Aspects on ADM1 Implementation within the BSM2 Framework

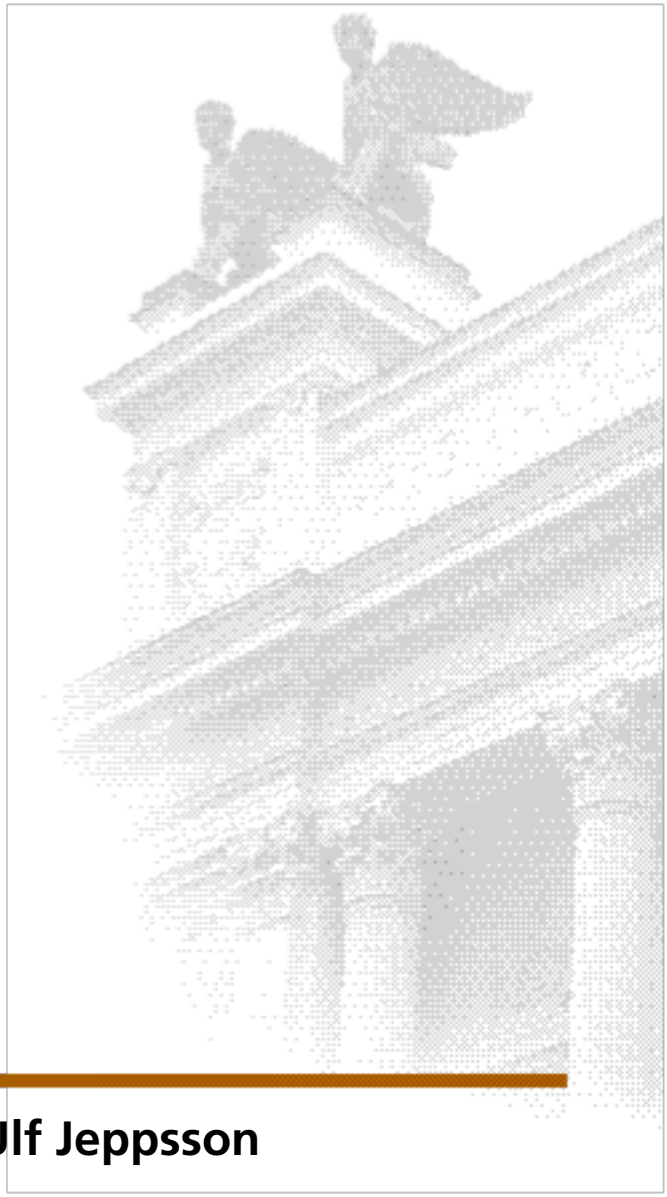
Rosén, Christian; Jeppsson, Ulf

2005

CODEN:LUTEDX/(TEIE-7224)/1-35/(2006)

Industrial Electrical Engineering and Automation

Aspects on ADM1 Implementation within the BSM2 Framework



Christian Rosén and Ulf Jeppsson

Dept. of Industrial Electrical Engineering and Automation
Lund University

Aspects on ADM1 implementation within the BSM2 framework

Christian Rosen and Ulf Jeppsson
Department of Industrial Electrical Engineering and Automation
Lund University, Box 118, SE-221 00 Lund, Sweden
christian.rosen@iea.lth.se, ulf.jeppsson@iea.lth.se

28th November 2006

1 Introduction

The current work of developing a control simulation benchmark goes back to the work carried out within the COST Action 682 ("Integrated Wastewater Management") and the IAWQ Task Group on Respirometry-Based control of the Activated Sludge Process in the late 90s. The work was continued in COST Action 624 until 2003 (Copp 2002). In 2005, the IWA Task Group on Benchmarking of Control Strategies for Wastewater Treatment Plants (BSM TG) was initiated and is now responsible for the continued development of the benchmark work.

This report describes the development of the Lund implementation of the Anaerobic Digester Model No 1 (ADM1). The aim of the report is to give an insight and rationale for the various changes and extensions made to the original model as reported in Batstone *et al.* (2002). Since the ADM1 was developed for general modelling of anaerobic digestion, opposed to for instance the ASM models, which were specifically developed for wastewater treatment, Batstone *et al.* (2002) leave some choices to the model implementor and this report will discuss how these choices were made in order to make this specific implementation as suitable for wastewater treatment sludge digestion as possible. The implementation was initiated by the inclusion of the sludge treatment in the IWA benchmark simulation model (BSM) to form a plant-wide or "within-the-fence" model.

2 The IWA benchmark simulation models

2.1 BSM1

The COST benchmark was originally defined as "a protocol to obtain a measure of performance of control strategies for activated sludge plants based on numerical, realistic simulations of the controlled plant". It was decided that the official plant layout would be aimed at carbon and nitrogen removal in a series of activated sludge reactors followed by a secondary settling tank, as this is perhaps the most common layout for full-scale WWT plants today. The selected model description for the biological processes was the recognised Activated Sludge Model no. 1, ASM1 (Henze *et al.* 2000), and the chosen settler model was a one-dimensional 10-layer model together with the double-exponential settling velocity model (Takacs *et al.* 1991).

To allow for uniform testing and evaluation three dynamic input data files have been developed, each representing different weather conditions (dry, rain and storm events) with realistic variations of the influent flow rate and composition. These files can be downloaded from the BSM TG website and have been widely used by various research groups also for other purposes than actual benchmarking. To represent a true challenge for on-line control strategies, the simulated plant is a high-loaded plant with significant influent variations.

In order to apply different control strategies a number of control handles (i.e. actuators) and measurement signals (i.e. sensors) must be available. A high degree of flexibility is required so that the implementation and evaluation of an innovative strategy is not limited. The default benchmark control strategy only use two measurement signals (dissolved oxygen and nitrate concentrations) and two control handles (air flow rate and internal recirculation flow rate). However, the benchmark simulation model allows for approximately 30 different control handles (different types of step-feed and step-recycling, external carbon source addition, etc.) and a wide variety of sensors. At this stage of development, all actuators are assumed to behave ideally whereas the sensors are divided into different classes

depending on their characteristics (delay, noise, detection limit, etc.). Consequently, just about every conceivable real-time control strategy for activated sludge systems can be implemented within the framework of the benchmark.

A general set of criteria has been defined within the benchmark description to assess the overall performance of the applied control strategy. Two system performance levels exist: (1) process performance and (2) control loop performance. The first level of assessment quantifies the effect of the control strategy on the plant in terms of an effluent quality index, effluent violations (related to defined limits for the plant) and operational costs (energy for pumping and aeration as well as sludge production). The second level of assessment quantifies the effect of the control strategy on controller performance by means of different statistical criteria on the controlled and manipulated variables. This more detailed analysis makes it possible to estimate the effect of a control strategy in terms of wear and tear of actuators, controller robustness, disturbance attenuation, etc.

All details of the current benchmark are available at the IWA TG website and also as a publication (Copp 2002). A substantial effort has gone into verifying the steady state and dynamic output data included in the description. Cross-platform checking of the benchmark has successfully demonstrated its use on a number of commercially available simulation software tools. The benchmark manual (Copp 2002) summarises the various tested implementations with helpful hints for new "benchmarkers". The complete manual can also be downloaded from the website. So far, results have been verified using BioWin™, EFOR™, GPS-X™, MATLAB/SIMULINK™, SIMBA®, STOAT™, WEST®, JASS and a user defined FORTRAN code.

2.2 BSM2

Although a valuable tool, the basic BSM1 does not allow for evaluation of control strategies on a plant-wide basis. BSM1 includes only an activated sludge system and a secondary clarifier. Consequently, only local control strategies can be evaluated. During the last decade the importance of integrated and plant-wide control has been stressed by the research community and the wastewater industry is starting to realise the benefits of such an approach. A WWTP should be considered as a unit, where primary and secondary clarification units, activated sludge reactors, anaerobic digesters, thickeners, dewatering systems, etc. are linked together and need to be operated and controlled not only on a local level as individual processes but by supervisory systems taking into account all the interactions between the processes. Otherwise, sub-optimisation will be an unavoidable outcome leading to reduced effluent quality and/or higher operational costs.

It is the intent of the proposed extended benchmark system, BSM2, to take the issues stated above into account. Consequently, wastewater pre-treatment and the sludge train of the WWTP are included in the BSM2. To allow for more thorough evaluation and additional control handles operating on longer time-scales, the benchmark evaluation period is extended to one year (compared to one week in BSM1). The slow dynamics of anaerobic digestion processes also necessitates a pro-longed evaluation period. With this extended evaluation period, it is reasonable to include seasonal effects on the WWTP in terms of temperature variations. The data files describing the BSM1 influent wastewater (dry, storm and rain weather data) have been used extensively by researchers. However, for the extended benchmark system a phenomenological mathematical model has been developed for raw wastewater and storm water generation, including urban drainage and sewer system effects (Gernaey *et al.* 2005, Gernaey *et al.* 2006). Additionally, intermittent loads reaching the plant by other means of transportation (e.g. trucks) may be included. A more detailed description of the BSM2 is given in Jeppsson *et al.* (2006).

3 ADM1 ODE implementation

In this section, the ordinary differential equation (ODE) model implementation reported in earlier versions of this document is presented.

3.1 Introduction

The ADM1 implementation in Matlab/Simulink deviates somewhat from the model description in Batstone *et al.* (2002). There are mainly three reasons for this. Firstly, the ADM1 is implemented so that it is consistent with the BSM2. Secondly, the computational requirements must be regarded. Thirdly, no explicit values are given in Batstone *et al.* (2002) with regard to carbon and nitrogen contents of some state variables. As the BSM TG had access to the "official" ADM1 implementation in Aquasim, which was developed by the ADM TG during their development of the model, the unknown parameter values were first chosen in accordance with the suggestions given there.

3.1.1 Mass balances

To maintain complete mass balances for all model components (COD, N, etc.) is a fundamental issue of any model. ASM1 only assures mass balances on a COD basis (as not all nitrogen components are included) whereas ASM2d maintains balances of COD, N, P and charge. ASM3 adds theoretical oxygen demand as a conservation variable. Based on Batstone *et al.* (2002), a few comments are required to avoid problems for anybody implementing the model.

The ADM1 includes a process referred to as disintegration, where a composite material (X_C) is transformed into various compounds (S_I, X_{ch}, X_{pr}, X_{li} and X_I). Assuming one COD mass unit of X_C completely disintegrating will produce:

$$f_{sI,xc}S_I + f_{xI,xc}X_I + f_{ch,xc}X_{ch} + f_{pr,xc}X_{pr} + f_{li,xc}X_{li} = 0.1S_I + 0.25X_I + 0.2X_{ch} + 0.2X_{pr} + 0.25X_{li}$$

A COD balance exists as long as the sum of all $f_{i,xc} = 1$. However, the proposed nitrogen content of X_C (N_{xc}) is 0.002 kmole N/kg COD. If we instead calculate the nitrogen content of the disintegration products (kmole N) using parameter values from Batstone *et al.* (2002), we get:

$$N_I 0.1S_I + N_I 0.25X_I + N_{ch} 0.2X_{ch} + N_{aa} 0.2X_{pr} + N_{li} 0.25X_{li} = 0.0002 + 0.0005 + 0.0014 = 0.0021$$

(note that carbohydrates and lipids contain no nitrogen). This means that for every kg of COD that disintegrates 0.1 mole of N is created (5% more than was originally there). Obviously, the nitrogen contents and yields from composites are highly variable and may need adjustment for every specific case study but the "default" parameter values should naturally close the mass balances. In this paper, we suggest new values for $f_{xI,xc} = 0.2$ and $f_{li,xc} = 0.3$. For the specific benchmark implementation we have also modified N_I to $0.06/14 \approx 0.00429$ kmole N/kg COD to be consistent with the ASM1 model, where inert particulate organic material is assumed to have a nitrogen content of 6 g N/g COD. Because of the latter modification N_{xc} is set to $0.0376/14 \approx 0.00269$ kmole N/kg COD to maintain the nitrogen balance.

The ADM1 contains the state variables inorganic carbon and inorganic nitrogen. These may act as source or sink terms to close mass balances. However, the provided stoichiometric matrix is not defined to take this into account. Let us take an example: decay of biomass (processes no 13-19) produces an equal amount of composites on a COD basis. However, the carbon content may certainly vary from biomass to composites resulting from decay. And what happens to the excess nitrogen within the biomass? It is suggested in Batstone *et al.* (2002) that the nitrogen content of bacteria (N_{bac}) is 0.00625 kmole N/kg COD, which is three times higher than the suggested value for N_{xc}. In such a case, it is logical to add a stoichiometric term ($N_{bac} - N_{xc}$) into the Petersen matrix, which will keep track of the fate of the excess nitrogen. The same principle holds for carbon during biomass decay, i.e. ($C_{bac} - C_{xc}$). A similar modification of the stoichiometric matrix should be done for the inorganic carbon for the processes "uptake of LCFA, valerate and butyrate" as well as for the disintegration and hydrolysis processes (both carbon and nitrogen).

The recommendation is to include stoichiometric relationships for all 19 processes regarding inorganic carbon and inorganic nitrogen. Basically all the required information is already given by Batstone *et al.* (2002). In many cases the expressions will be zero (depending on the selected values of the stoichiometric parameters) but there will be a guarantee that the mass balances are closed and the conservation law fulfilled at all times for COD, carbon and nitrogen. Moreover, such an approach makes model verification (finding coding errors in an implementation) much easier.

Using the proposed values of carbon content in the original ADM1 implementation it is stated that the carbon content of X_C is equal to 0.03 kmole C/kg COD. However, if we examine the carbon contents of the products arising from disintegration of composite material (based on the new fractionation parameters defined above) we get:

$$0.03 \cdot 0.1 \cdot S_I + 0.03 \cdot 0.2 \cdot X_I + 0.0313 \cdot 0.2 \cdot X_{ch} + 0.03 \cdot 0.2 \cdot X_{pr} + 0.022 \cdot 0.3 \cdot X_{li} = 0.02786 \text{ kmoleC/kgCOD}$$

If the original fractionation of composite material is used (i.e. $f_{xI,xc} = 0.25$ and $f_{li,xc} = 0.25$) we get a carbon content of the disintegrated products equal to 0.02826 kmole C/kg COD. In both cases, it is obvious that a significant amount of carbon "disappears" as a result of disintegration (6-7%). If the model is updated by adding the stoichiometric relationships to guarantee mass balances of carbon and nitrogen (described above) this disappearing carbon will end up as inorganic carbon and eventually it with lead to production of carbon dioxide in the gas phase. If the model is not updated as discussed above then 6-7% of the carbon content of composite material will simply be removed and the carbon mass balance will not hold. Moreover, as this extra carbon eventually ends up as carbon dioxide in the gas phase the original ADM1 model shows a tendency to produce a somewhat high percentage of CO_2 (32-35%) and somewhat low percentage of CH_4 (55-58%) in the produced gas using a reasonable sludge input. Note that the mass of produced CH_4 is still reasonable as this carbon unbalance leads to higher gas flow rate.

To avoid this problem it is suggested to use a value of 0.02786 kmole C/kg COD to describe the carbon content of composite material in the benchmark implementation. For reasonable sludge inputs this modification will normally lead to a production of 60-65% methane in the gas phase.

3.1.2 Acid-base equations

The acid-base equilibrium equations play an important role in ADM1 (e.g. for pH calculations). For persons primarily familiar with AS models these equations may create a problem as they do not normally appear in those. Moreover, (Batstone *et al.* 2002) focuses more on how the implementation should be done by implicit algebraic equations and is not completely clear on the ODE implementation. The general model matrix describes the transformations of valerate ($S_{va,total}$), butyrate, propionate, acetate, inorganic carbon and inorganic nitrogen. However, all these substances are made up by acid-base pairs (e.g. $S_{va,total} = S_{va^-} + S_{hva}$). It is suggested in Batstone *et al.* (2002) (Table B.4) that when using ODEs, the equations are defined for each acid and base, respectively. Based on our experiences it is more advantageous to implement the ODEs based on the total and one of the acid-base components instead. The remaining part can always be calculated as the total minus the calculated part. This approach actually makes the model more understandable also in other respects and due to numerical issues (we are subtracting very small and similar sized numbers) the error of calculated outputs are much closer to the solution a differential-algebraic equation (DAE) set-up would provide (when using a numerical solver with the same tolerance to integrate the ODEs). Using valerate as an example, the process rate (A4) in (Batstone *et al.* 2002) is:

$$K_{A,Bva}(S_{va^-}S_{H^+} - K_{a,va}S_{hva})$$

and herein we replace S_{hva} by $S_{va,total} - S_{va^-}$ and get

$$K_{A,Bva}(S_{va^-}(K_{a,va} + S_{H^+}) - K_{a,va}S_{va})$$

and, consequently, change the stoichiometry since S_{va} is not affected when the equilibrium ofS_{va^-} is changing. If using the suggested implicit solver to calculate the pH (or S_H^+) at every integration step (see below) then the above problem will no longer be an issue.

The choice of a ODE or DAE system for modelling the pH should not affect the overall results of the model. The DAE can be said to be a approximation of the ODE since, naturally, the pH dynamics are not instantaneous. However, it is very common to model the dynamics as a DAE system in biochemical/chemical engineering. Thus, we need to find the rate coefficients $k_{A,Bi}$ in such a way that the ODE produces the same results as the DAE. In Batstone *et al.* (2002), it is recommended that the coefficients should be chosen so that they are at least one order of magnitude faster (larger) than the fastest time constant of the remaining system and the value 1e8 M^{-1} d^{-1} is recommended. However, this is not sufficient. For the ODE to yield identical results, the rate coefficients need to be larger and a value of 1e10 M^{-1} d^{-1} is more appropriate. This will be illustrated in the results section of this report.

3.1.3 Temperature dependencies

In order to better allow for reasonable results for different temperatures within the digester, the benchmark ADM1 implementation now uses the complete information as stated in the ADM1 STR with regard to temperature dependency

of several physiochemical parameters (see the table for physiochemical parameters). This means that a model user can work with different temperatures when investigation the system without having to recalculate these parameters. The parameters that are now considered to be functions of temperature are:

K_w, $K_{a,co2}$, $K_{a,IN}$, $K_{H,co2}$, $K_{H,ch4}$, $K_{H,h2}$ and $p_{gas,h2o}$ (i.e. water vapour saturation pressure)

The K_a values for the organic acids are not assumed to vary within the selected temperature range (0 - 60 °C) and are assumed to be contants (see also Batstone *et al.* (2002), p. 39). For an even better temperature dependency of the AD model many of the biochemical parameter values would also need to be described as functions of temperature. Discussions between the ADM1 TG and the BSM TG on this topic are ongoing.

3.2 Model equations

3.2.1 Process rates

Biochemical process rates

$$
\begin{aligned}
\rho_1 &= k_{dis} \cdot X_c \\
\rho_2 &= k_{hyd,ch} \cdot X_{ch} \\
\rho_3 &= k_{hyd,pr} \cdot X_{pr} \\
\rho_4 &= k_{hyd,li} \cdot X_{li} \\
\rho_5 &= k_{m,su} \cdot \frac{S_{su}}{K_{S,su} + S_{su}} \cdot X_{su} \cdot I_5 \\
\rho_6 &= k_{m,aa} \cdot \frac{S_{aa}}{K_{S,aa} + S_{aa}} \cdot X_{aa} \cdot I_6 \\
\rho_7 &= k_{m,fa} \cdot \frac{S_{fa}}{K_{S,fa} + S_{fa}} \cdot X_{fa} \cdot I_7 \\
\rho_8 &= k_{m,c4} \cdot \frac{S_{va}}{K_{S,c4} + S_{va}} \cdot X_{c4} \cdot \frac{S_{va}}{S_{bu} + S_{va} + 1e-6} \cdot I_8 \\
\rho_9 &= k_{m,c4} \cdot \frac{S_{bu}}{K_{S,c4} + S_{bu}} \cdot X_{c4} \cdot \frac{S_{bu}}{S_{va} + S_{bu} + 1e-6} \cdot I_9 \\
\rho_{10} &= k_{m,pr} \cdot \frac{S_{pro}}{K_{S,pro} + S_{pro}} \cdot X_{pro} \cdot I_{10} \\
\rho_{11} &= k_{m,ac} \cdot \frac{S_{ac}}{K_{S,ac} + S_{ac}} \cdot X_{ac} \cdot I_{11} \\
\rho_{12} &= k_{m,h2} \cdot \frac{S_{h2}}{K_{S,h2} + S_{h2}} \cdot X_{h2} \cdot I_{12} \\
\rho_{13} &= k_{dec,Xsu} \cdot X_{su} \\
\rho_{14} &= k_{dec,Xaa} \cdot X_{aa} \\
\rho_{15} &= k_{dec,Xfa} \cdot X_{fa} \\
\rho_{16} &= k_{dec,Xc4} \cdot X_{c4} \\
\rho_{17} &= k_{dec,Xpro} \cdot X_{pro} \\
\rho_{18} &= k_{dec,Xac} \cdot X_{ac} \\
\rho_{19} &= k_{dec,Xh2} \cdot X_{h2}
\end{aligned}
$$

In the expressions for ρ_8 and ρ_9, a small constant (1e-6) has been added in order to avoid division by zero in the case of poor choice of initial conditions for S_{va} and S_{bu}, respectively.

Acid-base rates:

$$\rho_{A,4} = k_{A,Bva} \left(S_{va^-} (K_{a,va} + S_{H^+}) - K_{a,va} S_{va} \right)$$

$$
\begin{aligned}
\rho_{A,5} &= k_{A,Bbu}\left(S_{bu^-}(K_{a,bu}+S_{H^+})-K_{a,bu}S_{bu}\right)\\
\rho_{A,6} &= k_{A,Bpro}\left(S_{pro^-}(K_{a,pro}+S_{H^+})-K_{a,pro}S_{pro}\right)\\
\rho_{A,7} &= k_{A,Bac}\left(S_{ac^-}(K_{a,ac}+S_{H^+})-K_{a,ac}S_{ac}\right)\\
\rho_{A,10} &= k_{A,Bco2}\left(S_{hco3^-}(K_{a,co2}+S_{H^+})-K_{a,co2}S_{IC}\right)\\
\rho_{A,11} &= k_{A,BIN}\left(S_{nh3}(K_{a,IN}+S_{H^+})-K_{a,IN}S_{IN}\right)
\end{aligned}
$$

Gas transfer rates (note that S_{co2} is used in the expression for $\rho_{T,10}$, not S_{IC}):

$$
\begin{aligned}
\rho_{T,8} &= k_La\left(S_{h2}-16\cdot K_{H,h2}p_{gas,h2}\right)\\
\rho_{T,9} &= k_La\left(S_{ch4}-64\cdot K_{H,ch4}p_{gas,ch4}\right)\\
\rho_{T,10} &= k_La\left(S_{co2}-K_{H,co2}p_{gas,co2}\right)
\end{aligned}
$$

3.2.2 Process inhibition

Inhibition:

$$I_{5,6} = I_{pH,aa}\cdot I_{IN,lim}$$

$$I_7 = I_{pH,aa}\cdot I_{IN,lim}\cdot I_{h2,fa}$$

$$I_{8,9} = I_{pH,aa}\cdot I_{IN,lim}\cdot I_{h2,c4}$$

$$I_{10} = I_{pH,aa}\cdot I_{IN,lim}\cdot I_{h2,pro}$$

$$I_{11} = I_{pH,ac}\cdot I_{IN,lim}\cdot I_{nh3}$$

$$I_{12} = I_{pH,h2}\cdot I_{IN,lim}$$

$$
\begin{aligned}
I_{pH,aa} &= \begin{cases} exp\left(-3\left(\frac{pH-pH_{UL,aa}}{pH_{UL,aa}-pH_{LL,aa}}\right)^2\right) & : \quad pH<pH_{UL,aa}\\ 1 & : \quad pH>pH_{UL,aa}\end{cases}\\
I_{pH,ac} &= \begin{cases} exp\left(-3\left(\frac{pH-pH_{UL,ac}}{pH_{UL,ac}-pH_{LL,ac}}\right)^2\right) & : \quad pH<pH_{UL,ac}\\ 1 & : \quad pH>pH_{UL,ac}\end{cases}\\
I_{pH,h2} &= \begin{cases} exp\left(-3\left(\frac{pH-pH_{UL,h2}}{pH_{UL,h2}-pH_{LL,h2}}\right)^2\right) & : \quad pH<pH_{UL,h2}\\ 1 & : \quad pH>pH_{UL,h2}\end{cases}\\
I_{IN,lim} &= \frac{1}{1+K_{S,IN}/S_{IN}}\\
I_{h2,fa} &= \frac{1}{1+S_{h2}/K_{I,h2,fa}}\\
I_{h2,c4} &= \frac{1}{1+S_{h2}/K_{I,h2,c4}}\\
I_{h2,pro} &= \frac{1}{1+S_{h2}/K_{I,h2,pro}}\\
I_{nh3} &= \frac{1}{1+S_{nh3}/K_{I,nh3}}
\end{aligned}
$$

$$pH = -log_{10}(S_{H^+})$$

Batstone *et al.* (2002) used switch functions to account for inhibition due to pH. These functions are, however, discontinuous and in a stiff system, such a switch can favour numerical instabilities. To reduce this risk, a number of alternative functions can be used to express the inhibition due to pH. Expressions based on hyperbolic tangents are often used in chemical engineering:

$$I_{pH} = \frac{1}{2}(1 + tan(a\varphi)) \tag{1}$$

with

$$\varphi = \frac{pH - \frac{pH_{LL}+pH_{UL}}{2}}{\frac{pH_{LL}+pH_{UL}}{2}}$$

Values of $a = 11$ for $I_{pH,aa}$ and $a = 22$ for $I_{pH,ac}$ and $I_{pH,h2}$, respectively, are appropriate to fit the function given in Batstone *et al.* (2002). Another option is functions based on Hill functions:

$$I_{pH} = \frac{pH^n}{pH^n + K_{pH}^n} \tag{2}$$

with

$$K_{pH} = \frac{pH_{LL} + pH_{UL}}{2}$$

Siegrist *et al.* (2002) used a Hill inhibition function based on the hydrogen ion concentration instead. For the ADM1, this would give the following expression:

$$I_{pH} = \frac{K_{pH}^n}{S_{H^+}^n + K_{pH}^n} \tag{3}$$

with

$$K_{pH} = 10^{-\frac{pH_{LL}+pH_{UL}}{2}}$$

The appropriate values of n in the respective Hill functions are quite different. To fit the original function given in Batstone *et al.* (2002), $n = 24$ for $I_{pH,aa}$ when the pH based Hill function is used and $n = 2$ for the hydrogen ion based function. For $I_{pH,ac}$ and $I_{pH,h2}$ the respective values of n are 45 and 3.

NOTE! For any practical purpose, the choice of function among the three above is arbitrary but for BSM2 the hydrogen ion based function, i.e. Expression 3 above is chosen.

It should be noted that the appropriate values of n are dependent of the values of pH_{LL} and pH_{UL}. Therefore, if these limits are to be changed, it may be wise to implement:

$$n_{aa} = \frac{3.0}{pH_{UL,aa} - pH_{LL,aa}}$$
$$n_{ac} = \frac{3.0}{pH_{UL,ac} - pH_{LL,ac}}$$
$$n_{h2} = \frac{3.0}{pH_{UL,h2} - pH_{LL,h2}}$$

for the hydrogen ion based function.

3.2.3 Water phase equations

Differential equations 1-4, soluble matter: State No.

$$\frac{dS_{su}}{dt} = \frac{q_{in}}{V_{liq}}(S_{su,in} - S_{su}) + \rho_2 + (1 - f_{fa,li})\,\rho_4 - \rho_5 \quad (1)$$

$$\frac{dS_{aa}}{dt} = \frac{q_{in}}{V_{liq}}(S_{aa,in} - S_{aa}) + \rho_3 - \rho_6 \quad (2)$$

$$\frac{dS_{fa}}{dt} = \frac{q_{in}}{V_{liq}}(S_{fa,in} - S_{fa}) + f_{fa,li}\rho_4 - \rho_7 \quad (3)$$

$$\frac{dS_{va}}{dt} = \frac{q_{in}}{V_{liq}}(S_{va,in} - S_{va}) + (1 - Y_{aa})\,f_{va,aa}\rho_6 - \rho_8 \quad (4)$$

Differential equations 5-8, soluble matter: State No.

$$\frac{dS_{bu}}{dt} = \frac{q_{in}}{V_{liq}}(S_{bu,in} - S_{bu}) + (1 - Y_{su})\,f_{bu,su}\rho_5 + (1 - Y_{aa})\,f_{bu,aa}\rho_6 - \rho_9 \quad (5)$$

$$\frac{dS_{pro}}{dt} = \frac{q_{in}}{V_{liq}}(S_{pro,in} - S_{pro}) + (1 - Y_{su})\,f_{pro,su}\rho_5 + (1 - Y_{aa})\,f_{pro,aa}\rho_6 + (1 - Y_{c4})\,0.54\rho_8 - \rho_{10} \quad (6)$$

$$\frac{dS_{ac}}{dt} = \frac{q_{in}}{V_{liq}}(S_{ac,in} - S_{ac}) + (1 - Y_{su})\,f_{ac,su}\rho_5 + (1 - Y_{aa})\,f_{ac,aa}\rho_6 + (1 - Y_{fa})\,0.7\rho_7 + (1 - Y_{c4})\,0.31\rho_8 + (1 - Y_{c4})\,0.8\rho_9 + (1 - Y_{pro})\,0.57\rho_{10} - \rho_{11} \quad (7)$$

$$\frac{dS_{h2}}{dt} = \frac{q_{in}}{V_{liq}}(S_{h2,in} - S_{h2}) + (1 - Y_{su})\,f_{h2,su}\rho_5 + (1 - Y_{aa})\,f_{h2,aa}\rho_6 + (1 - Y_{fa})\,0.3\rho_7 + (1 - Y_{c4})\,0.15\rho_8 + (1 - Y_{c4})\,0.2\rho_9 + (1 - Y_{pro})\,0.43\rho_{10} - \rho_{12} - \rho_{T,8} \quad (8)$$

Differential equations 9-12, soluble matter: State No.

$$\frac{dS_{ch4}}{dt} = \frac{q_{in}}{V_{liq}}(S_{ch4,in} - S_{ch4}) + (1 - Y_{ac})\,\rho_{11} + (1 - Y_{h2})\,\rho_{12} - \rho_{T,9} \quad (9)$$

$$\frac{dS_{IC}{}^{*}}{dt} = \frac{q_{in}}{V_{liq}}(S_{IC,in} - S_{IC}) - \sum_{j=1}^{19}\left(\sum_{i=1-9,11-24} C_i v_{i,j}\rho_j\right) - \rho_{T,10} \quad (10)$$

$$\frac{dS_{IN}}{dt} = \frac{q_{in}}{V_{liq}}(S_{IN,in} - S_{IN}) - Y_{su}N_{bac}\rho_5 + (N_{aa} - Y_{aa}N_{bac})\,\rho_6 - Y_{fa}N_{bac}\rho_7 - Y_{c4}N_{bac}\rho_8 - Y_{c4}N_{bac}\rho_9 - Y_{pro}N_{bac}\rho_{10} - Y_{ac}N_{bac}\rho_{11} - Y_{h2}N_{bac}\rho_{12} + (N_{bac} - N_{xc})\sum_{i=13}^{19}\rho_i + (N_{xc} - f_{xI,xc}N_I - f_{sI,xc}N_I - f_{pr,xc}N_{aa})\,\rho_1 \quad (11)$$

$$\frac{dS_I}{dt} = \frac{q_{in}}{V_{liq}}(S_{I,in} - S_I) + f_{sI,xc}\rho_1 \quad (12)$$

* See next page for further explanation.

More specifically, the sum in Equation 10 is calculated as:

$$\sum_{j=1}^{19}\left(\sum_{i=1-9,11-24} C_i v_{i,j}\rho_j\right) = \sum_{k=1}^{12} s_k\rho_k + s_{13}(\rho_{13}+\rho_{14}+\rho_{15}+\rho_{16}+\rho_{17}+\rho_{18}+\rho_{19})$$

where

$$\begin{aligned}
s_1 &= -C_{xc} + f_{sI,xc}C_{sI} + f_{ch,xc}C_{ch} + f_{pr,xc}C_{pr} + f_{li,xc}C_{li} + f_{xI,xc}C_{xI}\\
s_2 &= -C_{ch} + C_{su}\\
s_3 &= -C_{pr} + C_{aa}\\
s_4 &= -C_{li} + (1-f_{fa,li})C_{su} + f_{fa,li}C_{fa}\\
s_5 &= -C_{su} + (1-Y_{su})(f_{bu,su}C_{bu} + f_{pro,su}C_{pro} + f_{ac,su}C_{ac}) + Y_{su}C_{bac}\\
s_6 &= -C_{aa} + (1-Y_{aa})(f_{va,aa}C_{va} + f_{bu,aa}C_{bu} + f_{pro,aa}C_{pro} + f_{ac,aa}C_{ac}) + Y_{aa}C_{bac}\\
s_7 &= -C_{fa} + (1-Y_{fa})\,0.7C_{ac} + Y_{fa}C_{bac}\\
s_8 &= -C_{va} + (1-Y_{c4})\,0.54C_{pro} + (1-Y_{c4})\,0.31C_{ac} + Y_{c4}C_{bac}\\
s_9 &= -C_{bu} + (1-Y_{c4})\,0.8C_{ac} + Y_{c4}C_{bac}\\
s_{10} &= -C_{pro} + (1-Y_{pro})\,0.57C_{ac} + Y_{pro}C_{bac}\\
s_{11} &= -C_{ac} + (1-Y_{ac})C_{ch4} + Y_{ac}C_{bac}\\
s_{12} &= (1-Y_{h2})C_{ch4} + Y_{h2}C_{bac}\\
s_{13} &= -C_{bac} + C_{xc}
\end{aligned}$$

Differential equations 13-16, particulate matter: State No.

$$\frac{dX_c}{dt} = \frac{q_{in}}{V_{liq}}(X_{c,in} - X_c) - \rho_1 + \sum_{i=13}^{19}\rho_i \quad (13)$$

$$\frac{dX_{ch}}{dt} = \frac{q_{in}}{V_{liq}}(X_{ch,in} - X_{ch}) + f_{ch,xc}\rho_1 - \rho_2 \quad (14)$$

$$\frac{dX_{pr}}{dt} = \frac{q_{in}}{V_{liq}}(X_{pr,in} - X_{pr}) + f_{pr,xc}\rho_1 - \rho_3 \quad (15)$$

$$\frac{dX_{li}}{dt} = \frac{q_{in}}{V_{liq}}(X_{li,in} - X_{li}) + f_{li,xc}\rho_1 - \rho_4 \quad (16)$$

Differential equations 17-20, particulate matter: State No.

$$\frac{dX_{su}}{dt} = \frac{q_{in}}{V_{liq}}(X_{su,in} - X_{su}) + Y_{su}\rho_5 - \rho_{13} \quad (17)$$

$$\frac{dX_{aa}}{dt} = \frac{q_{in}}{V_{liq}}(X_{aa,in} - X_{aa}) + Y_{aa}\rho_6 - \rho_{14} \quad (18)$$

$$\frac{dX_{fa}}{dt} = \frac{q_{in}}{V_{liq}}(X_{fa,in} - X_{fa}) + Y_{fa}\rho_7 - \rho_{15} \quad (19)$$

$$\frac{dX_{c4}}{dt} = \frac{q_{in}}{V_{liq}}(X_{c4,in} - X_{c4}) + Y_{c4}\rho_8 + Y_{c4}\rho_9 - \rho_{16} \quad (20)$$

Differential equations 21-24, particulate matter: State No.

$$\frac{dX_{pro}}{dt} = \frac{q_{in}}{V_{liq}}(X_{pro,in} - X_{pro}) + Y_{pro}\rho_{10} - \rho_{17} \quad (21)$$

$$\frac{dX_{ac}}{dt} = \frac{q_{in}}{V_{liq}}(X_{ac,in} - X_{ac}) + Y_{ac}\rho_{11} - \rho_{18} \quad (22)$$

$$\frac{dX_{h2}}{dt} = \frac{q_{in}}{V_{liq}}(X_{h2,in} - X_{h2}) + Y_{h2}\rho_{12} - \rho_{19} \quad (23)$$

$$\frac{dX_I}{dt} = \frac{q_{in}}{V_{liq}}(X_{I,in} - X_I) + f_{xI,xc}\rho_1 \quad (24)$$

Differential equations 25-26, cations and anions: State No.

$$\frac{dS_{cat^+}}{dt} = \frac{q_{in}}{V_{liq}}\left(S_{cat^+,in} - S_{cat^+}\right) \quad (25)$$

$$\frac{dS_{an^-}}{dt} = \frac{q_{in}}{V_{liq}}\left(S_{an^-,in} - S_{an^-}\right) \quad (26)$$

Differential equations 27-32, ion states: State No.

$$\frac{dS_{va^-}}{dt} = -\rho_{A,4} \quad (27)$$

$$\frac{dS_{bu^-}}{dt} = -\rho_{A,5} \quad (28)$$

$$\frac{dS_{pro^-}}{dt} = -\rho_{A,6} \quad (29)$$

$$\frac{dS_{ac^-}}{dt} = -\rho_{A,7} \quad (30)$$

$$\frac{dS_{hco3^-}}{dt} = -\rho_{A,10} \quad (31)$$

$$\frac{dS_{nh3}}{dt} = -\rho_{A,11} \quad (32)$$

Algebraic equation:

$$S_{H^+} = -\frac{\Theta}{2} + \frac{1}{2}\sqrt{\Theta^2 + 4K_W}$$

$$\Theta = S_{cat^+} + S_{nh4^+} - S_{hco3^-} - \frac{S_{ac^-}}{64} - \frac{S_{pro^-}}{112} - \frac{S_{bu^-}}{160} - \frac{S_{va^-}}{208} - S_{an^-}$$

$$S_{nh4^+} = S_{IN} - S_{nh3}$$

$$S_{co2} = S_{IC} - S_{hco3^-}$$

3.2.4 Gas phase equations

Differential equations: State No.

$$\frac{dS_{gas,h2}}{dt} = -\frac{S_{gas,h2}q_{gas}}{V_{gas}} + \rho_{T,8} \cdot \frac{V_{liq}}{V_{gas}} \quad (33)$$

$$\frac{dS_{gas,ch4}}{dt} = -\frac{S_{gas,ch4}q_{gas}}{V_{gas}} + \rho_{T,9} \cdot \frac{V_{liq}}{V_{gas}} \quad (34)$$

$$\frac{dS_{gas,co2}}{dt} = -\frac{S_{gas,co2}q_{gas}}{V_{gas}} + \rho_{T,10} \cdot \frac{V_{liq}}{V_{gas}} \quad (35)$$

Algebraic equations:

$$\begin{aligned} p_{gas,h2} &= S_{gas,h2} \cdot \frac{RT_{op}}{16} \\ p_{gas,ch4} &= S_{gas,ch4} \cdot \frac{RT_{op}}{64} \\ p_{gas,co2} &= S_{gas,co2} \cdot RT_{op} \\ q_{gas} &= \frac{RT_{op}}{P_{atm} - p_{gas,H_2O}} \cdot V_{liq} \left(\frac{\rho_{T,8}}{16} + \frac{\rho_{T,9}}{64} + \rho_{T,10} \right) \end{aligned}$$

A problem with this way of calculating the gas flow rate is that is may give rise to numerical problems in the solution of the equations. Multiple steady states as well as numerical instability have been reported among users. An alternative way of calculating the gas flow rate is also given in Batstone *et al.* (2002):

$$q_{gas} = k_p(P_{gas} - P_{atm})$$

with

$$P_{gas} = p_{gas,h2} + p_{gas,ch4} + p_{gas,co2} + p_{gas,h20}$$

The alternative expression assumes an overpressure in the head space. Consequently, the flow rate is calculated at a higher pressure compared to the first expression. To compensate for this, the expression needs to rewritten into

$$q_{gas} = k_p(P_{gas} - P_{atm}) \cdot \frac{P_{gas}}{P_{atm}}$$

to obtain the flow rate at atmospheric pressure. Although this compensation factor is included, the two expressions will not yield identical results. Depending on the operational overpressure, which is a function of the value of parameter k_p (related to the friction in the gas outlet), the alternative expression results in a slightly smaller flow rate. The reason for this is that the liquid-gas transfer rates ($\rho_{T,8}$, $\rho_{T,9}$, $\rho_{T,10}$) will be different. A comparison of the two expressions when the same overpressure is applied shows very similar results (the relative error in the range of 1e-5).

NOTE! For BSM2 the alternative way (assuming an overpressure in the head space) of calculating the gas flow rate is used.

4 ADM1 DAE implementation

It has been realised that the ODE implementation may be problematic for use in the BSM2 framework. Therefore, 2 DAE versions have been developed (Rosen *et al.* 2006). In this section, the differential algebraic equation model implementation of ADM1 is presented. Two different DAE models are discussed: a model with algebraic pH (S_{H^+}) calculations (DAE$_{pH}$) and a model with algebraic pH and S_{h2} calculations (DAE$_{pH,S_{h2}}$). The second model implementation is motivated by the stiffness problem of the ODE and DAE$_{pH}$ implementations.

4.1 Introduction

The fact that the Matlab/Simulink implementation discussed in this work aims at an integrated part of the BSM2 puts some requirements on the way the ADM1 is implemented. The model must be able to handle dynamic inputs, time-discrete and event-driven control actions as well as stochastic inputs or noise and still be sufficiently efficient and fast to allow for extensive simulations.

4.1.1 Dynamic inputs

BSM2 aims at developing and evaluating control strategies for wastewater treatment. The challenge to control a WWTP lies mainly in the changing influent wastewater characteristics. The generation of dynamic input is, thus, an integrated part of the BSM2. This means that, except in order to obtain initial conditions, BSM2 is always simulated using dynamic input and, consequently, no plant unit is ever at steady state. According to the BSM2 protocol, using dynamic influent data, the plant is simulated for 63 days to reach a pseudo steady state. This is followed by 182 days of simulation for initialisation of control and/or monitoring algorithms. The subsequent 364 days of simulation is the actual evaluation period. In total, this encompasses 609 days of dynamic simulations with new data every 15 minutes (Gernaey *et al.* 2005).

4.1.2 Control actions

The BSM2 is a control benchmark. It should, thus, be possible to test and evaluate various types of controllers (Jeppsson *et al.* 2006). From a numerical point of view, continuous controllers yield the least computational effort. However, discrete controllers are more common and many of the commercially available sensors are also discrete. Moreover, some control actions can be event driven when applying rule-based control (e.g. if-then-else rules). Introducing discrete controllers and sensors as well as event-driven control in the model results in a so-called hybrid system.

4.1.3 Noise sources

To obtain realistic and useful evaluation results from an investigation of a control strategy, the strategy must be subjected to various types of errors and disturbances encountered in real operation. One of the most important sources of errors is sensor noise and delays. Realistic sensor model behaviour requires the dynamic properties and disturbance sources to be represented. This typically includes modelling of noise and time response and, if not a continuous sensor, the sampling and measuring interval (Rieger *et al.* 2003). Noise generation must be done individually for each sensor in the system so that it is uncorrelated.

4.1.4 Simulating stiff systems in Matlab/Simulink

A system is called stiff, when the range of the time constants is large. This means that some of the system states react quickly whereas some react sluggishly. The ADM1 is a very stiff system with time constants ranging from fractions of a second to months. This makes the simulation of such a system challenging and in order to avoid excessively long simulation times, one needs to be somewhat creative when implementing the model.

Some of the solvers in Matlab/Simulink are so called stiff solvers and, consequently, capable of solving stiff systems. However, a problem common to all stiff solvers is the difficulty to handle dynamic input - including noise. The more stochastic or random an input variable behaves, the more problematic is the simulation using a stiff solver. The reason for this is that in stiff solvers, predictions of future state values are carried out. However, predictions of future state values affected by stochastic inputs will result in poor results, slowing down the solver by limiting its ability to use

long integration steps. Simulation of the BSM2 is, thus, subject to the following dilemma. BSM2, which includes ASM1 and ADM1 models, is a very stiff system and, consequently, a stiff solver should be used. However, since BSM2 is a control simulation benchmark, noise must be included, calling for an explicit (i.e. non-stiff) solver.

4.1.5 ODE and DAE systems

When the states of a system are described only by ordinary differential equations, the system is said to be an ODE system. If the system is stiff, it is sometimes possible to rewrite some of the system equations in order to omit the fastest states. The rationale for this is that from the slower state's point of view, the fast states can be considered instantaneous and possible to describe by algebraic equations. Such systems are normally referred to as differential algebraic equation (DAE) systems. By rewriting an ODE system to a DAE system, the stiffness can be decreased, allowing for explicit solvers to be used and for stochastic elements to be incorporated. The drawback is that the DAE system is only an approximation of the original system and the effect of this approximation must be considered and investigated for each specific simulation model.

4.1.6 Time constants in ADM1

As mentioned before, the ADM1 includes time constants in a wide range; from milliseconds for pH to weeks or months for the states describing various fractions of active biomass. Since most control actions affecting the anaerobic digester are fairly slow, it makes sense to investigate which fast states can be approximated by algebraic equations. In Batstone *et al.* (2002), it is suggested that the pH (S_{H^+}) state is calculated by algebraic equations. However, this will only partially solve the stiffness problem. There are other fast states and a closer investigation suggests that the state describing hydrogen (S_{h2}) also needs to be approximated by an algebraic equation.

4.2 DAE equations

4.2.1 pH and S_{h2} solvers

As mentioned above, stiffness of the ADM1 can be reduced by approximating the differential equations of the pH and S_{h2} states by algebraic equations. An implicit algebraic equation for the pH calculation is given in (Batstone *et al.* 2002) (Table B.3). It has been suggested to calculate the S_{H^+} and, consequently, the pH from the sum of all charges, which is supposed to be zero. The obtained implicit algebraic equations are non-linear and therefore can be solved only by an iterative numerical method. In this case, the Newton-Raphson method used in Volcke *et al.* (2005) for calculation of the pH and equilibrium concentrations was implemented. By using this method the new value of the unknown state is calculated at each iteration step k as:

$$S_{H^+,k+1} = S_{H^+,k} - \frac{E(S_{H^+,k})}{dE(S_{H^+})/dS_{H^+}|_{S_{H^+,k}}}$$

where $S_{H^+,k}$ is the value of the state obtained from the previous iteration step and $E(S_{H^+,k})$ is the value of the algebraic equation that has to be zero for the equilibrium, i.e.:

$$\begin{aligned} E(S_{H^+,k}) \quad = \quad & S_{cat^+,k} + S_{nh4^+,k} + S_{H^+,k} - S_{hco3^-,k} - \\ & \frac{S_{ac^-,k}}{64} - \frac{S_{pr^-,k}}{112} - \frac{S_{bu^-,k}}{160} - \frac{S_{va^-,k}}{208} - \frac{K_W}{S_{H^+,k}} - S_{an^-,k} \end{aligned}$$

The gradient of the algebraic equation $dE(S_{H^+,k})/dS_{H^+}|_{S_{H^+,k}}$ is also needed for calculation of the new state value. Since this expression is rather complicated it is not presented here. The reader is referred to Appendix A for details on the expression. The iteration is repeated as long as $E(S_{H^+,k})$ remains larger than the predefined tolerance value, which in our case is set to 10^{-12}. Normally only two or three iterations are required to solve the equation at each time step.

The differential equation for the S_{h2} state, explicitly given in the ODE implementation of this report, can be approximated by an algebraic equation in a similar way as was the case for the S_{H^+} state, simply by setting its differential to zero (assuming fast dynamics), see Equation 8. The iteration is carried out in the same way as for the S_{H^+} calculation,

this time using

$$E(S_{h2,k}) = \frac{q_{in}}{V_{liq}}(S_{h2,in} - S_{h2,k}) + (1 - Y_{su})\, f_{h2,su}\rho_5 + (1 - Y_{aa})\, f_{h2,aa}\rho_6 + (1 - Y_{fa})\, 0.3\rho_7 + (1 - Y_{c4})\, 0.15\rho_8 + (1 - Y_{c4})\, 0.2\rho_9 + (1 - Y_{pro})\, 0.43\rho_{10} - \rho_{12} - \rho_{T,8}$$

and the gradient of $E(S_{h2,k+1})$. The expression of the gradient is quite complex and the reader is referred to Appendix A for details. To obtain the gradients for the S_{H^+} and S_{h2} equations, it is recommended that a tool for handling mathematics symbolically is used (e.g. Maple or Mathematica).

5 Comparison between ODE and DAE implementations

5.1 Introduction

In order to verify the DAE implementation suggested here, it is compared with the ODE implementation. In steady state, the differences should be very small (close to machine precision). However, the dynamic errors must be studied closer – both for the BSM2 anticipated operating region as well as for other possible operational regions.

5.2 Steady-state comparison

The three model implementations discussed in this report, i.e. ODE, DAE with only pH solver and DAE with pH solver and S_{h2} solver, were simulated for 200 days to reach steady state. Both relative and absolute errors were investigated using the ODE simulation as a reference. Only minor errors were encountered – the largest relative errors in the range of 10^{-6}. The largest absolute errors, 10^{-5}, were found in the states with large steady-state values (no scaling of states in the implementations). The result is not surprising since the difference between the models is in the dynamic description of the equations.

5.3 Dynamic comparison

To test the dynamic differences between the model implementations in the BSM2 operational region, a preliminary version of the BSM2 is run to create sensible input data for the digester. The simulation period is 50 days and includes recycling streams from the digester. Thus, the digester is included in the model when data is produced. The input data are stored at 15 minute sampling intervals.

To evaluate the dynamic differences, the digester model is simulated alone with the stored input data as input for the whole duration. The last 7 days are used for evaluation be means of calculating the mean absolute relative error of each variable using the ODE implementation as reference. The main difference between DAE1 and DAE2 is in state variable 8, i.e. the hydrogen state. The mean error is slightly larger than 0.01% which must be considered as acceptable. Especially, since all the other state errors (perhaps except state 23 X_{h2}) are more or less equal.

The fact that the mean errors are in the range of 1e-4 when each sample time is investigated independently does not mean that the relative error of the mean value of each state variable is in the same range. The most deviating variable is the gas flow rate, which is not surprising for the reasons discussed earlier in conjunction with the choice of gas flow rate expression. Also here, the DAE1 and DAE2 behaves similar in terms of dynamic errors.

NOTE! The choice between ODE, DAE1 and DAE2 is up to the user. If acceptable computation times can be achieved with ODE or DAE1 implementations there is no other advantage to use DAE2. But for Matlab/Simulink it seems that with currently available solvers, DAE2 is the only feasible choice.

6 ADM1 benchmark model parameters

Stoichiometric parameter values

Parameter	Value	Unit	Process(es)	Comment
$f_{sI,xc}$	0.1	–	1	
$f_{xI,xc}$	0.2	–	1	
$f_{ch,xc}$	0.2	–	1	
$f_{pr,xc}$	0.2	–	1	
$f_{li,xc}$	0.3	–	1	Note: $1 - f_{ch,xc} - f_{pr,xc} - f_{sI,xc} - f_{xI,xc} - f_{li,xc} = 0$
N_{xc}	0.0376/14	kmole N (kg COD)$^{-1}$	1, 13-19	to maintain N mass balance for disintegration
N_I	0.06/14	kmole N (kg COD)$^{-1}$	1	6% on weight basis in benchmark ASM
N_{aa}	0.007	kmole N (kg COD)$^{-1}$	1,6	
C_{xc}	0.02786	kmole C (kg COD)$^{-1}$	1,13-19	C_{13} in Eq. 10
C_{sI}	0.03	kmole C (kg COD)$^{-1}$	1	C_{12} in Eq. 10
C_{ch}	0.0313	kmole C (kg COD)$^{-1}$	1,2	C_{14} in Eq. 10
C_{pr}	0.03	kmole C (kg COD)$^{-1}$	1,3	C_{15} in Eq. 10
C_{li}	0.022	kmole C (kg COD)$^{-1}$	1,4	C_{16} in Eq. 10
C_{xI}	0.03	kmole C (kg COD)$^{-1}$	1	C_{24} in Eq. 10
C_{su}	0.0313	kmole C (kg COD)$^{-1}$	2,5	C_1 in Eq. 10
C_{aa}	0.03	kmole C (kg COD)$^{-1}$	3,6	C_2 in Eq. 10
$f_{fa,li}$	0.95	–	4	
C_{fa}	0.0217	kmole C (kg COD)$^{-1}$	4,7	C_3 in Eq. 10
$f_{h2,su}$	0.19	–	5	
$f_{bu,su}$	0.13	–	5	
$f_{pro,su}$	0.27	–	5	
$f_{ac,su}$	0.41	–	5	
N_{bac}	0.08/14	kmole N (kg COD)$^{-1}$	5-19	8% on weight basis in benchmark ASM
C_{bu}	0.025	kmole C (kg COD)$^{-1}$	5,6,9	C_5 in Eq. 10
C_{pro}	0.0268	kmole C (kg COD)$^{-1}$	5,6,8,10	C_6 in Eq. 10
C_{ac}	0.0313	kmole C (kg COD)$^{-1}$	5-11	C_7 in Eq. 10
C_{bac}	0.0313	kmole C (kg COD)$^{-1}$	5-19	C_{17-23} in Eq. 10
Y_{su}	0.1	–	5	kg COD_x (kg COD_s)$^{-1}$
$f_{h2,aa}$	0.06	–	6	
$f_{va,aa}$	0.23	–	6	
$f_{bu,aa}$	0.26	–	6	
$f_{pro,aa}$	0.05	–	6	
$f_{ac,aa}$	0.40	–	6	
C_{va}	0.024	kmole C (kg COD)$^{-1}$	6,8	C_4 in Eq. 10
Y_{aa}	0.08	–	6	kg COD_x (kg COD_s)$^{-1}$
Y_{fa}	0.06	–	7	kg COD_x (kg COD_s)$^{-1}$
Y_{c4}	0.06	–	8,9	kg COD_x (kg COD_s)$^{-1}$
Y_{pro}	0.04	–	10	kg COD_x (kg COD_s)$^{-1}$
C_{ch4}	0.0156	kmole C (kg COD)$^{-1}$	11,12	C_9 in Eq. 10
Y_{ac}	0.05	–	11	kg COD_x (kg COD_s)$^{-1}$
Y_{h2}	0.06	–	12	kg COD_s (kg COD_x)$^{-1}$

-Note that C_{h2} and C_{IN}, i.e. C_8 and C_{11}, are equal to zero in Equation 10.

Biochemical parameter values

Parameter	Value	Unit	Process(es)	Comment
k_{dis}	0.5	d^{-1}	1	
$k_{hyd,ch}$	10	d^{-1}	2	
$k_{hyd,pr}$	10	d^{-1}	3	
$k_{hyd,li}$	10	d^{-1}	4	
$K_{S,IN}$	1e-4	M	5-12	
$k_{m,su}$	30	d^{-1}	5	
$K_{S,su}$	0.5	kg COD m^{-3}	5	
$pH_{UL,aa}$	5.5	–	5-10	in I_{5-10}
$pH_{LL,aa}$	4	–	5-10	in I_{5-10}
$k_{m,aa}$	50	d^{-1}	6	
$K_{S,aa}$	0.3	kg COD m^{-3}	6	
$k_{m,fa}$	6	d^{-1}	7	
$K_{S,fa}$	0.4	kg COD m^{-3}	7	
$K_{Ih2,fa}$	5e-6	kg COD m^{-3}	7	in I_7
$k_{m,c4}$	20	d^{-1}	8,9	
$K_{S,c4}$	0.2	kg COD m^{-3}	8,9	
$K_{Ih2,c4}$	1e-5	kg COD m^{-3}	8,9	in I_8 and I_9
$k_{m,pro}$	13	d^{-1}	10	
$K_{S,pro}$	0.1	kg COD m^{-3}	10	
$K_{Ih2,pro}$	3.5e-6	kg COD m^{-3}	10	in I_{10}
$k_{m,ac}$	8	d^{-1}	11	
$K_{S,ac}$	0.15	kg COD m^{-3}	11	
$K_{I,nh3}$	0.0018	M	11	in I_{11}
$pH_{UL,ac}$	7	–	11	in I_{11}
$pH_{LL,ac}$	6	–	11	in I_{11}
$k_{m,h2}$	35	d^{-1}	12	
$K_{S,h2}$	7e-6	kg COD m^{-3}	12	
$pH_{UL,h2}$	6	–	12	in I_{12}
$pH_{LL,h2}$	5	–	12	in I_{12}
$k_{dec,Xsu}$	0.02	d^{-1}	13	
$k_{dec,Xaa}$	0.02	d^{-1}	14	
$k_{dec,Xfa}$	0.02	d^{-1}	15	
$k_{dec,Xc4}$	0.02	d^{-1}	16	
$k_{dec,Xpro}$	0.02	d^{-1}	17	
$k_{dec,Xac}$	0.02	d^{-1}	18	
$k_{dec,Xh2}$	0.02	d^{-1}	19	

The unit M is defined as kmole m^{-3} according to Batstone *et al.* (2002).

Physiochemical parameter values

Parameter	Value	Unit	Comment
R	0.083145	bar M^{-1} K^{-1}	
T_{base}	298.15	K	
T_{op}	308.15	K	user defined
K_w	$\exp\left(\frac{55900}{R*100} * \left(\frac{1}{T_{base}} - \frac{1}{T_{op}}\right)\right)$	M 10^{-14}	$\approx$ 2.08e-14
$K_{a,va}$	$10^{-4.86}$	M	$\approx$ 1.38e-5
$K_{a,bu}$	$10^{-4.82}$	M	$\approx$ 1.5e-5
$K_{a,pro}$	$10^{-4.88}$	M	$\approx$ 1.32e-5
$K_{a,ac}$	$10^{-4.76}$	M	$\approx$ 1.74e-5
$K_{a,co2}$	$10^{-6.35}\exp\left(\frac{7646}{R*100}\left(\frac{1}{T_{base}} - \frac{1}{T_{op}}\right)\right)$	M	$\approx$ 4.94e-7
$K_{a,IN}$	$10^{-9.25}\exp\left(\frac{51965}{R*100}\left(\frac{1}{T_{base}} - \frac{1}{Top}\right)\right)$	M	$\approx$ 1.11e-9
$k_{A,Bva}$	1e10	M^{-1} d^{-1}	set to be at least three
$k_{A,Bbu}$	1e10	M^{-1} d^{-1}	orders of magnitude higher
$k_{A,Bpro}$	1e10	M^{-1} d^{-1}	than the fastest time
$k_{A,Bac}$	1e10	M^{-1} d^{-1}	constant of the system
$k_{A,Bco2}$	1e10	M^{-1} d^{-1}	
$k_{A,BIN}$	1e10	M^{-1} d^{-1}	
P_{atm}	1.013	bar	
$p_{gas,h2o}$	$0.0313\exp\left(5290\left(\frac{1}{T_{base}} - \frac{1}{T_{op}}\right)\right)$	bar	$\approx$ 0.0557
k_p	5e4	m^3 d^{-1} bar^{-1}	
k_La	200	d^{-1}	
$K_{H,co2}$	$0.035\exp\left(\frac{-19410}{R*100}\left(\frac{1}{T_{base}} - \frac{1}{T_{op}}\right)\right)$	M_{liq} bar^{-1}	$\approx$ 0.0271
$K_{H,ch4}$	$0.0014\exp\left(\frac{-14240}{R*100}\left(\frac{1}{T_{base}} - \frac{1}{T_{op}}\right)\right)$	M_{liq} bar^{-1}	$\approx$ 0.00116
$K_{H,h2}$	$7.8e\text{-}4\exp\left(\frac{-4180}{R*100}\left(\frac{1}{T_{base}} - \frac{1}{T_{op}}\right)\right)$	M_{liq} bar^{-1}	$\approx$ 7.38e-4

Physical parameter values

Parameter	Value	Unit	Comment
V_{liq}	3400	m^3	
V_{gas}	300	m^3	

7 ADM1 benchmark model steady-state results

In this section, the results of a steady state simulation are given. It should serve as a starting point for verification of any model implementation according to the description given in this report. The inputs may not be completely realistic for all variables but have been chosen so that every input is active (i.e. non-zero) thereby allowing all internal modes of the ADM model to be excited. Note that the resulting pH in the AD is about 7.5, i.e. the pH-inhibition functions are not really active in this test case. The retention time is 20 days and the operating temperature 35 řC. As a part of the BSM TG work model interfaces for connecting ASM1 and ADM1 (and vice versa) have been developed (although not presented here). The AS to AD interface fractionates the incoming COD into inerts, carbohydrates, proteins and lipids rather than putting the incoming COD into the AD as composite material. Consequently, the composite material value is low and the values of the other input variables are comparably high. The influent TSS concentration is about 4.5%. The produced gas contains about 61% methane and 34% carbon dioxide (the rest is mainly water vapour). For a more thorough dynamic verification, further information is required.

Steady-state input variable values

State No.	Variable	Value	Unit
1	$S_{su,in}$	0.01	kg COD m^{-3}
2	$S_{aa,in}$	0.001	kg COD m^{-3}
3	$S_{fa,in}$	0.001	kg COD m^{-3}
4	$S_{va,in}$	0.001	kg COD m^{-3}
5	$S_{bu,in}$	0.001	kg COD m^{-3}
6	$S_{pro,in}$	0.001	kg COD m^{-3}
7	$S_{ac,in}$	0.001	kg COD m^{-3}
8	$S_{h2,in}$	1.0E-8	kg COD m^{-3}
9	$S_{ch4,in}$	1.0E-5	kg COD m^{-3}
10	$S_{IC,in}$	0.04	kmole C m^{-3}
11	$S_{IN,in}$	0.01	kmole N m^{-3}
12	$S_{I,in}$	0.02	kg COD m^{-3}
13	$X_{xc,in}$	2.0	kg COD m^{-3}
14	$X_{ch,in}$	5.0	kg COD m^{-3}
15	$X_{pr,in}$	20.0	kg COD m^{-3}
16	$X_{li,in}$	5.0	kg COD m^{-3}
17	$X_{su,in}$	0.0	kg COD m^{-3}
18	$X_{aa,in}$	0.01	kg COD m^{-3}
19	$X_{fa,in}$	0.01	kg COD m^{-3}
20	$X_{c4,in}$	0.01	kg COD m^{-3}
21	$X_{pro,in}$	0.01	kg COD m^{-3}
22	$X_{ac,in}$	0.01	kg COD m^{-3}
23	$X_{h2,in}$	0.01	kg COD m^{-3}
24	$X_{I,in}$	25.0	kg COD m^{-3}
25	$S_{cat,in}$	0.04	kmole m^{-3}
26	$S_{an,in}$	0.02	kmole m^{-3}
–	q_{in}	170.0	$m^3 d^{-1}$
–	T_{op}	35.0	°C

Steady-state output variable values

State No.	Variable	Value	Unit
1	S_{su}	0.0119548297170	kg COD m^{-3}
2	S_{aa}	0.0053147401716	kg COD m^{-3}
3	S_{fa}	0.0986214009308	kg COD m^{-3}
4	S_{va}	0.0116250064639	kg COD m^{-3}
5	S_{bu}	0.0132507296663	kg COD m^{-3}
6	S_{pro}	0.0157836662845	kg COD m^{-3}
7	S_{ac}	0.1976297169375	kg COD m^{-3}
8	S_{h2}	0.0000002359451	kg COD m^{-3}
9	S_{ch4}	0.0550887764460	kg COD m^{-3}
10	S_{IC}	0.1526778706263	kmole C m^{-3}
11	S_{IN}	0.1302298158037	kmole N m^{-3}
12	S_I	0.3286976637215	kg COD m^{-3}
13	X_{xc}	0.3086976637215	kg COD m^{-3}
14	X_{ch}	0.0279472404350	kg COD m^{-3}
15	X_{pr}	0.1025741061067	kg COD m^{-3}
16	X_{li}	0.0294830497073	kg COD m^{-3}
17	X_{su}	0.4201659824546	kg COD m^{-3}
18	X_{aa}	1.1791717989237	kg COD m^{-3}
19	X_{fa}	0.2430353447194	kg COD m^{-3}
20	X_{c4}	0.4319211056360	kg COD m^{-3}
21	X_{pro}	0.1373059089340	kg COD m^{-3}
22	X_{ac}	0.7605626583132	kg COD m^{-3}
23	X_{h2}	0.3170229533613	kg COD m^{-3}
24	X_I	25.6173953274430	kg COD m^{-3}
25	S_{cat}	0.0400000000000	kmole m^{-3}
26	S_{an}	0.0200000000000	kmole m^{-3}
–	Q	170.000000000000	$m^{-3} d^{-1}$
–	T_{op}	35.0000000000000	°C
–	pH	7.4655377698929	
–	S_{H+}	0.0000000342344	kmole H^+ m^{-3}
27	S_{va^-}	0.0115962470726	kg COD m^{-3}
28	S_{bu^-}	0.0132208262485	kg COD m^{-3}
29	S_{pro^-}	0.0157427831916	kg COD m^{-3}
30	S_{ac^-}	0.1972411554365	kg COD m^{-3}
31	S_{hco3^-}	0.1427774793921	kmole C m^{-3}
–	S_{co2}	0.0099003912343	kmole C m^{-3}
32	S_{nh3}	0.0040909284584	kmole N m^{-3}
–	S_{nh4+}	0.1261388873452	kmole N m^{-3}
33	$S_{gas,h2}$	0.0000102410356	kg COD m^{-3}
34	$S_{gas,ch4}$	1.6256072099814	kg COD m^{-3}
35	$S_{gas,co2}$	0.0141505346784	kmole C m^{-3}
–	$p_{gas,h2}$	0.0000163991826	bar
–	$p_{gas,ch4}$	0.6507796328232	bar
–	$p_{gas,co2}$	0.3625527133281	bar
–	p_{gas}	1.0690164904089	bar
–	q_{gas}	2955.70345419378	$Nm^3 d^{-1}$

Note that the gas flow, q_{gas}, is the flow rate at atmospheric pressure and **not** at the pressure of the head space.

References

Batstone D.J., Keller J., Angelidaki I., Kalyuzhnyi S.V., Pavlostathis S.G., Rozzi A., Sanders W.T.M., Siegrist H. and Vavilin V.A. (2002). Anaerobic Digestion Model No. 1. *IWA STR No. 13*, IWA Publishing, London, UK.

Copp J.B. (ed.) (2002). *The COST Simulation Benchmark - Description and Simulator Manual*. ISBN 92-894-1658-0, Office for Official Publications of the European Communities, Luxembourg.

Gernaey, K.V., Rosen, C., Benedetti, L. and Jeppsson, U. (2005). Phenomenological modelling of wastewater treatment plant influent disturbances scenarios. *10th International Conference on Urban Drainage (10ICUD)* 21-26 August, Copenhagen, Denmark.

Gernaey, K.V., Rosen, C and Jeppsson, U. (2006). WWTP dynamic disturbance modelling - an eesential module for long-term benchmarking development. *Wat. Sci. Tech.* 53(4-5), 225-234.

Henze, M., Gujer, W., Mino, T. and van Loosdrecht, M. (2000). Activated sludge models AMS1, ASM2, ASM2d and ASM3. *IWA STR No. 9*, IWA Publishing, London, UK.

Jeppsson, U., Rosen, C., Alex, J., Copp, J., Gernaey, K.V., Pons, M.-N. and Vanrolleghem, P.A. (2006). Towards a benchmark simulation model for plant-wide control strategy performance evaluation of WWTPs. *Wat. Sci. Tech.* 51(1), 287-295.

Rieger, L., Alex, J., Winkler, S., Boehler, M., Thomann, M. and Siegrist, H. (2003). Towards a benchmark simulation model for plant-wide control strategy performance evaluation of WWTPs. *Wat. Sci. Tech.* 47(2), 103-112.

Rosen, C., Vrecko, D., Gernaey, K.V., Pons, M.N. and Jeppsson, U. (2006). Implementing ADM1 for plant-wide benchmark simulations in Matiab/Simulink. *Wat. Sci. Tech.* 54(4), 11-19.

Siegrist, H., Vogt, D., Garcia-Heras, J.L. and Gujer, W. (2002). Mathematical model for meso- and thermophilic anaerobic digestion. *Environ. Sci. Technol*. 36, 1113-1123.

Takacs, I.,Patry, G.G. and Nolasco, D. (1991). A dynamic model of the clarification-thinckening process. *Wat. Res.*, 25(10), 1263-1271.

Volcke, E.I.P., Van Hulle, S., Deksissa, T., Zaher, U. and Vanrolleghem, P.A. (2005). *Calculation of pH and concentration of equilibrium components during dynamic simulation by means of a charge balance*. BIOMATH Tech. Report, Ghent University, Ghent, Belgium.

A Code for DAE implementation in Matlab/Simulink

In this appendix, the solvers for pH and $Sh2$ used in the Lund implementation of ADM1 are presented. The solver files are written in C for Matlab/Simulink S-function utility. If used on this platform, they should work just as they are presented below. If another platform is used, the reader should focus on the functions `Equ` and `gradEqu` for calculation of the equations $E(S_X)$ and $dE(S_X)/dS_X|_{S_{X,k}}$, respectively, and the functions `pHsolver` and `Sh2solver`, respectively, for the iteration (the Newton-Raphson algorithm) to find the solution S_X.

A.1 C-file for pH solver

```
/*
 * pHsolv.c is a C-file S-function level 2 that calculates
 * the algebraic equations for pH and ion states of the ADM1 model.
 * This solver is based on the implementation proposed by Dr Eveline
 * Volcke, BIOMATH, Ghent University, Belgium.
 * Computational speed could be further enhanced by sending all
 * parameters directly from the adm1 module
 * instead of recalculating them within this module.
 *
 * Copyright (2006):
 * Dr Christian Rosen, Dr Darko Vrecko and Dr Ulf Jeppsson
 * Dept. Industrial Electrical Engineering and Automation (IEA)
 * Lund University, Sweden
 * http://www.iea.lth.se/
 */
#define S_FUNCTION_NAME pHsolv
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#include <math.h>
#define XINIT(S) ssGetSFcnParam(S,0)
#define PAR(S) ssGetSFcnParam(S,1)

/*
 * mdlInitializeSizes:
 * 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)
{
    ssSetNumSFcnParams(S, 2); /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
    /* Return if number of expected != number of actual parameters */
    return;
}
    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 7);
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 51); /*(S, port index, port width)*/
    ssSetInputPortDirectFeedThrough(S, 0, 0);
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 7);
    ssSetNumSampleTimes(S, 1);
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}
```

```
/*
 * mdlInitializeSampleTimes:
 * 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)
/* mdlInitializeConditions:
 * 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 i;

    for (i = 0;i < 7; i++) {
        x0[i] = mxGetPr(XINIT(S))[i];
    }
}
#endif /* MDL_INITIALIZE_CONDITIONS */

#undef MDL_START /* Change to #undef to remove function */
#if defined(MDL_START)
/* mdlStart:
 * 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 */

/*
 * mdlOutputs
 * 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;

    for (i = 0; i < 7; i++) {
        y[i] = x[i]; /* state variables are passed on as output variables */
    }
}

static real_T Equ(SimStruct *S)
{
     real_T *x = ssGetDiscStates(S);
     InputRealPtrsType u = ssGetInputPortRealSignalPtrs(S,0);

     static real_T K_w, pK_w_base, K_a_va, pK_a_va_base, K_a_bu, pK_a_bu_base,
K_a_pro, pK_a_pro_base, K_a_ac, pK_a_ac_base, K_a_co2, pK_a_co2_base,
K_a_IN,pK_a_IN_base, T_base, T_op, R, factor;

     R = mxGetPr(PAR(S))[77];
     T_base = mxGetPr(PAR(S))[78];
     T_op = mxGetPr(PAR(S))[79];
     pK_w_base = mxGetPr(PAR(S))[80];
     pK_a_va_base = mxGetPr(PAR(S))[81];
     pK_a_bu_base = mxGetPr(PAR(S))[82];
     pK_a_pro_base = mxGetPr(PAR(S))[83];
     pK_a_ac_base = mxGetPr(PAR(S))[84];
     pK_a_co2_base = mxGetPr(PAR(S))[85];
     pK_a_IN_base = mxGetPr(PAR(S))[86];

     factor = (1.0/T_base - 1.0/T_op)/(100.0*R);
     K_w = pow(10,-pK_w_base)*exp(55900.0*factor);
/* T adjustment for K_w */
     K_a_va = pow(10,-pK_a_va_base);
     K_a_bu = pow(10,-pK_a_bu_base);
     K_a_pro = pow(10,-pK_a_pro_base);
     K_a_ac = pow(10,-pK_a_ac_base);
     K_a_co2 = pow(10,-pK_a_co2_base)*exp(7646.0*factor);
/* T adjustment for K_a_co2 */
     K_a_IN = pow(10,-pK_a_IN_base)*exp(51965.0*factor);
/* T adjustment for K_a_IN */

     x[1] = K_a_va**u[3]/(K_a_va+x[0]);   /* Sva- */
     x[2] = K_a_bu**u[4]/(K_a_bu+x[0]);   /* Sbu- */
     x[3] = K_a_pro**u[5]/(K_a_pro+x[0]); /* Spro- */
     x[4] = K_a_ac**u[6]/(K_a_ac+x[0]);   /* Sac- */
     x[5] = K_a_co2**u[9]/(K_a_co2+x[0]); /* SHCO3- */
     x[6] = K_a_IN**u[10]/(K_a_IN+x[0]);  /* SNH3 */

     return *u[24]+(*u[10]-x[6])+x[0]-x[5]-x[4]/64-x[3]/112-
x[2]/160-x[1]/208-K_w/x[0]-*u[25]; /* SH+ equation */
}
```

```
static real_T gradEqu(SimStruct *S)
{
    real_T *x = ssGetDiscStates(S);
    InputRealPtrsType u = ssGetInputPortRealSignalPtrs(S,0);

    static real_T K_w, pK_w_base, K_a_va, pK_a_va_base, K_a_bu,
pK_a_bu_base, K_a_pro, pK_a_pro_base, K_a_ac, pK_a_ac_base, K_a_co2,
pK_a_co2_base, K_a_IN, pK_a_IN_base, T_base, T_op, R, factor;

     R = mxGetPr(PAR(S))[77];
     T_base = mxGetPr(PAR(S))[78];
     T_op = mxGetPr(PAR(S))[79];
     pK_w_base = mxGetPr(PAR(S))[80];
     pK_a_va_base = mxGetPr(PAR(S))[81];
     pK_a_bu_base = mxGetPr(PAR(S))[82];
     pK_a_pro_base = mxGetPr(PAR(S))[83];
     pK_a_ac_base = mxGetPr(PAR(S))[84];
     pK_a_co2_base = mxGetPr(PAR(S))[85];
     pK_a_IN_base = mxGetPr(PAR(S))[86];

     factor = (1.0/T_base - 1.0/T_op)/(100.0*R);
     K_w = pow(10,-pK_w_base)*exp(55900.0*factor); /* T adjustment for K_w */
     K_a_va = pow(10,-pK_a_va_base);
     K_a_bu = pow(10,-pK_a_bu_base);
     K_a_pro = pow(10,-pK_a_pro_base);
     K_a_ac = pow(10,-pK_a_ac_base);
     K_a_co2 = pow(10,-pK_a_co2_base)*exp(7646.0*factor);
/* T adjustment for K_a_co2 */
     K_a_IN = pow(10,-pK_a_IN_base)*exp(51965.0*factor);
/* T adjustment for K_a_IN */

    return 1+K_a_IN**u[10]/((K_a_IN+x[0])*(K_a_IN+x[0]))
/* Gradient of SH+ equation */
           +K_a_co2**u[9]/((K_a_co2+x[0])*(K_a_co2+x[0]))
           +1/64.0*K_a_ac**u[6]/((K_a_ac+x[0])*(K_a_ac+x[0]))
           +1/112.0*K_a_pro**u[5]/((K_a_pro+x[0])*(K_a_pro+x[0]))
           +1/160.0*K_a_bu**u[4]/((K_a_bu+x[0])*(K_a_bu+x[0]))
           +1/208.0*K_a_va**u[3]/((K_a_va+x[0])*(K_a_va+x[0]))
           +K_w/(x[0]*x[0]);
}

static void pHsolver(SimStruct *S)
{
    real_T *x = ssGetDiscStates(S);
    static real_T delta;
    static real_T S_H_ion0;
    static int_T i;
    static const real_T TOL = 1e-12;
    static const real_T MaxSteps= 1000;

    S_H_ion0 = x[0]; /* SH+ */
    i = 1;
    delta = 1.0;
```

```
    /* Newton-Raphson algorithm */
    while ( (delta > TOL || delta < -TOL) && (i <= MaxSteps) ) {
        delta = Equ(S);
        x[0] = S_H_ion0 - delta/gradEqu(S); /* Calculate the new SH+ */

        if (x[0] <= 0) {
            x[0] = 1e-12; /* to avoid numerical problems */
        }
        S_H_ion0 = x[0];
        ++i;
    }
}

#define MDL_UPDATE /* Change to #undef to remove function */
#if defined(MDL_UPDATE)
/*
 * mdlUpdate:
 * 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)
{
   pHsolver(S);
}
#endif /* MDL_UPDATE */

#undef MDL_DERIVATIVES /* Change to #undef to remove function */
#if defined(MDL_DERIVATIVES)
/*
 * mdlDerivatives:
 * 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 */

/*
 * mdlTerminate:
 * 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)
{
}
#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
```

A.2 C-file for S_{h2} solver

```
/*
 * Sh2solv.c is a C-file S-function level 2 that solves the algebraic
 * equation for Sh2 of the ADM1 model,thereby reducing the stiffness of the
 * system considerably (if used together with a pHsolver).
 *
 * Copyright (2006):
 * Dr Christian Rosen, Dr Darko Vrecko and Dr Ulf Jeppsson
 * Dept. Industrial Electrical Engineering and Automation (IEA)
 * Lund University, Sweden
 * http://www.iea.lth.se/
 */

#define S_FUNCTION_NAME Sh2solv
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#include <math.h>
#define XINIT(S) ssGetSFcnParam(S,0)
#define PAR(S) ssGetSFcnParam(S,1)
#define V(S)      ssGetSFcnParam(S,2)

/*
 * mdlInitializeSizes:
 * 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)
{
    ssSetNumSFcnParams(S, 3); /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
    /* Return if number of expected != number of actual parameters */
    return;
}
    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 1);
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 52); /*(S, port index, port width)*/
    ssSetInputPortDirectFeedThrough(S, 0, 0);
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 1);
    ssSetNumSampleTimes(S, 1);
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}

/*
 * mdlInitializeSampleTimes:
 * 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)
/* mdlInitializeConditions:
 * 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*/

    x0[0] = mxGetPr(XINIT(S))[0];
}
#endif /* MDL_INITIALIZE_CONDITIONS */


#undef MDL_START /* Change to #undef to remove function */
#if defined(MDL_START)
/* mdlStart:
 * 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 */


/*
 * mdlOutputs
 * 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);

    y[0] = x[0]; /* state variable is passed on as output variable */
}


static real_T Equ(SimStruct *S)
{
```

```
    real_T *x = ssGetDiscStates(S);
    InputRealPtrsType u = ssGetInputPortRealSignalPtrs(S,0);

    static real_T eps, f_h2_su, Y_su, f_h2_aa, Y_aa, Y_fa, Y_c4, Y_pro,
K_S_IN, k_m_su, K_S_su, pH_UL_aa, pH_LL_aa, k_m_aa;
    static real_T K_S_aa, k_m_fa, K_S_fa, K_Ih2_fa, k_m_c4, K_S_c4,
K_Ih2_c4, k_m_pro, K_S_pro, K_Ih2_pro;
    static real_T pH_UL_ac, pH_LL_ac, k_m_h2, K_S_h2, pH_UL_h2,
pH_LL_h2, R, T_base, T_op, kLa, K_H_h2, K_H_h2_base, V_liq, pH_op,I_pH_aa;
    static real_T I_pH_h2, I_IN_lim, I_h2_fa, I_h2_c4, I_h2_pro, inhib[6];
    static real_T proc5, proc6, proc7, proc8, proc9, proc10, proc12,
p_gas_h2, procT8, reac8;
    static real_T pHLim_aa, pHLim_h2, a_aa, a_h2, S_H_ion, n_aa, n_h2;

    eps = 0.000001;

    f_h2_su = mxGetPr(PAR(S))[18];
    Y_su = mxGetPr(PAR(S))[27];
    f_h2_aa = mxGetPr(PAR(S))[28];
    Y_aa = mxGetPr(PAR(S))[34];
    Y_fa = mxGetPr(PAR(S))[35];
    Y_c4 = mxGetPr(PAR(S))[36];
    Y_pro = mxGetPr(PAR(S))[37];
    K_S_IN = mxGetPr(PAR(S))[45];
    k_m_su = mxGetPr(PAR(S))[46];
    K_S_su = mxGetPr(PAR(S))[47];
    pH_UL_aa = mxGetPr(PAR(S))[48];
    pH_LL_aa = mxGetPr(PAR(S))[49];
    k_m_aa = mxGetPr(PAR(S))[50];
    K_S_aa = mxGetPr(PAR(S))[51];
    k_m_fa = mxGetPr(PAR(S))[52];
    K_S_fa = mxGetPr(PAR(S))[53];
    K_Ih2_fa = mxGetPr(PAR(S))[54];
    k_m_c4 = mxGetPr(PAR(S))[55];
    K_S_c4 = mxGetPr(PAR(S))[56];
    K_Ih2_c4 = mxGetPr(PAR(S))[57];
    k_m_pro = mxGetPr(PAR(S))[58];
    K_S_pro = mxGetPr(PAR(S))[59];
    K_Ih2_pro = mxGetPr(PAR(S))[60];
    pH_UL_ac = mxGetPr(PAR(S))[64];
    pH_LL_ac = mxGetPr(PAR(S))[65];
    k_m_h2 = mxGetPr(PAR(S))[66];
    K_S_h2 = mxGetPr(PAR(S))[67];
    pH_UL_h2 = mxGetPr(PAR(S))[68];
    pH_LL_h2 = mxGetPr(PAR(S))[69];
    R = mxGetPr(PAR(S))[77];
    T_base = mxGetPr(PAR(S))[78];
    T_op = mxGetPr(PAR(S))[79];
    kLa = mxGetPr(PAR(S))[94];
    K_H_h2_base = mxGetPr(PAR(S))[98];
    V_liq = mxGetPr(V(S))[0];

    K_H_h2 = K_H_h2_base*exp(-4180.0*(1.0/T_base - 1.0/T_op)/(100.0*R));
 /* T adjustment for K_H_h2 */
```

```
     pH_op = *u[33];   /* pH */
     S_H_ion = *u[34]; /* SH+ */

   /* STRs function
   if (pH_op < pH_UL_aa)
    I_pH_aa = exp(-3.0*(pH_op-pH_UL_aa)*(pH_op-pH_UL_aa)/((pH_UL_aa-pH_LL_aa)*
(pH_UL_aa-pH_LL_aa)));
   else
    I_pH_aa = 1.0;
   if (pH_op < pH_UL_h2)
    I_pH_h2 = exp(-3.0*(pH_op-pH_UL_h2)*(pH_op-pH_UL_h2)/((pH_UL_h2-pH_LL_h2)*
(pH_UL_h2-pH_LL_h2)));
   else
    I_pH_h2 = 1.0;
   */

/* Hill function on pH inhibition
pHLim_aa = (pH_UL_aa + pH_LL_aa)/2.0;
pHLim_ac = (pH_UL_ac + pH_LL_ac)/2.0;
pHLim_h2 = (pH_UL_h2 + pH_LL_h2)/2.0;
I_pH_aa = pow(pH_op,24)/(pow(pH_op,24)+pow(pHLim_aa ,24));
I_pH_ac = pow(pH_op,45)/(pow(pH_op,45)+pow(pHLim_ac ,45));
I_pH_h2 = pow(pH_op,45)/(pow(pH_op,45)+pow(pHLim_h2 ,45));
*/

/* MNPs function on pH inhibition, ADM1 Workshop, Copenhagen 2005.
a_aa = 25.0/(pH_UL_aa-pH_LL_aa+eps);
a_ac = 25.0/(pH_UL_ac-pH_LL_ac+eps);
a_h2 = 25.0/(pH_UL_h2-pH_LL_h2+eps);

I_pH_aa = 0.5*(1+tanh(a_aa*(pH_op/pHLim_aa - 1.0)));
I_pH_ac = 0.5*(1+tanh(a_ac*(pH_op/pHLim_ac - 1.0)));
I_pH_h2 = 0.5*(1+tanh(a_h2*(pH_op/pHLim_h2 - 1.0)));
*/

/* Hill function on SH+ used within BSM2, ADM1 Workshop, Copenhagen 2005. */
     pHLim_aa = pow(10,(-(pH_UL_aa + pH_LL_aa)/2.0));
     pHLim_h2 = pow(10,(-(pH_UL_h2 + pH_LL_h2)/2.0));
     n_aa=3.0/(pH_UL_aa-pH_LL_aa);
     n_h2=3.0/(pH_UL_h2-pH_LL_h2);
     I_pH_aa = pow(pHLim_aa,n_aa)/(pow(S_H_ion,n_aa)+pow(pHLim_aa ,n_aa));
     I_pH_h2 = pow(pHLim_h2,n_h2)/(pow(S_H_ion,n_h2)+pow(pHLim_h2 ,n_h2));

     I_IN_lim = 1.0/(1.0+K_S_IN/(*u[10]));
     I_h2_fa = 1.0/(1.0+x[0]/K_Ih2_fa);
     I_h2_c4 = 1.0/(1.0+x[0]/K_Ih2_c4);
     I_h2_pro = 1.0/(1.0+x[0]/K_Ih2_pro);

     inhib[0] = I_pH_aa*I_IN_lim;
     inhib[1] = inhib[0]*I_h2_fa;
     inhib[2] = inhib[0]*I_h2_c4;
     inhib[3] = inhib[0]*I_h2_pro;
     inhib[5] = I_pH_h2*I_IN_lim;
```

```
     proc5 = k_m_su**u[0]/(K_S_su+*u[0])**u[16]*inhib[0];
     proc6 = k_m_aa**u[1]/(K_S_aa+*u[1])**u[17]*inhib[0];
     proc7 = k_m_fa**u[2]/(K_S_fa+*u[2])**u[18]*inhib[1];
     proc8 = k_m_c4**u[3]/(K_S_c4+*u[3])**u[19]**u[3]/(*u[3]+
*u[4]+eps)*inhib[2];
     proc9 = k_m_c4**u[4]/(K_S_c4+*u[4])**u[19]**u[4]/(*u[3]+
*u[4]+eps)*inhib[2];
     proc10 = k_m_pro**u[5]/(K_S_pro+*u[5])**u[20]*inhib[3];

     proc12 = k_m_h2*x[0]/(K_S_h2+x[0])**u[22]*inhib[5];

     p_gas_h2 = *u[43]*R*T_op/16.0;
     procT8 = kLa*(x[0]-16.0*K_H_h2*p_gas_h2);

     reac8 = (1.0-Y_su)*f_h2_su*proc5+(1.0-Y_aa)*f_h2_aa*proc6+
(1.0-Y_fa)*0.3*proc7+(1.0-Y_c4)*0.15*proc8+(1.0-Y_c4)*0.2*proc9+
(1.0-Y_pro)*0.43*proc10-proc12-procT8;

     return 1/V_liq**u[26]*(*u[51]-x[0])+reac8; /* Sh2 equation */
}


static real_T gradEqu(SimStruct *S)
{
     real_T *x = ssGetDiscStates(S);
     InputRealPtrsType u = ssGetInputPortRealSignalPtrs(S,0);

     static real_T eps, f_h2_su, Y_su, f_h2_aa, Y_aa, Y_fa, Y_c4,
Y_pro, K_S_IN, k_m_su, K_S_su, pH_UL_aa, pH_LL_aa, k_m_aa;
     static real_T K_S_aa, k_m_fa, K_S_fa, K_Ih2_fa, k_m_c4,
K_S_c4, K_Ih2_c4, k_m_pro, K_S_pro, K_Ih2_pro;
     static real_T pH_UL_ac, pH_LL_ac, k_m_h2, K_S_h2, pH_UL_h2, pH_LL_h2,
R, T_base, T_op, kLa, K_H_h2, K_H_h2_base, V_liq, pH_op, I_pH_aa, I_pH_h2;
     static real_T pHLim_aa, pHLim_h2, a_aa, a_h2, S_H_ion, n_aa, n_h2;

     eps = 0.000001;

     f_h2_su = mxGetPr(PAR(S))[18];
     Y_su = mxGetPr(PAR(S))[27];
     f_h2_aa = mxGetPr(PAR(S))[28];
     Y_aa = mxGetPr(PAR(S))[34];
     Y_fa = mxGetPr(PAR(S))[35];
     Y_c4 = mxGetPr(PAR(S))[36];
     Y_pro = mxGetPr(PAR(S))[37];
     K_S_IN = mxGetPr(PAR(S))[45];
     k_m_su = mxGetPr(PAR(S))[46];
     K_S_su = mxGetPr(PAR(S))[47];
     pH_UL_aa = mxGetPr(PAR(S))[48];
     pH_LL_aa = mxGetPr(PAR(S))[49];
     k_m_aa = mxGetPr(PAR(S))[50];
     K_S_aa = mxGetPr(PAR(S))[51];
     k_m_fa = mxGetPr(PAR(S))[52];
     K_S_fa = mxGetPr(PAR(S))[53];
```

```
    K_Ih2_fa = mxGetPr(PAR(S))[54];
    k_m_c4 = mxGetPr(PAR(S))[55];
    K_S_c4 = mxGetPr(PAR(S))[56];
    K_Ih2_c4 = mxGetPr(PAR(S))[57];
    k_m_pro = mxGetPr(PAR(S))[58];
    K_S_pro = mxGetPr(PAR(S))[59];
    K_Ih2_pro = mxGetPr(PAR(S))[60];
    pH_UL_ac = mxGetPr(PAR(S))[64];
    pH_LL_ac = mxGetPr(PAR(S))[65];
    k_m_h2 = mxGetPr(PAR(S))[66];
    K_S_h2 = mxGetPr(PAR(S))[67];
    pH_UL_h2 = mxGetPr(PAR(S))[68];
    pH_LL_h2 = mxGetPr(PAR(S))[69];
    R = mxGetPr(PAR(S))[77];
    T_base = mxGetPr(PAR(S))[78];
    T_op = mxGetPr(PAR(S))[79];
    kLa = mxGetPr(PAR(S))[94];
    K_H_h2_base = mxGetPr(PAR(S))[98];
    V_liq = mxGetPr(V(S))[0];

    K_H_h2 = K_H_h2_base*exp(-4180.0*(1.0/T_base - 1.0/T_op)/(100.0*R));
/* T adjustment for K_H_h2 */

    pH_op = *u[33];   /* pH */
    S_H_ion = *u[34]; /* SH+ */

  /* STRs function
  if (pH_op < pH_UL_aa)
   I_pH_aa = exp(-3.0*(pH_op-pH_UL_aa)*(pH_op-pH_UL_aa)/((pH_UL_aa-pH_LL_aa)*
(pH_UL_aa-pH_LL_aa)));
  else
   I_pH_aa = 1.0;
  if (pH_op < pH_UL_h2)
   I_pH_h2 = exp(-3.0*(pH_op-pH_UL_h2)*(pH_op-pH_UL_h2)/((pH_UL_h2-pH_LL_h2)*
(pH_UL_h2-pH_LL_h2)));
  else
   I_pH_h2 = 1.0;
  */

  /* Hill function on pH inhibition
  pHLim_aa = (pH_UL_aa + pH_LL_aa)/2.0;
  pHLim_ac = (pH_UL_ac + pH_LL_ac)/2.0;
  pHLim_h2 = (pH_UL_h2 + pH_LL_h2)/2.0;
  I_pH_aa = pow(pH_op,24)/(pow(pH_op,24)+pow(pHLim_aa ,24));
  I_pH_ac = pow(pH_op,45)/(pow(pH_op,45)+pow(pHLim_ac ,45));
  I_pH_h2 = pow(pH_op,45)/(pow(pH_op,45)+pow(pHLim_h2 ,45));
  */
  /* MNPs function on pH inhibition, ADM1 Workshop, Copenhagen 2005.
  a_aa = 25.0/(pH_UL_aa-pH_LL_aa+eps);
  a_ac = 25.0/(pH_UL_ac-pH_LL_ac+eps);
  a_h2 = 25.0/(pH_UL_h2-pH_LL_h2+eps);

  I_pH_aa = 0.5*(1+tanh(a_aa*(pH_op/pHLim_aa - 1.0)));
  I_pH_ac = 0.5*(1+tanh(a_ac*(pH_op/pHLim_ac - 1.0)));
```

```
   I_pH_h2 = 0.5*(1+tanh(a_h2*(pH_op/pHLim_h2 - 1.0)));
   */
   /* Hill function on SH+ used within BSM2, ADM1 Workshop, Copenhagen 2005. */
   pHLim_aa = pow(10,(-(pH_UL_aa + pH_LL_aa)/2.0));
   pHLim_h2 = pow(10,(-(pH_UL_h2 + pH_LL_h2)/2.0));
   n_aa=3.0/(pH_UL_aa-pH_LL_aa);
   n_h2=3.0/(pH_UL_h2-pH_LL_h2);
   I_pH_aa = pow(pHLim_aa,n_aa)/(pow(S_H_ion,n_aa)+pow(pHLim_aa ,n_aa));
   I_pH_h2 = pow(pHLim_h2,n_h2)/(pow(S_H_ion,n_h2)+pow(pHLim_h2 ,n_h2));

    /* Gradient of Sh2 equation */
    return -1/V_liq**u[26]
            -3.0/10.0*(1-Y_fa)*k_m_fa**u[2]/(K_S_fa+*u[2])*
            *u[18]*I_pH_aa/(1+K_S_IN/(*u[10]))/((1+x[0]/K_Ih2_fa)*
            (1+x[0]/K_Ih2_fa))/K_Ih2_fa -3.0/20.0*(1-Y_c4)*k_m_c4*
            *u[3]**u[3]/(K_S_c4+*u[3])**u[19]/(*u[4]+*u[3]+eps)*
            I_pH_aa/(1+K_S_IN/(*u[10]))/((1+x[0]/K_Ih2_c4)*
            (1+x[0]/K_Ih2_c4))/K_Ih2_c4-1.0/5.0*(1-Y_c4)*k_m_c4**u[4]*
            *u[4]/(K_S_c4+*u[4])**u[19]/(*u[4]+*u[3]+eps)*
            I_pH_aa/(1+K_S_IN/(*u[10]))/((1+x[0]/K_Ih2_c4)*
            (1+x[0]/K_Ih2_c4))/K_Ih2_c4-43.0/100.0*(1-Y_pro)*k_m_pro**u[5]/
            (K_S_pro+*u[5])**u[20]*I_pH_aa/(1+K_S_IN/(*u[10]))/
            ((1+x[0]/K_Ih2_pro)*(1+x[0]/K_Ih2_pro))/K_Ih2_pro
            -k_m_h2/(K_S_h2+x[0])**u[22]*I_pH_h2/(1+K_S_IN/
            (*u[10]))+k_m_h2*x[0]/((K_S_h2+x[0])*(K_S_h2+x[0]))*
            *u[22]*I_pH_h2/(1+K_S_IN/(*u[10]))-kLa;
}

static void Sh2solver(SimStruct *S)
{

    real_T *x = ssGetDiscStates(S);

    static real_T delta;
    static real_T Sh20;
    static int_T i;

    static const real_T TOL = 1e-12;
    static const real_T MaxSteps= 1000;

    Sh20 = x[0]; /* Sh2 */

    i = 1;
    delta = 1.0;

    /* Newton-Raphson algorithm */

    while ( (delta > TOL || delta < -TOL) && (i <= MaxSteps) ) {
        delta = Equ(S);
        x[0] = Sh20-delta/gradEqu(S);     /* Calculate the new Sh2 */

        if (x[0] <= 0) {
            x[0] = 1e-12; /* to avoid numerical problems */
        }
```

```
        Sh20 = x[0];
        ++i;
    }
}


#define MDL_UPDATE /* Change to #undef to remove function */
#if defined(MDL_UPDATE)
/*
 * mdlUpdate:
 * 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)
{
   Sh2solver(S);
}
#endif /* MDL_UPDATE */


#undef MDL_DERIVATIVES /* Change to #undef to remove function */
#if defined(MDL_DERIVATIVES)
/*
 * mdlDerivatives:
 * 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 */


/*
 * mdlTerminate:
 * 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)
{
}


#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
```