Package 'deTestSet'

October 13, 2022

Version 1.1.7.3

Title Test Set for Differential Equations

Maintainer Karline Soetaert <karline.soetaert@nioz.nl>

Author Karline Soetaert [aut, cre],
Jeff Cash [aut],
Francesca Mazzia [aut],
Ernst Hairer [ctb] (files dopri8.f, dopri6.f),
Gerard Wanner [ctb] (files dopri8.f, dopri6.f),
T. Abdulla [ctb] (file mebdfi.f),
Cecilia Magherini [ctb] (files bimd.f, bimda.f),
Luigi Brugnano [ctb] (files bimd.f, bimda.f),
Cleve Moler [ctb] (file bimda.f)

Depends R (>= 2.01), deSolve

Imports graphics, grDevices, stats

Description Solvers and test set for stiff and non-stiff differential equations, and differential algebraic equations.
See Mazzia, F., Cash, J.R., and K. Soetaert (2012) <DOI:10.1016/j.cam.2012.03.014>.

License GPL

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-10-06 09:30:08 UTC

R topics documented:

`deTestSet-package` *Solvers and Test Set for Initial Value Problems of Ordinary Differential Equations (ODE), Partial Differential Equations (PDE) and for Differential Algebraic Equations (DAE)*

Description

R package deTestSet contains the R-version of the ODE and DAE initial value problems test set from the url: http://archimede.dm.uniba.it/~testset.

If the test model problem is small enough, then it is implemented in pure R. For larger models, the problem is specified in FORTRAN code.

These implementations were compiled as DLLs, and included in the package. The code of these models can be found in the packages `inst/doc/examples/dynload` subdirectory.

In addition to all solvers present in package `deSolve`, `deTestSet` contains the initial value problem solvers `gamd`, and `mebdfi`, implementing a generalised adams method and a differential algebraic equation solver of index up to three.

Details

Package:	deTestSet
Type:	Package
License:	GNU Public License 2 or above

Author(s)

Karline Soetaert (Maintainer),

Jeff Cash,

Francesca Mazzia

References

Mazzia, F., Cash, J.R. and K. Soetaert, 2012. A Test Set for Stiff Initial Value Problem Solvers in the open source software R: package deTestSet. Journal of Computational and Applied Mathematics 236: 4119-4131 DOI information: 10.1016/j.cam.2012.03.014.

See Also

ode for a general interface to most of the ODE solvers from package deSolve

ode.1D, ode.2D, ode.3D, for integrating 1-D, 2-D and 3-D models from package deSolve

dae, a general interface to the dae solvers, including mebdfi, gamd, and daspk and radau (deSolve)

Examples

```
## Not run:
## show examples (see respective help pages for details)
example(caraxis)
example(nand)
example(andrews)

## open the directory with R sourcecode examples
browseURL(paste(system.file(package = "deTestSet"), "/doc/examples", sep = ""))
## open the directory with C and FORTRAN sourcecode examples
browseURL(paste(system.file(package = "deTestSet"), "/doc/examples/dynload", sep = ""))

## show package vignette with how to use the test set
## + source code of the vignette
vignette("deTestSet")
edit(vignette("deTestSet"))

## End(Not run)
```

---

andrews *Andrews Squeezing Mechanism, Index 3 DAE*

Description

The andrews problem describes the motion of 7 rigid bodies connected by joints without friction

It is a non-stiff second order differential algebraic equation of index 3, consisting of 14 differential and 13 algebraic equations

Usage

```
andrews (times = seq(0, 0.03, by = 0.03/100), yini = NULL, dyini = NULL,
         parms = list(), printmescd = TRUE, method = mebdfi,
         atol = 1e-7, rtol = 1e-7, maxsteps = 1e+05, ...)
```

Arguments

yini	the initial (state) values for the DE system. If y has a name attribute, the names will be used to label the output matrix.
dyini	the initial derivatives of the state variables of the DE system.
atol	absolute error tolerance, either a scalar or a vector, one value for each y.
rtol	relative error tolerance, either a scalar or a vector, one value for each y,
times	time sequence for which output is wanted; the first value of times must be the initial time.
parms	list of parameters that overrule the default parameter values
printmescd	if TRUE the mixed error significant digits computed using the reference solution at time 0.03 are printed
method	the solver to use
maxsteps	maximal number of steps per output interval taken by the solver
...	additional arguments passed to the solver .

Details

The default parameters are: parameter <- c(m1 = .04325, m2 = .00365, m3 = .02373, m4 = .00706 , m5 = .07050, m6 = .00706, m7 = .05498 , xa = -.06934, ya = -.00227 , xb = -0.03635, yb = .03273 , xc = .014 , yc = .072, c0 = 4530 , i1 = 2.194e-6, i2 = 4.410e-7, i3 = 5.255e-6, i4 = 5.667e-7, i5 = 1.169e-5, i6 = 5.667e-7, i7 = 1.912e-5, d = 28e-3, da = 115e-4,e=2e-2, ea = 1421e-5, rr = 7e-3, ra = 92e-5, l0 = 7785e-5, ss = 35e-3, sa = 1874e-5, sb = 1043e-5, sc = 18e-3, sd = 2e-2, ta = 2308e-5, tb = 916e-5, u = 4e-2, ua = 1228e-5, ub = 449e-5, zf = 2e-2, zt = 4e-2,fa=1421e-5, mom = 33e-3)

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in yini, plus an additional column (the first) for the time value.

There will be one row for each element in times unless the solver returns with an unrecoverable error. If yini has a names attribute, it will be used to label the columns of the output value.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia <mazzia@dm.uniba.it>

References

https://archimede.dm.uniba.it/~testset/

Examples

```
out <- andrews()
plot(out, lwd = 2, which = 1:9)

refsol <- reference("andrews")
max(abs(out[nrow(out),-1] - refsol)/refsol)
```

---

beam *Motion of Inextensible Elastic Beam, ODE*

Description

The beam modulator problem is a problem from mechanics, describing the motion of an elastic beam, supposed inextensible, of length 1 and thin.

It is an ordinary differential equation of dimension 80.

Usage

```
beam (times=seq(0, 5, by = 0.05), yini = NULL,
      printmescd = TRUE, method = gamd,
      atol = 1e-6, rtol = 1e-6, ...)
```

Arguments

yini	the initial (state) values for the DE system. If y has a name attribute, the names will be used to label the output matrix.
times	time sequence for which output is wanted; the first value of times must be the initial time.
method	the solver to use
printmescd	if TRUE the mixed error significant digits computed using the reference solution at time 5 are printed
atol	absolute error tolerance, either a scalar or a vector, one value for each y.
rtol	relative error tolerance, either a scalar or a vector, one value for each y,
...	additional arguments passed to the solver .

Details

There are no parameters

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in yini, plus an additional column (the first) for the time value.

There will be one row for each element in times unless the solver returns with an unrecoverable error. If yini has a names attribute, it will be used to label the columns of the output value.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia <mazzia@dm.uniba.it>

References

https://archimede.dm.uniba.it/~testset/

Examples

```
out <- beam()
plot(out, col = "darkblue", lwd = 2, which = 1:16)
mtext(side = 3, line = -1.5, "beam", cex = 1.25, outer = TRUE)

image(out[,-1])

# compare with reference solution
refsol <- reference("beam")
max(abs(out[nrow(out),-1] - refsol)/refsol)
```

`bimd` *Blended Implicit Method for DAE*

Description

Solves the initial value problem for stiff or nonstiff systems of either:

- a system of ordinary differential equations (ODE) of the form

$$y' = f(t, y, ...)$$

or

- a system of linearly implicit DAES in the form

$$My' = f(t, y)$$

The R function `bimd` provides an interface to the Fortran DAE solver bimd, written by Cecilia Magherini and Luigi Bugnano.

It implements a Blended Implicit Methods of order 4-6-8-10-12 with step size control and continuous output.

The system of DAE's is written as an R function or can be defined in compiled code that has been dynamically loaded.

Usage

```
bimd(y, times, func, parms, nind = c(length(y), 0, 0),
  rtol = 1e-6, atol = 1e-6, jacfunc = NULL, jactype = "fullint",
  mass = NULL, massup = NULL, massdown = NULL, verbose = FALSE,
  hmax = NULL, hini = 0, ynames = TRUE, minord = NULL,
  maxord = NULL, bandup = NULL, banddown = NULL,
  maxsteps = 1e4, maxnewtit = c(10, 12, 14, 16, 18), wrkpars = NULL,
  dllname = NULL, initfunc = dllname, initpar = parms,
  rpar = NULL, ipar = NULL, nout=0, outnames = NULL, forcings = NULL,
  initforc = NULL, fcontrol = NULL, ...)
```

Arguments

y	the initial (state) values for the DAE or ODE system. If y has a name attribute, the names will be used to label the output matrix.
times	time sequence for which output is wanted; the first value of times must be the initial time; if only one step is to be taken; set times = NULL.
func	either an R-function that computes the values of the derivatives in the DAE or ODE system (the *model definition*) at time t, or a character string giving the name of a compiled function in a dynamically loaded shared library. If func is an R-function, it must be defined as: func <- function(t, y, parms,...). t is the current time point in the integration, y is the current estimate of the variables in the ODE system. If the initial values y has a names attribute, the names will be available inside func. parms is a vector or list of parameters; ... (optional) are any other arguments passed to the function. The return value of func should be a list, whose first element is a vector containing the derivatives of y with respect to time, and whose next elements are global values that are required at each point in times. The derivatives should be specified in the same order as the state variables y. If func is a string, then dllname must give the name of the shared library (without extension) which must be loaded before bimd() is called. See deSolve package vignette "compiledCode" for more details.
parms	vector or list of parameters used in func or jacfunc.
nind	if a DAE system: a three-valued vector with the number of variables of index 1, 2, 3 respectively. The equations must be defined such that the index 1 variables precede the index 2 variables which in turn precede the index 3 variables. The sum of the variables of different index should equal N, the total number of variables.
rtol	relative error tolerance, either a scalar or an array as long as y. See details.
atol	absolute error tolerance, either a scalar or an array as long as y. See details.
jacfunc	if not NULL, an R function that computes the Jacobian of the system of differential equations dydot(i)/dy(j), or a string giving the name of a function or subroutine in 'dllname' that computes the Jacobian (see vignette "compiledCode" from package deSolve, for more about this option).

In some circumstances, supplying jacfunc can speed up the computations, if the system is stiff. The R calling sequence for jacfunc is identical to that of func.

If the Jacobian is a full matrix, jacfunc should return a matrix dydot/dy, where the ith row contains the derivative of dy_i/dt with respect to y_j, or a vector containing the matrix elements by columns (the way R and FORTRAN store matrices).

If the Jacobian is banded, jacfunc should return a matrix containing only the nonzero bands of the Jacobian, rotated row-wise. See first example.

jactype — the structure of the Jacobian, one of "fullint", "fullusr", "bandusr" or "bandint" - either full or banded and estimated internally or by user.

mass — the mass matrix. If not NULL, the problem is a linearly implicit DAE and defined as $M\, dy/dt = f(t, y)$. If the mass-matrix M is full, it should be of dimension n^2 where n is the number of y-values; if banded the number of rows should be less than n, and the mass-matrix is stored diagonal-wise with element (i, j) stored in mass(i - j + mumas + 1, j).

If mass = NULL then the model is an ODE (default)

massup — number of non-zero bands above the diagonal of the mass matrix, in case it is banded.

massdown — number of non-zero bands below the diagonal of the mass matrix, in case it is banded.

verbose — if TRUE: full output to the screen, e.g. will print the diagnostiscs of the integration - see details.

hmax — an optional maximum value of the integration stepsize. If not specified, hmax is set to the largest difference in times, to avoid that the simulation possibly ignores short-term events. If 0, no maximal size is specified.

hini — initial step size to be attempted; if 0, the initial step size is set equal to 1e-6. Usually 1e-3 to 1e-5 is good for stiff equations

ynames — logical, if FALSE names of state variables are not passed to function func; this may speed up the simulation especially for multi-D models.

minord — the minimum order to be allowed, >= 3 and <= 9. NULL uses the default, 3.

maxord — the maximum order to be allowed, >= minord and <= 9. NULL uses the default, 9.

bandup — number of non-zero bands above the diagonal, in case the Jacobian is banded.

banddown — number of non-zero bands below the diagonal, in case the Jacobian is banded.

maxsteps — maximal number of steps taken by the solver, *for the entire integration*. This is different from the settings of this argument in the solvers from package deSolve!

maxnewtit — A five-valued integer vector, with the maximal number of splitting-Newton iterations for the solution of the iplicit system in each step for order 4, 6, 8, 10 and 12 respectively. The default is c(10, 12, 14, 16, 18)

wrkpars — A 12-valued real vector, with extra input parameters, put in the work vector work, at position work[3:14] in the fortran code - see details in fortran code. NULL uses the defaults

dllname	a string giving the name of the shared library (without extension) that contains all the compiled function or subroutine definitions refered to in func and jacfunc. See vignette "compiledCode" from package deSolve.
initfunc	if not NULL, the name of the initialisation function (which initialises values of parameters), as provided in 'dllname'. See vignette "compiledCode" from package deSolve.
initpar	only when 'dllname' is specified and an initialisation function initfunc is in the dll: the parameters passed to the initialiser, to initialise the common blocks (FORTRAN) or global variables (C, C++).
rpar	only when 'dllname' is specified: a vector with double precision values passed to the dll-functions whose names are specified by func and jacfunc.
ipar	only when 'dllname' is specified: a vector with integer values passed to the dll-functions whose names are specified by func and jacfunc.
nout	only used if dllname is specified and the model is defined in compiled code: the number of output variables calculated in the compiled function func, present in the shared library. Note: it is not automatically checked whether this is indeed the number of output variables calculed in the dll - you have to perform this check in the code - See vignette "compiledCode" from package deSolve.
outnames	only used if 'dllname' is specified and nout > 0: the names of output variables calculated in the compiled function func, present in the shared library. These names will be used to label the output matrix.
forcings	only used if 'dllname' is specified: a list with the forcing function data sets, each present as a two-columned matrix, with (time,value); interpolation outside the interval [min(times), max(times)] is done by taking the value at the closest data extreme. See forcings or package vignette "compiledCode".
initforc	if not NULL, the name of the forcing function initialisation function, as provided in 'dllname'. It MUST be present if forcings has been given a value. See forcings or package vignette "compiledCode".
fcontrol	A list of control parameters for the forcing functions. See forcings or vignette compiledCode.
...	additional arguments passed to func and jacfunc allowing this to be a generic function.

Details

The work is done by the FORTRAN 77 subroutine bimd, whose documentation should be consulted for details.

There are four standard choices for the jacobian which can be specified with jactype.

The options for **jactype** are

jactype = "fullint" a full Jacobian, calculated internally by the solver.

jactype = "fullusr" a full Jacobian, specified by user function jacfunc.

jactype = "bandusr" a banded Jacobian, specified by user function jacfunc; the size of the bands specified by bandup and banddown.

jactype = "bandint" a banded Jacobian, calculated by bimd; the size of the bands specified by bandup and banddown.

Inspection of the example below shows how to specify both a banded and full Jacobian.

The input parameters `rtol`, and `atol` determine the **error control** performed by the solver, which roughly keeps the local error of y(i) below rtol(i)*abs(y(i))+atol(i).

The diagnostics of the integration can be printed to screen by calling `diagnostics`. If `verbose = TRUE`, the diagnostics will written to the screen at the end of the integration.

See vignette("deSolve") from the `deSolve` package for an explanation of each element in the vectors containing the diagnostic properties and how to directly access them.

Models may be defined in compiled C or FORTRAN code, as well as in an R-function. See package vignette `"compiledCode"` from package `deSolve` for details.

Information about linking forcing functions to compiled code is in forcings (from package `deSolve`).

Value

A matrix of class `deSolve` with up to as many rows as elements in `times` and as many columns as elements in `y` plus the number of "global" values returned in the next elements of the return from `func`, plus and additional column for the time value. There will be a row for each element in `times` unless the FORTRAN routine 'bimd' returns with an unrecoverable error. If `y` has a names attribute, it will be used to label the columns of the output value.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia

References

L.BRUGNANO, C.MAGHERINI, F.MUGNAI. Blended Implicit Methods for the Numerical Solution of DAE problems. Jour. CAM 189 (2006) 34-50.

L.BRUGNANO, C.MAGHERINI The BiM code for the numerical solution of ODEs Jour. CAM 164-165 (2004) 145-158.

L.BRUGNANO, C.MAGHERINI Some Linear Algebra issues concerning the implementation of Blended Implicit Methods Numer. Linear Alg. Appl. 12 (2005) 305-314.

L.BRUGNANO, C.MAGHERINI Economical Error Estimates for Block Implicit Methods for ODEs via Deferred Correction. Appl. Numer. Math. 56 (2006) 608-617.

L.BRUGNANO, C.MAGHERINI Blended Implementation of Block Implicit Methods for ODEs Appl. Numer. Math. 42 (2002) 29-45.

See Also

- `gamd` another DAE solver from package `deTestSet`,
- `mebdfi` another DAE solver from package `deTestSet`,
- `daspk` another DAE solver from package `deSolve`,
- `ode` for a general interface to most of the ODE solvers from package `deSolve`,

- ode.1D for integrating 1-D models,
- ode.2D for integrating 2-D models,
- ode.3D for integrating 3-D models,
- mebdfi for integrating DAE models,
- dopri853 for the Dormand-Prince Runge-Kutta method of order 8(53)

diagnostics to print diagnostic messages.

Examples

```
## =====================================================================
## Example 1:
##   Various ways to solve the same model.
## =====================================================================

## the model, 5 state variables
f1 <- function  (t, y, parms)
{
  ydot <- vector(len = 5)

  ydot[1] <-  0.1*y[1] -0.2*y[2]
  ydot[2] <- -0.3*y[1] +0.1*y[2] -0.2*y[3]
  ydot[3] <-           -0.3*y[2] +0.1*y[3] -0.2*y[4]
  ydot[4] <-                     -0.3*y[3] +0.1*y[4] -0.2*y[5]
  ydot[5] <-                               -0.3*y[4] +0.1*y[5]

  return(list(ydot))
}

## the Jacobian, written as a full matrix
fulljac <- function  (t, y, parms)
{
   jac <- matrix(nrow = 5, ncol = 5, byrow = TRUE,
                 data = c(0.1, -0.2,  0  ,  0  ,  0  ,
                         -0.3,  0.1, -0.2,  0  ,  0  ,
                          0  , -0.3,  0.1, -0.2,  0  ,
                          0  ,  0  , -0.3,  0.1, -0.2,
                          0  ,  0  ,  0  , -0.3,  0.1)    )
   return(jac)
}

## the Jacobian, written in banded form
bandjac <- function  (t, y, parms)
{
   jac <- matrix(nrow = 3, ncol = 5, byrow = TRUE,
                 data = c( 0  , -0.2, -0.2, -0.2, -0.2,
                           0.1,  0.1,  0.1,  0.1,  0.1,
                          -0.3, -0.3, -0.3, -0.3,    0)    )
   return(jac)
}

## initial conditions and output times
```

```
yini  <- 1:5
times <- 1:20

## default: stiff method, internally generated, full Jacobian
out   <- bimd(yini, times, f1, parms = 0, jactype = "fullint")
plot(out)

## stiff method, user-generated full Jacobian
out2  <- bimd(yini, times, f1, parms = 0, jactype = "fullusr",
              jacfunc = fulljac)

## stiff method, internally-generated banded Jacobian
## one nonzero band above (up) and below(down) the diagonal
out3  <- bimd(yini, times, f1, parms = 0, jactype = "bandint",
                              bandup = 1, banddown = 1)

## stiff method, user-generated banded Jacobian
out4  <- bimd(yini, times, f1, parms = 0, jactype = "bandusr",
              jacfunc = bandjac, bandup = 1, banddown = 1)


## =============================================================================
## Example 2:
##   stiff problem from chemical kinetics
## =============================================================================
Chemistry <- function (t, y, p) {
     dy1 <- -.04*y[1] + 1.e4*y[2]*y[3]
     dy2 <- .04*y[1] - 1.e4*y[2]*y[3] - 3.e7*y[2]^2
     dy3 <- 3.e7*y[2]^2
     list(c(dy1,dy2,dy3))
}

times <- 10^(seq(0, 10, by = 0.1))
yini <- c(y1 = 1.0, y2 = 0, y3 = 0)

out <- bimd(func = Chemistry, times = times, y = yini, parms = NULL)
plot(out, log = "x", type = "l", lwd = 2)


## =============================================================================
## Example 3: DAE
## Car axis problem, index 3 DAE, 8 differential, 2 algebraic equations
## from
## F. Mazzia and C. Magherini. Test Set for Initial Value Problem Solvers,
## release 2.4. Department
## of Mathematics, University of Bari and INdAM, Research Unit of Bari,
## February 2008.
## Available at http://www.dm.uniba.it/~testset.
## =============================================================================

## Problem is written as M*y = f(t,y,p).
## caraxisfun implements the right-hand side:
```

```
caraxisfun <- function(t, y, parms) {
  with(as.list(y), {

    yb <- r * sin(w * t)
    xb <- sqrt(L * L - yb * yb)
    Ll <- sqrt(xl^2 + yl^2)
    Lr <- sqrt((xr - xb)^2 + (yr - yb)^2)

    dxl <- ul; dyl <- vl; dxr <- ur; dyr <- vr

    dul  <- (L0-Ll) * xl/Ll      + 2 * lam2 * (xl-xr) + lam1*xb
    dvl  <- (L0-Ll) * yl/Ll      + 2 * lam2 * (yl-yr) + lam1*yb - k * g

    dur  <- (L0-Lr) * (xr-xb)/Lr - 2 * lam2 * (xl-xr)
    dvr  <- (L0-Lr) * (yr-yb)/Lr - 2 * lam2 * (yl-yr) - k * g

    c1   <- xb * xl + yb * yl
    c2   <- (xl - xr)^2 + (yl - yr)^2 - L * L

    list(c(dxl, dyl, dxr, dyr, dul, dvl, dur, dvr, c1, c2))
  })
}

eps <- 0.01; M <- 10; k <- M * eps^2/2;
L <- 1; L0 <- 0.5; r <- 0.1; w <- 10; g <- 1

yini <- c(xl = 0, yl = L0, xr = L, yr = L0,
          ul = -L0/L, vl = 0,
          ur = -L0/L, vr = 0,
          lam1 = 0, lam2 = 0)

# the mass matrix
Mass      <- diag(nrow = 10, 1)
Mass[5,5] <- Mass[6,6] <- Mass[7,7] <- Mass[8,8] <- M * eps * eps/2
Mass[9,9] <- Mass[10,10] <- 0
Mass

# index of the variables: 4 of index 1, 4 of index 2, 2 of index 3
index <- c(4, 4, 2)

times <- seq(0, 3, by = 0.01)
out <- bimd(y = yini, mass = Mass, times = times, func = caraxisfun,
        parms = NULL, nind = index)

plot(out, which = 1:4, type = "l", lwd = 2)
```

---

caraxis *The Car Axis Mechanical Problem, Index 3 DAE*

Description

A rather simple multibody system, describing the behavior of a car axis on a bumpy road.

It is a differential algebraic equation of index 3

Usage

```
caraxis (times = seq(0, 3, by = 0.01), yini = NULL, dyini = NULL,
         parms = list(), printmescd = TRUE, method = mebdfi,
         atol=1e-6, rtol=1e-6,  ...)
```

Arguments

yini	the initial (state) values for the DE system. If y has a name attribute, the names will be used to label the output matrix.
dyini	the initial derivatives of the state variables of the DE system.
times	time sequence for which output is wanted; the first value of times must be the initial time.
method	the solver to use; only mebdfi available for now
atol	absolute error tolerance, either a scalar or a vector, one value for each y.
rtol	relative error tolerance, either a scalar or a vector, one value for each y,
printmescd	if TRUE the mixed error significant digits computed using the reference solution at time 3 are printed
parms	list of parameters that overrule the default parameter values
...	additional arguments passed to the solver .

Details

The default parameters are: eps = 1e-2, M = 10, L = 1, L0 = 0.5, r = 0.1, w = 10, g = 1

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in yini, plus an additional column (the first) for the time value.

There will be one row for each element in times unless the solver returns with an unrecoverable error. If yini has a names attribute, it will be used to label the columns of the output value.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia <mazzia@dm.uniba.it>

References

https://archimede.dm.uniba.it/~testset/

Examples

```
out <- caraxis()
plot(out, lwd = 2, mfrow = c(3, 4))

# compare with reference solution
out[nrow(out),2:11]-reference("caraxis")
```

---

crank *Slider Crank Mechanical Problem, Index 2 DAE*

Description

The crank problem is a constrained mechanical system including both rigid and elastic bodies

It is a differential algebraic equation of index 2, 24 equations.

Usage

```
crank (times=seq(0, 0.1, by = 0.001), yini = NULL, dyini = NULL,
       parms = list(), printmescd = TRUE, method = mebdfi,
       atol = 1e-6, rtol = 1e-6, maxsteps = 1e+06,
       options = list(), ...)
```

Arguments

yini	the initial (state) values for the DE system. If y has a name attribute, the names will be used to label the output matrix.
dyini	the initial derivatives of the state variables of the DE system.
times	time sequence for which output is wanted; the first value of times must be the initial time.
parms	list of parameters that overrule the default parameter values
method	the solver to use; only mebdfi available for now
maxsteps	maximal number of steps per output interval taken by the solver
atol	absolute error tolerance, either a scalar or a vector, one value for each y.
rtol	relative error tolerance, either a scalar or a vector, one value for each y,
printmescd	if TRUE the mixed error significant digits computed using the reference solution at time 0.1 are printed
options	a list which specifies the initial conditions used ini, whether the problem is stiff, and the damping. The default is list(ini=1,stiff=0,damp=0)
...	additional arguments passed to the solver .

Details

The default parameters are: M1 = 0.36, M2 = 0.151104, M3 = 0.075552, L1 = 0.15, L2 = 0.30, J1 = 0.002727, J2 = 0.0045339259, EE = 0.20e12, NUE = 0.30, BB = 0.0080, HH = 0.0080, RHO = 7870.0, GRAV = 0.0, OMEGA = 150.0

There are two default initial conditions - set with options(ini=x)

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in yini, plus an additional column (the first) for the time value.

There will be one row for each element in times unless the solver returns with an unrecoverable error. If yini has a names attribute, it will be used to label the columns of the output value.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia <mazzia@dm.uniba.it>

References

https://archimede.dm.uniba.it/~testset/

Simeon, B.: Modelling a flexible slider crank mechanism by a mixed system of DAEs and PDEs, Math. Modelling of Systems 2, 1-18 (1996);

Examples

```
out <- crank()
plot(out, lwd = 2, which = 1:9)

# compare  with reference solution (only  the first seven components)
refsol <- reference("crank")
max(abs(out[nrow(out),2:8] - refsol[1:7])/refsol[1:7])
```

---

dae *General Solver for Differential Algebraic Equations*

Description

Solves a system of differential algebraic equations; a wrapper around the implemented DAE solvers

Usage

```
dae(y, times, parms, dy, res = NULL, func = NULL,
    method = c("mebdfi", "daspk", "radau", "gamd", "bimd"), ...)
```

Arguments

y	the initial (state) values for the DAE system, a vector. If y has a name attribute, the names will be used to label the output matrix.
times	time sequence for which output is wanted; the first value of times must be the initial time.
parms	vector or list of parameters used in res
dy	the initial derivatives of the state variables of the DAE system.

`func` to be used if the model is an ODE, or a DAE written in linearly implicit form (M y' = f(t, y)). `func` should be an R-function that computes the values of the derivatives in the ODE system (the *model definition*) at time t.

`func` must be defined as: `func <- function(t, y, parms,...)`.
`t` is the current time point in the integration, `y` is the current estimate of the variables in the ODE system. If the initial values `y` has a `names` attribute, the names will be available inside `func`, unless `ynames` is FALSE. `parms` is a vector or list of parameters. `...` (optional) are any other arguments passed to the function.

The return value of `func` should be a list, whose first element is a vector containing the derivatives of `y` with respect to `time`, and whose next elements are global values that are required at each point in `times`. The derivatives should be specified in the same order as the specification of the state variables `y`.

`res` either an R-function that computes the residual function F(t,y,y') of the DAE system (the model defininition) at time `t`, or a character string giving the name of a compiled function in a dynamically loaded shared library.

If `res` is a user-supplied R-function, it must be defined as: `res <- function(t, y, dy, parms, ...)`.

Here `t` is the current time point in the integration, `y` is the current estimate of the variables in the DAE system, `dy` are the corresponding derivatives. If the initial `y` or `dy` have a `names` attribute, the names will be available inside `res`, unless `ynames` is `FALSE`. `parms` is a vector of parameters.

The return value of `res` should be a list, whose first element is a vector containing the residuals of the DAE system, i.e. delta = F(t,y,y'), and whose next elements contain output variables that are required at each point in `times`.

If `res` is a string, then `dllname` must give the name of the shared library (without extension) which must be loaded before `dae()` is called (see deSolve package vignette `"compiledCode"` for more information).

`method` the solver to use, either a string (`"mebdfi"`, `"daspk"`), `"radau"`, `"gamd"` or a function that performs integration.

`...` additional arguments passed to the solvers.

Details

This is simply a wrapper around the various dae solvers.

See package vignette for information about specifying the model in compiled code.

See the selected integrator for the additional options.

Value

A matrix of class `deSolve` with up to as many rows as elements in `times` and as many columns as elements in `y` plus the number of "global" values returned in the second element of the return from `res`, plus an additional column (the first) for the time value. There will be one row for each element in `times` unless the integrator returns with an unrecoverable error. If `y` has a names attribute, it will be used to label the columns of the output value.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

See Also

- ode for a wrapper around the ode solvers,
- ode.band for solving models with a banded Jacobian,
- ode.1D for integrating 1-D models,
- ode.2D for integrating 2-D models,
- ode.3D for integrating 3-D models,
- mebdfi, daspk,radau,gamd,bimd, for the dae solvers

diagnostics to print diagnostic messages.

Examples

```
## =============================================================================
## Chemical problem
## =============================================================================

daefun <- function(t, y, dy, parms) {
  with (as.list(c(y, dy, parms)), {
    res1 <- dA + dAB + lambda * A
    res2 <- dAB + dB
    alg <- B * A - K * AB
  list(c(res1, res2, alg), sumA = A + AB)
  })
}

parms <- c(lambda = 0.1, K = 1e-4)

yini  <- with(as.list(parms),
     c(A = 1, AB = 1, B = K))
dyini <- c(dA = 0, dAB = 0, dB = 0)

times <- 0:100

print(system.time(
out <-dae (y=yini, dy = dyini, times = times, res = daefun,
           parms = parms, method = "daspk")
))

plot(out, ylab = "conc.", xlab = "time", lwd = 2)
mtext("IVP DAE", side = 3, outer = TRUE, line = -1)
```

Description

Solves the initial value problem for systems of ordinary differential equations (ODE) in the form:

$$dy/dt = f(t, y)$$

The R function `dopri5` provides an interface to the Fortran ODE solver DOPRI5, written by E. Hairer and G. Wanner.

It implements the explicit Runge-Kutta method of order 4(5) due to Dormand & Prince with stepsize control and dense output

The R function `cashkarp` provides an interface to the Fortran ODE solver CASHCARP, written by J. Cash and F. Mazzia.

It implements the explicit Runge-Kutta method of order 4(5) due to Cash-Carp, with stepsize control and dense output

The system of ODE's is written as an R function or can be defined in compiled code that has been dynamically loaded.

Usage

```
dopri5   (y, times, func, parms, rtol = 1e-6, atol = 1e-6,
  verbose = FALSE, hmax = NULL, hini = hmax, ynames = TRUE,
  maxsteps = 10000, dllname = NULL, initfunc = dllname,
  initpar=parms, rpar = NULL, ipar = NULL, nout = 0,
  outnames = NULL, forcings = NULL, initforc = NULL, fcontrol = NULL, ...)

cashkarp (y, times, func, parms, rtol = 1e-6, atol = 1e-6,
  verbose = FALSE, hmax = NULL, hini = hmax, ynames = TRUE,
  maxsteps = 10000, dllname = NULL, initfunc = dllname, initpar = parms,
  rpar = NULL, ipar = NULL, nout = 0, outnames = NULL, forcings = NULL,
  initforc = NULL, fcontrol = NULL, stiffness = 2, ...)
```

Arguments

`y`	the initial (state) values for the ODE system. If `y` has a name attribute, the names will be used to label the output matrix.
`times`	time sequence for which output is wanted; the first value of `times` must be the initial time; if only one step is to be taken; set `times = NULL`.
`func`	either an R-function that computes the values of the derivatives in the ODE system (the *model definition*) at time `t`, or a character string giving the name of a compiled function in a dynamically loaded shared library. If `func` is an R-function, it must be defined as: `func <- function(t, y, parms, ...)`. `t` is the current time point in the integration, `y` is the current estimate of the variables in the ODE system. If the initial values `y` has a `names` attribute, the names will be available inside `func`. `parms` is a vector or list of parameters; ... (optional) are any other arguments passed to the function. The return value of `func` should be a list, whose first element is a vector containing the derivatives of `y` with respect to `time`, and whose next elements are

global values that are required at each point in `times`. The derivatives should be specified in the same order as the state variables `y`.

If `func` is a string, then `dllname` must give the name of the shared library (without extension) which must be loaded before `lsode()` is called. See package vignette `"compiledCode"` for more details.

`parms` vector or list of parameters used in `func` or `jacfunc`.

`rtol` relative error tolerance, either a scalar or an array as long as `y`. See details.

`atol` absolute error tolerance, either a scalar or an array as long as `y`. See details.

`verbose` if `TRUE`: full output to the screen, e.g. will print the `diagnostiscs` of the integration - if the method becomes stiff it will rpint a message.

`hmax` an optional maximum value of the integration stepsize. If not specified, `hmax` is set to the largest difference in `times`.

`hini` initial step size to be attempted.

`ynames` logical, if `FALSE` names of state variables are not passed to function `func`; this may speed up the simulation especially for multi-D models.

`maxsteps` maximal number of steps taken by the solver, *for the entire integration*. This is different from the settings of this argument in the solvers from package deSolve!

`dllname` a string giving the name of the shared library (without extension) that contains all the compiled function or subroutine definitions refered to in `func` and `jacfunc`. See vignette `"compiledCode"` from package `deSolve`.

`initfunc` if not `NULL`, the name of the initialisation function (which initialises values of parameters), as provided in '`dllname`'. See vignette `"compiledCode"` from package `deSolve`.

`initpar` only when '`dllname`' is specified and an initialisation function `initfunc` is in the dll: the parameters passed to the initialiser, to initialise the common blocks (FORTRAN) or global variables (C, C++).

`rpar` only when '`dllname`' is specified: a vector with double precision values passed to the dll-functions whose names are specified by `func` and `jacfunc`.

`ipar` only when '`dllname`' is specified: a vector with integer values passed to the dll-functions whose names are specified by `func` and `jacfunc`.

`nout` only used if `dllname` is specified and the model is defined in compiled code: the number of output variables calculated in the compiled function `func`, present in the shared library. Note: it is not automatically checked whether this is indeed the number of output variables calculed in the dll - you have to perform this check in the code - See vignette `"compiledCode"` from package `deSolve`.

`outnames` only used if '`dllname`' is specified and `nout` > 0: the names of output variables calculated in the compiled function `func`, present in the shared library. These names will be used to label the output matrix.

`forcings` only used if '`dllname`' is specified: a list with the forcing function data sets, each present as a two-columned matrix, with (time,value); interpolation outside the interval [min(`times`), max(`times`)] is done by taking the value at the closest data extreme.

See forcings or package vignette `"compiledCode"`.

initforc	if not NULL, the name of the forcing function initialisation function, as provided in 'dllname'. It MUST be present if forcings has been given a value. See forcings or package vignette "compiledCode".
fcontrol	A list of control parameters for the forcing functions. See forcings or vignette compiledCode.
stiffness	How the stiffness of the solution should be estimated. Default = stiffness based on eigenvalue approximation; when = stiffness = 0: no stiffness estimate; when = stiffness = 1 or -1: all stiffness estimates calculated ; when = stiffness = 2 or -2: stiffness based on eigenvalue approximation; when = stiffness = 3 or -3: stiffness based on error estimate; when = stiffness = 4 or -4: stiffness based on conditioning. Positive values of stiffness will cause the integration to stop; negative values will continue anyway.
...	additional arguments passed to func and jacfunc allowing this to be a generic function.

Details

The work is done by the FORTRAN subroutine dop853, whose documentation should be consulted for details. The implementation is based on the Fortran 77 version fromOctober 11, 2009.

The input parameters rtol, and atol determine the **error control** performed by the solver, which roughly keeps the local error of y(i) below rtol(i)*abs(y(i))+atol(i).

The diagnostics of the integration can be printed to screen by calling diagnostics. If verbose = TRUE, the diagnostics will written to the screen at the end of the integration.

See vignette("deSolve") from the deSolve package for an explanation of each element in the vectors containing the diagnostic properties and how to directly access them.

Models may be defined in compiled C or FORTRAN code, as well as in an R-function. See package vignette "compiledCode" from package deSolve for details.

Information about linking forcing functions to compiled code is in forcings (from package deSolve).

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in y plus the number of "global" values returned in the next elements of the return from func, plus and additional column for the time value. There will be a row for each element in times unless the FORTRAN routine 'lsoda' returns with an unrecoverable error. If y has a names attribute, it will be used to label the columns of the output value.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

References

E. Hairer, S.P. Norsett AND G. Wanner, Solving Ordinary Differential Equations I. Nonstiff Problems. 2nd Edition. Springer Series In Computational Mathematics, SPRINGER-VERLAG (1993)

See Also

- ode for a general interface to most of the ODE solvers from package deSolve,
- ode.1D for integrating 1-D models,
- ode.2D for integrating 2-D models,
- ode.3D for integrating 3-D models,
- mebdfi for integrating DAE models,
- bimd for blended implicit methods,
- gamd for the generalised adams method

diagnostics to print diagnostic messages.

Examples

```
## =====================================================================
## Example :
##   The Arenstorff orbit model
## =====================================================================

Arenstorff <- function(t, y, parms) {

  D1 <- ((y[1]+mu)^2+y[2]^2)^(3/2)
  D2 <- ((y[1]-(1-mu))^2+y[2]^2)^(3/2)

  dy1 <- y[3]
  dy2 <- y[4]
  dy3 <- y[1] + 2*y[4]-(1-mu)*(y[1]+mu)/D1 -mu*(y[1]-(1-mu))/D2
  dy4 <- y[2] - 2*y[3]-(1-mu)*y[2]/D1 - mu*y[2]/D2

  list(c(dy1,dy2,dy3,dy4))
}

#-----------------------------
# parameters, initial values and times
#-----------------------------
mu    <- 0.012277471
yini  <- c(x = 0.994, y = 0, dx = 0,
  dy = -2.00158510637908252240537862224)
times <- seq(0, 18, 0.01)

#-----------------------------
# solve the model
#-----------------------------

#out <- dopri5 (times=times, y=yini, func = Arenstorff, parms=NULL )
out  <- cashkarp (times = times, y = yini, func = Arenstorff, parms = NULL )
plot(out[,c("x", "y")], type = "l", lwd = 2, main = "Arenstorff")

#-----------------------------
# First and last value should be the same
```

```
#-----------------------------

times <- c(0, 17.065216560157962558891720624 9)

Test  <- dopri5 (times = times, y = yini, func = Arenstorff, parms = NULL)

diagnostics(Test)
```

---

dopri853 *Dormand-Prince Runge-Kutta of Order 8(5,3)*

Description

Solves the initial value problem for systems of ordinary differential equations (ODE) in the form:

$$dy/dt = f(t, y)$$

The R function dopri853 provides an interface to the Fortran ODE solver DOP853, written by Hairer and Wanner.

It implements the explicit Runge-Kutta method of order 8(5,3) due to Dormand & Prince with stepsize contral and dense output

The system of ODE's is written as an R function or can be defined in compiled code that has been dynamically loaded.

Usage

```
dopri853 (y, times, func, parms, rtol = 1e-6, atol = 1e-6,
  verbose = FALSE, hmax = NULL, hini = hmax, ynames = TRUE,
  maxsteps = 10000, dllname = NULL, initfunc = dllname,
  initpar = parms, rpar = NULL, ipar = NULL, nout = 0,
  outnames = NULL, forcings = NULL, initforc = NULL, fcontrol = NULL, ...)
```

Arguments

y	the initial (state) values for the ODE system. If y has a name attribute, the names will be used to label the output matrix.
times	time sequence for which output is wanted; the first value of times must be the initial time; if only one step is to be taken; set times = NULL.
func	either an R-function that computes the values of the derivatives in the ODE system (the *model definition*) at time t, or a character string giving the name of a compiled function in a dynamically loaded shared library. If func is an R-function, it must be defined as: func <- function(t, y, parms, ...). t is the current time point in the integration, y is the current estimate of the variables in the ODE system. If the initial values y has a names attribute, the names will be available inside func. parms is a vector or list of parameters; ... (optional) are any other arguments passed to the function.

The return value of func should be a list, whose first element is a vector containing the derivatives of y with respect to time, and whose next elements are global values that are required at each point in times. The derivatives should be specified in the same order as the state variables y.

If func is a string, then dllname must give the name of the shared library (without extension) which must be loaded before lsode() is called. See package vignette "compiledCode" for more details.

parms — vector or list of parameters used in func or jacfunc.

rtol — relative error tolerance, either a scalar or an array as long as y. See details.

atol — absolute error tolerance, either a scalar or an array as long as y. See details.

verbose — if TRUE: full output to the screen, e.g. will print the diagnostiscs of the integration - if the method becomes stiff it will rpint a message.

hmax — an optional maximum value of the integration stepsize. If not specified, hmax is set to the largest difference in times.

hini — initial step size to be attempted.

ynames — logical, if FALSE names of state variables are not passed to function func; this may speed up the simulation especially for multi-D models.

maxsteps — maximal number of steps taken by the solver, *for the entire integration*. This is different from the settings of this argument in the solvers from package deSolve!

dllname — a string giving the name of the shared library (without extension) that contains all the compiled function or subroutine definitions refered to in func and jacfunc. See vignette "compiledCode" from package deSolve.

initfunc — if not NULL, the name of the initialisation function (which initialises values of parameters), as provided in 'dllname'. See vignette "compiledCode" from package deSolve.

initpar — only when 'dllname' is specified and an initialisation function initfunc is in the dll: the parameters passed to the initialiser, to initialise the common blocks (FORTRAN) or global variables (C, C++).

rpar — only when 'dllname' is specified: a vector with double precision values passed to the dll-functions whose names are specified by func and jacfunc.

ipar — only when 'dllname' is specified: a vector with integer values passed to the dll-functions whose names are specified by func and jacfunc.

nout — only used if dllname is specified and the model is defined in compiled code: the number of output variables calculated in the compiled function func, present in the shared library. Note: it is not automatically checked whether this is indeed the number of output variables calculed in the dll - you have to perform this check in the code - See vignette "compiledCode" from package deSolve.

outnames — only used if 'dllname' is specified and nout > 0: the names of output variables calculated in the compiled function func, present in the shared library. These names will be used to label the output matrix.

forcings — only used if 'dllname' is specified: a list with the forcing function data sets, each present as a two-columned matrix, with (time,value); interpolation outside the interval [min(times), max(times)] is done by taking the value at the closest data extreme.

See forcings or package vignette "compiledCode".

initforc	if not NULL, the name of the forcing function initialisation function, as provided in 'dllname'. It MUST be present if forcings has been given a value. See forcings or package vignette "compiledCode".
fcontrol	A list of control parameters for the forcing functions. See forcings or vignette compiledCode.
...	additional arguments passed to func and jacfunc allowing this to be a generic function.

Details

The work is done by the FORTRAN subroutine dop853, whose documentation should be consulted for details. The implementation is based on the Fortran 77 version fromOctober 11, 2009.

The input parameters rtol, and atol determine the **error control** performed by the solver, which roughly keeps the local error of y(i) below rtol(i)*abs(y(i))+atol(i).

The diagnostics of the integration can be printed to screen by calling diagnostics. If verbose = TRUE, the diagnostics will written to the screen at the end of the integration.

See vignette("deSolve") from the deSolve package for an explanation of each element in the vectors containing the diagnostic properties and how to directly access them.

Models may be defined in compiled C or FORTRAN code, as well as in an R-function. See package vignette "compiledCode" from package deSolve for details.

Information about linking forcing functions to compiled code is in forcings (from package deSolve).

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in y plus the number of "global" values returned in the next elements of the return from func, plus and additional column for the time value. There will be a row for each element in times unless the FORTRAN routine 'lsoda' returns with an unrecoverable error. If y has a names attribute, it will be used to label the columns of the output value.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

References

E. Hairer, S.P. Norsett AND G. Wanner, Solving Ordinary Differential Equations I. Nonstiff Problems. 2nd Edition. Springer Series In Computational Mathematics, SPRINGER-VERLAG (1993)

See Also

- ode for a general interface to most of the ODE solvers from package deSolve,
- ode.1D for integrating 1-D models,
- ode.2D for integrating 2-D models,
- ode.3D for integrating 3-D models,
- mebdfi for integrating DAE models,

- gamd for the generalised adams method

diagnostics to print diagnostic messages.

Examples

```
## =====================================================================
## Example :
##   The Arenstorff orbit model
## =====================================================================

Arenstorff <- function(t, y, parms) {

  D1 <- ((y[1]+mu)^2+y[2]^2)^(3/2)
  D2 <- ((y[1]-(1-mu))^2+y[2]^2)^(3/2)

  dy1 <- y[3]
  dy2 <- y[4]
  dy3 <- y[1] + 2*y[4]-(1-mu)*(y[1]+mu)/D1 -mu*(y[1]-(1-mu))/D2
  dy4 <- y[2] - 2*y[3]-(1-mu)*y[2]/D1 - mu*y[2]/D2

  list(c(dy1, dy2, dy3, dy4))
}

#-----------------------------
# parameters, initial values and times
#-----------------------------
mu    <- 0.012277471

yini  <- c(x = 0.994, y = 0, dx = 0, dy = -2.00158510637908252240537862224)

times <- seq(0, 18, 0.01)

#-----------------------------
# solve the model
#-----------------------------

out <- dopri853 (times = times, y = yini, func = Arenstorff, parms = NULL,
  rtol = 1e-17, atol = 1e-17)

plot(out[,c("x", "y")], type = "l", lwd = 2, main = "Arenstorff")

#-----------------------------
# First and last value should be the same
#-----------------------------

times <- c(0, 17.065216560157962558891720624 9)

Test  <- dopri853 (times = times, y = yini, func = Arenstorff, parms = NULL)

diagnostics(Test)
```

E5 *E5 Model for Chemical Pyrolysis, ODE*

Description

It is an ODE, 4 equations

Usage

```
E5 (times = c(0, 10^(seq(-5, 13, by = 0.1))), yini = NULL,
    parms = list(), printmescd = TRUE,
    atol = 1.11e-24, rtol = 1e-6,  maxsteps = 1e5, ...)
```

Arguments

yini	the initial (state) values for the DE system. If y has a name attribute, the names will be used to label the output matrix.
times	time sequence for which output is wanted; the first value of times must be the initial time.
parms	list of parameters that overrule the default parameter values
atol	absolute error tolerance, either a scalar or a vector, one value for each y.
rtol	relative error tolerance, either a scalar or a vector, one value for each y,
printmescd	if TRUE the mixed error significant digits computed using the reference solution at time 1e13 are printed
maxsteps	maximal number of steps per output interval taken by the solver
...	additional arguments passed to the solver .

Details

The default parameters are: A = 7.89e-10, B = 1.1e7, C = 1.13e3, M = 1e6

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in yini, plus an additional column (the first) for the time value.

There will be one row for each element in times unless the solver returns with an unrecoverable error. If yini has a names attribute, it will be used to label the columns of the output value.

Note

This model is implemented in R

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia <mazzia@dm.uniba.it>

References

https://archimede.dm.uniba.it/~testset/

Examples

```
out <- E5()
plot(out, lwd = 2, log = "xy")
# compare with reference solution
out[nrow(out),-1] - reference("E5")
```

---

emep *Emep MSC-W Ozone Chemistry Problem, ODE*

Description

The problem is a stiff system of 66 ordinary differential equations. The 'Mathematics and the Environment' project group at CWI contributed this problem to the test set. The software part of the problem is in the file emep.f available at [MM08].

Usage

```
emep (times = seq(14400, 417600, by = 400), yini = NULL,
      parms = list(), printmescd = TRUE, method = bimd,
      atol = 0.1, rtol = 1e-5, maxsteps = 1e5, ...)
```

Arguments

yini	the initial (state) values for the DE system. If y has a name attribute, the names will be used to label the output matrix.
times	time sequence for which output is wanted; the first value of times must be the initial time.
parms	list of parameters that overrule the default parameter values
rtol	relative error tolerance, either a scalar or a vector, one value for each y,
atol	absolute error tolerance, either a scalar or a vector, one value for each y.
method	the solver to use
printmescd	if TRUE the mixed error significant digits computed using the reference solution at time 417600 are printed
maxsteps	maximal number of steps per output interval taken by the solver
...	additional arguments passed to the solver .

Details

The default parameters are:

c = 1.6e-8 , cs = 2e-12 , cp = 1e-8 , r = 25e3 , rp = 50, lh = 4.45 , ls1 = 2e-3 , ls2 = 5e-4 , ls3 = 5e-4, rg1 = 36.3 , rg2 = 17.3 , rg3 = 17.3 , ri = 50 , rc = 600, gamma = 40.67286402e-9 , delta = 17.7493332

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in yini, plus an additional column (the first) for the time value.

There will be one row for each element in times unless the solver returns with an unrecoverable error. If yini has a names attribute, it will be used to label the columns of the output value.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia <mazzia@dm.uniba.it>

References

https://archimede.dm.uniba.it/~testset/

[MM08] F. Mazzia and C. Magherini. Test Set for Initial Value Problem Solvers, release 2.4. Department of Mathematics, University of Bari and INdAM, Research Unit of Bari, February 2008.

[SASJ93] D. Simpson, Y. Andersson-Skold, and M.E. Jenkin. Updating the chemical scheme for the EMEP MSC-W model: Current status. Report EMEP MSC-W Note 2/93, The Norwegian Meteorological Institute, Oslo, 1993.

[Sim93] D. Simpson. Photochemical model calculations over Europe for two extended summer periods: 1985 and 1989. model results and comparisons with observations. Atmospheric Environment, 27A:921-943, 1993.

[VS94] J.G. Verwer and D. Simpson. Explicit methods for stiff ODEs from atmospheric chemistry. Report NM-R9409, CWI, Amsterdam, 1994.

Examples

```
out <- emep()
plot(out, lwd = 2, col = "darkblue",
  which = c("NO", "NO2", "SO2", "CH4", "O3", "N2O5"))

plot(out, col = "darkblue", lwd = 2, which = 1:16)
mtext(side = 3,line = -1.5, "emep", cex = 1.25, outer = TRUE)

# compare with reference solution (component 36 and 38 not included)
refsol <- reference("emep")
inderr <- c(1:35,37,39:66)
max(abs(out[nrow(out),inderr+1] - refsol[inderr])/refsol[inderr])
```

`fekete` *Elliptic Fekete Points, Mechanical Problem, Index 2 DAE*

Description

The fekete problem computes the elliptic Fekete points.

Usage

```
fekete (times = seq(0, 1e3, by = 10 ), yini = NULL, dyini = NULL,
       parms=list(), printmescd = TRUE, method = mebdfi,
       atol = 1e-6, rtol = 1e-6, maxsteps = 1e+05, ...)
```

Arguments

`yini`	the initial (state) values for the DE system. If `y` has a name attribute, the names will be used to label the output matrix.
`dyini`	the initial derivatives of the state variables of the DE system.
`times`	time sequence for which output is wanted; the first value of `times` must be the initial time.
`parms`	list of parameters that overrule the default parameter values
`method`	the solver to use
`rtol`	relative error tolerance, either a scalar or a vector, one value for each y,
`atol`	absolute error tolerance, either a scalar or a vector, one value for each y.
`printmescd`	if TRUE the mixed error significant digits computed using the reference solution at time 0.1 are printed
`maxsteps`	maximal number of steps per output interval taken by the solver
`...`	additional arguments passed to the solver .

Details

There are no parameters

Value

A matrix of class `deSolve` with up to as many rows as elements in `times` and as many columns as elements in `yini`, plus an additional column (the first) for the time value.

There will be one row for each element in `times` unless the solver returns with an unrecoverable error. If `yini` has a names attribute, it will be used to label the columns of the output value.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia <mazzia@dm.uniba.it>

References

https://archimede.dm.uniba.it/~testset/

Examples

```
out <- fekete()
plot(out, lwd = 2, which = 1:20)

# reference run compared with output at end of interval for first 7 components
out1 <- fekete(times = c(0, 1000))
max(abs(out1[nrow(out1),-1] - reference("fekete")))
```

---

gamd *Generalised Adams IVP Method for DAE*

Description

Solves the initial value problem for stiff or nonstiff systems of either:

- a system of ordinary differential equations (ODE) of the form

$$y' = f(t, y, ...)$$

or

- a system of linearly implicit DAES in the form

$$My' = f(t, y)$$

The R function gamd provides an interface to the Fortran DAE solver gamd, written by Felice Iavernaro and Francesca Mazzia.

It implements the generalized adams methods of order 3-5-7-9 with step size control and continuous output.

The system of DAE's is written as an R function or can be defined in compiled code that has been dynamically loaded.

Usage

```
gamd(y, times, func, parms, nind = c(length(y), 0, 0),
  rtol = 1e-6, atol = 1e-6, jacfunc = NULL, jactype = "fullint",
  mass = NULL, massup = NULL, massdown = NULL, verbose = FALSE,
  hmax = NULL, hini = 0, ynames = TRUE, minord = NULL,
  maxord = NULL, bandup = NULL, banddown = NULL,
  maxsteps = 1e4, maxnewtit = c(12, 18, 26, 36),
  dllname = NULL, initfunc = dllname, initpar = parms,
  rpar = NULL, ipar = NULL, nout=0, outnames = NULL, forcings = NULL,
  initforc = NULL, fcontrol = NULL, ...)
```

Arguments

y	the initial (state) values for the DAE or ODE system. If y has a name attribute, the names will be used to label the output matrix.
times	time sequence for which output is wanted; the first value of times must be the initial time; if only one step is to be taken; set times = NULL.
func	either an R-function that computes the values of the derivatives in the DAE or ODE system (the *model definition*) at time t, or a character string giving the name of a compiled function in a dynamically loaded shared library. If func is an R-function, it must be defined as: func <- function(t, y, parms,...). t is the current time point in the integration, y is the current estimate of the variables in the ODE system. If the initial values y has a names attribute, the names will be available inside func. parms is a vector or list of parameters; ... (optional) are any other arguments passed to the function. The return value of func should be a list, whose first element is a vector containing the derivatives of y with respect to time, and whose next elements are global values that are required at each point in times. The derivatives should be specified in the same order as the state variables y. If func is a string, then dllname must give the name of the shared library (without extension) which must be loaded before gamd() is called. See deSolve package vignette "compiledCode" for more details.
parms	vector or list of parameters used in func or jacfunc.
nind	if a DAE system: a three-valued vector with the number of variables of index 1, 2, 3 respectively. The equations must be defined such that the index 1 variables precede the index 2 variables which in turn precede the index 3 variables. The sum of the variables of different index should equal N, the total number of variables.
rtol	relative error tolerance, either a scalar or an array as long as y. See details.
atol	absolute error tolerance, either a scalar or an array as long as y. See details.
jacfunc	if not NULL, an R function that computes the Jacobian of the system of differential equations dydot(i)/dy(j), or a string giving the name of a function or subroutine in 'dllname' that computes the Jacobian (see vignette "compiledCode" from package deSolve, for more about this option). In some circumstances, supplying jacfunc can speed up the computations, if the system is stiff. The R calling sequence for jacfunc is identical to that of func. If the Jacobian is a full matrix, jacfunc should return a matrix dydot/dy, where the ith row contains the derivative of dy_i/dt with respect to y_j, or a vector containing the matrix elements by columns (the way R and FORTRAN store matrices). If the Jacobian is banded, jacfunc should return a matrix containing only the nonzero bands of the Jacobian, rotated row-wise. See first example.
jactype	the structure of the Jacobian, one of "fullint", "fullusr", "bandusr" or "bandint" - either full or banded and estimated internally or by user.
mass	the mass matrix. If not NULL, the problem is a linearly implicit DAE and defined as $M\, dy/dt = f(t, y)$. If the mass-matrix M is full, it should be of dimension

n^2 where n is the number of y-values; if banded the number of rows should be less than n, and the mass-matrix is stored diagonal-wise with element (i, j) stored in `mass(i - j + mumas + 1, j)`.

If `mass = NULL` then the model is an ODE (default)

`massup` number of non-zero bands above the diagonal of the `mass` matrix, in case it is banded.

`massdown` number of non-zero bands below the diagonal of the `mass` matrix, in case it is banded.

`verbose` if `TRUE`: full output to the screen, e.g. will print the `diagnostiscs` of the integration - see details.

`hmax` an optional maximum value of the integration stepsize. If not specified, `hmax` is set to the largest difference in `times`, to avoid that the simulation possibly ignores short-term events. If 0, no maximal size is specified.

`hini` initial step size to be attempted; if 0, the initial step size is set equal to 1e-6. Usually 1e-3 to 1e-5 is good for stiff equations

`ynames` logical, if `FALSE` names of state variables are not passed to function `func`; this may speed up the simulation especially for multi-D models.

`minord` the minimum order to be allowed, >= 3 and <= 9. `NULL` uses the default, 3.

`maxord` the maximum order to be allowed, >= `minord` and <= 9. `NULL` uses the default, 9.

`bandup` number of non-zero bands above the diagonal, in case the Jacobian is banded.

`banddown` number of non-zero bands below the diagonal, in case the Jacobian is banded.

`maxsteps` maximal number of steps taken by the solver, *for the entire integration*. This is different from the settings of this argument in the solvers from package deSolve!

`maxnewtit` A four-valued vector, with the maximal number of splitting-Newton iterations for the solution of the iplicit system in each step for order 3, 5, 7 and 9 respectively. The default is c(10,18,26,36).

`dllname` a string giving the name of the shared library (without extension) that contains all the compiled function or subroutine definitions refered to in `func` and `jacfunc`. See vignette `"compiledCode"` from package `deSolve`.

`initfunc` if not `NULL`, the name of the initialisation function (which initialises values of parameters), as provided in '`dllname`'. See vignette `"compiledCode"` from package `deSolve`.

`initpar` only when '`dllname`' is specified and an initialisation function `initfunc` is in the dll: the parameters passed to the initialiser, to initialise the common blocks (FORTRAN) or global variables (C, C++).

`rpar` only when '`dllname`' is specified: a vector with double precision values passed to the dll-functions whose names are specified by `func` and `jacfunc`.

`ipar` only when '`dllname`' is specified: a vector with integer values passed to the dll-functions whose names are specified by `func` and `jacfunc`.

`nout` only used if `dllname` is specified and the model is defined in compiled code: the number of output variables calculated in the compiled function `func`, present in the shared library. Note: it is not automatically checked whether this is indeed the number of output variables calculed in the dll - you have to perform this check in the code - See vignette `"compiledCode"` from package `deSolve`.

outnames	only used if 'dllname' is specified and nout > 0: the names of output variables calculated in the compiled function func, present in the shared library. These names will be used to label the output matrix.
forcings	only used if 'dllname' is specified: a list with the forcing function data sets, each present as a two-columned matrix, with (time,value); interpolation outside the interval [min(times), max(times)] is done by taking the value at the closest data extreme. See forcings or package vignette "compiledCode".
initforc	if not NULL, the name of the forcing function initialisation function, as provided in 'dllname'. It MUST be present if forcings has been given a value. See forcings or package vignette "compiledCode".
fcontrol	A list of control parameters for the forcing functions. See forcings or vignette compiledCode.
...	additional arguments passed to func and jacfunc allowing this to be a generic function.

Details

The work is done by the FORTRAN 90 subroutine gamd, whose documentation should be consulted for details. The implementation is based on the Fortran 90 version from 2007/24/05.

There are four standard choices for the jacobian which can be specified with jactype.

The options for **jactype** are

jactype = "fullint" a full Jacobian, calculated internally by the solver.

jactype = "fullusr" a full Jacobian, specified by user function jacfunc.

jactype = "bandusr" a banded Jacobian, specified by user function jacfunc; the size of the bands specified by bandup and banddown.

jactype = "bandint" a banded Jacobian, calculated by gamd; the size of the bands specified by bandup and banddown.

Inspection of the example below shows how to specify both a banded and full Jacobian.

The input parameters rtol, and atol determine the **error control** performed by the solver, which roughly keeps the local error of y(i) below rtol(i)*abs(y(i))+atol(i).

The diagnostics of the integration can be printed to screen by calling diagnostics. If verbose = TRUE, the diagnostics will written to the screen at the end of the integration.

See vignette("deSolve") from the deSolve package for an explanation of each element in the vectors containing the diagnostic properties and how to directly access them.

Models may be defined in compiled C or FORTRAN code, as well as in an R-function. See package vignette "compiledCode" from package deSolve for details.

Information about linking forcing functions to compiled code is in forcings (from package deSolve).

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in y plus the number of "global" values returned in the next elements of the return from func, plus and additional column for the time value. There will be a row for each element in times unless the FORTRAN routine 'gamd' returns with an unrecoverable error. If y has a names attribute, it will be used to label the columns of the output value.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia

References

L.Brugnano, D.Trigiante, Solving Differential Problems by Multistep Initial and Boundary Value Methods, Gordon & Breach, Amsterdam, 1998.

F.Iavernaro, F.Mazzia, Block-Boundary Value Methods for the solution of Ordinary Differential Equation. Siam J. Sci. Comput. 21 (1) (1999) 323–339.

F.Iavernaro, F.Mazzia, Solving Ordinary Differential Equations by Generalized Adams Methods: properties and implementation techniques, proceedings of NUMDIFF8, Appl. Num. Math. 28 (2-4) (1998) 107-126.

See Also

- bimd another DAE solver from package deTestSet,
- mebdfi another DAE solver from package deTestSet,
- daspk another DAE solver from package deSolve,
- ode for a general interface to most of the ODE solvers from package deSolve,
- ode.1D for integrating 1-D models,
- ode.2D for integrating 2-D models,
- ode.3D for integrating 3-D models,
- mebdfi for integrating DAE models,
- dopri853 for the Dormand-Prince Runge-Kutta method of order 8(53)

diagnostics to print diagnostic messages.

Examples

```
## =============================================================================
## Example 1:
##   Various ways to solve the same model.
## =============================================================================

## the model, 5 state variables
f1 <- function  (t, y, parms)
{
  ydot <- vector(len = 5)
```

```
  ydot[1] <-  0.1*y[1] -0.2*y[2]
  ydot[2] <- -0.3*y[1] +0.1*y[2] -0.2*y[3]
  ydot[3] <-           -0.3*y[2] +0.1*y[3] -0.2*y[4]
  ydot[4] <-                     -0.3*y[3] +0.1*y[4] -0.2*y[5]
  ydot[5] <-                               -0.3*y[4] +0.1*y[5]

  return(list(ydot))
}

## the Jacobian, written as a full matrix
fulljac <- function  (t, y, parms)
{
   jac <- matrix(nrow = 5, ncol = 5, byrow = TRUE,
                data = c(0.1, -0.2,  0  ,  0  ,  0  ,
                        -0.3,  0.1, -0.2,  0  ,  0  ,
                         0  , -0.3,  0.1, -0.2,  0  ,
                         0  ,  0  , -0.3,  0.1, -0.2,
                         0  ,  0  ,  0  , -0.3,  0.1)    )
   return(jac)
}

## the Jacobian, written in banded form
bandjac <- function  (t, y, parms)
{
   jac <- matrix(nrow = 3, ncol = 5, byrow = TRUE,
                 data = c( 0  , -0.2, -0.2, -0.2, -0.2,
                           0.1,  0.1,  0.1,  0.1,  0.1,
                          -0.3, -0.3, -0.3, -0.3,    0)    )
   return(jac)
}

## initial conditions and output times
yini  <- 1:5
times <- 1:20

## default: stiff method, internally generated, full Jacobian
out   <- gamd(yini, times, f1, parms = 0, jactype = "fullint")
plot(out)

## stiff method, user-generated full Jacobian
out2  <- gamd(yini, times, f1, parms = 0, jactype = "fullusr",
              jacfunc = fulljac)

## stiff method, internally-generated banded Jacobian
## one nonzero band above (up) and below(down) the diagonal
out3  <- gamd(yini, times, f1, parms = 0, jactype = "bandint",
                              bandup = 1, banddown = 1)

## stiff method, user-generated banded Jacobian
out4  <- gamd(yini, times, f1, parms = 0, jactype = "bandusr",
              jacfunc = bandjac, bandup = 1, banddown = 1)
```

```
## ========================================================================
## Example 2:
##   stiff problem from chemical kinetics
## ========================================================================
Chemistry <- function (t, y, p) {
     dy1 <- -.04*y[1] + 1.e4*y[2]*y[3]
     dy2 <- .04*y[1] - 1.e4*y[2]*y[3] - 3.e7*y[2]^2
     dy3 <- 3.e7*y[2]^2
     list(c(dy1,dy2,dy3))
}

times <- 10^(seq(0, 10, by = 0.1))
yini <- c(y1 = 1.0, y2 = 0, y3 = 0)

out <- gamd(func = Chemistry, times = times, y = yini, parms = NULL)
plot(out, log = "x", type = "l", lwd = 2)


## ========================================================================
## Example 3: DAE
## Car axis problem, index 3 DAE, 8 differential, 2 algebraic equations
## from
## F. Mazzia and C. Magherini. Test Set for Initial Value Problem Solvers,
## release 2.4. Department
## of Mathematics, University of Bari and INdAM, Research Unit of Bari,
## February 2008.
## Available at http://www.dm.uniba.it/~testset.
## ========================================================================

## Problem is written as M*y = f(t,y,p).
## caraxisfun implements the right-hand side:

caraxisfun <- function(t, y, parms) {
  with(as.list(y), {

    yb <- r * sin(w * t)
    xb <- sqrt(L * L - yb * yb)
    Ll <- sqrt(xl^2 + yl^2)
    Lr <- sqrt((xr - xb)^2 + (yr - yb)^2)

    dxl <- ul; dyl <- vl; dxr <- ur; dyr <- vr

    dul  <- (L0-Ll) * xl/Ll      + 2 * lam2 * (xl-xr) + lam1*xb
    dvl  <- (L0-Ll) * yl/Ll      + 2 * lam2 * (yl-yr) + lam1*yb - k * g

    dur  <- (L0-Lr) * (xr-xb)/Lr - 2 * lam2 * (xl-xr)
    dvr  <- (L0-Lr) * (yr-yb)/Lr - 2 * lam2 * (yl-yr) - k * g

    c1   <- xb * xl + yb * yl
    c2   <- (xl - xr)^2 + (yl - yr)^2 - L * L

    list(c(dxl, dyl, dxr, dyr, dul, dvl, dur, dvr, c1, c2))
```

```
  })
}

eps <- 0.01; M <- 10; k <- M * eps^2/2;
L <- 1; L0 <- 0.5; r <- 0.1; w <- 10; g <- 1

yini <- c(xl = 0, yl = L0, xr = L, yr = L0,
          ul = -L0/L, vl = 0,
          ur = -L0/L, vr = 0,
          lam1 = 0, lam2 = 0)

# the mass matrix
Mass      <- diag(nrow = 10, 1)
Mass[5,5] <- Mass[6,6] <- Mass[7,7] <- Mass[8,8] <- M * eps * eps/2
Mass[9,9] <- Mass[10,10] <- 0
Mass

# index of the variables: 4 of index 1, 4 of index 2, 2 of index 3
index <- c(4, 4, 2)

times <- seq(0, 3, by = 0.01)
out <- gamd(y = yini, mass = Mass, times = times, func = caraxisfun,
        parms = NULL, nind = index)

plot(out, which = 1:4, type = "l", lwd = 2)
```

`hires` *High Irradiance Response model, from Plant Physiology, ODE*

Description

This IVP is a stiff system of 8 non-linear Ordinary Differential Equations.

It was proposed by Schafer in 1975 [Sch75].

The name HIRES was given by Hairer & Wanner [HW96]. It refers to 'High Irradiance RESponse', which is described by this ODE.

The parallel-IVP-algorithm group of CWI contributed this problem to the test set. The software part of the problem is in the file hires.f available at [MM08].

Usage

```
hires (yini = NULL, times = seq(0, 321.8122, by = 321.8122/500),
       parms = list(), printmescd = TRUE, method = mebdfi,
       atol = 1e-6, rtol = 1e-6, ...)
```

Arguments

yini	the initial (state) values for the DE system. If y has a name attribute, the names will be used to label the output matrix.
times	time sequence for which output is wanted; the first value of times must be the initial time.
atol	absolute error tolerance, either a scalar or a vector, one value for each y.
rtol	relative error tolerance, either a scalar or a vector, one value for each y,
method	the solver to use
parms	list of parameters that overrule the default parameter values
printmescd	if TRUE the mixed error significant digits computed using the reference solution at time 321.8122 are printed
...	additional arguments passed to the solver .

Details

The default parameters are: k1 = 1.71, k2 = 0.43, k3 = 8.32, k4 = 0.69, k5 = 0.035, k6 = 8.32, k7 = 280, k8 = 0.69, k9 = 0.69, Oks = 0.0007

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in yini, plus an additional column (the first) for the time value.

There will be one row for each element in times unless the solver returns with an unrecoverable error. If yini has a names attribute, it will be used to label the columns of the output value.

Note

This model is implemented in R

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia <mazzia@dm.uniba.it>

References

https://archimede.dm.uniba.it/~testset/

[Got77] B.A. Gottwald. MISS - ein einfaches Simulations-System fur biologische und chemische Prozesse. EDV in Medizin und Biologie, 3:85-90, 1977.

[HW96] E. Hairer and G. Wanner. Solving Ordinary Differential Equations II: Stiff and Differential-algebraic Problems. Springer-Verlag, second revised edition, 1996.

[MM08] F. Mazzia and C. Magherini. Test Set for Initial Value Problem Solvers, release 2.4. Department of Mathematics, University of Bari and INdAM, Research Unit of Bari, February 2008. Available at http://www.dm.uniba.it/testset.

[Sch75] E. Schafer. A new approach to explain the 'high irradiance responses' of photomorphogenesis on the basis of phytochrome. J. of Math. Biology, 2:41 - 56, 1975.

[SL98] J.J.B. de Swart and W.M. Lioen. Collecting real-life problems to test solvers for implicit differential equations. CWI Quarterly, 11(1):83 - 100, 1998.

Examples

```
out <- hires()
plot(out, lwd = 2)

# compare with reference solution
out1 <- hires(times = c(0, 321.8122))

max(abs(out1[nrow(out1),-1] - reference("hires")))
```

`mebdfi` *Solver for Differential Algebraic Equations (DAE) up to index 3*

Description

Solves either:

- a system of ordinary differential equations (ODE) of the form

$$y' = f(t, y, ...)$$

  or

- a system of differential algebraic equations (DAE) of the form

$$F(t, y, y') = 0$$

  or

- a system of linearly implicit DAES in the form

$$My' = f(t, y)$$

using the Modified Extended Backward Differentiation formulas for stiff fully implicit inital value problems

These formulas increase the absolute stability regions of the classical BDFs.

The orders of the implemented formulae range from 1 to 8.

The R function `mebdfi` provides an interface to the Fortran DAE solver of the same name, written by T.J. Abdulla and J.R. Cash.

The system of DE's is written as an R function or can be defined in compiled code that has been dynamically loaded.

Usage

```
mebdfi(y, times, func = NULL, parms, dy = NULL, res = NULL,
  nind=c(length(y),0,0), rtol = 1e-6, atol = 1e-6, jacfunc = NULL,
  jacres = NULL, jactype = "fullint", mass = NULL, verbose = FALSE,
  tcrit = NULL,  hini = 0, ynames = TRUE, maxord = 7, bandup = NULL,
  banddown = NULL, maxsteps = 5000, dllname = NULL,
  initfunc = dllname, initpar = parms, rpar = NULL,
  ipar = NULL, nout = 0, outnames = NULL,
  forcings=NULL, initforc = NULL, fcontrol=NULL,  ...)
```

Arguments

y	the initial (state) values for the DE system. If y has a name attribute, the names will be used to label the output matrix.
times	time sequence for which output is wanted; the first value of times must be the initial time; if only one step is to be taken; set times = NULL.
func	cannot be used if the model is a DAE system. If an ODE system, func should be an R-function that computes the values of the derivatives in the ODE system (the *model definition*) at time t. func must be defined as: func <- function(t, y, parms,...). t is the current time point in the integration, y is the current estimate of the variables in the ODE or DAE system. If the initial values y has a names attribute, the names will be available inside func, unless ynames is FALSE. parms is a vector or list of parameters. ... (optional) are any other arguments passed to the function. The return value of func should be a list, whose first element is a vector containing the derivatives of y with respect to time, and whose next elements are global values that are required at each point in times. The derivatives should be specified in the same order as the specification of the state variables y. Note that it is not possible to define func as a compiled function in a dynamically loaded shared library. Use res instead.
parms	vector or list of parameters used in func, jacfunc, or res
dy	the initial derivatives of the state variables of the DE system. Ignored if an ODE.
res	if a DAE system: either an R-function that computes the residual function F(t,y,y') of the DAE system (the model defininition) at time t, or a character string giving the name of a compiled function in a dynamically loaded shared library. If res is a user-supplied R-function, it must be defined as: res <- function(t, y, dy, parms, ...). Here t is the current time point in the integration, y is the current estimate of the variables in the DAE system, dy are the corresponding derivatives. If the initial y or dy have a names attribute, the names will be available inside res, unless ynames is FALSE. parms is a vector of parameters. The return value of res should be a list, whose first element is a vector containing the residuals of the DAE system, i.e. delta = F(t,y,y'), and whose next elements contain output variables that are required at each point in times.

If res is a string, then dllname must give the name of the shared library (without extension) which must be loaded before mebdfi() is called (see package vignette "compiledCode" for more information).

nind — if a DAE system: a three-valued vector with the number of variables of index 1, 2, 3 respectively. The equations must be defined such that the index 1 variables precede the index 2 variables which in turn precede the index 3 variables. The sum of the variables of different index should equal N, the total number of variables.

rtol — relative error tolerance, either a scalar or a vector, one value for each y,

atol — absolute error tolerance, either a scalar or a vector, one value for each y.

jacfunc — if not NULL, an R function that computes the Jacobian of the system of differential equations. Only used in case the system is an ODE (y' = f(t,y)), specified by func. The R calling sequence for jacfunc is identical to that of func.

If the Jacobian is a full matrix, jacfunc should return a matrix dydot/dy, where the ith row contains the derivative of dy_i/dt with respect to y_j, or a vector containing the matrix elements by columns (the way R and Fortran store matrices).

If the Jacobian is banded, jacfunc should return a matrix containing only the nonzero bands of the Jacobian, rotated row-wise. See first example of lsode.

jacres — jacres and not jacfunc should be used if the system is specified by the residual function F(t,y,y'), i.e. jacres is used in conjunction with res.

If jacres is an R-function, the calling sequence for jacres is identical to that of res, but with extra parameter cj. Thus it should be defined as: jacres <- function(t, y, dy, parms, cj, ...). Here t is the current time point in the integration, y is the current estimate of the variables in the ODE system, y' are the corresponding derivatives and cj is a scalar, which is normally proportional to the inverse of the stepsize. If the initial y or dy have a names attribute, the names will be available inside jacres, unless ynames is FALSE. parms is a vector of parameters (which may have a names attribute).

If the Jacobian is a full matrix, jacres should return the matrix dG/dy + cj*dG/dyprime, where the ith row is the sum of the derivatives of G_i with respect to y_j and the scaled derivatives of G_i with respect to dy_j.

If the Jacobian is banded, jacres should return only the nonzero bands of the Jacobian, rotated rowwise. See details for the calling sequence when jacres is a string.

jactype — the structure of the Jacobian, one of "fullint", "fullusr", "bandusr" or "bandint" - either full or banded and estimated internally or by the user.

mass — the mass matrix. If not NULL, the problem is a linearly implicit DAE and defined as $massdy/dt = f(t, y)$. The mass-matrix should be of dimension n*n where n is the number of y-values.

If mass=NULL then the model is either an ODE or a DAE, specified with res

verbose — if TRUE: full output to the screen, e.g. will print the diagnostiscs of the integration - see details.

tcrit — the Fortran routine mebdfi overshoots its targets (times points in the vector times), and interpolates values for the desired time points. If there is a time beyond which integration should not proceed (perhaps because of a singularity), that should be provided in tcrit.

hini	initial step size to be attempted; if 0, the initial step size is set to 1e-6, but it may be better to set it equal to rtol. The solver is quite sensitive to values of hini; sometimes if it fails, it helps to decrease/increase hini
ynames	logical, if FALSE names of state variables are not passed to function func; this may speed up the simulation especially for large models.
maxord	the maximum order to be allowed, an integer between 2 and 7. The default is maxord = 7, but values of 4-5 may be better for difficult problems; hihger order methods are more efficient but less stable.
bandup	number of non-zero bands above the diagonal, in case the Jacobian is banded (and jactype one of "bandint","bandusr")
banddown	number of non-zero bands below the diagonal, in case the Jacobian is banded (and jactype one of "bandint","bandusr")
maxsteps	maximal number of steps per output interval taken by the solver.
dllname	a string giving the name of the shared library (without extension) that contains all the compiled function or subroutine definitions referred to in res and jacres. See package vignette "compiledCode".
initfunc	if not NULL, the name of the initialisation function (which initialises values of parameters), as provided in 'dllname'. See package vignette "compiledCode".
initpar	only when 'dllname' is specified and an initialisation function initfunc is in the dll: the parameters passed to the initialiser, to initialise the common blocks (fortran) or global variables (C, C++).
rpar	only when 'dllname' is specified: a vector with double precision values passed to the dll-functions whose names are specified by res and jacres.
ipar	only when 'dllname' is specified: a vector with integer values passed to the dll-functions whose names are specified by res and jacres.
nout	only used if 'dllname' is specified and the model is defined in compiled code: the number of output variables calculated in the compiled function res, present in the shared library. Note: it is not automatically checked whether this is indeed the number of output variables calculed in the dll - you have to perform this check in the code - See package vignette "compiledCode".
outnames	only used if 'dllname' is specified and nout > 0: the names of output variables calculated in the compiled function res, present in the shared library. These names will be used to label the output matrix.
forcings	only used if 'dllname' is specified: a list with the forcing function data sets, each present as a two-columned matrix, with (time,value); interpolation outside the interval [min(times), max(times)] is done by taking the value at the closest data extreme. See package vignette "compiledCode".
initforc	if not NULL, the name of the forcing function initialisation function, as provided in 'dllname'. It MUST be present if forcings has been given a value. See package vignette "compiledCode".
fcontrol	A list of control parameters for the forcing functions. vignette compiledCode from package deSolve.
...	additional arguments passed to func, jacfunc, res and jacres, allowing this to be a generic function.

Details

The mebdfi solver uses modified extended backward differentiation formulas of orders one through eight (specified with `maxord`) to solve either:

- an ODE system of the form
$$y' = f(t, y, ...)$$
for y = Y, or
- a DAE system of the form
$$F(t, y, y') = 0$$
for y = Y and y' = YPRIME.

The recommended value of `maxord` is eight, unless it is believed that there are severe stability problems in which case `maxord` = 4 or 5 should be tried instead.

ODEs are specified in `func`, DAEs are specified in `res`.

If a DAE system, Values for Y *and* YPRIME at the initial time must be given as input. Ideally,these values should be consistent, that is, if T, Y, YPRIME are the given initial values, they should satisfy F(T,Y,YPRIME) = 0.

The form of the **Jacobian** can be specified by `jactype`. This is one of:

jactype = "fullint": a full Jacobian, calculated internally by `mebdfi`, the default,

jactype = "fullusr": a full Jacobian, specified by user function `jacfunc` or `jacres`,

jactype = "bandusr": a banded Jacobian, specified by user function `jacfunc` or `jacres`; the size of the bands specified by `bandup` and `banddown`,

jactype = "bandint": a banded Jacobian, calculated by `mebdfi`; the size of the bands specified by `bandup` and `banddown`.

If `jactype` = "fullusr" or "bandusr" then the user must supply a subroutine `jacfunc`.

If jactype = "fullusr" or "bandusr" then the user must supply a subroutine `jacfunc` or `jacres`.

The input parameters `rtol`, and `atol` determine the **error control** performed by the solver. If the request for precision exceeds the capabilities of the machine, mebdfi will return an error code.

res and jacres may be defined in compiled C or Fortran code, as well as in an R-function. See deSolve's vignette `"compiledCode"` for details. Examples in Fortran are in the 'dynload' subdirectory of the `deSolve` package directory.

The diagnostics of the integration can be printed to screen by calling `diagnostics`. If `verbose` = `TRUE`, the diagnostics will written to the screen at the end of the integration.

See vignette("deSolve") for an explanation of each element in the vectors containing the diagnostic properties and how to directly access them.

Value

A matrix of class `deSolve` with up to as many rows as elements in `times` and as many columns as elements in `y` plus the number of "global" values returned in the next elements of the return from `func` or `res`, plus an additional column (the first) for the time value. There will be one row for each element in `times` unless the Fortran routine 'mebdfi' returns with an unrecoverable error. If `y` has a names attribute, it will be used to label the columns of the output value.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Jeff Cash

References

J. R. Cash, The integration of stiff initial value problems in O.D.E.S using modified extended backward differentiation formulae, Comp. and Maths. with applics., 9, 645-657, (1983).

J.R. Cash and S. Considine, an MEBDF code for stiff initial value problems, ACM Trans Math Software, 142-158, (1992).

J.R. Cash, Stable recursions with applications to the numerical solution of stiff systems, Academic Press,(1979).

See Also

- gamd and bimd two other DAE solvers,
- daspk another DAE solver from package deSolve,

diagnostics to print diagnostic messages.

Examples

```
## ======================================================================
## Coupled chemical reactions including an equilibrium
## modeled as (1) an ODE and (2) as a DAE
##
## The model describes three chemical species A,B,D:
## subjected to equilibrium reaction D <- > A + B
## D is produced at a constant rate, prod
## B is consumed at 1s-t order rate, r
## Chemical problem formulation 1: ODE
## ======================================================================

## Dissociation constant
K <- 1

## parameters
pars <- c(
        ka   = 1e6,     # forward rate
        r    = 1,
        prod = 0.1)


Fun_ODE <- function (t, y, pars)
{
  with (as.list(c(y, pars)), {
    ra  <- ka*D        # forward rate
    rb  <- ka/K *A*B   # backward rate

    ## rates of changes
```

```
    dD  <- -ra + rb + prod
    dA  <-  ra - rb
    dB  <-  ra - rb - r*B
    return(list(dy = c(dA, dB, dD),
                CONC = A+B+D))
  })
}

## =============================================================================
## Chemical problem formulation 2: DAE
## 1. get rid of the fast reactions ra and rb by taking
## linear combinations   : dD+dA = prod (res1) and
##                         dB-dA = -r*B (res2)
## 2. In addition, the equilibrium condition (eq) reads:
## as ra = rb : ka*D = ka/K*A*B = >      K*D = A*B
## =============================================================================

Res_DAE <- function (t, y, yprime, pars)
{
  with (as.list(c(y, yprime, pars)), {
    ## residuals of lumped rates of changes
    res1 <- -dD - dA + prod
    res2 <- -dB + dA - r*B

    ## and the equilibrium equation
    eq   <- K*D - A*B

    return(list(c(res1, res2, eq),
                CONC = A+B+D))
  })
}

times <- seq(0, 100, by = 1)

## Initial conc; D is in equilibrium with A,B
y     <- c(A = 2, B = 3, D = 2*3/K)

## ODE model solved with mebdfi
ODE <- as.data.frame(mebdfi(y = y, times = times, func = Fun_ODE,
                     parms = pars, atol = 1e-8, rtol = 1e-8))

## Initial rate of change
dy  <- c(dA = 0, dB = 0, dD = 0)
## DAE model solved with mebdfi
DAE <- as.data.frame(mebdfi(y = y, dy = dy, times = times,
         res = Res_DAE, parms = pars, atol = 1e-8, rtol = 1e-8))


## =============================================================================
## Chemical problem formulation 3: Mass * Func
## Based on the DAE formulation
## =============================================================================
```

```
Mass_FUN <- function (t, y, pars)
{
  with (as.list(c(y, pars)), {

    ## as above, but without the
    f1 <- prod
    f2 <- - r*B

    ## and the equilibrium equation
    f3   <- K*D - A*B

    return(list(c(f1, f2, f3),
                CONC = A+B+D))
  })
}
Mass <- matrix(nr=3, nc=3, byrow = TRUE,
  data=c(1,  0, 1,          # dA + 0 + dB
        -1,  1, 0,          # -dA + dB +0
         0,  0, 0))         # algebraic

times <- seq(0, 100, by = 2)

## Initial conc; D is in equilibrium with A,B
y     <- c(A = 2, B = 3, D = 2*3/K)

## ODE model solved with daspk
ODE <- as.data.frame(daspk(y = y, times = times, func = Fun_ODE,
                     parms = pars, atol = 1e-10, rtol = 1e-10))

## Initial rate of change
dy  <- c(dA = 0, dB = 0, dD = 0)

## DAE model solved with daspk
DAE <- as.data.frame(daspk(y = y, dy = dy, times = times,
         res = Res_DAE, parms = pars, atol = 1e-10, rtol = 1e-10))

MASS<- mebdfi(y = y, times = times, func = Mass_FUN,
              parms = pars, mass = Mass)

## ================
## plotting output
## ================
opa <- par(mfrow = c(2, 2))

for (i in 2:5)
{
plot(ODE$time, ODE[, i], xlab = "time",
     ylab = "conc", main = names(ODE)[i], type = "l")
points(DAE$time, DAE[,i], col = "red")
}
legend("bottomright",lty = c(1,NA),pch = c(NA,1),
       col = c("black","red"),legend = c("ODE","DAE"))
```

```
# difference between both implementations:
max(abs(ODE-DAE))

par(mfrow = opa)

## =====================================================================
##
## Example 3: higher index DAE
##
## Car axis problem, index 3 DAE, 8 differential, 2 algebraic equations
## from
## F. Mazzia and C. Magherini. Test Set for Initial Value Problem Solvers,
## release 2.4. Department
## of Mathematics, University of Bari and INdAM, Research Unit of Bari,
## February 2008.
## Available at http://www.dm.uniba.it/~testset.
## =====================================================================

# car returns the residuals of the implicit DAE
car <- function(t, y, dy, pars){
  with(as.list(c(pars, y)), {
      f <- rep(0, 10)

      yb  <- r*sin(w*t)
      xb  <- sqrt(L*L - yb*yb)
      Ll  <- sqrt(xl^2 + yl^2)
      Lr  <- sqrt((xr-xb)^2 + (yr-yb)^2)

      f[1:4] <- y[5:8]
      k <- M*eps*eps/2

      f[5]  <- (L0-Ll)*xl/Ll + lam1*xb+2*lam2*(xl-xr)
      f[6]  <- (L0-Ll)*yl/Ll + lam1*yb+2*lam2*(yl-yr)-k*g
      f[7]  <- (L0-Lr)*(xr-xb)/Lr - 2*lam2*(xl-xr)
      f[8]  <- (L0-Lr)*(yr-yb)/Lr - 2*lam2*(yl-yr)-k*g

      f[9]  <- xb*xl+yb*yl
      f[10] <- (xl-xr)^2+(yl-yr)^2-L*L

      delt       <- dy-f
      delt[5:8]  <- k*dy[5:8]-f[5:8]
      delt[9:10] <- -f[9:10]

      list(delt=delt,f=f)
  })
}


# parameters
pars <- c(eps = 1e-2, M = 10, L = 1, L0 = 0.5,
          r   = 0.1,  w = 10, g = 1)

# initial conditions: state variables
```

```
yini <-  with (as.list(pars),
   c(xl = 0, yl = L0, xr = L, yr = L0, xla = -L0/L,
     yla = 0, xra = -L0/L, yra = 0, lam1 = 0, lam2 = 0)
              )

# initial conditions: derivates
dyini <- rep(0, 10)
FF    <- car(0, yini, dyini, pars)
dyini[1:4] <- yini[5:8]
dyini[5:8] <- 2/pars["M"]/(pars["eps"])^2*FF$f[5:8]

# check consistency of initial condition: delt should be = 0.
car(0, yini, dyini, pars)

# running the model
times <- seq(0, 3, by = 0.01)
nind  <- c(4, 4, 2)   # index 1, 2 and 3 variables
out   <- mebdfi(y = yini, dy = dyini, times, res = car, parms = pars,
                nind = nind, rtol = 1e-5, atol = 1e-5)

plot(out, which = 1:4, type = "l", lwd=2)

mtext(outer = TRUE, side = 3, line = -0.5, cex = 1.5, "car axis")
```

---

nand *Nand Gate, Index 1 IDE*

Description

It is an index 1 IDE, 14 equations

Usage

```
nand  (times = 0:80, yini =NULL, dyini = NULL,
       parms = list(), printmescd = TRUE, method = mebdfi,
       atol = 1e-6, rtol = 1e-6, maxsteps = 1e5, ...)
```

Arguments

yini	the initial (state) values for the DE system. If y has a name attribute, the names will be used to label the output matrix.
dyini	the initial derivatives of the state variables of the DE system.
times	time sequence for which output is wanted; the first value of times must be the initial time.
parms	list of parameters that overrule the default parameter values
method	the solver to use; only mebdfi available for now
maxsteps	maximal number of steps per output interval taken by the solver

rtol	relative error tolerance, either a scalar or a vector, one value for each y,
atol	absolute error tolerance, either a scalar or a vector, one value for each y.
printmescd	if TRUE the mixed error significant digits computed using the reference solution at time 417600 are printed
...	additional arguments passed to the solver .

Details

The default parameters are: RGS = 4, RGD = 4, RBS = 10, RBD = 10, CGS = 0.6e-4, CGD = 0.6e-4, CBD = 2.4e-5, CBS = 2.4e-5, C9 = 0.5e-4, DELTA = 0.2e-1, CURIS = 1.e-14, VTH = 25.85, VDD = 5., VBB = -2.5

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in yini, plus an additional column (the first) for the time value.

There will be one row for each element in times unless the solver returns with an unrecoverable error. If yini has a names attribute, it will be used to label the columns of the output value.

Note

This model is implemented in FORTRAN

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia

References

https://archimede.dm.uniba.it/~testset/

Examples

```
out <- nand(method = "daspk")
plot(out, lwd = 2, which = 1:9)

# compare with reference solution
max(abs(out[nrow(out),-1] - reference("nand")))
```

orego *The Oregonator Chemistry Model, ODE*

Description

Chemical model implementing the Belousov-Zhabotinskii reaction.

It is an ODE, 3 equations

Usage

```
orego (times = 0:360, yini = NULL,
       parms = list(), printmescd = TRUE,
       atol = 1e-6, rtol = 1e-6, ...)
```

Arguments

yini	the initial (state) values for the DE system. If y has a name attribute, the names will be used to label the output matrix.
times	time sequence for which output is wanted; the first value of times must be the initial time.
parms	list of parameters that overrule the default parameter values
atol	absolute error tolerance, either a scalar or a vector, one value for each y.
rtol	relative error tolerance, either a scalar or a vector, one value for each y,
printmescd	if TRUE the mixed error significant digits computed using the reference solution at time 1e13 are printed
...	additional arguments passed to the solver .

Details

The default parameters are: k1 = 77.27, k2 = 8.375e-6, k3 = 77.27, k4 = 0.161

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in yini, plus an additional column (the first) for the time value.

There will be one row for each element in times unless the solver returns with an unrecoverable error. If yini has a names attribute, it will be used to label the columns of the output value.

Note

This model is implemented in R

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia

References

https://archimede.dm.uniba.it/~testset/

Examples

```
out <- orego()
plot(out, lwd = 2, log = "y")

# compare with exact solution
out[nrow(out),-1] - reference("orego")
```

`pleiades` *Motion of Inextensible Elastic Beam, ODE*

Description

The pleiades problem is a problem from celestial mechanics, describing the motion of seven stars in the plane of coordinates xi, yi and masses mi = i (i = 1, ... 7).

It is a set of nonstiff ordinary differential equations of dimension 28.

Usage

```
pleiades (times = seq(0, 3.0, by = 0.01), yini = NULL,
          printmescd = TRUE, method = lsoda,
          atol = 1e-6, rtol = 1e-6, ...)
```

Arguments

`yini`	the initial (state) values for the DE system. If `y` has a name attribute, the names will be used to label the output matrix.
`times`	time sequence for which output is wanted; the first value of `times` must be the initial time.
`method`	the solver to use
`atol`	absolute error tolerance, either a scalar or a vector, one value for each y.
`rtol`	relative error tolerance, either a scalar or a vector, one value for each y,
`printmescd`	if TRUE the mixed error significant digits computed using the reference solution at time 1e13 are printed
`...`	additional arguments passed to the solver .

Details

There are no parameters

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in yini, plus an additional column (the first) for the time value.

There will be one row for each element in times unless the solver returns with an unrecoverable error. If yini has a names attribute, it will be used to label the columns of the output value.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia

References

https://archimede.dm.uniba.it/~testset/

Examples

```
out <- pleiades()
par(mfrow = c(3,3))
for (i in 1:7) plot(out[,i+1], out[,i+8], type = "l", main = paste("body ",i),
     xlab = "x", ylab = "y")


plot(0, 0 , type = "n", main = "ALL",
     xlab = "x", ylab = "y", xlim = c(-3, 4), ylim = c(-4, 5))
for (i in 1:7) lines(out[,i+1], out[,i+8], col = i, lwd = 2)

# compare with reference solution
max(abs(out[nrow(out),-1]- reference("pleiades")))
```

---

pollution *Pollution Problem, from Chemistry, ODE*

Description

This IVP is a stiff system of 20 non-linear Ordinary Differential Equations.

It is the chemical reaction part of the air pollution model developed at The Dutch National Institute of Public Health and Environmental Protection (RIVM) and it is described by Verwer in [Ver94].

The parallel-IVP-algorithm group of CWI contributed this problem to the test set. The software part of the problem is in the file pollu.f available at [MM08].

Usage

```
pollution (times = seq(0, 60, 1), yini = NULL,
           parms = list(), printmescd = TRUE, method = mebdfi,
           atol = 1e-6, rtol = 1e-6,  ...)
```

Arguments

yini	the initial (state) values for the DE system. If y has a name attribute, the names will be used to label the output matrix.
times	time sequence for which output is wanted; the first value of times must be the initial time.
parms	list of parameters that overrule the default parameter values
method	the solver to use
atol	absolute error tolerance, either a scalar or a vector, one value for each y.
rtol	relative error tolerance, either a scalar or a vector, one value for each y,
printmescd	if TRUE the mixed error significant digits computed using the reference solution at time 1e13 are printed
...	additional arguments passed to the solver .

Details

The default parameters are: k1 = .35, k2 = .266e2, k3 = .123e5, k4 = .86e-3, k5 = .82e-3, k6 = .15e5, k7 = .13e-3, k8 = .24e5,k9 = .165e5, k10 = .9e4, k11 = .22e-1, k12 = .12e5, k13 = .188e1, k14 = .163e5, k15 = .48e7, k16 = .35e-3, k17 = .175e-1, k18 = .1e9, k19 = .444e12, k20 = .124e4, k21 = .21e1, k22 = .578e1, k23 = .474e-1, k24 = .178e4, k25 = .312e1

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in yini, plus an additional column (the first) for the time value.

There will be one row for each element in times unless the solver returns with an unrecoverable error. If yini has a names attribute, it will be used to label the columns of the output value.

Note

This model is implemented in R

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia

References

https://archimede.dm.uniba.it/~testset/

[MM08] F. Mazzia and C. Magherini. Test Set for Initial Value Problem Solvers, release 2.4. Department of Mathematics, University of Bari and INdAM, Research Unit of Bari, February 2008.

[Ver94] J.G. Verwer. Gauss-Seidel iteration for stiff ODEs from chemical kinetics. SIAM J. Sci.bComput., 15(5):1243 – 1259,

Examples

```
out <- pollution()
plot(out, lwd = 2, which = 1:9)

out1 <- pollution(times = 0:60)

# compare with reference solution
max(abs(out1[nrow(out1),-1] -  reference("pollution")))
```

`reference` *Reference Value of Test Set Problems*

Description

Estimates the reference solution of the problem

Usage

```
reference (name = c("andrews", "beam", "caraxis", "crank", "E5",
    "emep", "fekete", "vdpol", "hires", "nand", "orego",
    "pleiades", "pollution", "ring", "rober", "transistor",
    "tube", "twobit", "wheelset"))
```

Arguments

`name`	the name of the problem whose reference solution is to be estimated

Value

A vector with the reference solution

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia

References

https://archimede.dm.uniba.it/~testset/

Examples

```
reference("ring")
```

ring *The Ring Modulator Problem, Electrical Circuit Analysis, ODE*

Description

The probelm describes the behavior of the ring modulator, an electrical circuit.

The type of the problem depends on the parameter Cs. If Cs is not equal 0, then it is a stiff system of 15 non-linear ordinary differential equations.

For Cs = 0 we have a DAE of index 2, consisting of 11 differential equations and 4 algebraic equations. The numerical results presented here refer to Cs = 2 e-12. The problem has been taken from [KRS92], where the approach of Horneber [Hor76] is followed. The parallel-IVP-algorithm group of CWI contributed this problem to the test set. The software part of the problem is in the file ringmod.f available at [MM08].

Usage

```
ring (times = seq(0, 0.001, by = 5e-06), yini = NULL, dyini = NULL,
      parms = list(), printmescd = TRUE, method = mebdfi,
      atol = 1e-8, rtol = 1e-8, maxsteps = 1e+06, ...)
```

Arguments

yini	the initial (state) values for the DE system. If y has a name attribute, the names will be used to label the output matrix.
dyini	the initial derivatives of the state variables of the DE system.
times	time sequence for which output is wanted; the first value of times must be the initial time.
parms	list of parameters that overrule the default parameter values
method	the solver to use
atol	absolute error tolerance, either a scalar or a vector, one value for each y.
rtol	relative error tolerance, either a scalar or a vector, one value for each y,
maxsteps	maximal number of steps per output interval taken by the solver
printmescd	if TRUE the mixed error significant digits computed using the reference solution at time 1e13 are printed
...	additional arguments passed to the solver .

Details

The default parameters are: M1 = 0.36, M2 = 0.151104, M3 = 0.075552, L1 = 0.15, L2 = 0.30, J1 = 0.002727, J2 = 0.0045339259, EE = 0.20e12, NUE= 0.30, BB = 0.0080, HH = 0.0080, RHO= 7870.0, GRAV= 0.0, OMEGA = 150.0

There are two default initial conditions - set with options(ini=x)

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in yini, plus an additional column (the first) for the time value.

There will be one row for each element in times unless the solver returns with an unrecoverable error. If yini has a names attribute, it will be used to label the columns of the output value.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia <mazzia@dm.uniba.it>

References

https://archimede.dm.uniba.it/~testset/

[Hor76] E.H. Horneber. Analyse nichtlinearer RLCU-Netzwerke mit Hilfe der gemischten Potentialfunktion mit einer systematischen Darstellung der Analyse nichtlinearer dynamischer Netzwerke. PhD thesis, Universitat Kaiserslautern, 1976.

[KRS92] W. Kampowski, P. Rentrop, and W. Schmidt. Classiffcation and numerical simulation of electric circuits. Surveys on Mathematics for Industry, 2(1):23–65, 1992.

[MM08] F. Mazzia and C. Magherini. Test Set for Initial Value Problem Solvers, release 2.4. Department of Mathematics, University of Bari and INdAM, Research Unit of Bari, February 2008

Examples

```
out <- ring()
plot(out, col = "darkblue", lwd = 2)
mtext(side = 3, line = -1.5, "RING modulator",cex = 1.25, outer = TRUE)

# compare with reference solution
max(abs(out[nrow(out),-1]- reference("ring")))
```

---

rober *Autocatalytic Chemical Reaction of Robertson, ODE*

Description

Describes the kinetics of an autocatalytic reation.

It is an ODE, 3 equations

Usage

```
rober  (times = 10^(seq(-5, 11, by = 0.1)), yini = NULL,
        parms = list(), printmescd = TRUE,
        atol = 1e-14, rtol = 1e-10, maxsteps = 1e5, ...)
```

Arguments

yini	the initial (state) values for the DE system. If y has a name attribute, the names will be used to label the output matrix.
times	time sequence for which output is wanted; the first value of times must be the initial time.
parms	list of parameters that overrule the default parameter values
atol	absolute error tolerance, either a scalar or a vector, one value for each y.
rtol	relative error tolerance, either a scalar or a vector, one value for each y,
maxsteps	maximal number of steps per output interval taken by the solver
printmescd	if TRUE the mixed error significant digits computed using the reference solution at time 1e13 are printed
...	additional arguments passed to the solver .

Details

The default parameters are: k1 = 0.04, k2 = 3e7, k3 = 1e4

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in yini, plus an additional column (the first) for the time value.

There will be one row for each element in times unless the solver returns with an unrecoverable error. If yini has a names attribute, it will be used to label the columns of the output value.

Note

This model is implemented in R

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia

References

https://archimede.dm.uniba.it/~testset/

Examples

```
out <- rober()
plot(out, lwd = 2, log = "x")

# compare to reference solution
out[nrow(out),-1] - reference("rober")
```

`transistor` *The Transistor Amplifier, Index 1 DAE*

Description

Electrical circuit model for the transistor amplifier.

It is an index 1 DAE, 8 equations

Usage

```
transistor (times = seq(0, 0.2, 0.001), yini = NULL, dyini = NULL,
            parms = list(), printmescd = TRUE, method = mebdfi,
            atol = 1e-6, rtol = 1e-6, maxsteps = 1e5, ...)
```

Arguments

`yini`	the initial (state) values for the DE system. If `y` has a name attribute, the names will be used to label the output matrix.
`dyini`	the initial derivatives of the state variables of the DE system.
`times`	time sequence for which output is wanted; the first value of `times` must be the initial time.
`parms`	list of parameters that overrule the default parameter values
`method`	the solver to use
`maxsteps`	maximal number of steps per output interval taken by the solver
`atol`	absolute error tolerance, either a scalar or a vector, one value for each y.
`rtol`	relative error tolerance, either a scalar or a vector, one value for each y,
`printmescd`	if TRUE the mixed error significant digits computed using the reference solution at time 1e13 are printed
`...`	additional arguments passed to the solver .

Details

The default parameters are: ub=6, uf=0.026, alpha=0.99, beta=1e-6, r0=1000, r1=9000, r2=9000, r3=9000, r4 = 9000, r5=9000, r6=9000, r7=9000, r8 = 9000, r9 = 9000, c1=1e-6, c2=2e-6, c3=3e-6, c4=4e-6, c5=5e-6

Value

A matrix of class `deSolve` with up to as many rows as elements in `times` and as many columns as elements in `yini`, plus an additional column (the first) for the time value.

There will be one row for each element in `times` unless the solver returns with an unrecoverable error. If `yini` has a names attribute, it will be used to label the columns of the output value.

Note

This model is implemented in FORTRAN

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia

References

https://archimede.dm.uniba.it/~testset/

Examples

```
out <- transistor()
plot(out, lwd = 2)

out[nrow(out),-1]-reference("transistor")
```

---

tube *Water Tube System, Mechanics problem, DAE of Index 2*

Description

The tube problem describes the water flow through a tube system, taking into account turbulence and the roughness of the tube walls.

It is an index 2 system of 49 non-linear Differential-Algebraic Equations.

Usage

```
tube (times = seq(0, 17.0*3600, by = 100), yini = NULL, dyini = NULL,
      parms = list(), printmescd = TRUE, method = radau,
      atol = 1e-6, rtol = 1e-6, maxsteps = 1e+05, ...)
```

Arguments

yini	the initial (state) values for the DE system. If y has a name attribute, the names will be used to label the output matrix.
dyini	the initial derivatives of the state variables of the DE system.
times	time sequence for which output is wanted; the first value of times must be the initial time.
parms	list of parameters that overrule the default parameter values
method	the solver to use; only mebdfi available for now
maxsteps	maximal number of steps per output interval taken by the solver
atol	absolute error tolerance, either a scalar or a vector, one value for each y.

rtol	relative error tolerance, either a scalar or a vector, one value for each y,
printmescd	if TRUE the mixed error significant digits computed using the reference solution at time 1e13 are printed
...	additional arguments passed to the solver .

Details

parameter <- c(nu = 1.31e-6, g = 9.8, rho = 1.0e3, rcrit = 2.3e3, length= 1.0e3, k = 2.0e-4, d= 1.0e0, b = 2.0e2)

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in yini, plus an additional column (the first) for the time value.

There will be one row for each element in times unless the solver returns with an unrecoverable error. If yini has a names attribute, it will be used to label the columns of the output value.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia

References

https://archimede.dm.uniba.it/~testset/

Examples

```
out <- tube()
plot(out, lwd = 2, which = 1:9)
plot(out, which = "phi3.4", lwd = 2, xlim = c(10000, 60000),
 ylim = c(0.000145, 0.000185))

# compare with reference solution
max(abs(out[nrow(out),-1]- reference("tube")))
```

---

twobit *The Two Bit Adding Unit, Index 1 DAE*

Description

Computes the sum of two base-2 numbers, each two digits long, and a carry bit. These numbers are fed into the circuit in the form of input signals.

Index 1 DAE of dimension 350

Usage

```
twobit (times = seq(0, 320, by = 0.5), yini = NULL, dyini = NULL,
        printmescd = TRUE, method = radau,
        atol = 1e-4, rtol = 1e-4, maxsteps = 1e5, hmax = 0.1, ...)
```

Arguments

yini	the initial (state) values for the DE system. If y has a name attribute, the names will be used to label the output matrix.
dyini	the initial derivatives of the state variables of the DE system.
times	time sequence for which output is wanted; the first value of times must be the initial time.
method	the solver to use
atol	absolute error tolerance, either a scalar or a vector, one value for each y.
rtol	relative error tolerance, either a scalar or a vector, one value for each y,
maxsteps	maximal number of steps per output interval taken by the solver
hmax	maximal size of step; if too large: will fail.
printmescd	if TRUE the mixed error significant digits computed using the reference solution at time 1e13 are printed
...	additional arguments passed to the solver .

Details

This model has no parameters

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in yini, plus an additional column (the first) for the time value.

There will be one row for each element in times unless the solver returns with an unrecoverable error. If yini has a names attribute, it will be used to label the columns of the output value.

Note

This model is implemented in FORTRAN

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia

References

https://archimede.dm.uniba.it/~testset/

Examples

```
out <- twobit(times = seq(0, 100, by = 0.5))
plot(out, lwd = 2, which = c("x49", "x130", "x148"), mfrow = c(3, 1))

## Not run:
  out <- twobit()
# compare with reference solution
  max(abs(out[nrow(out),-1] - reference("twobit")))

## End(Not run)
```

---

vdpol *van der Pol Equation, Nonlinear Vacuum Tube Circuit, ODE*

Description

Problem originating from electronics, describing the behavior of nonlinear vacuum tube circuots.

It is an ODE, 2 equations.

Usage

```
vdpol  (times = 0:2000, yini = NULL,
        parms = list(), printmescd = TRUE,
        atol = 1e-6, rtol = 1e-6, ...)
```

Arguments

yini	the initial (state) values for the DE system. If y has a name attribute, the names will be used to label the output matrix.
times	time sequence for which output is wanted; the first value of times must be the initial time.
parms	list of parameters that overrule the default parameter values
atol	absolute error tolerance, either a scalar or a vector, one value for each y.
rtol	relative error tolerance, either a scalar or a vector, one value for each y,
printmescd	if TRUE the mixed error significant digits computed using the reference solution at time 5 are printed
...	additional arguments passed to the solver .

Details

The default parameters are: mu=1000

The default initial conditions are: y1 = 2, y2 = 0

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in yini, plus an additional column (the first) for the time value.

There will be one row for each element in times unless the solver returns with an unrecoverable error. If yini has a names attribute, it will be used to label the columns of the output value.

Note

This model is implemented in R

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia <mazzia@dm.uniba.it>

References

https://archimede.dm.uniba.it/~testset/

Examples

```
out <- vdpol()
plot(out, lwd = 2, which = 1)

# compare to reference solution
out[nrow(out),-1] - reference("vdpol")
```

---

wheelset *Wheel Set problem, mechanics, Index 2 IDE*

Description

Describes the motion of a simple wheelset on a rail track.

It is a differential algebraic equation of index 2, 17 equations.

Usage

```
wheelset (times = seq(0, 10, by = 0.01), yini = NULL, dyini = NULL,
          parms = list(), printmescd = TRUE, method = mebdfi,
          atol = 1e-6, rtol = 1e-6, maxsteps = 1e5, ...)
```

Arguments

yini	the initial (state) values for the DE system. If y has a name attribute, the names will be used to label the output matrix.
dyini	the initial derivatives of the state variables of the DE system.
times	time sequence for which output is wanted; the first value of times must be the initial time.
parms	list of parameters that overrule the default parameter values
method	the solver to use
maxsteps	maximal number of steps per output interval taken by the solver
atol	absolute error tolerance, either a scalar or a vector, one value for each y.
rtol	relative error tolerance, either a scalar or a vector, one value for each y,
printmescd	if TRUE the mixed error significant digits computed using the reference solution at time 0.1 are printed
...	additional arguments passed to the solver .

Details

The default parameters are: MR = 16.08, G = 9.81, V = 30., RN0 = 0.1, LI1 = 0.0605, LI2 = 0.366, MA = 0.0, HA = 0.2, MU = 0.12 , XL = 0.19, CX = 6400., CZ = 6400. , E = 1.3537956, GG = 0.7115218, SIGMA = 0.28, GM = 7.92e10, C11 = 4.72772197, C22 = 4.27526987, C23 = 1.97203505, DELTA0 = 0.0262, AR = 0.1506, RS = 0.06, EPS = 0.00001, B1 = 0.0, B2 = 4.0

Value

A matrix of class deSolve with up to as many rows as elements in times and as many columns as elements in yini, plus an additional column (the first) for the time value.

There will be one row for each element in times unless the solver returns with an unrecoverable error. If yini has a names attribute, it will be used to label the columns of the output value.

Note

This model is implemented in R.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Francesca Mazzia

References

https://archimede.dm.uniba.it/~testset/

Examples

```
out <- wheelset()
plot(out, which = 1:9, lwd = 2)
max(abs(out[nrow(out), -1] - reference("wheelset")))
```

Index