One-dimensional reaction-transport model in an aquifer

Spread of organic contamination through an aquifer

Willem Poelman, Denis Power and Jorrit Bakker

22-03-2023

Contents

R implementation

Load the required packages.

```
require(marelac)
require(ReacTran)
```

The model grid and associated properties

Define the model grid and model parameters that vary in space.

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

# Spatial domain of aquifer
Length    <- 500   # Lenght of model grid (i.e. aquifer length [m]
N         <- 500   # number of grid cells

# grid with equally sized boxes
Grid      <- setup.grid.1D(L = Length, N = N)
```

Additional model parameters

Definition of parameters not varing in space

```
temp         <- 10        # Temperature in degrees
salinity     <- 0         # Salinity
O2_sol       <- gas_solubility(S = salinity, t = temp, species = "O2") / 1000 * 0.21 # Oxygen solubilit

f            <- 1/1000    # Factor to convert from µM to mol/m3

def.pars <- c(
 r_aeromin   = 0.002,     # [/h]            Aerobic mineralization rate constant
 r_denitr    = 0.002,     # [/h]            Denitrification rate constant
 r_nitri     = 0.36,      # [/(mol/m3) /h]  Nitrification rate constant
 r_aera      = 0.0003,    # [/h]            Aeration rate constant
 Length      = 500,       # [m]             Modelled length of aquifer
 v_adv       = 10 / 100,  # [m/h]           Transport velocity
 a           = 1.5,       # [m]             Dispersivity
 kO2         = 20 * f,    # [mol/m3]        The affinity constant in the Michaelis-Menten rate limitati
 kNO3        = 35 * f,    # [mol/m3]        The affinity constant in the Michaelis-Menten rate limitati
 riverDON    = 6/12e-3*f, # [mol/m3]        River dissolved organic matter
 riverO2     = 210 * f,   # [mol/m3]        River oxygen concentration
 riverNO3    = 100 * f,   # [mol/m3]        River Nitrate concentration
 riverNH3    = 0 * f,     # [mol/m3]        River oxygen concentration
 O2_sol      = O2_sol,    # [mol/m3]        Solubility of oxygen
 por         = 0.4        # [-]             Porosity
)
```

Definition and initialisation of state variables

```
# Create a vector for each state variable of length N which contains the initial condition
DON       <- rep(0, times = N)
Oxygen    <- rep(0, times = N)
Nitrate   <- rep(0, times = N)
Ammonia   <- rep(0, times = N)
Nitrogen  <- rep(0, times = N)

# Save the state variables as one long vector and make a vector with their respective names
state.ini <- c(DON, Oxygen, Nitrate, Ammonia, Nitrogen)
SVnames   <- c("DON", "Oxygen", "Nitrate", "Ammonia", "Nitrogen")
nspec     <- length(SVnames)
```

Definition of the model function~

Model definition calculating the time-derivatives for each state variable.

```
AquiModel <- function (t, state, parms)
{
  with (as.list(parms),{

    # Unpacking state variables
```

```
  DON      <- state[ (0*N+1) : (1*N) ]    # Next N elements: DON
  O2       <- state[ (1*N+1) : (2*N) ]    # Next N elements: O2
  NO3      <- state[ (2*N+1) : (3*N) ]    # Next N elements: NO3
  NH3      <- state[ (3*N+1) : (4*N) ]    # Next N elements: NH3
  N2       <- state[ (4*N+1) : (5*N) ]    # Last N elements: N2

  # === transport rates ===
  # Transport by dispersion and advection
  # Lower boundaries are zero gradient by default

  tran.DON <- tran.1D(C = DON, C.up = riverDON,    # upper boundary: fixed concentration
                      dx = Grid, VF = por,         # grid and volume fraction (por)
                      D = a * v_adv, v = v_adv)    # dispersion (dispersivity * flow velocity) and ad

  tran.O2 <- tran.1D(C = O2, C.up = riverO2,
                      dx = Grid, VF = por,
                      D = a * v_adv, v = v_adv)

  tran.NO3 <- tran.1D(C = NO3, C.up = riverNO3,
                      dx = Grid, VF = por,
                      D = a * v_adv, v = v_adv)

  tran.NH3 <- tran.1D(C = NH3, C.up = riverNH3,
                      dx = Grid, VF = por,
                      D = a * v_adv, v = v_adv)

  tran.N2 <- tran.1D(C = N2, C.up = 0,
                      dx = Grid, VF = por,
                      D = a * v_adv, v = v_adv)

  # === reaction rates ===

  # DON mineralisation, first order relation with DON concentration, limited by oxygen concentration
  aeroMin <- r_aeromin * (O2 / (O2 + kO2)) * DON

  # Denitrification, first order relation with DON concentration, limited by nitrate concentration, i
  denitri <- r_denitr * (NO3 / (NO3 + kNO3)) * (kO2 / (O2 + kO2)) * DON

  # Nitrification, first order relation with oxygen and ammonia
  nitri   <- r_nitri * O2 * NH3

  # Aeration rate, calculated as the difference between the maximum solubility of oxygen and oxygen c
  aeration <- r_aera * (O2_sol - O2)

  # === mass balances : dC/dt = transport + reactions ===

  dDON.dt  <- ( tran.DON$dC -                      # transport
              aeroMin - denitri)                   # reactions, [mol DON /m3]

  dO2.dt   <- ( tran.O2$dC + aeration              # transport
              - aeroMin - 2 * nitri)               # reactions, [mol O2 /m3]

  dNO3.dt  <- ( tran.NO3$dC -                      # transport
```

```
                  4/5 * denitri + nitri)                # reactions, [mol NO3 /m3]

    dNH3.dt  <- ( tran.NH3$dC +                         # transport
                  (aeroMin + denitri) * 16/106 - nitri) # reactions, [mol NH3 /m3]

    dN2.dt   <- ( tran.N2$dC +                          # transport
                  + 2/5 * denitri)                      # reactions, [mol N2 /m3]

    # Reaction rates integrated over aquifer length
    TotalAeroDeg      = sum(aeroMin * Grid$dx * por)    # [mol DON /m2 /h]
    TotalDenitri      = sum(denitri * Grid$dx * por)    # [mol DON /m2 /h]
    TotalNitri        = sum(nitri * Grid$dx * por)      # [mol NH3 /m2 /h]
    TotalAeration     = sum(aeration * Grid$dx * por)   # [mol O2  /m2 /h]

    return(list(c(dDON.dt, dO2.dt, dNO3.dt, dNH3.dt, dN2.dt), # The time-derivatives, as a long vector

          # Reaction rates
          Aerobic_mineralisation = aeroMin,             # Aerobic mineralisation rate [mol DON /m3 /h]
          Denitrification   = denitri,                  # Denitrification rate [mol DON /m3 /h]
          Nitrification     = nitri,                    # Nitrification rate [mol NH3 /m3 /h]
          Aeration          = aeration,                 # Aeration rate [mol O2 / m3 / h]

          # Aquifer integrated reaction rates for budget
          TotalAeroDeg      = TotalAeroDeg,             # [mol DON /m2 /h]
          TotalDenitri      = TotalDenitri,             # [mol DON /m2 /h]
          TotalNitri        = TotalNitri,               # [mol NH3 /m2 /h]
          TotalAeration     = TotalAeration,            # [mol O2 /m2 /h]

          # Transport fluxes at system boundaries for budget
          DON.up.Flux       = tran.DON$flux.up,         # [mol DON /m2 /h]
          DON.down.Flux     = tran.DON$flux.down,       # [mol DON /m2 /h]
          O2.up.Flux        = tran.O2$flux.up,          # [mol O2 /m2 /h]
          O2.down.Flux      = tran.O2$flux.down,        # [mol O2 /m2 /h]
          NO3.up.Flux       = tran.NO3$flux.up,         # [mol NO3 /m2 /h]
          NO3.down.Flux.    = tran.NO3$flux.down,       # [mol NO3 /m2 /h]
          NH3.up.Flux       = tran.NH3$flux.up,         # [mol NH3 /m2 /h]
          NH3.down.Flux     = tran.NH3$flux.down,       # [mol NH3 /m2 /h]
          N2.up.Flux        = tran.N2$flux.up,          # [mol N2 /m2 /h]
          N2.down.Flux      = tran.N2$flux.down))       # [mol N2 /m2 /h]
 })
}
```

Steady-state solution

Find a steady-state solution with the function *steady.1D* from the package *rootSolve*.

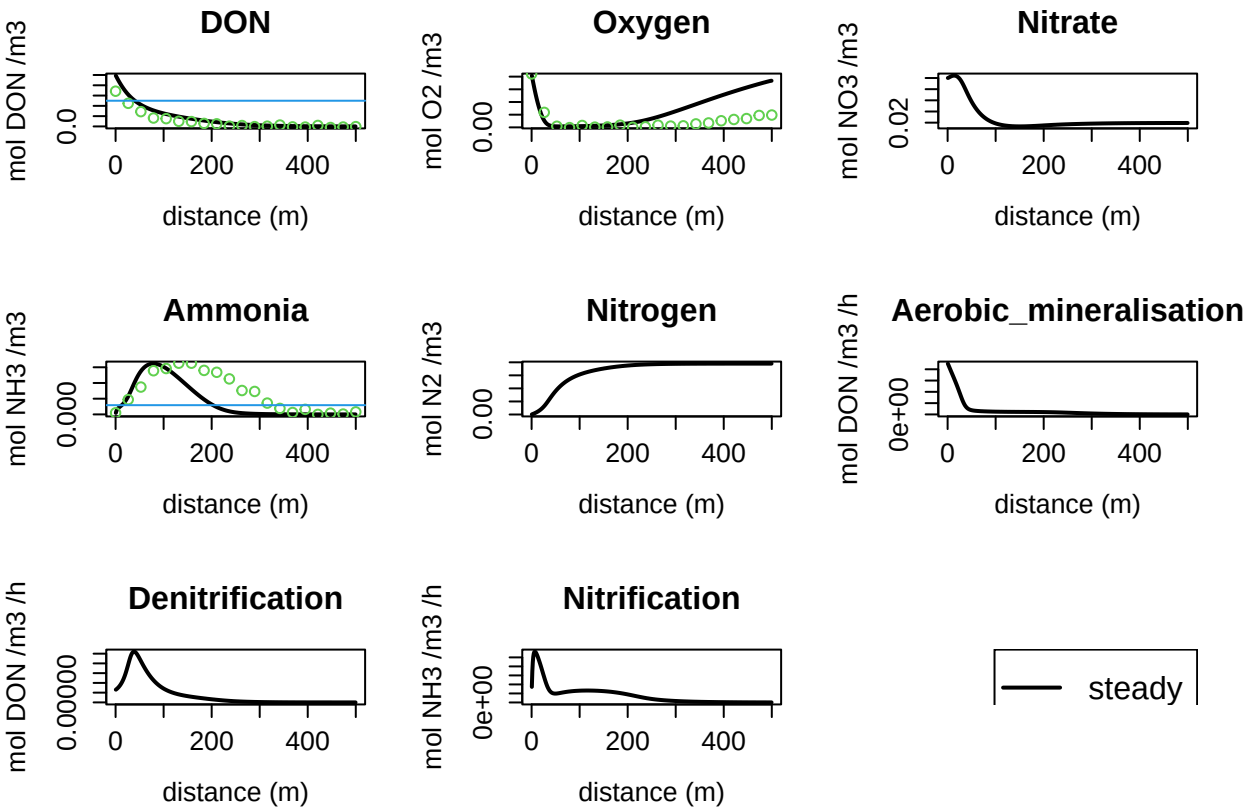
```
std0 <-    steady.1D (y=state.ini, func=AquiModel, parms=def.pars,
                      nspec=nspec, dimens=N, names=SVnames,
                      positive = TRUE, atol = 1e-10, rtol = 1e-10)     # to have only positive values!
def.std <- steady.1D (y = state.ini, func = AquiModel, parms = def.pars,
                      nspec = nspec, dimens = N, names = SVnames,
                      positive = TRUE, atol = 1e-10, rtol = 1e-10)
```

Plotting

Visualize the steady state solutions and load in data

```
# Load in the data file with field data, to be used to calibrate the parameters. Fill in your own path

aquifer_fielddata <- read.csv("/Users/jorrit/Library/CloudStorage/Dropbox/Documenten/ESW_Jaar 1/Reactive
```

```
# Make vectors for the variables to be plotted, the y-axis names and the safety concentrations of some
plt_variables <- c("DON", "Oxygen", "Nitrate", "Ammonia", "Nitrogen", "Aerobic_mineralisation", "Denitri
plt_ylabel <- c("mol DON /m3", "mol O2 /m3", "mol NO3 /m3", "mol NH3 /m3", "mol N2 /m3", "mol DON /m3 /
plt_savelimit <- c(2.498e-1, 0, 4.032e-1, 2.937e-3, 0, 0, 0, 0)

# Make a plot with a 3x3 matrix
par(mfrow = c(3, 3))

# Loop over the to be plotted variables and plot every single one with their respective y-axis labels
for (i in 1:length(plt_variables)) {
  plt_variables[i]
  plot(def.std, grid=Grid$x.mid,
       lty=1, lwd=2,
       which = plt_variables[i], mfrow=NULL,
       ylab  = plt_ylabel[i],
       xlab  = "distance (m)",
       cex.main = 1.5, cex.axis = 1.25, cex.lab = 1.25)

  # If statements state plot the field data and water safety concentrations for the right variables
  if (plt_variables[i] == "DON") {
     points(aquifer_fielddata$distance_m, aquifer_fielddata[[plt_variables[i]]] / (16/106),  col = 3)
  }

  if (plt_variables[i] %in% c("Oxygen", "Ammonia")) {
     points(aquifer_fielddata$distance_m, aquifer_fielddata[[plt_variables[i]]],  col = 3)
  }

  if (plt_variables[i] %in% c("DON", "Nitrate", "Ammonia")) {
     abline(h = plt_savelimit[i], col = 4)
  }
}

# Plot the legend in the remaining panel of the matrix
plot.new()

legend("topright", legend = c("steady", "field", "savety"),
       lty = c(1, -1, 1), lwd = c(2, 1, 2), pch = c(-1, 1, -1), col = c(1,3,4),
       cex = 1.5)
```



Budget

Check the model fluxes and create a system budget

```
# Select which output form the steady state solution to include in the budget
toselect <- c("TotalAeroDeg", "TotalDenitri", "TotalNitri", "TotalAeration", "DON.up.Flux", "DON.down.Fl
BUDGET   <- std0[toselect]
unlist(BUDGET)     # display BUDGET as a vector with named elements rather than a list
```

```
##  TotalAeroDeg  TotalDenitri    TotalNitri TotalAeration   DON.up.Flux
##  1.283854e-02  7.850950e-03  3.066426e-03  1.720822e-02  2.070328e-02
## DON.down.Flux    O2.up.Flux  O2.down.Flux   NO3.up.Flux   NH3.up.Flux
##  1.379106e-05  9.114649e-03  7.351473e-03  3.991361e-03 -5.645526e-05
## NH3.down.Flux    N2.up.Flux  N2.down.Flux
##  6.093405e-08 -2.053100e-05  3.119849e-03
```

Shiny

Use the shiny package to be able to change model parameters and view the model results interactively

```
require(shiny)
```

```
## Loading required package: shiny
```

```
# Define UI (user interface)
UI.Aquifer <- shinyUI(pageWithSidebar(

  # Application title
  headerPanel("OM degradation in aquifer model"),

  # Make a panel in the sidebar which contains sliders for the various parameters
  sidebarPanel(
   sliderInput(inputId="r_aeromin",
               label = "r_aeromin: rate constant of aerobic mineralization [/h]",
               min = 0, max = 0.02, step = 0.0002, value = def.pars["r_aeromin"]),
   sliderInput(inputId="r_denitr",
               label = "r_denitr: rate constant of denitrification [/h]",
               min = 0, max = 0.02, step = 0.0002, value = def.pars["r_denitr"]),
   sliderInput(inputId="r_nitri",
               label = "r_nitri: rate constant of nitrification [/(mol/m3)/h]",
               min = 0, max = 1, step = 0.01, value = def.pars["r_nitri"]),
   sliderInput(inputId="r_aera",
               label = "r_aera: rate constant of aeration [/h]",
               min = 0, max = 0.003, step = 0.00003, value = def.pars["r_aera"]),
   sliderInput(inputId="kO2",
               label = "kO2: Affinity constant in Michaelis-Mentan rate limitation/inhibition term for O
               min = 0, max = 100/1000, step = 1/1000, value = def.pars["kO2"]),
   sliderInput(inputId="kNO3",
               label = "kNO3: Affinity constant in Michaelis-Mentan rate limitation term for NO3 [mol/m3
               min = 0, max = 100/1000, step = 1/1000, value = def.pars["kNO3"]),
   sliderInput(inputId="riverDON",
               label = "riverDON: Dissolved organic nitrogen concentration in river [mol/m3]",
               min = 0, max = 1000/1000, step = 10/1000, value = def.pars["riverDON"]),
   sliderInput(inputId="riverO2",
               label = "riverO2: O2 concentration in river [mol/m3]",
               min = 0, max = 1000/1000, step = 10/1000, value = def.pars["riverO2"]),
   sliderInput(inputId="riverNO3",
               label = "riverNO3: NO3 concentration river [mol/m3]",
               min = 0, max = 500/1000, step = 5/1000, value = def.pars["riverNO3"]),
   sliderInput(inputId="riverNH3",
               label = "riverNH3: NH3 concentration river [mol/m3]",
               min = 0, max = 100/1000, step = 1/1000, value = def.pars["riverNH3"]),

   # Make an action button, which triggers an effect once pressed (defined in server)
   actionButton (inputId="resetButton",
                 label="Reset Parameters"),

   # Make an check box, which toggles an effect on or off
   checkboxInput(inputId="defaultRun",
                 label=strong("Add default run"), value = TRUE),

   checkboxInput(inputId="fieldData",
                 label=strong("Add calibration field data"), value = TRUE),

   checkboxInput(inputId="savetyLimit",
                 label=strong("Add water concentration savety limit"), value = TRUE),
```

```
   br()    # HTML break
  ),

  # Define contents of the main panel
  mainPanel(
      plotOutput("PlotAquifer", height = "700"))
))
```

```
# Define server (back-end)
Server.Aquifer <- shinyServer(function(input, output, session) {

  # Define the action of clicking on clicking the action button
  observeEvent(input$resetButton, {

    # Convert the default parameters to a list
    def.pars.list <- as.list(def.pars)

    # Change the current parameter values to the default ones
    updateNumericInput(session, "r_aeromin",  value = def.pars.list$r_aeromin)
    updateNumericInput(session, "r_denitr",   value = def.pars.list$r_denitr)
    updateNumericInput(session, "r_nitri",    value = def.pars.list$r_nitri)
    updateNumericInput(session, "r_aera",     value = def.pars.list$r_aera)
    updateNumericInput(session, "kO2",        value = def.pars.list$kO2)
    updateNumericInput(session, "kNO3",       value = def.pars.list$kNO3)
    updateNumericInput(session, "riverDON",   value = def.pars.list$riverDON)
    updateNumericInput(session, "riverO2",    value = def.pars.list$riverO2)
    updateNumericInput(session, "riverNO3",   value = def.pars.list$riverNO3)
    updateNumericInput(session, "riverNH3",   value = def.pars.list$riverNH3)
  })

  # Get the model parameters set by the sliders as defined in the UI
  getpars <- reactive( {

    # Get the default parameters as a list
    pars           <- as.list(def.pars)

    # Set the adaptable model parameters to the ones given by the sliders
    pars$r_aeromin <- input$r_aeromin
    pars$r_denitr  <- input$r_denitr
    pars$r_nitri   <- input$r_nitri
    pars$r_aera    <- input$r_aera
    pars$kO2       <- input$kO2
    pars$kNO3      <- input$kNO3
    pars$riverDON  <- input$riverDON
    pars$riverO2   <- input$riverO2
    pars$riverNO3  <- input$riverNO3
    pars$riverNH3  <- input$riverNH3

    # Return pars and convert to a vector
    return(unlist(pars))
  })

  # Get and plot the output, which is visible in the main panel
```

```r
output$PlotAquifer <- renderPlot({

  # Get the adapted model parameters, as defined above
  pars <- getpars()

  # Calculate the steady state solution with these new parameters
  out  <- steady.1D(y = state.ini, parms = pars, func = AquiModel,
                    names = SVnames, nspec = nspec, dimens = N,
                    positive = TRUE, atol = 1e-10, rtol = 1e-10)

  # Make vectors for the variables to be plotted, the y-axis names and the safety concentrations of s
  plt_variables <- c("DON", "Oxygen", "Nitrate", "Ammonia", "Nitrogen", "Aerobic_mineralisation", "Den

  plt_ylabel <- c("mol DON /m3", "mol O2 /m3", "mol NO3 /m3", "mol NH3 /m3", "mol N2 /m3", "mol DON /m

  plt_savelimit <- c(2.498e-1, 0, 4.032e-1, 2.937e-3, 0, 0, 0, 0)

  # Make the plots as a 3x3 matrix
  par(mfrow = c(3, 3))

  # Loop over each variable to be plotted
  for (i in 1:length(plt_variables)) {
    # If the defaultRun checkbox is toggled, plot both the default values and dynamix output
    if (input$defaultRun) {
       plot(def.std, out, grid=Grid$x.mid, lty=1, lwd=2, col = 1:2,
       which = plt_variables[i], mfrow=NULL,
       ylab  = plt_ylabel[i],
       xlab  = "distance (m)",
       cex.main = 2, cex.axis = 1.5, cex.lab = 1.5)
    # Otherwise, only plot the dynamix output
    } else {
       plot(out, grid=Grid$x.mid, lty=1, lwd=2, col = 2,
       which = plt_variables[i], mfrow=NULL,
       ylab  = plt_ylabel[i],
       xlab  = "distance (m)",
       cex.main = 2, cex.axis = 1.5, cex.lab = 1.5)
    }

    # For specific variables, plot the field data if the check box fieldData is toggled
    if (plt_variables[i] == "DON" & input$fieldData) {
       points(aquifer_fielddata$distance_m, aquifer_fielddata[[plt_variables[i]]] / (16/106),  col = 3
    }

    if (plt_variables[i] %in% c("Oxygen", "Ammonia") & input$fieldData) {
       points(aquifer_fielddata$distance_m, aquifer_fielddata[[plt_variables[i]]],  col = 3)
    }

    # For specific variables, plot the water savety concentration if the check box savetyLimit is tog
    if (plt_variables[i] %in% c("DON", "Nitrate", "Ammonia") & input$savetyLimit) {
       abline(h = plt_savelimit[i], col = 4)
    }
  }
```

```
    # Plot the legend in the remaining panel of the matrix
    plot.new()

    legend("topright", legend = c("default", "output", "field", "save"),
           lty = c(1, 1, -1, 1), lwd = c(2, 2, 1, 2), pch = c(-1, -1, 1, -1), col = 1:4,
           cex = 2)

  })  # end output$plot

})  # end of the definition of shinyServer
```

To run the shiny application, run the following chunk

```
shinyApp(ui = UI.Aquifer, server = Server.Aquifer)
```