Package 'sundialr'

June 9, 2019

Type Package

Title An Interface to 'SUNDIALS' Ordinary Differential Equation (ODE) Solvers

Version 0.1.3

Maintainer Satyaprakash Nayak <sn248@cornell.edu>

URL https://github.com/sn248/sundialr

BugReports https://github.com/sn248/sundialr/issues

Description Provides a way to call the functions in 'SUNDIALS' C ODE solving library (<https://computation.llnl.gov/projects/sundials>). Currently the serial version of ODE solver, 'CVODE' and sensitivity calculator 'CVODES' from the 'SUNDIALS' library are implemented. The package requires ODE to be written as an 'R' or 'Rcpp' function and does not require the 'SUNDIALS' library to be installed on the local machine.

License GPL (>= 2)

LazyData TRUE

Imports Rcpp (>= 0.12.5)

LinkingTo Rcpp

RoxygenNote 6.1.1

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation yes

Author Satyaprakash Nayak [aut, cre],
Scott D Cohen [ctb],
Alan C Hindmarsh [ctb],
Radu Serban [ctb],
Dan Shumaker [ctb],
Daniel R Reynolds [ctb],
Aaron Collier [ctb],
David Gardner [ctb],
Carol Woodward [ctb],
Slaven Peles [ctb],
Peter Brown [ctb],

Hilari C Tiedeman [ctb],
Ting Yan [ctb],
Lawrence Livermore National Security [cph],
Southern Methodist University [cph]

Repository CRAN

Date/Publication 2019-06-09 21:20:03 UTC

R topics documented:

---

cvode *cvode*

Description

CVODE solver to solve stiff ODEs

Usage

```
cvode(time_vector, IC, input_function, Parameters, reltolerance,
  abstolerance)
```

Arguments

time_vector	time vector
IC	Initial Conditions
input_function	Right Hand Side function of ODEs
Parameters	Parameters input to ODEs
reltolerance	Relative Tolerance (a scalar)
abstolerance	Absolute Tolerance (a vector with length equal to ydot)

Examples

```
# ODEs described by an R function
ODE_R <- function(t, y, p){

  # vector containing the right hand side gradients
  ydot = vector(mode = "numeric", length = length(y))

  # R indices start from 1
  ydot[1] = -p[1]*y[1] + p[2]*y[2]*y[3]
  ydot[2] = p[1]*y[1] - p[2]*y[2]*y[3] - p[3]*y[2]*y[2]
```

```
  ydot[3] = p[3]*y[2]*y[2]

  # ydot[1] = -0.04 * y[1] + 10000 * y[2] * y[3]
  # ydot[3] = 30000000 * y[2] * y[2]
  # ydot[2] = -ydot[1] - ydot[3]

  ydot

}

# ODEs can also be described using Rcpp
Rcpp::sourceCpp(code = '

                #include <Rcpp.h>
                using namespace Rcpp;

                // ODE functions defined using Rcpp
                // [[Rcpp::export]]
                NumericVector ODE_Rcpp (double t, NumericVector y, NumericVector p){

                // Initialize ydot filled with zeros
                NumericVector ydot(y.length());

                ydot[0] = -p[0]*y[0] + p[1]*y[1]*y[2];
                ydot[1] = p[0]*y[0] - p[1]*y[1]*y[2] - p[2]*y[1]*y[1];
                ydot[2] = p[2]*y[1]*y[1];

                return ydot;

                }')



# R code to genrate time vector, IC and solve the equations
time_vec <- c(0.0, 0.4, 4.0, 40.0, 4E2, 4E3, 4E4, 4E5, 4E6, 4E7, 4E8, 4E9, 4E10)
IC <- c(1,0,0)
params <- c(0.04, 10000, 30000000)
reltol <- 1e-04
abstol <- c(1e-8,1e-14,1e-6)

## Solving the ODEs using cvode function
df1 <- cvode(time_vec, IC, ODE_R , params, reltol, abstol)          ## using R
df2 <- cvode(time_vec, IC, ODE_Rcpp , params, reltol, abstol)       ## using Rcpp

## Check that both solutions are identical
# identical(df1, df2)
```

---

cvodes *cvodes*

Description

CVODES solver to solve ODEs and calculate sensitivities

Usage

```
cvodes(time_vector, IC, input_function, Parameters, reltolerance,
  abstolerance, SensType = "STG", ErrCon = "F")
```

Arguments

time_vector	time vector
IC	Initial Conditions
input_function	Right Hand Side function of ODEs
Parameters	Parameters input to ODEs
reltolerance	Relative Tolerance (a scalar)
abstolerance	Absolute Tolerance (a vector with length equal to ydot)
SensType	Sensitivity Type - allowed values are Staggered (default)", "STG" (for Staggered) or "SIM" (for Simultaneous)
ErrCon	Error Control - allowed values are TRUE or FALSE (default)

Examples

```
# ODEs described by an R function
ODE_R <- function(t, y, p){

  # vector containing the right hand side gradients
  ydot = vector(mode = "numeric", length = length(y))

  # R indices start from 1
  ydot[1] = -p[1]*y[1] + p[2]*y[2]*y[3]
  ydot[2] = p[1]*y[1] - p[2]*y[2]*y[3] - p[3]*y[2]*y[2]
  ydot[3] = p[3]*y[2]*y[2]

  # ydot[1] = -0.04 * y[1] + 10000 * y[2] * y[3]
  # ydot[3] = 30000000 * y[2] * y[2]
  # ydot[2] = -ydot[1] - ydot[3]

  ydot

}

# ODEs can also be described using Rcpp
Rcpp::sourceCpp(code = '

                #include <Rcpp.h>
                using namespace Rcpp;

                // ODE functions defined using Rcpp
                // [[Rcpp::export]]
```

```
                NumericVector ODE_Rcpp (double t, NumericVector y, NumericVector p){

                // Initialize ydot filled with zeros
                NumericVector ydot(y.length());

                ydot[0] = -p[0]*y[0] + p[1]*y[1]*y[2];
                ydot[1] = p[0]*y[0] - p[1]*y[1]*y[2] - p[2]*y[1]*y[1];
                ydot[2] = p[2]*y[1]*y[1];

                return ydot;

                }')


# R code to genrate time vector, IC and solve the equations
time_vec <- c(0.0, 0.4, 4.0, 40.0, 4E2, 4E3, 4E4, 4E5, 4E6, 4E7, 4E8, 4E9, 4E10)
IC <- c(1,0,0)
params <- c(0.04, 10000, 30000000)
reltol <- 1e-04
abstol <- c(1e-8,1e-14,1e-6)

## Solving the ODEs using cvode function
df1 <- cvodes(time_vec, IC, ODE_R , params, reltol, abstol,"STG",FALSE)         ## using R
df2 <- cvodes(time_vec, IC, ODE_Rcpp , params, reltol, abstol,"STG",FALSE)      ## using Rcpp

## Check that both solutions are identical
# identical(df1, df2)
```

Index