Package 'paropt'

July 6, 2020

Type Package

Title Parameter Optimizing of ODE-Systems

Version 0.1

Date 2020-07-28

Author Krämer Konrad [aut, cre],
Krämer Johannes [aut],
Heyer Arnd [ths],
University of Stuttgart [uvp],
Institute of Biomaterials and Biomolecular Systems at the University of Stuttgart [his]
| file AUTHORS

Maintainer Krämer Konrad <Konrad_kraemer@yahoo.de>

Description Enable optimization of parameters of ordinary differential equations. Therefore, using 'SUNDIALS' to solve the ODE-System (see Hindmarsh, Alan C., Peter N. Brown, Keith E. Grant, Steven L. Lee, Radu Serban, Dan E. Shumaker, and Carol S. Woodward. (2005) <doi:10.1145/1089014.1089020>). Furthermore, for optimization the particle swarm algorithm is used (see: Akman, Devin, Olcay Akman, and Elsa Schaefer. (2018) <doi:10.1155/2018/9160793> and Sengupta, Saptarshi, Sanchita Basak, and Richard Peters. (2018) <doi:10.3390/make1010010>). The ODE-System has to be passed as 'Rcpp'-function. The information for the parameter boundaries and states are conveyed using text-files.

License GPL-3 | file LICENSE

Imports Rcpp (>= 1.0.4)

LinkingTo Rcpp, RcppArmadillo

Suggests knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.1.0

Encoding UTF-8

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-07-06 14:10:02 UTC

R topics documented:

`ode_solving` *Solves ode-system and compare result to measured states*

Description

ode_solving solves ode system and calculates error between solved and measured values.

Usage

```
ode_solving(
  integration_times,
  ode_system,
  relative_tolerance,
  absolute_tolerances,
  start,
  states,
  where_to_save_output_states,
  solvertype
)
```

Arguments

`integration_times`	a vector containing the time course to solve the ode-system (see Details for more Information)
`ode_system`	the ode-system which will be integrated by the solver (see Details for more Information).
`relative_tolerance`	a number defining the relative tolerance used by the ode-solver.
`absolute_tolerances`	a vector containing the absolute tolerance(s) for each state used by the ode-solver.
`start`	a string-path to a tab seperated text-file containing values for the parameters (see Details for more Information).
`states`	a string-path to a tab seperated text-file containing the measured states (see Details for more Information).
`where_to_save_output_states`	a string-path defining a name for a textfile where the result of the integration of the states is saved. Using the previously optimized parameter in this integration.
`solvertype`	a string defines the type of solver which should be used (bdf, ADAMS, ERK or ARK. see Details for more Information).

Details

The vector containing the time course to solve the ode-system should contain the same entries as the time vector in the text file containing the states (of course it can be also be a different variable instead of time). It is possible that the vector is shorter than the time vector defined in the state-file in order to optimize only a part of the problem.

The ode system should be a Rcpp-function with a specific signature. The name of the function is free to choose. The following parameters have to be passed: a double t, a std::vector<double> params, and a Rcpp::NumericVector y.

The first entry defines the time point when the function is called.

The second argument defines the parameter which should be optimized. There exist two different types of parameters. Parameters can be either constant or variabel. In order to calculate a variable parameter at a specific timepoint the Catmull-Rom-Spline is used. This vector contains the already splined parameters, in the same order as defined in the text-files containing the start-values and the lower- and upper-boundaries.

The last argument is a vector containing the states in the same order as defined in the text-file containing the state-information. Thus, it is obligatory that the state-derivates in the ode-system are in the same order defined as in the text-file.

Furthermore, it is mandatory that the function return a Rcpp::NumericVector with the same dimension as the input vector containing the states. Naturally, the vector should contain the right hand side of the ode-system.

The file containing the start values for the parameter must have the following layout. In the first column the time is defined. In the following columns the parameters are defined. Consider that the parameter order is the same as used in the ode-system.

For constant parameters use only the first row (below the headers) if other parameters are variable use “NA“ in the following rows for the constant parameters.

For variable parameters at least four points are needed. If a variable parameter is not available at every time point use “NA“ instead.

Furthermore, it is notably that the time of the parameter should be within the time vector defined in the text-file containing the state information.

The file containing the state information should contain in the first column the time. The header-name time is compulsory. The following columns contain the states. Take care that the state order is the same as defined in the ode system. If a state is not available use “NA“. This is possible for every time points except the first one. The ode solver need a start value for each state which is extracted from the first row of this file (below the headers).

The error between the solver output and the measured states is the sum of the absolute differences divided by the number of time points. It is crucial that the states are in the same order in the text file cointaining the state-information and in the ode-system to compare the states correctly!

For solving the ode system the SUNDIALS Software is used (https://computing.llnl.gov/projects/sundials). The last argument defines the solver-type which is used during optimization: “bdf“, “ADAMS“, “ERK“ or “ARK“. bdf = Backward Differentiation Formulas, ADAMS = Adams-Moulton, ERK = explicite Runge-Kutta and ARK = implicite Runge-Kutta. All solvers are used in the NORMAL-Step method in a for-loop using the time-points defined in the text-file containing the states as output-points. The bdf- and ARK-Solver use the SUNLinSol_Dense as linear solver. Notably here is that for the ARK-Solver the ode system is fully implicit solved (not only part of it).

Examples

```
path <- system.file("examples", package = "paropt")
library(paropt)
#Rcpp::sourceCpp(paste(path,"/ode.cpp", sep = ""))
#if you want compile ode-system on your system (already precompiled in package)
df <- read.table(paste(path,"/states_LV.txt", sep = ""), header = TRUE)
time <- df$time
param_start <- paste(path, "/start.txt", sep = "")
states <- paste(path, "/states_LV.txt", sep = "")
state_output <- paste(tempdir(), "/final_stateoutput.txt", sep = "")
ode_solving(time, paropt:::ode_example, 1e-6, c(1e-8, 1e-8),
            param_start,
            states,
            state_output, "bdf")
df_in_silico <- read.table(paste(tempdir(), "/final_stateoutput.txt", sep = ""), header = TRUE)
plot(df$time, df$n1, pch = 19, main = "predator", ylab = "predator", xlab = "time")
points(df_in_silico$time, df_in_silico$n1, pch = 19, col = "darkred")
legend(1, 26, legend = c("measured", "in silico"),
       col = c("black", "darkred"),lty = 1:2, cex = 0.8)
plot(df$time, df$n2, pch = 19, main = "prey",ylab = "prey", xlab = "time")
points(df_in_silico$time, df_in_silico$n2, pch = 19, col = "darkred")
legend(1, 26, legend = c("measured", "in silico"),
       col = c("black", "darkred"),lty = 1:2, cex = 0.8)
```

`optimizer` *Optimize parameters of ode-systems*

Description

optimizer() finds parameters of an ode-system to match measured states.

Usage

```
optimizer(
  integration_times,
  ode_system,
  relative_tolerance,
  absolute_tolerances,
  start,
  lower,
  upper,
  states,
  npop,
  ngen,
  error,
  where_to_save_output_states,
  where_to_save_output_parameter,
  solvertype
)
```

Arguments

`integration_times`
: a vector containing the time course to solve the ode-system (see Details for more Information)

`ode_system`
: the ode-system which will be integrated by the solver (see Details for more Information).

`relative_tolerance`
: a number defining the relative tolerance used by the ode-solver.

`absolute_tolerances`
: a vector containing the absolute tolerance(s) for each state used by the ode-solver.

`start`
: a string-path to a tab seperated text-file containing startvalues for the parameters (see Details for more Information).

`lower`
: a string-path to a tab seperated text-file containing the lower bounds for the parameters (see Details for more Information).

`upper`
: a string-path to a tab seperated text-file containing the upper bounds for the parameters (see Details for more Information).

`states`
: a string-path to a tab seperated text-file containing the measured states (see Details for more Information).

`npop`
: a number defining the number of particles used by the Particle Swarm Optimizer.

`ngen`
: a number defining the number of generations the Particle Swarm Optimizer (PSO) should run.

`error`
: a number defining a sufficient small error. When the PSO reach this value optimization is stopped.

`where_to_save_output_states`
: a string-path defining a name for a textfile where the result of the integration of the states is saved. Using the previously optimized parameter for this integration.

`where_to_save_output_parameter`
: a string-path defining a name for a textfile where the optimized parameters are saved.

`solvertype`
: a string defines the type of solver which should be used (bdf, ADAMS, ERK or ARK. see Details for more Information).

Details

The vector containing the time course to solve the ode-system should contain the same entries as the time vector in the text file containing the states (of course it can be also be a different variable instead of time). It is possible that the vector is shorter than the time vector defined in the state-file in order to optimize only a part of the problem.

The ode system should be a Rcpp-function with a specific signature. The name of the function is free to choose. The following parameters have to be passed: a double t, a std::vector<double> params, and a Rcpp::NumericVector y.

The first entry defines the time point when the function is called.

The second argument defines the parameter which should be optimized. There exist two different types of parameters. Parameters can be either constant or variabel. In order to calculate a variable parameter at a specific timepoint the Catmull-Rom-Spline is used. This vector contains the already splined parameters, in the same order as defined in the text-files containing the start-values and the lower- and upper-boundaries.

The last argument is a vector containing the states in the same order as defined in the text-file containing the state-information. Thus, it is obligatory that the state-derivates in the ode-system are in the same order defined as in the text-file.

Furthermore, it is mandatory that the function return a Rcpp::NumericVector with the same dimension as the input vector containing the states. Naturally, the vector should contain the right hand side of the ode-system.

The files containing the start values (used to test integration) for the parameter, the lower- and upper-boundaries must have the following layout. In the first column the time is defined. In the following columns the parameters are defined. Consider that the parameter order is the same as used in the ode-system.

For constant parameters use only the first row (below the headers) if other parameters are variable use "NA" in the following rows for the constant parameters.

For variable parameters at least four points are needed. If a variable parameter is not available at every time point use "NA" instead. .

The three files start-values, lower and upper-boundaries need the parameter in the same order. The particles are randomly created within the lower and upper boundary.

The file containing the state information should contain in the first column the time. The header-name time is compulsory. The following columns contain the states. Take care that the state order is the same as defined in the ode system. If a state is not available use "NA". This is possible for every time points except the first one. The ode solver need a start value for each state which is extracted from the first row of this file (below the headers).

The error between the solver output and the measured states is the sum of the absolute differences divided by the number of time points. It is crucial that the states are in the same order in the text file cointaining the state-information and in the ode-system to compare the states correctly!

For solving the ode system the SUNDIALS Software is used (https://computing.llnl.gov/projects/sundials). The last argument defines the solver-type which is used during optimization: "bdf", "ADAMS", "ERK" or "ARK". bdf = Backward Differentiation Formulas, ADAMS = Adams-Moulton, ERK = explicite Runge-Kutta and ARK = implicite Runge-Kutta. All solvers are used in the NORMAL-Step method in a for-loop using the time-points defined in the text-file containing the states as output-points. The bdf- and ARK-Solver use the SUNLinSol_Dense as linear solver. Notably here is that for the ARK-Solver the ode system is fully implicit solved (not only part of it).

Examples

```
path <- system.file("examples", package = "paropt")
library(paropt)
#Rcpp::sourceCpp(paste(path,"/ode.cpp", sep = ""))
#if you want compile ode-system on your system (already precompiled in package)
df <- read.table(paste(path,"/states_LV.txt", sep = ""), header = TRUE)
time <- df$time
param_start <- paste(path, "/start.txt", sep = "")
param_lb <- paste(path, "/lb.txt", sep = "")
```

```
param_ub <- paste(path, "/ub.txt", sep = "")
states <- paste(path, "/states_LV.txt", sep = "")
state_output <- paste(tempdir(), "/final_stateoutput.txt", sep = "")
par_output <- paste(tempdir(), "/optimized_params.txt", sep = "")
set.seed(1)
optimizer(time, paropt:::ode_example, 1e-6, c(1e-8, 1e-8),
          param_start, param_lb, param_ub,
          states, npop = 10, ngen = 200, error = 3,
          state_output, par_output, "bdf")
df_in_silico <- read.table(paste(tempdir(), "/final_stateoutput.txt", sep = ""), header = TRUE)
plot(df$time, df$n1, pch = 19, main = "predator", ylab = "predator", xlab = "time")
points(df_in_silico$time, df_in_silico$n1, pch = 19, col = "darkred")
legend(1, 26, legend = c("measured", "in silico"),
       col = c("black", "darkred"),lty = 1:2, cex = 0.8)
plot(df$time, df$n2, pch = 19, main = "prey",ylab = "prey", xlab = "time")
points(df_in_silico$time, df_in_silico$n2, pch = 19, col = "darkred")
legend(1, 26, legend = c("measured", "in silico"),
       col = c("black", "darkred"),lty = 1:2, cex = 0.8)
```

Index