One-dimensional reaction-transport model

A template R Markdown file for 1-D reaction-transport models without porosity

your name here

Date of creation here

Contents

Introduction

This template file contains a simple one-dimensional reaction-transport model describing the dynamics of molecular oxygen (O_2, in $mol\ m^{-3}$) and biochemical oxygen demand (BOD, in $mol\ m^{-3}$) in an estuary that is in contact with the air.

The model domain is divided into a grid with N equally sized boxes ($N = 100$).

Both BOD and O2 are *vectors* with a length of N. They represent the concentrations of the state variables in the *center* of the boxes.

With regard to the reaction, a simple first-order decay of BOD that is limited by oxygen is assumed. This reaction consumes both BOD and O2 with a stoichiometry of 1:1. O2 is also exchanged with the atmosphere.

The species are modeled with the following boundary conditions:

- BOD: imposed flux upstream (*flux.up*), imposed concentration downstream ($C.down$).
- O2: imposed concentration upstream ($C.up$), zero-gradient boundary downstream (default, need not be specified).

The partial derivatives related to *transport* are approximated with function `tran.1D()` from the *ReacTran* package. The steady-state and dynamic solutions are obtained using functions from the *rootSolve* and *deSolve* packages, respectively. The latter two packages are loaded together with *ReacTran*.

Model definition

```r
require(ReacTran)  # package with solution methods - includes deSolve, rootSolve

# units: time=days, space=meters, amount=moles, concentration=mol/m3

# model grid
Length <- 1000                                  # [m]
N      <- 100                                   # [-] number of boxes
Grid   <- setup.grid.1D(L = Length, N = N)      # grid of N equally-sized boxes

# Modeled state variables
SVnames <- c("O2", "BOD")

# initial conditions - state variables are defined in the middle of grid cells
O2      <- rep(0.1,   times = N)                # [mol/m3]
BOD     <- rep(0.001, times = N)                # [mol/m3]

# initial state of the system: a vector with all state variables (2*N)
yini   <- c(O2, BOD)

# model parameters
pars <- c(
  D        = 100,   # [m2/d]      dispersion coefficient (tidal mixing)
  v        = 10,    # [m/d]       advection velocity
  rDecay   = 0.05 , # [/d]        rate constant of BOD decay (first-order process)
  kO2      = 0.001, # [mol/m3]    half-saturation O2 concentration for BOD decay
  inputBOD = 10,    # [mol/m2/d]  BOD input rate upstream
  BODdown  = 0.1,   # [mol/m3]    BOD concentration downstream
  O2up     = 0.25 , # [mol/m3]    O2 concentration upstream
  satO2    = 0.3  , # [mol/m3]    saturation concentration of O2 (i.e., solubility)
  k        = 0.1    # [/d]        rate constant for air-water O2 exchange
)

# Model function
BOD1D <-function(t, state, pars) {  # state is a long vector
  with (as.list(pars),{

  # The vectors of the state variables O2 and BOD are
  # "extracted" from the LONG vector state passed to the function as input.
    O2  <- state[   1 :   N ]   # first N elements for O2
    BOD <- state[(N+1):(2*N)]   # second N elements for BOD

  # Transport - tran.1D approximates the spatial derivatives
    # note: for O2: zero-gradient boundary downstream (default)
    tranO2  <- tran.1D(C = O2,
                       C.up = O2up,         # imposed conc upstream,
                       D = D, v = v,        # dispersion, advection
                       dx = Grid)           # Grid

    tranBOD <- tran.1D(C       = BOD,
                       flux.up = inputBOD,  # imposed boundary flux upstream
                       C.down  = BODdown,   # imposed boundary conc downstream
                       D = D, v = v,        # dispersion, advection
                       dx = Grid)           # Grid
```

```
  # rate expressions [mol/m3/d]
    Decay    <- rDecay * BOD * O2/(O2+kO2) # BOD decay, limited by O2
    Aeration <- k * (satO2-O2)             # air-water exchange of O2

  # Time-derivatives: dC/dt = transport + production - consumption [mol/m3/d]
    dO2dt    <- tranO2$dC  + Aeration - Decay
    dBODdt   <- tranBOD$dC            - Decay

  # return vector of time-derivatives and ordinary variables as a list
  list(c(dO2dt, dBODdt),  # time-derivatives
                          # (the same order as state variables!)
  # additional output:

    # process rates along the domain (1D vector)
    Decay           = Decay,                # mol/m3/d
    Aeration        = Aeration,             # mol/m3/d

    # mean process rates (a number)
    MeanDecay       = mean(Decay),          # mol/m3/d
    MeanAeration    = mean(Aeration),       # mol/m3/d

    # rates integrated along the domain (for budgetting)
    TotalDecay      = sum(Decay*Grid$dx),    # mol/m2/d
    TotalAeration   = sum(Aeration*Grid$dx), # mol/m2/d

    # fluxes at domain boundaries (for budgetting)
    BODinflux = tranBOD$flux.up,     # BOD flux INto the system upstream,     mol/m2/d
    BODefflux = tranBOD$flux.down,   # BOD flux OUT of the system downstream, mol/m2/d
    O2influx  = tranO2$flux.up,      # O2 flux INto the system upstream,      mol/m2/d
    O2efflux  = tranO2$flux.down)    # O2 flux OUT of the system downstream,  mol/m2/d
  })
}
```

Model solution

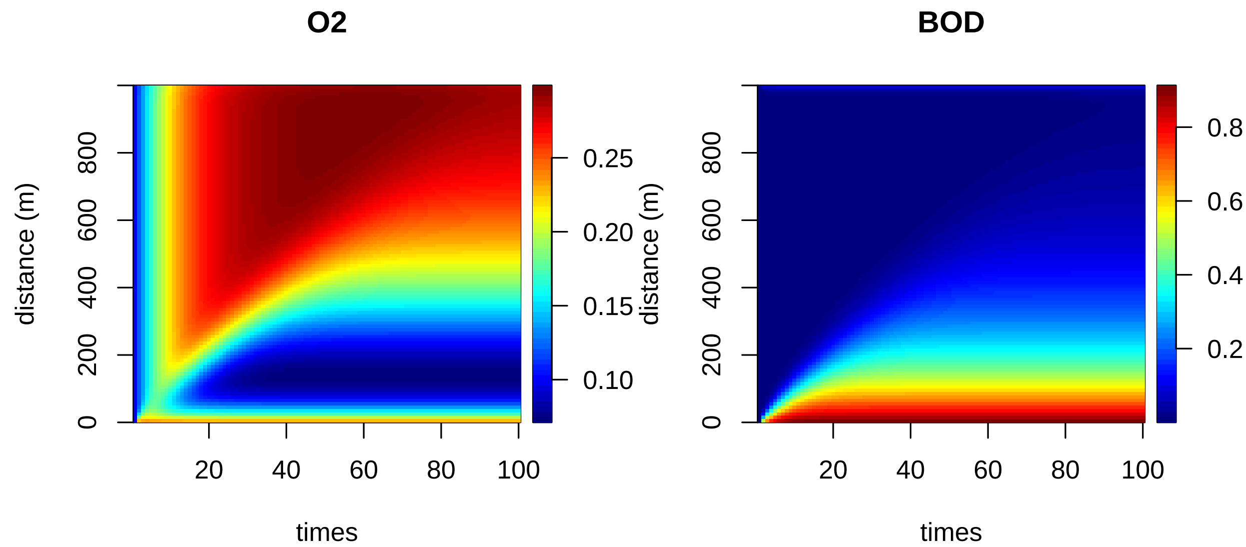
Dynamic solution

Model is solved over 100 days with the initial conditions specified above.

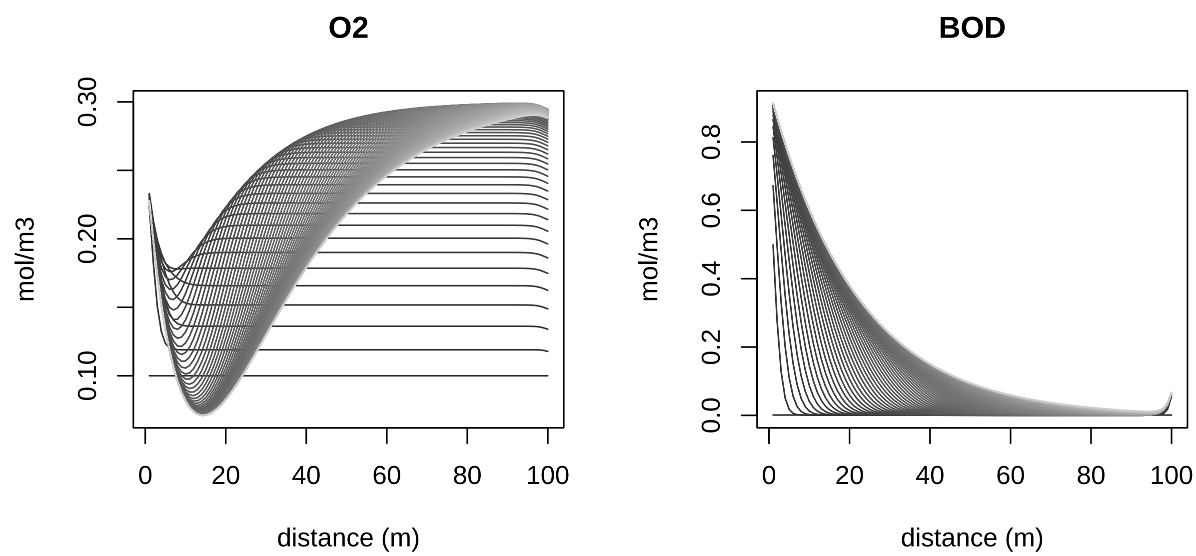
```
outtimes <- seq(from = 1, to = 100, length.out = 100)   # output times

# ode.1D integrates the 1D model
out <- ode.1D(y = yini, parms = pars, func = BOD1D, times = outtimes,
              names = SVnames, nspec = length(SVnames), dimens = N)
```

Display the solution as an image, time plot, and lines.

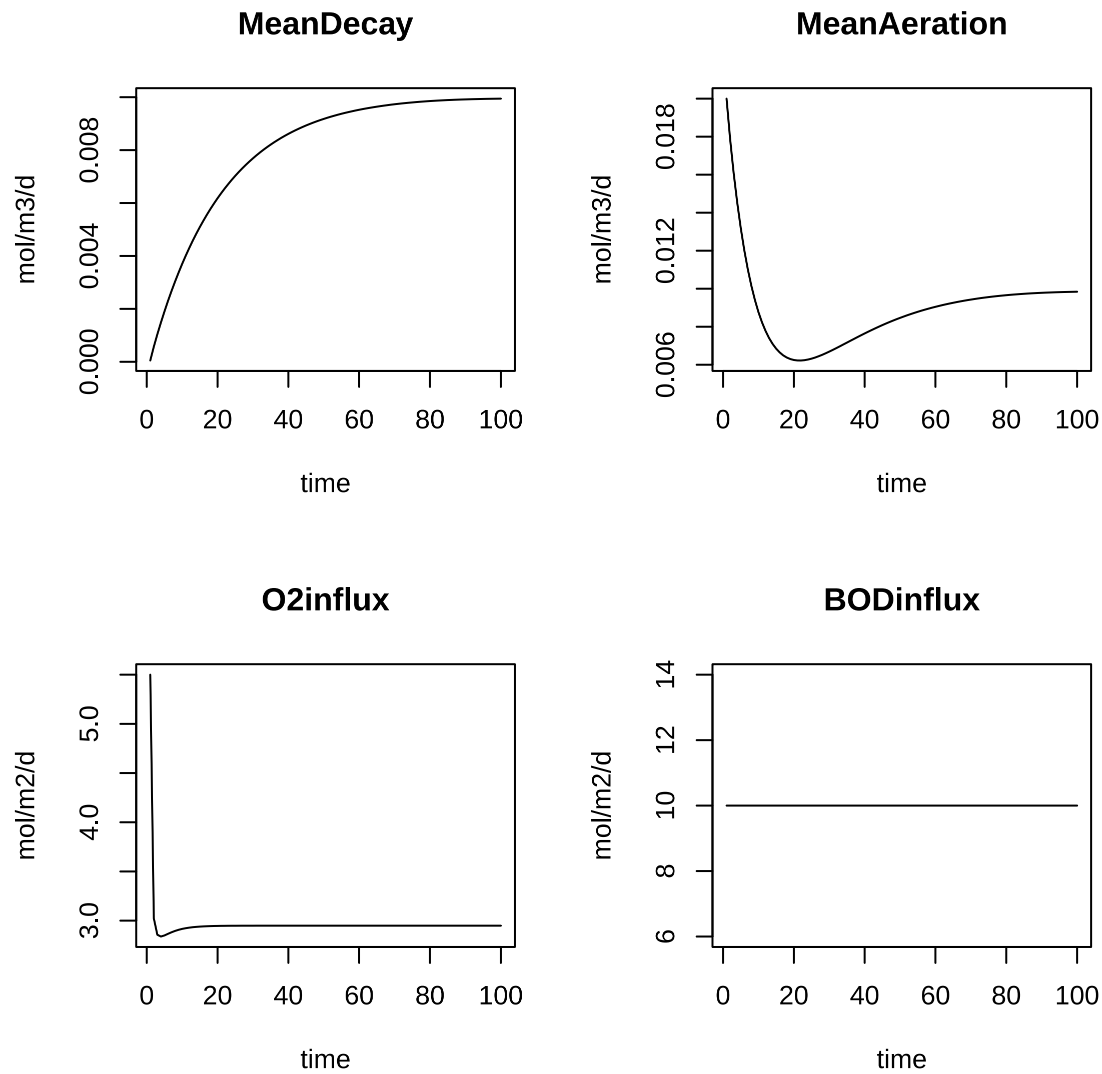
```
image(out, grid = Grid$x.mid, ylab = "distance (m)", legend = TRUE)
```




```
# each curve represents the solution at a given time point
matplot.1D(out, type = "l", lty = 1,
           col = gray(seq(0.2,0.8,len=length(outtimes))),
           xlab = "distance (m)", ylab = "mol/m3")
```




```
plot(out, which=c("MeanDecay", "MeanAeration", "O2influx", "BODinflux"),
          ylab=c("mol/m3/d","mol/m3/d","mol/m2/d","mol/m2/d"))
```



Steady-state solution

Find a steady state solution to the model.

```
# find steady state solution
std <- steady.1D(y = yini, parms = pars, func = BOD1D,
          positive = TRUE,               # to ensure that the solution is positive
          names = SVnames, nspec = length(SVnames), dimens = N,
          atol = 1e-10, rtol = 1e-10)     # to increase the precision of the solution

names(std)          # std holds the state variables (y) and ordinary variables
```

```
##  [1] "y"              "Decay"          "Aeration"       "MeanDecay"
##  [5] "MeanAeration"   "TotalDecay"     "TotalAeration"  "BODinflux"
##  [9] "BODefflux"      "O2influx"       "O2efflux"
```

```r
head(std$y, n = 2)  # std$y contains the state variables (matrix)
```

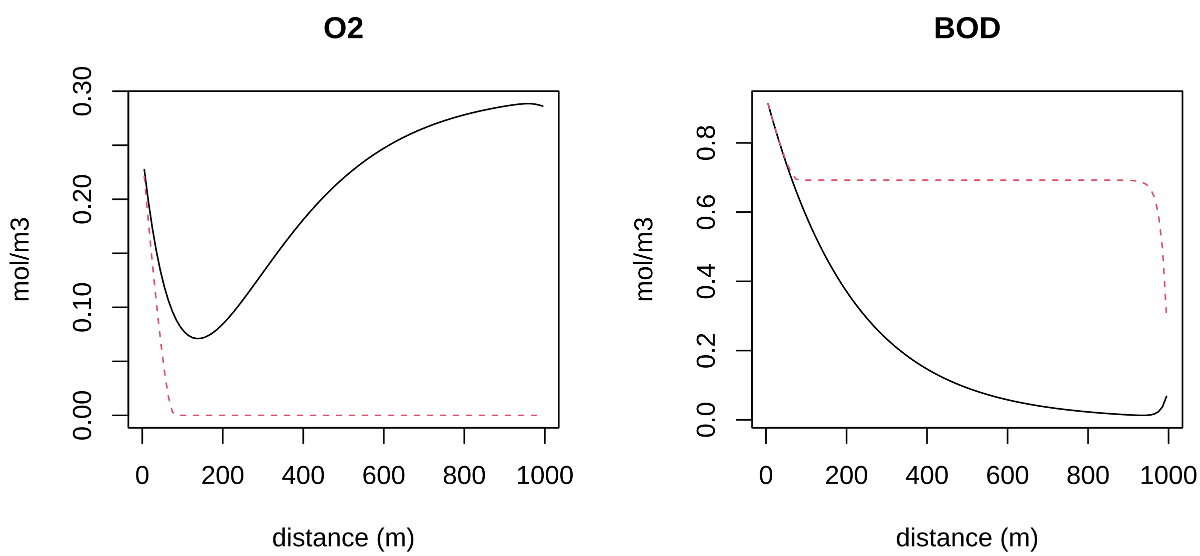
```
##              O2       BOD
## [1,] 0.2275553 0.9130921
## [2,] 0.1984315 0.8716390
```

A second run without aeration.

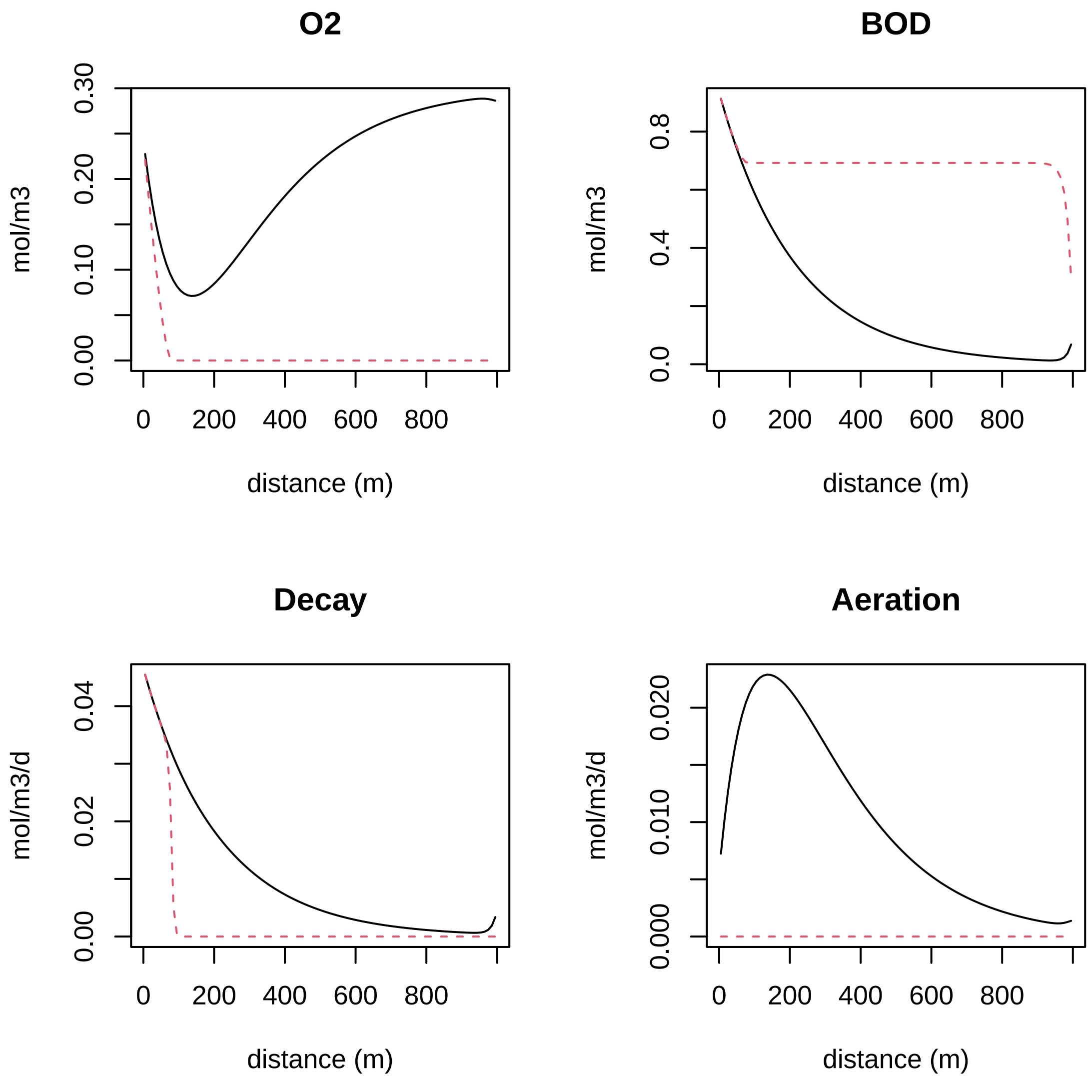
```r
p2 <- pars                # copy of parameter vector
p2["k"] <- 0              # set reaeration rate constant to 0

std2 <- steady.1D(y = yini, parms = p2, func = BOD1D,
          positive = TRUE,                # to ensure that the solution is positive
          names = SVnames, nspec = length(SVnames), dimens = N,
          atol = 1e-10, rtol = 1e-10)     # to increase the precision of the solution
```

Display results.

```r
plot(std, std2, grid = Grid$x.mid,  # plot state variables
     ylab = "mol/m3", xlab = "distance (m)")
```




```r
plot(std, std2, grid = Grid$x.mid,
     which=c("O2", "BOD",            # plot state variables
             "Decay", "Aeration"),   # and process rates
     ylab =c("mol/m3","mol/m3","mol/m3/d", "mol/m3/d"), xlab = "distance (m)")
```



Budgetting of the steady-state model

```
toselect <- c("TotalDecay", "TotalAeration",
              "O2influx", "O2efflux", "BODinflux", "BODefflux")
BUDGET   <- std[toselect]
unlist(BUDGET)
```

```
##    TotalDecay TotalAeration      O2influx      O2efflux     BODinflux
##     9.9598947     9.8737590     2.9488942     2.8627584    10.0000000
##     BODefflux
##     0.0401053
```

```
# should be same
BUDGET$BODinflux - BUDGET$BODefflux
```

```
## [1] 9.959895
```

```r
BUDGET$TotalDecay
```

```
## [1] 9.959895
```

```r
# should be ~0
BUDGET$O2influx - BUDGET$O2efflux -BUDGET$TotalDecay + BUDGET$TotalAeration
```

```
## [1] 0
```

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

Soetaert Karline, 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

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.