Aspects on ADM1 Implementation within the BSM2 Framework

Rosen, C. and Jeppsson, U.

Summary

This document presents detailed information regarding numerous implementation issues with respect to the Anaerobic Digestion Model No 1 (ADM1). A number of enhancements and modifications to the original ADM1 are suggested based on the extensive experiences gained when including the ADM1 into the framework of the IWA Benchmark Simulation Model No 2 (BSM2). However, the report is also valid when using the ADM1 as a stand-alone model. It summarizes all necessary information – some of which is not easily accessible in the ADM1 STR – needed to implement the model. Examples and computer code for Matlab/Simulink is also provided. Aspects related to ODE (ordinary differential equation) vs DAE (differential algebraic equation) implementations, system stiffness and varying time constants, mass balances, acid-base equilibrium and algebraic solvers for pH and other troublesome state variables, numerical solvers and simulation time are discussed. A main conclusion is that if implemented properly, the enhanced ADM1 will produce high-quality results also in dynamic plant-wide simulations including noise, discrete sub-systems, etc. without imposing any major restrictions due to extensive computational efforts.

Copyright © 2008 Christian Rosen and Ulf Jeppsson

Table of Contents

1. INTRODUCTION

This report describes the development of the Lund university 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, in contrast 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 into the IWA Benchmark Simulation Model (BSM) to form a plant-wide or "within-the-fence" model, i.e. BSM2.

2. THE IWA BENCHMARK SIMULATION MODELS

2.1. BSM1

The original COST benchmark system was 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” (Copp, 2002). 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 uses 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, aeration, etc. 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 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 prolonged 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; 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; 2007).

3. ADM1 ODE IMPLEMENTATION

In this section, the ordinary differential equation (ODE) model implementation of the ADM1 in BSM2 is presented.

3.1. Introduction

It should be noted that the issues discussed in this report are sometimes referred to as Matlab/Simulink issues but it is the Matlab/Simulink model implementation that was later used as the blueprint for the consequent implementations of the ADM1 on the other BSM2 platforms (WEST, SIMBA, GPS-X and Fortran). The ADM1 implementation in Matlab/Simulink (BSM2) 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 full 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 Task Group (TG) had access to the "official" ADM1 implementation in Aquasim, which was developed and used by the ADM TG during their development of the model, the unknown parameter values were first chosen in accordance with the suggestions given there.

3.2. Elemental balances

To maintain complete elemental 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), see Appendix 2 for the complete ADM1 Petersen matrix. Assuming one COD mass unit of X_c completely disintegrating will produce:

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

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^{-1} COD. If we instead calculate the nitrogen content of the disintegration products (kmole N.kg^{-1} COD) using parameter values from Batstone *et al.* (2002), we get:

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

(note that carbohydrates and lipids contain no nitrogen). In Table 1 (Section 6.1) the values of all parameters are listed. 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 elemental balances. Therefore, we suggest new values for $f_{XI,xc}$ = 0.2 and $f_{li,xc}$ = 0.3. There are numerous ways by which the balance may be closed but the above way was considered a simple and reasonable solution and based on discussions with the ADM TG. For the specific BSM2 implementation we have also modified N_I to 0.06/14 ≈ 0.00429 kmole N.kg^{-1} 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^{-1} COD. Because of the latter modification N_{xc} is set to 0.0376/14 ≈ 0.00269 kmole N.kg^{-1} 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^{-1} 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. In principle this means adding stoichiometry expressions into all empty cells of the Petersen matrix for variables 10 and 11 (see Section 3.5.3 and Appendix 2). Basically all the required information are 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. This modification represents a change to the original ADM1.

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^{-1} 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, i.e. $f_{XI,xc}$ = 0.2 and $f_{li,xc}$ = 0.3) we get:

$$0.03 \cdot 0.1(S_I) + 0.03 \cdot 0.2(X_I) + 0.0313 \cdot 0.2(X_{ch}) + 0.03 \cdot 0.2(X_{pr}) + 0.022 \cdot 0.3(X_{li}) = 0.02786 \text{ kmole C.kg}^{-1} \text{ COD}$$

The parameter values indicating the carbon content of the different COD fractions are described in Table 1 (Section 6.1). If the original fractionation of composite material is used (i.e. $f_{XI,xc}$ = 0.25 and $f_{li,xc}$ = 0.25) we instead get a carbon content of the disintegrated products equal to 0.02826 kmole C.kg^{-1} 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 a somewhat low percentage of CH_4 (55-58%) in the produced gas using a realistic sludge input. Note that the mass of produced CH_4 is still reasonable as this carbon unbalance leads to higher gas flow rates due to excess CO_2.

To avoid the above problem (i.e. to achieve a realistic percentage of CH_4 in the gas *and* a realistic gas flow rate) it is suggested to use a value of 0.02786 kmole C.kg^{-1} COD to describe the carbon content of composite material

in the benchmark implementation (BSM2). For reasonable sludge inputs this modification will normally lead to a production of 60-65% methane in the gas phase. If different parameter values are chosen the above principle should be used to calculate a correct value of the carbon content of composite material.

3.3. 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 (see Sub-section 4.1.5 for a short definition of ODE vs DAE models). 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) (see Table B.4 of the ADM1 STR) 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 of S_{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 an 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 $k_{A,Bi}$ (where index *i* indicates any acid-base, i.e. valerate, butyrate, propionate, acetate, inorganic carbon and inorganic nitrogen) 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 1.10^8 $M^{-1}.d^{-1}$ 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 1.10^{10} $M^{-1}.d^{-1}$ is more appropriate.

3.4. Temperature dependencies

In order to better allow for reasonable results for different temperatures within the digester, the benchmark ADM1 implementation uses the complete information as stated in the ADM1 STR with regard to temperature dependency of several physico-chemical parameters (see the Table 3 in Section 6.3 for physico-chemical 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 constants (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. However, such a modification falls outside the scope of the work of BSM TG.

3.5. Model equations

In the sub-sections below all the required equations for a full ODE implementation of ADM1 are given.

3.5.1. Process rates

The biochemical process rates are defined below.

Disintegration:

$$\rho_1 = k_{\text{dis}} \cdot X_{\text{c}}$$

Hydrolysis of carbohydrates:

$$\rho_2 = k_{\text{hyd,ch}} \cdot X_{\text{ch}}$$

Hydrolysis of proteins:

$$\rho_3 = k_{\text{hyd,pr}} \cdot X_{\text{pr}}$$

Hydrolysis of lipids:

$$\rho_4 = k_{\text{hyd,li}} \cdot X_{\text{li}}$$

Uptake of sugars:

$$\rho_5 = k_{\text{m,su}} \cdot \frac{S_{\text{su}}}{K_{\text{S,su}} + S_{\text{su}}} \cdot X_{\text{su}} \cdot I_5$$

Uptake of amino-acids:

$$\rho_6 = k_{\text{m,aa}} \cdot \frac{S_{\text{aa}}}{K_{\text{S,aa}} + S_{\text{aa}}} \cdot X_{\text{aa}} \cdot I_6$$

Uptake of LCFA (long-chain fatty acids):

$$\rho_7 = k_{\text{m,fa}} \cdot \frac{S_{\text{fa}}}{K_{\text{S,fa}} + S_{\text{fa}}} \cdot X_{\text{fa}} \cdot I_7$$

Uptake of valerate:

$$\rho_8 = k_{\text{m,c4}} \cdot \frac{S_{\text{va}}}{K_{\text{S,c4}} + S_{\text{va}}} \cdot X_{\text{c4}} \cdot \frac{S_{\text{va}}}{S_{\text{bu}} + S_{\text{va}}} \cdot I_8$$

Uptake of butyrate:

$$\rho_9 = k_{\text{m,c4}} \cdot \frac{S_{\text{bu}}}{K_{\text{S,c4}} + S_{\text{bu}}} \cdot X_{\text{c4}} \cdot \frac{S_{\text{bu}}}{S_{\text{va}} + S_{\text{bu}}} \cdot I_9$$

Uptake of propionate:

$$\rho_{10} = k_{\text{m,pro}} \cdot \frac{S_{\text{pro}}}{K_{\text{S,pro}} + S_{\text{pro}}} \cdot X_{\text{pro}} \cdot I_{10}$$

Uptake of acetate:

$$\rho_{11} = k_{\text{m,ac}} \cdot \frac{S_{\text{ac}}}{K_{\text{S,ac}} + S_{\text{ac}}} \cdot X_{\text{ac}} \cdot I_{11}$$

Uptake of hydrogen:

$$\rho_{12} = k_{\text{m,h2}} \cdot \frac{S_{\text{h2}}}{K_{\text{S,h2}} + S_{\text{h2}}} \cdot X_{\text{h2}} \cdot I_{12}$$

Decay of X_{su}:

$$\rho_{13} = k_{\text{dec,Xsu}} \cdot X_{\text{su}}$$

Decay of X_{aa}:

$$\rho_{14} = k_{\text{dec,Xaa}} \cdot X_{\text{aa}}$$

Decay of X_{fa}:

$$\rho_{15} = k_{\text{dec\_Xfa}} \cdot X_{\text{fa}}$$

Decay of X_{c4}:

$$\rho_{16} = k_{dec,Xc4} \cdot X_{c4}$$

Decay of X_{pro}:

$$\rho_{17} = k_{dec,Xpro} \cdot X_{pro}$$

Decay of X_{ac}:

$$\rho_{18} = k_{dec,Xac} \cdot X_{ac}$$

Decay of X_{h2}:

$$\rho_{19} = k_{dec,Xh2} \cdot X_{h2}$$

In the expressions for ρ_8 and ρ_9, a small constant (1.10^{-6}) should be been added at the denominator to the sum ($S_{va} + S_{bu}$) in order to avoid division by zero in the case of poor choice of initial conditions for S_{va} and S_{bu}, respectively. Apart from this, the above equations are identical to the original ADM1.

The acid-base rates for the BSM2 ODE implementation (see Chapter 4 and Appendix 1 for the corresponding DAE implementation) are as follows:

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

$$\rho_{A,5} = k_{A,Bbu}\left(S_{bu^-}\left(K_{a,bu} + S_{H^+}\right) - K_{a,bu}S_{bu}\right)$$

$$\rho_{A,6} = k_{A,Bpro}\left(S_{pro^-}\left(K_{a,pro} + S_{H^+}\right) - K_{a,pro}S_{pro}\right)$$

$$\rho_{A,7} = k_{A,Bac}\left(S_{ac^-}\left(K_{a,ac} + S_{H^+}\right) - K_{a,ac}S_{ac}\right)$$

$$\rho_{A,10} = k_{A,Bco2}\left(S_{hco3^-}\left(K_{a,co2} + S_{H^+}\right) - K_{a,co2}S_{IC}\right)$$

$$\rho_{A,11} = k_{A,BIN}\left(S_{nh3}\left(K_{a,IN} + S_{H^+}\right) - K_{a,IN}S_{IN}\right)$$

The modifications of the above equations compared to the original ADM1 was discussed in Section 3.3. The gas transfer rates (note that S_{co2} is used in the expression for $\rho_{T,10}$, not S_{IC}) are defined as:

$$\rho_{T,8} = K_L a\left(S_{h2} - 16 \cdot K_{H,h2} p_{gas,h2}\right)$$

$$\rho_{T,9} = K_L a\left(S_{ch4} - 64 \cdot K_{H,ch4} p_{gas,ch4}\right)$$

$$\rho_{T,10} = K_L a\left(S_{co2} - K_{H,co2} p_{gas,co2}\right)$$

3.5.2. Process inhibition

The general process inhibition terms are expressed as:

$$I_5 = I_6 = I_{pH,aa} \cdot I_{IN,lim}$$

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

$$I_8 = I_9 = I_{pH,aa} \cdot I_{IN,lim} \cdot I_{h2,c4}$$

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

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

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

$$I_{\text{pH,aa}} = \begin{cases} \exp\left(-3\left(\dfrac{pH - pH_{\text{UL,aa}}}{pH_{\text{UL,aa}} - pH_{\text{LL,aa}}}\right)^2\right) & : \; pH < pH_{\text{UL,aa}} \\ 1 & : \; pH > pH_{\text{UL,aa}} \end{cases}$$

$$I_{\text{pH,ac}} = \begin{cases} \exp\left(-3\left(\dfrac{pH - pH_{\text{UL,ac}}}{pH_{\text{UL,ac}} - pH_{\text{LL,ac}}}\right)^2\right) & : \; pH < pH_{\text{UL,ac}} \\ 1 & : \; pH > pH_{\text{UL,ac}} \end{cases}$$

$$I_{\text{pH,h2}} = \begin{cases} \exp\left(-3\left(\dfrac{pH - pH_{\text{UL,h2}}}{pH_{\text{UL,h2}} - pH_{\text{LL,h2}}}\right)^2\right) & : \; pH < pH_{\text{UL,h2}} \\ 1 & : \; pH > pH_{\text{UL,h2}} \end{cases}$$

$$I_{\text{IN,lim}} = \frac{1}{1 + K_{\text{S,IN}} / S_{\text{IN}}}$$

$$I_{\text{h2,fa}} = \frac{1}{1 + S_{\text{h2}} / K_{\text{I,h2,fa}}}$$

$$I_{\text{h2,c4}} = \frac{1}{1 + S_{\text{h2}} / K_{\text{I,h2,c4}}}$$

$$I_{\text{h2,pro}} = \frac{1}{1 + S_{\text{h2}} / K_{\text{I,h2,pro}}}$$

$$I_{\text{nh3}} = \frac{1}{1 + S_{\text{nh3}} / K_{\text{I,nh3}}}$$

where

$$pH = -\log_{10}\left(S_{\text{H}^+}\right)$$

Batstone *et al.* (2002) use switching functions to account for inhibition due to pH. These functions are, however, discontinuous (as seen above) 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 frequently used instead, e.g.:

$$I_{\text{pH}} = \frac{1}{2}\left(1 + \tan\left(a \cdot \varphi\right)\right)$$

where

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

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

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

where

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

Siegrist *et al.* (2002) use a Hill inhibition function based on the hydrogen ion concentration instead. *This solution*

has been chosen for the ADM1 in BSM2. For the ADM1, this gives the following expressions:

$$I_{\mathrm{pH,aa}} = \frac{K_{\mathrm{pH}}^{n_{\mathrm{aa}}}}{S_{\mathrm{H^+}}^{n_{\mathrm{aa}}} + K_{\mathrm{pH}}^{n_{\mathrm{aa}}}} \text{ with } K_{\mathrm{pH}} = 10^{-\frac{pH_{\mathrm{LL,aa}} + pH_{\mathrm{UL,aa}}}{2}} \text{ and } n_{\mathrm{aa}} = \frac{3.0}{pH_{\mathrm{UL,aa}} - pH_{\mathrm{LL,aa}}}$$

$$I_{\mathrm{pH,ac}} = \frac{K_{\mathrm{pH}}^{n_{\mathrm{ac}}}}{S_{\mathrm{H^+}}^{n_{\mathrm{ac}}} + K_{\mathrm{pH}}^{n_{\mathrm{ac}}}} \text{ with } K_{\mathrm{pH}} = 10^{-\frac{pH_{\mathrm{LL,ac}} + pH_{\mathrm{UL,ac}}}{2}} \text{ and } n_{\mathrm{ac}} = \frac{3.0}{pH_{\mathrm{UL,ac}} - pH_{\mathrm{LL,ac}}}$$

$$I_{\mathrm{pH,h2}} = \frac{K_{\mathrm{pH}}^{n_{\mathrm{h2}}}}{S_{\mathrm{H^+}}^{n_{\mathrm{h2}}} + K_{\mathrm{pH}}^{n_{\mathrm{h2}}}} \text{ with } K_{\mathrm{pH}} = 10^{-\frac{pH_{\mathrm{LL,h2}} + pH_{\mathrm{UL,h2}}}{2}} \text{ and } n_{\mathrm{h2}} = \frac{3.0}{pH_{\mathrm{UL,h2}} - pH_{\mathrm{LL,h2}}}$$

The appropriate values of n in the respective types of Hill functions above are quite different. To fit the original function given in Batstone *et al.* (2002), n = 24 for $I_{\mathrm{pH,aa}}$ when the pH-based Hill function is used and n = 2 for the hydrogen ion-based function. For $I_{\mathrm{pH,ac}}$ and $I_{\mathrm{pH,h2}}$ the respective values of n are 45 and 3. Since the appropriate values of n are dependent on the values of pH_{LL} and pH_{UL}, it is wise to implement n as exemplified above for the hydrogen ion-based solution. If the value of pH_{LL} or pH_{UL} is changed then the value of n will automatically be corrected.

For any practical purpose, the choice of function among the three principle ones discussed above is arbitrary. However, for the BSM2 implementation the solution according to Siegrist *et al.* (2002) has been chosen.

3.5.3. Water phase equations

The influent liquid flow rate to the anaerobic digester is denoted Q_{ad} below. All the differential equations are explicitly stated below although they can obviously be summarized in the general form, i.e.:

$$\frac{dS_{\mathrm{i}}}{dt} = \frac{Q_{\mathrm{ad}}}{V_{\mathrm{ad,liq}}}\left(S_{\mathrm{i,in}} - S_{\mathrm{i}}\right) + \sum_{j=1}^{19} \nu_{\mathrm{i,j}}\rho_{\mathrm{j}} + \text{transport terms; where } i = 1\ldots12$$

$$\frac{dX_{\mathrm{i}}}{dt} = \frac{Q_{\mathrm{ad}}}{V_{\mathrm{ad,liq}}}\left(X_{\mathrm{i,in}} - X_{\mathrm{i}}\right) + \sum_{j=1}^{19} \nu_{\mathrm{i,j}}\rho_{\mathrm{j}} + \text{transport terms; where } i = 13\ldots24$$

and the terms ν and ρ defined in a traditional Petersen matrix format. The complete water-phase equations are written as:

Differential equations 1 to 12 (soluble matter) State no.

$$\frac{dS_{\mathrm{su}}}{dt} = \frac{Q_{\mathrm{ad}}}{V_{\mathrm{ad,liq}}}\left(S_{\mathrm{su,i}} - S_{\mathrm{su}}\right) + \rho_2 + \left(1 - f_{\mathrm{fa,li}}\right)\rho_4 - \rho_5 \quad (1)$$

$$\frac{dS_{\mathrm{aa}}}{dt} = \frac{Q_{\mathrm{ad}}}{V_{\mathrm{ad,liq}}}\left(S_{\mathrm{aa,i}} - S_{\mathrm{aa}}\right) + \rho_3 - \rho_6 \quad (2)$$

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

$$\frac{dS_{\mathrm{va}}}{dt} = \frac{Q_{\mathrm{ad}}}{V_{\mathrm{ad,liq}}}\left(S_{\mathrm{va,i}} - S_{\mathrm{va}}\right) + \left(1 - Y_{\mathrm{aa}}\right)f_{\mathrm{va,aa}}\rho_6 - \rho_8 \quad (4)$$

$$\frac{dS_{\mathrm{bu}}}{dt} = \frac{Q_{\mathrm{ad}}}{V_{\mathrm{ad,liq}}}\left(S_{\mathrm{bu,i}} - S_{\mathrm{su}}\right) + \left(1 - Y_{\mathrm{su}}\right)f_{\mathrm{bu,su}}\rho_5 + \left(1 - Y_{\mathrm{aa}}\right)f_{\mathrm{bu,aa}}\rho_6 - \rho_9 \quad (5)$$

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

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

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

$$\frac{dS_{ch4}}{dt} = \frac{Q_{ad}}{V_{ad,liq}}\left(S_{ch4,i} - S_{ch4}\right) + \left(1 - Y_{ac}\right)\rho_{11} + \left(1 - Y_{h2}\right)\rho_{12} - \rho_{T,9} \quad (9)$$

$$\frac{dS_{IC}}{dt} = \frac{Q_{ad}}{V_{ad,liq}}\left(S_{IC,i} - S_{IC}\right) - \sum_{j=1}^{19}\left(\sum_{k=1-9,11-24} C_k \nu_{k,j}\rho_j\right) - \rho_{T,10} \quad (10)$$

see below

$$\frac{dS_{IN}}{dt} = \frac{Q_{ad}}{V_{ad,liq}}\left(S_{IN,i} - S_{IN}\right) - Y_{su}N_{bac}\rho_5 + \left(N_{aa} - Y_{aa}N_{bac}\right)\rho_6 - Y_{fa}N_{bac}\rho_7 - Y_{c4}N_{bac}\rho_8 - Y_{c4}N_{bac}\rho_9 - Y_{pro}N_{bac}\rho_{10} - Y_{ac}N_{bac}\rho_{11} - Y_{h2}N_{bac}\rho_{12} + \left(N_{bac} - N_{xc}\right)\sum_{k=13}^{19}\rho_k + \left(N_{xc} - f_{xi,xc}N_I - f_{si,xc}N_I - f_{pr,xc}N_{aa}\right)\rho_1 \quad (11)$$

$$\frac{dS_I}{dt} = \frac{Q_{ad}}{V_{ad,liq}}\left(S_{I,i} - S_I\right) + f_{si,xc}\rho_1 \quad (12)$$

(Note that state equation (8) is not used in the final DAE implementation of ADM1, see Sub-section 4.2.2 and Appendix 1.2 for details.)

More specifically, the sum in the equation for state number 10 is calculated as (explicitly given below for clarity):

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

where

$$s_1 = -C_{xc} + f_{sI,xc}C_{sI} + f_{ch,xc}C_{ch} + f_{pr,xc}C_{pr} + f_{li,xc}C_{li} + f_{xI,xc}C_{xI}$$

$$s_2 = -C_{ch} + C_{su}$$

$$s_3 = -C_{pr} + C_{aa}$$

$$s_4 = -C_{li} + \left(1 - f_{la,li}\right)C_{su} + f_{fa,li}C_{fa}$$

$$s_5 = -C_{su} + \left(1 - Y_{su}\right)\left(f_{bu,su}C_{bu} + f_{pro,su}C_{pro} + f_{ac,su}C_{ac}\right) + Y_{su}C_{bac}$$

$$s_6 = -C_{aa} + \left(1 - Y_{aa}\right)\left(f_{va,aa}C_{va} + f_{bu,aa}C_{bu} + f_{pro,aa}C_{pro} + f_{ac,aa}C_{ac}\right) + Y_{aa}C_{bac}$$

$$s_7 = -C_{fa} + \left(1 - Y_{fa}\right)0.7C_{ac} + Y_{fa}C_{bac}$$

$$s_8 = -C_{va} + \left(1 - Y_{c4}\right)0.54C_{pro} + \left(1 - Y_{c4}\right)0.31C_{ac} + Y_{c4}C_{bac}$$

$$s_9 = -C_{bu} + (1 - Y_{c4})\,0.8C_{ac} + Y_{c4}C_{bac}$$

$$s_{10} = -C_{pro} + (1 - Y_{pro})\,0.57C_{ac} + Y_{pro}C_{bac}$$

$$s_{11} = -C_{ac} + (1 - Y_{ac})C_{ch4} + Y_{ac}C_{bac}$$

$$s_{12} = (1 - Y_{h2})C_{ch4} + Y_{h2}C_{bac}$$

$$s_{13} = -C_{bac} + C_{xc}$$

Differential equations 13 to 24 (particulate matter) State no.

$$\frac{dX_c}{dt} = \frac{Q_{ad}}{V_{ad,liq}}(X_{c,i} - X_c) - \rho_1 + \rho_{13} + \rho_{14} + \rho_{15} + \rho_{16} + \rho_{17} + \rho_{18} + \rho_{19} \quad (13)$$

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

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

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

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

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

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

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

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

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

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

$$\frac{dX_I}{dt} = \frac{Q_{ad}}{V_{ad,liq}}(X_{I,i} - X_I) + f_{xi\_xc}\rho_1 \quad (24)$$

Differential equations 25 and 26 (cations and anions) State no.

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

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

Differential equations 27 to 32 (ion states, only for ODE implementation) State no.

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

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

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

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

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

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

Note that the differential equations for states 27-32 are not used in either of the DAE implementations. Instead the ion states are then calculated using algebraic equations. However, an algebraic equation for the pH calculation is used also for the ODE implementation as:

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

where

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

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

The above equation for calculating S_{H+} is based on the general charge balance in ADM1, i.e.:

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

where

$S_{OH^-} - \frac{K_W}{S_{H^+}} = 0$ and K_W is given in Table 3 (Section 6.3). Substituting S_{OH-} by K_W/S_{H+} in the above equation and rewriting it yields a quadratic expression of S_{H+}, which can be easily solved analytically since all other variables are known. Obviously the negative root is not a valid solution.

We also calculate the concentration of dissolved carbon dioxide algebraically as:

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

3.5.4. Gas phase equations

Differential equations 33 to 35 describe the fate of the gas phase components: State no.

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

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

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

where Q_{gas} denote the gas flow rate. The associated algebraic equations required for the gas transfer rates $\rho_{T,8}$, $\rho_{T,9}$ and $\rho_{T,10}$ are:

$$p_{gas,h2} = S_{gas,h2} \cdot \frac{RT_{ad}}{16}$$

$$p_{gas,ch4} = S_{gas,ch4} \cdot \frac{RT_{ad}}{64}$$

$$p_{gas,co2} = S_{gas,co2} \cdot RT_{ad}$$

$$Q_{gas} = \frac{RT_{ad}}{P_{atm} - p_{gas,h2o}} \cdot V_{ad,liq} \cdot \left(\frac{\rho_{T,8}}{16} + \frac{\rho_{T,9}}{64} + \rho_{T,10} \right)$$

where T_{ad} denote the operational temperature of the digester. The gas flow rate and partial pressures are obviously essential output variables from the model. The expression for $p_{gas,h2o}$ is given in Table 3 (Section 6.3). A potential problem when using the above manner for calculating the gas flow rate is that is may give rise to some numerical problems in the solution of the equations. Multiple steady states as well as numerical instability have been reported among some users. An alternative way of calculating the gas flow rate is also given in Batstone *et al.* (2002):

$$Q_{gas} = k_p \left(P_{gas} - P_{atm} \right)$$

where $P_{gas} = p_{gas,h2} + p_{gas,ch4} + p_{gas,co2} + p_{gas,h2o}$

The alternative expression assumes an overpressure in the head space. Note that the equations for $p_{gas,h2}$, $p_{gas,ch4}$, $p_{gas,co2}$ and $p_{gas,h2o}$ are identical for both ways of calculating Q_{gas}. Consequently, the gas flow rate is calculated at a slightly higher pressure compared to the first expression. To compensate for this, the expression needs to rewritten into

$$Q_{gas} = k_p \left(P_{gas} - P_{atm} \right) \cdot \frac{P_{gas}}{P_{atm}}$$

to obtain the gas 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 gas 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 1.10^{-5}). For BSM2, the alternative way (assuming an overpressure in the head space) of calculating the gas flow rate is used. Also note that if the physical or operational conditions of the digester model

are changed (volume, load etc.), for example if applying the ADM1 as a stand-alone model outside the framework of BSM2, then the parameter k_p will have to be adjusted to achieve a reasonable overpressure in the head space.

4. ADM1 DAE IMPLEMENTATION

It has been realized that the ODE implementation is problematic for use in the BSM2 framework due to the computational effort. Therefore, two DAE versions have been developed (Rosen *et al.*, 2006). In this section, the differential algebraic equation model implementations of ADM1 are presented. Two different DAE models are discussed: a model based on algebraic pH (S_{H+}) calculations (DAE_{pH}) and a model based on 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. Motivation

The fact that the Matlab/Simulink ADM1 implementation discussed here aims at being 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 of controlling a WWTP lies mainly in the changing influent wastewater characteristics. The generation of dynamic input is, thus, an integrated part of BSM2. This means that, except in order to obtain initial conditions, BSM2 is basically 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; 2006).

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; 2007). From a numerical point of view, continuous controllers yield the least computational effort. However, discrete controllers are more common and many commercially available sensors are also discrete in nature. 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. Numerical solvers designed to handle stiff systems do not work well on hybrid systems.

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. Some of the most important sources of errors are sensor noise and time 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 the applied noise is uncorrelated. Numerical solvers designed to handle stiff systems do not work well on systems affected by noise.

4.1.4. Simulating stiff systems in Matlab/Simulink

A system is called stiff, when the range of the model 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 time, one needs to be somewhat creative when implementing the model.

Some of the numerical solvers in Matlab/Simulink are so called stiff solvers and, consequently, capable of solving stiff systems efficiently. However, a problem common to all stiff solvers is the difficulty to handle dynamic input – including noise as well as hybrid systems. The more stochastic or random an input variable behaves, the more problematic is the numerical 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 states' 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 covering 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 in order to enhance the performance when simulating the ADM1 using a explicit solver.

4.2. DAE equations

4.2.1. pH solver

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 our case, the Newton-Raphson method used in Volcke (2006) for calculation of the pH and equilibrium concentrations (i.e. the acid-base equations (states 27-32) in Sub-section 3.5.3) was implemented. By using this method the new value of the unknown state is calculated at each iteration step *k* as:

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

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.:

$$E\left(S_{H^+,k}\right) = S_{cat^+,k} + S_{nh4^+,k} + S_{H^+,k} - S_{hco3^-,k} - \frac{S_{ac^-,k}}{64} - \frac{S_{pro^-,k}}{112} - \frac{S_{bu^-,k}}{160} - \frac{S_{va^-,k}}{208} - \frac{K_W}{S_{H^+,k}} - S_{an^-,k}$$

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 complex it is not stated here. The reader is referred to Appendix 1.1 or Volcke (2006) 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 1.10^{-12}. Normally only two or three iterations are required to solve the equation at each time step.

4.2.2. S_{h2} solver

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

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

and the gradient of $E(S_{h2,k+1})$. The expression of the gradient is fairly complex and the reader is referred to Appendix 1.2 for exact details of the mathematical expression. For the interested reader to obtain the gradients for the S_{H+} and S_{h2} equations on his/her own, it is recommended that a tool for handling mathematics symbolically is used (e.g. Maple or Mathematica). Generic expressions for each term are also listed in Volcke (2006).

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, and are, very small (close to machine numerical precision). The differences during dynamic conditions have been studied extensively as part of the benchmarking work and within other projects and have been found to be fully acceptable (some small differences are unavoidable due to the mathematics and numerics of the ODE and DAE implementations). However, it is a good strategy to always make a dynamic simulation using the ODE implementation and comparing the results with those from any of the DAE implementations when applying the ADM1 models to different input data and different operating regions than what is used within the benchmark system. Thereby any potential (and unforeseen) problems can be identified at an early stage.

5.2. Steady-state comparison

The three model implementations discussed in this technical report, i.e. ODE, DAE with only a pH solver (DAE1) and DAE with a pH solver and a S_{h2} solver (DAE2), 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 1.10^{-6}. The largest absolute errors, around 1.10^{-5}, were found for the states with large steady-state values (no scaling of states in the used 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 was run to create sensible input data for the digester. The simulation period was 50 days and included recycling streams from the digester. Thus, the digester was included in the model when data was produced. The input data were stored at 15-minute sampling intervals.

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

The fact that the mean errors were in the range of 1.10^{-4} when each sample time was 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 reasons discussed earlier in conjunction with the choice of gas flow rate expression. Also here, the DAE1 and 2 behaved similar in terms of dynamic errors.

NOTE! The choice between the three implementations of ADM1 – ODE, DAE1 and DAE2 – is up to the user. If acceptable computation times can be achieved with the ODE or DAE1 implementations there is no other advantage in using DAE2. However, for Matlab/Simulink it appears that with currently available solvers, DAE2 is the only practically feasible choice for BSM2.

6. ADM1 BENCHMARK MODEL PARAMETERS

In the sections below the ADM1 parameters used for the BSM2 implementation are presented. In cases where the parameter value in ADM1 for BSM2 differs from the default ADM1 parameter value (Batstone *et al.*, 2002) the explicit row is marked in grey and the default value is given in the commentary field. In a few cases it is not really possible to determine explicitly what the ADM1 suggested default parameter values are, since several possible values may be stated (depending on temperature, different references, etc.) or no value at all is given (in some cases the selected values below are based on the original ADM1 implementation in AquaSim by the ADM1 Task Group). Also, in cases where different options have been given for mesophilic high rate, mesophilic solids and thermophilic solids AD systems in Batstone *et al.* (2002), the values below refer to mesophilic solids systems.

Obviously many of the parameter values are application specific and should, if possible, be determined or estimated based on measured data. However, for the purpose of the BSM2, the values suggested below produce satisfactory results. It should be noted that the parameter values for the hydrolysis rates (10 d^{-1}) used in both ADM1-BSM2 and given as default parameter values in the ADM1 STR are nowadays considered to be at least ten times too large (Batstone 2002-2008, personal communication).

6.1. Stoichiometric parameter values

Table 1: Stoichiometric parameter values for the BSM2 ADM1 implementation

Parameter	Value	Unit	Process(es)	Comments
$f_{SI,xc}$	0.1	–	1	
$f_{XI,xc}$	0.2	–	1	ADM1 default value = 0.25
$f_{ch,xc}$	0.2	–	1	
$f_{pr,xc}$	0.2	–	1	
$f_{li,xc}$	0.3	–	1	ADM1 default value = 0.25 Note: $1 - f_{ch,xc} - f_{pr,xc} - f_{SI,xc} - f_{li,xc} - f_{XI,xc} = 0$
N_{xc}	0.0376/14	–	1, 13-19	ADM1 default value = 0.002 0.0376/14: to maintain N balance for disintegration, see Section 3.1.1.
N_I	0.06/14	kmole N.kg^{-1} COD	1	ADM1 default value = 0.002 Here: 6% on weight basis based on ASM1
N_{aa}	0.007	kmole N.kg^{-1} COD	1, 6	
C_{xc}	0.02786	kmole C.kg^{-1} COD	1, 13-19	C_{13} in state equation 10 Not stated in ADM1 STR but value of 0.03 used in original AquaSim implementation 0.02786: to maintain C balance for disintegration, see Section 3.1.1.
C_{SI}	0.03	kmole C.kg^{-1} COD	1	C_{12} in state equation 10 Not stated in ADM1 STR but value of 0.03 used in original TG AquaSim implementation
C_{ch}	0.0313	kmole C.kg^{-1} COD	1, 2	C_{14} in state equation 10
C_{pr}	0.03	kmole C.kg^{-1} COD	1, 3	C_{15} in state equation 10 Not stated in ADM1 STR but value of 0.03 used in original TG AquaSim implementation
C_{li}	0.022	kmole C.kg^{-1} COD	1, 4	C_{16} in state equation 10
C_{XI}	0.03	kmole C.kg^{-1} COD	1	C_{24} in state equation 10 Not stated in ADM1 STR but value of 0.03 used in original TG AquaSim implementation
C_{su}	0.0313	kmole C.kg^{-1} COD	2, 5	C_1 in state equation 10
C_{aa}	0.03	kmole C.kg^{-1} COD	3, 6	C_2 in state equation 10

				Not stated in ADM1 STR but value of 0.03 used in original TG AquaSim implementatio
$f_{fa,li}$	0.95	–	4	
C_{fa}	0.0217	kmole C.kg^{-1} COD	4, 7	C_3 in state equation 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^{-1} COD	5-19	ADM1 default value = 0.00625 Here: 8% on weight basis based on ASM1
C_{bu}	0.025	kmole C.kg^{-1} COD	5, 6, 9	C_5 in state equation 10
C_{pro}	0.0268	kmole C.kg^{-1} COD	5, 6, 8, 10	C_6 in state equation 10
C_{ac}	0.0313	kmole C.kg^{-1} COD	5-11	C_7 in state equation 10
C_{bac}	0.0313	kmole C.kg^{-1} COD	5-19	$C_{17–23}$ in state equation 10
Y_{su}	0.1	–	5	kmole COD_X.kg^{-1} COD_S
$f_{h2,aa}$	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^{-1} COD	6, 8	C_4 in state equation 10
Y_{aa}	0.08	–	6	kmole COD_X.kg^{-1} COD_S
Y_{fa}	0.06	–	7	kmole COD_X.kg^{-1} COD_S
Y_{c4}	0.06	–	8, 9	kmole COD_X.kg^{-1} COD_S
Y_{pro}	0.04	–	10	kmole COD_X.kg^{-1} COD_S
C_{ch4}	0.0156	kmole C.kg^{-1} COD	11, 12	C_9 in state equation 10
Y_{ac}	0.05	–	11	kmole COD_X.kg^{-1} COD_S
Y_{h2}	0.06	–	12	kmole COD_X.kg^{-1} COD_S

Note that C_{h2} and C_{IN}, i.e. C_8 and C_{11}, are equal to zero in state equation 10 (S_{IC})

6.2. Biochemical parameter values

Table 2: Biochemical parameter values for the BSM2 ADM1 implementation

Parameter	Value	Unit	Process(es)	Comments
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	
$K_{S,IN}$	1.10^{-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 process inhibition equations $I_{5–10}$
$pH_{LL,aa}$	4	-	5-10	in process inhibition equations $I_{5–10}$
$k_{m,aa}$	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	
$K_{I,h2,fa}$	5.10^{-6}	kg COD.m^{-3}	7	in process inhibition equation I_7
$k_{m,c4}$	20	d^{-1}	8, 9	
$K_{S,c4}$	0.2	kg COD.m^{-3}	8, 9	
$K_{I,h2,c4}$	1.10^{-5}	kg COD.m^{-3}	8, 9	in process inhibition equations I_8 and I_9
$k_{m,pro}$	13	d^{-1}	10	
$K_{S,pro}$	0.1	kg COD.m^{-3}	10	
$K_{I,h2,pro}$	$3.5{\cdot}10^{-6}$	kg COD.m^{-3}	10	in process inhibition equations I_8 and I_9
$k_{m,ac}$	8	d^{-1}	11	
$K_{S,ac}$	0.15	kg COD.m^{-3}	11	
$K_{I,NH3}$	0.0018	M	11	in process inhibition equation I_{11}
$pH_{UL,ac}$	7	-	11	in process inhibition equation I_{11}

$pH_{LL,ac}$	6	-	11	in process inhibition equation I_{11}
$k_{m,h2}$	35	d^{-1}	12	
$K_{S,h2}$	7.10^{-6}	kg COD.m^{-3}	12	
$pH_{UL,h2}$	6	-	12	in process inhibition equation I_{12}
$pH_{LL,h2}$	5	-	12	in process inhibition equation 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}	13	

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

6.3. Physico-chemical parameter values

Table 3: Physico-chemical parameter values for the BSM2 ADM1 implementation

Parameter	Value	Unit	Comments
R	0.083145	bar.M^{-1}.K^{-1}	ADM1 default value = 0.08314
T_{base}	298.15	K	ADM1 default value = 298
T_{ad}	308.15	K	= 35°C
K_w	$10^{-14.0}\exp\left(\frac{55900}{100\cdot R}\cdot\left(\frac{1}{T_{base}}-\frac{1}{T_{ad}}\right)\right)$	M	$\approx 2.08\cdot10^{-14}$
$K_{a,va}$	$10^{-4.86}$	M	$\approx 1.38\cdot10^{-5}$
$K_{a,bu}$	$10^{-4.82}$	M	$\approx 1.5\cdot10^{-5}$
$K_{a,pro}$	$10^{-4.88}$	M	$\approx 1.32\cdot10^{-5}$
$K_{a,ac}$	$10^{-4.76}$	M	$\approx 1.74\cdot10^{-5}$
$K_{a,co2}$	$10^{-6.35}\exp\left(\frac{7646}{100\cdot R}\cdot\left(\frac{1}{T_{base}}-\frac{1}{T_{ad}}\right)\right)$	M	$\approx 4.94\cdot10^{-7}$
$K_{a,IN}$	$10^{-9.25}\exp\left(\frac{51965}{100\cdot R}\cdot\left(\frac{1}{T_{base}}-\frac{1}{T_{ad}}\right)\right)$	M	$\approx 1.11\cdot10^{-9}$
$k_{A,Bva}$	1.10^{10}	$M^{-1}.d^{-1}$	Set to be at least three orders of magnitude higher than the fastest time constant of the system ADM1 suggested default value = 1.10^8
$k_{A,Bbu}$	1.10^{10}	$M^{-1}.d^{-1}$	
$k_{A,Bpro}$	1.10^{10}	$M^{-1}.d^{-1}$	
$k_{A,Bac}$	1.10^{10}	$M^{-1}.d^{-1}$	
$k_{A,Bco2}$	1.10^{10}	$M^{-1}.d^{-1}$	
$k_{A,BIN}$	1.10^{10}	$M^{-1}.d^{-1}$	
P_{atm}	1.013	bar	
$p_{gas,h2o}$	$0.0313\cdot\exp\left(5290\cdot\left(\frac{1}{T_{base}}-\frac{1}{T_{ad}}\right)\right)$	bar	≈ 0.0557
k_p	5.10^4	$m^3.d^{-1}.bar^{-1}$	Explicit for BSM2 AD conditions to achieve a reasonable head space pressure. Must be recalibrated for other cases.
K_La	200	d^{-1}	
$K_{H,co2}$	$0.035\cdot\exp\left(\frac{-19410}{100\cdot R}\cdot\left(\frac{1}{T_{base}}-\frac{1}{T_{ad}}\right)\right)$	$M_{liq}.bar^{-1}$	≈ 0.0271
$K_{H,ch4}$	$0.0014\cdot\exp\left(\frac{-14240}{100\cdot R}\cdot\left(\frac{1}{T_{base}}-\frac{1}{T_{ad}}\right)\right)$	$M_{liq}.bar^{-1}$	≈ 0.00116
$K_{H,h2}$	$7.8\cdot10^{-4}\cdot\exp\left(\frac{-4180}{100\cdot R}\cdot\left(\frac{1}{T_{base}}-\frac{1}{T_{ad}}\right)\right)$	$M_{liq}.bar^{-1}$	$\approx 7.38\cdot10^{-4}$

6.4. Physical parameter values used in BSM2

Table 4: Physical parameter values for the BSM2 ADM1 implementation

Parameter	Value	Unit
$V_{ad,liq}$	3400	m^3
$V_{ad,gas}$	300	m^3

7. ADM1 BENCHMARK MODEL STEADY-STATE RESULTS

In this section, the results of a steady state simulation are given. They 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 °C. As a part of the BSM Task Group's work model interfaces for connecting ASM1 and ADM1 (and vice versa) have been developed (see e.g. Nopens *et al.* (2008) and BSM Task Group Technical Report on Interfacing). The ASM1 to ADM1 interface fractionates the incoming COD into inerts, carbohydrates, proteins and lipids rather than putting the incoming COD into the ADM1 as composite material. Consequently, the composite material value in this example 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 verification of the dynamics, results from the full BSM2 ring test procedure may be used (see BSM Task Group Technical Report on Ring-testing).

Table 5: Steady-state input variable values for BSM2 ADM1

State no.	Variable	Value	Unit
1	$S_{su,i}$	0.01	kg COD.m^{-3}
2	$S_{aa,i}$	0.001	kg COD.m^{-3}
3	$S_{fa,i}$	0.001	kg COD.m^{-3}
4	$S_{va,i}$	0.001	kg COD.m^{-3}
5	$S_{bu,i}$	0.001	kg COD.m^{-3}
6	$S_{pro,i}$	0.001	kg COD.m^{-3}
7	$S_{ac,i}$	0.001	kg COD.m^{-3}
8	$S_{h2,i}$	1.10^{-8}	kg COD.m^{-3}
9	$S_{ch4,i}$	1.10^{-5}	kg COD.m^{-3}
10	$S_{IC,i}$	0.04	kmole C.m^{-3}
11	$S_{IN,i}$	0.01	kmole N.m^{-3}
12	$S_{I,i}$	0.02	kg COD.m^{-3}
13	$X_{C,I}$	2.0	kg COD.m^{-3}
14	$X_{ch,i}$	5.0	kg COD.m^{-3}
15	$X_{pr,i}$	20.0	kg COD.m^{-3}
16	$X_{li,i}$	5.0	kg COD.m^{-3}
17	$X_{su,i}$	0.0	kg COD.m^{-3}
18	$X_{aa,i}$	0.01	kg COD.m^{-3}
19	$X_{fa,i}$	0.01	kg COD.m^{-3}
20	$X_{c4,i}$	0.01	kg COD.m^{-3}
21	$X_{pro,i}$	0.01	kg COD.m^{-3}
22	$X_{ac,i}$	0.01	kg COD.m^{-3}
23	$X_{h2,i}$	0.01	kg COD.m^{-3}
24	$X_{I,i}$	25.0	kg COD.m^{-3}
25	$S_{cat,i}$	0.04	kmole.m^{-3}
26	$S_{an,i}$	0.02	kmole.m^{-3}
–	Q_{ad}	170.0	$m^3.d^{-1}$
–	T_{op}	35.0	°C

Table 6: Steady-state output variable values for BSM2 ADM1

State no.	Variable	Value	Unit
1	S_{su}	0.011954829716958	kg COD.m^{-3}
2	S_{aa}	0.005314740171633	kg COD.m^{-3}
3	S_{fa}	0.098621400930799	kg COD.m^{-3}
4	S_{va}	0.011625006463861	kg COD.m^{-3}
5	S_{bu}	0.013250729666269	kg COD.m^{-3}
6	S_{pro}	0.015783666284542	kg COD.m^{-3}
7	S_{ac}	0.197629716937552	kg COD.m^{-3}
8	S_{h2}	0.000000235945059	kg COD.m^{-3}
9	S_{ch4}	0.055088776445959	kg COD.m^{-3}
10	S_{IC}	0.152677870626333	kmole C.m^{-3}
11	S_{IN}	0.130229815803682	kmole N.m^{-3}
12	S_{I}	0.328697663721532	kg COD.m^{-3}
13	X_{C}	0.308697663721532	kg COD.m^{-3}
14	X_{ch}	0.027947240435040	kg COD.m^{-3}
15	X_{pr}	0.102574106106682	kg COD.m^{-3}
16	X_{li}	0.029483049707287	kg COD.m^{-3}
17	X_{su}	0.420165982454567	kg COD.m^{-3}
18	X_{aa}	1.179171798923700	kg COD.m^{-3}
19	X_{fa}	0.243035344719442	kg COD.m^{-3}
20	X_{c4}	0.431921105635979	kg COD.m^{-3}
21	X_{pro}	0.137305908933954	kg COD.m^{-3}
22	X_{ac}	0.760562658313215	kg COD.m^{-3}
23	X_{h2}	0.317022953361272	kg COD.m^{-3}
24	X_{I}	25.617395327443063	kg COD.m^{-3}
25	S_{cat}	0.040000000000000	kmole.m^{-3}
26	S_{an}	0.020000000000000	kmole.m^{-3}
–	Q_{ad}	170.0000000000000	m^{3}.d^{-1}
–	T_{op}	35.00000000000000	°C
–	pH	7.465537769904638	–
–	S_{H+}	0.000000034234361	kmole H^{+}.m^{-3}
27	S_{va-}	0.011596247072545	kg COD.m^{-3}
28	S_{bu-}	0.013220826248532	kg COD.m^{-3}
29	S_{pro-}	0.015742783191567	kg COD.m^{-3}
30	S_{ac-}	0.197241155436605	kg COD.m^{-3}
31	S_{hco3-}	0.142777479392078	kmole C.m^{-3}
–	S_{co2}	0.009900391234255	kmole C.m^{-3}
32	S_{nh3}	0.004090928458444	kmole N.m^{-3}
–	S_{nh4+}	0.126138887345238	kmole N.m^{-3}
33	$S_{gas,h2}$	0.000010241035595	kg COD.m^{-3}
34	$S_{gas,ch4}$	1.625607209981422	kg COD.m^{-3}
35	$S_{gas,co2}$	0.014150534678395	kmole C.m^{-3}
–	$p_{gas,h2}$	0.000016399182640	bar
–	$p_{gas,ch4}$	0.650779632823186	bar
–	$p_{gas,co2}$	0.362552713328102	bar
–	P_{gas}	1.069016490408918	bar
–	Q_{gas}	2955.703454193784	Nm^{3}.d^{-1}

Note that the gas flow rate, Q_{gas}, in Table 6 is the flow rate at atmospheric pressure and not at the pressure of the head space. The simulation above was carried out using the ode15s stiff solver with a relative tolerance of 1.10^{-10} and an absolute tolerance of 1.10^{-12}.

8. SIMULATION EFFICIENCY ANALYSIS

To evaluate the model implementations described in this report, a number of simulation tests were carried out. These tests included 1) steady-state simulations, 2) dynamic simulations for two weeks to compare the transient behaviour in detail and 3) dynamic simulations for 609 days to compare overall simulation times. The model implementations investigated were:

1. ODE – the differential equation implementation;
2. DAE1 – differential equations with algebraic solution of pH (S_{H+});
3. DAE2 – differential equations with algebraic solution of pH and S_{h2}.

All three models were tested as a part of the BSM2. This means that the behaviour reported here refers to the whole BSM2 system. The simulations were carried out using a standard PC running Windows XP and Matlab/Simulink Release 13. The ASM1, the clarifiers and the ADM1 were all implemented as MEX-files based on C source code.

8.1. Steady state simulations

The three model implementations 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.

8.2. Transient behaviour

Although the model implementations differ in the description of pH and S_{h2} dynamics, no significant differences were obtained when the transients in the states were investigated. The relative errors are typically in the range of 10^{-6} or smaller, again using the ODE implementation as reference. However, one important exception is the gas flow rate. Although not a true state of the model, the gas flow rate seems to be highly sensitive to the integration algorithm and the step length used. The use of some solvers resulted in a very nervous behaviour in the gas flow rate with noise in the range of a few percent. This nervousness appears in all model implementations discussed in this report, especially when a noise generator is present, and is most likely caused by integration numerics. Since the gas flow rate is a very important variable this must be taken into consideration.

8.3. Simulation speed

The simulation speed was tested using different solvers. A discontinuous sensor with a noise source and time delay as well as a discrete feed-back controller were implemented to test the three implementations in a hybrid system with noise. The results are shown in Table 7 as relative simulation times, using DAE2 with ode45 as the reference (simulation time for all 609 days was 50 minutes using a relative integration tolerance of 10^{-5}). It is interesting to note that the implementation of only a pH solver does not give any significant improvement in simulation speed. Substantial improvement is not obtained unless the fast state associated with S_{h2} is removed. To better understand this dramatic improvement in simulation speed, Figure 1 shows the eigenvalues (λ) of the linearised ADM1 implementations, respectively. Investigation of the eigenvalues of the system matrix provides an indication on how the distribution of time constants appears in the system at a certain operating point. From the figure, it is evident that the eigenvalue associated with the pH (S_{H+}) is a factor 10 larger than the one associated with the S_{h2} state. However, the third largest eigenvalue is only about 1/100 of that of associated with S_{h2}. Clearly, both pH and S_{h2} must be removed to significantly reduce the stiffness of the system. However, it should be noted that the working point investigated is the anticipated working point of the BSM2.

Table 7: Relative simulation times for the three different implementations of ADM1

	ode45	ode23	ode23tb (stiff)	ode15s (stiff)
ODE	53	96	28	18
DAE1	53	85	27	17
DAE2	1 (ref.)	1.2	28	18
ODE*	65	51	5	4
DAE1*	87	48	5	4
DAE2*	0.75	0.5	5	4

* Implementation without noise and without time discrete sensor/controller.

Rewriting the ODE system into a DAE system, representing both the pH and S_{h2} state by algebraic equations, yields a significant simulation time reduction. As seen in Table 7, the improvement when noise and discrete sensors are present is significant. This also holds in the absence of noise or when discrete sensors are implemented. Compared to the ODE system or the DAE1 system, an increase in speed by a factor 8 is achieved (ODE* simulation time: 4, DAE1* simulation time: 4 and DAE2* simulation time: 0.5, see Table 7).

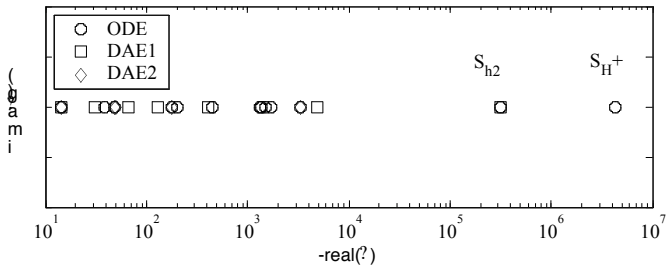

Figure 1. The eigenvalues of the linearised systems. Note the logarithmic scale and that small the values are not shown (x-axis: –real(λ), y-axis: im(λ)).

9. 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 essential module for long-term benchmarking development. *Water Sci. Technol.*, **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.B., 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. *Water Sci. Technol.*, **51**(1), 287-295.

Jeppsson U., Pons M.-N., Nopens I., Alex J., Copp J.B., Gernaey K.V., Rosen C., Steyer J.-P. and Vanrolleghem P.A. (2007). Benchmark Simulation Model No 2 – General protocol and exploratory case studies. *Water Sci. Technol.*, **56**(8), 67-78.

Nopens I., Batstone D., Copp J.B., Jeppsson U., Volcke E.I.P., Alex J. and Vanrolleghem P.A. (2008). A practical ASM/ADM model interface for enhanced dynamic plant-wide simulation. *Water Res.* (in revision).

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. *Water Sci. Technol.*, **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 Matlab/Simulink. *Water Sci. Technol.,* **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. *Water Res.*, **25**(10), 1263-1271.

Volcke E.I.P. (2006). *Modelling, analysis and control of partial nitritation in a SHARON reactor*. PhD thesis, Ghent University, Belgium, pp. 300 (Section 3.4. pH calculation available at http://biomath.ugent.be/publications/download/VolckeEveline_PhD.pdf).

Appendix 1. Code for ADM1 DAE implementation in Matlab/Simulink

In this appendix, the solvers for pH and S_{h2} used in the Lund University implementation of ADM1 are presented. The solver files are written in C for the 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 $E(S_{X,k})$ and $dE(S_{X,k})/dS_X|_{SX,k}$, respectively, and the functions `pHsolver` and `Sh2solver`, respectively, for the iteration (based on the Newton-Raphson algorithm) in order to find the solution S_X (index X here representing H^+ or h2).

In the computer code below (Appendicies 1.1 and 1.2), parameter names should be self explanatory. For the input variables the following translation table applies:

u[0] = S_{su}	u[1] = S_{aa}	u[2] = S_{fa}	u[3] = S_{va}	u[4] = S_{bu}	u[5] = S_{pro}
u[6] = S_{ac}	u[9] = S_{IC}	u[10] = S_{IN}	u[16] = X_{su}	u[17] = X_{aa}	u[18] = X_{fa}
u[19] = X_{c4}	u[20] = X_{pro}	u[22] = X_{h2}	u[24] = S_{cat}	u[25] = S_{an}	u[26] = Q_{ad}
u[33] = pH	u[34] = S_{H+}	u[43] = $S_{gas,h2}$	u[51] = $S_{h2,i}$		

For the state variables the following translation tables applies:

In `pHsolver`:	x[0] = S_{H+}	x[1] = S_{va-}	x[2] = S_{bu-}	x[3] = S_{pro-}	x[4] = S_{ac-}
	x[5] = S_{hco3-}	x[6] = S_{nh3}			
In `Sh2solver`:	x[0] = S_{h2}				

Appendix 1.1. C-file for pH solver

```
/*
 * pHsolv_bsm2.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_bsm2
#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]))
           +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]);
           /* Gradient of SH+ equation */
}

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
```

Appendix 1.2. C-file for S_{h2} solver

```
/*
 * Sh2solv_bsm2.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_bsm2
#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+ */

/* 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+ */

/* 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
```

Appendix 2. Petersen matrix representation of original ADM1

The complete Petersen matrix describing the liquid phase reactions in the original ADM1 is shown below (from Batstone *et al.*, 2002).

Table 8: Biochemical rate coefficients ($\nu_{i,j}$) and kinetic equations ($\rho_{i,j}$) for soluble components (i = 1–12, j = 1–19)

Component → i	1	2	3	4	5	6	7	8	9	10	11	12	Rate (ρ_j, kg COD.m^{-3}.d^{-1})
j Process ↓	S_{su}	S_{aa}	S_{fa}	S_{va}	S_{bu}	S_{pro}	S_{ac}	S_{h2}	S_{ch4}	S_{IC}	S_{IN}	S_I	
1 Disintegration												$f_{sI,xc}$	$k_{dis}X_c$
2 Hydrolysis Carbohydrates	1												$k_{hyd,ch}X_{ch}$
3 Hydrolysis of Proteins		1											$k_{hyd,pr}X_{pr}$
4 Hydrolysis of Lipids	$1-f_{fa,li}$		$f_{fa,li}$										$k_{hyd,li}X_{li}$
5 Uptake of Sugars	-1				$(1-Y_{su})\,f_{bu,su}$	$(1-Y_{su})\,f_{pro,su}$	$(1-Y_{su})\,f_{ac,su}$	$(1-Y_{su})\,f_{h2,su}$		$-\sum_{i=1-9,11-24} C_i\nu_{i,5}$	$-(Y_{su})\,N_{bac}$		$k_{m,su}\frac{S_{su}}{K_S+S}X_{su}I_1$
6 Uptake of Amino Acids		-1		$(1-Y_{aa})\,f_{va,aa}$	$(1-Y_{aa})\,f_{bu,aa}$	$(1-Y_{aa})\,f_{pro,aa}$	$(1-Y_{aa})\,f_{ac,aa}$	$(1-Y_{aa})\,f_{h2,aa}$		$-\sum_{i=1-9,11-24} C_i\nu_{i,6}$	$N_{aa}-(Y_{aa})\,N_{bac}$		$k_{m,aa}\frac{S_{aa}}{K_S+S_{aa}}X_{aa}I_1$
7 Uptake of LCFA			-1				$(1-Y_{fa})\,0.7$	$(1-Y_{fa})\,0.3$			$-(Y_{fa})\,N_{bac}$		$k_{m,fa}\frac{S_{fa}}{K_S+S_{fa}}X_{fa}I_2$
8 Uptake of Valerate				-1		$(1-Y_{c4})\,0.54$	$(1-Y_{c4})\,0.31$	$(1-Y_{c4})\,0.15$			$-(Y_{c4})\,N_{bac}$		$k_{m,c4}\frac{S_{va}}{K_S+S_{va}}X_{c4}\frac{1}{1+S_{bu}/S_{va}}I_2$
9 Uptake of Butyrate					-1		$(1-Y_{c4})\,0.8$	$(1-Y_{c4})\,0.2$			$-(Y_{c4})\,N_{bac}$		$k_{m,c4}\frac{S_{bu}}{K_S+S_{bu}}X_{c4}\frac{1}{1+S_{va}/S_{bu}}I_2$
10 Uptake of Propionate						-1	$(1-Y_{pro})\,0.57$	$(1-Y_{pro})\,0.43$		$-\sum_{i=1-9,11-24} C_i\nu_{i,10}$	$-(Y_{pro})\,N_{bac}$		$k_{m,pr}\frac{S_{pro}}{K_S+S_{pro}}X_{pro}I_2$
11 Uptake of Acetate							-1		$(1-Y_{ac})$	$-\sum_{i=1-9,11-24} C_i\nu_{i,11}$	$-(Y_{ac})\,N_{bac}$		$k_{m,ac}\frac{S_{ac}}{K_S+S_{ac}}X_{ac}I_3$
12 Uptake of Hydrogen								-1	$(1-Y_{h2})$	$-\sum_{i=1-9,11-24} C_i\nu_{i,12}$	$-(Y_{h2})\,N_{bac}$		$k_{m,h2}\frac{S_{h2}}{K_S+S_{h2}}X_{h2}I_1$
13 Decay of X_{su}													$k_{dec,Xsu}X_{su}$
14 Decay of X_{aa}													$k_{dec,Xaa}X_{aa}$
15 Decay of X_{fa}													$k_{dec,Xfa}X_{fa}$
16 Decay of X_{c4}													$k_{dec,Xc4}X_{c4}$
17 Decay of X_{pro}													$k_{dec,Xpro}X_{pro}$
18 Decay of X_{ac}													$k_{dec,Xac}X_{ac}$
19 Decay of X_{h2}													$k_{dec,Xh2}X_{h2}$
	Monosaccharides (kgCOD m^{-3})	Amino Acids (kgCOD m^{-3})	Long chain fatty acids (kgCOD m^{-3})	Total valerate (kgCOD m^{-3})	Total butyrate (kgCOD m^{-3})	Total propionate (kgCOD m^{-3})	Total acetate (kgCOD m^{-3})	Hydrogen gas (kgCOD m^{-3})	Methane gas (kgCOD m^{-3})	Inorganic Carbon (kmoleC m^{-3})	Inorganic nitrogen (kmoleN m^{-3})	Soluble inerts (kgCOD m^{-3})	Inhibition factors (3.7): $I_1 = I_{pH}I_{IN,lim}$; $I_2 = I_{pH}I_{IN,lim}I_{h2}$; $I_3 = I_{pH}I_{IN,lim}I_{NH3,Xac}$

Table 9: Biochemical rate coefficients ($\nu_{i,j}$) and kinetic equations ($\rho_{i,j}$) for particulate components (i = 13–24, j = 1–19)

j	Component → / Process ↓	13 X_c	14 X_{ch}	15 X_{pr}	16 X_{li}	17 X_{su}	18 X_{aa}	19 X_{fa}	20 X_{c4}	21 X_{pro}	22 X_{ac}	23 X_{h2}	24 X_I	Rate (ρ_j, kg COD.m^{-3}.d^{-1})
1	Disintegration	-1	$f_{ch,xc}$	$f_{pr,xc}$	$f_{li,xc}$								$f_{xI,xc}$	$k_{dis}X_c$
2	Hydrolysis Carbohydrates		-1											$k_{hyd,ch}X_{ch}$
3	Hydrolysis of Proteins			-1										$k_{hyd,pr}X_{pr}$
4	Hydrolysis of Lipids				-1									$k_{hyd,li}X_{li}$
5	Uptake of Sugars					Y_{su}								$k_{m,su}\frac{S_{su}}{K_S+S}X_{su}I_1$
6	Uptake of Amino Acids						Y_{aa}							$k_{m,aa}\frac{S_{aa}}{K_S+S_{aa}}X_{aa}I_1$
7	Uptake of LCFA							Y_{fa}						$k_{m,fa}\frac{S_{fa}}{K_S+S_{fa}}X_{fa}I_2$
8	Uptake of Valerate								Y_{c4}					$k_{m,c4}\frac{S_{va}}{K_S+S_{va}}X_{c4}\frac{1}{1+S_{bu}/S_{va}}I_2$
9	Uptake of Butyrate								Y_{c4}					$k_{m,c4}\frac{S_{bu}}{K_S+S_{bu}}X_{c4}\frac{1}{1+S_{va}/S_{bu}}I_2$
10	Uptake of Propionate									Y_{pro}				$k_{m,pr}\frac{S_{pro}}{K_S+S_{pro}}X_{pro}I_2$
11	Uptake of Acetate										Y_{ac}			$k_{m,ac}\frac{S_{ac}}{K_S+S_{ac}}X_{ac}I_3$
12	Uptake of Hydrogen											Y_{h2}		$k_{m,h2}\frac{S_{h2}}{K_S+S_{h2}}X_{h2}I_1$
13	Decay of X_{su}	1				-1								$k_{dec,Xsu}X_{su}$
14	Decay of X_{aa}	1					-1							$k_{dec,Xaa}X_{aa}$
15	Decay of X_{fa}	1						-1						$k_{dec,Xfu}X_{fa}$
16	Decay of X_{c4}	1							-1					$k_{dec,Xc4}X_{c4}$
17	Decay of X_{pro}	1								-1				$k_{dec,Xpro}X_{pro}$
18	Decay of X_{ac}	1									-1			$k_{dec,Xac}X_{ac}$
19	Decay of X_{h2}	1										-1		$k_{dec,Xh2}X_{h2}$
		Composites (kgCOD m^{-3})	Carbohydrates (kgCOD m^{-3})	Proteins (kgCOD m^{-3})	Lipids (kgCOD m^{-3})	Sugar degraders (kgCOD m^{-3})	Amino acid degraders (kgCOD m^{-3})	LCFA degraders (kgCOD m^{-3})	Valerate and butyrate degraders (kgCOD m^{-3})	Propionate degraders (kgCOD m^{-3})	Acetate degraders (kgCOD m^{-3})	Hydrogen degraders (kgCOD m^{-3})	Particulate inerts (kgCOD m^{-3})	Inhibition factors (3.7): $I_1 = I_{pH}I_{IN,lim}$ $I_2 = I_{pH}I_{IN,lim}I_{h_2}$ $I_3 = I_{pH}I_{IN,lim}I_{NH3,Xac}$