Visualisation of a Reaction Transport Model Output together with Observed Data in R

Reader Accompanying the Course Reaction Transport Modelling in the Hydrosphere

Karline Soetaert and Lubos Polerecky, Utrecht University

March 2021

Abstract

Here we show how to include observed data in the same graph as the output of a reaction-transport model. This can be used to estimate values of model parameters that best fit the observed data.

```
require(deSolve)
require(marelac)
require(ReacTran)
require(rootSolve)
```

Data visualisation with model output

It is relatively simple to visualise observed data together with model output, provided that:

- The names of data columns are the same as the names of the (state) variables in the model.
- The first column contains *time* for dynamic models, or the *space* variable for 1D steady-state models.

To plot the data together with the model output, all you need to do is to specify the name of the data object (matrix or data.frame) in the argument `obs` when calling the `plot` function. You can also set the characteristics of the observed data through the argument `obspar`.

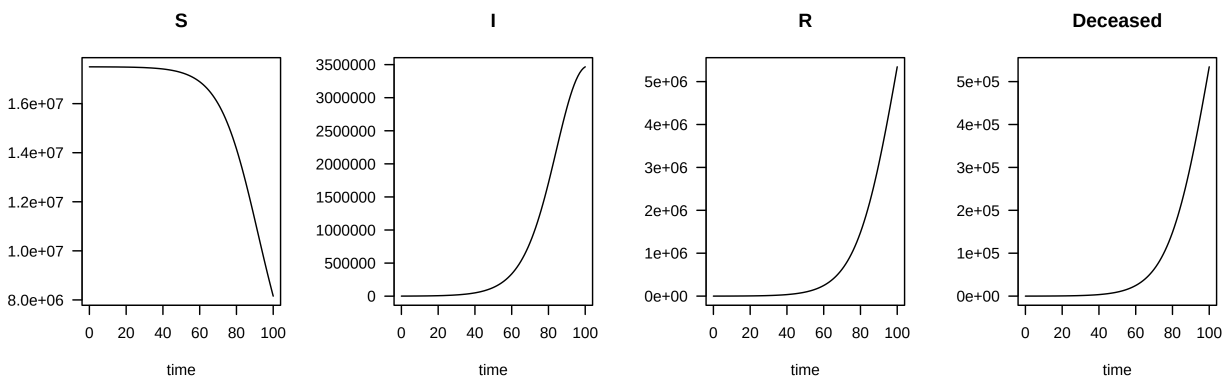
Dynamic model

Consider the simple SIR model that you made in the class:

```
state.SIR   <- c(S = 17500000, I = 1000, R = 0, Deceased = 0)

parms.SIR   <- c(
  b = 0.00000001,     # [1/ind/d], infection parameter
  g = 0.07,           # [1/d],     recovery rate of infected individuals
  m = 0.007           # [1/d],     mortality rate of infected individuals
)

SIR <- function(t, state, parameters) {
  with (as.list(c(state, parameters)), {

    Infection  <- b*S*I
    Recovery   <- g*I
    Mortality  <- m*I
    dS          <- -Infection
    dI          <-  Infection - Recovery - Mortality
    dR          <-  Recovery
    dDeceased   <-  Mortality    # to track number of deceased people

    return (list(c(dS, dI, dR, dDeceased)))
  })
}
```

The output of the model looks as shown in the following figure:

```
time.seq   <- seq(from=0, to=100, by=1)    # time sequence, in days
out <- ode(y=state.SIR, times=time.seq, func=SIR, parms=parms.SIR)
plot(out, las=1, col=1:2, lty=1, mfrow=c(1,4))
```



Compare to data

Suppose we have the following records of people that died (cumulatively).

```
SIRdata <- data.frame(
     time = seq(from = 0, to = 100, by = 10),
     Deceased = c(0, 50, 150, 250, 350, 450, 600, 700, 850, 1050, 1250))
head(SIRdata, n=2)
```

```
##   time Deceased
## 1    0        0
## 2   10       50
```

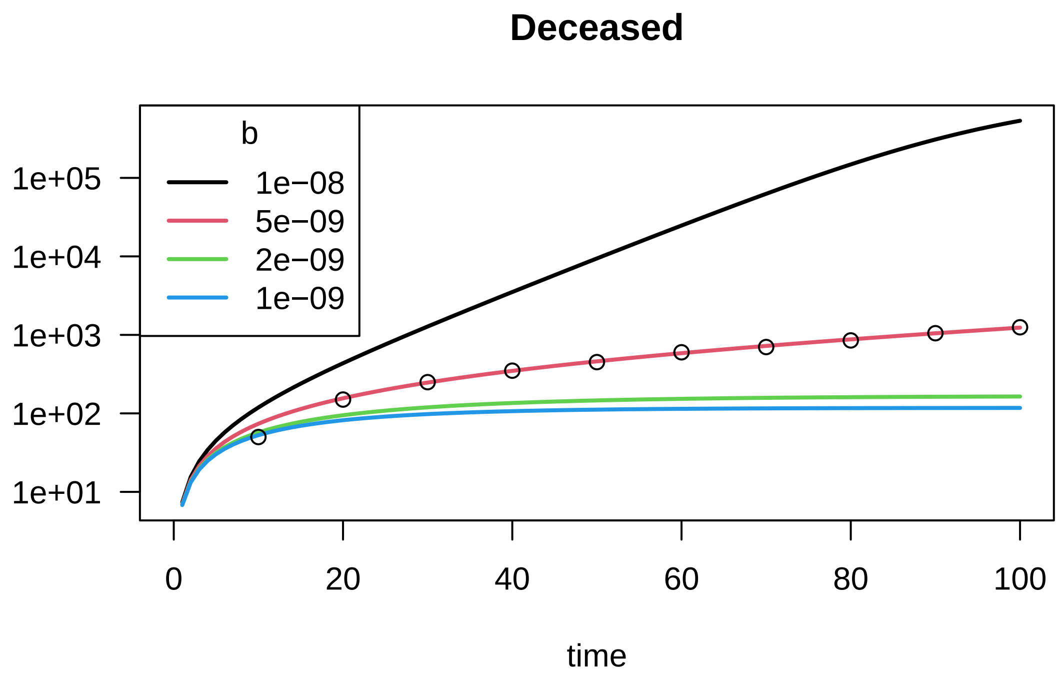
In the model this state variable is called "Deceased", so we use the same name in the data.frame. Additionally, because the model output is a function of time, we use the name "time" for the first column in the data.frame.

We run the model with different values of the parameter b:

```
p1 <- p2 <- p3 <- parms.SIR
p1["b"] <- parms.SIR["b"]/2
p2["b"] <- parms.SIR["b"]/5
p3["b"] <- parms.SIR["b"]/10

out1 <- ode(y=state.SIR, times=time.seq, func=SIR, parms=p1)
out2 <- ode(y=state.SIR, times=time.seq, func=SIR, parms=p2)
out3 <- ode(y=state.SIR, times=time.seq, func=SIR, parms=p3)
```

Now we compare the model output to our records by setting "SIRdata" as the value of the input parameter `obs`:

```
plot(out, out1, out2, out3, obs=SIRdata, log="y", las=1, lty=1, lwd=2)
legend("topleft", legend=c(parms.SIR["b"],p1["b"],p2["b"],p3["b"]),
      col=1:4, lty=1, lwd=2, title="b")
```



It is not so difficult to extract the best parameter value from the above figure.

Steady-state 1D model

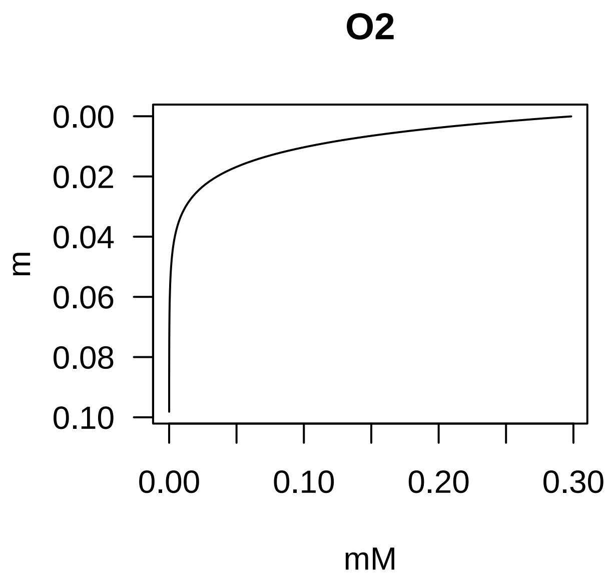
Consider a very simple model that describes first-order consumption of oxygen in the sediment.

```
N      <- 100           # number of boxes
L      <- 0.1           # depth of sediment [m]
por    <- 0.8           # porosity

Grid   <- setup.grid.1D(dx.1=0.0001, L=L, N=N)
DiffO2 <- diffcoeff()$O2*86400    # molecular diffusion, m2/d
tort   <- 1-log(por^2)            # tortuosity

parms.O2 <- c(
  kO2cons = 1.5,              # rate constant, [/d]
  por     = por,              # porosity, [-]
  O2BW    = 0.3,              # BW concentration of O2, [mol/m3]
  DO2     = DiffO2/tort       # Sediment diffusion coeff, [m2/d]
)
```

```
names    <- "O2"              # name of state variable
nspec    <- 1                 # number of species
state    <- rep(0, times=N)  # initial condition

O2func <- function(t, O2, parms){
  with (as.list(parms), {

    O2.tran <- tran.1D(C=O2, C.up = O2BW, D = DO2, VF = por, dx = Grid)
    O2cons  <- kO2cons*O2

    dO2 <- O2.tran$dC - O2cons
    return(list(dO2))
  })
}
```

The model output for the default model parameters looks as follows:

```
std <- steady.1D(y=state, func=O2func, parms=parms.O2,
      nspec=nspec, dimens=N, names=names, positive=TRUE)
plot(std, xyswap=TRUE, grid=Grid$x.mid, las=1, ylab="m", xlab="mM")
```



Including data stored in a file

Assume we measured oxygen concentrations in the sediment porewater every cm and stored the data in a file `obsdata.csv`. It is simple to include this data together with the model results in the same graph.

First, we load the data from the input file:

```
# load data from a csv file and convert it to a data.frame
O2dat.ext <- read.csv(file="obsdata.csv")
O2dat.ext <- data.frame(O2dat.ext)
```

```
# display to see column names and 1 data row
head(O2dat.ext, n=1)
```

```
##   depth_m  O2_mM
## 1       0 0.3034
```

Because the names of the data columns do not match the names of the state variables, we modify them:

```
# modify column names to match the state variables
colnames(O2dat.ext) <- c("x", "O2")
# display to see column names
head(O2dat.ext, n=1)
```

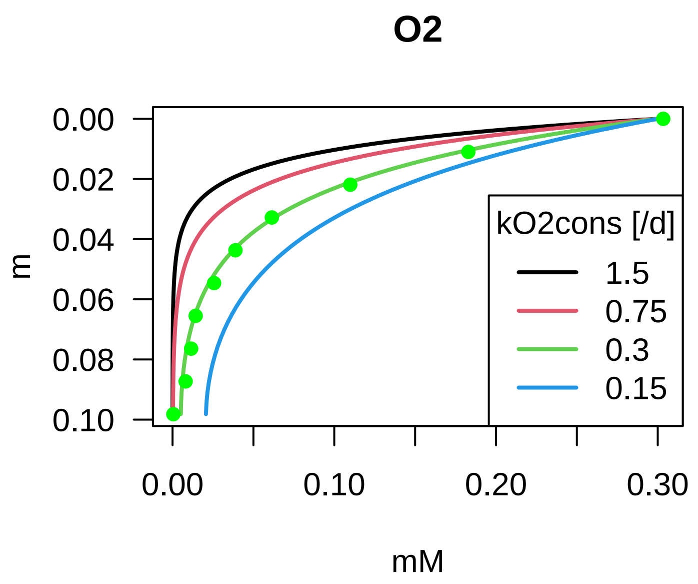
```
##   x     O2
## 1 0 0.3034
```

We run the model with different values of the parameter *kO2cons*.

```
p1 <- p2 <- p3 <- parms.O2
p1["kO2cons"] <- parms.O2["kO2cons"]/2
p2["kO2cons"] <- parms.O2["kO2cons"]/5
p3["kO2cons"] <- parms.O2["kO2cons"]/10

std1 <- steady.1D(y=state, func=O2func, parms=p1,
      nspec=nspec, dimens=N, names=names, positive=TRUE)
std2 <- steady.1D(y=state, func=O2func, parms=p2,
      nspec=nspec, dimens=N, names=names, positive=TRUE)
std3 <- steady.1D(y=state, func=O2func, parms=p3,
      nspec=nspec, dimens=N, names=names, positive=TRUE)
```

Finally, we plot the model results together with the external data. We include the external data by setting `O2dat.ext` as the value of the input parameter `obs`. Here, we additionally modify the shape and color of the data points by specifying the value for the input parameter `obspar`.

```
plot(std, std1, std2, std3, xyswap=TRUE, grid=Grid$x.mid,
     obs = O2dat.ext, obspar = list(pch=16, col="green"),
     lty=1, lwd=2, las=1, ylab="m", xlab="mM")
legend("bottomright", col=1:4, lty=1, lwd=2, title="kO2cons [/d]",
   legend=c(parms.O2["kO2cons"], p1["kO2cons"], p2["kO2cons"], p3["kO2cons"]))
```



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/.

Soetaert Karline (2009). rootSolve: Nonlinear root finding, equilibrium and steady-state analysis of ordinary differential equations. R-package version 1.6. https://CRAN.R-project.org/package=rootSolve

Soetaert, Karline and Meysman, Filip (2012). Reactive transport in aquatic ecosystems: Rapid model prototyping in the open source software R Environmental Modelling & Software, 32, 49-60.

Soetaert, Karline and Thomas Petzoldt (2020). marelac: Tools for Aquatic Sciences. R package version 2.1.10. https://CRAN.R-project.org/package=marelac