Package 'ecode'

July 2, 2024

Title Ordinary Differential Equation Systems in Ecology

Version 0.1.0

Description
A framework to simulate ecosystem dynamics through ordinary differential equations (ODEs). You create an ODE model, tells 'ecode' to explore its behaviour, and perform numerical simulations on the model. 'ecode' also allows you to fit model parameters by machine learning algorithms. Potential users include researchers who are interested in the dynamics of ecological community and biogeochemical cycles.

Imports ggplot2, cowplot, rlang, stringr

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

BugReports https://github.com/HaoranPopEvo/ecode/issues

URL https://github.com/HaoranPopEvo/ecode

NeedsCompilation no

Author Haoran Wu [aut, cre] (<https://orcid.org/0000-0001-9449-6112>),
Chen Peng [aut] (<https://orcid.org/0000-0002-9449-7606>)

Maintainer Haoran Wu <haoran.wu@wolfson.ox.ac.uk>

Repository CRAN

Date/Publication 2024-07-02 06:20:02 UTC

Contents

`*.pp` *Multiply Operator for Phase Points*

Description

The multiply operator used to perform arithmetic on phase points.

Usage

```
## S3 method for class 'pp'
x * y
```

Arguments

x	an object of "pp" class representing a phase point.
y	an object of "pp" class representing another phase point.

Value

an object of "pp" class representing the calculated result.

Examples

```
pp(list(x = 1, y = 1)) * pp(list(x = 3, y = 4))
```

+.pp — *Add Operator for Phase Points*

Description

The add operator used to perform arithmetic on phase points.

Usage

```
## S3 method for class 'pp'
x + y
```

Arguments

x	an object of "pp" class representing a phase point.
y	an object of "pp" class representing another phase point.

Value

an object of "pp" class representing the calculated result.

Examples

```
pp(list(x = 1, y = 1)) + pp(list(x = 3, y = 4))
```

-.pp *Subtract Operator for Phase Points*

Description

The subtract operator used to perform arithmetic on phase points.

Usage

```
## S3 method for class 'pp'
x - y
```

Arguments

x	an object of "pp" class representing a phase point.
y	an object of "pp" class representing another phase point.

Value

an object of "pp" class representing the calculated result.

Examples

```
pp(list(x = 1, y = 1)) - pp(list(x = 3, y = 4))
```

/.pp *Divide Operator for Phase Points*

Description

The devide operator used to perform arithmetic on phase points.

Usage

```
## S3 method for class 'pp'
x / y
```

Arguments

x	an object of "pp" class representing a phase point.
y	an object of "pp" class representing another phase point.

Value

an object of "pp" class representing the calculated result.

Examples

```
pp(list(x = 1, y = 1)) / pp(list(x = 3, y = 4))
```

---

eode *Create an ODE System*

Description

Creates an object of class "eode" representing an ODE system that describes population or community dynamics.

Usage

```
eode(..., constraint = NULL)
```

Arguments

`...`	Objects of class "`function`", each representing a single differential equation.
`constraint`	Character strings that must have a format of "`variable>value`" or "`variable<value`", such as "x>1", "y>3", etc. If not specified, then all model variables are considered as positive real numbers by default.

Value

An object of class "eode" describing population or community dynamics.

Examples

```
## Example1: Lotka-Volterra competition model
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
x

## Example2: Susceptible-infected model
dX_Cdt <- function(X_C, Y_C, X_A, Y_A, nu = 0.15, beta = 0.1, mu = 0.15, g = 0.04) {
  nu * (X_A + Y_A) - beta * X_C * (Y_C + Y_A) - (mu + g) * X_C
}

dY_Cdt <- function(X_C, Y_C, Y_A, beta = 0.1, mu = 0.15, g = 0.04, rho = 0.2) {
  beta * X_C * (Y_C + Y_A) - (mu + g + rho) * Y_C
}

dX_Adt <- function(X_C, Y_C, X_A, Y_A, beta = 0.1, g = 0.04) {
  g * X_C - beta * X_A * (Y_C + Y_A)
}

dY_Adt <- function(X_A, Y_C, Y_A, beta = 0.1, g = 0.04, rho = 0.2) {
  beta * X_A * (Y_C + Y_A) + g * Y_C - rho * Y_A
```

```
}

x <- eode(
  dX_Cdt = dX_Cdt, dY_Cdt = dY_Cdt, dX_Adt = dX_Adt, dY_Adt = dY_Adt,
  constraint = c("X_C>=0", "Y_C>=0", "X_A>=0", "Y_A>=0")
)
x
```

`eode_get_cripoi` *Find Equilibrium Point*

Description

Finds an equilibrium point (or critical point) of an ODE system based on Newton iteration method.

Usage

```
eode_get_cripoi(
  x,
  init_value,
  eps = 0.001,
  max_step = 0.01,
  method = c("Newton", "GradDesc"),
  verbose = TRUE
)
```

Arguments

`x`	Object of class "eode" representing an ODE system.
`init_value`	An object of class "pp" representing a phase point giving start estimates.
`eps`	Precision for the stopping criterion. Iteration will stop after the movement of the phase point in a single step is smaller than eps.
`max_step`	Maximum number
`method`	one of "Newton", "GradDesc"
`verbose`	Logical, whether to print the iteration process.

Value

An object of class "pp" representing an equilibrium point found.

Examples

```
## Example 1: Lotka-Volterra competition model
eq1 <- function(x, y, r1 = 1, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
eode_get_cripoi(x, init_value = pp(list(x = 0.5, y = 0.5)))
```

```
## Example 2: Susceptible-infected model
dX_Cdt <- function(X_C, Y_C, X_A, Y_A, nu = 0.15, beta = 0.1, mu = 0.15, g = 0.04) {
  nu * (X_A + Y_A) - beta * X_C * (Y_C + Y_A) - (mu + g) * X_C
}

dY_Cdt <- function(X_C, Y_C, Y_A, beta = 0.1, mu = 0.15, g = 0.04, rho = 0.2) {
  beta * X_C * (Y_C + Y_A) - (mu + g + rho) * Y_C
}

dX_Adt <- function(X_C, Y_C, X_A, Y_A, beta = 0.1, g = 0.04) {
  g * X_C - beta * X_A * (Y_C + Y_A)
}

dY_Adt <- function(X_A, Y_C, Y_A, beta = 0.1, g = 0.04, rho = 0.2) {
  beta * X_A * (Y_C + Y_A) + g * Y_C - rho * Y_A
}

x <- eode(
  dX_Cdt = dX_Cdt, dY_Cdt = dY_Cdt, dX_Adt = dX_Adt, dY_Adt = dY_Adt,
  constraint = c("X_C>=0", "Y_C>=0", "X_A>=0", "Y_A>=0")
)
eode_get_cripoi(x, init_value = pp(list(X_C = 1, Y_C = 1, X_A = 1, Y_A = 1)))
```

`eode_get_sysmat` *System Matrix*

Description

Linearlise an ODE system and calculate its system matrix.

Usage

```
eode_get_sysmat(x, value, delta = 0.00001)
```

Arguments

`x`	object of class "eode" representing an ODE system.
`value`	an object of class "pp" representing a phase point in the ODE system under consideration.
`delta`	Spacing or step size of a numerical grid used for calculating numerical differentiation.

Value

An object of class "`matrix`". Each matrix entry, (i,j), represents the partial derivative of the i-th equation of the system with respect to the j-th variable taking other variables as a constant.

Examples

```
## Example1: Lotka-Volterra competition model
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
eode_get_sysmat(x, value = pp(list(x = 1, y = 1)), delta = 10e-6)

## Example2: Susceptible-infected model
dX_Cdt <- function(X_C, Y_C, X_A, Y_A, nu = 0.15, beta = 0.1, mu = 0.15, g = 0.04) {
  nu * (X_A + Y_A) - beta * X_C * (Y_C + Y_A) - (mu + g) * X_C
}

dY_Cdt <- function(X_C, Y_C, Y_A, beta = 0.1, mu = 0.15, g = 0.04, rho = 0.2) {
  beta * X_C * (Y_C + Y_A) - (mu + g + rho) * Y_C
}

dX_Adt <- function(X_C, Y_C, X_A, Y_A, beta = 0.1, g = 0.04) {
  g * X_C - beta * X_A * (Y_C + Y_A)
}

dY_Adt <- function(X_A, Y_C, Y_A, beta = 0.1, g = 0.04, rho = 0.2) {
  beta * X_A * (Y_C + Y_A) + g * Y_C - rho * Y_A
}

x <- eode(
  dX_Cdt = dX_Cdt, dY_Cdt = dY_Cdt, dX_Adt = dX_Adt, dY_Adt = dY_Adt,
  constraint = c("X_C>=0", "Y_C>=0", "X_A>=0", "Y_A>=0")
)
eode_get_sysmat(x, value = pp(list(X_A = 4, Y_A = 4, X_C = 4, Y_C = 4)), delta = 10e-6)
```

---

eode_get_velocity *Velocity Vector*

Description

Calculates the velocity vector at a given phase point.

Usage

```
eode_get_velocity(x, value, phase_curve = NULL)
```

Arguments

x	The ODE system under consideration. An object of class "eode".
value	an object of "pp" classs representing a phase point in the ODE system under consideration.
phase_curve	an object of "pc" class representing a phase curve. Only used when there is delayed items in the ODE system.

Value

an object of class "velocity" representing a velocity vector at a given phase point.

Examples

```
## Example1: Lotka-Volterra competition model
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
eode_get_velocity(x, value = pp(list(x = 1, y = 1)))

## Example2: Susceptible-infected model
dX_Cdt <- function(X_C, Y_C, X_A, Y_A, nu = 0.15, beta = 0.1, mu = 0.15, g = 0.04) {
  nu * (X_A + Y_A) - beta * X_C * (Y_C + Y_A) - (mu + g) * X_C
}

dY_Cdt <- function(X_C, Y_C, Y_A, beta = 0.1, mu = 0.15, g = 0.04, rho = 0.2) {
  beta * X_C * (Y_C + Y_A) - (mu + g + rho) * Y_C
}

dX_Adt <- function(X_C, Y_C, X_A, Y_A, beta = 0.1, g = 0.04) {
  g * X_C - beta * X_A * (Y_C + Y_A)
}

dY_Adt <- function(X_A, Y_C, Y_A, beta = 0.1, g = 0.04, rho = 0.2) {
  beta * X_A * (Y_C + Y_A) + g * Y_C - rho * Y_A
}

x <- eode(
  dX_Cdt = dX_Cdt, dY_Cdt = dY_Cdt, dX_Adt = dX_Adt, dY_Adt = dY_Adt,
  constraint = c("X_C>=0", "Y_C>=0", "X_A>=0", "Y_A>=0")
)
eode_get_velocity(x, value = pp(list(X_A = 5, Y_A = 5, X_C = 3, Y_C = 2)))
```

---

eode_gridSearch *Grid Search For Optimal Parameters*

Description

Find optimal parameters in the ODE system using grid search method.

Usage

```
eode_gridSearch(x, pdat, space, step = 0.01)
```

Arguments

x	the ODE system under consideration. An object of "eode" class.
pdat	observed population dynamics. An object of "pdata" class.
space	space
step	interval of time for running simulations. Parameter of the function "eode_proj()".

Value

a data frame showing attempted parameters along with the corresponding values of loss function.

Examples

```
dX_Cdt <- function(X_C, Y_C, X_A, Y_A, nu = 0.15, beta = 0.1, mu = 0.15, g = 0.04) {
  nu * (X_A + Y_A) - beta * X_C * (Y_C + Y_A) - (mu + g) * X_C
}

dY_Cdt <- function(X_C, Y_C, Y_A, beta = 0.1, mu = 0.15, g = 0.04, rho = 0.2) {
  beta * X_C * (Y_C + Y_A) - (mu + g + rho) * Y_C
}

dX_Adt <- function(X_C, Y_C, X_A, Y_A, beta = 0.1, g = 0.04) {
  g * X_C - beta * X_A * (Y_C + Y_A)
}

dY_Adt <- function(X_A, Y_C, Y_A, beta = 0.1, g = 0.04, rho = 0.2) {
  beta * X_A * (Y_C + Y_A) + g * Y_C - rho * Y_A
}

x <- eode(
  dX_Cdt = dX_Cdt, dY_Cdt = dY_Cdt, dX_Adt = dX_Adt, dY_Adt = dY_Adt,
  constraint = c("X_C>=0", "Y_C>=0", "X_A>=0", "Y_A>=0")
)
training_data <- pdata(x,
  init = data.frame(
    X_A = c(9, 19, 29, 39),
    Y_A = c(1, 1, 1, 1),
    X_C = c(5, 5, 5, 5),
    Y_C = c(0, 0, 0, 0)
  ),
  t = c(3, 3, 3, 3),
  lambda = data.frame(incidence = c(0.4, 0.8, 0.9, 0.95)),
  formula = "incidence = (Y_A + Y_C)/(X_A + X_C + Y_A + Y_C)"
)
res <- eode_gridSearch(x, pdat = training_data, space = list(beta = seq(0.05, 0.15, 0.01)))
res
```

eode_is_centre *Centre*

Description

Check whether an equilibrium point is a neutral centre or not.

Usage

```
eode_is_centre(x, value, eps = 0.001)
```

Arguments

x	Object of class "eode" representing an ODE system.
value	an object of class "pp" representing a phase point in the ODE system under consideration.
eps	Precision used to check whether the input phase point is an equilibrium point. If the absolute value of any component of the phase velocity vector at a phase point is lower than eps, an error would be thrown out.

Value

a TRUE or FALSE.

Examples

```
eq1 <- function(x, y, r1 = 1, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
eode_is_centre(x, value = pp(list(x = 0.3333, y = 0.3333)))
```

eode_is_saddle *Saddle*

Description

Check whether an equilibrium point is a saddle or not.

Usage

```
eode_is_saddle(x, value, eps = 0.001)
```

Arguments

x	Object of class "eode" representing an ODE system.
value	an object of class "pp" representing a phase point in the ODE system under consideration.
eps	Precision used to check whether the input phase point is an equilibrium point. If the absolute value of any component of the phase velocity vector at a phase point is lower than eps, an error would be thrown out.

Value

a TRUE or FALSE.

Examples

```
eq1 <- function(x, y, r1 = 1, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
eode_is_saddle(x, value = pp(list(x = 0.3333, y = 0.3333)))
```

---

eode_is_stafoc *Stable Focus*

Description

Check whether an equilibrium point is a stable focus or not.

Usage

```
eode_is_stafoc(x, value, eps = 0.001)
```

Arguments

x	Object of class "eode" representing an ODE system.
value	an object of class "pp" representing a phase point in the ODE system under consideration.
eps	Precision used to check whether the input phase point is an equilibrium point. If the absolute value of any component of the phase velocity vector at a phase point is lower than eps, an error would be thrown out.

Value

a TRUE or FALSE.

Examples

```
eq1 <- function(x, y, r1 = 1, a11 = 2, a12 = 1) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 1, a22 = 2) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
eode_is_stafoc(x, value = pp(list(x = 0.3333, y = 0.3333)))
```

eode_is_stanod *Stable Node*

Description

Check whether an equilibrium point is a stable node or not.

Usage

```
eode_is_stanod(x, value, eps = 0.001)
```

Arguments

x	Object of class "eode" representing an ODE system.
value	an object of class "pp" representing a phase point in the ODE system under consideration.
eps	Precision used to check whether the input phase point is an equilibrium point. If the absolute value of any component of the phase velocity vector at a phase point is lower than eps, an error would be thrown out.

Value

a TRUE or FALSE.

Examples

```
eq1 <- function(x, y, r1 = 1, a11 = 2, a12 = 1) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 1, a22 = 2) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
eode_is_stanod(x, value = pp(list(x = 0.3333, y = 0.3333)))
```

eode_is_stapoi *Stable Equilibrium Point*

Description

Check whether an equilibrium point is stable or not.

Usage

```
eode_is_stapoi(x, value, eps = 0.001)
```

Arguments

x	Object of class "eode" representing an ODE system.
value	an object of class "pp" representing a phase point in the ODE system under consideration.
eps	Precision used to check whether the input phase point is an equilibrium point. If the absolute value of any component of the phase velocity vector at a phase point is lower than eps, an error would be thrown out.

Value

a TRUE or FALSE.

Examples

```
eq1 <- function(x, y, r1 = 1, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
eode_is_stapoi(x, value = pp(list(x = 0.3333, y = 0.3333)))

## Example2: Susceptible-infected model
dX_Cdt <- function(X_C, Y_C, X_A, Y_A, nu = 0.15, beta = 0.1, mu = 0.15, g = 0.04) {
  nu * (X_A + Y_A) - beta * X_C * (Y_C + Y_A) - (mu + g) * X_C
}

dY_Cdt <- function(X_C, Y_C, Y_A, beta = 0.1, mu = 0.15, g = 0.04, rho = 0.2) {
  beta * X_C * (Y_C + Y_A) - (mu + g + rho) * Y_C
}

dX_Adt <- function(X_C, Y_C, X_A, Y_A, beta = 0.1, g = 0.04) {
  g * X_C - beta * X_A * (Y_C + Y_A)
}

dY_Adt <- function(X_A, Y_C, Y_A, beta = 0.1, g = 0.04, rho = 0.2) {
  beta * X_A * (Y_C + Y_A) + g * Y_C - rho * Y_A
}

x <- eode(
  dX_Cdt = dX_Cdt, dY_Cdt = dY_Cdt, dX_Adt = dX_Adt, dY_Adt = dY_Adt,
  constraint = c(
    "X_C>=0", "Y_C>=0", "X_A>=0", "Y_A>=0",
    "X_C<5", "Y_C<5", "X_A<5", "Y_A<5"
  )
)
eode_is_stapoi(x, value = pp(list(X_A = 1.3443, Y_A = 0.2304, X_C = 1.0655, Y_C = 0.0866)))
```

`eode_is_unsfoc` *Unstable Focus*

Description

Check whether an equilibrium point is an unstable focus or not.

Usage

```
eode_is_unsfoc(x, value, eps = 0.001)
```

Arguments

x	Object of class "eode" representing an ODE system.
value	an object of class "pp" representing a phase point in the ODE system under consideration.
eps	Precision used to check whether the input phase point is an equilibrium point. If the absolute value of any component of the phase velocity vector at a phase point is lower than eps, an error would be thrown out.

Value

a TRUE or FALSE.

Examples

```
eq1 <- function(x, y, r1 = 1, a11 = 2, a12 = 1) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 1, a22 = 2) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
eode_is_unsfoc(x, value = pp(list(x = 0.3333, y = 0.3333)))
```

`eode_is_unsnod` *Unstable Node*

Description

Check whether an equilibrium point is an unstable node or not.

Usage

```
eode_is_unsnod(x, value, eps = 0.001)
```

Arguments

x	Object of class "eode" representing an ODE system.
value	an object of class "pp" representing a phase point in the ODE system under consideration.
eps	Precision used to check whether the input phase point is an equilibrium point. If the absolute value of any component of the phase velocity vector at a phase point is lower than eps, an error would be thrown out.

Value

a TRUE or FALSE.

Examples

```
eq1 <- function(x, y, r1 = 1, a11 = 2, a12 = 1) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 1, a22 = 2) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
eode_is_unsnod(x, value = pp(list(x = 0.3333, y = 0.3333)))
```

---

eode_is_validval — *Test the Validity of a Phase Point*

Description

Test whether a phase point sits out of the boundary of an ODE system.

Usage

```
eode_is_validval(x, value)
```

Arguments

x	object of class "eode" representing an ODE system.
value	an object of class "pp" representing a phase point in the ODE system under consideration.

Value

returns TRUE or FALSE depending on whether the values of the system variables are within the boundary or not.

Examples

```
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
eode_is_validval(x, value = pp(list(x = 1, y = 1)))
```

`eode_lossf` *Calculate Loss Function*

Description

Calculate the quadratic loss function given an ODE system and observed population dynamics.

Usage

```
eode_lossf(x, pdat, step = 0.01)
```

Arguments

x	the ODE system under consideration. An object of "eode" class.
pdat	observed population dynamics. An object of "pdata" class.
step	interval of time for each step. Parameter of the function "eode_proj()".

Value

a value of loss function

Examples

```
dX_Cdt <- function(X_C, Y_C, X_A, Y_A, nu = 0.15, beta = 0.1, mu = 0.15, g = 0.04) {
  nu * (X_A + Y_A) - beta * X_C * (Y_C + Y_A) - (mu + g) * X_C
}

dY_Cdt <- function(X_C, Y_C, Y_A, beta = 0.1, mu = 0.15, g = 0.04, rho = 0.2) {
  beta * X_C * (Y_C + Y_A) - (mu + g + rho) * Y_C
}

dX_Adt <- function(X_C, Y_C, X_A, Y_A, beta = 0.1, g = 0.04) {
  g * X_C - beta * X_A * (Y_C + Y_A)
}

dY_Adt <- function(X_A, Y_C, Y_A, beta = 0.1, g = 0.04, rho = 0.2) {
  beta * X_A * (Y_C + Y_A) + g * Y_C - rho * Y_A
}

x <- eode(
  dX_Cdt = dX_Cdt, dY_Cdt = dY_Cdt, dX_Adt = dX_Adt, dY_Adt = dY_Adt,
  constraint = c("X_C>=0", "Y_C>=0", "X_A>=0", "Y_A>=0")
)
training_data <- pdata(x,
  init = data.frame(
    X_A = c(9, 19, 29, 39),
    Y_A = c(1, 1, 1, 1),
    X_C = c(5, 5, 5, 5),
    Y_C = c(0, 0, 0, 0)
```

```
  ),
  t = c(3, 3, 3, 3),
  lambda = data.frame(incidence = c(0.4, 0.8, 0.9, 0.95)),
  formula = "incidence = (Y_A + Y_C)/(X_A + X_C + Y_A + Y_C)"
)
eode_lossf(x, pdat = training_data)
```

`eode_proj` *Solve ODEs*

Description

Provides the numerical solution of an ODE under consideration given an initial condition.

Usage

```
eode_proj(x, value0, N, step = 0.01)
```

Arguments

`x`	the ODE system under consideration. An object of class "eode".
`value0`	value an object of class "pp" representing the phase point at the initial state.
`N`	number of iterations
`step`	interval of time for each step

Value

an object of class "pc" that represents a phase curve

Examples

```
## Example1: Lotka-Volterra competition model
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2, constraint = c("x<1", "y<1"))
eode_proj(x, value0 = pp(list(x = 0.2, y = 0.1)), N = 100)

## Example2: Susceptible-infected model
dX_Cdt <- function(X_C, Y_C, X_A, Y_A, nu = 0.15, beta = 0.1, mu = 0.15, g = 0.04) {
  nu * (X_A + Y_A) - beta * X_C * (Y_C + Y_A) - (mu + g) * X_C
}

dY_Cdt <- function(X_C, Y_C, Y_A, beta = 0.1, mu = 0.15, g = 0.04, rho = 0.2) {
  beta * X_C * (Y_C + Y_A) - (mu + g + rho) * Y_C
}

dX_Adt <- function(X_C, Y_C, X_A, Y_A, beta = 0.1, g = 0.04) {
  g * X_C - beta * X_A * (Y_C + Y_A)
}
```

```
dY_Adt <- function(X_A, Y_C, Y_A, beta = 0.1, g = 0.04, rho = 0.2) {
  beta * X_A * (Y_C + Y_A) + g * Y_C - rho * Y_A
}

x <- eode(
  dX_Cdt = dX_Cdt, dY_Cdt = dY_Cdt, dX_Adt = dX_Adt, dY_Adt = dY_Adt,
  constraint = c("X_C>=0", "Y_C>=0", "X_A>=0", "Y_A>=0")
)
eode_proj(x, value0 = pp(list(X_A = 5, Y_A = 5, X_C = 3, Y_C = 2)), N = 100)
```

`eode_sensitivity_proj` *Sensitivity Analysis*

Description

Run a sensitivity analysis on an ODE system.

Usage

```
eode_sensitivity_proj(x, valueSpace, N, step = 0.01)
```

Arguments

x	object of class "pc" that represents the ODE system under consideration.
valueSpace	a list indicating initial conditions and parameters. Model variables must be included to specify initial values of each variable. Values can be a vector, indicating all the potential values to be considered in the sensitivity analysis.
N	number of iterations
step	interval of time for running simulations. Parameter of the function "`eode_proj()`".

Value

an object of "`pcfamily`" class, each component having three sub-components: `$grid_var`: variables or parameters whose values are changed throughout the sensitivity analysis. `$fixed_var`: variables whose values are not changed. $pc: phase curve. An object of "pc" class.

Examples

```
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2, constraint = c("x<100", "y<100"))
eode_sensitivity_proj(x, valueSpace = list(x = c(0.2, 0.3, 0.4), y = 0.1, a11 = 1:3), N = 100)
```

eode_set_constraint *Set New Constraints*

Description

Set new constraints for an ODE system.

Usage

```
eode_set_constraint(x, new_constraint)
```

Arguments

x	an object of class "eode".
new_constraint	a vector of characters indicating new constraints to be assigned to the ODE system.

Value

an object of "eode" class

Examples

```
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
x
eode_set_constraint(x, c("x<5", "y<5"))
```

eode_set_parameter *Set New Parameters*

Description

Set new parameters for an ODE system.

Usage

```
eode_set_parameter(x, ParaList)
```

Arguments

x	an object of class "eode".
ParaList	a list of names of parameters with their corresponding values to be assigned to the ODE system.

Value

an object of "eode" class

Examples

```
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
x
eode_set_parameter(x, ParaList = list(r1 = 3))
```

---

eode_simuAnnealing *Simulated Annealing For Optimal Parameters*

Description

Find optimal parameters in the ODE system using simulated annealing.

Usage

```
eode_simuAnnealing(
  x,
  pdat,
  paras = "ALL",
  max_disturb = 0.2,
  AnnN = 100,
  step = 0.01,
  prop.train = 1
)
```

Arguments

x	the ODE system under consideration. An object of "eode" class.
pdat	observed population dynamics. An object of "pdata" class.
paras	parameters to be optimised. A character vector. If multiple parameters are specified, the simulation annealing process will proceed by altering multiple parameters at the same time, and accept an alteration if it achieves a lower value of the loss function. Default is "ALL", which means to choose all the parameters.
max_disturb	maximum disturbance in proportion. The biggest disturbance acts on parameters at the beginning of the simulated annealing process.
AnnN	steps of simulated annealing.
step	interval of time for running simulations. Parameter of the function "eode_proj()".
prop.train	proportion of training data set. In each step of annealing, a proportion of dataset will be randomly decided for model training, and the rest for model validation.

Value

a data frame showing attempted parameters along with the corresponding values of loss function.

Examples

```
dX_Cdt <- function(X_C, Y_C, X_A, Y_A, nu = 0.15, beta = 0.1, mu = 0.15, g = 0.04) {
  nu * (X_A + Y_A) - beta * X_C * (Y_C + Y_A) - (mu + g) * X_C
}

dY_Cdt <- function(X_C, Y_C, Y_A, beta = 0.1, mu = 0.15, g = 0.04, rho = 0.2) {
  beta * X_C * (Y_C + Y_A) - (mu + g + rho) * Y_C
}

dX_Adt <- function(X_C, Y_C, X_A, Y_A, beta = 0.1, g = 0.04) {
  g * X_C - beta * X_A * (Y_C + Y_A)
}

dY_Adt <- function(X_A, Y_C, Y_A, beta = 0.1, g = 0.04, rho = 0.2) {
  beta * X_A * (Y_C + Y_A) + g * Y_C - rho * Y_A
}

x <- eode(
  dX_Cdt = dX_Cdt, dY_Cdt = dY_Cdt, dX_Adt = dX_Adt, dY_Adt = dY_Adt,
  constraint = c("X_C>=0", "Y_C>=0", "X_A>=0", "Y_A>=0")
)
training_data <- pdata(x,
  init = data.frame(
    X_A = c(9, 19, 29, 39),
    Y_A = c(1, 1, 1, 1),
    X_C = c(5, 5, 5, 5),
    Y_C = c(0, 0, 0, 0)
  ),
  t = c(3, 3, 3, 3),
  lambda = data.frame(incidence = c(0.4, 0.8, 0.9, 0.95)),
  formula = "incidence = (Y_A + Y_C)/(X_A + X_C + Y_A + Y_C)"
)
res <- eode_simuAnnealing(x, pdat = training_data, paras = "beta", max_disturb = 0.05, AnnN = 20)
res
```

---

eode_stability_type *Stability Analysis*

Description

Check whether an equilibrium point is stable or not, and give the specific type (i.e. stable node, stable focus, unstable node, unstable focus, saddle, or centre). Check whether an equilibrium point is stable or not.

Usage

```
eode_stability_type(x, value, eps = 0.001)
```

Arguments

x	Object of class "eode" representing an ODE system.
value	an object of class "pp" representing a phase point in the ODE system under consideration.
eps	Precision used to check whether the input phase point is an equilibrium point. If the absolute value of any component of the phase velocity vector at a phase point is lower than eps, an error would be thrown out.

Value

a string character indicate the type of the equilbrium point.

Examples

```
eq1 <- function(x, y, r1 = 1, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
eode_stability_type(x, value = pp(list(x = 0.3333, y = 0.3333)))
```

---

length.eode *Length of an ODE System*

Description

Get the number of equations in an ODE system.

Usage

```
## S3 method for class 'eode'
length(x, ...)
```

Arguments

x	Object of "eode" class representing an ODE system under consideration.
...	Ignored.

Value

The number of equations in the ODE system.

Examples

```
## Example 1: Lotka-Volterra competition model
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
length(x)

## Example 2: Susceptible-infected model
dX_Cdt <- function(X_C, Y_C, X_A, Y_A, nu = 0.15, beta = 0.1, mu = 0.15, g = 0.04) {
  nu * (X_A + Y_A) - beta * X_C * (Y_C + Y_A) - (mu + g) * X_C
}

dY_Cdt <- function(X_C, Y_C, Y_A, beta = 0.1, mu = 0.15, g = 0.04, rho = 0.2) {
  beta * X_C * (Y_C + Y_A) - (mu + g + rho) * Y_C
}

dX_Adt <- function(X_C, Y_C, X_A, Y_A, beta = 0.1, g = 0.04) {
  g * X_C - beta * X_A * (Y_C + Y_A)
}

dY_Adt <- function(X_A, Y_C, Y_A, beta = 0.1, g = 0.04, rho = 0.2) {
  beta * X_A * (Y_C + Y_A) + g * Y_C - rho * Y_A
}

x <- eode(
  dX_Cdt = dX_Cdt, dY_Cdt = dY_Cdt, dX_Adt = dX_Adt, dY_Adt = dY_Adt,
  constraint = c("X_C>=0", "Y_C>=0", "X_A>=0", "Y_A>=0")
)
length(x)
```

---

length.pdata *Get Length Of Population Dynamics Data*

Description

Get the number of observations of population dynamics data.

Usage

```
## S3 method for class 'pdata'
length(x, ...)
```

Arguments

`x`	Object of class "pdata".
`...`	Ignored.

Value

A scalar.

Examples

```
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
pdata(x,
  init = data.frame(x = c(10, 20), y = c(5, 15)),
  t = c(3, 3),
  lambda = data.frame(z = c(15, 30)),
  formula = "z = x + y"
)
```

---

pc_calculator *Variable Calculator In ODE Systems*

Description

Calculate one or several variables during the simulation of an ODE system

Usage

```
pc_calculator(x, formula)
```

Arguments

x	the ODE system under consideration. An object of "eode" class.
formula	formula that specifies how to calculate the variable to be traced.

Value

an object of "pc" class with extra variables being attached.

Examples

```
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2, constraint = c("x<100", "y<100"))
pc <- eode_proj(x, value0 = pp(list(x = 0.2, y = 0.1)), N = 100)
pc_calculator(pc, formula = "z = x + y")
```

pc_plot *Plot a Phase Curve in Phase Plane*

Description

Creates a plot of a phase curve in its phase plane

Usage

```
pc_plot(x)
```

Arguments

x	object of class "pc" that represents a phase curve.

Value

a graphic object

Examples

```
eq1 <- function(x, y, r1 = 1, a11 = 2, a12 = 1) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 1, a22 = 2) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2, constraint = c("x<100", "y<100"))
pc <- eode_proj(x, value0 = pp(list(x = 0.2, y = 0.1)), N = 50, step = 0.1)
pc_plot(pc)
```

pdata *Create Population Dynamics Data*

Description

Create a new data set to describe population dynamics with initial conditions. The data is mainly used to train the ODE model described by an object of "eode" class.

Usage

```
pdata(x, init, t, lambda, formula)
```

Arguments

x	an object of class "eode" describing the ODE system in focus.
init	a data frame describing the initial conditions of the population. Each column should correspond to a variable in the ODE system, hence the name of column will be automatically checked to make sure it matches a variable in the ODE system. Each row represents an independent observation
t	a numerical vector that describes the time for each observation
lambda	a data frame describing the observation values of population dynamics. Each column is an variable and each row represents an independent observation.
formula	a character that specifies how the observed variable can be predicted by the ODE system.

Value

an object of "pdata" class

Examples

```
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
pdata(x,
  init = data.frame(x = c(10, 20), y = c(5, 15)),
  t = c(3, 3),
  lambda = data.frame(z = c(15, 30)),
  formula = "z = x + y"
)
```

---

pdist *Distance between Phase Points*

Description

Computes and returns the distance between two phase points.

Usage

```
pdist(x, y)
```

Arguments

x	an object of "pp" class representing a phase point.
y	an object of "pp" class representing another phase point.

Value

A scalar.

Examples

```
a <- pp(list(x = 1, y = 1))
b <- pp(list(x = 3, y = 4))
pdist(a, b)
```

---

plot.eode — *Plot Phase Velocity Vector Field*

Description

Creates a plot of phase velocity vector (or its field) of an ODE system. With two free variables the function creates a plot of the phase velocity vector field within the range defined by model constraints. With a single free variable the function creates a plot of one-dimensional phase velocity vector field. With more than two variables the function will automatically set a value (the middle of the lower and upper boundaries) for any extra variable to reduce axes. The values can also be set using parameter "`set_covar`". If all model variables have had their values, the function creates a plot to show the exact values of a single phase velocity vector.

Usage

```
## S3 method for class 'eode'
plot(
  x,
  n.grid = 20,
  scaleL = 1,
  scaleAH = 1,
  scaleS = 1,
  set_covar = NULL,
  ...
)
```

Arguments

`x`	The ODE system under consideration. An object of class "`eode`".
`n.grid`	number of grids per dimension. A larger number indicates a smaller grid size. Default 20.
`scaleL`	scale factor of the arrow length.
`scaleAH`	scale factor of the arrow head.
`scaleS`	scale factor of the arrow size.
`set_covar`	give certain values of model variables that are going to be fixed.
`...`	ignored arguments

Value

a graphic object

Examples

```
## Example 1: Lotka-Volterra competition model
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
plot(x)

## Example 2: Susceptible-infected model
dX_Cdt <- function(X_C, Y_C, X_A, Y_A, nu = 0.15, beta = 0.1, mu = 0.15, g = 0.04) {
  nu * (X_A + Y_A) - beta * X_C * (Y_C + Y_A) - (mu + g) * X_C
}

dY_Cdt <- function(X_C, Y_C, Y_A, beta = 0.1, mu = 0.15, g = 0.04, rho = 0.2) {
  beta * X_C * (Y_C + Y_A) - (mu + g + rho) * Y_C
}

dX_Adt <- function(X_C, Y_C, X_A, Y_A, beta = 0.1, g = 0.04) {
  g * X_C - beta * X_A * (Y_C + Y_A)
}

dY_Adt <- function(X_A, Y_C, Y_A, beta = 0.1, g = 0.04, rho = 0.2) {
  beta * X_A * (Y_C + Y_A) + g * Y_C - rho * Y_A
}

x <- eode(
  dX_Cdt = dX_Cdt, dY_Cdt = dY_Cdt, dX_Adt = dX_Adt, dY_Adt = dY_Adt,
  constraint = c("X_C>=0", "Y_C>=0", "X_A>=0", "Y_A>=0")
)
plot(x)
plot(x, set_covar = list(X_A = 60, Y_A = 80))
plot(x, set_covar = list(Y_C = 80, X_A = 60, Y_A = 80))
plot(x, set_covar = list(X_C = 60, Y_C = 80, X_A = 60, Y_A = 80))
```

---

plot.pc *Plot a Phase Curve With Time*

Description

Creates a plot of a phase curve

Usage

```
## S3 method for class 'pc'
plot(x, model_var_label = NULL, model_var_color = NULL, ...)
```

Arguments

x	object of class "pc" that represents a phase curve.

model_var_label
: a list indicating labels of model variables. Name should be old variable names and values should be names of labels.

model_var_color
: a list indicating colors of model variable. Name should be old variable names and values should be 16-bit color codes.

...
: additional parameters accepted by the function "geom_line".

Value

a graphic object

Examples

```
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2, constraint = c("x<1", "y<1"))
pc <- eode_proj(x, value0 = pp(list(x = 0.2, y = 0.1)), N = 100)
plot(pc)
```

---

plot.pcfamily *Plot Phase Curve Family*

Description

Creates a plot of a phase curve family.

Usage

```
## S3 method for class 'pcfamily'
plot(
  x,
  model_var_label = NULL,
  model_var_color = NULL,
  facet_grid_label = NULL,
  facet_paras = TRUE,
  ...
)
```

Arguments

x
: object of class "pcfamily" that represents a phase curve family.

model_var_label
: a list indicating labels of model variables. Name should be old variable names and values should be names of labels.

model_var_color
: a list indicating colors of model variable. Name should be old variable names and values should be 16-bit color codes.

facet_grid_label
: a list indicating facet labels. Name should be old variable names and values should be facet labels.

facet_paras
: a logical vector indicating whether facet?

...
: other parameters

Value

a graphic object

Examples

```
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2, constraint = c("x<1", "y<1"))
value_space <- list(x = c(0.2, 0.3, 0.4), y = 0.1, a11 = 1:3)
plot(eode_sensitivity_proj(x, valueSpace = value_space, N = 100))
```

---

plot.velocity *Create a Plot of a Phase Velocity Vector*

Description

Plot a phase velocity vector

Usage

```
## S3 method for class 'velocity'
plot(x, ...)
```

Arguments

x
: Object of class "velocity".

...
: Ignored.

Value

ggplot object

Examples

```
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
plot(eode_get_velocity(x, value = pp(list(x = 1, y = 1))))
```

pp — *Create a Phase Point*

Description

Creates an object of class "pp" representing a phase point in the phase space.

Usage

```
pp(value)
```

Arguments

value	A list of several elements, each giving a value on one dimension.

Value

An object of class "pp" describing a phase point.

Examples

```
pp(list(x = 1, y = 1))
```

print.eode — *Print Brief Details of an ODE System*

Description

Prints a very brief description of an ODE system.

Usage

```
## S3 method for class 'eode'
print(x, ...)
```

Arguments

x	Object of class "eode".
...	Ignored.

Value

No value

Examples

```
## Example 1: Lotka-Volterra competition model
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
x

## Example 2: Susceptible-infected model
dX_Cdt <- function(X_C, Y_C, X_A, Y_A, nu = 0.15, beta = 0.1, mu = 0.15, g = 0.04) {
  nu * (X_A + Y_A) - beta * X_C * (Y_C + Y_A) - (mu + g) * X_C
}

dY_Cdt <- function(X_C, Y_C, Y_A, beta = 0.1, mu = 0.15, g = 0.04, rho = 0.2) {
  beta * X_C * (Y_C + Y_A) - (mu + g + rho) * Y_C
}

dX_Adt <- function(X_C, Y_C, X_A, Y_A, beta = 0.1, g = 0.04) {
  g * X_C - beta * X_A * (Y_C + Y_A)
}

dY_Adt <- function(X_A, Y_C, Y_A, beta = 0.1, g = 0.04, rho = 0.2) {
  beta * X_A * (Y_C + Y_A) + g * Y_C - rho * Y_A
}

x <- eode(
  dX_Cdt = dX_Cdt, dY_Cdt = dY_Cdt, dX_Adt = dX_Adt, dY_Adt = dY_Adt,
  constraint = c("X_C>=0", "Y_C>=0", "X_A>=0", "Y_A>=0")
)
x
```

---

print.pc — *Print Brief Details of a Phase Curve*

Description

Prints a very brief description of a phase curve.

Usage

```
## S3 method for class 'pc'
print(x, ...)
```

Arguments

x	Object of class "pc".
...	Ignored.

Value

No value

Examples

```
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2, constraint = c("x<1", "y<1"))
eode_proj(x, value0 = pp(list(x = 0.9, y = 0.9)), N = 8)
```

---

print.pcfamily *Print Brief Details of a Phase Curve Family*

Description

Prints a very brief description of a phase curve family.

Usage

```
## S3 method for class 'pcfamily'
print(x, ...)
```

Arguments

x	Object of class "pcfamily".
...	Ignored.

Value

No value

Examples

```
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2, constraint = c("x<100", "y<100"))
eode_sensitivity_proj(x, valueSpace = list(x = c(0.2, 0.3, 0.4), y = 0.1, a11 = 1:3), N = 100)
```

---

print.pdata *Print Brief Details of Population Dynamics Data*

Description

Prints a very brief description of population dynamics data.

Usage

```
## S3 method for class 'pdata'
print(x, ...)
```

Arguments

`x`	Object of class "pdata".
`...`	Ignored.

Value

No value

Examples

```
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
pdata(x,
  init = data.frame(x = c(10, 20), y = c(5, 15)),
  t = c(3, 3),
  lambda = data.frame(z = c(15, 30)),
  formula = "z = x + y"
)
```

`print.pp` *Print Brief Details of a Phase Point*

Description

Prints a very brief description of a phase point.

Usage

```
## S3 method for class 'pp'
print(x, ...)
```

Arguments

`x`	Object of class "pp".
`...`	Ignored.

Value

No value

Examples

```
pp(list(x = 1, y = 1))
```

print.velocity *Print Brief Details of a Phase Velocity Vector*

Description

Prints a very brief description of a phase velocity vector.

Usage

```
## S3 method for class 'velocity'
print(x, ...)
```

Arguments

x	Object of class "velocity".
...	Ignored.

Value

No value

Examples

```
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
eode_get_velocity(x, value = pp(list(x = 1, y = 1)))
```

[.pdata *Extract Parts of an Population Dynamics Data*

Description

Operators acting on population dynamics data.

Usage

```
## S3 method for class 'pdata'
x[i]
```

Arguments

x	Object of class "pdata".
i	indice specifying elements to extract.

Value

pdata object

Examples

```
eq1 <- function(x, y, r1 = 4, a11 = 1, a12 = 2) (r1 - a11 * x - a12 * y) * x
eq2 <- function(x, y, r2 = 1, a21 = 2, a22 = 1) (r2 - a21 * x - a22 * y) * y
x <- eode(dxdt = eq1, dydt = eq2)
pdat <- pdata(x,
  init = data.frame(x = c(10, 20), y = c(5, 15)),
  t = c(3, 3),
  lambda = data.frame(z = c(15, 30)),
  formula = "z = x + y"
)
pdat[1]
```

Index