Package bvpSolve, solving boundary value problems in R

Karline Soetaert
Centre for Estuarine and Marine Ecology
Netherlands Institute of Ecology
The Netherlands

Abstract

This document is about package **bvpSolve** (Soetaert 2009a), designed for the numerical solution of boundary value problems for (first-order) ordinary differential equations (ODE) in R .

Package **bvpSolve** contains:

- function `bvpshoot` which implements the shooting method. This method makes use of the initial value problem solvers from packages **deSolve** (Soetaert, Petzoldt, and Setzer 2009) and the root-finding solver from package **rootSolve** (Soetaert 2009b).
- function `bvptwp`, the mono-implicit Runge-Kutta (MIRK) method with deferred corrections, code TWPBVP (Cash and Wright 1991), for solving two-point boundary value problems

The R functions have an interface which is similar to the interface of the solvers in package **deSolve**

Keywords: ordinary differential equations, boundary value problems, shooting method, mono-implicit Runge-Kutta, R.

1. Introduction

bvpSolve numerically solves boundary value problems (BVP) of first-order ordinary differential equations (ODE), which for one ODE can be written as:

$$\begin{aligned}\frac{dy}{dx} &= f(x,y)\\ a \leq x &\leq b\\ g_1(y)|_a &= 0\\ g_2(y)|_b &= 0\end{aligned}$$

where `y` is the dependent, `x` the independent variable, function `f` is the differential equation, g_a and g_b the boundary conditions at the end points a and b.

The problem must be specified as a first-order system. Thus, higher-order ODEs need to be rewritten as a set of first-order systems. For instance:

$$\frac{d^2y}{dx^2} = f(x, y, \frac{dy}{dx})$$

can be rewritten as:

$$\frac{dy}{dx} = z$$
$$\frac{dz}{dx} = f(x, y, z)$$

Note that in the current implementation, the boundary conditions must be defined at the end of the interval over which the ODE is specified (i.e. at `a` and/or `b`).

More examples of boundary value problems can be found in the packages `examples` subdirector. They include a.o. all problems found in `http://www.ma.ic.ac.uk/~jcash/BVP_software`.

2. A simple BVP example

Here is a simple ODE (which is problem 7 from a test problem available from http://www.ma.ic.ac.uk/~jcash/BVP_software/readme.php):

$$\begin{aligned}
\xi y'' + xy' - y &= -(1 + \xi\pi^2)\cos(\pi x) - \pi x \sin(\pi x) \\
y(-1) &= -1 \\
y(1) &= 1
\end{aligned}$$

The second-order ODE is expanded as two first-order ODEs as:

$$\begin{aligned}
y_1' &= y_2 \\
y_2' &= 1/\xi \cdot (-xy_2 + y_1 - (1 + \xi\pi^2)\cos(\pi x) - \pi x \sin(\pi x))
\end{aligned}$$

with boundary conditions

$$\begin{aligned}
y_1(-1) &= -1 \\
y_1(1) &= 1
\end{aligned}$$

This is implemented as:

```
> fun<- function(x,y,pars)
+ {
+  list(c(y[2],
+     1/ks*(-x*y[2]+y[1]-(1+ks*pi*pi)*cos(pi*x)-pi*x*sin(pi*x)))
+        )
+ }
```

and solved, using the two methods, as:

```
> ks <- 0.1
> x  <- seq(-1,1,by=0.01)
> print(system.time(
+ sol1  <- bvpshoot(yini=c(-1,NA),yend=c(1,NA),x=x,func=fun,guess=0)
+ ))

   user  system elapsed
   0.05    0.00    0.05

> print(system.time(
+ sol2  <- bvptwp(yini=c(-1,NA),yend=c(1,NA),x=x,func=fun, guess=0)
+ ))

   user  system elapsed
   0.15    0.00    0.14
```

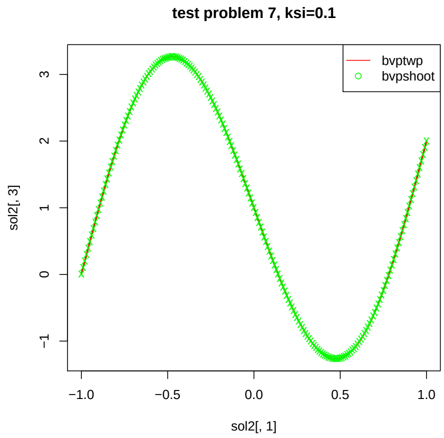

Figure 1: Solution of the simple BVP, for ksi=0.1 - see text for R -code

Note how the boundary conditions at the start (`yini`) and end `yend` of the integration interval are specified, where `NA` is used for boundary conditions that are not known.

A reasonable `guess` of the unknown initial condition is also inputted.

As is often the case, the shooting method is faster than the other method. However, there are particular problems where `bvpshoot` does not lead to a solution, whereas the MIRK method does (see below).

The plot shows that the two methods give the same solution:

```
> plot(sol2[,1],sol2[,3],type="l",main="test problem 7, ksi=0.1",
+      lwd=2,col="red")
> points(sol1[,1],sol1[,3],col="green",pch="x")
> legend("topright",c("bvptwp", "bvpshoot"),
+         lty=c(1,NA,NA), pch=c(NA,1,3),col=c("red", "green") )
```

When the parameter ξ is decreased, `bvpshoot` cannot solve the problem anymore, due to the presence of a zone of rapid change near x=0.

However, it can still easily be solved with the MIRK method:

```
> ks <-0.0001
> print(system.time(
+ sol2  <- bvptwp(yini=c(-1,NA),yend=c(1,NA),x=seq(-1,1,by=0.01),
+            func=fun, guess=0)
+ ))

   user  system elapsed 
    1.8     0.0     1.8 

> plot(sol2[,1],sol2[,3],type="l",main="test problem 7, ksi=0.0001",
+      lwd=2,col="red")
```

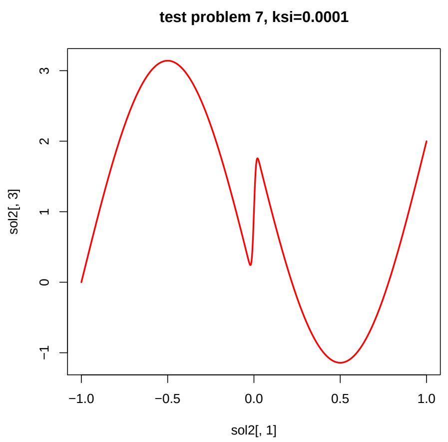

Figure 2: Solution of the simple BVP, for ksi=0.0001 - see text for R -code. Note that this problem cannot be solved with `bvpshoot`

3. A more complex BVP example

Here the test problem referred to as "swirling flow III" is solved (Ascher, Mattheij, and Russell 1995).

The original problem definition is:

$$
\begin{aligned}
g'' &= (gf' - fg')/\xi \\
f'''' &= (-ff''' - gg')/\xi
\end{aligned}
$$

on the interval [0,1] and subject to boundary conditions:

$$
\begin{aligned}
g(0) = -1, f(0) = 0, f'(0) = 0 \\
g(1) = 1, f(1) = 0, f'(1) = 0
\end{aligned}
$$

This is rewritten as a set of 1st order ODEs as follows:

$$
\begin{aligned}
y_1' &= y_2 \\
y_2' &= (y_1 * y_4 - y_3 * y_2)/\xi \\
y_3' &= y_4 \\
y_4' &= y_5 \\
y_5' &= y_6 \\
y_6' &= (-y_3 y_6 - y_1 y_2)/\xi
\end{aligned}
$$

Its implementation in R is:

```
> fsub <- function (t,Y,pars)
+ { return(list(c(f1 = Y[2],
+                  f2 = (Y[1]*Y[4] - Y[3]*Y[2])/eps,
+                        f3 = Y[4],
+                        f4 = Y[5],
+                        f5 = Y[6],
+                        f6 = (-Y[3]*Y[6] - Y[1]*Y[2])/eps)))
+ }
> eps <- 0.001
> x <- seq(0,1,len=100)
```

This model cannot be solved with the shooting method. However, it can be solved using `bvptwp`:

```
> print(system.time(
+ Soltwp <- bvptwp(x=x,func=fsub,guess= c(2,0,0),
+                    yini=c(-1,NA,0,0,NA,NA), yend=c(1,NA,0,0,NA,NA))
+ ))

   user  system elapsed
   1.80    0.00    1.79
```

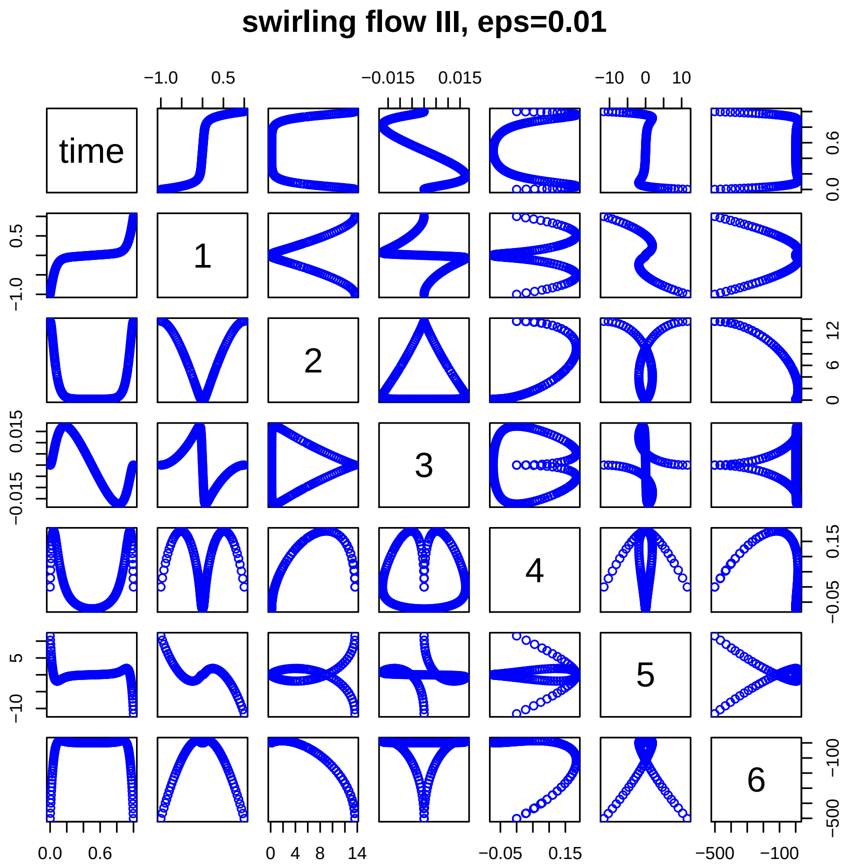

Figure 3: Solution of the swirling flow III problem - see text for R -code. Note that this problem cannot be solved with `bvpshoot`

where the reported system time is in seconds

The problem cannot be solved with too small values of eps:

```
> eps <- 1e-9
> Soltwp2 <- NA
> Soltwp2 <- try(bvptwp(x=x,func=fsub,guess= c(2,0,0),
+                     yini=c(-1,NA,0,0,NA,NA), yend=c(1,NA,0,0,NA,NA)),
+     silent = TRUE)
> cat(Soltwp2)

Error in bvptwp(x = x, func = fsub, guess = c(2, 0, 0), yini = c(-1, NA,  :
  The Expected No. Of Subintervals Exceeds Storage Specifications.

> pairs(Soltwp, main="swirling flow III, eps=0.01", col="blue")
```

The problem is more efficiently solved if an initial guess of the solution is given:

```
> eps <- 0.001
> xguess <- x
> yguess <- matrix(nr=6, nc=length(xguess), data=0.)
> slope  <- 0.375-0.9129
> yguess[1,] <- 2*xguess -1
> yguess[2,] <- 2
> print(system.time(Sol2 <- bvptwp(x=x,func=fsub,guess= c(2,0,0),
+   xguess=xguess,yguess=yguess,yini=c(-1,NA,0,0,NA,NA),
+   yend=c(1,NA,0,0,NA,NA))))

   user  system elapsed 
   1.41    0.00    1.40 
```

4. More complex initial or end conditions

Problem musn was described in (Ascher *et al.* 1995).

The problem is:

$$\begin{aligned}
u' &= 0.5u(w-u)/v \\
v' &= -0.5(w-u) \\
w' &= (0.9-1000(w-y)-0.5w(w-u))/z \\
z' &= 0.5(w-u) \\
y' &= -100(y-w)
\end{aligned}$$

on the interval [0,1] and subject to boundary conditions:

$$\begin{aligned}
u(0)=v(0)=w(0) &= 1 \\
z(0) &= -10 \\
w(1) &= y(1)
\end{aligned}$$

Note the last boundary conditions which expresses w as a function of y.

Implementation of the ODE function is simple:

```
> musn <- function(x,Y,pars)
+ {
+   with (as.list(Y),
+   {
+    du=0.5*u*(w-u)/v
+    dv=-0.5*(w-u)
+    dw=(0.9-1000*(w-y)-0.5*w*(w-u))/z
+    dz=0.5*(w-u)
+    dy=-100*(y-w)
+    return(list(c(du,dv,dw,dz,dy)))
+   })
+ }
```

There are 4 boundary values specified at the start of the interval; a value for y is lacking:

```
> init <- c(u=1,v=1,w=1,z=-10,y=NA)
```

The boundary condition at the end of the integration interval (1) specifies the value of w as a function of y.

Because of that, yend cannot be simply inputted as a vector. It is rather implemented as a function that has as input the values at the end of the integration interval (Y), the values at the start (yini) and the parameters, and that returns the residual function (w-y):

```
> yend  <- function (Y,yini,pars)  with (as.list(Y), w-y)
```

This problem is most efficiently solved with bvpshoot: [1]

[1] Note that there are at least two solutions to this problem, the second solution can simply be found by setting guess equal to 0.9.

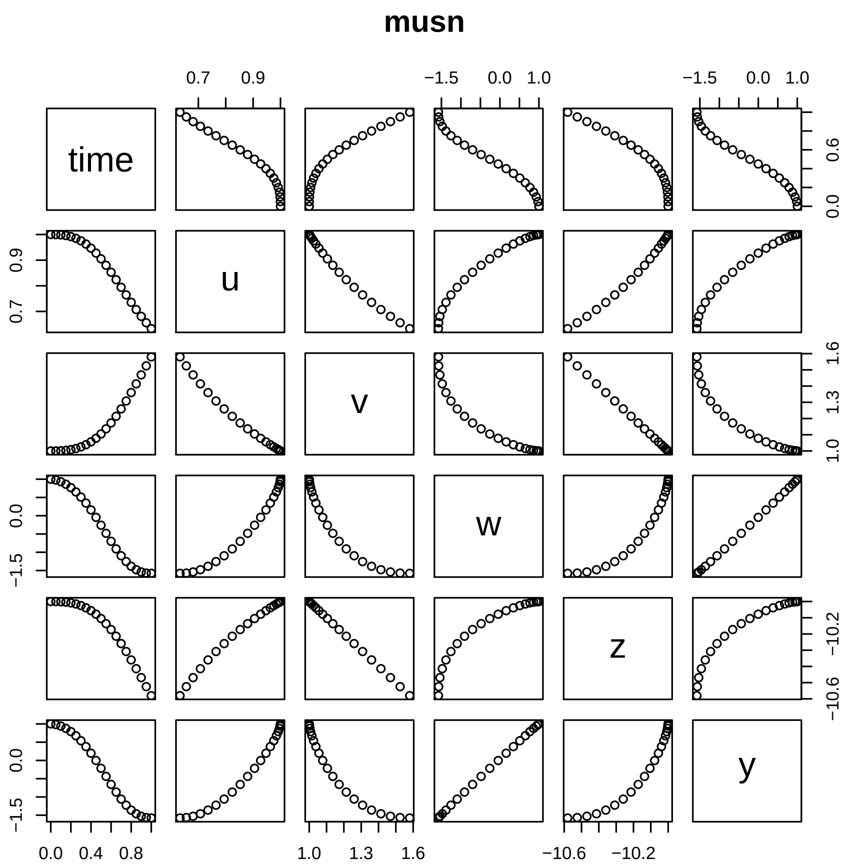

Figure 4: Solution of the musn model, using `bvpshoot` - see text for R -code.

```
> print(system.time(
+ sol   <-bvpshoot(yini= init, x=seq(0,1,by=0.05),func=musn,
+            yend=yend,guess=1,atol=1e-10,rtol=0)
+ ))

   user  system elapsed
   0.34    0.00    0.35

> pairs(sol,main="musn")
```

5. a BVP problem including an unknown parameter

In the next BVP problem, a parameter λ is to be found such that:

$$\frac{d^2y}{dt^2} + (\lambda - 10\cos(2t)) \cdot y = 0$$

on $[0,\pi]$ with boundary conditions $\frac{dy}{dt}(0) = 0$ and $\frac{dy}{dt}(\pi) = 0$ and $y(0) = 1$
Here all the initial values (at t=0) are prescribed. If λ would be known the problem would be overdetermined.
The 2^{nd} order differential equation is first rewritten as two 1^{st}-order equations:

$$\begin{aligned} \frac{dy}{dt} &= y2 \\ \frac{dy2}{dt} &= -(\lambda - 10\cos(2t)) \cdot y \end{aligned}$$

and the function that estimates these derivatives is written (`derivs`).

```
> mathieu <- function(x,y,lambda)
+     list(c(y[2],
+         -(lambda-10*cos(2*x))*y[1]))
```

which is easily solved using `bvpshoot`:

```
> init <- c(1,0)
> sol  <- bvpshoot(yini=init,yend=c(NA,0),x=seq(0,pi,by=0.01),
+          func=mathieu, guess=NULL, extra=15)
```

and plotted:

```
> plot(sol)
```

Note how the extra parameter to be fitted is passed (`extra`). The value of `lam` can be printed:

```
> attr(sol,"roots")  # root gives the value of "lam" (17.10684)

      root       f.root iter
2 17.10683 2.347269e-12    6
```

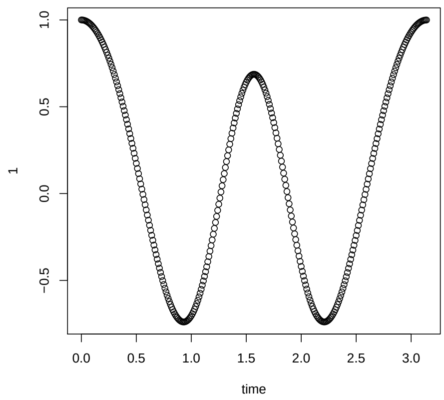

Figure 5: Solution of the BVP ODE problem including an unknown parameter, see text for R-code

References

Ascher U, Mattheij R, Russell R (1995). *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. Philadelphia, PA.

Cash J, Wright M (1991). "A deferred correction method for nonlinear two-point boundary value problems: implementation and numerical evaluation." *SIAM J. Sci. Stat. Comput.*, **12**, 971–989.

Soetaert K (2009a). *bvpSolve: General solvers for boundary value problems of ordinary differential equations*. R package version 1.0.

Soetaert K (2009b). *rootSolve: Nonlinear root finding, equilibrium and steady-state analysis of ordinary differential equations*. R package version 1.4.

Soetaert K, Petzoldt T, Setzer RW (2009). *deSolve: General solvers for initial value problems of ordinary differential equations (ODE), partial differential equations (PDE) and differential algebraic equations (DAE)*. R package version 1.3.

Affiliation:

Karline Soetaert
Centre for Estuarine and Marine Ecology (CEME)
Netherlands Institute of Ecology (NIOO)
4401 NT Yerseke, Netherlands E-mail: `k.soetaert@nioo.knaw.nl`
URL: `http://www.nioo.knaw.nl/users/ksoetaert`