

points  As per input.

xlab  As per input.

y  A numeric vector containing the values of the dependent variable for which the derivative was evaluated.

ylab  As per input.

ylim  As per input.

ystep  As per input.

Author(s)

Michael J Grayling

See Also

arrows, plot
Examples

# A one-dimensional autonomous ODE system, example2.
example2_phasePortrait <- phasePortrait(example2,
    ylim = c(-0.5, 2.5),
    points = 10,
    frac = 0.5)

simplePendulum

The simple pendulum model

Description

The derivative function of the simple pendulum model, an example of a two-dimensional autonomous ODE system.

Usage

simplePendulum(t, y, parameters)

Arguments

t The value of \( t \), the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.

y The values of \( x \) and \( y \), the dependent variables, to evaluate the derivative at. Should be a numeric vector of length two.

parameters The values of the parameters of the system. Should be a numeric vector prescribing the value of \( l \).

Details

\[
\frac{dx}{dt} = y, \quad \frac{dy}{dt} = -g \sin(x) / l.
\]

Its format is designed to be compatible with \texttt{ode} from the \texttt{deSolve} package.

Value

Returns a list containing the values of the two derivatives at \((t, x, y)\).

Author(s)

Michael J Grayling

See Also

\texttt{ode}
Examples

```r
# Plot the velocity field, nullclines and several trajectories
simplePendulum_flowField <- flowField(simplePendulum,
    xlim = c(-7, 7),
    ylim = c(-7, 7),
    parameters = 5,
    points = 19,
    add = FALSE)

y0 <- matrix(c(0, 1, 0, 4, -6,
    1, 5, 0.5, 0, -3), 5, 2,
    byrow = TRUE)

simplePendulum_nullclines <- nullclines(simplePendulum,
    xlim = c(-7, 7),
    ylim = c(-7, 7),
    parameters = 5,
    points = 500)

simplePendulum_trajectory <- trajectory(simplePendulum,
    y0 = y0,
    tlim = c(0, 10),
    parameters = 5)

# Determine the stability of two equilibrium points
simplePendulum_stability_1 <- stability(simplePendulum,
    ystar = c(0, 0),
    parameters = 5)

simplePendulum_stability_2 <- stability(simplePendulum,
    ystar = c(pi, 0),
    parameters = 5)
```

---

The SIR epidemic model

**Description**

The derivative function of the SIR epidemic model, an example of a two-dimensional autonomous ODE system.

**Usage**

```r
SIR(t, y, parameters)
```

**Arguments**

- `t` The value of `t`, the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.
- `y` The values of `x` and `y`, the dependent variables, to evaluate the derivative at. Should be a numeric vector of length two.
- `parameters` The values of the parameters of the system. Should be a numeric vector with parameters specified in the following order: `β, ν`. 
Details

SIR evaluates the derivative of the following ODE at the point \((t, x, y)\):

\[
\frac{dx}{dt} = -\beta xy, \quad \frac{dy}{dt} = \beta xy - \nu y.
\]

Its format is designed to be compatible with \texttt{ode} from the \texttt{deSolve} package.

Value

Returns a \texttt{list} containing the values of the two derivatives at \((t, x, y)\).

Author(s)

Michael J Grayling

See Also

\texttt{ode}

---

**stability**  
*Stability analysis*

Description

Uses stability analysis to classify equilibrium points. Uses the Taylor Series approach (also known as perturbation analysis) to classify equilibrium points of a one-dimensional autonomous ODE system, or the Jacobian approach to classify equilibrium points of a two-dimensional autonomous ODE system. In addition, it can be used to return the Jacobian at any point of a two-dimensional system.

Usage

\[
\text{stability(deriv, ystar = NULL, parameters = NULL, system = "two.dim", h = 1e-07, summary = TRUE, state.names = if (system == "twodim") c("x", "y") else "y")}
\]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deriv</td>
<td>A function computing the derivative at a point for the ODE system to be analysed. Discussion of the required structure of these functions can be found in the package vignette, or in the help file for the function \texttt{ode}.</td>
</tr>
<tr>
<td>ystar</td>
<td>The point at which to perform stability analysis. For a one-dimensional system this should be a \texttt{numeric vector} of \texttt{length} one, for a two-dimensional system this should be a \texttt{numeric vector} of \texttt{length} two (i.e., presently only one equilibrium point’s stability can be evaluated at a time). Alternatively this can be specified as \texttt{NULL}, and then \texttt{locator} can be used to choose a point to perform the analysis for. However, given you are unlikely to locate exactly the equilibrium point, if possible enter ystar yourself. Defaults to \texttt{NULL}.</td>
</tr>
</tbody>
</table>
parameters Parameters of the ODE system, to be passed to deriv. Supplied as a numeric vector; the order of the parameters can be found from the deriv file. Defaults to NULL.

system Set to either "one.dim" or "two.dim" to indicate the type of system being analysed. Defaults to "two.dim".

h Step length used to approximate the derivative(s). Defaults to 1e-7.

summary Set to either TRUE or FALSE to determine whether a summary of the stability analysis is returned. Defaults to TRUE.

state.names The state names for ode functions that do not use positional states.

Value

Returns a list with the following components (the exact make up is dependent upon the value of system):

classification The classification of ystar.

Delta In the two-dimensional system case, the value of the Jacobian’s determinant at ystar.

deriv As per input.

discriminant In the one-dimensional system case, the value of the discriminant used in perturbation analysis to assess stability. In the two-dimensional system case, the value of tr^2 -4*Delta.

eigenvalues In the two-dimensional system case, the value of the Jacobian’s eigenvalues at ystar.

eigenvectors In the two-dimensional system case, the value of the Jacobian’s eigenvectors at ystar.

jacobian In the two-dimensional system case, the Jacobian at ystar.

h As per input.

parameters As per input.

summary As per input.

system As per input.

tr In the two-dimensional system case, the value of the Jacobian’s trace at ystar.

ystar As per input.

Author(s)

Michael J Grayling

Examples

# Determine the stability of the equilibrium points of the one-dimensional autonomous ODE system example2
example2_stability_1 <- stability(example2, ystar = 0, system = "one.dim")
example2_stability_2 <- stability(example2, ystar = 1, system = "one.dim")
example2_stability_3 <- stability(example2, ystar = 2, system = "one.dim")
# Determine the stability of the equilibrium points of the two-dimensional
# autonomous ODE system example11
example11_stability_1 <- stability(example11, ystar = c(0, 0))
example11_stability_2 <- stability(example11, ystar = c(0, 2))
example11_stability_3 <- stability(example11, ystar = c(1, 1))
example11_stability_4 <- stability(example11, ystar = c(3, 0))

toggle (The genetic toggle switch model)

Description

The derivative function of a simple genetic toggle switch model, an example of a two-dimensional autonomous ODE system.

Usage

toggle(t, y, parameters)

Arguments

t The value of t, the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.
y The values of x and y, the dependent variables, to evaluate the derivative at. Should be a numeric vector of length two.
parameters The values of the parameters of the system. Should be a numeric vector with parameters specified in the following order: \(\alpha, \beta, \gamma\).

Details

toggle evaluates the derivative of the following ODE at the point \((t, x, y)\):

\[
\frac{dx}{dt} = -x + \alpha (1 + y^\beta), \quad \frac{dy}{dt} = -y + \alpha (1 + x^\gamma).
\]

Its format is designed to be compatible with ode from the deSolve package.

Value

Returns a list containing the values of the two derivatives at \((t, x, y)\).

Author(s)

Michael J Grayling

See Also

ode
trajectory

Phase plane trajectory plotting

Description

Performs numerical integration of the chosen ODE system, for a user specified set of initial conditions. Plots the resulting solution(s) in the phase plane.

Usage

trajectory(deriv, y0 = NULL, n = NULL, tlim, tstep = 0.01,
parameters = NULL, system = "two.dim", col = "black", add = TRUE,
state.names = if (system == "two.dim") c("x", "y") else "y", ...)

Arguments

deriv A function computing the derivative at a point for the ODE system to be analysed. Discussion of the required structure of these functions can be found in the package vignette, or in the help file for the function ode.

y0 The initial condition(s). In the case of a one-dimensional system, this can either be a numeric vector of length one, indicating the location of the dependent variable initially, or a numeric vector indicating multiple initial locations of the independent variable. In the case of a two-dimensional system, this can either be a numeric vector of length two, reflecting the location of the two dependent variables initially, or it can be numeric matrix where each row reflects a different initial condition. Alternatively this can be specified as NULL, and then locator can be used to specify initial condition(s) on a plot. In this case, for one-dimensional systems, all initial conditions are taken at tlim[1], even if not selected so on the graph. Defaults to NULL.

n If y0 is left NULL, such initial conditions can be specified using locator, n sets the number of initial conditions to be chosen. Defaults to NULL.

tlim Sets the limits of the independent variable for which the solution should be plotted. Should be a numeric vector of length two. If tlim[2] > tlim[1], then tstep should be negative to indicate a backwards trajectory.

tstep The step length of the independent variable, used in numerical integration. Decreasing the absolute magnitude of tstep theoretically makes the numerical integration more accurate, but increases computation time. Defaults to 0.01.

parameters Parameters of the ODE system, to be passed to deriv. Supplied as a numeric vector; the order of the parameters can be found from the deriv file. Defaults to NULL.

system Set to either "one.dim" or "two.dim" to indicate the type of system being analysed. Defaults to "two.dim".

col The colour(s) to plot the trajectories in. Should be a character vector. Will be reset accordingly if it is not of the length of the number of initial conditions. Defaults to "black".
add Logical. If TRUE, the trajectories added to an existing plot. If FALSE, a new plot is created. Defaults to TRUE.

state.names The state names for ode functions that do not use positional states.

... Additional arguments to be passed to plot.

Value

Returns a list with the following components (the exact make up is dependent on the value of system):

- **add** As per input.
- **col** As per input, but with possible editing if a character vector of the wrong length was supplied.
- **deriv** As per input.
- **n** As per input.
- **parameters** As per input.
- **system** As per input.
- **tlim** As per input.
- **tstep** As per input.
- **t** A numeric vector containing the values of the independent variable at each integration step.
- **x** In the two-dimensional system case, a numeric matrix whose columns are the numerically computed values of the first dependent variable for each initial condition.
- **y** In the two-dimensional system case, a numeric matrix whose columns are the numerically computed values of the second dependent variable for each initial condition. In the one-dimensional system case, a numeric matrix whose columns are the numerically computed values of the dependent variable for each initial condition.
- **y0** As per input, but converted to a numeric matrix if supplied as a vector initially.

Author(s)

Michael J Grayling

See Also

ode, plot

Examples

```r
# Plot the flow field, nullclines and several trajectories for the
# one-dimensional autonomous ODE system logistic
logistic_flowField <- flowField(logistic,
   xlim = c(0, 5),
   ylim = c(-1, 3),
```

```r
```
vanDerPol

The Van der Pol oscillator

Description

The derivative function of the Van der Pol Oscillator, an example of a two-dimensional autonomous ODE system.

Usage

vanDerPol(t, y, parameters)
Arguments
- **t**
  The value of \( t \), the independent variable, to evaluate the derivative at. Should be a **numeric vector** of **length** one.

- **y**
  The values of \( x \) and \( y \), the dependent variables, to evaluate the derivative at. Should be a **numeric vector** of **length** two.

- **parameters**
  The values of the parameters of the system. Should be a **numeric vector** prescribing the value of \( \mu \).

Details

`vanDerPol` evaluates the derivative of the following ODE at the point \((t, x, y)\):

\[
\frac{dx}{dt} = y, \quad \frac{dy}{dt} = \mu(1 - x^2)y - x.
\]

Its format is designed to be compatible with `ode` from the `deSolve` package.

Value

Returns a **list** containing the values of the two derivatives at \((t, x, y)\).

Author(s)

Michael J Grayling

See Also

`ode`

Examples

```r
# Plot the velocity field, nullclines and several trajectories.
vanDerPol_flowField <- flowField(vanDerPol,
   xlim = c(-5, 5),
   ylim = c(-5, 5),
   parameters = 3,
   points = 15,
   add = FALSE)

y0 <- matrix(c(2, 0, 0, 2, 0.5, 0.5), 3, 2,
   byrow = TRUE)

vanDerPol_nullclines <- nullclines(vanDerPol,
   xlim = c(-5, 5),
   ylim = c(-5, 5),
   parameters = 3,
   points = 500)

vanDerPol_trajectory <- trajectory(vanDerPol,
   y0 = y0,
   tlim = c(0, 10),
   parameters = 3)
```
# Plot x and y against t
vanDerPol_numericalSolution <- numericalSolution(vanDerPol,
y0 = c(4, 2),
tlim = c(0, 100),
parameters = 3)

# Determine the stability of the equilibrium point
vanDerPol_stability <- stability(vanDerPol,
ystar = c(0, 0),
parameters = 3)

---

The von Bertalanffy growth model

**Description**

The derivative function of the von Bertalanffy growth model, an example of a one-dimensional autonomous ODE system.

**Usage**

```
vonBertalanffy(t, y, parameters)
```

**Arguments**

- `t` The value of `t`, the independent variable, to evaluate the derivative at. Should be a numeric vector of length one.
- `y` The value of `y`, the dependent variable, to evaluate the derivative at. Should be a numeric vector of length one.
- `parameters` The values of the parameters of the system. Should be a numeric vector with parameters specified in the following order: `α, β`.

**Details**

`vonBertalanffy` evaluates the derivative of the following ODE at the point `(t, y)`:\[
\frac{dy}{dt} = \alpha y^{2/3} - \beta y.
\]

Its format is designed to be compatible with `ode` from the `deSolve` package.

**Value**

Returns a list containing the values of the two derivatives at `(t, x, y)`.

**Author(s)**

Michael J Grayling
See Also

ode
Index

.paramDummy, 4
arrows, 27, 28, 41
character, 6, 26, 27, 35, 37, 38, 41, 47, 48
competition, 4
contour, 35, 36
desolve, 4, 5, 7–10, 12–14, 16, 18–23, 29, 30, 32, 33, 37, 42, 44, 46, 50, 51
drawManifolds, 3, 5, 40
example1, 7
t example10, 8
t example11, 9
t example12, 10
t example13, 11
t example14, 12
t example15, 13
t example2, 14
t example3, 15
t example4, 16
t example5, 17
t example6, 19
t example7, 20
t example8, 21
t example9, 22
exponential, 23
findEquilibrium, 3, 24, 39
flowField, 3, 26, 39

legend, 35, 37
lindemannMechanism, 29
list, 5–9, 11–14, 16–22, 24, 27, 29, 30, 32–35, 38, 42, 44–46, 50, 51
locator, 5, 24, 44, 47
logistic, 30
lotkaVolterra, 31

matrix, 6, 27, 35, 47, 48
monomolecular, 32
morrisLecar, 33
null, 5, 24, 44, 47
nullclines, 3, 34, 39
numericalSolution, 3, 37
ode, 4–27, 29, 30, 32–35, 37–42, 44–48, 50–52
phasesPlaneAnalysis, 3, 38
phasesPortrait, 3, 40
phasesR (phasesR-package), 3
phasesR-package, 3
plot, 27, 28, 35–38, 41, 48

simplePendulum, 42
SIR, 43
stability, 3, 44
toggle, 46
trajectory, 3, 39, 40, 47
vanDerPol, 49
vonBertalanffy, 51