A User's Guide to the R library *ddesolve*
Version 0.50 – March, 2007

Alex Couture-Beil, Jon Schnute, and Rowan Haigh

1. Introduction

The R library *ddesolve* generates numerical solutions for systems of delay differential equations (DDEs) and ordinary differential equations (ODEs). The numerical routines come from Simon Wood's *solv95* program (http://www.maths.bath.ac.uk/~sw283/simon/dde.html), originally written in C for the Microsoft Windows operating systems. With *ddesolve*, a user can write the gradient code for a system of DDEs or ODEs in the R language, rather than C. The code will then run on all platforms supported by R, and the results can be inspected using R's extensive graphics capabilities.

For more information on Simon Wood and his work at the University of Bath, Bath, UK, see his home page http://www.maths.bath.ac.uk/~sw283/index.html. He has very generously given us permission to publish *ddesolve* (including his embedded routines) under the GNU GENERAL PUBLIC LICENSE Version 2

We have designed *ddesolve* to perform similarly to the earlier R library *odesolve*, written by R. Woodrow Setzer with Fortran algorithms by Linda R. Petzold and Alan C. Hindmarsh at the Lawrence Livermore National Laboratory in Livermore, California.

The demos included with *ddesolve* are designed to run with *PBSmodelling* version 1.16. It is possible to use the numerical routines without *PBSmodelling*; however, installing *PBSmodelling* will provide a greater experience and understanding of *ddesolve*.

2. Defining DDEs

To define a system of DDEs, a user supplied R function must be created to calculate the gradient of each variable in the system with respect to time. This gradient function must have one of the following two function definitions.

```
1. yprime <- function(t, y)
2. yprime <- function(t, y, parms)
```

Where `t` is the current time of integration; `y` is a vector of estimated values at time `t`; and `parms` is an optional argument used for passing extra constant parameters.

If the system of DDEs has multiple variables, then `y` will be a vector of n variables, and can be accessed from `y[1]` to `y[n]`.

The function must calculate the gradient for each variable in the system of DDEs and return the values in one of the following two return types.

1. A vector of gradients for each value of y
2. A list whose first element is a vector of gradients for each value of y, and whose second element is a numerical vector of any additional values required at each time step of integration.

Lagged values are accessed with calls to `pastvalue()` and `pastgradient()`. Both functions take a single argument, time, and may only be called for times greater or equal to the start time, and less than the current time of integration. Both functions return a vector of past y values. Lags should be accessed by calls similar to `pastvalue(t - constant)`. Passing a constant time value to either function (i.e. `pastvalue(constant)`) should **<u>never</u>** be done, since this will require a very large memory buffer, and defeats the purpose of a ring buffer.

3. Solving DDEs

Simon Wood's numerical routines create the core functionality of *ddesolve.* The function

```
dde(y, func, parms=NULL, from=0, to=10, by=0.01, tol=1e-8,
dt=0.1, hbsize=10000)
```

is used to invoke the C routines used to numerically solve systems of DDEs.

The return value of `dde()` is a data frame containing a time `"t"` column for every solved time point. A column for each variable of the system, named `"y1", "y2", ..., "yn"`. And if func() returned a list containing additional information, then additional columns named `"extra1", "extra2", ..., "extran"`.

If the initial values vector `y` was named, then the variable columns will use those names instead of the default `"y*"` names.

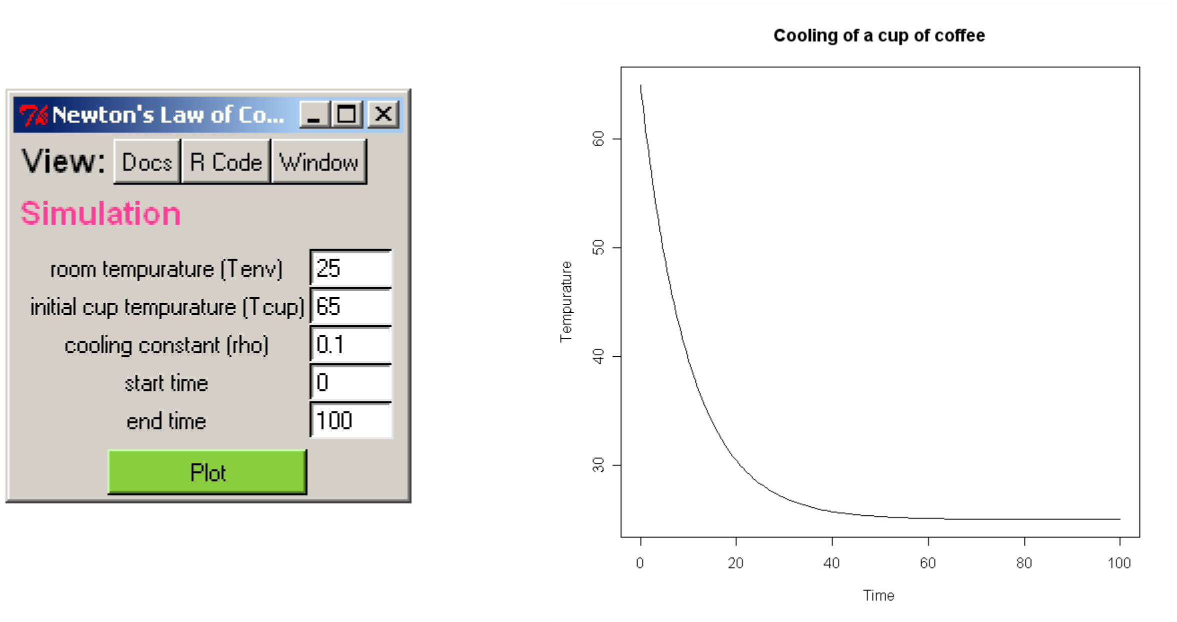
If `func()` returned additional information that included a names attribute, then those names will override the default `"extra*"` names.

4. Demos

Three demos are included with *ddesolve* to illustrate possible uses of ddesolve. These demos require the R library *PBSmodelling* which is utilized to create graphical user interfaces (GUIs) designed to aid the exploration of various models.

Once *PBSmodelling* is installed, run the `runDemos()` function and select *ddesolve* to access the included demos. Alternatively, R's native `demo()` function may be used in lieu of `runDemos()`.

4.1 Cooling - Newton's Law of Cooling (ODE Example)



room tempurature (Tenv) 25
initial cup tempurature (Tcup) 65
cooling constant (rho) 0.1
start time 0
end time 100

Figure 1. Newton's Law of Cooling demonstration.

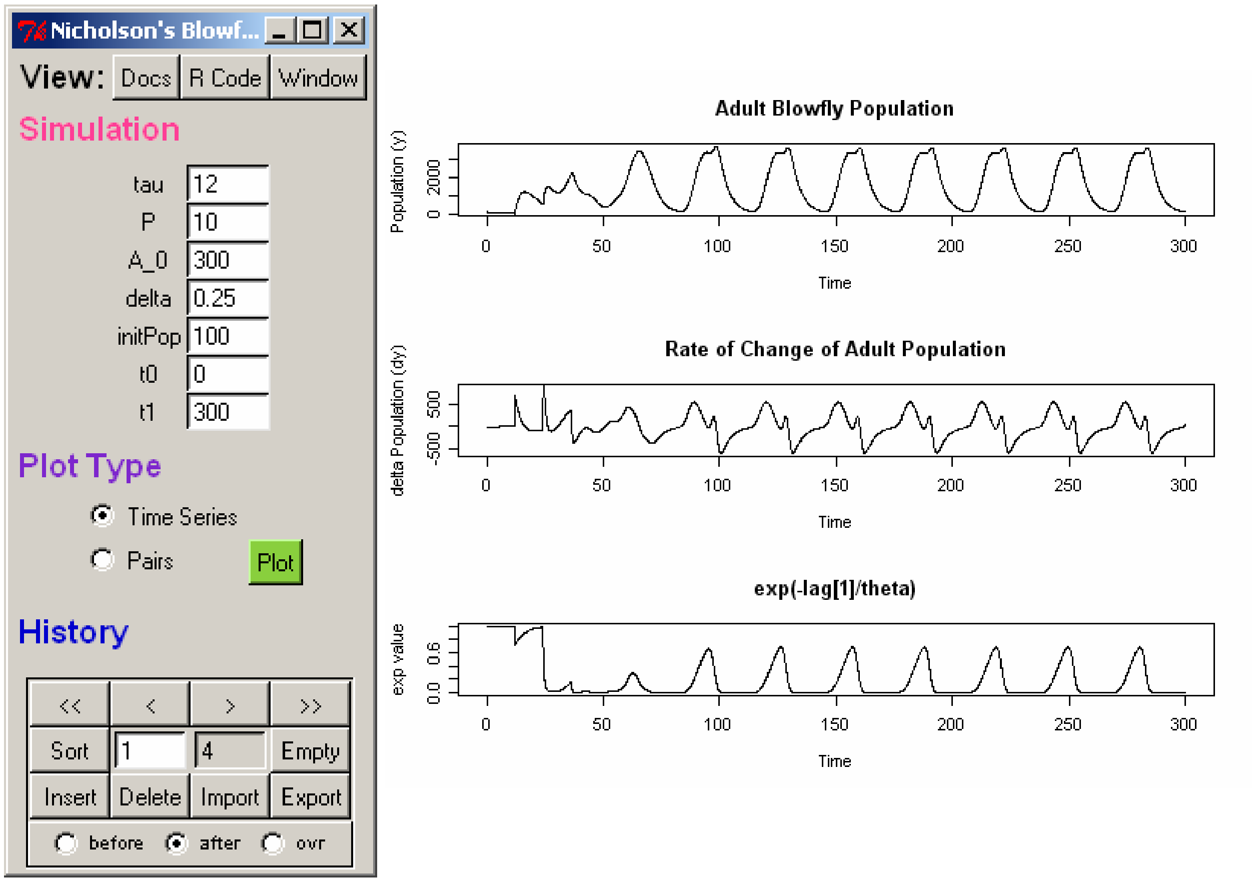
The cooling demo illustrates how to setup and solve ODEs with *ddesolve* by solving the rather easy ordinary differential equation

dy/dt = -rho * (y - Tenv)

which is known to have the analytical solution

y = Tenv + (Tcup - Tenv) * exp(-rho * t)

4.2 Blowflies – Gurney and Nisbet's (1981) Model of Nicholson's (1954) Blowflies (DDE Example)



tau	12
P	10
A_0	300
delta	0.25
initPop	100
t0	0
t1	300

Plot Type: [✓ Time Series] Pairs

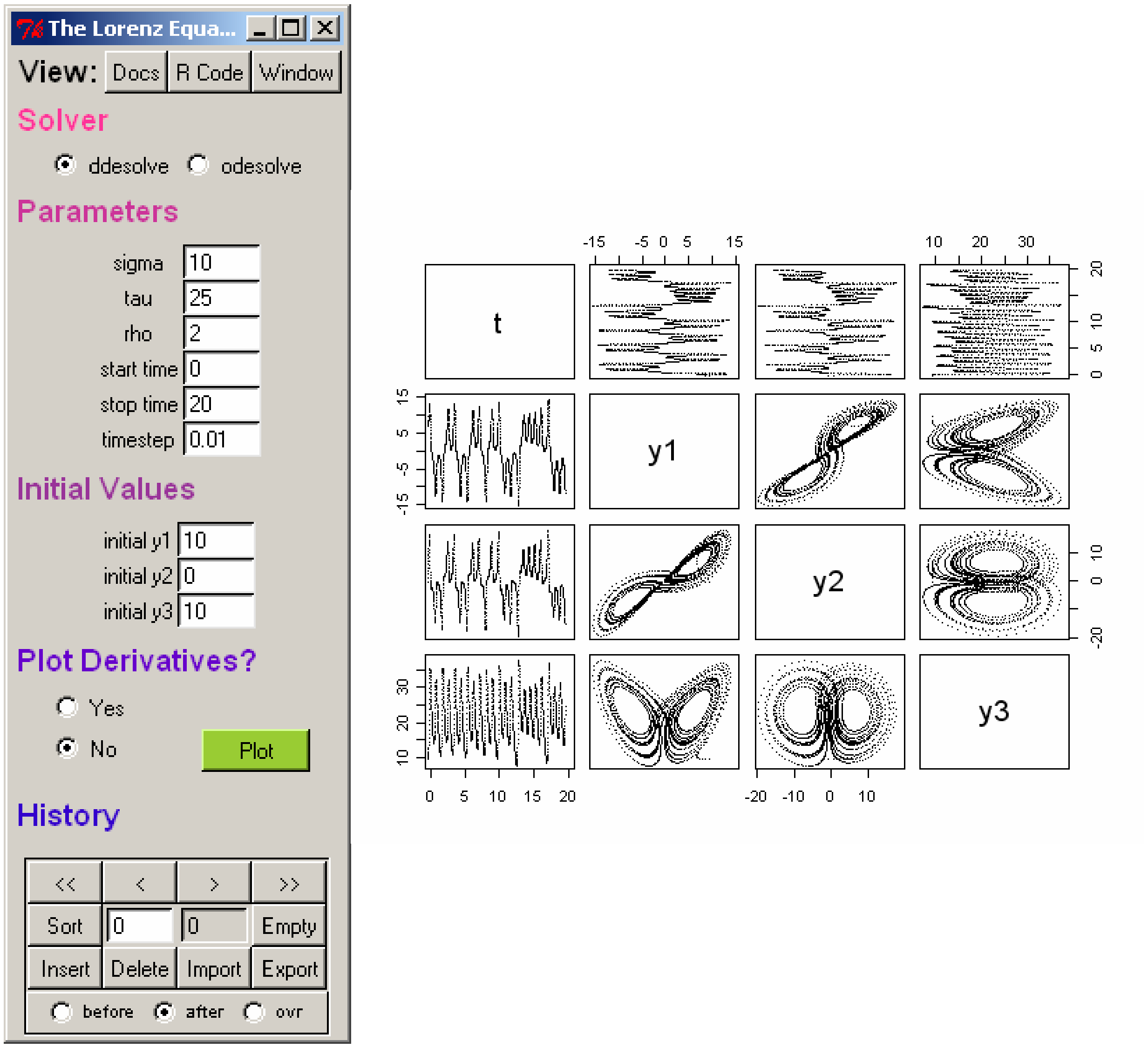
History: 1 4

before [✓ after] ovr

Figure 2. Nicholson's Blowflies Model Demonstration

This demonstration was included in Simon Wood's Solv95 User Manual as an example of solving a DDE.

4.3 Lorenz – The Lorenz Equation (ODE Example)


The Lorenz Equa...

View: Docs | R Code | Window

Solver

[✓ ddesolve] odesolve

Parameters

sigma	10
tau	25
rho	2
start time	0
stop time	20
timestep	0.01

Initial Values

initial y1	10
initial y2	0
initial y3	10

Plot Derivatives?

Yes [✓ No]

History

0 0

before [✓ after] ovr

Figure 3. Lorenz Equation Demonstration

The Lorenz Equation demonstrates chaotic behaviour in differential equations. This demonstration includes the ability to evaluate the system of ODEs in either *ddesolve*, or *odesolve*. Selecting either numerical solver should not affect the results of the plot. Both packages are capable of solving ODEs and have completely different underlying code. This demonstration illustrates that both solvers return comparable results, and is used as a sanity check to show *ddesolve* is working correctly.

5. The Algorithm

The R library *ddesolve* provides an interface to Simon Wood's numerical routines found in solv95. The algorithm used is *ddesolve* is the same as the one used by solv95.

> The method used for integration is an embedded RK2(3) scheme due to Fehlberg, and reported on page 170 of Hairer *et al.* (1987). Lagged variables (and gradients) are stored in a ring buffer at each step of the integrator. Interpolation is required to estimate values of the lagged variables between storage times. For numerical probity it is essential that the interpolation of lagged variables is of a higher order of approximation than the integrator, otherwise the assumptions underlying the error estimate from the RK pair will not be met. The algorithm used in Solv95 uses cubic hermite interpolation (e.g. Burden and Faires 1987) to achieve this (which is the reason that gradients need to be stored along with lagged values). The consequences of not using consistent interpolation and integration schemes are vividly illustrated in Highman (1993). Paul (1992) was also influential in the design of the method used here, and the step size selection is straight out of Press *et al.* (1992) (method, not code!). The RK2(3) pair used is not actually optimal - it should be possible to derive an improved scheme - see Butcher (1987) for an explanation of how to go about it.[1]

The solv95 software requires users to define DDEs as a model in C code. Users then have to compile solv95 on their own to solve the DDE. If the DDE gradient function ever changed, solv95 would have to be recompiled. Rather than taking this approach, *ddesolve* provides a generic C model to the numerical routines of solv95, which acts as a glue between the numerical C routines and R. This eliminates the need for the compiling process and by returning the results to R, the user is able to interpret the results using any of R's rich libraries and internal functions.

The numerical routines have been preserved in the files `ddeq.c` and `ddeq.h`. The interface to `dde()` has been significantly altered and is now in the file `ddesolve95.c`, which were originally found in solv95.c. The generic C model, mentioned above, started from a basic model template from solve95, but now contains many R API calls.

6. References

Burden, R.L. and J.D. Faires (1985) Numerical Analysis. Pridle Weber and Schmidt, Boston.
Butcher, J.C. (1987) The Numerical Analysis of Ordinary Differential
Hairer, E., S.P.Norsett & G.Wanner (1987) Solving Ordinary differential Equations I. Springer-Verlag Berlin. p170 RKF2(3)B
Highman, D.J. (1993) Appl. Numer. Math. 12:403-414
Paul, C.A.H (1992) Appl. Numer. Math. 9:403-414
Press *et al.* (1992) Numerical Recipes in C. CUP

[1] From Simon Wood's User Manual

Schnute, J.T., Couture-Beil, A., and Haigh, R. 2006. PBS Modelling 1: User's Guide. Can. Tech. Rep. Fish. Aquat. Sci. 2674: viii + 112 p.

Shampine, L.F. Solving Delay Differential Equations with dde23
URL: http://www.radford.edu/~thompson/webddes/tutorial.html

Wood, S N. Solv95: a numerical solver for systems of delay differential equations with switches
URL: http://www.maths.bath.ac.uk/~sw283/simon/dde.html