UNIVERSITEIT GENT

DEPARTMENT OF APPLIED MATHEMATICS,
BIOMETRICS AND PROCESS CONTROL

Calculation of pH and concentration of equilibrium components during dynamic simulation by means of a charge balance

Eveline I.P. Volcke, Stijn Van Hulle, Tolessa Deksissa, Usama Zaher and Peter Vanrolleghem

March, 2005

Coupure Links 653, B-9000 Gent, Belgium
tel.: +32(9)264.59.32 / fax: +32(9)264.62.20 / WWW: http://biomath.rug.ac.be

1. Introduction

Biological conversion reactions that involve proton consumption or production affect the pH of the medium in which they take place and as a result also the concentrations of all components that are involved in chemical equilibria at the same time. Significant pH effects and resulting equilibrium component concentration changes will also influence the biological conversion rates.

Consequently, when modelling systems in which significant pH changes are to be expected, both biological conversion reactions and chemical dissociation reactions (acid/base chemistry) must be modelled adequately in order to obtain accurate simulation results. This can be done in two ways

1. All components involved in an equilibrium, e.g. CO_2, HCO_3^- and CO_3^{2-}, are considered separately. The stoichiometry of the biological conversion reactions is expressed in one of these components (to be chosen), the biological conversion rates are expressed in the actual form by which they are influenced, typically the unionized form, e.g. the substrate for ammonia oxidation by *Nitrosomonas* is NH_3 rather than NH_4^+. Mass balances (differential equations) are set up for all individual components (including H^+).

The conversion terms include biological and chemical conversions. However, chemical dissocation reactions proceed much faster than the biological conversions and these different time scales hamper the integration of the differential equations. Stiff solvers are required to solve the mass balances.

2. In the second approach, the chemical dissociation reactions are considered to be in equilibrium in comparison to the biological conversion reactions. Lumped components are introduced, of which the concentrations equal the total concentration of all equilibrium forms of a certain component, e.g. $TIC = CO_2 + HCO_3^- + CO_3^{2-}$. Mass balances (differential equations) are set up for lumped components and for components that are not involved in chemical equilibrium reactions and only consider biological conversion reactions. The stoichiometry of the biological conversion reactions is expressed in terms of the lumped components, the biological process rates are expressed in the actual form by which they are influenced, as in the first approach. The mass balances are solved for the concentration of the lumped components and the components that are not involved in equilibria. Subsequently, the pH is calculated from the concentrations of the lumped components and the components that don't take part in equilibria by means of a charge balance (electro-neutrality equation). Once the pH is known, besides the concentrations of the lumped components, the corresponding concentrations of individual components involved in equilibria can be calculated as well. This report describes a general procedure for the calculation of pH and corresponding equilibrium concentrations. The implementation of this procedure is illustrated by three applications in which important pH changes occur.

2. General procedure for calculating pH and equilibrium concentrations by means of a charge balance

2.1. Identification of chemical equilibria and definition of lumped components

The following 'general' procedure includes the following types of chemical dissociation reactions:

Type 1: Monoprotic acid with monovalent positive charge, dissociates into neutral base form.

$$HA^+ \leftrightarrow H^+ + A$$

Type 2: Monoprotic neutral acid, dissociates into base with monovalent negative charge.

$$HB \leftrightarrow H^+ + B^-$$

Type 3: Biprotic neutral acid, dissociates in two steps into base with bivalent negative charge.

$$H_2D \leftrightarrow H^+ + HD^-$$

$$HD^- \leftrightarrow H^+ + D^{2-}$$

Type 4: Triprotic neutral acid, dissociates in three steps into base with trivalent negative charge.

$$H_3P \leftrightarrow H^+ + H_2P^-$$

$$H_2P^- \leftrightarrow H^+ + HP^{2-}$$

$$HP^{2-} \leftrightarrow H^+ + P^{3-}$$

Besides, also the water equilibrium is taken into account:

$$H_2O \leftrightarrow H^+ + OH^-$$

As chemical equilibrium reactions proceed much faster than the biological conversion reactions, they are assumed to be in steady state, so the following equations apply:

$$K_A = \frac{H^+ \cdot A}{HA^+} \quad (1)$$

$$K_B = \frac{H^+ \cdot B^-}{HB} \quad (2)$$

$$K_{D1} = \frac{H^+ \cdot HD^-}{H_2D} \quad (3)$$

$$K_{D2} = \frac{H^+ \cdot D^{2-}}{HD^-} \quad (4)$$

$$K_{P1} = \frac{H^+ \cdot H_2P^-}{H_3P} \quad (5)$$

$$K_{P2} = \frac{H^+ \cdot HP^{2-}}{H_2P^-} \quad (6)$$

$$K_{P3} = \frac{H^+ \cdot P^{3-}}{HP^{2-}} \quad (7)$$

$$K_W = H^+ \cdot OH^- \quad (8)$$

Note that the symbol *Q* for a component *Q* represents the concentration of this component, C_Q. One must pay attention to express the concentrations in the unit corresponding with the one of the equilibrium constants, usually mole l^{-1}.

When significant temperature changes occur in the system under study, the temperature will be considered as a separate state variable, besides the concentrations of the components of interest. The temperature dependency of the equilibrium constants *K* will be taken into account, typically by means of an Arrhenius equation, and their value will be calculated every time step from the prevailing temperature.

Lumped components are defined, of which the concentrations equal the total concentration of all equilibrium forms of a certain component:

$$TA = HA^+ + A \quad (9)$$

$$TB = HB + B^- \quad (10)$$

$$TD = H_2D + HD^- + D^{2-} \quad (11)$$

$$TP = H_3P + H_2P^- + HP^{2-} + P^{3-} \quad (12)$$

2.2. Set-up of the electro-neutrality equation (charge balance)

The electro-neutrality equation or charge balance in the reactor can be written as

$$\Delta_{ch} = H^+ - OH^- + HA^+ - B^- - HD^- - 2D^{2-} - H_2P^- - 2HP^{2-} - 3P^{3-} + C^+ + Z^+ \quad (13)$$

in which Δ_{ch} stands for the `gap' in the charge balance, which should be close to zero.

If multiple components of one or more type(s) are present, additional analogous terms are inserted in the charge balance (e.g. HA_1^+ ,HA_2^+) and in the subsequent calculation.

Z^+ is a lumped component that represents the concentration of net positive charges that are not influenced by the establishment of an equilibrium pH and that are not involved in any biological conversion reactions.
C^+ is a lumped component that represents the concentration of net positive charges that are not influenced by the establishment of an equilibrium pH but are involved in the biological conversion reactions. It is further assumed that all components involved in biological reactions are considered in the model, so that it is exactly known which components C^+ consists of. This is not the case for Z^+, of which the composition is not known as one usually doesn't consider components that do not effect the biological conversion reactions. Note that the concentrations of Z^+ and C^+ are negative if there are more negative than positive charges present.

The charge balance will now be rewritten in terms of the known concentrations of the lumped components, the components Z^+ and C^+ and the concentration of H^+, that needs to be calculated.

By substituting the concentrations of the lumped components (9)-(12) into the steady state expressions for equilibria (1)-(7), the concentration of every charged component can be rewritten in terms of the concentration of protons and lumped components:

$$HA^- = \frac{TA \cdot H^+}{H^+ + K_A} \equiv \frac{TA \cdot H^+}{N_A} \tag{14}$$

$$B^- = \frac{TB \cdot K_B}{H^+ + K_B} \equiv \frac{TB \cdot K_B}{N_B} \tag{15}$$

$$HD^- = \frac{TD \cdot H^+ \cdot K_{D1}}{H^{+2} + H^+ \cdot K_{D1} + K_{D1} \cdot K_{D2}} \equiv \frac{TD \cdot H^+ \cdot K_{D1}}{N_D} \tag{16}$$

$$D^{2-} = \frac{TD \cdot K_{D1} \cdot K_{D2}}{H^{+2} + H^+ \cdot K_{D1} + K_{D1} \cdot K_{D2}} \equiv \frac{TD \cdot K_{D1} \cdot K_{D2}}{N_D} \tag{17}$$

$$H_2P^- = \frac{TP \cdot H^{+2} \cdot K_{P1}}{H^{+3} + H^{+2} \cdot K_{P1} + H^+ \cdot K_{P1} \cdot K_{P2} + K_{P1} \cdot K_{P2} \cdot K_{P3}} \equiv \frac{TP \cdot H^{+2} \cdot K_{P1}}{N_P} \tag{18}$$

$$HP^{2-} = \frac{TP \cdot H^+ \cdot K_{P1} \cdot K_{P2}}{H^{+3} + H^{+2} \cdot K_{P1} + H^+ \cdot K_{P1} \cdot K_{P2} + K_{P1} \cdot K_{P2} \cdot K_{P3}} \equiv \frac{TP \cdot H^+ \cdot K_{P1} \cdot K_{P2}}{N_P} \tag{19}$$

$$P^{3-} = \frac{TP \cdot K_{P1} \cdot K_{P2} \cdot K_{P3}}{H^{+3} + H^{+2} \cdot K_{P1} + H^+ \cdot K_{P1} \cdot K_{P2} + K_{P1} \cdot K_{P2} \cdot K_{P3}} \equiv \frac{TP \cdot K_{P1} \cdot K_{P2} \cdot K_{P3}}{N_P} \tag{20}$$

After substitution of the above equations (14)-(20) and the OH^- concentration through (8), the charge balance (13) becomes

$$\begin{aligned} \Delta_{ch} = H^+ - \frac{K_W}{H^+} + \frac{TA \cdot H^+}{N_A} - \frac{TB \cdot K_B}{N_B} - \frac{TD \cdot H^+ \cdot K_{D1}}{N_D} - 2\frac{TD \cdot K_{D1} \cdot K_{D2}}{N_D} \\ - \frac{TP \cdot H^{+2} \cdot K_{P1}}{N_P} - 2\frac{TP \cdot H^+ \cdot K_{P1} \cdot K_{P2}}{N_P} - 3\frac{TP \cdot K_{P1} \cdot K_{P2} \cdot K_{P3}}{N_P} + C^+ + Z^+ \end{aligned} \tag{21}$$

The concentrations of the lumped components *TA, TB, TD, TP*, C^+ and Z^+ are calculated every time step from the corresponding mass balances. As it is not known which components Z^+ comprises, its influent concentration Z^+_{influent} and its initial concentration Z^+_{init}, that need to be known to solve the corresponding mass balance, are calculated from the charge balance calculated from the charge balances for the influent and for the initial concentrations. This is demonstrated for Z^+_{influent}:

$$Z^+_{\text{influent}} = -H^+_{\text{influent}} + \frac{K_W}{H^+_{\text{influent}}} - \frac{TA_{\text{influent}} \cdot H^+_{\text{influent}}}{N_{A\,\text{influent}}} + \frac{TB_{\text{influent}} \cdot K_{B\,\text{influent}}}{N_{B\,\text{influent}}}$$
$$+ \frac{TD_{\text{influent}} \cdot H^+_{\text{influent}} \cdot K_{D1\,\text{influent}}}{N_{D\,\text{influent}}} + 2\frac{TD_{\text{influent}} \cdot K_{D1\,\text{influent}} \cdot K_{D2\,\text{influent}}}{N_{D\,\text{influent}}}$$
$$+ \frac{TP_{\text{influent}} \cdot {H^+_{\text{influent}}}^2 \cdot K_{P1\,\text{influent}}}{N_{P\,\text{influent}}} + 2\frac{TP_{\text{influent}} \cdot H^+_{\text{influent}} \cdot K_{P1\,\text{influent}} \cdot K_{P2\,\text{influent}}}{N_{P\,\text{influent}}} \quad (22)$$
$$+ 3\frac{TP_{\text{influent}} \cdot K_{P1\,\text{influent}} \cdot K_{P2\,\text{influent}} \cdot K_{P3\,\text{influent}}}{N_{P\,\text{influent}}} - C^+_{\text{influent}}$$

The calculation of Z^+_{init} is performed completely analogously.

Note that temperature changes can cause the initial equilibrium constants and the equilibrium constants for the influent to be different from the ones in the reactor. This is not only the case when temperature effects are taken into account explicitly, i.e. temperature is a state variable, but also when the reactor temperature is assumed constant in time but not equal to the constant temperature of the reactor influent.

Once the concentrations of the lumped components *TA, TB, TD, TP*, C^+ and Z^+ have been calculated, H^+ remains the only unknown in the charge balance (21):

$$\Delta_{ch} = \Delta_{ch}(H^+) \quad (23)$$

2.3. Calculation of pH and equilibrium concentrations

The algebraic equation (23) has to be solved for the H^+ concentration for which the sum of all charges is zero:

$$\Delta_{ch}(H^+) = 0 \quad (24)$$

Since this is an implicit equation in H^+, it has to be solved iteratively. Different numerical solution methods can be applied, e.g. the Newton-Raphson method. The Newton-Raphson method starts from an initial guess H^+_0, that deviates from the actual value of H^+, for which the charge balance is fulfilled:

$$H^+ = H^+_0 + \delta H^+ \quad (25)$$

Performing a Taylor approximation of (23) around (25) yields

$$\Delta_{\text{ch}}(\text{H}^+) \approx \Delta_{\text{ch}}(\text{H}^+_0) + \left.\frac{\text{d}\Delta_{\text{ch}}(\text{H}^+)}{\text{dH}^+}\right|_{\text{H}^+_0} \cdot \left(\text{H}^+ - \text{H}^+_0\right) \quad (26)$$

To close the gap in the charge balance, H^+ is calculated from eq. (26) as

$$\text{H}^+ \approx \text{H}^+_0 - \frac{\Delta_{\text{ch}}(\text{H}^+_0)}{\left.\frac{\text{d}\Delta_{\text{ch}}(\text{H}^+)}{\text{dH}^+}\right|_{\text{H}^+_0}} \quad (27)$$

The derivative of the gap in the charge balance (21) to H^+ can be calculated analytically; one finds

$$\frac{d\Delta_{ch}(H^+)}{dH^+} = 1 + \frac{K_W}{H^{+2}} + \frac{TA \cdot K_A}{N_A{}^2} + \frac{TB \cdot K_B}{N_B{}^2} + \frac{TD \cdot K_{D1} \cdot \left(H^{+2} - K_{D1} \cdot K_{D2}\right)}{N_D{}^2} + 2\frac{TD \cdot K_{D1} \cdot K_{D2} \cdot \left(2H^+ + K_{D1}\right)}{N_D{}^2} + \frac{TP \cdot H^+ \cdot K_{P1} \cdot \left(H^{+3} - H^+ \cdot K_{P1} \cdot K_{P2} - 2K_{P1} \cdot K_{P2} \cdot K_{P3}\right)}{N_P{}^2} + 2\frac{TP \cdot K_{P1} \cdot K_{P2} \cdot \left(2H^{+3} + H^{+2} \cdot K_{P1} - K_{P1} \cdot K_{P2} \cdot K_{P3}\right)}{N_P{}^2} + 3\frac{TP \cdot K_{P1} \cdot K_{P2} \cdot K_{P3} \cdot \left(3H^{+2} + 2H^+ \cdot K_{P1} + K_{P1} \cdot K_{P2}\right)}{N_P{}^2} \quad (28)$$

considering Z^+ and C^+ are pH independent.

Equation (28) can be further simplified to

$$\frac{d\Delta_{ch}(H^+)}{dH^+} = 1 + \frac{K_W}{H^{+2}} + \frac{TA \cdot K_A}{N_A{}^2} + \frac{TB \cdot K_B}{N_B{}^2} + \frac{TD \cdot \left(H^{+2} \cdot K_{D1} + 4H^+ \cdot K_{D1} \cdot K_{D2} + K_{D1}{}^2 \cdot K_{D2}\right)}{N_D{}^2} + \frac{TP \cdot \left(\begin{array}{l} H^{+4} \cdot K_{P1} + 4H^{+3} \cdot K_{P1} \cdot K_{P2} + H^{+2} \cdot K_{P1}{}^2 \cdot K_{P2} + 9H^{+2} \cdot K_{P1} \cdot K_{P2} \cdot K_{P3} \\ \quad + 4H^+ \cdot K_{P1}{}^2 \cdot K_{P2} \cdot K_{P3} + K_{P1}{}^2 \cdot K_{P2}{}^2 \cdot K_{P3} \end{array}\right)}{N_P{}^2} \quad (29)$$

Evaluating the gap in the charge balance (21) and its derivative (28) in H_0^+ and substituting these expressions in eq. (25) yields a new estimation for H^+, noted as H_1^+:

$$H_1^+ = H_0^+ - \frac{\Delta_{ch}(H_0^+)}{\left.\frac{d\Delta_{ch}(H^+)}{dH^+}\right|_{H_0^+}}$$

Because of the Taylor series approximation, H_1^+ will not be the value of H^+ that sets $\Delta_{ch}(H^+) = 0$. Therefore, in a next iteration step, the gap in the charge balance and its derivative are evaluated for the latest estimation of the proton concentration H_1^+ .This yields another estimation for H^+, now noted as H_2^+

$$H_2^+ = H_1^+ - \frac{\Delta_{ch}(H_1^+)}{\left.\frac{d\Delta_{ch}(H^+)}{dH^+}\right|_{H_1^+}}$$

The iteration is repeated until the absolute value of the gap in the charge balance is smaller than a predefined tolerance value, *TOL*:

$$\left|\Delta_{ch}(H_i^+)\right| < \mathrm{TOL}$$

To avoid endless loops when the solution doesn't converge, the iteration is also stopped when exceeding a predefined maximum number of iteration steps.

Figure 1 gives the graphical representation of the general Newton-Raphson algorithm, solving $f(x) = 0$ for x. For each estimated value x_n of the independent variable x (in this case the H^+ concentration), the

corresponding values of the function $f(x_n)$ (in this case $\Delta_{ch}(H^+)$) and of its derivative $\left.\frac{df}{dx}\right|_{x_n}$ are calculated. The value x for which $f(x)$ becomes zero, is stepwisely approximated as $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.

The iteration procedure continues until $f(x)$ approximates zero within a predefined tolerance value.

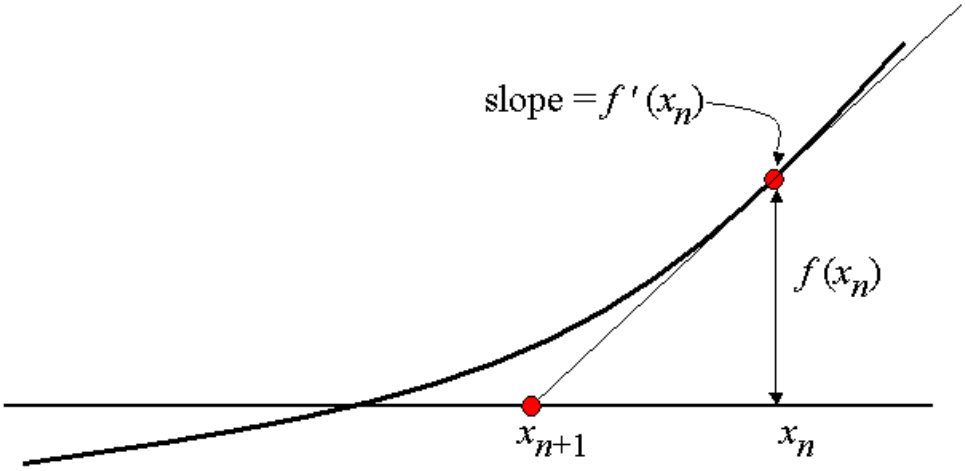


Figure 1: Graphical representation of the Newton-Raphson algorithm

Once the pH is known besides the concentrations of the lumped components, the concentration of the individual components, involved in an equilibrium, can be calculated from the expressions (14)-(20) and (9)-(13).

As stated, the above reasoning assumes the concentration of net positive charges not to vary with varying pH, assuming these net positive charges originate from strong acids/bases. In reality, weak acids or bases of which the equilibrium reactions were not considered (because they were assumed to be negligible at the prevailing nominal pH concentrations) could contribute to the concentration of these net positive charges. Also, the steady state assumption might not be appropriate for the equilibria such as the bicarbonate/carbon dioxide equilibrium, that are rather slow.
These assumptions can explain possible differences in the pH calculated by the model from acid-base equilibria and the measured pH.

The general procedure explained above can be applied to all systems that contain chemical equilibria of one of the forms (1)-(8) by selecting the appropriate terms in expressions for the charge balance (21) and its derivative (29) for biprotic and triprotic acids of which all dissociation reactions are taken into account.

For biprotic and triprotic acids one can also choose to consider only those dissociation reactions that are likely to occur in the normal pH operating range of the system. E.g. for the diprotic acid H_3Q, one could choose only to consider the second reaction:

$$H_2Q^- \leftrightarrow H^+ + HQ^{2-}$$

with equilibrium constant

$$K_{Q2} = \frac{H^+ \cdot HQ^{2-}}{H_2Q^-}$$

which means that $\mathrm{H_2Q^-}$ and $\mathrm{HQ^{2-}}$ are the only equilibrium forms considered for the component TQ:

$$TQ = H_2Q^- + HQ^{2-}$$

The corresponding terms in the charge balance and in its derivative to the proton concentration are found as:

$$\Delta_{ch} = \ldots - H_2Q - 2HQ^{2-} + \ldots$$

$$= \ldots - \frac{TQ \cdot H^+}{H^+ + K_{Q2}} - 2\frac{TQ \cdot K_{Q2}}{H^+ + K_{Q2}} + \ldots \quad (29)$$

$$\frac{d\Delta_{ch}(H^+)}{dH^+} = \ldots + \frac{TQ \cdot K_{Q2}}{\left(H^+ + K_{Q2}\right)^2} + \ldots \quad (30)$$

This procedure might be preferred over the one in which all dissocation reactions of the multiprotic acid are considered, as the calculation of very small concentrations of equilibrium forms, that are not likely to occur under the operating conditions considered, might cause numerical problems.

3. General ion recruiting procedure for pH-based calculation

In appendix F, a simple procedure is developed to numerically evaluate the total equivalents introduced by any buffer component in a solution. The same procedure numerically calculates the corresponding derivative with respect to the hydrogen ion. The procedure avoids the inclusion of complex formulae for the equivalent concentrations and derivatives. The procedure is useful for generalising the pH calculation for any pH-dependent model with reduction of the model stiffness. The procedure avoids the inclusion of “if ...then” statements to check the charge of each ion since the equivalent concentration for all ions of any buffer is calculated numerically. Excluding the conditional if statements simplified the calculation and improved the simulation speed. For instance, it was possible to extend the ADM1 WEST implementation with more buffer components without noticeable reduction in the simulation speed. The procedure is also useful to calculate total alkalinity and to simulate titration experiments for a solution of known buffer composition.

4. Applications

4.1. The SHARON model

In the SHARON model, the following biological conversion reactions are considered:

- partial nitrification of ammonium to nitrite
- further nitrification of nitrite to nitrate
- denitrification of nitrite
- denitrification of nitrate
- aerobic methanol oxidation

as well as the following chemical equilibrium reactions:

of type 1:

$$NH_4^+ \leftrightarrow H^+ + NH_3$$

of type 2:

$$HNO_2 \leftrightarrow H^+ + NO_2^-$$

of type 3:

$$H_2O + CO_2 \leftrightarrow H^+ + HCO_3^-$$

(note that this reaction includes the formation of H_2CO_3 from H_2O and CO_2 as well as the dissociation of H_2CO_3 into H^+ and HCO_3^-.)

$$HCO_3^- \leftrightarrow H^+ + CO_3^{2-}$$

of type 4 (only the second dissociation reaction is considered here!)

$$H_2PO_4^- \leftrightarrow H^+ + HPO_4^{2-}$$

besides the water equilibrium:

$$H_2O \leftrightarrow H^+ + OH^-$$

with the corresponding equilibrium constants:

$$K_{NH_4^+} = \frac{H^+ \cdot NH_3}{NH_4^+}$$

$$K_{HNO_2} = \frac{H^+ \cdot NO_2^-}{HNO_2}$$

$$K_{CO_2} = \frac{H^+ \cdot HCO_3^-}{CO_2}$$

$$K_{HCO_3^-} = \frac{H^+ \cdot CO_3^{2-}}{HCO_3^-}$$

$$K_{H_2PO_4^-} = \frac{H^+ \cdot HPO_4^{2-}}{H_2PO_4^-}$$

The following lumped components are considered:

$$TNH = NH_4^+ + NH_3$$

$$TNO2 = HNO_2 + NO_2^-$$

$$TIC = H_2CO_3 + HCO_3^- + CO_3^{2-}$$

$$TIP = H_2PO_4^- + HPO_4^{2-}$$

Nitrate is the only charged component that is involved in the biological conversion reactions but doesn't take part in chemical equilibrium reactions:

$$C^+ = -NO_3^-$$

Consequently, the charge balance in the reactor can be written as

$$\Delta_{ch} = H^+ - OH^- + NH_4^+ - NO_2^- - HCO_3^- - 2CO_3^{2-} - H_2PO_4^- - 2HPO_4^{2-} - NO_3^- + Z^+$$

or, in terms of the lumped components, the equilibrium constants and H^+ as the only unknown:

$$\Delta_{ch} = H^+ - \frac{K_W}{H^+} + \frac{TNH \cdot H^+}{H^+ + K_{NH_4^+}} - \frac{TNO2 \cdot K_{HNO_2}}{H^+ + K_{HNO_2}}$$
$$- \frac{TIC \cdot H^+ \cdot K_{CO_2}}{H^{+2} + H^+ \cdot K_{CO_2} + K_{CO_2} \cdot K_{HCO_3^-}} - 2\frac{TIC \cdot K_{CO_2} \cdot K_{HCO_3^-}}{H^{+2} + H^+ \cdot K_{CO_2} + K_{CO_2} \cdot K_{HCO_3^-}}$$
$$- \frac{TIP \cdot H^+}{H^+ + K_{H_2PO_4^-}} - 2\frac{TIP \cdot K_{H_2PO_4^-}}{H^+ + K_{H_2PO_4^-}} - NO_3^- + Z^+$$

Its derivative to H^+ is calculated analytically as

$$\frac{d\Delta_{ch}(H^+)}{dH^+} = 1 + \frac{K_W}{H^{+2}} + \frac{TNH \cdot K_{NH_4^+}}{\left(H^+ + K_{NH_4^+}\right)^2} + \frac{TNO2 \cdot K_{NO_2^-}}{\left(H^+ + K_{NO_2^-}\right)^2}$$
$$+ \frac{TIC \cdot K_{CO_2} \cdot \left(H^{+2} + 4H^+ \cdot K_{HCO_3^-} + K_{CO_2} \cdot K_{HCO_3^-}\right)}{\left(H^{+2} + H^+ \cdot K_{CO_2} + K_{CO_2} \cdot K_{HCO_3^-}\right)^2} + \frac{TIP \cdot K_{H_2PO_4^-}}{\left(H^+ + K_{H_2PO_4^-}\right)^2}$$

Every time step, the concentrations of the lumped components TNH, $TNO2$, TIC, TIP and Z^+ and the concentration of $C^+ = -NO_3^-$ are calculated from the corresponding mass balances (differential equations). Subsequently, the pH and the concentrations of the individual components NH_4^+, NH_3, HNO_2, NO_2^-, CO_2, HCO_3^-, CO_3^{2-}, $H_2PO_4^-$ and HPO_4^{2-} are calculated according to the general procedure of § 2.3 (with tolerance value TOL=10^{-12} mole/m^3 and maximum number of iteration steps = 1000).

4.1.1. Implementation in Matlab-Simulink

The procedure for calculating pH and corresponding equilibrium concentrations is implemented in Simulink by means of a c-mex S-function, of which the code is listed in Appendix A. This function has been written on the basis of the c-mex S-function template provided by Matlab R13 (sfuntmpl_basic.c, in the matlabroot/simulink/src directory), that contains skeleton implementations of all the required and optional callback routines that a c-mex-file S-function can implement.

The simulation stages in a Simulink S-function are the same as the general simulation stages in Simulink and are shown in Figure 2.

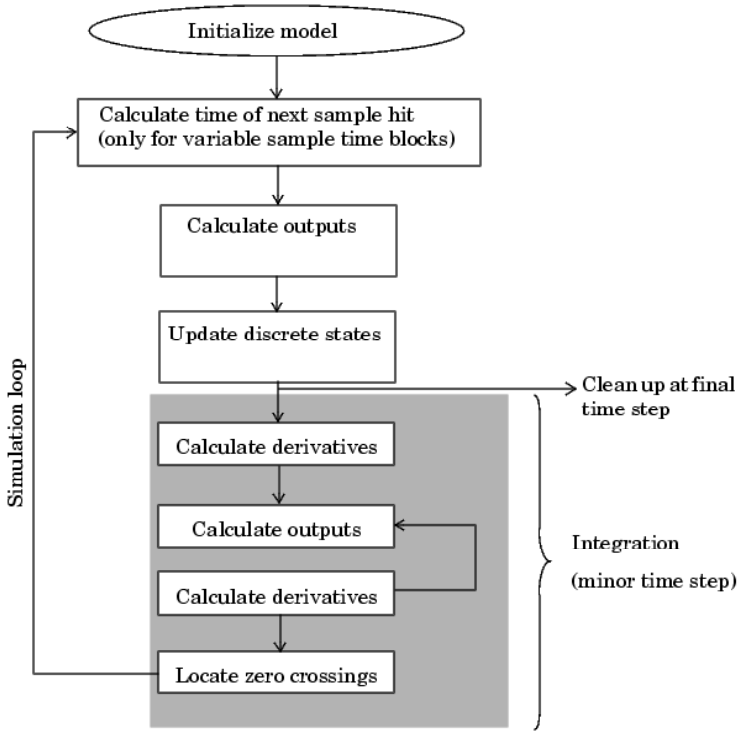


Figure 2: Simulation stages in Simulink (from Matlab R13 help)

First comes the initialization phase, in which the numbers of parameters, continuous and discrete states, input and output variables are defined (function `mdlInitializeSizes`), as well as the sample time

(`mdlInitializeSampleTimes`). In this phase, also initial values are given to the state variables (in `mdlInitializeConditions`). Then a simulation loop is entered. During each simulation step, Simulink computes the block's states (`mdlUpdate`, for discrete states), derivatives (`mdlDerivatives`, for continuous states), and outputs (`mdlOutputs`) for the current sample time. This continues until the simulation is complete.

The inputs for the S-function block for pH-calculation in Simulink are the equilibrium constants K_W, $K_{NH_4^+}$, K_{HNO_2}, K_{CO_2}, K_{HCO_3}, $K_{H_2PO_4}$, that are calculated every time step for the current temperature, and the concentrations of the lumped components TNH, $TNO2$, TIC, TIP, NO_3^- and Z^+. The block outputs are the concentrations of H^+, NH_4^+, NH_3, HNO_2, NO_2^-, CO_2, HCO_3^-, CO_3^{2-}, $H_2PO_4^-$, HPO_4^{2-} and Z^+ (all in given order). The initial estimations for the concentrations of the components H^+, NH_4^+, NH_3, HNO_2, NO_2^-, CO_2, HCO_3^-, CO_3^{2-}, $H_2PO_4^-$ and HPO_4^{2-} are passed on as block parameters and are in this way the same for every time step. All states in the S-function block for pH-calculation are discrete states, of which the values are updated through the function. This function calls for the user-defined functions `NewtonRaphson`, `Gap` and `dGapdH` in order to calculate the current block states, i.e. the pH and the equilibrium concentrations, according to the iterative Newton Raphson procedure explained above.

Table 1 gives the relationships for temperature dependency of the equilibrium constants implemented in the model. Figure 3 visualizes the equilibria by means of a buffer capacity curve for a mixture of 60 mole m^{-3} TNH, 20 mole m^{-3} TNO2, 40 mole m^{-3} TIC and 40 mole m^{-3} TIP in water.

Table 1: Chemical equilibrium coefficients

symbol	unit	expression	reference
Ke_{NH_4}	mole m^{-3}	$10^3 \cdot \exp\left(-\frac{6344}{T}\right)$	(1)
Ke_{HNO_2}	mole m^{-3}	$10^3 \cdot \exp\left(-\frac{2300}{T}\right)$	(1)
Ke_{CO_2}	mole m^{-3}	$10^3 \cdot 10^{(-356.3094-0.06091964 \cdot T+\frac{21834.37}{T}+126.8339 \cdot \log_{10} T-\frac{1684915}{T^2})}$	(2)
Ke_{HCO_3}	mole m^{-3}	$10^3 \cdot 10^{(-107.8871-0.03252849 \cdot T+\frac{5151.79}{T}+38.92561 \cdot \log_{10} T-\frac{563713.9}{T^2})}$	(2)
$Ke_{H_2PO_4^-}$	mole m^{-3}	$10^3 \cdot \left(-\frac{1979.5}{T}+5.3541-0.01984 \cdot T\right)$	(3)
K_W	mole^2 m^{-6}	$10^6 \cdot 10^{(-283.971+\frac{13323}{T}-0.05069842 \cdot T+102.24447 * \log T-\frac{1119669}{T^2})}$	(2)

(1) Anthonisen et al. (1976)
(2) Stumm and Morgan (1996)
(3) Helgeson (1967)

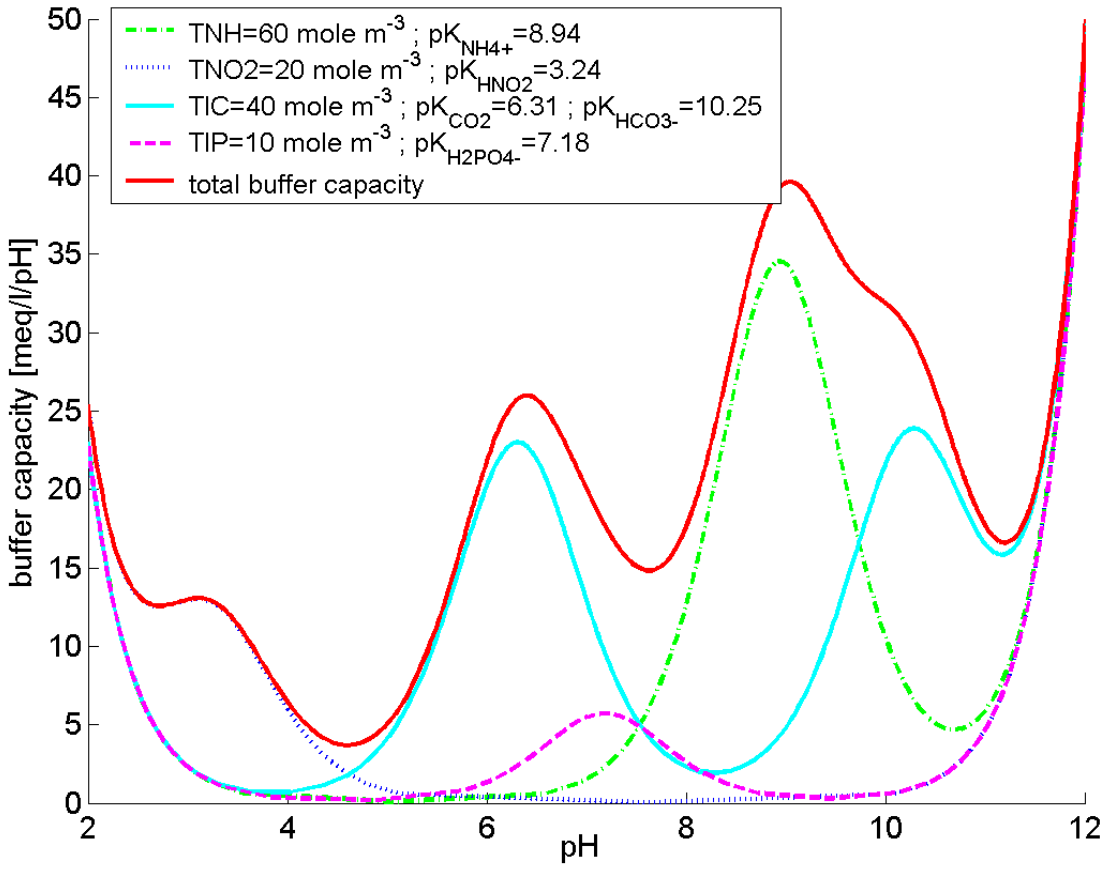


Figure 3: Buffer capacity curve for a given mixture of TNH, TNO2, TIC and TIP in water

4.1.2. Implementation in WEST

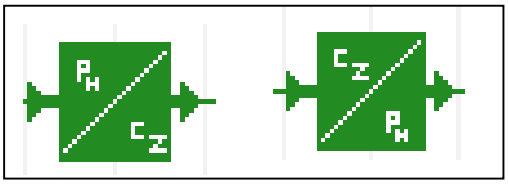
In this section the implementation of the pH calculation in WEST and the structure of the external C-function will be discussed.

In contrast to the Matlab-Simulink implementation, where each time step the same initial guess for the proton concentration is used, the WEST implementation uses the value of the proton concentration of the previous time step as an initial value.

The charge of the component Z^+ discussed above is arbitrarely taken positive, so that the concentration can become negative. This would however cause problems in WEST since the range of the TYPE Concentration is from [0,+∞]. Therefore an extra component (Z^-) is added to the ASM1.e modelbase because in WEST concentrations can not become negative. Because in WEST 2 components are introduced to model Z^+, no adaptation of the TYPE components is necessary. The two new virtual components introduced in WEST will be called S_Z_Plus and S_Z_Min. They stand for all charged (respectively positively and negatively charged) components that don't take part in any reaction and that are not influenced by the establishment of an equilibrium pH (e.g. Na^+ and Cl^-, …). The concentration of the component Z^+ discussed above is thus equal to the difference of the concentrations S_Z_Plus and S_Z_Min. Both S_Z_Plus and S_Z_Min are assumed to have a molecular weight of 1 g/mole. This necessary as all concentrations have the same unit, i.e. g/m^3

Because S_Z_Plus and S_Z_Min are considered as components in WEST, also a mass balance to the two components is applied similarly to all other components. Since S_Z_Plus and S_Z_Min do not take part in any reaction only transport is considered in this mass balance. It should also be noted that correct mass balances should be applied for all components. All processes that occur should be considered. These processes are generally transport, reaction and transfer to and from the gas phase. Especially the transfer to and from the gas phase is a process that is often forgotten, although for example stripping of CO_2 has an important influence on the pH. Therefore also the component S_IC was introduced to the ASM1.e modelbase. This component stands for the total inorganic carbon.

Because the pH and not S_Z_Plus or S_Z_Min are normally measured, a pH to S_Z and a S_Z to pH convertor is created in WEST (Figure 1). These convertors should be placed between the C to F and F to C convertors. Instead of the concentration of S_Z_Plus the pH can be specified in the inputfile. The concentration of S_Z_Min should be kept zero in the inputfile, because it has no physical meaning in the input if a pH value is specified.



4.1.2.1 New Files in WEST

Two new files were created in the WEST modelbase: wwtp.state.pH.msl and wwtp.equations.pH.msl. Both files are given in Appendix B. In the first file the state variables charge (valence of the ions/molecules), molecular_weight (reference molecular weight of the ions/molecules), equilibrium_constant1 (first equilibrium constants) and equilibrium_constant2 (second equilibrium constants) are defined. These variables are all vectors with length NrOfComponents. In the second file a value is given to these variables. Only the values for the relevant components have to be supplied.

The two new files have to be included in respectively the state and equation section of every model where pH calculation is used. Also the ConcentrationVector C and the parameter temperature have to be defined. pH calculation itself is done by calling the external C-function with the following command:

MSLUCalculatepH(state.pH_previous, ref(state.C[H2O]),ref(state.charge[H2O], ref(state.molecular_weight[H2O]), ref(state.equilibrium_constant1[H2O]), ref(state.equilibrium_constant2[H2O]), NrOfComponents)

The pH is returned from this function. E.g. for the activated sludge units the files are included in the wwtp.conversionmodel.body. In addition a C-function was created to calculate S_Z for given pH:

MSLUCalculateCz(state.pH_In,ref(state.C[H2O]), ref(state.charge[H2O]), ref(state.molecular_weight[H2O]), ref(state.equilibrium_constant1[H2O]), ref(state.equilibrium_constant2[H2O]), NrOfComponents)

The external C-function is written very general so that it can be used for every ASM, RWQM or ADM model as long as there are only mono- or bivalent ions present. However the function is easily upgradable for more valent ions. The listing of the function can be found in Appendix C.
In the main function **MSLUCalculatepH** the Newton-Raphson algorithm is implemented. First the concentrations are transformed to mole/l. Then the charge balance is calculated for the first time and it is checked whether the charge balance is already smaller than the tolerance. Then the next guess for the pH is calculated and the charge balance is checked again. This goes on until the charge balance is smaller than the tolerance or if the number of iterations is larger than 10000.
In the other functions the charge balance (**Calculatef**) and the analytical derivative (**Calculatedf**) of the charge balance are calculated. In addition to the pH calculation a function was also written to recalculate the concentrations in mole/l.

4.2. The anaerobic digestion model

In this section, the proposed Newton-Raphson pH calculation was applied for the IWA Anaerobic Digestion Model No. 1, ADM1, (Batstone et al., 2002) in two different ways in WEST. The pH calculation was generalised in WEST for the aerobic models, section 4.1.2. This generalised method was not applicable to anaerobic models since they are defined as different category than aerobic models in WEST, i.e. different components, units…etc.

Therefore, two different pH implementation in WEST are introduced in this section. The first is designed for the standard ADM1 DAE implementation illustrated in the model report (Batstone et al., 2002) that considers only monoprotic buffers. Second is another general method that is applicable for all model categories in WEST. It uses the general ion recruiting procedure and it was applied to extend the ADM1 with more buffer components.

4.2.1. pH calculation for standard ADM1 DAE implementation.

According the ADM1 report and for the proposed DAE implementation, the ion concentrations are calculated from the algebraic equations in Table 1 column (1). Using a DAE solver, these equations and equation (0.0.1) can be solved simultaneously with the differential equations pertaining to the biological reactions. When implementing the pH calculation by the algebraic equations, it is assumed that the chemical equilibrium is reached instantaneously. When using a DE solver for the model, the solution of the algebraic equations should be carefully looked after to avoid very slow simulations. WEST® uses a set of DE solvers but also allows the implementation of C++ functions to perform some external calculation that can be linked to the model. Therefore, this feature was used and a C++ function is programmed to solve the set of algebraic equations externally at each integration step of the DE solver. The idea of external calculation of pH was already applied in Matlab/Simulink for modelling the advanced nitrogen removal processes, see section 4.1. The nested solution of the model system of equations is found to be successful in improving the simulation speed.

$$S_{H^+} + S_{cat} + S_{nh4^+} - S_{an} - S_{oh^-} - S_{hco3^-} - \frac{S_{ac^-}}{64} - \frac{S_{pro^-}}{112} - \frac{S_{bu^-}}{160} - \frac{S_{va^-}}{208} = 0 \qquad (0.0.1)$$

The system of algebraic equations for pH calculation is nonlinear. The Newton-Raphson nonlinear procedure is used to solve the system of equations using the first derivative with respect to the hydrogen ion that is also listed in Table 1- column (2).

Table 1 IWA ADM1 DAE implementation functions

	(1)	(2)
Unknown Algebraic	Ion concentration functions	First derivative with respect to S_{H^+}
S_{OH^-}	$S_{OH^-} = \frac{K_w}{S_{H^+}}$	$S'_{OH^-} = -\frac{K_W}{(S_{H^+})^2}$
S_{va^-}	$S_{va^-} = \frac{K_{a,va} S_{va,total}}{K_{a,va} + S_{H^+}}$	$S'_{va^-} = -\frac{K_{a,va} S_{va,total}}{(K_{a,va} + S_{H^+})^2}$
S_{Bu^-}	$S_{Bu^-} = \frac{K_{a,bu} S_{bu,total}}{K_{a,bu} + S_{H^+}}$	$S'_{Bu^-} = -\frac{K_{a,bu} S_{bu,total}}{(K_{a,bu} + S_{H^+})^2}$
S_{Pro^-}	$S_{Pro^-} = \frac{K_{a,pro} S_{pro,total}}{K_{a,pro} + S_{H^+}}$	$S'_{Pro^-} = -\frac{K_{a,pro} S_{pro,total}}{(K_{a,pro} + S_{H^+})^2}$
S_{Ac^-}	$S_{Ac^-} = \frac{K_{a,ac} S_{ac,total}}{K_{a,ac} + S_{H^+}}$	$S'_{Ac^-} = -\frac{K_{a,ac} S_{ac,total}}{(K_{a,ac} + S_{H^+})^2}$
$S_{HCO_3^-}$	$S_{HCO_3^-} = \frac{K_{a,CO2} S_{IC,total}}{K_{a,CO2} + S_{H^+}}$	$S'_{HCO_3^-} = -\frac{K_{a,CO2} S_{IC,total}}{(K_{a,CO2} + S_{H^+})^2}$
$S_{NH_4^+}$	$S_{NH_4^+} = \frac{S_{H^+} S_{IN}}{K_{a,NH_4} + S_{H^+}}$	$S_{NH_4^+} = \frac{K_{a,NH_4} S_{IN}}{(K_{a,NH_4} + S_{H^+})^2}$

As such, the pH calculation for the standard ADM1 DAE implementation in WEST is programmed as an external C++ function, appendix D, so that it requires only the following call from the MSL code:

```
state.S_h_ion= Newton_Raphson(  parameters.Ka_nh4,
                                parameters.Ka_co2,
                                parameters.Ka_ac,
                                parameters.Ka_bu,
                                parameters.Ka_va,
                                parameters.Ka_pro,
                                parameters.Ka_h2o,
                                state.C_An[S_INN],
                                state.C_An[S_IC],
                                state.C_An[S_ac]/64,
                                state.C_An[S_bu]/160,
                                state.C_An[S_va]/208,
```

state.C_An[S_pro]/112,
state.C_An[S_cat],
state.C_An[S_an],
previous(state.S_h_ion));

4.2.2. General pH calculation using the general ion recruiting procedure.

Two new MSL files had to be added to the MSL library: “wwtp.BufferDefinitions.msl” and “wwtp.baseWithpH.msl” listed in appendix G. No further updates is needed in those two files to define the pH calculation with models. These two files defines a new class “PhysicalDAEModelType_WithpH” in the MSL library, from which models that requires pH calculations can be extended. All buffer components are defined once for all models in the WEST MSL library. Three enumerated arrays are defined in the top of the MSL tree for mono-, di-and tri-protic buffers. Therefore, only the acidity constants and concentrations are assigned in each model MSL to call the general ion recruiting procedure for the general pH calculation. For instance the ADM1 model had to be extended from the new class “PhysicalDAEModelType_WithpH” instead of the class “PhysicalDAEModelType_WithpH” . Accordingly some few updates are needed in the model MSL as shown in BOX 1. The acidity constants can be declared in the “initial” block or be defined latter in the experimentation environment. In the “equation” block, the concentration of the model buffer components are assigned to the new concentration arrays. Than the pH function call is added. The function call is standard and needn’t to be changed from one model to another.

BOX 1

```
  initial <-
{..............
parameters.K_Monoprotic[M_water][1] :=parameters.Ka_h2o/55.5;
arameters.K_Monoprotic[M_ammonia][1] :=parameters.Ka_nh4;
.........
parameters.K_Diprotic[carbon][1] := parameters.Ka_co2;
parameters.K_Diprotic[carbon][2] := 4.69e-11;
...........
parameters.K_Triprotic[T_phosphorus][1] := 7.11e-3;
parameters.K_Triprotic[T_phosphorus][2] := 6.23e-8;
parameters.K_Triprotic[T_phosphorus][3] := 4.55e-13;};


equations <-
{
state.C_Monoprotic[M_water]=55.5;
state.C_Monoprotic[M_ammonia]=-state.C_An[S_INN];
...........
state.C_Diprotic[D_carbon]=state.C_An[S_IC];
...........
state.C_Triprotic[T_phosphorus]=state.C_An[S_phos];
..............
state.pH_=pH(previous(state.S_h_ion),state.S_cat_net,
ref(state.C_Triprotic[T_phosphorus]),
ref(parameters.K_Triprotic[1][1]),NrOfTriproticBufferComponents,
ref(state.C_Diprotic[1]),
ref(parameters.K_Diprotic[1][1]),NrOfDiproticBufferComponents,
ref(state.C_Monoprotic[1]),
ref(parameters.K_Monoprotic[1][1]),NrOfMonoproticBufferComponents);
..............};
:};
```

This proposed new structure in MSL library for pH calculation, simplifies the inclusion of the pH calculation within any model. Moreover, it facilitated the implementation of a net cation (total alkalinity) calculator and titrimetric simulator. The C++ code for the pH, cation calculation and titration are all listed in the titration C++ file in appendix H.

4.3. River water quality model

In this section, the proposed pH calculation method was applied for a river water quality studies. In the River Water Quality Model No. 1 (RWQM1) (Reichert *et. al.*, 2001), chemical equilibrium between the following components are considered:

$$H_2O \xleftrightarrow{K_{eq,w}} S_{OH^-} + S_{H^+}$$

$$H_2O + S_{CO_2} \xleftrightarrow{K_{eq,1}} S_{HCO_3^-} + S_{H^+}$$

$$S_{HCO_3^-} \xleftrightarrow{K_{eq,2}} S_{CO_3^-} + S_{H^+}$$

$$S_{NH_4^+} \xleftrightarrow{K_{eq,N}} S_{NH_3} + S_{H^+}$$

$$S_{H_2PO_4^-} \xleftrightarrow{K_{eq,p}} S_{HPO_4^{2-}} + S_{H^+}$$

$$S_{CaCO_{3(s)}} \xleftrightarrow{K_{eq,s0}} S_{Ca^{2+}} + S_{CO_3^{2-}}$$

Each species in the chemical equilibrium can be calculated using equations given in Table 1. As indicate in Table 1, the concentration of $S_{HCO_3^-}$ must be calculated first using charge balance in order to proceed to quantify the others.

Table 1: Equations for the concentration of different species

Equations	Unit
$S_{H^+} = 10^{-pH}$	g H m^{-3}
$S_{OH^-} = \frac{K_{eq,w}}{S_{H^+}}$	g H m^{-3}
$S_{NH_4^+} = \frac{S_H + S_{IN}}{K_{eq,N} + S_{H^+}}$	g N m^{-3}
$S_{NH_3} = S_{IN} - S_{NH_4}$	g N m^{-3}
$S_{HPO_4} = \frac{K_{eq,p} S_H}{K_{eq,p} + S_{H^+}}$	g P m^{-3}
$S_{H_2PO_4} = \frac{S_{PO} S_H}{K_{eq,p} + S_{H^+}}$	g P m^{-3}
$S_{HCO_3^-}$ = see charge balance	g C m^{-3}
$S_{CO_3^-} = \frac{K_{eq,2} S_{HCO_3}}{S_{H^+}}$	g C m^{-3}

$S_{CO_2^-} = \frac{S_{H^+} S_{HCO_3}}{K_{eq,1}}$	g C m^{-3}

The charge balance can be calculated based on the method introduced above in section 3.1.2. The concept of S_Z_plus (Z+) and S_Z_Min(Z+) can be applied. In order to simulate the value of pH in rivers however, more data related to the most important anions and cathions need to be collected. For the dynamic simulation, the variation of these cathion concentrations influences the pH of the river waters. Naturally, there is also seasonal variation of these ions. If concentrations of the most of important anions and cathions are known, the following charge balance can be applied:

$$\frac{S_{NH_4^+}}{14} + S_{H^+} + \frac{S_{Mg^{2-}}}{24.3} + \frac{S_{K^+}}{39} + \frac{S_{Na^+}}{23} + 2\frac{S_{Ca^{2+}}}{40} + Z^+ - \frac{S_{HCO_3^-}}{12} - \frac{S_{CO_3^{2-}}}{12} - S_{OH^-} - \frac{S_{NO_2}}{14}$$
$$- \frac{S_{NO_3}}{14} - \frac{S_{H_2PO_4}}{31} - \frac{S_{HPO_4}}{31} - \frac{S_{Cl^-}}{35.45} - 2\frac{S_{SO_4^{2-}}}{96} - Z^- = 0$$

Based on equations indicated in Table 1 and charge balance, the concentration of HCO_3^- can be calculated as follows:

$$S_{HCO_3} = \frac{12}{1 + 2.K_{eq,2} / S_{H^+}} \left[\begin{array}{l} \frac{S_{NH_4^+}}{14} + S_{H^+} + \frac{S_{Mg^{2-}}}{24.3} + \frac{S_{K^+}}{39} + \frac{S_{Na^+}}{23} + 2\frac{S_{Ca}}{40} + Z^+ - S_{OH^-} - \frac{S_{NO_2}}{14} \\ - \frac{S_{NO_3}}{14} - \frac{S_{H_2PO_4}}{31} - \frac{S_{HPO_4}}{31} - \frac{S_{Cl^-}}{35.45} - 2\frac{S_{SO_4^{2-}}}{96} - Z^- \end{array} \right]$$

Based on this charge balance, the pH value can be calculated based on the external C function given in annex C.

For simplicity, the lumped components can be used. In this case the lumped components include total ammonium nitrogen as NH, total nitrate as NO and total Inorganic Carbonate as IC total Phosphate as PO as indicated below:

$$S_{NH} = S_{NH_4} + S_{NH_3}$$
$$S_{NO} = S_{NO_3} + S_{NO_2}$$
$$S_{IC} = S_{CO_3} + S_{HCO_3} + S_{H_2CO_3}$$
$$S_{PO} = S_{HPO_4} + S_{H_2PO_4}$$

This will reduce the number of components used in the model and therefore the stochiometric coefficients must be adapted to such adjustment. In the other words, this results in the model reduction. Based on the reduced model, two type of problem definition can be considered: the first problem definition is using all available data of cathions and anions. This approach also allows one to calculate the electrical conductivity of the water under consideration. The can be defined in the WEST® simulator as follows:

RealType Component =
ENUM {H2O, S_I, S_S, S_O, S_NO, S_PO, S_IC, S_NH, S_SO4, S_Mg, S_K, S_Na, S_Cl, S_Ca, S_Z_Plus, S_Z_Min, S_ALK, X_I, X_S, X_H, X_N, X_ALG, X_P, X_ND};

The complete msl code of this approach is given in the Appendix E.

In the second approach, most of the cathions are not known except S_NH. In such case the problem definition in the WEST® simulator looks as follows:

RealType Component =
ENUM {H2O, S_I, S_S, S_O, S_NO, S_PO, S_IC, S_NH, S_Z_Plus, S_Z_Min, S_ALK, X_I, X_S, X_H, X_N, X_ALG, X_P, X_ND};

Both approaches were applied in the Corocodile River case study (South Africa), and the first approach performs better than the second approach. This is due to the fact that more data is used in the first approach than in the second approach (Deksissa, 2004).

5. References

Deksissa T. (2004). Dynamic integrated modelling of basic water quality and organic contaminant fate and effect in rivers. Ph.D. thesis, Ghent University, Belgium.

Reichert P., Borchardt D., Henze M., Rauch W., Shanaahan P., Somlyody L. and Vanrolleghem PA. (2001). River water quality model no.1 (RWQM1): II. Biochemical processes equations. *Wat. Sci. Tech.,* 43(5), 11-30.

Appendix A: code of Simulink c-mex function for pH calculation

```
/*
 * pHeqdisc.c
 */

/*
 * To run this function in Simulink:
 * - create an S-function block
 *     Sfunction name: pHeqdisc
 *     Sfunction parameters: EquiInit
 * - type 'mex pHeqdisc.c' in Matlab command window to create .dll file
 *   (make sure this file is in the same directory as model)
 */

#define S_FUNCTION_NAME  pHeqdisc
#define S_FUNCTION_LEVEL 2

/*
 * Need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions.
 */
#include "simstruc.h"

#define Cinit(S) ssGetSFcnParam(S,0)


/* Error handling
 * --------------
 *
 * You should use the following technique to report errors encountered within
 * an S-function:
 *
 *       ssSetErrorStatus(S,"Error encountered due to ...");
 *       return;
 *
 * Note that the 2nd argument to ssSetErrorStatus must be persistent memory.
 * It cannot be a local variable. For example the following will cause
 * unpredictable errors:
 *
 *      mdlOutputs()
 *      {
 *        char msg[256];         {ILLEGAL: to fix use "static char msg[256];"}
 *        sprintf(msg,"Error due to %s", string);
 *        ssSetErrorStatus(S,msg);
 *        return;
 *      }
 *
 * See matlabroot/simulink/src/sfuntmpl_doc.c for more details.
```

```
 */

/*====================*
 * S-function methods *
 *====================*/

/* Function: mdlInitializeSizes ===============================================
 * Abstract:
 *    The sizes information is used by Simulink to determine the S-function
 *    block's characteristics (number of inputs, outputs, states, etc.).
 */
static void mdlInitializeSizes(SimStruct *S)
{
    /* See sfuntmpl_doc.c for more details on the macros below */

    ssSetNumSFcnParams(S, 1);  /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        /* Return if number of expected != number of actual parameters */
        return;
    }

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 10);

    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 12);              /*(S, port index, port width)*/
    /* ssSetInputPortRequiredContiguous(S, 0, false);*/
    /*Signal elements entering the specified port must occupy contiguous areas of memory */
    /* This allows a method to access the elements of the signal simply */
    /* by incrementing the signal pointer returned by ssGetInputPortSignal*/

    /*
     * Set direct feedthrough flag (1=yes, 0=no).
     * A port has direct feedthrough if the input is used in either
     * the mdlOutputs or mdlGetTimeOfNextVarHit functions.
     * See matlabroot/simulink/src/sfuntmpl_directfeed.txt.
     */
    ssSetInputPortDirectFeedThrough(S, 0, 0);

    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 10);

    ssSetNumSampleTimes(S, 1); /* ?? */

   /* ssSetNumRWork(S, 0); */
   /* ssSetNumIWork(S, 0); */
   /* ssSetNumPWork(S, 0); */
   /* ssSetNumModes(S, 0); */
   /* ssSetNumNonsampledZCs(S, 0); */
```

```
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE); /* p3 sfuntmpl_doc*/
}



/* Function: mdlInitializeSampleTimes =========================================
 * Abstract:
 *    This function is used to specify the sample time(s) for your
 *    S-function. You must register the same number of sample times as
 *    specified in ssSetNumSampleTimes.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    /* executes whenever driving block executes */
    ssSetOffsetTime(S, 0, 0.0);

}



#define MDL_INITIALIZE_CONDITIONS   /* Change to #undef to remove function */
#if defined(MDL_INITIALIZE_CONDITIONS)
  /* Function: mdlInitializeConditions ========================================
   * Abstract:
   *    In this function, you should initialize the continuous and discrete
   *    states for your S-function block.  The initial states are placed
   *    in the state vector, ssGetContStates(S) or ssGetRealDiscStates(S).
   *    You can also perform any other initialization activities that your
   *    S-function may require. Note, this routine will be called at the
   *    start of simulation and if it is present in an enabled subsystem
   *    configured to reset states, it will be call when the enabled subsystem
   *    restarts execution to reset the states.
   */
  static void mdlInitializeConditions(SimStruct *S)
  {
    real_T *x0 = ssGetDiscStates(S); /*x0 is pointer*/
          int_T  lp;
    /* get the real_T continuous state vector */
          /* can also be used in mdlstart, misschien beter*/

          for (lp=0;lp<10;lp++)
          {
    x0[lp] = mxGetPr(Cinit(S))[lp];
    }

                /* The initial conditions are passed in as the first S-function parameter */
  }
#endif /* MDL_INITIALIZE_CONDITIONS */
```

```
#undef MDL_START  /* Change to #undef to remove function */
#if defined(MDL_START)
  /* Function: mdlStart =======================================================
   * Abstract:
   *    This function is called once at start of model execution. If you
   *    have states that should be initialized once, this is the place
   *    to do it.
   */
  static void mdlStart(SimStruct *S)
  {
  }
#endif /*  MDL_START */



/* Function: mdlOutputs =====================================================
 * Abstract:
 *    In this function, you compute the outputs of your S-function
 *    block. Generally outputs are placed in the output vector, ssGetY(S).
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T            *y    = ssGetOutputPortRealSignal(S,0);
    real_T            *x    = ssGetDiscStates(S);
    int_T i;

    /* UNUSED_ARG(tid);  not used in single tasking mode */

    for (i=0; i<10; i++)
    {
                    y[i] = x[i];     /* state variables are passed on as output variables */
          }
}


/* The following functions
 * are used in MdlUpdate
 * and thus need to be defined before */

static real_T Gap(SimStruct *S)
{
    real_T            *x    = ssGetDiscStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    static real_T TNO2,TNH,TIC,TIP,NO3,Zplus;
    static real_T Kw,KeCO2,KeHCO3,KeHNO2,KeNH4,KeH2PO4;

  Kw      = *uPtrs[0];
  KeNH4   = *uPtrs[1];
```

```
  KeHNO2  = *uPtrs[2];
  KeCO2   = *uPtrs[3];
  KeHCO3  = *uPtrs[4];
  KeH2PO4 = *uPtrs[5];
  TNH     = *uPtrs[6];
  TNO2    = *uPtrs[7];
  TIC     = *uPtrs[8];
  TIP     = *uPtrs[9];
  NO3     = *uPtrs[10];
  Zplus   = *uPtrs[11];

  if (TIC<=1E-15)
  {TIC=1E-15;
  }

  x[6] = TIC/(1+KeHCO3/x[0]+x[0]/KeCO2)              ;   /* HCO3 */
  x[7] = TIC/(1+x[0]/KeHCO3+x[0]*x[0]/KeHCO3/KeCO2)   ;   /* CO3  */
  x[4] = TNO2*KeHNO2/(x[0]+KeHNO2)                   ;   /* NO2  */
  x[1] = TNH*x[0]/(x[0]+KeNH4)                       ;   /* NH4  */
  x[8] = TIP*x[0]/(KeH2PO4+x[0])                     ;   /* H2PO4 */
  x[9] = TIP-x[8]                                    ;   /* HPO4  */

    return x[0]-Kw/x[0]+x[1]-x[4]-x[6]-2*x[7]-x[8]-2*x[9]-NO3+Zplus ;
}


static real_T dGapdH(SimStruct *S)
{
    real_T           *x    = ssGetDiscStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    static real_T TNO2,TNH,TIC,TIP, NO3,Zplus;
    static real_T Kw,KeCO2,KeHCO3,KeHNO2,KeNH4,KeH2PO4;
    static real_T DNH4,DNO2,DTIC,DTIP;

  Kw      = *uPtrs[0];
  KeNH4   = *uPtrs[1];
  KeHNO2  = *uPtrs[2];
  KeCO2   = *uPtrs[3];
  KeHCO3  = *uPtrs[4];
  KeH2PO4 = *uPtrs[5];
  TNH     = *uPtrs[6];
  TNO2    = *uPtrs[7];
  TIC     = *uPtrs[8];
  TIP     = *uPtrs[9];
  NO3     = *uPtrs[10];
  Zplus   = *uPtrs[11];

  if (TIC<=1E-15)
  {TIC=1E-15;
  }
```

```
  DNH4   = x[0]+KeNH4;
  DNO2   = x[0]+KeHNO2;
  DTIC   = x[0]*x[0]+x[0]*KeCO2+KeHCO3*KeCO2;
  DTIP   = KeH2PO4+x[0];

  return 1+Kw/x[0]/x[0]
        +TNH*KeNH4/(DNH4*DNH4)
        +TNO2*KeHNO2/(DNO2*DNO2)
        +TIC*KeCO2*(x[0]*x[0]+4*x[0]*KeHCO3+KeCO2*KeHCO3)/(DTIC*DTIC)
        +TIP*KeH2PO4/(DTIP*DTIP);
  }


static void NewtonRaphson(SimStruct *S)
{
    real_T           *x    = ssGetDiscStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T    delta;
  static real_T   H0;
  static int_T      i;
  static const real_T TOL     =1E-12;
  static const real_T MaxSteps=  1000;

  H0=x[0];

  i   =1;
  delta = 1.0;
  while ( (delta>TOL || delta < -TOL) && (i<=MaxSteps) )
  {
    delta=Gap(S);
    x[0]=H0-delta/dGapdH(S);
    if (x[0]<=0)
    { x[0]=1E-12;
    }
    H0  =x[0];
    ++i;
  }
}



#define MDL_UPDATE  /* Change to #undef to remove function */
#if defined(MDL_UPDATE)
  /* Function: mdlUpdate ======================================================
   * Abstract:
   *    This function is called once for every major integration time step.
   *    Discrete states are typically updated here, but this function is useful
   *    for performing any tasks that should only take place once per
   *    integration step.
```

```
  */
 static void mdlUpdate(SimStruct *S, int_T tid)
 {
         real_T     *x    = ssGetDiscStates(S);
   InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

   x[0]=1E-12;
   NewtonRaphson(S);
   x[3]= *uPtrs[7]-x[4];      /* HNO2 */
   x[2]= *uPtrs[6]-x[1];      /* NH3 */
   x[5]= *uPtrs[8]-x[6]-x[7]; /* CO2 */
 }
#endif /* MDL_UPDATE */



#undef MDL_DERIVATIVES  /* Change to #undef to remove function */
#if defined(MDL_DERIVATIVES)
  /* Function: mdlDerivatives =============================================
   * Abstract:
   *    In this function, you compute the S-function block's derivatives.
   *    The derivatives are placed in the derivative vector, ssGetdX(S).
   */
  static void mdlDerivatives(SimStruct *S)
  {

  }
#endif /* MDL_DERIVATIVES */



/* Function: mdlTerminate =================================================
 * Abstract:
 *    In this function, you should perform any actions that are necessary
 *    at the termination of a simulation.  For example, if memory was
 *    allocated in mdlStart, this is the place to free it.
 */
static void mdlTerminate(SimStruct *S)
{
}

/*============================*
 * Required S-function trailer *
 *============================*/

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Appendix B: the additional WEST files for pH calculation

- wwtp.state.pH.msl

```
#ifdef ASM1e

// "
// The charge, molecular weight and equilibrium constants of the biological components considered in the ASM1e models
// "
OBJ charge "valence of the ions/molecules" : Real[NrOfComponents;] ;
OBJ molecular_weight "reference molecular weight of the ions/molecules " : Real[NrOfComponents;]  ;
OBJ equilibrium_constant1 "first equilibrium constants " : Real[NrOfComponents;]  ;
OBJ equilibrium_constant2 "second equilibrium constants " : Real[NrOfComponents;]  ;

#endif // ASM1e
```

- wwtp.equations.pH.msl

```
#ifdef Anammox

 /*" The biological components considered in the ASM1e models"
Charge and molecular weight should be supplied. Molecular weight of non charged components is not taken into account, so it can be set zero
 */

  state.charge[H2O]=0.0;
  state.charge[S_I]=0.0;
  state.charge[S_S]=0.0;
  state.charge[S_IC]=-2.0;
  state.charge[S_O]=0.0;
  state.charge[S_NO2]=-1.0;
  state.charge[S_NO3]=-1.0;
  state.charge[S_N2]=0.0;
  state.charge[S_NH]=1.0;
  state.charge[S_Z_Plus]=1.0;
  state.charge[S_Z_Min]=-1.0;
  state.charge[S_ALK]=0.0;

  {FOREACH Comp_Index IN {X_I .. X_ND}: state.charge[Comp_Index]=0.0  ;};

  state.molecular_weight[H2O]=18.0;
  state.molecular_weight[S_I]=0.0;
  state.molecular_weight[S_S]=0.0;
  state.molecular_weight[S_IC]=12.0;
  state.molecular_weight[S_O]=32.0;
  state.molecular_weight[S_NO2]=14.0;
  state.molecular_weight[S_NO3]=14.0;
  state.molecular_weight[S_N2]=14.0;
  state.molecular_weight[S_NH]=14.0;
  state.molecular_weight[S_Z_Plus]=1e-3;
  state.molecular_weight[S_Z_Min]=1e-3;
  state.molecular_weight[S_ALK]=0.0;
```

```
 {FOREACH Comp_Index IN {X_I .. X_ND}: state.molecular_weight[Comp_Index]=0.0 ;};

/*
        Calculation equilibrium constants. Equilibrium constants are always written
        for the reaction in which acids produce protons
        e.g. NH4+ <-> NH3 + H+
            H2O + CO2 <-> HCO3- + H+ <-> CO3-- + H+
            AH++ <-> AH+ + H+ <-> A + 2H+
            HNO2 <-> NO2- + H+
            H2A <-> H+ + A- <-> 2H+ + A--
*/
state.equilibrium_constant1[H2O] =( 2.08e-12 - 1.47e-14*(273,15+parameters.T)+2.6e-17*(273,15+parameters.T)
*(273,15+parameters.T));
state.equilibrium_constant1[S_I] = 0.0;
state.equilibrium_constant1[S_S] = 0.0;
state.equilibrium_constant1[S_IC] = (-1.02e-5+6.62e-8*(273,15+parameters.T)-1.02e-10*(273,15+parameters.T)*
(273,15+parameters.T));
state.equilibrium_constant1[S_O] = 0.0;
state.equilibrium_constant1[S_NO2] = exp(-2300/(273,15+parameters.T));
state.equilibrium_constant1[S_NO3] = 0.0;
state.equilibrium_constant1[S_N2] = 0.0;
state.equilibrium_constant1[S_NH] = exp(-6344/(273,15+parameters.T));
state.equilibrium_constant1[S_Z_Plus] = 0.0;
state.equilibrium_constant1[S_Z_Min] = 0.0;
state.equilibrium_constant1[S_ALK] = 0.0;

{FOREACH Comp_Index IN {X_I .. X_ND}: state.equilibrium_constant1[Comp_Index]=0.0  ;};

state.equilibrium_constant2[H2O] = 0.0;
state.equilibrium_constant2[S_I] = 0.0;
state.equilibrium_constant2[S_S] = 0.0;
state.equilibrium_constant2[S_IC] =(-7.97e-10 + 4.69e-12 * (273,15+parameters.T) - 6.24e-15 *(273,15+parameters.T)*
(273,15+parameters.T));
state.equilibrium_constant2[S_O] = 0.0;
state.equilibrium_constant2[S_NO2] = 0.0;
state.equilibrium_constant2[S_NO3] = 0.0;
state.equilibrium_constant2[S_N2] = 0.0;
state.equilibrium_constant2[S_NH] = 0.0;
state.equilibrium_constant2[S_Z_Plus] = 0.0;
state.equilibrium_constant2[S_Z_Min] = 0.0;
state.equilibrium_constant2[S_ALK] = 0.0;

{FOREACH Comp_Index IN {X_I .. X_ND}: state.equilibrium_constant2[Comp_Index]=0.0  ;};

#endif // ASM1e
```

Appendix C: the external C-function for pH calculation in WEST

```
/*************************************************************
 general pH calculation
 *************************************************************/

/*****************************************************************************************
In MSLUCalculateC the concentrations are internally converted from g/m3 to mol/l
 *****************************************************************************************/

RealType
MSLUCalculateC(   RealType S,
                  RealType MW)
{ RealType C;
        if (MW != 0.0 )
            {C = S/(1000.0*MW); }
                              else
                              {C = 0.0; }
  return C;
        }

/*************************************************************************************************************************
In Calculatef the charge balance is calculated. Every component has a specific term associated with it
 *************************************************************************************************************************/

RealType
Calculatef(RealType pH,
        RealVector C,
        RealVector charge,
        RealVector K1,
        RealVector K2,
        RealType no_values)

{

   RealType C_H,C_H_2,sum,f;
   IntegerType i;
   RealVector term = new RealType[no_values];

      C_H = pow(10,-pH);
      C_H_2 = C_H * C_H;
            for (i = 0; i < (IntegerType)no_values; i++)
                  {  if (charge[i] == 0.0)
                      {term[i] = 0.0;}

/*************************************************************
Negatively charged component without equilibrium
 *************************************************************/
```

```
                    if ((charge[i] < 0.0) && (K1[i] == 0.0))
                                {term[i] = C[i];}

/*************************************************************
Positively charged component without equilibrium
 *************************************************************/

                    if ((charge[i] > 0.0) && (K1[i] == 0.0))
                                {term[i] = - C[i];}

/*************************************************************
Charge = +1, with equilibrium
 *************************************************************/

                    if ((charge[i] == 1.0) && (K1[i]!=0.0))
                                {term[i] = - C[i]/(1+K1[i]/C_H);}

/*************************************************************
Charge = -1, with equilibrium
 *************************************************************/

                    if ((charge[i] == -1.0) && (K1[i]!=0.0))
                                {term[i] =  C[i]/(1+C_H/K1[i]);}

/*************************************************************
Charge = +2, with equilibrium
 *************************************************************/

                    if ((charge[i] == 2.0) && (K1[i]!=0.0) && (K2[i]!=0.0))
                                {term[i] = - C[i]/(C_H/K1[i]+1+K2[i]/C_H)
                              - 2 * C[i]/((K1[i]*K2[i])/C_H_2+K1[i]/C_H+1);}

/*************************************************************
Charge = -2, with equilibrium
 *************************************************************/

                    if ((charge[i] == -2.0) && (K1[i]!=0.0) && (K2[i]!=0.0))
                                {term[i] =  C[i]/(C_H/K1[i]+1+K2[i]/C_H)
                          + 2 * C[i]/(C_H_2/(K1[i]*K2[i])+C_H/K2[i]+1);}

          }

/*************************************************************
Sum of all the terms
 *************************************************************/

          sum =0.0;
          for (i = 0; i < (IntegerType)no_values; i++)
    { sum += term[i];  }
```

```
                        f=-C_H+K1[0]/C_H+sum;

        delete [] term;

        return f;

                                    }

/*****************************************************************************************************************
In CalculateDf the charge balance is calculated. Every component has a specific term associated with it
 *****************************************************************************************************************/

RealType
CalculateDf(RealType pH,
              RealVector C,
              RealVector charge,
              RealVector K1,
              RealVector K2,
              RealType no_values
           )

{ RealType Dsum,Df,C_H,C_H_2 ;
  RealVector Dterm = new RealType[no_values];
  IntegerType i;

           C_H = pow(10,-pH);
           C_H_2 = C_H * C_H;
                        for (i = 0; i < (IntegerType)no_values; i++)
                                    { if (charge[i] == 0.0)
                                       {Dterm[i] = 0.0;}

/*****************************************************************
Negatively charged component without equilibrium
 *****************************************************************/

                                  if ((charge[i] < 0.0) && (K1[i] == 0.0))
                                                {Dterm[i] = 0.0;}

/*****************************************************************
Positivively charged component without equilibrium
 *****************************************************************/

                                    if ((charge[i] > 0.0) && (K1[i] == 0.0))
                                      {Dterm[i] = 0.0;}

/*****************************************************************
Charge = +1, with equilibrium
 *****************************************************************/

                                    if ((charge[i] == 1.0) && (K1[i]!=0.0))
```

```
                                {Dterm[i] = - C[i]*K1[i]/pow(K1[i]+C_H,2);}

/*************************************************************
Charge = -1, with equilibrium
 *************************************************************/

                                if ((charge[i] == -1.0) && (K1[i]!=0.0))
                                  {Dterm[i] = - C[i]*K1[i]/pow(K1[i]+C_H,2);}

/*************************************************************
Charge = +2, with equilibrium
 *************************************************************/

                            if ((charge[i] == 2.0) && (K1[i]!=0.0) && (K2[i]!=0.0))
                                        {Dterm[i] =  C[i]*K1[i]*(pow(10,-2*pH)-K1[i]*K2[i])
                                                                    /pow(pow(10,-2*pH)+K1[i]*pow(10,-pH)+K1[i]*K2[i],2)
                                                - 2 *C[i]*K1[i]*(pow(10,-2*pH)+K2[i]*pow(10,-pH))
                                                /pow(pow(10,-2*pH)+K1[i]*pow(10,-pH)+K1[i]*K2[i],2);}

/*************************************************************
Charge = -2, with equilibrium
 *************************************************************/

                                if ((charge[i] == -2.0) && (K1[i]!=0.0) && (K2[i]!=0.0))
                                        {Dterm[i] = -C[i]/pow(C_H/K1[i]+1+K2[i]/C_H,2)
                                                                *(-K2[i]/(C_H_2)+1/K1[i])
                                                                -2 * C[i]/pow(1+C_H/K2[i]+C_H_2/(K1[i]*K2[i]),2)
                                                                *(1/K2[i]+2*C_H/(K1[i]*K2[i]));}
                        }

/*************************************************************
Sum of all the terms
 *************************************************************/

                        Dsum =0.0;
                        for (i = 0; i < (IntegerType)no_values; i++)
                                  { Dsum += Dterm[i];}
                                 Df=-C_H * log(10)*(-1-K1[0]/(C_H_2)+Dsum);

                         delete [] Dterm;

                        return Df;
                                  }

/*************************************************************************************************************
In CalculateCz S_Z can be calculated for given pH
 *************************************************************************************************************/
RealType
MSLUCalculateCz( RealType pHstart,
                         RealVector S,
```

```
                        RealVector charge,
                        RealVector MW,
                        RealVector K1,
                         RealVector K2,
                         RealType no_values
            )
{
RealVector C = new RealType[no_values];
RealType Cz;
IntegerType i;

for (i = 0; i < (IntegerType)no_values; i++)
                {C[i] = MSLUCalculateC(S[i],MW[i]);}

Cz = Calculatef(pHstart,C,charge,K1,K2,no_values);

delete [] C;

return Cz;

}

/*****************************************************************
 general pH calculation
 *****************************************************************/

/*In MSLUCalculatepH the pH is calculated with the use of the charge balance according
to the Newton Raphson method.
*/

RealType
MSLUCalculatepH( RealType pHstart,
                        RealVector S,
                         RealVector charge,
                        RealVector MW,
                        RealVector K1,
                        RealVector K2,
                        RealType no_values
             )
{
            RealType TOL,eps1,MAXIT,
                    pH1,pH,control,
                    f_initial,f,df,
                    f_control,pH_control, counter;
            RealVector C = new RealType[no_values];
            IntegerType i;

  TOL=1E-12;
  eps1=1E-12;
  MAXIT=10000.0;
```

```
    for (i = 0; i < (IntegerType)no_values; i++)
                {C[i] = MSLUCalculateC(S[i],MW[i]);}

if (pHstart<=eps1)
    {pHstart=eps1;}

/*
START OF NEWTON-RAPHSON
*/

/*
Calculation of the initial charge gap
*/

  f_initial=Calculatef(pHstart,C,charge,K1,K2,no_values);


  control=0.0;
  counter=0.0;
  pH=pHstart;
/*
If ABS(f)<=TOL: stop
*/

    if (fabs(f_initial)<=TOL)
      {
      pH1=pH;
      }
    else
     { while (control==0.0)
     { while (counter < MAXIT)
      {
       /*
              Calculation of the derivative of the charge gap */
       f = Calculatef(pH,C,charge,K1,K2,no_values);
       df = CalculateDf(pH,C,charge,K1,K2,no_values);

/*Calculation of next pH*/
       pH_control=pH-f/df;
       f_control= Calculatef(pH_control,C,charge,K1,K2,no_values);
       /* If ABS(f_control)<=eps1: stop
       Else retry with new pH */

                    if (fabs(f_control)<=TOL)
                          { pH1=pH_control;
                           control=1.0;
                           counter = MAXIT; }
                          else
                          {pH=pH_control;
                          teller = teller + 1.0;}}
```

```
      /*Case number of iterations is higher than MAXIT */
        pH1=pH_control;
        control=1;} }

delete [] C;
return(pH1) ;}
/*****************************************************************************************/

/*************************************************************
 End general pH calculation
 *************************************************************/
```

Appendix D: pH C++ function for the standard ADM1 DAE implementation in WEST

```
/*
// ---------------------------------------------------------------------
//  HEMMIS - Ghent University, BIOMATH
//  Implementation: Usama Zaher
//  Description: Algebraic solution for the ADM1 pH and ion concentration calculations
//  File: pH.h
// Comment: External C function header file to be copied in the same configuration build directory with the pH.CPP
// ---------------------------------------------------------------------

*/
/* ion functions' prototype */

extern "C" double NH_ion (double Ka_NH4, double S_IN, double S_H);
extern "C" double HCO_ion (double Ka_CO2, double S_IC, double S_H);
extern "C" double ac_ion (double Ka_ac, double S_ac, double S_H);
extern "C" double bu_ion (double Ka_bu, double S_bu, double S_H);
extern "C" double va_ion (double Ka_va, double S_va, double S_H);
extern "C" double pro_ion (double Ka_pro, double S_pro, double S_H);
extern "C" double OH_ion (double Kw, double S_H);

/* drivatives of ion functions' prototype */
extern "C" double NH_ion_D (double Ka_NH4, double S_IN, double S_H);
extern "C" double HCO_ion_D (double Ka_CO2, double S_IC, double S_H);
extern "C" double ac_ion_D (double Ka_ac, double S_ac, double S_H);
extern "C" double bu_ion_D (double Ka_bu, double S_bu, double S_H);
extern "C" double va_ion_D (double Ka_va, double S_va, double S_H);
extern "C" double pro_ion_D (double Ka_pro, double S_pro, double S_H);
extern "C" double OH_ion_D (double Kw, double S_H);

/*  functions prototype needed to evaluate dh for Newton_Raphson*/

extern "C" double Newton_Raphson (double Ka_NH4, double Ka_CO2,double Ka_ac,double Ka_bu,double Ka_va,double Ka_pro,double Kw,double S_IN, double S_IC, double S_ac, double S_bu, double S_va, double S_pro, double S_cat, double S_an,double H_ion_ini );
```

```
extern "C" double DeltaH (double Ka_NH4, double Ka_CO2,double Ka_ac,double Ka_bu,double Ka_va,double Ka_pro,double Kw,double S_IN, double S_IC, double S_ac, double S_bu, double S_va, double S_pro, double S_H,double S_cat, double S_an );

extern "C" double d_by_dH_DeltaH (double Ka_NH4, double Ka_CO2,double Ka_ac,double Ka_bu,double Ka_va,double Ka_pro,double Kw,double S_IN, double S_IC, double S_ac, double S_bu, double S_va, double S_pro, double S_H);

/*
// ---------------------------------------------------------------------------
//  HEMMIS - Ghent University, BIOMATH
//  Implementation: Usama Zaher
//  Description: Algebraic solution for the ADM1 pH and ion concentration calculations
//  File: pH.CPP
// Comment: External CPP function to be copied in the same configuration build directory with the pH.h  header file
// ---------------------------------------------------------------------------

*/
#include  <math.h>
#include "pH.h"

/* ion functions */
double NH_ion (double Ka_NH4, double S_IN, double S_H)
{
        return S_H * S_IN / (Ka_NH4 + S_H);
}
double HCO_ion (double Ka_CO2, double S_IC, double S_H)
{
        return Ka_CO2 * S_IC / (Ka_CO2 + S_H);
}
double ac_ion (double Ka_ac, double S_ac, double S_H)
{
        return Ka_ac * S_ac / (Ka_ac + S_H);
}
double bu_ion (double Ka_bu, double S_bu, double S_H)
{
        return Ka_bu * S_bu / (Ka_bu + S_H);
}
double va_ion (double Ka_va, double S_va, double S_H)
{
        return Ka_va * S_va / (Ka_va + S_H);
}
double pro_ion (double Ka_pro, double S_pro, double S_H)
{
        return Ka_pro * S_pro / (Ka_pro + S_H);
}
double OH_ion (double Kw, double S_H)
{
        return Kw / S_H;
}
/* drivatives of ion functions  they are needed to evaluate dh for Newton_Raphson*/
```

```
double NH_ion_D (double Ka_NH4, double S_IN, double S_H)
{
        return Ka_NH4 * S_IN / (Ka_NH4 + S_H)/ (Ka_NH4 + S_H);
}

double HCO_ion_D (double Ka_CO2, double S_IC, double S_H)
{
        return -Ka_CO2 * S_IC / (Ka_CO2 + S_H)/ (Ka_CO2 + S_H);
}
double ac_ion_D (double Ka_ac, double S_ac, double S_H)
{
        return -Ka_ac * S_ac / (Ka_ac + S_H)/ (Ka_ac + S_H);
}
double bu_ion_D (double Ka_bu, double S_bu, double S_H)
{
        return -Ka_bu * S_bu / (Ka_bu + S_H)/ (Ka_bu + S_H);
}
double va_ion_D (double Ka_va, double S_va, double S_H)
{
        return -Ka_va * S_va / (Ka_va + S_H)/ (Ka_va + S_H);
}
double pro_ion_D (double Ka_pro, double S_pro, double S_H)
{
        return -Ka_pro * S_pro / (Ka_pro + S_H)/ (Ka_pro + S_H);
}
double OH_ion_D (double Kw, double S_H)
{
        return  -Kw / S_H / S_H;
}

/*  functions prototype needed to evaluate dh for Newton_Raphson*/
double DeltaH (double Ka_NH4, double Ka_CO2,double Ka_ac,double Ka_bu,double Ka_va,double Ka_pro,double Kw,
double S_IN, double S_IC, double S_ac, double S_bu, double S_va, double S_pro, double S_H,double S_cat, double S_an )
{
return S_H + S_cat - S_an + NH_ion (Ka_NH4,S_IN,S_H) - HCO_ion(Ka_CO2,S_IC,S_H) - ac_ion(Ka_ac,S_ac,S_H) - bu_ion
(Ka_bu,S_bu,S_H) -va_ion(Ka_va,S_va,S_H) - pro_ion(Ka_pro,S_pro,S_H) - OH_ion(Kw,S_H);
}

double d_by_dH_DeltaH (double Ka_NH4, double Ka_CO2,double Ka_ac,double Ka_bu,double Ka_va,double Ka_pro,double Kw,
double S_IN, double S_IC, double S_ac, double S_bu, double S_va, double S_pro, double S_H)
{
return 1 + NH_ion_D(Ka_NH4,S_IN,S_H) - HCO_ion_D(Ka_CO2,S_IC,S_H) - ac_ion_D(Ka_ac,S_ac,S_H) -
bu_ion_D(Ka_bu,S_bu,S_H) -va_ion_D(Ka_va,S_va,S_H) - pro_ion_D(Ka_pro,S_pro,S_H) - OH_ion_D(Kw,S_H);
}

double Newton_Raphson (double Ka_NH4, double Ka_CO2,double Ka_ac,double Ka_bu,double Ka_va,double Ka_pro,double Kw,
double S_IN, double S_IC, double S_ac, double S_bu, double S_va, double S_pro, double S_cat, double S_an, double H_ion_ini )
{
        double   H0;
        double  S_H;
```

```
            double Delta;
            int     i,cont;
  const double TOL     =1E-9;
  const double MaxSteps=  100;

if (H_ion_ini<=1E-12)
    H0=1E-7;
 else
   H0= H_ion_ini;
 i   =1;
 cont=1;
 while (cont==1)
 {
 Delta=DeltaH(Ka_NH4,Ka_CO2,Ka_ac,Ka_bu,Ka_va,Ka_pro,Kw,S_IN,S_IC,S_ac,S_bu,S_va,S_pro,H0,S_cat,S_an);

 S_H= H0 -0.01 * Delta / d_by_dH_DeltaH(Ka_NH4,Ka_CO2,Ka_ac,Ka_bu,Ka_va,Ka_pro,Kw,
                                        S_IN,S_IC,S_ac,S_bu,S_va,S_pro,H0);
  H0  =S_H;
cont= (fabs(Delta)>TOL) && (i<=MaxSteps);
   ++i;
 };
 return S_H;
}
```

Appendix E: River water quality model

/**********

The pH calculation implemented in the WEST® simulator for the river water quality modelling was similar to Appendix B and C. The only difference is number and type of components considered in the model. This model is based on the biological components considered in the simplified River Water Quality Model number 1 (Reichert *et al*., 2001). This model considers H2PO4- + HPO4 = H2PO4; NH4+ + NH3 = NH4; HCO3- + CO3-- = HCO3-; and the most important cathions are also included as a component.

**********/

```
state.charge[H2O]=0.0;
state.charge[S_I]=0.0;
state.charge[S_S]=0.0;
state.charge[S_O]=0.0;
state.charge[S_NO]=0.0;
state.charge[S_PO]=-2.0;
state.charge[S_IC]=-1.0;
state.charge[S_NH]=1.0;
state.charge[S_SO4]=-2.0;
state.charge[S_Mg]=2.0;
state.charge[S_K]=1.0;
state.charge[S_Na]=1.0;
state.charge[S_Cl]=-1.0;
state.charge[S_Z_Plus]=1.0;
state.charge[S_Z_Min]=-1.0;
state.charge[S_ALK]=0.0;
```

```
{FOREACH Comp_Index IN {X_I .. X_ND}:
      state.charge[Comp_Index]=0.0  ;};


state.molecular_weight[H2O]=18.0;
state.molecular_weight[S_I]=0.0;
state.molecular_weight[S_S]=0.0;
state.molecular_weight[S_O]=32.0;
state.molecular_weight[S_NO]=14.0;
state.molecular_weight[S_PO]=31.0;
state.molecular_weight[S_IC]=12.0;
state.molecular_weight[S_NH]=14.0;
state.molecular_weight[S_SO4]=32.0;
state.molecular_weight[S_Ca]=40.0;
state.molecular_weight[S_Mg]=24.3;
state.molecular_weight[S_K]=39.1;
state.molecular_weight[S_Na]=22.99;
state.molecular_weight[S_Cl]=35.45;
state.molecular_weight[S_Z_Plus]=1e-3;
state.molecular_weight[S_Z_Min]=1e-3;
state.molecular_weight[S_ALK]=0.0;

{FOREACH Comp_Index IN {X_I .. X_ND}:
   state.molecular_weight[Comp_Index]=0.0 ;};

            /*
            Calculation equilibrium constants. Equilibrium constants are always written
            for the reaction in which acids produce protons
            e.g. H2O + CO2 <-> HCO3- + H+ <-> CO3-- + H+
                H20 <-> H+ + OH-
                HNO2 <-> NO2- + H+
                NH4+ <-> NH3 + H+
                H2PO4- <-> HPO4-- + H+
                CaCO3 <-> Ca++ + CO3--
            */
/*
Temperature correction factors for the equilibrium constants
*/

state.equilibrium_constant1[H2O]=pow(10, (-4470.99/(273.15 + state.T_w)+ 12.0875 - 0.01706*(273.15 + state.T_w)));
state.equilibrium_constant1[S_I] = 0.0;
state.equilibrium_constant1[S_S] = 0.0;
state.equilibrium_constant1[S_O] = 0.0;
state.equilibrium_constant1[S_NO] = 0.0;
state.equilibrium_constant1[S_PO]=pow(10, (-3.46 - 219.4/(273.15 + state.T_w)));
state.equilibrium_constant1[S_IC]=pow(10, (17.843 - 3404.71/(273.15 + state.T_w) - 0.0332786*(273.15 + state.T_w)));
state.equilibrium_constant1[S_NH]=pow(10, (2.891-2727/(273.15 + state.T_w)));
state.equilibrium_constant1[S_SO4]= 0.0;
state.equilibrium_constant1[S_Ca]= 0.0;
```

```
state.equilibrium_constant1[S_Mg]=0.0;
state.equilibrium_constant1[S_K]=0.0;
state.equilibrium_constant1[S_Na]=0.0;
state.equilibrium_constant1[S_Cl]=0.0;
state.equilibrium_constant1[S_Z_Plus] = 0.0;
state.equilibrium_constant1[S_Z_Min] = 0.0;
state.equilibrium_constant1[S_ALK] = 0.0;

{FOREACH Comp_Index IN {X_I .. X_ND}:
      state.equilibrium_constant1[Comp_Index]=0.0  ;};

state.equilibrium_constant2[H2O] = 0.0;
state.equilibrium_constant2[S_I] = 0.0;
state.equilibrium_constant2[S_S] = 0.0;
state.equilibrium_constant2[S_O] = 0.0;
state.equilibrium_constant2[S_NO] = 0.0;
state.equilibrium_constant2[S_PO]=0.0;
state.equilibrium_constant2[S_IC]=pow(10, (9.494 - 2902.39/(273.15 + state.T_w) - 0.02379*(273.15 + state.T_w)));
state.equilibrium_constant2[S_NH]=0.0;
state.equilibrium_constant2[S_SO4]= 0.0;
state.equilibrium_constant2[S_Ca]= 0.0;
state.equilibrium_constant2[S_Mg]=0.0;
state.equilibrium_constant2[S_K]=0.0;
state.equilibrium_constant2[S_Na]=0.0;
state.equilibrium_constant2[S_Cl]=0.0;
state.equilibrium_constant2[S_Z_Plus] = 0.0;
state.equilibrium_constant2[S_Z_Min] = 0.0;
state.equilibrium_constant2[S_ALK] = 0.0;

{FOREACH Comp_Index IN {X_I .. X_ND}:
      state.equilibrium_constant2[Comp_Index]=0.0  ;};

//################
//C02 + H2O <-> HCO3 + H+ <->  CO3 + H+
state.K_eq_1 =  state.equilibrium_constant1[S_IC];
state.K_eq_2 =  state.equilibrium_constant2[S_IC];

// H2O <-> OH- + H+
state.K_eq_w = state.equilibrium_constant1[H2O];

//  NH4 <-> NH3 + H+
state.K_eq_N = state.equilibrium_constant1[S_NH];

// H2PO4 <-> HPO4- + H+
state.K_eq_P = state.equilibrium_constant1[S_PO];

// CaCO3 <-> Ca++ + CO3--
//state.K_eq_so = state.equilibrium_constant1[S_Ca];
```

```
state.pH_previous= previous(state.pHRM);


state.pHRM =
                    MSLUCalculatepH(state.pH_previous,
                    ref(state.C[H2O]),ref(state.charge[H2O]),
                    ref(state.molecular_weight[H2O]),
                    ref(state.equilibrium_constant1[H2O]),
                    ref(state.equilibrium_constant2[H2O]),
                NrOfComponents);

state.S_H = 1000*pow(10, -(state.pHRM));
state.S_CO3 = state.K_eq_2*state.S_HCO3/state.S_H;
```

Appendix F: General ion recruiting procedure

Environmental Modelling and Software (submitted)

General ion recruiting procedure for pH-based calculations

U. Zaher* and P.A. Vanrolleghem

BIOMATH - Department of Applied Mathematics, Biometrics and Process Control, Ghent University, Coupure Links 653, 9000 Gent, Belgium:
usama.zaher@biomath.ugent.be, peter.vanrolleghem@ugent.be
Phone: +32 (9) 264.59.32 Fax: +32 (9) 264.62.20

Abstract

In this paper, a simple procedure is developed to numerically evaluate the total equivalents introduced by any buffer component in a solution. The same procedure calculates the corresponding derivative with respect to the hydrogen ion. The procedure is useful for generalising the pH calculation for any pH-dependent model with reduction of the model stiffness. It is also found to be useful to calculate total alkalinity and to simulate titration experiments for a solution with known buffer composition.

Keywords: Cations estimation; equivalent concentration; pH; mathematical modelling; simulation; titration

1. Introduction

There are many pH-dependent models and applications. Examples of the International Water Association (IWA) models that consider chemical equilibrium to calculate the pH are the River Water Quality Model No.1, RWQM1, (Reichert et al., 2001) and the Anaerobic Digestion Model no1, ADM1 (Batstone et al., 2002). Also, IWA's Activated Sludge Models: ASM1, ASM2, ASM2d and ASM3 (Henze et al., 2000) deal with proton consumption and alkalinity. Therefore, pH and alkalinity calculations are evident extensions for these models. However, solving the chemical equilibrium within these models as a set of differential equations that describe the equilibrium reactions introduces stiffness since the chemical reactions are significantly faster compared to the

biological reactions. As a result, the simulation will be slow and a stiff solver is needed, which may not be always available. Therefore, it is advantageous to solve the chemical equilibrium directly and not by solving the corresponding differential equations to determine the pH. Solving the chemical equilibrium asks for the solution of the charge balance of the system which involves a set of implicit nonlinear equations by a search routine, such as Newton-Raphson procedure. In this paper, this problem is tackled and the external pH calculation is generalised for any combination of buffer systems. The general procedure is therefore suitable for application with any model. Moreover, it is applicable to the calculation of the total alkalinity (net cation concentration) and the simulation of titration experiments.

2. Procedure

The structure of the general pH function consists of three functions that work together in the hierarchy depicted in Figure 1.

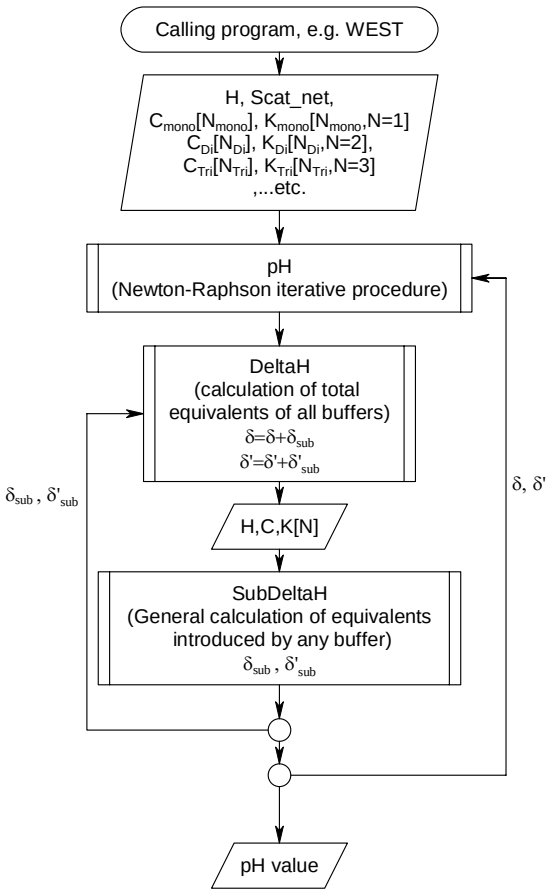


Figure 1 General ion recruiting procedure for pH calculation

The innovation in this paper is that the function SubDeltaH at the bottom of the hierarchy has a general form that calculates the equivalent concentration (δ_{sub}) introduced by a buffer component regardless of its type being mono-, di-, triprotic or even higher in case of organic compounds. Also, the function has a general form that allows to calculate the derivative of the equivalent concentration that is needed in the nonlinear solution of the charge balance using the Newton-Raphson iterative procedure. In bottom-up direction according to Figure 1, the development of the general procedure is illustrated in the following sections.

2.1. Algorithm

The equivalent concentration introduced by mono-, di- and triprotic buffers is evaluated symbolically as function of total buffer concentration C_T, acidity constants K_i and the hydrogen ion concentration H, equations (2), (3) and (4). The symbolic evaluation shows that the δ_{sub} is evolving systematically with the number of ions N.

$$\delta_{sub,N=1} = \frac{K_1 \cdot C_T}{H + K_1} \tag{2}$$

$$\delta_{sub,N=2} = \frac{K_1 \cdot C_T \cdot H + 2K_1 \cdot K_2 \cdot C_T}{H^2 + K_1 \cdot H + K_1 \cdot K_2} \tag{3}$$

$$\delta_{sub,N=3} = \frac{K_1 C_T H^2 + 2K_1 K_2 C_T H + 3K_1 K_2 K_3 C_T}{H^3 + K_1 H^2 + K_1 K_2 H + K_1 K_2 K_3} \tag{4}$$

Similarly, the derivative with respect to H, δ'_{sub}, is systematic with the number of ions N according equations (5), (6) and (7).

$$\delta'_{sub,N=1} = -\frac{K_1 C_T}{(H + K_1)^2} \tag{5}$$

$$\delta'_{sub,N=2} = \frac{K_1 C_T}{H^2 + K_1 H + K_1 K_2} - \frac{(K_1 C_T H + 2K_1 K_2 C_T)(2H + K_1)}{(H^2 + K_1 H + K_1 K_2)^2} \tag{6}$$

$$\delta'_{sub,N=3} = \frac{(2K_1 C_T H + 2K_1 K_2 C_T)}{H^3 + K_1 H^2 + K_1 K_2 H + K_1 K_2 K_3} - \frac{(K_1 C_T H^2 + 2K_1 K_2 C_T H + 3K_1 K_2 K_3 C_T)(3H^2 + 2K_1 H + K_1 K_2)}{(H^3 + K_1 H^2 + K_1 K_2 H + K_1 K_2 K_3)^2} \tag{7}$$

2.2. Ion recruiting

The complexity of δ_{sub} and δ'_{sub} functions increases with the increase of the number N of ions introduced by a buffer component. Moreover, for organic buffer components that have more than 3 dissociation steps the functions will be more complex and difficult to evaluate symbolically. The general ion recruiting procedure presented in Figure 2 evaluates the δ_{sub} and δ'_{sub} numerically and avoids the inclusion of complex equations in the pH calculation. Also, with its generality it avoids the inclusion of conditional if statements that would have been needed to select the appropriate formula for each buffer. Accordingly, the implementation of this procedure significantly improves the calculation speed and simplifies the programming code.

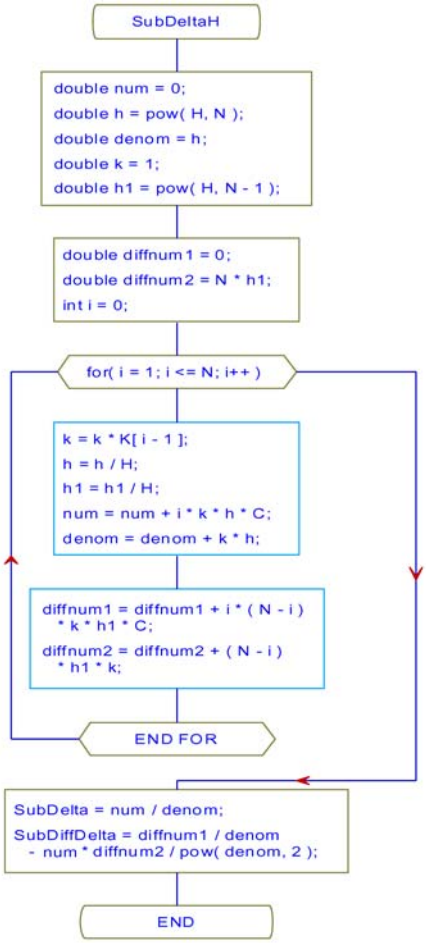


Figure 2 General procedure of ion recruiting for calculation of total equivalents and derivatives effected by a buffer component

For a buffer B that have N dissociation reactions, the input to the procedure in Figure 2 is the hydrogen ion concentration H (mol/l), the total concentration of B (mol/l) and an array of the B acidity constants $K[i]$ of a dimension N. The procedure iterates N iterations to calculate the terms

num, *denom*, *diffnum1* and *diffnum2*. Then the δ_{sub} and δ'_{sub} are calculated for the buffer by rearranging those terms according equation (8) and (9).

$$\delta_{sub,B_j} = \frac{\sum_{i=1}^{N} num_{i,B_j}}{denom_{B_j}} \tag{8}$$

$$\delta'_{sub,B_j} = \frac{\sum_{i=1}^{N} diffnum1_{i,B_j}}{denom_{B_j}} - \frac{\sum_{i=1}^{N} num_{i,B_j} \cdot diffnum2_{i,B_j}}{\left(denom_{B_j}\right)^2} \tag{9}$$

2.3. Calculation of total equivalents

Referring to Figure 1 the function DeltaH iterates to pass buffer components one by one to SubDeltaH and calculates the cumulative sum of the total equivalents introduced by all buffer components. Some important aspects need to be considered for the general application of the procedure. The net cation (ions of strong bases minus ions of strong acids) concentration is known and passed with the DeltaH arguments so that it will be added to the δ-value but not to its derivative, the derivative of the cation concentration is zero. For consideration of OH^-, the water buffer is passed to the DeltaH function as a mono-protic buffer with acidity constant $K_w^{\dagger} = K_w / [H_2O]$ and $[H_2O] = 55.5$ mol/l. Weak bases (e.g. ammonia can also be considered to calculate the net equivalent sum. This can be achieved without changing the SubDeltaH function by passing on the total concentration of the weak base with a negative sign and with its base constant K_b rather than the acidity constant K_a. Note that K_b . K_a = K_w.

3. Application

3.1. pH calculation

The calculated δ and δ' are used to calculate the pH that maintains the charge balance of the buffer mixture. When the buffer components in the solution are known and the pH is unknown, the Newton-Raphson procedure, Figure 1, iterates to correct a previous *H* value according equation

(10) till an accepted tolerance is reached, i.e. till δ/δ' is not significant any more. Accordingly the pH is calculated as the $-\log_{10} H$.

$$H_i = H_{i-1} - \frac{\delta}{\delta'} \tag{10}$$

This procedure was successfully implemented to simulate the pH in a lab-CSTR digester (Zaher et al., 2004), using the ADM1 model. The results agreed with the pH measurement. Applying this procedure for pH simulation reduced the model stiffness and improved the simulation speed. Also, with this improvement it was possible to simulate the digester pH in a plant-wide model (Zaher et al. 2005) using ADM1 to model the anaerobic digester and ASM1 to model the activated sludge plant. The procedure proved its applicability to any pH-dependent model to reduce its stiffness and improve its simulation speed. Indeed, the Newton-Raphson solution for pH was also successful in other different implementations for modelling the advanced nitrogen removal process SHARON-Anammox (Hellinga et al., 1999; Volcke et al., 2002).

3.2. Total alkalinity calculation

If the pH and the buffer composition are known, the procedure can be implemented to calculate the total alkalinity Z, equation (11).

$$Z = \delta - H \tag{11}$$

The procedure was used to calculate the influent net cation concentration in an upflow anaerobic sludge bed reactor with known influent pH and concentrations of phosphorous, acetate, bicarbonate and cyanide buffers (Zaher et al. 2005).

3.3. Titration simulation

The same procedure can also be implemented to simulate titration curve. At different pH steps the acid volume V_a that needs to be added to shift the equilibrium at each pH can be calculated according to equation (12).

$$V_a = \frac{V_s}{N_a - H} \cdot (H - \delta) \tag{12}$$

where V_s is the sample volume and N_a is the acid Normality.

The procedure was applied to simulate titration experiments of mono-, di- and triprotic buffers, Figure 3. The figure shows the nice fit of the simulation to the experimental data titrating different acetate and lactate mixtures. Also, accurate simulations are obtained for bicarbonate and phosphorus mixtures.

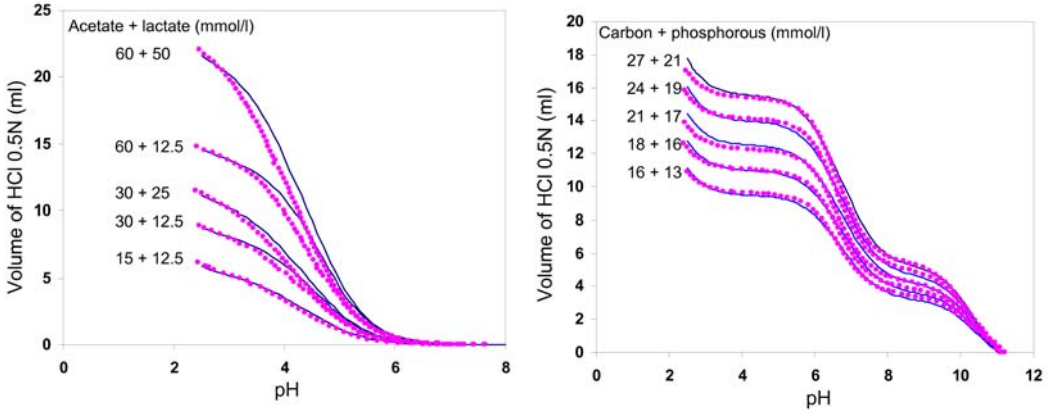


Figure 3 application of the procedure to simulate titration experiments Conclusions

4. Conclusion

The systematic evolution of the equivalent concentrations (δ_{sub}) and the corresponding *H*-derivatives (δ'_{sub}) with the buffer types and the number of their ions was implemented in a general procedure "SubDeltaH". A higher level function "DeltaH" can arrange the properties of any group of buffers (including the water buffer) and iterate calling "SubDeltaH" to calculate the total equivalents (δ) and *H*-derivative (δ'). This generalisation was useful to simplify different calculation and simulation procedures that are dependent on the charge balance and pH. It reduces the stiffness of pH-dependent models, thus improving the simulation speed. It was used to calculate the net cation concentration (total alkalinity) of a sample given a known buffer composition and pH. Also, it is useful for the simulation of titration experiments.

References

Batstone D.J., Keller J., Angelidaki R.I., Kalyuzhnyi S.V., Pavlostathis S.G., Rozzi A., Sanders W.T.M., Siegrist H. and Vavilin V.A. 2002. Anaerobic Digestion Model no1, IWA publishing, London, UK, pp.77.

Hellinga C., van Loosdrecht M.C.M. and Heijnen J.J. 1999. Model based design of a novel process for nitrogen removal from concentrated flows. Mathematical and Computer Modelling of Dynamical Systems, 5, 351-371.

Henze M., Gujer W., Mino T., and van Loosdrecht M.C.M 2000. Activated Sludge Models: ASM1, ASM2, ASM2d and ASM3. Scientific and Technical Report No:9, IWA Publishing, London, UK.

Reichert P., Borchardt D., Henze M., Rauch W., Shanahan P., Somlyódy L. and Vanrolleghem P.A. 2001. River Water Quality Model No.1, Scientific and Technical Report No. 12, IWA Publishing, London, UK.

Volcke E.I.P., Hellinga C., Van Den Broeck S., van Loosdrecht M.C.M. and Vanrolleghem P.A. 2002. Modelling the SHARON process in view of coupling with Anammox. In: Proceedings 1st IFAC International Scientific and Technical Conference on Technology, Automation and Control of Wastewater and Drinking Water Systems (TiASWiK'02). Gdansk-Sobieszewo, Poland, June 19-21, 2002, 65-72.

Zaher U., Grau P., Benedetti L., Ayesa E. and Vanrolleghem P.A. 2005. Transformers for interfacing anaerobic digestion models to pre- and post-treatment processes, Environmental Modelling & Software. (Submitted)

Zaher U., Rodríguez J., Franco A. and Vanrolleghem P.A. 2004. Conceptual approach for ADM1 application. Water and Environment Management Series (WEMS), 249-258.

Zaher U., Widyatmika I.N., Moussa M.S., van der Steen P., Gijzen H.J. and Vanrolleghem P.A. 2005. Update of the IWA ADM1 for modelling anaerobic digestion of cyanide-contaminated wastewater, Bioresource Technology. (Submitted)

Appendix G: New MSL standard files for the pH calculation using the ion recruiting procedure

```
// ---------------------------------------------------------------------
//  HEMMIS - Ghent University, BIOMATH
//  Implementation:  by Usama Zaher, Peter Vanrolleghem
//  Update for Buffer definitions for General pH calculation, alkalinity calculation and titration simulation
//  Description: MSL-USER/WWTP/Buffer definitions
//               extending generic modules, category definitions and
//               vector (matrix) definitions.
// ---------------------------------------------------------------------

#ifndef WWTP_BufferDefinitions
#define WWTP_BufferDefinitions

//
// Declarations of the classes used to define Buffer systems

//          for parameters

// pka values at 298k and corresponding Acidity
/*
CLASS pKa
   "A class for pKa"
   SPECIALISES PhysicalQuantityType :=
   {:
    quantity  <- "pKa";
    interval  <- {: lowerBound <- 0; upperBound <- 14 :};
   :};

*/

//=================================================================
//=========================Begin of components def.=====
//=================================================================

TYPE MonoproticBufferComponents
  "Monoprotic buffer components"
  = ENUM { M_water, M_ammonia, M_phenol, M_sulphide, M_VFA, M_acetate , M_butyrate , M_propionate,
M_valerate, M_bicarbonate, M_lactic, M_nitrite, M_cyanide};

TYPE DiproticBufferComponents
  "Dinoprotic buffer components"
  = ENUM { D_carbon};

TYPE TriproticBufferComponents
  "Trinoprotic buffer components"
  = ENUM { T_phosphorus, T};
```

```
TYPE BufferComponents
  "The biological components considered in the ADM1 model"
  = ENUM { water, ammonia, phenol, sulphide, VFA, acetate, butyrate, propionate, valerate, bicarbonate,
lactic, nitrite, cyanide, carbon, phosphorus};

OBJ NrOfMonoproticBufferComponents "The number of monoprotic buffer components"
  : Integer := Cardinality(MonoproticBufferComponents);

OBJ NrOfDiproticBufferComponents "The number of diprotic buffer components"
  : Integer := Cardinality(DiproticBufferComponents);

OBJ NrOfTriproticBufferComponents "The number of triprotic buffer components"
  : Integer := Cardinality(TriproticBufferComponents);

OBJ NrOfBufferComponents "The number of buffer components"
  : Integer := Cardinality(BufferComponents);

CLASS BufferConcentration
  "
   All buffer variables in concentrations
  "
  = Real[NrOfBufferComponents;];

CLASS MonoproticBufferConcentration
  "
   Monoprotic buffer variables in concentrations
  "
    = Real[NrOfMonoproticBufferComponents;];

CLASS DiproticBufferConcentration
  "
   Diprotic buffer variables in concentrations
  "
    = Real[NrOfDiproticBufferComponents;];

CLASS TriproticBufferConcentration
  "
  Triprotic buffer variables in concentrations
  "
  = Real[NrOfTriproticBufferComponents;];

CLASS InBuffer SPECIALISES BufferConcentration; //used to indicate inflow concentrations
CLASS OutBuffer SPECIALISES BufferConcentration; //used to indicate outflow concentrations
/* additional defintion for cation calculator*/
TYPE Measurement
  "pH measurement"
  = ENUM {pH_m};

OBJ NrOfMeasured "The number of measured parameters"
```

```
 : Integer := Cardinality(Measurement);

 CLASS measuredClass
   "
    Diprotic buffer variables in concentrations
   "
   = Real[NrOfMeasured;];

CLASS InpH SPECIALISES measuredClass; //used to indicate inflow concentrations

// End of Vector Classes
#endif // BufferDefinitions
 //=====================================================================
//=========================End of components def.=====
//=====================================================================


// ---------------------------------------------------------------------
//  HEMMIS - Ghent University, BIOMATH
//  Implementation of PhysicalDAEModelType_WithpH : Usama Zaher
//
//  Description: MSL-USER/WWTP/BaseWithpH
// ---------------------------------------------------------------------
#ifndef WWTP_BASEWITHPH
#define WWTP_BASEWITHPH

CLASS PhysicalDAEModelType_WithpH

//
// Literature :
//
SPECIALISES PhysicalDAEModelType :=
{:
   parameters <-
  {
 /*parameters for pH calculation*************************************************************************/

OBJ K_Triprotic  "Acid / base constants for triprotic buffers": Real[NrOfTriproticBufferComponents;][3;];
OBJ K_Diprotic  "Acid / base constants for diprotic buffers": Real[NrOfDiproticBufferComponents;][2;];
OBJ K_Monoprotic "Acid / base constants for monotic buffers":
Real[NrOfMonoproticBufferComponents;][1;];

  /*************************************************************************************************/
};

state <-
{   /* for pH calculation ***********************************************************************/

   OBJ C_Triprotic "Acid / base constants for triprotic buffers":TriproticBufferConcentration ;
   OBJ C_Diprotic "Acid / base constants for diprotic buffers": DiproticBufferConcentration;
```

```
    OBJ C_Monoprotic "Acid / base constants for monotic buffers":MonoproticBufferConcentration ;
    OBJ S_cat_net "cations of strong bases - anions of strong acids":Real ;

    /*****************************************************************************************/
   };
:};

#endif // WWTP_BASEWITHPH
```

Appendix H: C++ standard titration files for the pH and cation calculation and titration simulation using the ion recruiting procedure

```
/*
// ---------------------------------------------------------------------
//  HEMMIS - Ghent University, BIOMATH
//  Implementation: Usama Zaher
//  Description: Algebraic solution for the ADM1 pH and ion concentration calculations
//  File: Titration.h
// Comment: External C function header file to be copied in the same configuration build directory with the pH.CPP
// ---------------------------------------------------------------------

*/

/* ion functions' prototype */

 double Titration(double t, int tit_Alg, double pH_start, double pH_end, double pH_step,
                                        double Acid_N, double base_N, double Sample_Vol,

                                        double* C_Triprotic,
                                        double* K_Triprotic,
                                        int NrOfTriproticBufferComponents,
                                        double* C_Diprotic,
                                        double* K_Diprotic,
                                        int NrOfDiproticBufferComponents,
                                        double* C_Monoprotic,
                                        double* K_Monoprotic,
                                        int NrOfMonoproticBufferComponents);

extern "C" double pH (double H,
                                    double Scat_net,
                                        double* C_Triprotic,
                                        double* K_Triprotic,
                                        int NrOfTriproticBufferComponents,
                                        double* C_Diprotic,
                                        double* K_Diprotic,
                                        int NrOfDiproticBufferComponents,
                                        double* C_Monoprotic,
                                        double* K_Monoprotic,
                                        int NrOfMonoproticBufferComponents);

 void DeltaH(double &Delta, double &DiffDelta,
                                    double H,
                                    double &Scat_net,
                                        double* C_Triprotic,
                                        double* K_Triprotic,
                                        int NrOfTriproticBufferComponents,
                                        double* C_Diprotic,
```

```
                                    double* K_Diprotic,
                                    int NrOfDiproticBufferComponents,
                                    double* C_Monoprotic,
                                    double* K_Monoprotic,
                                    int NrOfMonoproticBufferComponents);


void SubDeltaH(double &SubDelta, double &SubDiffDelta,
                                double H,
                                double C,
                                double K[],
                                int N);

/*
// -------------------------------------------------------------------
//  Implementation: Usama Zaher
//  Description: Algebraic solution for the ADM1 pH and ion concentration calculations
//  File: Titration.cpp
// Comment: External C function header file to be copied in the same configuration build directory with the
titration.h
// -------------------------------------------------------------------

*/
#include  <math.h>
#include <fstream>
#include <string>
#include <iostream>
#include <stdlib.h>
#include <stdio.h>

#include "Types.h"
//#include "MSLU.h"
#include "Titration.h"
/* ion functions */
using namespace std;

double Titration(double t, int tit_Alg, double pH_start, double pH_end, double pH_step,
                                    double Acid_N, double base_N, double Sample_Vol,

                                    double* C_Triprotic,
                                    double* K_Triprotic,
                                    int NrOfTriproticBufferComponents,
                                    double* C_Diprotic,
                                    double* K_Diprotic,
                                    int NrOfDiproticBufferComponents,
                                    double* C_Monoprotic,
                                    double* K_Monoprotic,
                                    int NrOfMonoproticBufferComponents)
{
double Scat_net=0;
```

```
double Delta=0;
double DiffDelta=0;
double Va=0;
 int  decimal, sign;
 char *tt;
char cmd[]="copy tit_curve.txt ";
int N= fabs((pH_start-pH_end)/pH_step);//estimated number of titration points
double H= -pow(10,pH_start);

ofstream out("tit_curve.txt",ios::app);


DeltaH(Delta, DiffDelta,
                                H,
                                Scat_net,
                                C_Triprotic,
                                K_Triprotic,
                                NrOfTriproticBufferComponents,
                                C_Diprotic,
                                K_Diprotic,
                                NrOfDiproticBufferComponents,
                                C_Monoprotic,
                                K_Monoprotic,
                                NrOfMonoproticBufferComponents);
Scat_net=Delta-H;

 switch (tit_Alg)
  {
  case 0:// model based

   if( pH_end < pH_start)
{//down titration i.e. step wise acid addition
out <<  Va*1000 << "\t" << pH_start << endl;
             for( double pH=pH_start-pH_step;pH < pH_end; pH=pH-pH_step)
             {
                          H= -pow(10,pH);
                        DeltaH(Delta, DiffDelta,
                                H,
                                Scat_net,
                                C_Triprotic,
                                K_Triprotic,
                                NrOfTriproticBufferComponents,
                                C_Diprotic,
                                K_Diprotic,
                                NrOfDiproticBufferComponents,
                                C_Monoprotic,
                                K_Monoprotic,
                                NrOfMonoproticBufferComponents);
                       Va=Sample_Vol/(Acid_N-H)*(H+Scat_net-Delta);
                     out <<  Va*1000 << "\t" << pH << endl;
```

```
            }
out.close();

// list titration curve in a text file at the sampled time step
   tt = _fcvt( t, 3, &decimal, &sign );

string NewTitFileName(tt);
NewTitFileName.insert(decimal, ".");
strcat( cmd,NewTitFileName.c_str());
system(cmd);
system("del tit_curve.txt");


}
else
{ //up titration i.e. step wise base addition
          out <<  Va*1000 << "\t" << pH_start << endl;
            for( double pH=pH_end+pH_step;pH > pH_end; pH=pH+pH_step)
            {
                     H= -pow(10,pH);
                    DeltaH(Delta, DiffDelta,
                            H,
                            Scat_net,
                            C_Triprotic,
                            K_Triprotic,
                            NrOfTriproticBufferComponents,
                            C_Diprotic,
                            K_Diprotic,
                            NrOfDiproticBufferComponents,
                            C_Monoprotic,
                            K_Monoprotic,
                            NrOfMonoproticBufferComponents);
                   Va=Sample_Vol/(Acid_N-H)*(H+Scat_net-Delta);
                 out <<  Va*1000 << "\t" << pH << endl;

            }
out.close();

// list titration curve in a text file at the sampled time step

   tt = _fcvt( t, 3, &decimal, &sign );
string NewTitFileName(tt);
NewTitFileName.insert(decimal, ".");
strcat( cmd,NewTitFileName.c_str());
system(cmd);
system("del tit_curve.txt");
}
    break;
  case 1:
```

```
   // Data based
   break;
 default:
   ;//
 }

return 0.0000001;
}


double pH (        double H,
                        double Scat_net,
                                double* C_Triprotic,
                                double* K_Triprotic,
                                int NrOfTriproticBufferComponents,
                                double* C_Diprotic,
                                double* K_Diprotic,
                                int NrOfDiproticBufferComponents,
                                double* C_Monoprotic,
                                double* K_Monoprotic,
                                int NrOfMonoproticBufferComponents)
{
          double   H0;
          double  S_H;
          double Delta=0;
   double DiffDelta=0;
   int      i,cont;

 const double TOL     =1E-9;
 const double MaxSteps=  100;

if (H<=1E-12)
    H0=1E-7;
 else
  H0= H;

 i   =1;
 cont=1;

 while (cont==1)
 {
           DeltaH(Delta, DiffDelta,
                        H,
                        Scat_net,
                        C_Triprotic,
                        K_Triprotic,
                        NrOfTriproticBufferComponents,
                        C_Diprotic,
                        K_Diprotic,
                        NrOfDiproticBufferComponents,
```

```
                                C_Monoprotic,
                                K_Monoprotic,
                                NrOfMonoproticBufferComponents);

   S_H=H0-0.01 * Delta / DiffDelta;

   if (S_H<=1E-20)
   { S_H=1E-12;
   }

   H0  =S_H;
         cont= (fabs(Delta)>TOL) && (i<=MaxSteps);
   ++i;
 };
 return -log10(S_H);
}


void DeltaH(double &Delta, double &DiffDelta,
                                double H,
                                double &Scat_net,
                                        double* C_Triprotic,
                                        double* K_Triprotic,
                                        int NrOfTriproticBufferComponents,
                                        double* C_Diprotic,
                                        double* K_Diprotic,
                                        int NrOfDiproticBufferComponents,
                                        double* C_Monoprotic,
                                        double* K_Monoprotic,
                                        int NrOfMonoproticBufferComponents)
{
         double SubDelta;
         double SubDiffDelta;
         int i=0;
         int j=0;
         int l=0;
         double C=0;
         double KT[3];
         double KD[2];
         double KM[1];
         double Kb[1];//assuming that buffering bases are only monoprotic
         double OH;
   Delta=-Scat_net;
         DiffDelta=0;

         for(i = 0; i < NrOfTriproticBufferComponents  ; i++)
         {
         C=C_Triprotic[i];
                  for(j = i; j <(NrOfTriproticBufferComponents * 3)  ; j=j+NrOfTriproticBufferComponents)
                  {KT[l]= K_Triprotic[j];
```

```
                l=l+1;
                }
        l=0;
        SubDeltaH(SubDelta,SubDiffDelta,H,C,KT,3);
        Delta=Delta+SubDelta;
        DiffDelta=DiffDelta+SubDiffDelta;
        }

        for(i = 0; i < NrOfDiproticBufferComponents  ; i++)
        {
        C=C_Diprotic[i];
                for(j = i; j <(NrOfDiproticBufferComponents * 2)  ; j=j+NrOfDiproticBufferComponents)
                {KD[l]= K_Diprotic[j];
                l=l+1;
                }
        l=0;
        SubDeltaH(SubDelta,SubDiffDelta,H,C,KD,2);
        Delta=Delta+ SubDelta;
        DiffDelta=DiffDelta+SubDiffDelta;
        }

        for(i = 0; i < NrOfMonoproticBufferComponents  ; i++)
        {
        C=C_Monoprotic[i];
                if (C < 0.0)
                        {Kb[0]=1e-14/K_Monoprotic[i];
                        OH=1e-14/H;
                        SubDeltaH(SubDelta,SubDiffDelta,OH,C,Kb,1);
                        Delta=Delta+ SubDelta;
                        DiffDelta=DiffDelta+SubDiffDelta;}
                else
                        {KM[0]=K_Monoprotic[i];
                         SubDeltaH(SubDelta,SubDiffDelta,H,C,KM,1);
                        Delta=Delta+ SubDelta;
                        DiffDelta=DiffDelta+SubDiffDelta;}
        }
}

void SubDeltaH(double &SubDelta, double &SubDiffDelta,
                                double H,
                                double C,
                                double K[],
                                int N)
{
        double nom=0;
        double h=pow(H,N);
        double denom = h;
        double k=1;
        double h1=pow(H,N-1);
        double diffnom1= 0;
```

```
        double diffnom2=N*h1;

        int i=0;

for(i = 1; i <= N  ; i++)
{
        k=k*K[i-1];
        h=h/H;
        h1=h1/H;


        nom = nom + i*k*h*C;
        denom = denom + k*h;
        diffnom1=diffnom1+i*(N-i)*k*h1*C;
        diffnom2=diffnom2+(N-i)*h1*k;
}

SubDelta=nom/denom;
SubDiffDelta= diffnom1/denom - nom * diffnom2/pow(denom,2);
}
```