Events in Dynamic Models developed in R

Reader Accompanying the Course Reaction Transport Modelling in the Hydrosphere

Karline Soetaert and Lubos Polerecky, Utrecht University

April 2021

Abstract

In dynamic models, an event corresponds to a situation when the value of a state variable is *suddenly* changed. Here we show how events are implemented in dynamic models developed in R. We illustrate this using the Crops and Weed model, where we implement soil fertilisation as discrete events occuring in weekly intervals rather than continuously.

Events

In dynamic models, an event corresponds to a situation when the value of a state variable is *suddenly* changed (e.g., because a value is added, subtracted, or multiplied). Integration routines such as `ode` cannot deal easily with such state variable changes, as there is no possibility to change the values of state variables directly within the model function. One way to do this is to run the simulation until the event, then change the state variables, and restart the simulation again until the next event. This approach would be, however, rather cumbersome.

In deSolve, events can be imposed without stopping the simulation by means of an input `data.frame` that specifies how and when a certain state variable is altered. We illustrate this using the Crops and Weed model developed in class. Note that there are also more complex and more flexible ways to implement events, but their explanation would go beyond the scope of this Reader.

First, we reproduce the Crops and Weed model implementation. The default model has continuous fertilisation.

```
require(deSolve)  # package with solution methods

# state variables, units = molP/m2
state <- c(WEED=0.001, CROP=0.005, P1=0.1, P2=0.1, Plost=0)

# parameters
parms <- c(
  Paddition    = 0.9/90,  # [molP/m2/d] Rate of P supply
  rPercolation = 0.05,    # [/d]        Rain rate dilution parameter
  ksCrop       = 2e-3,    # [molP/m2]   Monod coefficient for P uptake
  ksWeed       = 0.5e-3,  # [molP/m2]   Monod coefficient
  ktot         = 0.300,   # [molP/m2]   Carrying capacity (space limitation)
  rGcrop       = 0.125,   # [/d]        Growth rate,
  rGweed       = 0.1,
  rMcrop       = 0.0,     # [/d]        Loss rate (e.g. mortality),
  rMweed       = 0.0,
  N            = 25,      # [ind/m2]    Density of crop plants
  P2WW         = 62000    # [gWW/molP]  31/0.25/0.002
)
```

```
# Model function
Crops <- function(t, state, params) {
  with (as.list(c(state, params)), {

    # variables needed for rate expressions
    TotBiom    <- WEED + CROP       # total plant biomass
    NutWeed    <- P1 + P2           # nutrients accessible to weed
    NutCrop    <- P1                # nutrients accessible to crops
    partP1     <- P1/NutWeed        # part of P for weed from first layer

    # Rate expressions - all rates in molP/m2/day
    WeedGrowth <- rGweed * WEED*(1-TotBiom/ktot) *NutWeed/(NutWeed+ksWeed)
    CropGrowth <- rGcrop * CROP*(1-TotBiom/ktot) *NutCrop/(NutCrop+ksCrop)

    WeedLoss   <- rMweed * WEED     # death and other loss terms
    CropLoss   <- rMcrop * CROP

    Percolate  <- P1 * rPercolation  # transfer of P from layer 1 to layer 2
    Ploss      <- P2 * rPercolation  # transfer of P from layer 2 to deep soil

    # Mass balances [mmolP/m2/day]
    dWEED.dt      <- WeedGrowth - WeedLoss
    dCROP.dt      <- CropGrowth - CropLoss
    dP1.dt        <- Paddition - Percolate - CropGrowth - WeedGrowth * partP1
    dP2.dt        <- Percolate - Ploss - WeedGrowth * (1 - partP1)
    dPlost.dt     <- Ploss + WeedLoss + CropLoss

    return(list(c(dWEED.dt, dCROP.dt, dP1.dt, dP2.dt, dPlost.dt),
      TotBiom = TotBiom,
      Weight = CROP/N*P2WW,          # ind. plant wet weight, gram per plant
      TotP = WEED + CROP + P1 + P2 + Plost,       # total P, mass balance check
      SpaceLimFact = (1-TotBiom/ktot),            # "space-limitation" factor
      NutLimFactCrop = NutCrop/(NutCrop+ksCrop),  # nutrient limitation factor
      NutLimFactWeed = NutWeed/(NutWeed+ksWeed))  # for both crop and weed
      )
    })
}
```

Using events

The `data.frame` that specifies the fertilisation events has four columns:

1. the name of the state variable to be changed;
2. the time at which to change it;
3. the value describing the change;
4. the type of change.

In this example, we *add* a fixed amount of P to the state variable `P1` every 7 days, starting from day 7 (i.e., 12 additions). The added amount is chosen so that after 90 days the same total amount of P will be added as during the continuous fertilization (0.9 over 90 days, so 0.9/12 per addition). The corresponding `data.frame` for these events is defined as follows:

```
ferilisation.events <- data.frame(var = "P1",
                                  time  = seq(7, to = 90, by = 7),
                                  value = 0.9/12,
                                  method = "add")
knitr::kable(ferilisation.events)
```

var	time	value	method
P1	7	0.075	add
P1	14	0.075	add
P1	21	0.075	add
P1	28	0.075	add
P1	35	0.075	add
P1	42	0.075	add
P1	49	0.075	add
P1	56	0.075	add
P1	63	0.075	add
P1	70	0.075	add
P1	77	0.075	add
P1	84	0.075	add

Comparison of continuous and weekly fertilisation

First, we run the model with continuous fertilisation using model parameters defined above.

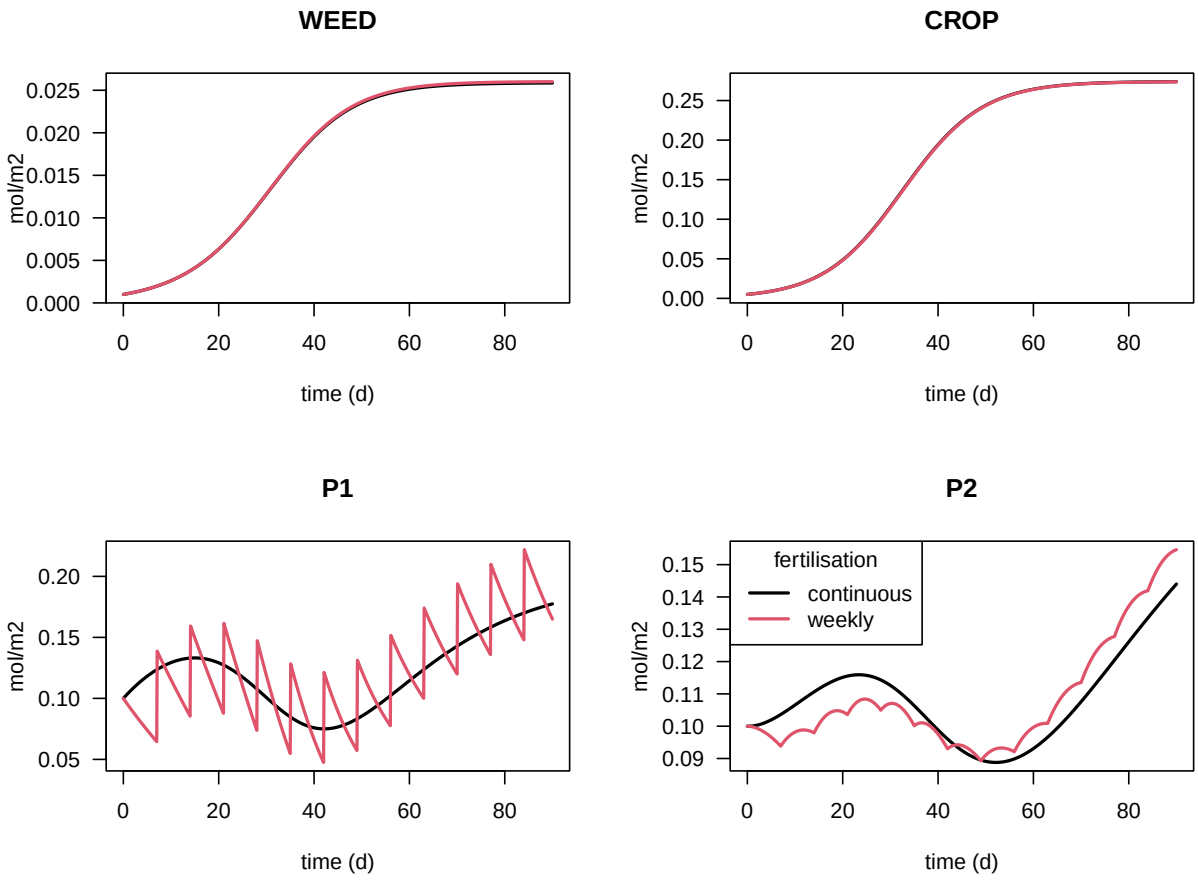
```
outtimes <- seq(from=0, to=90, by=0.1)  # run for 3 months
out <- ode(y=state, parms=parms, func=Crops, times=outtimes)
```

Subsequently, we run the model with the events. First, we set the parameter describing the rate of continuous P-addition to 0. Then, we pass the event information to the `ode` solver as the input argument `events`.

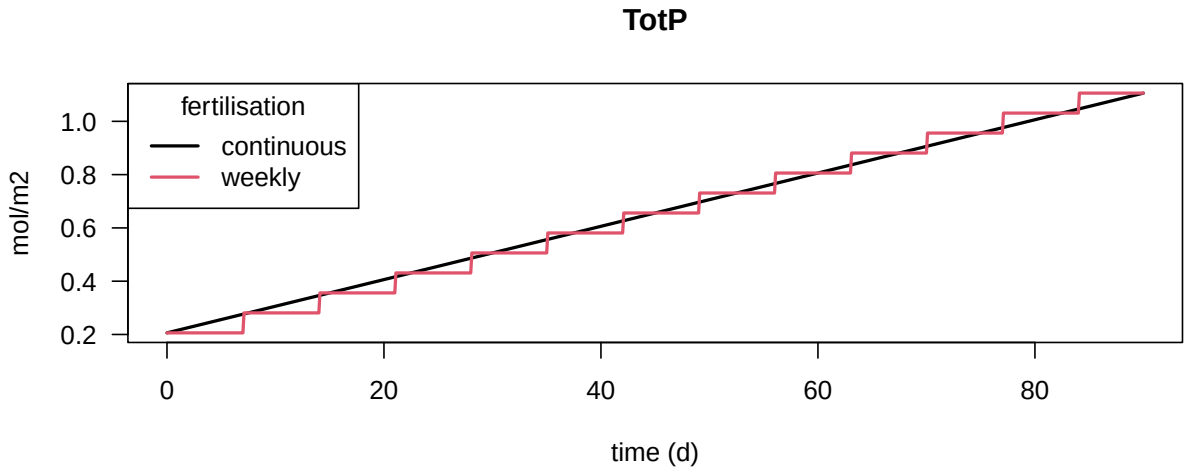
```
p2 <- parms
p2["Paddition"] <- 0
out2 <- ode(y=state, parms=p2, func=Crops, times=outtimes,
            events=list(data=ferilisation.events)) # include events!
```

A comparison of both runs shows little differences in the crop and weed biomass, and fluctuating P contents in the top and bottom soil layers. The last graph shows a linearly increasing amount of total P in the system during continuous fertilisation, in contrast to a step-wise increase during weekly fertilisation, as expected.

```
plot(out, out2, which=c("WEED","CROP","P1","P2"),
     mfrow=c(2,2), lty=1, lwd=2, las=1, ylab="mol/m2", xlab="time (d)")
legend("topleft", c("continuous",  "weekly"),
       title="fertilisation", col=1:2, lty=1, lwd=2)
```




```
plot(out, out2, which="TotP", lty=1, lwd=2, las=1, ylab="mol/m2", xlab="time (d)")
legend("topleft", c("continuous",  "weekly"),
       title="fertilisation", col=1:2, lty=1, lwd=2)
```



You can learn more about how to impose events by reading the help file that is part of the deSolve package:

```
?events
```

References

R Core Team (2020). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Karline Soetaert, Thomas Petzoldt, R. Woodrow Setzer (2010). Solving Differential Equations in R: Package deSolve. Journal of Statistical Software, 33(9), 1–25. URL http://www.jstatsoft.org/v33/i09/ DOI 10.18637/jss.v033.i09