Local Equilibrium Chemistry in R — part I: One Equilibrium Reaction

Exercises Accompanying the Course Reaction Transport Modelling in the Hydrosphere

Karline Soetaert and Lubos Polerecky, Utrecht University

March 2021

1 Problem

Suppose we have a system with chemical species A, B and AB (such as dissolved NH_3, H^+ and NH_4^+, respectively) that undergo the following processes:

- species AB can dissociate into A and B at a rate controlled by the rate constant k_f ("forward reaction"),
- species A and B can react to form species AB at a rate controlled by the rate constant k_b ("backward reaction"), and
- species A is removed from the system at a rate controlled by the rate constant λ ("net removal process").

In class you have seen that if the equilibration process

$$AB \leftrightarrow A + B$$

is *vastly faster* than the net removal process, then the evolution of the concentrations of A, B and AB can be modelled by assuming that on the time-scale of the slow removal process, the species A, B and AB are in an equilibrium (the so-called *local equilibrium assumption*).

This document will showcase how such local equilibrium chemical models can be implemented in R.

2 The full model

First, we model the dynamics of the chemical species A, B and AB, *without* making the local equilibrium assumption. Thus, the state variables of the model are concentrations [A], [B] and [AB].

2.1 Mass balances

As shown in class, the mass balance equations for the concentrations of A, B and AB are

$$\begin{aligned} \frac{d[A]}{dt} &= R_f - R_b - \lambda \cdot [A] \\ \frac{d[B]}{dt} &= R_f - R_b \\ \frac{d[AB]}{dt} &= -R_f + R_b \end{aligned} \tag{1}$$

where the rates for the forward and backward reactions are $R_f = k_f \cdot [AB]$ and $R_b = k_b \cdot [A] \cdot [B]$, respectively.

If the rate of the slow removal process is zero ($\lambda = 0$), the system will reach an equilibrium. By definition, equilibrium corresponds to the situation where state variables do not change in time, i.e., all time derivatives are equal to zero. This occurs when the rates of the forward and backward reactions are equal, i.e., $R_f = R_b$. Using the rate laws, this equality yields

$$K_{eq} \equiv \frac{k_f}{k_b} = \frac{[A]_{eq} \cdot [B]_{eq}}{[AB]_{eq}}. \tag{2}$$

Here, K_{eq} denotes the *equilibrium constant*, which is defined as the ratio of the rate constants for the forward and backward reactions. Equation 2 states that in equilibrium, the *composition* of the system is such that the quotient, defined as

$$Q = \frac{[A] \cdot [B]}{[AB]}, \tag{3}$$

is equal to the equilibrium constant K_{eq}. Thus, Q is a *measure* of how far from an equilibrium the *system* is.

2.2 Model implementation

To solve this model numerically, we will assume the following parameter values:

parameter	Value	Unit
k_f	1000	s^{-1}
k_b	2×10^6	$(mol\ m^{-3})^{-1}\ s^{-1}$
λ	0.1	s^{-1}

These parameters imply an equilibrium constant of $K_{eq} = 0.5 \times 10^{-3}\ mol\ m^{-3}$.

Additionally, we will assume that *total* concentration of species A ($[Atot] = [A] + [AB]$) is 0.070 $mol\ m^{-3}$, the *total* concentration of species B ($[Btot] = [B] + [AB]$) is 0.06 $mol\ m^{-3}$, and the *initial* concentration of species (state variable) B is 0.001 $mol\ m^{-3}$. Using this information, the initial concentrations of the species (state variables) AB and A can easily be calculated as $[AB] = [Btot] - [B]$ and $[A] = [Atot] - [AB]$.

With that information, we make a model that considers explicitly both the rapid equilibration process and the slower net removal process. That is, we model the dynamics of species A, B and AB based on the mass balance equations (1). We also calculate the *total* concentrations of elements A and B (*Atot* and *Btot*), and the quotient $Q = [A] \cdot [B]/[AB]$, as output variables. As noted above, the quotient is useful because it tells us *how far from equilibrium the system* (described by state variables A, B and AB) *is*. We run the model for 1 millisecond and 10 seconds to illustrate what happens on the time scales of the fast equilibration and slow removal processes.

2.3 R-implementation

We need the R-package deSolve to solve this model:

```
library(deSolve)
```

The parameter values are in a vector called *parms*. The elements in this vector have a name; we will be able to use this name in the model function.

```
parms <- c (kf      = 1000,    # [/s]
            kb      = 2e6,     # [/mol m3 /s]
            lambda  = 0.1)     # [/s]
```

The initial conditions are calculated based on the values given above. The names of the state variables in the initial condition vector *yini.full* also determine the names that we can use in the model function.

```
Atot.ini <- 0.070  # [mol/m3] Initial concentration of total A
Btot.ini <- 0.060  # [mol/m3] Initial concentration of total B
B.ini    <- 0.001  # [mol/m3] Initial concentration of B (STATE VARIABLE)

# initial concentrations of the STATE VARIABLES
AB.ini <- Btot.ini - B.ini
A.ini  <- Atot.ini - AB.ini

# initial conditions of all state variables
yini.full <- c(A = A.ini, B = B.ini, AB = AB.ini)
```

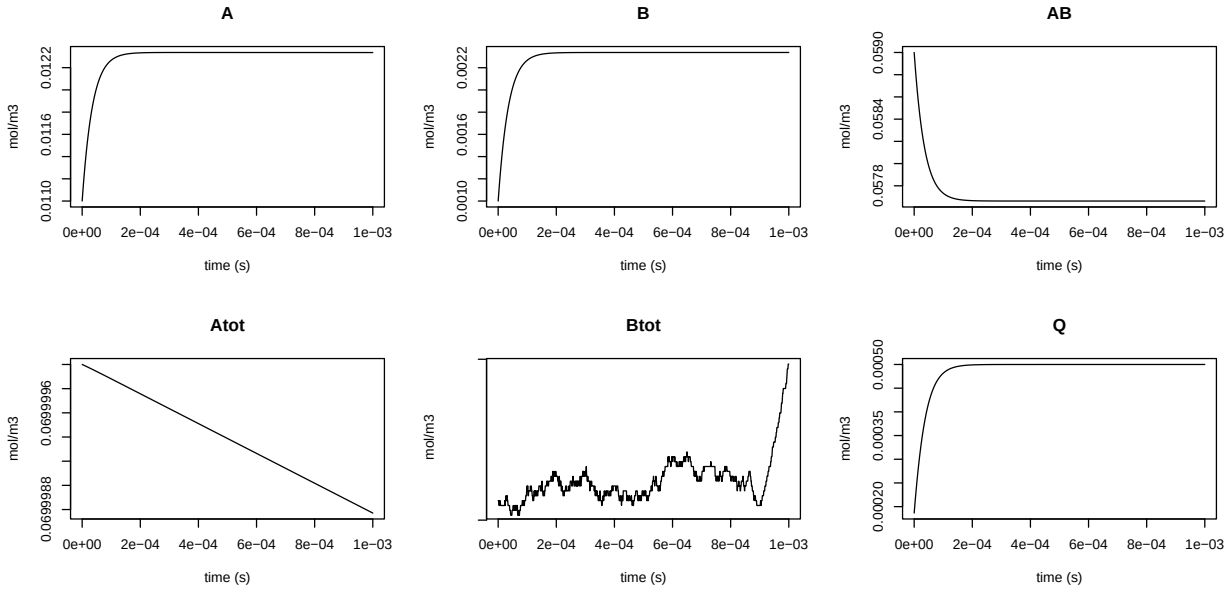
Now we define the model function based on the differential equations (1).

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

  # rate expressions

    # AB <-> A+B
    ratef <- kf * AB     # forward  AB  -> A+B
    rateb <- kb * A*B    # backward A+B -> AB

    rateA <- lambda*A    # slow removal, specific to A

    # mass balances
    dA  <-   ratef - rateb - rateA
    dB  <-   ratef - rateb
    dAB <- - ratef + rateb

    return(list(c(dA, dB, dAB),       # the time derivatives
           Atot = A+AB, Btot = B+AB,  # total A and B
           Q = A*B/AB))               # quotient
  })
}
```

To run the model we define the time sequence for which we want output (*times*), and solve the model using the *ode* function. Finally, we plot the dynamics of the state variables and of the three output variables.

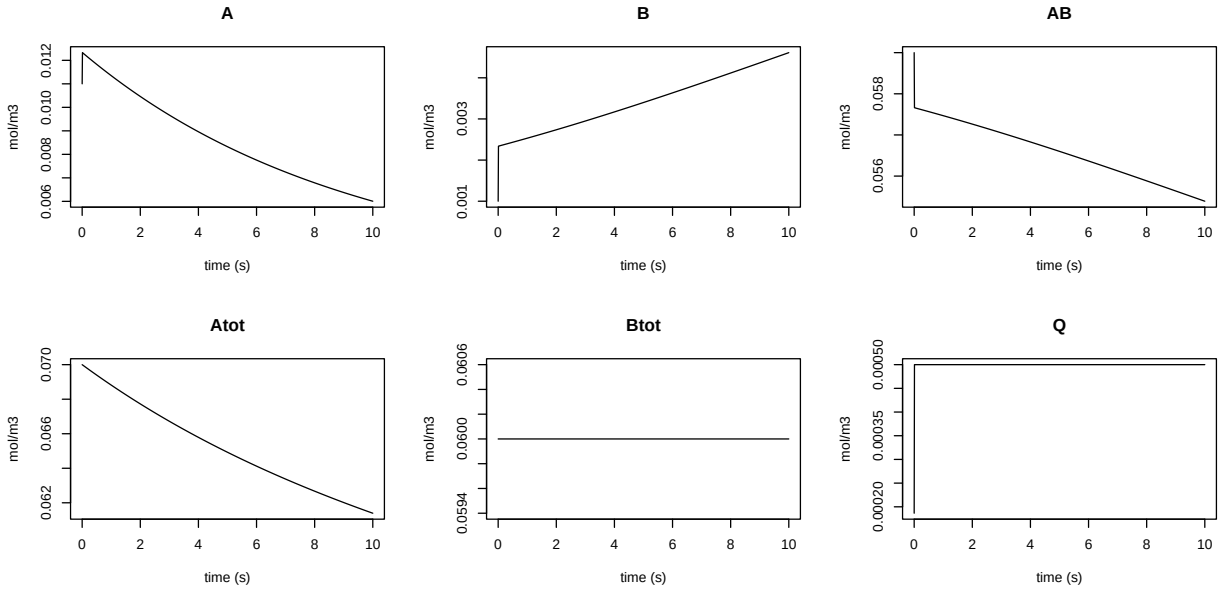
First, we run the model for 1 ms to illustrate how fast the equilibrium is reached. We see that, initially, the system is in a *disequilibrium*, as the quotient Q is different (lower) than the equilibrium constant $K_{eq} = 0.5$. However, we see that Q reaches the value of 0.5 in about 0.2 ms, which is the time-scale on which the equilibrium is reached. We also see that on this time-scale the effect of the slow removal process is negligible ([A] practically does not change after reaching equilibrium).

```
times     <- seq(from=0, to=1e-3, length.out = 1000)
out.full  <- ode(y=yini.full, times=times, func=FullModel, parms=parms)
plot(out.full, mfrow=c(2,3), ylab="mol/m3", xlab="time (s)")
```



Now, we run the model for 10 s to illustrate the effects on the time-scale of the slow removal process. As we can see from the quotient, on this time-scale the species A, B and AB can be considered to be always in an equilibrium.

```
times     <- seq(from=0, to=10, length.out=1000)
out.full  <- ode(y=yini.full, times=times, func=FullModel, parms=parms)
plot(out.full, mfrow=c(2,3), ylab="mol/m3", xlab="time (s)")
```



3 Model based on the local equilibrium approximation

Now we model the system assuming a *local equilibrium of the fast reversible reactions.* As explained in class, this is done by considering *different state variables*, namely the *total A* ($Atot = A + AB$) and *total B*

($Btot = B + AB$), and by assuming that the species A, B and AB are *always* in an equilibrium, meaning that at *any time* the following relationship is valid (compare with equations 2 and 3):

$$K_{eq} = \frac{[A] \cdot [B]}{[AB]}. \tag{4}$$

3.1 Mass balance equations and additional relationships

As explained in class, the local equilibrium condition implies the following mass balance equations for the new state variables:

$$\begin{aligned} \frac{d[Atot]}{dt} &= -\lambda \cdot [A] \\ \frac{d[Btot]}{dt} &= 0 \end{aligned} \tag{5}$$

Additionally, it implies the following relationships between the new and the original state variables:[1]

$$[B] + \frac{[B]}{K_{eq} + [B]} \cdot [Atot] = [Btot] \tag{6}$$

$$[A] = \frac{K_{eq}}{K_{eq} + [B]} \cdot [Atot] \tag{7}$$

$$[AB] = \frac{[B]}{K_{eq} + [B]} \cdot [Atot] \tag{8}$$

It is important to note the meaning of equations 5–8. Equation 6 shows how to calculate the concentration of B for any given concentration of Atot and Btot. Once this concentration is known, the concentrations of A and AB are calculated using equations 7 and 8, respectively. Then, using the concentration of A, the rate of change of Atot can be calculated from equation 5. These steps outline the *approach* of how to model the system under the local equilibrium assumption.

3.2 Solving for the equilibrium concentration of B

As explained above, finding the concentration of B is the critical step in this approach. This is done by solving for [B] in equation 6. This can be done in two ways: analytically or numerically.

3.2.1 An analytical approach

First, because the equation for finding [B] is not that complicated, it *is* possible to find an *analytical solution*. Indeed, rearranging equation 6 yields a quadratic equation from which [B] can readily be calculated. As shown in class, the solution is

$$[B] = \frac{1}{2}\left(-\beta + \sqrt{\beta^2 + 4K_{eq} \cdot [Btot]}\right), \tag{9}$$

where

$$\beta = K_{eq} + [Atot] - [Btot].$$

[1] From $Keq = \frac{[A]\cdot[B]}{[AB]}$ we derive $[AB] = \frac{[A]\cdot[B]}{Keq}$, which yields $[Atot] = [A] + [AB] = [A] \cdot \left(1 + \frac{[B]}{Keq}\right)$ and $[A] = \frac{K_{eq}}{K_{eq}+[B]} \cdot [Atot]$.

3.2.2 A numerical approach

For more complex equilibria, such as those involving more than three chemical species,[2] finding an analytical solution may be *too difficult* or even *impossible*. In this case, *numerical approximation* can be used. For this particular case, we rearrange equation 6 as

$$f([B]) = [B] + \frac{[B]}{K_{eq} + [B]} \cdot [Atot] - [Btot] = 0, \tag{10}$$

which shows that [B] can be found by finding the root[3] of the function $f([B])$ defined on the left-hand side of this equation. In R, this is done using the *uniroot* function, as shown below.

3.3 R-implementation

Now we are ready to implement the model in R. We illustrate both approaches, one based on the analytical solution of [B] and the other based on the numerical solution of [B].

3.3.1 An analytical approach

Implementing this approach in R is easier because we have done the necessary work already by solving equation 6 for [B]. In addition to the time-derivatives of the state variables Atot and Btot, we also return the values of A, B, AB and Q as the output.

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

    # equilibrium constant
    Keq  <- kf/kb

    # calculate B from Atot and Btot (eq. 9)
    beta <- Keq + Atot - Btot
    B    <- 0.5 * (-beta + sqrt(beta^2 + 4*Keq*Btot))
    # calculate A from Atot and B (eq. 7)
    A    <- Keq /(Keq+B)*Atot

    # mass balance equations for Atot and Btot (eq. 5)
    dAtot <- -lambda * A
    dBtot <- 0

    return(list(c(dAtot, dBtot),
                B = B, A = A, AB = Atot-A,
                Q = A*B/(Atot-A) ))
  })
}
```

3.3.2 A numerical approach

To implement this approach, we first define a function *solveB* for finding [B] at equilibrium, i.e., the root of equation 10. Note that, within the scope of function *solveB*, another function is defined (*rootFun*), that implements the function of [B] in equation 10. This function is then used by R's *uniroot*. We need to input

[2] For example, carbonate equilibrium in water involves four species: H_2CO_3, HCO_3^-, CO_3^{2-}, and H^+.
[3] Root of a function $f(x)$ is a value of x where $f(x) = 0$.

suitable *lower* and *upper* boundaries between which the root is sought. We choose 0 and Btot, since [B] cannot be negative or larger than [Btot].

```
solveB <- function(Keq, Atot, Btot){

  # function whose root has to be sought (eq. 10)
  rootFun <- function(B) {
    return( B + B/(Keq+B)*Atot - Btot )
  }

  # uniroot will find the root; it returns a list with $root being the solution
  r <- uniroot(f = rootFun, lower = 0, upper = Btot, tol = 1e-20)
  return( r$root )
}
```

The remaining steps are similar as in the analytical approach. The only difference is that instead of calculating B analytically, we use the above function *solveB*.

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

    Keq  <- kf/kb

    # calculate B from Atot and Btot numerically, using solveB
    B    <- solveB(Keq = Keq, Atot = Atot, Btot = Btot)
    # calculate A from Atot and B (eq. 7)
    A    <- Keq /(Keq+B)*Atot

    # mass balance equations for Atot and Btot (eq. 5)
    dAtot <- -lambda * A
    dBtot <- 0

    return(list(c(dAtot, dBtot),
                B = B, A = A, AB = Atot-A,
                Q = A*B/(Atot-A) ))
  })
}
```

Finally, we run both models for 10 seconds.

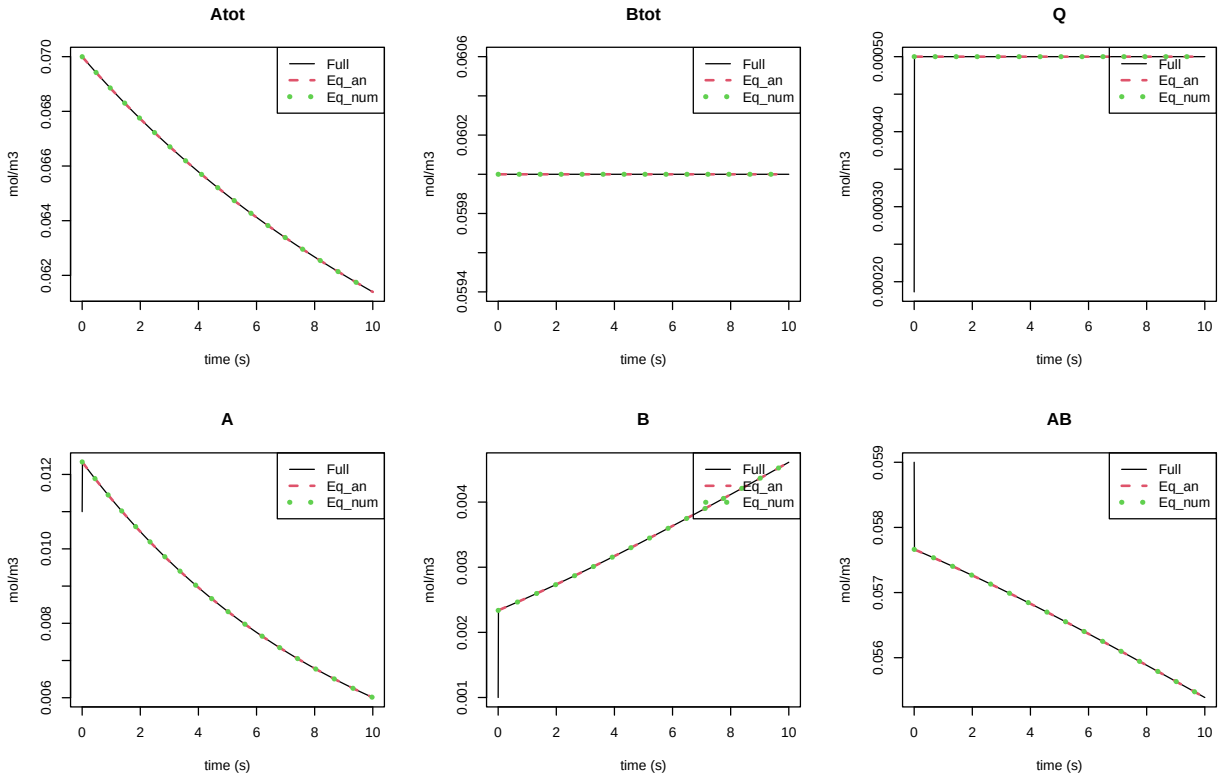
```
yini.eq <- c(Atot = Atot.ini, Btot = Btot.ini)
out.eq_an  <- ode(y=yini.eq, times=times, func=EqModel_an,  parms=parms)
out.eq_num <- ode(y=yini.eq, times=times, func=EqModel_num, parms=parms)
```

4 Compare all models

We output the dynamics of the full model (black), alongside the equilibrium formulation (red and green). To optimize the plotting code, we introduce a function *plot_AB* that plots a specific variable (A, B, etc.) for all models in one graph. This function illustrates the usefulness of having named columns in matrices, as we do not need to worry that the *order* of state variables in columns of the outputs "out.full", "out.eq_an" and "out.eq_num" is *different*.

```
# plotting function
plot_AB <- function(varname, out1, out2, out3) {
  plot (out1[,"time"], out1[,varname], type="l", main=varname,
```

```
      ylab="mol/m3", xlab="time (s)", col=1, lty=1, lwd=1)
  lines(out2[,"time"], out2[,varname],  col=2, lty=2, lwd=2)
  lines(out3[,"time"], out3[,varname],  col=3, lty=3, lwd=4)
  legend("topright",
      legend=c("Full", "Eq_an", "Eq_num"),
      col=c(1,2,3), lty=c(1,2,3), lwd=c(1,2,4))
}

# plot variables Atot, Btot, Q, A, B and AB in separate graphs
par(mfrow = c(2,3))
plot_AB("Atot", out.full, out.eq_an, out.eq_num)
plot_AB("Btot", out.full, out.eq_an, out.eq_num)
plot_AB("Q",    out.full, out.eq_an, out.eq_num)
plot_AB("A",    out.full, out.eq_an, out.eq_num)
plot_AB("B",    out.full, out.eq_an, out.eq_num)
plot_AB("AB",   out.full, out.eq_an, out.eq_num)
```



The graphs show that in the full model, the initial concentrations of A, B and AB were not at equilibrium (the quotient Q calculated from equation 3 is *not* equal to K_{eq}). However, the equilibrium concentration is rapidly reached, as indicated by the quotient Q rapidly reaching K_{eq}. The full and approximate models yield indistinguishable results once the equilibrium has been reached.

4.1 Benchmarking

In class, it was mentioned that if we use the approximate model instead of the full model (because we are not interested in, or can ignore, the fast equilibration process), we can save considerable computational resources.

To see the effects in this particular case, we compare the speed of solution for the three implementations: the full model, and the analytical and numerical versions of the local equilibrium model.

We run each model 100 times, so that random effets due to the computer being otherwise engaged are filtered out. The time output by R-function *system.time* is in seconds.

```
times <- seq(from=0, to=10, length.out=100)

cat("full model: \n")
```

```
## full model:
```

```
print(system.time(
  for (i in 1:100)
    out.full <- ode(y=yini.full, times=times, func=FullModel, parms=parms)
  )/100)
```

```
##    user  system elapsed
## 0.00474 0.00000 0.00474
```

```
cat("analytical: \n")
```

```
## analytical:
```

```
print(system.time(
  for (i in 1:100)
    out.eq_an <- ode(y=yini.eq, times=times, func=EqModel_an, parms=parms)
  )/100)
```

```
##    user  system elapsed
## 0.00497 0.00000 0.00497
```

```
cat("numerical:  \n")
```

```
## numerical:
```

```
print(system.time(
  for (i in 1:100)
    out.eq_num <- ode(y=yini.eq, times=times, func=EqModel_num, parms=parms)
  )/100)
```

```
##    user  system elapsed
## 0.01942 0.00000 0.01942
```

Two notes:

- The fact that the full model is so efficiently solved is because the default deSolve integrator (ode) is particularly well suited to solve so-called "stiff" problems (i.e., problems with largely different time scales). When using a traditional integration routine (Euler or Runge-Kutta), the solution would take several orders of magnitude longer.
- The apparant inefficiency of the numerically solved equilibrium model is due to the fact that *uniroot* is not the most efficient solver for this problem.

5 Application to ammonia degassing

Here you will check your understanding of the concepts introduced above by modeling the effect of ammonia degassing on water pH. A possible environmental scenario could be a lake polluted with a spill of ammonia from an industrial source.

5.1 Task

Consider a water body with an initial pH of 8 ($pH = -log([H^+])$, where $[H^+]$ is the concentration of protons in $mol\ L^{-1}$!) and an initial concentration of *total* dissolved ammonia ($NH_x = NH_3 + NH_4^+$) of 1 $mol\ m^{-3}$. Dissolved ammonia species NH_3 is degassing according to the first-order kinetics (assume a rate constant of 1 d^{-1}).

Write a model that will predict the change of water pH, along with the change of $[NH_x]$, as a function of time due to ammonia (NH_3) degassing.

Assume that the equilibration process $NH_4^+ \leftrightarrow NH_3 + H^+$ is much faster than the degasing (equilibrium constant for this process is approximately $K_{eq} = 5.38 \times 10^{-10}\, mol\, L^{-1}$). Thus, you can use the local equilibrium assumption to solve the model. Additionally, assume that degassing of ammonia is the *only* process that affects the water pH. That is, assume that other chemical species, such as dissolved CO_2, which would be a strong buffer counter-acting the pH changes induced by ammonia degassing, are *not* present in the water. This last assumption is required to make the model simple and focused on one process: ammonia degassing.

You can start with the R-markdown template file *RTM_0D.Rmd* to implement this model.[4]

[4]You can obtain this file from Rstudio: File → new File → Rmarkdown → from template → RTM_0D. Save this file under a different name. Do not forget to change the heading of this file.

5.2 Solution

The solution to this problem follows directly from the above explanation once we realize that the species A corresponds to NH_3, species B corresponds to H^+, species AB corresponds to NH_4^+, the lump-sum species *Atot* corresponds to total dissolved ammonium ($NH_x = NH_3 + NH_4^+$), and *Btot* corresponds to total positive charge ($H_x^+ = H^+ + NH_4^+$). A small modification is required, though, because we do not know the actual rate constants for the forward and backward reaction (this is typically the case for this type of problems). Thus, we will use K_{eq} as the model parameter instead of k_f and k_b.

We define the model function in analogy to the *EqModel_num* function. We can leave the solveB function untouched and reuse it.

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

    # calculate H from NHx and Hx numerically, using solveB
    H   <- solveB(Keq = Keq, Atot = NHx, Btot = Hx)
    # calculate NH3 and NH4+ from NHx and H (eq. 7 and 8)
    NH3 <- Keq/(Keq+H) * NHx
    NH4 <- NHx - NH3

    # mass balance equations for NHx and Hx (eq. 5)
    dNHx <- -lambda * NH3
    dHx  <- 0

    return(list(c(dNHx, dHx),
                H = H, NH3 = NH3, NH4 = NH4, # NH4 corresponds to [NH4+]
                pH = -log10(H),      # return also pH
                Q = NH3*H/NH4 ))     # return also Q
  })
}
```

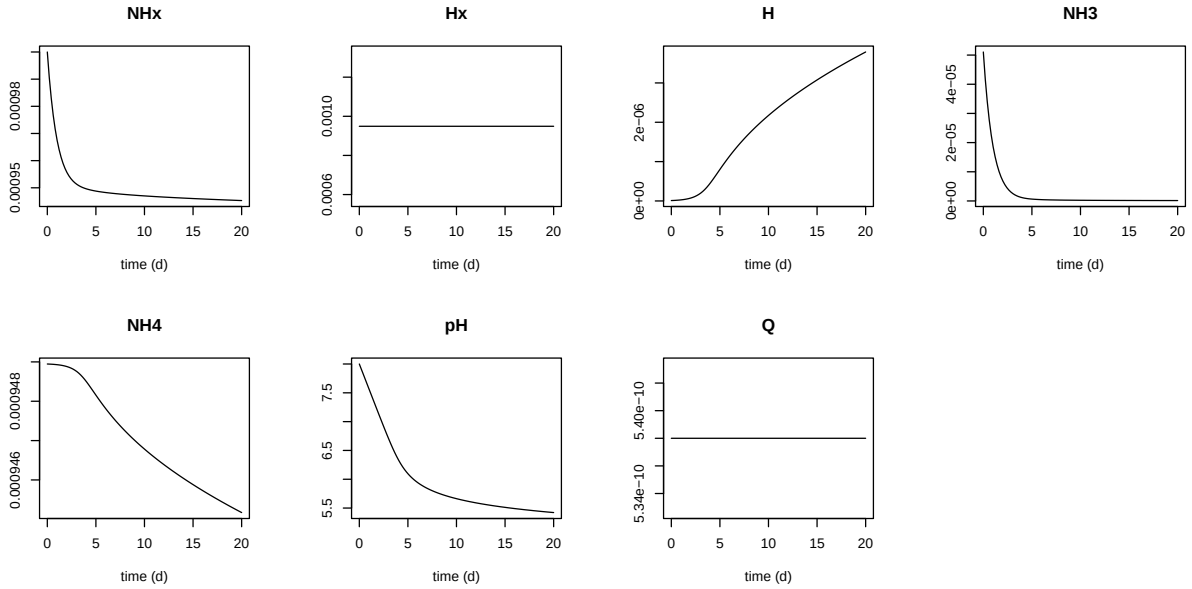
Now, we define the model parameters and initial conditions. Here, we need to keep in mind that $pH = -log([H^+])$, where the proton concentration is in mol/L. Thus, all state variables, as well as the equilibrium constant K_{eq}, must be in this unit, too. Because the rate constant is in the unit of d^{-1}, we are going to solve the model on the time-scale of days.

```
parms <- c (Keq = 5.38e-10,      # mol/L
            lambda = 1)          # 1/d

# Initial concentration of total dissolved ammonia (given)
NHx.ini <- 1e-3 # mol/L
# Initial proton concentration, calculated from pH (given)
H.ini  <- 10^(-8) # 10^(-pH), mol/L
# Initial total positive charge concentration, calculated from eq. 6
Hx.ini <- H.ini + H.ini/(parms[["Keq"]] + H.ini) * NHx.ini

# initial conditions, lump-sum state variables NHx and Hx
yini <- c(NHx = NHx.ini, Hx = Hx.ini)
```

Now we are ready to run the model and plot the results as a function of time. We calculate the results for 20 days.

```
require(deSolve)
times <- seq(from=0, to=20, length.out=100) # days
out <- ode(y=yini, times=times, func=AmmoniaDegassing, parms=parms)
```

```
plot(out, mfrow=c(2,4), xlab="time (d)")
```



We see that the most dramatic changes occur within the first 5 days, where the pH decreases from 8 until about 6. This shift towards a lower pH (higher $[H^+]$) causes the speciation of total ammonium to shift towards low $[NH_3]$. After 5 days, the NH_3 concentration is so low that the rate of its removal is close to zero (because the rate is proportional to $[NH_3]$), leading to a much slower pH dynamics.

An interesting outcome of the model is that the *total* ammonia concentration in the lake does *not* decrease appreciably (from 1 mmol/L to about 0.95 mmol/L). This means that degassing of ammonia from a pH-unbuffered system is *not* going to remove the ammonia from the lake water. The lake would stop smelling bad, but it would not stop being polluted. Note, however, that this corresponds to an unrealistic assumption that ammonia degassing is the only process that affects the pH of the lake water.

6 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