

# Package ‘rODE’

November 10, 2017

**Type** Package

**Title** Ordinary Differential Equation (ODE) Solvers Written in R Using S4 Classes

**Version** 0.99.6

**Description** Show physics, math and engineering students how an ODE solver is made and how effective R classes can be for the construction of the equations that describe natural phenomena. Inspiration for this work comes from the book on “Computer Simulations in Physics” by Harvey Gould, Jan Tobochnik, and Wolfgang Christian.  
Book link: <<http://www.compadre.org/osp/items/detail.cfm?ID=7375>>.

**Depends** R (>= 3.3.0)

**License** GPL-2

**Encoding** UTF-8

**Imports** methods, data.table

**LazyData** true

**Suggests** knitr, testthat, rmarkdown, ggplot2, dplyr, tidyr, covr, scales

**RoxygenNote** 6.0.1

**Collate** 'ode\_generics.R' 'ODESolver.R' 'ODE.R' 'AbstractODESolver.R' 'ODEAdaptiveSolver.R' 'DormandPrince45.R' 'Euler.R' 'EulerRichardson.R' 'ODESolverFactory.R' 'RK4.R' 'RK45.R' 'Verlet.R' 'rODE-package.r' 'utils.R'

**VignetteBuilder** knitr

**URL** <https://github.com/f0nzie/rODE>

**NeedsCompilation** no

**Author** Alfonso R. Reyes [aut, cre]

**Maintainer** Alfonso R. Reyes <[alfonso.reyes@oilgainsanalytics.com](mailto:alfonso.reyes@oilgainsanalytics.com)>

**Repository** CRAN

**Date/Publication** 2017-11-10 04:17:51 UTC

**R topics documented:**

rODE-package	2
AbstractODESolver-class	3
DormandPrince45-class	4
doStep	8
enableRuntimeExceptions	10
Euler-class	11
EulerRichardson-class	16
getEnergy	17
getErrorCode	19
getExactSolution	20
getODE	22
getRate	22
getRateCounter	24
getRateCounts	25
getState	25
getStepSize	27
getTime	29
getTolerance	31
importFromExamples	32
init	32
ODE-class	33
ODEAdaptiveSolver-class	36
ODESolver-class	37
ODESolverFactory-class	38
RK4-class	40
RK45-class	44
run_test_applications	46
setSolver<-	46
setState	46
setStepSize	48
setTolerance	50
showMethods2	53
step	53
Verlet-class	54
<b>Index</b>	<b>59</b>

rODE-package

*Ordinary Differential Equations***Description**

Ordinary Differential Equations rODE.

---

AbstractODESolver-class

*AbstractODESolver class*

---

### Description

Defines the basic methods for all the ODE solvers.

AbstractODESolver generic

AbstractODESolver constructor missing

AbstractODESolver constructor ODE. Uses this constructor when ODE object is passed

### Usage

```
AbstractODESolver(ode, ...)
```

```
## S4 method for signature 'AbstractODESolver'  
step(object, ...)
```

```
## S4 method for signature 'AbstractODESolver'  
getODE(object, ...)
```

```
## S4 method for signature 'AbstractODESolver'  
setStepSize(object, stepSize, ...)
```

```
## S4 method for signature 'AbstractODESolver'  
init(object, stepSize, ...)
```

```
## S4 replacement method for signature 'AbstractODESolver'  
init(object, ...) <- value
```

```
## S4 method for signature 'AbstractODESolver'  
getStepSize(object, ...)
```

```
## S4 method for signature 'missing'  
AbstractODESolver(ode, ...)
```

```
## S4 method for signature 'ODE'  
AbstractODESolver(ode, ...)
```

### Arguments

ode	an ODE object
...	additional parameters
object	a class object
stepSize	the size of the step
value	the step size value

**Details**

Inherits from: ODESolver class

**Examples**

```
# This is how we start defining a new ODE solver: Euler
.Euler <- setClass("Euler",          # Euler solver very simple; no slots
  contains = c("AbstractODESolver"))

# Here we define the ODE solver Verlet
.Verlet <- setClass("Verlet", slots = c(
  rate1 = "numeric",                # Verlet calculates two rates
  rate2 = "numeric",
  rateCounter = "numeric"),
  contains = c("AbstractODESolver"))

# This is the definition of the ODE solver Runge-Kutta 4
.RK4 <- setClass("RK4", slots = c(   # On the other hand RK4 uses 4 rates
  rate1 = "numeric",
  rate2 = "numeric",
  rate3 = "numeric",
  rate4 = "numeric",
  estimated_state = "numeric"),     # and estimates another state
  contains = c("AbstractODESolver"))
```

---

DormandPrince45-class *DormandPrince45 ODE solver class*

---

**Description**

DormandPrince45 ODE solver class

DormandPrince45 generic

DormandPrince45 constructor ODE

**Usage**

```
DormandPrince45(ode, ...)
```

```
## S4 method for signature 'DormandPrince45'
init(object, stepSize, ...)
```

```
## S4 replacement method for signature 'DormandPrince45'
init(object, ...) <- value
```

```

## S4 method for signature 'DormandPrince45'
step(object, ...)

## S4 method for signature 'DormandPrince45'
enableRuntimeExceptions(object, enable)

## S4 method for signature 'DormandPrince45'
setStepSize(object, stepSize, ...)

## S4 method for signature 'DormandPrince45'
getStepSize(object, ...)

## S4 method for signature 'DormandPrince45'
setTolerance(object, tol)

## S4 replacement method for signature 'DormandPrince45'
setTolerance(object, ...) <- value

## S4 method for signature 'DormandPrince45'
getTolerance(object)

## S4 method for signature 'DormandPrince45'
getErrorCode(object)

## S4 method for signature 'ODE'
DormandPrince45(ode, ...)

```

### Arguments

ode	ODE object
...	additional parameters
object	a class object
stepSize	size of the step
value	step size to set
enable	a logical flag
tol	tolerance

### Examples

```

# ~~~~~ base class: KeplerVerlet.R

setClass("KeplerDormandPrince45", slots = c(
  GM = "numeric",
  odeSolver = "DormandPrince45",
  counter = "numeric"
),
contains = c("ODE")

```

```

)

setMethod("initialize", "KeplerDormandPrince45", function(.Object, ...) {
  .Object@GM <- 4 * pi * pi      # gravitation constant times combined mass
  .Object@state <- vector("numeric", 5) # x, vx, y, vy, t
  .Object@odeSolver <- DormandPrince45(.Object)
  .Object@counter <- 0
  return(.Object)
})

setMethod("doStep", "KeplerDormandPrince45", function(object, ...) {
  object@odeSolver <- step(object@odeSolver)
  object@state <- object@odeSolver@ode@state
  object
})

setMethod("getTime", "KeplerDormandPrince45", function(object, ...) {
  return(object@state[5])
})

setMethod("getEnergy", "KeplerDormandPrince45", function(object, ...) {
  ke <- 0.5 * (object@state[2] * object@state[2] +
              object@state[4] * object@state[4])
  pe <- -object@GM / sqrt(object@state[1] * object@state[1] +
                        object@state[3] * object@state[3])
  return(pe+ke)
})

setMethod("init", "KeplerDormandPrince45", function(object, initState, ...) {
  object@state <- initState
  # call init in AbstractODESolver
  object@odeSolver <- init(object@odeSolver, getStepSize(object@odeSolver))
  object@counter <- 0
  object
})

setReplaceMethod("init", "KeplerDormandPrince45", function(object, ..., value) {
  object@state <- value
  # call init in AbstractODESolver
  object@odeSolver <- init(object@odeSolver, getStepSize(object@odeSolver))
  object@counter <- 0
  object
})

setMethod("getRate", "KeplerDormandPrince45", function(object, state, ...) {
  # Computes the rate using the given state.
  r2 <- state[1] * state[1] + state[3] * state[3] # distance squared
  r3 <- r2 * sqrt(r2) # distance cubed
  object@rate[1] <- state[2]
  object@rate[2] <- (- object@GM * state[1]) / r3
  object@rate[3] <- state[4]
  object@rate[4] <- (- object@GM * state[3]) / r3
  object@rate[5] <- 1 # time derivative
})

```



```

                                rate.counts = getRateCounts(ode),
                                time = time )
  ode_solver <- step(ode_solver)      # advance solver one step
  stepSize <- getStepSize(ode_solver) # get the current step size
  time <- time + stepSize
  ode <- getODE(ode_solver)          # get updated ODE object
  state <- getState(ode)             # get the `state` vector
  i <- i + 1                          # add a row vector
}
DT <- data.table::rbindlist(rowVector) # create data table
return(DT)
}

solution <- ComparisonRK45ODEApp()
plot(solution)

# additional plot for analytics solution vs. RK45 solver
solution.multi <- solution %>%
  select(t, ODE, exact)
plot(solution.multi)          # 3x3 plot

# plot comparative curves analytical vs ODE solver
solution.2x1 <- solution.multi %>%
  gather(key, value, -t)      # make a table of 3 variables. key: ODE/exact

g <- ggplot(solution.2x1, mapping = aes(x = t, y = value, color = key))
g <- g + geom_line(size = 1) +
  labs(title = "ODE vs Exact solution",
        subtitle = "tolerance = 1E-6")
print(g)

```

---

doStep

*doStep*


---

### Description

Perform a step

### Usage

```
doStep(object, ...)
```

### Arguments

object	a class object
...	additional parameters



**Examples**

```

# ++++++ example: PlanetApp.R
# Simulation of Earth orbiting around the Sun using the Euler ODE solver

importFromExamples("Planet.R")      # source the class

PlanetApp <- function(verbose = FALSE) {
  # x = 1, AU or Astronomical Units. Length of semimajor axis or the orbit
  # of the Earth around the Sun.
  x <- 1; vx <- 0; y <- 0; vy <- 6.28; t <- 0
  state <- c(x, vx, y, vy, t)
  dt <- 0.01
  planet <- Planet()
  planet@odeSolver <- setStepSize(planet@odeSolver, dt)
  planet <- init(planet, initState = state)
  rowvec <- vector("list")
  i <- 1
  # run infinite loop. stop with ESCAPE.
  while (getState(planet)[5] <= 90) {      # Earth orbit is 365 days around the sun
    rowvec[[i]] <- list(t = getState(planet)[5],      # just doing 3 months
                       x = getState(planet)[1],      # to speed up for CRAN
                       vx = getState(planet)[2],
                       y = getState(planet)[3],
                       vy = getState(planet)[4])
    for (j in 1:5) {                          # advances time
      planet <- doStep(planet)
    }
    i <- i + 1
  }
  DT <- data.table::rbindlist(rowvec)
  return(DT)
}
# run the application
solution <- PlanetApp()
select_rows <- seq(1, nrow(solution), 10)      # do not overplot
solution <- solution[select_rows,]
plot(solution)

# ++++++ application: Logistic.R
# Simulates the logistic equation
importFromExamples("Logistic.R")

# Run the application
LogisticApp <- function(verbose = FALSE) {
  x <- 0.1
  vx <- 0
  r <- 2      # Malthusian parameter (rate of maximum population growth)
  K <- 10.0   # carrying capacity of the environment
  dt <- 0.01; tol <- 1e-3; tmax <- 10

  population <- Logistic()      # create a Logistic ODE object

```

```

# Two ways of initializing the object
# population <- init(population, c(x, vx, 0), r, K)
init(population) <- list(initState = c(x, vx, 0),
                        r = r,
                        K = K)

odeSolver <- Verlet(population)      # select the solver

# Two ways of initializing the solver
# odeSolver <- init(odeSolver, dt)
init(odeSolver) <- dt

population@odeSolver <- odeSolver
# setSolver(population) <- odeSolver

rowVector <- vector("list")
i <- 1
while (getTime(population) <= tmax) {
  rowVector[[i]] <- list(t = getTime(population),
                        s1 = getState(population)[1],
                        s2 = getState(population)[2])
  population <- doStep(population)
  i <- i + 1
}
DT <- data.table::rbindlist(rowVector)
return(DT)
}
# show solution
solution <- LogisticApp()
plot(solution)

```

---

enableRuntimeExceptions

*enableRuntimeExceptions*

---

## Description

Enable Runtime Exceptions

## Usage

```
enableRuntimeExceptions(object, enable, ...)
```

## Arguments

object	a class object
enable	a boolean to enable exceptions
...	additional parameters

**Examples**

```
setMethod("enableRuntimeExceptions", "DormandPrince45", function(object, enable) {  
  object@enableExceptions <- enable  
})
```

---

Euler-class

*Euler ODE solver class*

---

**Description**

Euler ODE solver class

Euler generic

Euler constructor when 'ODE' passed

Euler constructor 'missing' is passed

**Usage**

```
Euler(ode, ...)
```

```
## S4 method for signature 'Euler'  
init(object, stepSize, ...)
```

```
## S4 method for signature 'Euler'  
step(object, ...)
```

```
## S4 method for signature 'Euler'  
setStepSize(object, stepSize, ...)
```

```
## S4 method for signature 'Euler'  
getStepSize(object, ...)
```

```
## S4 method for signature 'ODE'  
Euler(ode, ...)
```

```
## S4 method for signature 'missing'  
Euler(ode, ...)
```

**Arguments**

ode	an ODE object
...	additional parameters
object	an internal object of the class
stepSize	the size of the step

**Examples**

```

# ++++++ application: RigidBodyNXFApp.R
# example of a nonstiff system is the system of equations describing
# the motion of a rigid body without external forces.

importFromExamples("RigidBody.R")

# run the application
RigidBodyNXFApp <- function(verbose = FALSE) {
  # load the R class that sets up the solver for this application
  y1 <- 0 # initial y1 value
  y2 <- 1 # initial y2 value
  y3 <- 1 # initial y3 value
  dt <- 0.01 # delta time for step

  body <- RigidBodyNXF(y1, y2, y3)
  solver <- Euler(body)
  solver <- setStepSize(solver, dt)
  rowVector <- vector("list")
  i <- 1
  # stop loop when the body hits the ground
  while (getState(body)[4] <= 12) {
    rowVector[[i]] <- list(t = getState(body)[4],
                          y1 = getState(body)[1],
                          y2 = getState(body)[2],
                          y3 = getState(body)[3])

    solver <- step(solver)
    body <- getODE(solver)
    i <- i + 1
  }
  DT <- data.table::rbindlist(rowVector)
  return(DT)
}

# get the data table from the app
solution <- RigidBodyNXFApp()
plot(solution)

# ++++++ example: FallingParticleApp.R
# Application that simulates the free fall of a ball using Euler ODE solver

importFromExamples("FallingParticleODE.R") # source the class

FallingParticleODEApp <- function(verbose = FALSE) {
  # initial values
  initial_y <- 10
  initial_v <- 0
  dt <- 0.01
  ball <- FallingParticleODE(initial_y, initial_v)
  solver <- Euler(ball) # set the ODE solver
  solver <- setStepSize(solver, dt) # set the step
  rowVector <- vector("list")

```

```

    i <- 1
    # stop loop when the ball hits the ground, state[1] is the vertical position
    while (getState(ball)[1] > 0) {
      rowVector[[i]] <- list(t = getState(ball)[3],
                            y = getState(ball)[1],
                            vy = getState(ball)[2])
      solver <- step(solver) # move one step at a time
      ball <- getODE(solver) # update the ball state
      i <- i + 1
    }
    DT <- data.table::rbindlist(rowVector)
    return(DT)
  }
# show solution
solution <- FallingParticleODEApp()
plot(solution)
# KeplerVerlet.R

setClass("Kepler", slots = c(
  GM = "numeric",
  odeSolver = "Euler",
  counter = "numeric"
),
  contains = c("ODE")
)

setMethod("initialize", "Kepler", function(.Object, ...) {
  .Object@GM <- 4 * pi * pi # gravitation constant times combined mass
  .Object@state <- vector("numeric", 5) # x, vx, y, vy, t
  .Object@odeSolver <- Euler(.Object)
  .Object@counter <- 0
  return(.Object)
})

setMethod("doStep", "Kepler", function(object, ...) {
  # cat("state@doStep=", object@state, "\n")
  object@odeSolver <- step(object@odeSolver)

  object@state <- object@odeSolver@ode@state

  # object@rate <- object@odeSolver@ode@rate
  # cat("\t", object@odeSolver@ode@state)
  object
})

setMethod("getTime", "Kepler", function(object, ...) {
  return(object@state[5])
})

setMethod("getEnergy", "Kepler", function(object, ...) {
  ke <- 0.5 * (object@state[2] * object@state[2] +

```

```

        object@state[4] * object@state[4])
    pe <- -object@GM / sqrt(object@state[1] * object@state[1] +
        object@state[3] * object@state[3])
    return(pe+ke)
  })

setMethod("init", "Kepler", function(object, initState, ...) {
  object@state <- initState
  object@odeSolver <- init(object@odeSolver, getStepSize(object@odeSolver))
  object@counter <- 0
  object
})

setReplaceMethod("init", "Kepler", function(object, ..., value) {
  object@state <- value
  object@odeSolver <- init(object@odeSolver, getStepSize(object@odeSolver))
  object@counter <- 0
  object
})

setMethod("getRate", "Kepler", function(object, state, ...) {
  # Computes the rate using the given state.
  r2 <- state[1] * state[1] + state[3] * state[3] # distance squared
  r3 <- r2 * sqrt(r2) # distance cubed
  object@rate[1] <- state[2]
  object@rate[2] <- (- object@GM * state[1]) / r3
  object@rate[3] <- state[4]
  object@rate[4] <- (- object@GM * state[3]) / r3
  object@rate[5] <- 1 # time derivative

  # object@state <- object@odeSolver@ode@state <- state
  # object@state <- state
  object@counter <- object@counter + 1
  object@rate
})

setMethod("getState", "Kepler", function(object, ...) {
  # Gets the state variables.
  return(object@state)
})

# constructor
Kepler <- function() {
  kepler <- new("Kepler")
  return(kepler)
}
# ++++++ example: PlanetApp.R
# Simulation of Earth orbiting around the Sun using the Euler ODE solver

importFromExamples("Planet.R") # source the class

```

```

PlanetApp <- function(verbose = FALSE) {
  # x = 1, AU or Astronomical Units. Length of semimajor axis or the orbit
  # of the Earth around the Sun.
  x <- 1; vx <- 0; y <- 0; vy <- 6.28; t <- 0
  state <- c(x, vx, y, vy, t)
  dt <- 0.01
  planet <- Planet()
  planet@odeSolver <- setStepSize(planet@odeSolver, dt)
  planet <- init(planet, initState = state)
  rowvec <- vector("list")
  i <- 1
  # run infinite loop. stop with ESCAPE.
  while (getState(planet)[5] <= 90) { # Earth orbit is 365 days around the sun
    rowvec[[i]] <- list(t = getState(planet)[5], # just doing 3 months
                       x = getState(planet)[1], # to speed up for CRAN
                       vx = getState(planet)[2],
                       y = getState(planet)[3],
                       vy = getState(planet)[4])
    for (j in 1:5) { # advances time
      planet <- doStep(planet)
    }
    i <- i + 1
  }
  DT <- data.table::rbindlist(rowvec)
  return(DT)
}
# run the application
solution <- PlanetApp()
select_rows <- seq(1, nrow(solution), 10) # do not overplot
solution <- solution[select_rows,]
plot(solution)

# ++++++ application: RigidBodyNXFApp.R
# example of a nonstiff system is the system of equations describing
# the motion of a rigid body without external forces.

importFromExamples("RigidBody.R")

# run the application
RigidBodyNXFApp <- function(verbose = FALSE) {
  # load the R class that sets up the solver for this application
  y1 <- 0 # initial y1 value
  y2 <- 1 # initial y2 value
  y3 <- 1 # initial y3 value
  dt <- 0.01 # delta time for step

  body <- RigidBodyNXF(y1, y2, y3)
  solver <- Euler(body)
  solver <- setStepSize(solver, dt)
  rowVector <- vector("list")
  i <- 1
  # stop loop when the body hits the ground
  while (getState(body)[4] <= 12) {

```

```

        rowVector[[i]] <- list(t = getState(body)[4],
                              y1 = getState(body)[1],
                              y2 = getState(body)[2],
                              y3 = getState(body)[3])
        solver <- step(solver)
        body <- getODE(solver)
        i <- i + 1
    }
    DT <- data.table::rbindlist(rowVector)
    return(DT)
}

# get the data table from the app
solution <- RigidBodyNXFApp()
plot(solution)

```

---

EulerRichardson-class *EulerRichardson ODE solver class*

---

## Description

EulerRichardson ODE solver class

EulerRichardson generic

EulerRichardson constructor ODE

## Usage

EulerRichardson(ode, ...)

## S4 method for signature 'EulerRichardson'  
init(object, stepSize, ...)

## S4 method for signature 'EulerRichardson'  
step(object, ...)

## S4 method for signature 'ODE'  
EulerRichardson(ode, ...)

## Arguments

ode	an ODE object
...	additional parameters
object	internal passing object
stepSize	the size of the step



**Examples**

```

# ++++++ example: PendulumApp.R
# Simulation of a pendulum using the EulerRichardson ODE solver

suppressPackageStartupMessages(library(ggplot2))

importFromExamples("Pendulum.R")      # source the class

PendulumApp <- function(verbose = FALSE) {
  # initial values
  theta <- 0.2
  thetaDot <- 0
  dt <- 0.1
  pendulum <- Pendulum()
  # pendulum@state[3] <- 0      # set time to zero, t = 0
  pendulum <- setState(pendulum, theta, thetaDot)
  pendulum <- setStepSize(pendulum, dt = dt) # using stepSize in RK4
  pendulum@odeSolver <- setStepSize(pendulum@odeSolver, dt) # set new step size
  rowvec <- vector("list")
  i <- 1
  while (getState(pendulum)[3] <= 40) {
    rowvec[[i]] <- list(t      = getState(pendulum)[3], # time
                       theta  = getState(pendulum)[1], # angle
                       thetadot = getState(pendulum)[2]) # derivative of angle
    pendulum <- step(pendulum)
    i <- i + 1
  }
  DT <- data.table::rbindlist(rowvec)
  return(DT)
}
# show solution
solution <- PendulumApp()
plot(solution)

```

---

getEnergy

*getEnergy*


---

**Description**

Get the calculated energy level

**Usage**

```
getEnergy(object, ...)
```

**Arguments**

object	a class object
...	additional parameters

**Examples**

```

# KeplerEnergy.R
#

setClass("KeplerEnergy", slots = c(
  GM      = "numeric",
  odeSolver = "Verlet",
  counter  = "numeric"
),
  contains = c("ODE")
)

setMethod("initialize", "KeplerEnergy", function(.Object, ...) {
  .Object@GM <- 4 * pi * pi      # gravitation constant times combined mass
  .Object@state <- vector("numeric", 5) # x, vx, y, vy, t
  # .Object@odeSolver <- Verlet(ode = .Object)
  .Object@odeSolver <- Verlet(.Object)
  .Object@counter <- 0
  return(.Object)
})

setMethod("doStep", "KeplerEnergy", function(object, ...) {
  object@odeSolver <- step(object@odeSolver)
  object@state <- object@odeSolver@ode@state
  object
})

setMethod("getTime", "KeplerEnergy", function(object, ...) {
  return(object@state[5])
})

setMethod("getEnergy", "KeplerEnergy", function(object, ...) {
  ke <- 0.5 * (object@state[2] * object@state[2] +
    object@state[4] * object@state[4])
  pe <- -object@GM / sqrt(object@state[1] * object@state[1] +
    object@state[3] * object@state[3])
  return(pe+ke)
})

setMethod("init", "KeplerEnergy", function(object, initState, ...) {
  object@state <- initState
  object@odeSolver <- init(object@odeSolver, getStepSize(object@odeSolver))
  object@counter <- 0
  object
})

setReplaceMethod("init", "KeplerEnergy", function(object, ..., value) {
  initState <- value
  object@state <- initState
  object@odeSolver <- init(object@odeSolver, getStepSize(object@odeSolver))
})

```

```

    object@counter <- 0
    object
  })

  setMethod("getRate", "KeplerEnergy", function(object, state, ...) {
    # Computes the rate using the given state.
    r2 <- state[1] * state[1] + state[3] * state[3] # distance squared
    r3 <- r2 * sqrt(r2) # distance cubed
    object@rate[1] <- state[2]
    object@rate[2] <- (- object@GM * state[1]) / r3
    object@rate[3] <- state[4]
    object@rate[4] <- (- object@GM * state[3]) / r3
    object@rate[5] <- 1 # time derivative

    object@counter <- object@counter + 1
    object@rate

  })

  setMethod("getState", "KeplerEnergy", function(object, ...) {
    # Gets the state variables.
    return(object@state)
  })

  # constructor
  KeplerEnergy <- function() {
    kepler <- new("KeplerEnergy")
    return(kepler)
  }

```

---

getErrorCode

*getErrorCode*


---

### Description

Get an error code

### Usage

```
getErrorCode(object, tol, ...)
```

### Arguments

object	a class object
tol	tolerance
...	additional parameters

**Examples**

```
setMethod("getErrorCode", "DormandPrince45", function(object) {
  return(object@error_code)
})
```

---

```
getExactSolution      getExactSolution
```

---

**Description**

Compare analytical and calculated solutions

**Usage**

```
getExactSolution(object, t, ...)
```

**Arguments**

object	a class object
t	time at which we are performing the evaluation
...	additional parameters

**Examples**

```
# ++++++ example: ComparisonRK45App.R
# Compares the solution by the RK45 ODE solver versus the analytical solution
# Example file: ComparisonRK45App.R
# ODE Solver: Runge-Kutta 45
# ODE class : RK45
# Base class: ODETest

importFromExamples("ODETest.R")

ComparisonRK45App <- function(verbose = FALSE) {
  ode <- new("ODETest")           # create an `ODETest` object
  ode_solver <- RK45(ode)         # select the ODE solver
  ode_solver <- setStepSize(ode_solver, 1) # set the step

  # Two ways of setting the tolerance
  # ode_solver <- setTolerance(ode_solver, 1e-8) # set the tolerance
  setTolerance(ode_solver) <- 1e-8

  time <- 0
  rowVector <- vector("list")
  i <- 1
  while (time < 50) {
    rowVector[[i]] <- list(t = getState(ode)[2],
                          s1 = getState(ode)[1],
```

```

        s2 = getState(ode)[2],
        xs = getExactSolution(ode, time),
        counts = getRateCounts(ode),
        time = time
    )
    ode_solver <- step(ode_solver)      # advance one step
    stepSize <- getStepSize(ode_solver)
    time <- time + stepSize
    ode <- getODE(ode_solver)          # get updated ODE object
    i <- i + 1
}
return(data.table::rbindlist(rowVector)) # a data table with the results
}
# show solution
solution <- ComparisonRK45App()        # run the example
plot(solution)
# ODETest.R
# Called as base class for examples:
#           ComparisonRK45App.R
#           ComparisonRK45ODEApp.R

#' ODETest as an example of ODE class inheritance
#'
#' ODETest is a base class for examples ComparisonRK45App.R and
#' ComparisonRK45ODEApp.R. ODETest also uses an environment variable to store
#' the rate counts.
#'
#' @rdname ODE-class-example
#' @include ODE.R
setClass("ODETest", slots = c(
  n      = "numeric",      # counts the number of getRate evaluations
  stack = "environment"   # environment object to accumulate rate counts
),
  contains = c("ODE")
)

setMethod("initialize", "ODETest", function(.Object, ...) {
  .Object@stack$rateCounts <- 0      # counter for rate calculations
  .Object@state <- c(5.0, 0.0)
  return(.Object)
})

#' @rdname getExactSolution-method
setMethod("getExactSolution", "ODETest", function(object, t, ...) {
  return(5.0 * exp(-t))
})

#' @rdname getState-method
setMethod("getState", "ODETest", function(object, ...) {
  object@state
})

#' @rdname getRate-method

```

```

setMethod("getRate", "ODETest", function(object, state, ...) {
  object@rate[1] <- - state[1]
  object@rate[2] <- 1          # rate of change of time, dt/dt
  # accumulate how many times the rate has been called to calculate
  object@stack$rateCounts <- object@stack$rateCounts + 1
  object@state <- state
  object@rate
})

#' @rdname getRateCounts-method
setMethod("getRateCounts", "ODETest", function(object, ...) {
  # use environment stack to accumulate rate counts
  object@stack$rateCounts
})

# constructor
ODETest <- function() {
  odetest <- new("ODETest")
  odetest
}

```

---

getODE

*getODE*


---

### Description

Get the ODE status from the solver

### Usage

```
getODE(object, ...)
```

### Arguments

object	a class object
...	additional parameters

---

getRate

*getRate*


---

### Description

Get a new rate given a state

### Usage

```
getRate(object, state, ...)
```

**Arguments**

object	a class object
state	current state
...	additional parameters

**Examples**

```
# Kepler models Keplerian orbits of a mass moving under the influence of an
# inverse square force by implementing the ODE interface.
# Kepler.R
#

setClass("Kepler", slots = c(
  GM = "numeric"
),
  contains = c("ODE")
)

setMethod("initialize", "Kepler", function(.Object, ...) {
  .Object@GM <- 1.0 # gravitation constant times combined mass
  .Object@state <- vector("numeric", 5) # x, vx, y, vy, t
  return(.Object)
})

setMethod("getState", "Kepler", function(object, ...) {
  # Gets the state variables.
  return(object@state)
})

setMethod("getRate", "Kepler", function(object, state, ...) {
  # Computes the rate using the given state.
  r2 <- state[1] * state[1] + state[3] * state[3] # distance squared
  r3 <- r2 * sqrt(r2) # distance cubed
  object@rate[1] <- state[2]
  object@rate[2] <- (- object@GM * state[1]) / r3
  object@rate[3] <- state[4]
  object@rate[4] <- (- object@GM * state[3]) / r3
  object@rate[5] <- 1 # time derivative

  object@rate
})

# constructor
Kepler <- function(r, v) {
  kepler <- new("Kepler")
  kepler@state[1] = r[1]
  kepler@state[2] = v[1]
  kepler@state[3] = r[2]
  kepler@state[4] = v[2]
  kepler@state[5] = 0
}
```

```

    return(kepler)
}

```

---

getRateCounter	<i>getRateCounter</i>
----------------	-----------------------

---

## Description

Get the rate counter

## Usage

```
getRateCounter(object, ...)
```

## Arguments

object	a class object
...	additional parameters

## Details

How many times the rate has changed with a step

## Examples

```

# ++++++ example: ComparisonRK45App.R
# Compares the solution by the RK45 ODE solver versus the analytical solution
# Example file: ComparisonRK45App.R
# ODE Solver: Runge-Kutta 45
# ODE class : RK45
# Base class: ODETest

importFromExamples("ODETest.R")

ComparisonRK45App <- function(verbose = FALSE) {
  ode <- new("ODETest")           # create an `ODETest` object
  ode_solver <- RK45(ode)         # select the ODE solver
  ode_solver <- setStepSize(ode_solver, 1) # set the step

  # Two ways of setting the tolerance
  # ode_solver <- setTolerance(ode_solver, 1e-8) # set the tolerance
  setTolerance(ode_solver) <- 1e-8

  time <- 0
  rowVector <- vector("list")
  i <- 1
  while (time < 50) {
    rowVector[[i]] <- list(t = getState(ode)[2],
                          s1 = getState(ode)[1],

```



```

        s2 = getState(ode)[2],
        xs = getExactSolution(ode, time),
        counts = getRateCounts(ode),
        time = time
    )
    ode_solver <- step(ode_solver)      # advance one step
    stepSize <- getStepSize(ode_solver)
    time <- time + stepSize
    ode <- getODE(ode_solver)          # get updated ODE object
    i <- i + 1
  }
  return(data.table::rbindlist(rowVector)) # a data table with the results
}
# show solution
solution <- ComparisonRK45App()      # run the example
plot(solution)

```

---

getRateCounts	<i>getRateCounts</i>
---------------	----------------------

---

### Description

Get the number of times that the rate has been calculated

### Usage

```
getRateCounts(object, ...)
```

### Arguments

object	a class object
...	additional parameters

---

getState	<i>getState</i>
----------	-----------------

---

### Description

Get current state of the system

### Usage

```
getState(object, ...)
```

**Arguments**

object            a class object  
 ...               additional parameters

**Examples**

```
# ++++++ application: VanderPolApp.R
# Solution of the Van der Pol equation
#
importFromExamples("VanderPol.R")

# run the application
VanderpolApp <- function(verbose = FALSE) {
  # set the orbit into a predefined state.
  y1 <- 2; y2 <- 0; dt <- 0.1;
  rigid_body <- VanderPol(y1, y2)
  solver <- RK45(rigid_body)
  rowVector <- vector("list")
  i <- 1
  while (getState(rigid_body)[3] <= 20) {
    rowVector[[i]] <- list(t = getState(rigid_body)[3],
                          y1 = getState(rigid_body)[1],
                          y2 = getState(rigid_body)[2])
    solver <- step(solver)
    rigid_body <- getODE(solver)
    i <- i + 1
  }
  DT <- data.table::rbindlist(rowVector)
  return(DT)
}

# show solution
solution <- VanderpolApp()
plot(solution)

# ++++++ application: SpringRK4App.R
# Simulation of a spring considering no friction

importFromExamples("SpringRK4.R")

# run application
SpringRK4App <- function(verbose = FALSE) {
  theta <- 0
  thetaDot <- -0.2
  tmax <- 22; dt <- 0.1
  spring <- SpringRK4()
  spring@state[3] <- 0 # set time to zero, t = 0
  spring <- setState(spring, theta, thetaDot)
  # spring <- setStepSize(spring, dt = dt) # using stepSize in RK4
  spring@odeSolver <- setStepSize(spring@odeSolver, dt) # set new step size
  rowvec <- vector("list")
```

```

    i <- 1
    while (getState(spring)[3] <= tmax) {
      rowvec[[i]] <- list(t = getState(spring)[3], # angle
                        y1 = getState(spring)[1], # derivative of the angle
                        y2 = getState(spring)[2]) # time
      i <- i + 1
      spring <- step(spring)
    }
    DT <- data.table::rbindlist(rowvec)
    return(DT)
  }

# show solution
solution <- SpringRK4App()
plot(solution)

```

---

getStepSize

*getStepSize*


---

### Description

Get the current value of the step size

### Usage

```
getStepSize(object, ...)
```

### Arguments

object	a class object
...	additional parameters

### Examples

```

# ++++++ Example: ComparisonRK45ODEApp.R
# Updates the ODE state instead of using the internal state in the ODE solver
# Also plots the solver solution versus the analytical solution at a
# tolerance of 1e-6
# Example file: ComparisonRK45ODEApp.R
# ODE Solver: Runge-Kutta 45
# ODE class : RK45
# Base class: ODETest

library(ggplot2)
library(dplyr)
library(tidyr)

importFromExamples("ODETest.R")

ComparisonRK45ODEApp <- function(verbose = FALSE) {

```

```

ode <- new("ODETest")                # new ODE instance
ode_solver <- RK45(ode)               # select ODE solver
ode_solver <- setStepSize(ode_solver, 1) # set the step

# two ways to set tolerance
# ode_solver <- setTolerance(ode_solver, 1e-6)
setTolerance(ode_solver) <- 1e-6

time <- 0
rowVector <- vector("list")          # row vector
i <- 1 # counter
while (time < 50) {
  # add solution objects to a row vector
  rowVector[[i]] <- list(t = getState(ode)[2],
                        ODE = getState(ode)[1],
                        s2 = getState(ode)[2],
                        exact = getExactSolution(ode, time),
                        rate.counts = getRateCounts(ode),
                        time = time )

  ode_solver <- step(ode_solver)      # advance solver one step
  stepSize <- getStepSize(ode_solver) # get the current step size
  time <- time + stepSize
  ode <- getODE(ode_solver)           # get updated ODE object
  state <- getState(ode)              # get the `state` vector
  i <- i + 1                          # add a row vector
}
DT <- data.table::rbindlist(rowVector) # create data table
return(DT)
}

solution <- ComparisonRK45ODEApp()
plot(solution)

# additional plot for analytics solution vs. RK45 solver
solution.multi <- solution %>%
  select(t, ODE, exact)
plot(solution.multi)                # 3x3 plot

# plot comparative curves analytical vs ODE solver
solution.2x1 <- solution.multi %>%
  gather(key, value, -t)            # make a table of 3 variables. key: ODE/exact

g <- ggplot(solution.2x1, mapping = aes(x = t, y = value, color = key))
g <- g + geom_line(size = 1) +
  labs(title = "ODE vs Exact solution",
        subtitle = "tolerance = 1E-6")
print(g)

```

---

getTime	<i>getTime</i>
---------	----------------

---

## Description

Get the elapsed time

## Usage

```
getTime(object, ...)
```

## Arguments

object	a class object
...	additional parameters

## Examples

```
# ++++++ application: Logistic.R
# Simulates the logistic equation
importFromExamples("Logistic.R")

# Run the application
LogisticApp <- function(verbose = FALSE) {
  x <- 0.1
  vx <- 0
  r <- 2      # Malthusian parameter (rate of maximum population growth)
  K <- 10.0   # carrying capacity of the environment
  dt <- 0.01; tol <- 1e-3; tmax <- 10

  population <- Logistic()      # create a Logistic ODE object

  # Two ways of initializing the object
  # population <- init(population, c(x, vx, 0), r, K)
  init(population) <- list(initState = c(x, vx, 0),
                           r = r,
                           K = K)

  odeSolver <- Verlet(population)      # select the solver

  # Two ways of initializing the solver
  # odeSolver <- init(odeSolver, dt)
  init(odeSolver) <- dt

  population@odeSolver <- odeSolver
  # setSolver(population) <- odeSolver

  rowVector <- vector("list")
  i <- 1
```

```

while (getTime(population) <= tmax) {
  rowVector[[i]] <- list(t = getTime(population),
                        s1 = getState(population)[1],
                        s2 = getState(population)[2])
  population <- doStep(population)
  i <- i + 1
}
DT <- data.table::rbindlist(rowVector)
return(DT)
}
# show solution
solution <- LogisticApp()
plot(solution)
# KeplerEnergy.R
#

setClass("KeplerEnergy", slots = c(
  GM      = "numeric",
  odeSolver = "Verlet",
  counter  = "numeric"
),
  contains = c("ODE")
)

setMethod("initialize", "KeplerEnergy", function(.Object, ...) {
  .Object@GM <- 4 * pi * pi # gravitation constant times combined mass
  .Object@state <- vector("numeric", 5) # x, vx, y, vy, t
  # .Object@odeSolver <- Verlet(ode = .Object)
  .Object@odeSolver <- Verlet(.Object)
  .Object@counter <- 0
  return(.Object)
})

setMethod("doStep", "KeplerEnergy", function(object, ...) {
  object@odeSolver <- step(object@odeSolver)
  object@state <- object@odeSolver@ode@state
  object
})

setMethod("getTime", "KeplerEnergy", function(object, ...) {
  return(object@state[5])
})

setMethod("getEnergy", "KeplerEnergy", function(object, ...) {
  ke <- 0.5 * (object@state[2] * object@state[2] +
              object@state[4] * object@state[4])
  pe <- -object@GM / sqrt(object@state[1] * object@state[1] +
                          object@state[3] * object@state[3])
  return(pe+ke)
})

setMethod("init", "KeplerEnergy", function(object, initState, ...) {

```

```

    object@state <- initState
    object@odeSolver <- init(object@odeSolver, getStepSize(object@odeSolver))
    object@counter <- 0
    object
  })

  setReplaceMethod("init", "KeplerEnergy", function(object, ..., value) {
    initState <- value
    object@state <- initState
    object@odeSolver <- init(object@odeSolver, getStepSize(object@odeSolver))
    object@counter <- 0
    object
  })

  setMethod("getRate", "KeplerEnergy", function(object, state, ...) {
    # Computes the rate using the given state.
    r2 <- state[1] * state[1] + state[3] * state[3] # distance squared
    r3 <- r2 * sqrt(r2) # distance cubed
    object@rate[1] <- state[2]
    object@rate[2] <- (- object@GM * state[1]) / r3
    object@rate[3] <- state[4]
    object@rate[4] <- (- object@GM * state[3]) / r3
    object@rate[5] <- 1 # time derivative

    object@counter <- object@counter + 1
    object@rate
  })

  setMethod("getState", "KeplerEnergy", function(object, ...) {
    # Gets the state variables.
    return(object@state)
  })

  # constructor
  KeplerEnergy <- function() {
    kepler <- new("KeplerEnergy")
    return(kepler)
  }
}

```

---

getTolerance

*getTolerance*


---

### Description

Get the tolerance for the solver

**Usage**

```
getTolerance(object, ...)
```

**Arguments**

object	a class object
...	additional parameters

---

```
importFromExamples     importFromExamples
```

---

**Description**

Source the R script

**Usage**

```
importFromExamples(aClassFile, aFolder = "examples")
```

**Arguments**

aClassFile	a file containing one or more classes
aFolder	a folder where examples are located

---

```
init                     init
```

---

**Description**

Set initial values before starting the ODE solver

Set initial values before starting the ODE solver

**Usage**

```
init(object, ...)
```

```
init(object, ...) <- value
```

**Arguments**

object	a class object
...	additional parameters
value	a value to set



**Details**

Sets the tolerance like this: `solver <- init(solver, dt)` Not all super classes require an `init` method.

Sets the tolerance like this: `init(solver) <- dt`

**Examples**

```
# init method in Kepler.R
setMethod("init", "Kepler", function(object, initState, ...) {
  object@state <- initState
  object@odeSolver <- init(object@odeSolver, getStepSize(object@odeSolver))
  object@counter <- 0
  object
})

# init method in LogisticApp.R
setMethod("init", "Logistic", function(object, initState, r, K, ...) {
  object@r <- r
  object@K <- K
  object@state <- initState
  object@odeSolver <- init(object@odeSolver, getStepSize(object@odeSolver))
  object@counter <- 0
  object
})

# init method in Planet.R
setMethod("init", "Planet", function(object, initState, ...) {
  object@state <- object@odeSolver@ode@state <- initState
  # initialize providing the step size
  object@odeSolver <- init(object@odeSolver, getStepSize(object@odeSolver))
  object@rate <- object@odeSolver@ode@rate
  object@state <- object@odeSolver@ode@state
  object
})
```

---

ODE-class

*ODE class*

---

**Description**

Defines an ODE object for any solver

ODE constructor

**Usage**

`ODE()`

```
## S4 method for signature 'ODE'
getState(object, ...)
```

```
## S4 method for signature 'ODE'
getRate(object, state, ...)
```

### Arguments

```
object      a class object
...         additional parameters
state      current state
```

### Examples

```
# ++++++ example: PendulumApp.R
# Simulation of a pendulum using the EulerRichardson ODE solver

suppressPackageStartupMessages(library(ggplot2))

importFromExamples("Pendulum.R") # source the class

PendulumApp <- function(verbose = FALSE) {
  # initial values
  theta <- 0.2
  thetaDot <- 0
  dt <- 0.1
  pendulum <- Pendulum()
  # pendulum@state[3] <- 0 # set time to zero, t = 0
  pendulum <- setState(pendulum, theta, thetaDot)
  pendulum <- setStepSize(pendulum, dt = dt) # using stepSize in RK4
  pendulum@odeSolver <- setStepSize(pendulum@odeSolver, dt) # set new step size
  rowvec <- vector("list")
  i <- 1
  while (getState(pendulum)[3] <= 40) {
    rowvec[[i]] <- list(t = getState(pendulum)[3], # time
                      theta = getState(pendulum)[1], # angle
                      thetadot = getState(pendulum)[2]) # derivative of angle
    pendulum <- step(pendulum)
    i <- i + 1
  }
  DT <- data.table::rbindlist(rowvec)
  return(DT)
}
# show solution
solution <- PendulumApp()
plot(solution)
# ++++++ example: PendulumEulerApp.R
# Pendulum simulation with the Euler ODE solver
# Notice how Euler is not applicable in this case as it diverges very quickly
# even when it is using a very small `delta t`?ODE

importFromExamples("PendulumEuler.R") # source the class

PendulumEulerApp <- function(verbose = FALSE) {
```

```

# initial values
theta <- 0.2
thetaDot <- 0
dt <- 0.01
pendulum <- PendulumEuler()
pendulum@state[3] <- 0 # set time to zero, t = 0
pendulum <- setState(pendulum, theta, thetaDot)
stepSize <- dt
pendulum <- setStepSize(pendulum, stepSize)
pendulum@odeSolver <- setStepSize(pendulum@odeSolver, dt) # set new step size
rowvec <- vector("list")
i <- 1
while (getState(pendulum)[3] <= 50) {
  rowvec[[i]] <- list(t = getState(pendulum)[3],
                    theta = getState(pendulum)[1],
                    thetaDot = getState(pendulum)[2])
  pendulum <- step(pendulum)
  i <- i + 1
}
DT <- data.table::rbindlist(rowvec)
return(DT)
}

solution <- PendulumEulerApp()
plot(solution)
# ++++++ example KeplerApp.R
# KeplerApp solves an inverse-square law model (Kepler model) using an adaptive
# stepsize algorithm.
# Application showing two planet orbiting
# File in examples: KeplerApp.R

importFromExamples("Kepler.R") # source the class Kepler

KeplerApp <- function(verbose = FALSE) {

  # set the orbit into a predefined state.
  r <- c(2, 0) # orbit radius
  v <- c(0, 0.25) # velocity
  dt <- 0.1
  planet <- Kepler(r, v) # make up an ODE object
  solver <- RK45(planet)
  rowVector <- vector("list")
  i <- 1
  while (getState(planet)[5] <= 10) {
    rowVector[[i]] <- list(t = planet@state[5],
                        planet1.r = getState(planet)[1],
                        planet1.v = getState(planet)[2],
                        planet2.r = getState(planet)[3],
                        planet2.v = getState(planet)[4])
    solver <- step(solver)
    planet <- getODE(solver)
    i <- i + 1
  }
}

```

```

    DT <- data.table::rbindlist(rowVector)

    return(DT)
}

solution <- KeplerApp()
plot(solution)

# ~~~~~ base class: FallingParticleODE.R
# Class definition for application FallingParticleODEApp.R

setClass("FallingParticleODE", slots = c(
  g = "numeric"
),
  prototype = prototype(
    g = 9.8
  ),
  contains = c("ODE")
)

setMethod("initialize", "FallingParticleODE", function(.Object, ...) {
  .Object@state <- vector("numeric", 3)
  return(.Object)
})

setMethod("getState", "FallingParticleODE", function(object, ...) {
  # Gets the state variables.
  return(object@state)
})

setMethod("getRate", "FallingParticleODE", function(object, state, ...) {
  # Gets the rate of change using the argument's state variables.
  object@rate[1] <- state[2]
  object@rate[2] <- - object@g
  object@rate[3] <- 1

  object@rate
})

# constructor
FallingParticleODE <- function(y, v) {
  .FallingParticleODE <- new("FallingParticleODE")
  .FallingParticleODE@state[1] <- y
  .FallingParticleODE@state[2] <- v
  .FallingParticleODE@state[3] <- 0
  .FallingParticleODE
}

```

---

ODEAdaptiveSolver-class

*ODEAdaptiveSolver class*

### Description

Base class to be inherited by adaptive solvers such as RK45

ODEAdaptiveSolver generic

ODEAdaptiveSolver constructor

### Usage

ODEAdaptiveSolver(...)

## S4 method for signature 'ODEAdaptiveSolver'  
setTolerance(object, tol)

## S4 replacement method for signature 'ODEAdaptiveSolver'  
setTolerance(object, ...) <- value

## S4 method for signature 'ODEAdaptiveSolver'  
getTolerance(object)

## S4 method for signature 'ODEAdaptiveSolver'  
getErrorCode(object)

## S4 method for signature 'ANY'  
ODEAdaptiveSolver(...)

### Arguments

...	additional parameters
object	a class object
tol	tolerance
value	the value for the tolerance

ODESolver-class

*ODESolver virtual class*

### Description

A virtual class inherited by AbstractODESolver

ODESolver constructor

Set initial values and get ready to start the solver

Set the size of the step

**Usage**

```
ODESolver(object, stepSize, ...)  
  
## S4 method for signature 'ODESolver'  
init(object, stepSize, ...)  
  
## S4 method for signature 'ODESolver'  
step(object, ...)  
  
## S4 method for signature 'ODESolver'  
getODE(object, ...)  
  
## S4 method for signature 'ODESolver'  
setStepSize(object, stepSize, ...)  
  
## S4 method for signature 'ODESolver'  
getStepSize(object, ...)
```

**Arguments**

object	a class object
stepSize	size of the step
...	additional parameters

**See Also**

Other ODESolver helpers: [ODESolverFactory-class](#)

---

ODESolverFactory-class

*ODESolverFactory*

---

**Description**

ODESolverFactory helps to create a solver given only the name as string

ODESolverFactory generic

This is a factory method that creates an ODESolver using a name.

ODESolverFactory constructor

**Usage**

```
ODESolverFactory(...)  
  
createODESolver(object, ...)
```

```
## S4 method for signature 'ODESolverFactory'
createODESolver(object, ode, solverName, ...)
```

```
## S4 method for signature 'ANY'
ODESolverFactory(...)
```

### Arguments

```
...          additional parameters
object       an solver object
ode          an ODE object
solverName   the desired solver as a string
```

### See Also

Other ODESolver helpers: [ODESolver-class](#)

Other ODESolver helpers: [ODESolver-class](#)

### Examples

```
# This example uses ODESolverFactory

importFromExamples("SHO.R")

# SHOApp.R
SHOApp <- function(...) {
  x <- 1.0; v <- 0; k <- 1.0; dt <- 0.01; tolerance <- 1e-3
  sho <- SHO(x, v, k)

  # Use ODESolverFactory
  solver_factory <- ODESolverFactory()
  solver <- createODESolver(solver_factory, sho, "DormandPrince45")
  # solver <- DormandPrince45(sho) # this can also be used

  # Two ways of setting the tolerance
  # solver <- setTolerance(solver, tolerance) # or this below
  setTolerance(solver) <- tolerance

  # Two ways of initializing the solver
  # solver <- init(solver, dt)
  init(solver) <- dt

  i <- 1; rowVector <- vector("list")
  while (getState(sho)[3] <= 500) {
    rowVector[[i]] <- list(x = getState(sho)[1],
                          v = getState(sho)[2],
                          t = getState(sho)[3])
    solver <- step(solver)
    sho <- getODE(solver)
    i <- i + 1
  }
}
```

```

    }
    return(data.table::rbindlist(rowVector))
}

solution <- SHOApp()
plot(solution)

# This example uses ODESolverFactory

importFromExamples("SHO.R")

# SHOApp.R
SHOApp <- function(...) {
  x <- 1.0; v <- 0; k <- 1.0; dt <- 0.01; tolerance <- 1e-3
  sho <- SHO(x, v, k)

  # Use ODESolverFactory
  solver_factory <- ODESolverFactory()
  solver <- createODESolver(solver_factory, sho, "DormandPrince45")
  # solver <- DormandPrince45(sho) # this can also be used

  # Two ways of setting the tolerance
  # solver <- setTolerance(solver, tolerance) # or this below
  setTolerance(solver) <- tolerance

  # Two ways of initializing the solver
  # solver <- init(solver, dt)
  init(solver) <- dt

  i <- 1; rowVector <- vector("list")
  while (getState(sho)[3] <= 500) {
    rowVector[[i]] <- list(x = getState(sho)[1],
                          v = getState(sho)[2],
                          t = getState(sho)[3])

    solver <- step(solver)
    sho <- getODE(solver)
    i <- i + 1
  }
  return(data.table::rbindlist(rowVector))
}

solution <- SHOApp()
plot(solution)

```



**Description**

RK4 class  
 RK4 generic  
 RK4 class constructor

**Usage**

```
RK4(ode, ...)

## S4 method for signature 'RK4'
init(object, stepSize, ...)

## S4 replacement method for signature 'RK4'
init(object, ...) <- value

## S4 method for signature 'RK4'
step(object, ...)

## S4 method for signature 'ODE'
RK4(ode, ...)
```

**Arguments**

ode	an ODE object
...	additional parameters
object	internal passing object
stepSize	the size of the step
value	value for the step

**Examples**

```
# ~~~~~ base class: Projectile.R
# Projectile class to be solved with Euler method

setClass("Projectile", slots = c(
  g = "numeric",
  odeSolver = "RK4"
),
  prototype = prototype(
    g = 9.8
  ),
  contains = c("ODE")
)

setMethod("initialize", "Projectile", function(.Object) {
  .Object@odeSolver <- RK4(.Object)
  return(.Object)
})
```

```

}))

setMethod("setStepSize", "Projectile", function(object, stepSize, ...) {
  # use explicit parameter declaration
  # setStepSize generic has two step parameters: stepSize and dt
  object@odeSolver <- setStepSize(object@odeSolver, stepSize)
  object
}))

setMethod("step", "Projectile", function(object) {
  object@odeSolver <- step(object@odeSolver)
  object@rate <- object@odeSolver@odeRate
  object@state <- object@odeSolver@odeState
  object
}))

setMethod("setState", signature("Projectile"), function(object, x, vx, y, vy, ...) {
  object@state[1] <- x
  object@state[2] <- vx
  object@state[3] <- y
  object@state[4] <- vy
  object@state[5] <- 0 # t + dt
  object@odeSolver@odeState <- object@state
  object
}))

setMethod("getState", "Projectile", function(object) {
  object@state
}))

setMethod("getRate", "Projectile", function(object, state, ...) {
  object@rate[1] <- state[2] # rate of change of x
  object@rate[2] <- 0 # rate of change of vx
  object@rate[3] <- state[4] # rate of change of y
  object@rate[4] <- - object@g # rate of change of vy
  object@rate[5] <- 1 # dt/dt = 1

  object@rate
}))

# constructor
Projectile <- function() new("Projectile")
# ++++++ example: PendulumApp.R
# Simulation of a pendulum using the EulerRichardson ODE solver

suppressPackageStartupMessages(library(ggplot2))

importFromExamples("Pendulum.R") # source the class

```

```

PendulumApp <- function(verbose = FALSE) {
  # initial values
  theta <- 0.2
  thetaDot <- 0
  dt <- 0.1
  pendulum <- Pendulum()
  # pendulum@state[3] <- 0      # set time to zero, t = 0
  pendulum <- setState(pendulum, theta, thetaDot)
  pendulum <- setStepSize(pendulum, dt = dt) # using stepSize in RK4
  pendulum@odeSolver <- setStepSize(pendulum@odeSolver, dt) # set new step size
  rowvec <- vector("list")
  i <- 1
  while (getState(pendulum)[3] <= 40) {
    rowvec[[i]] <- list(t      = getState(pendulum)[3], # time
                      theta   = getState(pendulum)[1], # angle
                      thetadot = getState(pendulum)[2]) # derivative of angle
    pendulum <- step(pendulum)
    i <- i + 1
  }
  DT <- data.table::rbindlist(rowvec)
  return(DT)
}

# show solution
solution <- PendulumApp()
plot(solution)
# ++++++ application: ReactionApp.R
# ReactionApp solves an autocatalytic oscillating chemical
# reaction (Brusselator model) using
# a fourth-order Runge-Kutta algorithm.

importFromExamples("Reaction.R") # source the class

ReactionApp <- function(verbose = FALSE) {
  X <- 1; Y <- 5;
  dt <- 0.1

  reaction <- Reaction(c(X, Y, 0))
  solver <- RK4(reaction)
  rowvec <- vector("list")
  i <- 1
  while (getState(reaction)[3] < 100) { # stop at t = 100
    rowvec[[i]] <- list(t = getState(reaction)[3],
                      X = getState(reaction)[1],
                      Y = getState(reaction)[2])
    solver <- step(solver)
    reaction <- getODE(solver)
    i <- i + 1
  }
  DT <- data.table::rbindlist(rowvec)
  return(DT)
}

```



```

        counts = getRateCounts(ode),
        time   = time
      )
      ode_solver <- step(ode_solver)      # advance one step
      stepSize  <- getStepSize(ode_solver)
      time <- time + stepSize
      ode <- getODE(ode_solver)          # get updated ODE object
      i <- i + 1
    }
    return(data.table::rbindlist(rowVector)) # a data table with the results
  }
# show solution
solution <- ComparisonRK45App()          # run the example
plot(solution)
# ++++++ example KeplerApp.R
# KeplerApp solves an inverse-square law model (Kepler model) using an adaptive
# stepsize algorithm.
# Application showing two planet orbiting
# File in examples: KeplerApp.R

importFromExamples("Kepler.R") # source the class Kepler

KeplerApp <- function(verbose = FALSE) {

  # set the orbit into a predefined state.
  r <- c(2, 0)                          # orbit radius
  v <- c(0, 0.25)                        # velocity
  dt <- 0.1
  planet <- Kepler(r, v)                 # make up an ODE object
  solver <- RK45(planet)
  rowVector <- vector("list")
  i <- 1
  while (getState(planet)[5] <= 10) {
    rowVector[[i]] <- list(t = planet@state[5],
                          planet1.r = getState(planet)[1],
                          planet1.v = getState(planet)[2],
                          planet2.r = getState(planet)[3],
                          planet2.v = getState(planet)[4])

    solver <- step(solver)
    planet <- getODE(solver)
    i <- i + 1
  }
  DT <- data.table::rbindlist(rowVector)

  return(DT)
}

solution <- KeplerApp()
plot(solution)

```

---

```
run_test_applications  run_test_applications
```

---

**Description**

Run test all the examples

**Usage**

```
run_test_applications()
```

---

```
setSolver<-           setSolver
```

---

**Description**

Set a solver over an ODE object

**Usage**

```
setSolver(object) <- value
```

**Arguments**

object	a class object
value	value to be set

---

```
setState             setState
```

---

**Description**

New setState that should work with different methods "theta", "thetaDot": used in PendulumApp  
"x", "vx", "y", "vy": used in ProjectileApp

**Usage**

```
setState(object, ...)
```

**Arguments**

object	a class object
...	additional parameters

**Examples**

```

# ++++++ application: ProjectileApp.R
#                               test Projectile with RK4
#                               originally uses Euler

# suppressMessages(library(data.table))

importFromExamples("Projectile.R")    # source the class

ProjectileApp <- function(verbose = FALSE) {
  # initial values
  x <- 0; vx <- 10; y <- 0; vy <- 10
  state <- c(x, vx, y, vy, 0)          # state vector
  dt <- 0.01

  projectile <- Projectile()
  projectile <- setState(projectile, x, vx, y, vy)

  projectile@odeSolver <- init(projectile@odeSolver, 0.123)

  # init(projectile) <- 0.123

  projectile@odeSolver <- setStepSize(projectile@odeSolver, dt)
  rowV <- vector("list")
  i <- 1
  while (getState(projectile)[3] >= 0) {
    rowV[[i]] <- list(t = getState(projectile)[5],
                     x = getState(projectile)[1],
                     vx = getState(projectile)[2],
                     y = getState(projectile)[3],      # vertical position
                     vy = getState(projectile)[4])
    projectile <- step(projectile)
    i <- i + 1
  }
  DT <- data.table::rbindlist(rowV)
  return(DT)
}

solution <- ProjectileApp()
plot(solution)
# ++++++ example: PendulumApp.R
# Simulation of a pendulum using the EulerRichardson ODE solver

suppressPackageStartupMessages(library(ggplot2))

importFromExamples("Pendulum.R")    # source the class

PendulumApp <- function(verbose = FALSE) {
  # initial values
  theta <- 0.2
  thetaDot <- 0

```

```

dt <- 0.1
pendulum <- Pendulum()
# pendulum@state[3] <- 0      # set time to zero, t = 0
pendulum <- setState(pendulum, theta, thetaDot)
pendulum <- setStepSize(pendulum, dt = dt) # using stepSize in RK4
pendulum@odeSolver <- setStepSize(pendulum@odeSolver, dt) # set new step size
rowvec <- vector("list")
i <- 1
while (getState(pendulum)[3] <= 40) {
  rowvec[[i]] <- list(t      = getState(pendulum)[3], # time
                    theta   = getState(pendulum)[1], # angle
                    thetadot = getState(pendulum)[2]) # derivative of angle
  pendulum <- step(pendulum)
  i <- i + 1
}
DT <- data.table::rbindlist(rowvec)
return(DT)
}
# show solution
solution <- PendulumApp()
plot(solution)

```

---

setStepSize

*setStepSize*


---

## Description

setStepSize uses either of two step parameters: stepSize and dt stepSize works for most of the applications dt is used in Pendulum

## Usage

```
setStepSize(object, ...)
```

## Arguments

object	a class object
...	additional parameters

## Examples

```

# ++++++application: SpringRK4App.R
# Simulation of a spring considering no friction

importFromExamples("SpringRK4.R")

# run application
SpringRK4App <- function(verbose = FALSE) {

```



```

theta    <- 0
thetaDot <- -0.2
tmax     <- 22; dt <- 0.1
spring <- SpringRK4()
spring@state[3] <- 0      # set time to zero, t = 0
spring <- setState(spring, theta, thetaDot)
# spring <- setStepSize(spring, dt = dt) # using stepSize in RK4
spring@odeSolver <- setStepSize(spring@odeSolver, dt) # set new step size
rowvec <- vector("list")
i <- 1
while (getState(spring)[3] <= tmax) {
  rowvec[[i]] <- list(t = getState(spring)[3],      # angle
                    y1 = getState(spring)[1],     # derivative of the angle
                    y2 = getState(spring)[2])     # time
  i <- i + 1
  spring <- step(spring)
}
DT <- data.table::rbindlist(rowvec)
return(DT)
}

# show solution
solution <- SpringRK4App()
plot(solution)
# ++++++ example: ComparisonRK45App.R
# Compares the solution by the RK45 ODE solver versus the analytical solution
# Example file: ComparisonRK45App.R
# ODE Solver: Runge-Kutta 45
# ODE class : RK45
# Base class: ODETest

importFromExamples("ODETest.R")

ComparisonRK45App <- function(verbose = FALSE) {
  ode <- new("ODETest")      # create an `ODETest` object
  ode_solver <- RK45(ode)    # select the ODE solver
  ode_solver <- setStepSize(ode_solver, 1)  # set the step

  # Two ways of setting the tolerance
  # ode_solver <- setTolerance(ode_solver, 1e-8) # set the tolerance
  setTolerance(ode_solver) <- 1e-8

  time <- 0
  rowVector <- vector("list")
  i <- 1
  while (time < 50) {
    rowVector[[i]] <- list(t = getState(ode)[2],
                          s1 = getState(ode)[1],
                          s2 = getState(ode)[2],
                          xs = getExactSolution(ode, time),
                          counts = getRateCounts(ode),
                          time = time
                          )
  }
}

```

```

        ode_solver <- step(ode_solver)           # advance one step
        stepSize  <- getStepSize(ode_solver)
        time <- time + stepSize
        ode <- getODE(ode_solver)               # get updated ODE object
        i <- i + 1
    }
    return(data.table::rbindlist(rowVector))    # a data table with the results
}
# show solution
solution <- ComparisonRK45App()               # run the example
plot(solution)

```

---

setTolerance                      *setTolerance*

---

### Description

Set the tolerance for the solver

Set the tolerance for the solver

### Usage

```
setTolerance(object, tol)
```

```
setTolerance(object, ...) <- value
```

### Arguments

object	a class object
tol	tolerance
...	additional parameters
value	a value to set

### Details

Sets the tolerance like this: `odeSolver <- setTolerance(odeSolver, tol)`

Sets the tolerance like this: `setTolerance(odeSolver) <- tol`

### Examples

```

# ++++++ example: ComparisonRK45App.R
# Compares the solution by the RK45 ODE solver versus the analytical solution
# Example file: ComparisonRK45App.R
# ODE Solver:   Runge-Kutta 45
# ODE class :   RK45
# Base class:   ODETest

importFromExamples("ODETest.R")

```

```

ComparisonRK45App <- function(verbose = FALSE) {
  ode <- new("ODETest")           # create an `ODETest` object
  ode_solver <- RK45(ode)         # select the ODE solver
  ode_solver <- setStepSize(ode_solver, 1) # set the step

  # Two ways of setting the tolerance
  # ode_solver <- setTolerance(ode_solver, 1e-8) # set the tolerance
  setTolerance(ode_solver) <- 1e-8

  time <- 0
  rowVector <- vector("list")
  i <- 1
  while (time < 50) {
    rowVector[[i]] <- list(t = getState(ode)[2],
                          s1 = getState(ode)[1],
                          s2 = getState(ode)[2],
                          xs = getExactSolution(ode, time),
                          counts = getRateCounts(ode),
                          time = time
                          )
    ode_solver <- step(ode_solver) # advance one step
    stepSize <- getStepSize(ode_solver)
    time <- time + stepSize
    ode <- getODE(ode_solver)     # get updated ODE object
    i <- i + 1
  }
  return(data.table::rbindlist(rowVector)) # a data table with the results
}

# show solution
solution <- ComparisonRK45App() # run the example
plot(solution)
# ++++++ example: KeplerDormandPrince45App.R
# Demonstration of the use of ODE solver RK45 for a particle subjected to
# a inverse-law force. The difference with the example KeplerApp is we are
# seeing the effect in the x and y axis on the particle.
# The original routine used the Verlet ODE solver

importFromExamples("KeplerDormandPrince45.R")

set_solver <- function(ode_object, solver) {
  slot(ode_object, "odeSolver") <- solver
  ode_object
}

KeplerDormandPrince45App <- function(verbose = FALSE) {
  # values for the examples
  x <- 1
  vx <- 0
  y <- 0
  vy <- 2 * pi
  dt <- 0.01 # step size
  tol <- 1e-3 # tolerance
}

```

```

particle <- KeplerDormandPrince45() # use class Kepler

# Two ways of initializing the ODE object
# particle <- init(particle, c(x, vx, y, vy, 0)) # enter state vector
init(particle) <- c(x, vx, y, vy, 0)

odeSolver <- DormandPrince45(particle) # select the ODE solver

# Two ways of initializing the solver
# odeSolver <- init(odeSolver, dt) # start the solver
init(odeSolver) <- dt

# Two ways of setting the tolerance
# odeSolver <- setTolerance(odeSolver, tol) # this works for adaptive solvers
setTolerance(odeSolver) <- tol
setSolver(particle) <- odeSolver

initialEnergy <- getEnergy(particle) # calculate the energy
rowVector <- vector("list")
i <- 1
while (getTime(particle) < 1.5) {
  rowVector[[i]] <- list(t = getState(particle)[5],
                        x = getState(particle)[1],
                        vx = getState(particle)[2],
                        y = getState(particle)[3],
                        vy = getState(particle)[4],
                        energy = getEnergy(particle) )
  particle <- doStep(particle) # advance one step
  energy <- getEnergy(particle) # calculate energy
  i <- i + 1
}
DT <- data.table::rbindlist(rowVector)
return(DT)
}

solution <- KeplerDormandPrince45App()
plot(solution)

importFromExamples("AdaptiveStep.R")

# running function
AdaptiveStepApp <- function(verbose = FALSE) {
  ode <- new("Impulse")
  ode_solver <- RK45(ode)

  # Two ways to initialize the solver
  # ode_solver <- init(ode_solver, 0.1)
  init(ode_solver) <- 0.1

  # two ways to set tolerance
  # ode_solver <- setTolerance(ode_solver, 1.0e-4)
  setTolerance(ode_solver) <- 1.0e-4
}

```

```

    i <- 1; rowVector <- vector("list")
    while (getState(ode)[1] < 12) {
      rowVector[[i]] <- list(s1 = getState(ode)[1],
                           s2 = getState(ode)[2],
                           t  = getState(ode)[3])
      ode_solver <- step(ode_solver)
      ode <- getODE(ode_solver)
      i <- i + 1
    }
    return(data.table::rbindlist(rowVector))
  }

# run application
solution <- AdaptiveStepApp()
plot(solution)

```

---

showMethods2

*showMethods2*


---

### Description

Get the methods in a class. But only those specific to the class

### Usage

```
showMethods2(theClass)
```

### Arguments

theClass      class to analyze

---

step

*step*


---

### Description

Advances a step within the ODE solver

### Usage

```
step(object, ...)
```

### Arguments

object      a class object  
...          additional parameters

**Examples**

```

# ++++++ application: ReactionApp.R
# ReactionApp solves an autocatalytic oscillating chemical
# reaction (Brusselator model) using
# a fourth-order Runge-Kutta algorithm.

importFromExamples("Reaction.R")      # source the class

ReactionApp <- function(verbose = FALSE) {
  X <- 1; Y <- 5;
  dt <- 0.1

  reaction <- Reaction(c(X, Y, 0))
  solver <- RK4(reaction)
  rowvec <- vector("list")
  i <- 1
  while (getState(reaction)[3] < 100) { # stop at t = 100
    rowvec[[i]] <- list(t = getState(reaction)[3],
                       X = getState(reaction)[1],
                       Y = getState(reaction)[2])
    solver <- step(solver)
    reaction <- getODE(solver)
    i <- i + 1
  }
  DT <- data.table::rbindlist(rowvec)
  return(DT)
}

solution <- ReactionApp()
plot(solution)

```

---

Verlet-class

*Verlet ODE solver class*


---

**Description**

Verlet ODE solver class

Verlet generic

Verlet class constructor ODE

**Usage**

Verlet(ode, ...)

```
## S4 method for signature 'Verlet'
init(object, stepSize, ...)
```

```
## S4 method for signature 'Verlet'
getRateCounter(object, ...)

## S4 method for signature 'Verlet'
step(object, ...)

## S4 method for signature 'ODE'
Verlet(ode, ...)
```

### Arguments

ode	an ODE object
...	additional parameters
object	a class object
stepSize	size of the step

### Examples

```
# ++++++ example: KeplerEnergyApp.R
# Demonstration of the use of the Verlet ODE solver
#

importFromExamples("KeplerEnergy.R") # source the class Kepler

KeplerEnergyApp <- function(verbose = FALSE) {
  # initial values
  x <- 1
  vx <- 0
  y <- 0
  vy <- 2 * pi
  dt <- 0.01
  tol <- 1e-3
  particle <- KeplerEnergy()

  # Two ways of initializing the ODE object
  # particle <- init(particle, c(x, vx, y, vy, 0))
  init(particle) <- c(x, vx, y, vy, 0)

  odeSolver <- Verlet(particle)

  # Two ways of initializing the solver
  # odeSolver <- init(odeSolver, dt)
  init(odeSolver) <- dt

  particle@odeSolver <- odeSolver
  initialEnergy <- getEnergy(particle)
  rowVector <- vector("list")
  i <- 1
  while (getTime(particle) <= 1.20) {
    rowVector[[i]] <- list(t = getState(particle)[5],
```

```

        x = getState(particle)[1],
        vx = getState(particle)[2],
        y = getState(particle)[3],
        vy = getState(particle)[4],
        E = getEnergy(particle)
    particle <- doStep(particle)
    energy <- getEnergy(particle)
    i <- i + 1
  }
  DT <- data.table::rbindlist(rowVector)
  return(DT)
}

solution <- KeplerEnergyApp()
plot(solution)

# ++++++ application: Logistic.R
# Simulates the logistic equation
importFromExamples("Logistic.R")

# Run the application
LogisticApp <- function(verbose = FALSE) {
  x <- 0.1
  vx <- 0
  r <- 2      # Malthusian parameter (rate of maximum population growth)
  K <- 10.0   # carrying capacity of the environment
  dt <- 0.01; tol <- 1e-3; tmax <- 10

  population <- Logistic()      # create a Logistic ODE object

  # Two ways of initializing the object
  # population <- init(population, c(x, vx, 0), r, K)
  init(population) <- list(initState = c(x, vx, 0),
                           r = r,
                           K = K)

  odeSolver <- Verlet(population)      # select the solver

  # Two ways of initializing the solver
  # odeSolver <- init(odeSolver, dt)
  init(odeSolver) <- dt

  population@odeSolver <- odeSolver
  # setSolver(population) <- odeSolver

  rowVector <- vector("list")
  i <- 1
  while (getTime(population) <= tmax) {
    rowVector[[i]] <- list(t = getTime(population),
                           s1 = getState(population)[1],
                           s2 = getState(population)[2])
    population <- doStep(population)
    i <- i + 1
  }
}

```



```

    }
    DT <- data.table::rbindlist(rowVector)
    return(DT)
  }
  # show solution
  solution <- LogisticApp()
  plot(solution)
  # ++++++ example: KeplerEnergyApp.R
  # Demonstration of the use of the Verlet ODE solver
  #

importFromExamples("KeplerEnergy.R") # source the class Kepler

KeplerEnergyApp <- function(verbose = FALSE) {
  # initial values
  x <- 1
  vx <- 0
  y <- 0
  vy <- 2 * pi
  dt <- 0.01
  tol <- 1e-3
  particle <- KeplerEnergy()

  # Two ways of initializing the ODE object
  # particle <- init(particle, c(x, vx, y, vy, 0))
  init(particle) <- c(x, vx, y, vy, 0)

  odeSolver <- Verlet(particle)

  # Two ways of initializing the solver
  # odeSolver <- init(odeSolver, dt)
  init(odeSolver) <- dt

  particle@odeSolver <- odeSolver
  initialEnergy <- getEnergy(particle)
  rowVector <- vector("list")
  i <- 1
  while (getTime(particle) <= 1.20) {
    rowVector[[i]] <- list(t = getState(particle)[5],
                          x = getState(particle)[1],
                          vx = getState(particle)[2],
                          y = getState(particle)[3],
                          vy = getState(particle)[4],
                          E = getEnergy(particle))

    particle <- doStep(particle)
    energy <- getEnergy(particle)
    i <- i + 1
  }
  DT <- data.table::rbindlist(rowVector)
  return(DT)
}

solution <- KeplerEnergyApp()

```

```

plot(solution)

# ++++++ application: Logistic.R
# Simulates the logistic equation
importFromExamples("Logistic.R")

# Run the application
LogisticApp <- function(verbose = FALSE) {
  x <- 0.1
  vx <- 0
  r <- 2      # Malthusian parameter (rate of maximum population growth)
  K <- 10.0   # carrying capacity of the environment
  dt <- 0.01; tol <- 1e-3; tmax <- 10

  population <- Logistic()      # create a Logistic ODE object

  # Two ways of initializing the object
  # population <- init(population, c(x, vx, 0), r, K)
  init(population) <- list(initState = c(x, vx, 0),
                           r = r,
                           K = K)

  odeSolver <- Verlet(population)      # select the solver

  # Two ways of initializing the solver
  # odeSolver <- init(odeSolver, dt)
  init(odeSolver) <- dt

  population@odeSolver <- odeSolver
  # setSolver(population) <- odeSolver

  rowVector <- vector("list")
  i <- 1
  while (getTime(population) <= tmax) {
    rowVector[[i]] <- list(t = getTime(population),
                          s1 = getState(population)[1],
                          s2 = getState(population)[2])
    population <- doStep(population)
    i <- i + 1
  }
  DT <- data.table::rbindlist(rowVector)
  return(DT)
}
# show solution
solution <- LogisticApp()
plot(solution)

```

# Index

- .AbstractODESolver
  - (AbstractODESolver-class), 3
- .DormandPrince45
  - (DormandPrince45-class), 4
- .Euler (Euler-class), 11
- .EulerRichardson
  - (EulerRichardson-class), 16
- .ODEAdaptiveSolver
  - (ODEAdaptiveSolver-class), 37
- .ODESolver (ODESolver-class), 37
- .ODESolverFactory
  - (ODESolverFactory-class), 38
- .RK4 (RK4-class), 40
- .Verlet (Verlet-class), 54
  
- AbstractODESolver
  - (AbstractODESolver-class), 3
- AbstractODESolver,missing-method
  - (AbstractODESolver-class), 3
- AbstractODESolver,ODE-method
  - (AbstractODESolver-class), 3
- AbstractODESolver-class, 3
  
- createODESolver
  - (ODESolverFactory-class), 38
- createODESolver,ODESolverFactory-method
  - (ODESolverFactory-class), 38
  
- DormandPrince45
  - (DormandPrince45-class), 4
- DormandPrince45,ODE-method
  - (DormandPrince45-class), 4
- DormandPrince45-class, 4
- doStep, 8
  
- enableRuntimeExceptions, 10
- enableRuntimeExceptions,DormandPrince45-method
  - (DormandPrince45-class), 4
- enableRuntimeExceptions,enableRuntimeExceptions-method
  - (DormandPrince45-class), 4
  
- Euler (Euler-class), 11
- Euler,missing-method (Euler-class), 11
- Euler,ODE-method (Euler-class), 11
- Euler-class, 11
- EulerRichardson
  - (EulerRichardson-class), 16
- EulerRichardson,ODE-method
  - (EulerRichardson-class), 16
- EulerRichardson-class, 16
  
- getEnergy, 17
- getErrorCode, 19
- getErrorCode,DormandPrince45-method
  - (DormandPrince45-class), 4
- getErrorCode,getErrorCode-method
  - (DormandPrince45-class), 4
- getErrorCode,ODEAdaptiveSolver-method
  - (ODEAdaptiveSolver-class), 37
- getExactSolution, 20
- getODE, 22
- getODE,AbstractODESolver-method
  - (AbstractODESolver-class), 3
- getODE,ODESolver-method
  - (ODESolver-class), 37
- getRate, 22
- getRate,getRate-method (ODE-class), 33
- getRate,ODE-method (ODE-class), 33
- getRateCounter, 24
- getRateCounter,getRateCounter-method
  - (Verlet-class), 54
- getRateCounter,Verlet-method
  - (Verlet-class), 54
- getRateCounts, 25
- getState, 25
- getState,getState-method (ODE-class), 33
- getState,ODE-method (ODE-class), 33
- getStepSize, 27
- getStepSize,AbstractODESolver-method
  - (AbstractODESolver-class), 3

- getStepSize,DormandPrince45-method  
(DormandPrince45-class), 4
- getStepSize,Euler-method (Euler-class),  
11
- getStepSize,getStepSize-method  
(Euler-class), 11
- getStepSize,ODESolver-method  
(ODESolver-class), 37
- getTime, 29
- getTolerance, 31
- getTolerance,DormandPrince45-method  
(DormandPrince45-class), 4
- getTolerance,getTolerance-method  
(DormandPrince45-class), 4
- getTolerance,ODEAdaptiveSolver-method  
(ODEAdaptiveSolver-class), 37
  
- importFromExamples, 32
- init, 32
- init,AbstractODESolver-method  
(AbstractODESolver-class), 3
- init,DormandPrince45-method  
(DormandPrince45-class), 4
- init,Euler-method (Euler-class), 11
- init,EulerRichardson-method  
(EulerRichardson-class), 16
- init,init-method (Euler-class), 11
- init,ODESolver-method  
(ODESolver-class), 37
- init,RK4-method (RK4-class), 40
- init,Verlet-method (Verlet-class), 54
- init-methods (Verlet-class), 54
- init<- (init), 32
- init<-,AbstractODESolver-method  
(AbstractODESolver-class), 3
- init<-,DormandPrince45-method  
(DormandPrince45-class), 4
- init<-,RK4-method (RK4-class), 40
  
- ODE (ODE-class), 33
- ODE-class, 33
- ODEAdaptiveSolver  
(ODEAdaptiveSolver-class), 37
- ODEAdaptiveSolver,ANY-method  
(ODEAdaptiveSolver-class), 37
- ODEAdaptiveSolver-class, 36
- ODESolver (ODESolver-class), 37
- ODESolver-class, 37
  
- ODESolverFactory  
(ODESolverFactory-class), 38
- ODESolverFactory,ANY-method  
(ODESolverFactory-class), 38
- ODESolverFactory-class, 38
  
- RK4 (RK4-class), 40
- RK4,ODE-method (RK4-class), 40
- RK4-class, 40
- RK45 (RK45-class), 44
- RK45-class, 44
- rODE-package, 2
- run\_test\_applications, 46
  
- setSolver<-, 46
- setState, 46
- setStepSize, 48
- setStepSize,AbstractODESolver-method  
(AbstractODESolver-class), 3
- setStepSize,DormandPrince45-method  
(DormandPrince45-class), 4
- setStepSize,Euler-method (Euler-class),  
11
- setStepSize,ODESolver-method  
(ODESolver-class), 37
- setStepSize,setStepSize-method  
(Euler-class), 11
- setTolerance, 50
- setTolerance,DormandPrince45-method  
(DormandPrince45-class), 4
- setTolerance,ODEAdaptiveSolver-method  
(ODEAdaptiveSolver-class), 37
- setTolerance,setTolerance-method  
(DormandPrince45-class), 4
- setTolerance<- (setTolerance), 50
- setTolerance<-,DormandPrince45-method  
(DormandPrince45-class), 4
- setTolerance<-,ODEAdaptiveSolver-method  
(ODEAdaptiveSolver-class), 37
- showMethods2, 53
- step, 53
- step,AbstractODESolver-method  
(AbstractODESolver-class), 3
- step,DormandPrince45-method  
(DormandPrince45-class), 4
- step,Euler-method (Euler-class), 11
- step,EulerRichardson-method  
(EulerRichardson-class), 16

step,ODESolver-method  
    (ODESolver-class), [37](#)  
step,RK4-method (RK4-class), [40](#)  
step,step-method (Euler-class), [11](#)  
step,Verlet-method (Verlet-class), [54](#)

Verlet (Verlet-class), [54](#)  
Verlet,ODE-method (Verlet-class), [54](#)  
Verlet-class, [54](#)