CODEN:LUTEDX/(TEIE-7258)/1-75/(2015)

Activated Sludge Model No. 3 with bioP module (ASM bioP) implemented within the Benchmark Simulation Model No.1

Kimberly Solon

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

Activated Sludge Model No. 3 with bioP module (ASM3 bioP) implemented within the Benchmark Simulation Model No. 1 (BSM1)

REPORT

KIMBERLY SOLON

2015 SEPTEMBER 10

CONTENTS

PAGE NO.

1. INTRODUCTION

This is a modelling, simulation and ring-testing exercise as a part of the self-study course required for my PhD studies at IEA, LTH. The specific objective of this task is to create a Matlab/Simulink implementation of the EAWAG bio-P module for IWA Activated Sludge Model No. 3 (ASM3 bioP) within the Benchmark Simulation Model No. 1 (BSM1) platform.

1.1. Activated Sludge Models (ASMs)

The activated sludge models are mathematical models describing biological and chemical reactions occurring in activated sludge systems. The Activated Sludge Model No. 1 (**ASM1**) was published in 1987 (Henze et al., 1987) and is the foundation for further developments of the ASMs and their extensions. ASM1 describes biological oxidation of carbon, nitrification and denitrification processes and is therefore used to simulate carbon and nitrogen removal in activated sludge systems. As any other model, ASM1 has its limitations, one of which is the modelling of phosphorus removal. This restriction led to the development of the Activated Sludge Model No. 2 (**ASM2**) and its publication in 1995 (Henze et al., 1995). ASM2 includes biological phosphorus removal processes in addition to description of carbon and nitrogen removal. After four years, the ASM2 was also extended to **ASM2d** (Henze et al., 1999) as understanding of the role of denitrification grew (in particular, denitrifying phosphorus accumulating organisms). And in 2000, **ASM3** (Gujer et al., 2000) was published which deals with some other limitations of ASM1, such as difference of decay rates of nitrifiers under aerobic and anoxic conditions, and also includes cell internal storage compounds (as in ASM2). One other important difference between ASM1 and ASM3 is in the COD flow – in ASM1, the death-regeneration cycle of the heterotrophs and the decay of nitrifiers are strongly interrelated while for ASM3, all the conversion processes of the nitirifiers and heterotrophs are clearly separated and the decay processes are identical. Similarly, an extension of the ASM3 model was developed (i.e. **ASM3 bioP**) which allows for the prediction of enhanced biological phosphorus removal through inclusion of modified processes from ASM2d but without the fermentation of readily degradable substrates (Rieger et al., 2001). These activated sludge models are commonly used in numerous simulation platforms and are mainly used for scientific research to study biological processes in hypothetical systems. Inasmuch, an application of the ASMs could be for optimisation of the operation of wastewater treatment plants (WWTPs) provided that they are calibrated well.

1.2. Benchmark Simulation Models (BSMs)

Optimisation of wastewater treatment plant process performance could be done through numerous control strategies (e.g. control of recirculation flow rate, control of aeration, etc.), however, they are difficult to evaluate and compare. In order to compare different control strategies, there should be a benchmark and standardized evaluation criteria. The Benchmark Simulation Models (BSMs) have been created for this purpose. The BSM is a platform with a defined plant layout, bioprocess model/s, influent loads, sensors and actuator models, as well as a set of evaluation criteria. Using the BSM, the user can apply a specific control strategy and then evaluate it using a set of criteria, thus, it is possible to compare different control strategies by looking at the results of the evaluation criteria. The first BSM is the Benchmark Simulation Model No. 1 (**BSM1**) (Copp et al., 2002; Alex et al., 2008) composed of a five-compartment activated sludge reactor followed by a secondary clarifier, which is a common configuration for biological nitrogen removal. BSM1 was further developed to Long Term Benchmark Simulation Model No. 1 (**BSM1_LT**) (Rosen et al., 2004) wherein the evaluation period is extended significantly to include seasonal variations. The BSM1 was also developed significantly into the Benchmark Simulation Model No. 2 (**BSM2**) to include the sludge line and thus have in addition a primary clarifier, thickener, anaerobic digester and dewatering unit (Jeppsson et al., 2007; Nopens et al., 2010; Gernaey et al., 2014). This allows the number of possible control handles to substantially increase.

2. METHODOLOGY

In order to have a structured approach for this modelling and ring-testing exercise, the following steps were carried out:

Step 1. Implement ASM3 bioP, with kinetic parameters corrected for temperature effects (TEMPMODEL = 1, see sidenote), in a one reactor model (called **model A2**). The motivation for choosing this configuration is so that it can be ring-tested with a pre-existing implementation of the ASM3 bioP model which was implemented as such.

Step 2. Ring-test the model in Step 1 with an earlier implementation of a one reactor ASM3 bioP model (called **model A1**).

Step 3. Implement the ring-tested **model A2** in BSM1 open-loop. Update or make relevant modifications to the other c-, m-, and mat-files based on changes in parameters, influent files, initial values, evaluation criteria, etc. as used in ASM3 bioP. More importantly, changes in the Simulink model is necessary in order to simulate a wastewater treatment plant configuration designed for bioP removal (i.e. A2O plant configuration of the activated sludge unit as shown in **Figure 1**). Provide results (steady state and dynamic values: TEMPMODEL = 0 and 1).

Step 4. Reduce the ASM3 bioP model to ASM3 (named **model B2**) and ring-test with another pre-existing implementation (called **model B1**). Provide results (steady state and dynamic influent, reactive and nonreactive settler, TEMPMODEL = 0 and 1).

<u>TEMPERATURE EFFECTS</u>

Different biological kinetics have temperature dependencies and this is accounted for by interpolating kinetic parameters to different temperatures. In the BSM1 platform used in this study there are two possibilities for which to base the effect of temperature on the kinetic parameters. The first option considers that the temperature at the influent of the reactor is the same as the temperature throughout the reactor (**TEMPMODEL = 0**). While the second option considers the temperature dynamics in each reactor and are modelled based on the heat content of the wastewater and assuming completely mixed conditions (**TEMPMODEL = 1**).

2.1. Plant Layout

The original BSM1 (Copp et al., 2002) plant is comprised of an activated sludge unit and a secondary clarifier. The activated sludge unit is a five-compartment reactor, wherein the first two are anoxic while the last three are aerated. The biological processes taking place in the activated sludge unit is described by the Activate Sludge Model No. 1 (ASM1) while the secondary clarifier is modelled as a 10-layer non-reactive unit (Takács et al., 1991).

In this exercise, ASM1 will be replaced by the ASM3 extended with biological phosphorus removal (i.e. ASM3 bioP) (Rieger et al., 2001). As a consequence, the plant layout is also changed with an increased anoxic zone (i.e. addition of two anoxic compartments) and an internal recycle from the last aerated tank (compartment 7) into the first anoxic tank (compartment 3). This is, in fact, an A2O (Anaerobic-Anoxic-Oxic) configuration designed for both phosphorus and nitrogen removal.

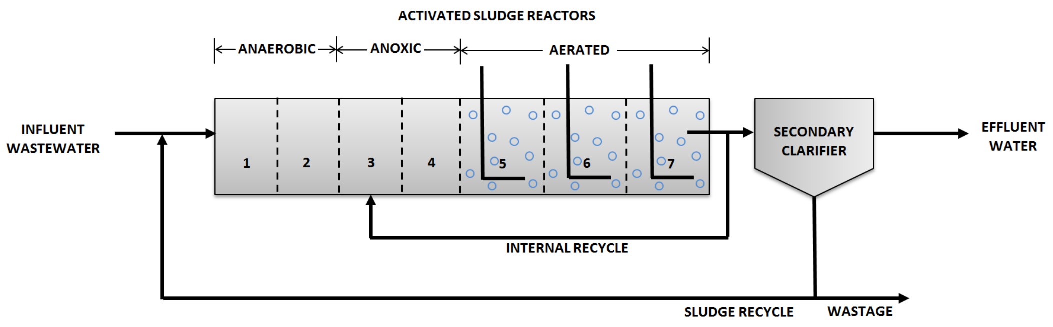

Figure 1. Modified BSM1 plant layout for biological phosphorus removal.

2.2. Influent Data

The wastewater influent values and characteristics were adapted from the ASM2d influent files used in the paper of Gernaey and Jørgensen (2004). The only difference is that S_F (fermentation products) and S_A (fermentable, readily biodegradable substrates) in ASM2d are combined as S_S in ASM3 bioP.

2.3. State Variables and Processes

Basically, ASM3 bioP is an extended model of the ASM3 to include biological phosphorus removal. In terms of state variables, ASM3 has 13, and 4 new state variables from ASM2d are also included (see **Table 1**) to account for bioP related variables.

Table 1. State variables for ASM3 bioP.

	SYMBOL	UNIT	DESCRIPTION
1	S_O	g (COD) m^{-3}	Dissolved oxygen
2	S_S	g (COD) m^{-3}	Readily biodegradable organic substrate
3	S_I	g (COD) m^{-3}	Inert soluble organic material
4	S_NH	g (N) m^{-3}	Ammonium and ammonia nitrogen ($NH_4^+ - N$ and $NH_3 - N$)
5	S_NO	g (N) m^{-3}	Nitrate and nitrite nitrogen ($NO_3^- - N$ and $NO_2^- - N$)
6	S_N2	g (N) m^{-3}	Dinitrogen (N_2)
7	S_PO4	g (P) m^{-3}	Inorganic soluble phosphorus, primarily orthophosphates
8	S_HCO	mol (HCO_3) m^{-3}	Alkalinity of the wastewater
9	X_I	g (COD) m^{-3}	Inert particulate organic material
10	X_S	g (COD) m^{-3}	Slowly biodegradable substrates
11	X_H	g (COD) m^{-3}	Heterotrophic organisms
12	X_STO	g (COD) m^{-3}	Cell internal storage product of heterotrophic organisms

13	X_PAO	g (COD) m^{-3}	Phosphate-accumulating organisms (PAO)
14	X_PP	g (P) m^{-3}	Polyphosphate
15	X_PHA	g (COD) m^{-3}	A cell internal storage product of PAOs, primarily polyhydroxyalkanoates (PHA)
16	X_A	g (COD) m^{-3}	Nitrifying organisms
17	X_TSS	g (SS) m^{-3}	Suspended solids

Highlighted in blue are components taken from ASM2d while the rest are ASM3 state variables.

Similarly, ASM3 processes are extended with ASM2d processes to include bioP reactions but without the precipitation reactions. The processes related to hydrolysis, heterotrophic and autotrophic growth are modelled similar to ASM3, but with phosphorus limitation in the growth processes. In order to model biological phosphorus removal, an additional 11 processes are included (P1, P2, …, P11) as reported in Rieger et al. (2001).

2.4. Other Additions

Temperature dependencies (TEMPMODEL = 0 and 1) of kinetic parameters, oxygen saturation concentration and kLa have been included in ASM3 bioP. Kinetic parameters and other values are taken at 20°C.

Hauduc et al. (2010) has found some inconsistencies/errors in the published ASM2d model as well as the ASM3 bioP model, thus, these were also taken into account in this modelling exercise.

3. RESULTS AND DISCUSSIONS

During the initial ring-testing of model A1 and model A2, the two models were found to have dissimilar model outputs. Although the values are close to each other, the goal is to have the two models give exactly the same results since everything is made equal in the two models (parameter values, initial values, influent data, etc.). After further checking of the earlier implemented model A1, it was found out that the process rates were not yet updated with the corrections reported by Hauduc et al. (2010). Specifically, the process rates for Proc11 and Proc12 were not corrected in model A1 (i.e. $K_{O,A}$ should be used instead of $K_{O,H}$). One important thing learned from this exercise is that during ring-testing, not only should one check the independent model implementation for errors but also the earlier version of the model implementation as it could also contain errors. After modifying the incorrect parameter values in model A1, the two models were able to give exactly the same model outputs as seen in **Table 2**.

Table 2. Steady state values of ASM3 bioP state variables when a one reactor model corrected for temperature effects (TEMPMODEL = 1) is run for 1000 days using constant influent.

ASM3 bioP STATE VARIABLES		MODEL A1 (pre-existing implementation)	MODEL A2 (this implementation)
1	S_O	0.0000	0.0000
2	S_S	79.3165	79.3165
3	S_I	30.0000	30.0000

4	S_NH	40.0891	40.0891
5	S_NO	0.0000	0.0000
6	S_N2	0.0000	0.0000
7	S_PO4	9.0591	9.0591
8	S_HCO	7.0011	7.0011
9	X_I	51.2000	51.2000
10	X_S	192.5035	192.5035
11	X_H	28.1700	28.1700
12	X_STO	0.0000	0.0000
13	X_PAO	0.0000	0.0000
14	X_PP	0.0000	0.0000
15	X_PHA	0.0000	0.0000
16	X_A	0.0000	0.0000
17	X_TSS	208.1276	208.1276

Model A2, after being ring-tested with model A1, was then implemented in BSM1 but with an A2O configuration for the activated sludge unit. **Table 3** gives the results of the dynamic simulation using dry influent data run with temperature compensation of kinetic parameters (TEMPMODEL = 0 and 1). In this study, all simulations were performed with constant temperature therefore results are identical for TEMPMODEL = 0 and 1. To achieve these values, steady state simulation was performed (200 days, constant influent data) followed by dynamic simulation (28 days, dynamic dry weather influent data) in BSM1 open-loop version, wherein the performance evaluation is performed on days 21 to 28.

Table 3. Performance indices of ASM3 bioP implementation in BSM1.

EVALUATION CRITERIA	**DYNAMIC DRY INFLUENT (TEMPMODEL = 0,1; SETTLER = 0)**	**DYNAMIC DRY INFLUENT (TEMPMODEL = 0,1; SETTLER = 1)**
Influent Quality Index (kg poll. units · d^{-1})	71527.2296	71527.2296
Effluent Quality Index (kg poll. units · d^{-1})	16254.4724	13284.0516
Operational Cost Index	17635.8337	18132.3437

Initial simulation results of the BSM1 with ASM3 bioP implementation have shown a very high biological phosphorus removal and a very low nitrification which were not expected. It was found out that the system is sensitive to initial values. It was found out that a kinetic term in one of the bioP processes (see **Eq. 1**) caused these unexpected values in the results. As seen in **Figure 2**, $\frac{X_{PP}}{X_{PAO}}$ values should only have values less than or equal to $K_{max,PAO}$ (0.20). When $\frac{X_{PP}}{X_{PAO}} > K_{max,PAO}$, this could lead to unreliable results and even numerical issues during simulation. People who use models with such kinetic (inhibition) terms (e.g. ASM2d) should be aware of their implications and limitations. Thus in this study, care was taken such that all $\frac{X_{PP}}{X_{PAO}}$ ratios of the initial values do not go beyond the limit values.

P2	Aerobic storage of X_{PP}	$q_{PP} \cdot \frac{S_O}{K_{O,PAO} + S_O} \cdot \frac{S_{PO_4}}{K_{PO_4,PP} + S_{PO_4}} \cdot \frac{S_{HCO}}{K_{HCO,PAO} + S_{HCO}} \cdot \frac{\frac{X_{PHA}}{X_{PAO}}}{K_{PHA} + \frac{X_{PHA}}{X_{PAO}}} \cdot \frac{K_{max,PAO} - \frac{X_{PP}}{X_{PAO}}}{K_{iPP,PAO} + K_{max,PAO} - \frac{X_{PP}}{X_{PAO}}} \cdot X_{PAO}$	**Eq. 1**

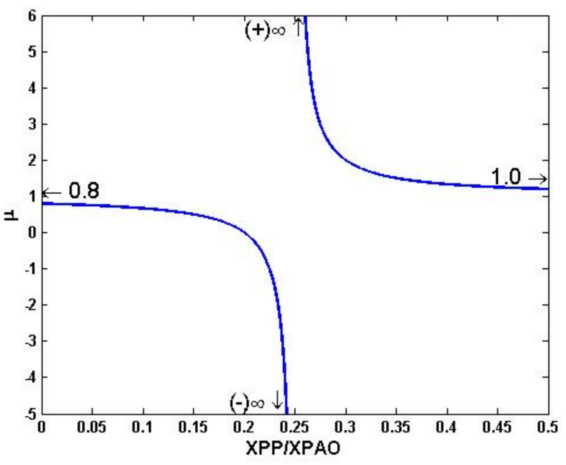

Figure 2. Plot of the kinetic term $\mu = \frac{K_{max,PAO} - \frac{X_{PP}}{X_{PAO}}}{K_{iPP,PAO} + K_{max,PAO} - \frac{X_{PP}}{X_{PAO}}}$ over a range of $\frac{X_{PP}}{X_{PAO}}$ ratios.

The results were not ring-tested since there is no pre-existing implementation of ASM3 bioP within the BSM1 platform. However, an ASM3 implementation within BSM1 is available (model B1). Thus, ASM3 bioP was reduced to ASM3 and implemented in BSM1 (model B2) and then ring-tested. Model B1 and model B2 both have a reactive settler with temperature-dependency of the kinetic parameters (TEMPMODEL = 1). The steady state simulation was performed (200 days, constant influent data); while the dynamic simulation was performed with one steady state simulation followed by 28 days of dynamic dry weather influent data in BSM1 open-loop version. The results of steady state and dynamic simulations are shown in **Table 4**. These values are obtained after modifying some inconsistencies found in the model B1 implementation (see **Appendix A1**).

Table 4. OCI and EQI values during dynamic and steady-state simulations for models B1 and B2 corrected for temperature effects (TEMPMODEL = 1).

EVALUATION CRITERIA	**STEADY STATE SIMULATION**		**DYNAMIC SIMULATION**	
	MODEL B1 **(pre-existing implementation)**	**MODEL B2** **(this implementation)**	**MODEL B1** **(pre-existing implementation)**	**MODEL B2** **(this implementation)**
Influent Quality Index (kg poll. units · d^{-1})	47041.8067	47041.8067	47040.0126	47040.0126
Effluent Quality Index (kg poll. units · d^{-1})	4296.7689	4296.7689	5086.4803	5086.4803
Operational Cost Index	16442.2023	16442.2023	16361.4824	16361.4824

4. REFERENCES

Alex, J., Benedetti, L., Copp, J., Gernaey, K.V., Jeppsson, U., Nopens, I., Pons, M.-N., Rieger, L., Rosen, C., Steyer, J.P., Vanrolleghem, P.A. & Winkler, S. (2008). *Benchmark Simulation Model No. 1 (BSM1)*. Technical Report No. LUTEDX/(TEIE-7229)/1-62/(2008). Lund, Sweden: Division of Industrial Electrical Engineering and Automation (IEA), Faculty of Engineering, Lund University.

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., Jeppsson, U., Vanrolleghem, P.A. & Copp, J.B. (2014). *Benchmarking of Control Strategies for Wastewater Treatment Plants.* IWA Scientific and Technical Report No. 23. London, UK: IWA Publishing.

Gernaey, K.V. & Jørgensen, S.B. (2004). Benchmarking combined biological phosphorus and nitrogen removal wastewater treatment processes. *Control Engineering Practice*, **12**(3), 357-373.

Gujer, W., Henze, M., Mino, T. & van Loosdrecht M.C.M. (2000). Activated Sludge Model No. 3. *Water Science and Technology,* **39**(1), 183–193.

Hauduc, H., Rieger, L., Takács, I., Héduit, A., Vanrolleghem, P.A. & Gillot, S. (2010). A systematic approach for model verification: application on seven published Activated Sludge Models. *Water Science and Technology*, 61(4), 825-839.

Henze, M., Gujer, W., Mino, T., Matsuo, T., Wentzel, M.C., Marais, G.v.R. & van Loosdrecht, M.C.M. (1999). *Activated Sludge Model No. 2d*. *Water Science and Technology*, **39**(1), 165–182.

Henze, M., Gujer, W., Mino, T., Matsuo, T., Wentzel, M.C. & Marais, G.v.R. (1995). *Activated Sludge Model No. 2.* IAWQ Scientific and Technical Report No. 3. London, UK: IAWQ.

Henze, M., Grady, C.P.L. Jr, Gujer, W., Marais, G.v.R. & Matsuo, T. (1987). *Activated Sludge Model No. 1.* IAWPRC Scientific and Technical Report No. 1. London, UK: IAWPRC.

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

Nopens, I., Benedetti, L., Jeppsson, U., Pons, M.-N., Alex, J., Copp, J.B., Gernaey, K.V., Rosen, C, Steyer, J.-P. & Vanrolleghem, P.A. (2010). Benchmark Simulation Model No. 2 – Finalisation of plant layout and default control strategy. *Water Science and Technology*, **62**(9), 1967-1974.

Rieger, L., Koch, G., Kühni, M., Gujer, W. & Siegrist, H. (2001). The EAWAG bio-P module for Activated Sludge Model No. 3. *Water Research*, **35**(16), 3887–3903.

Rosen, C., Jeppsson, U. & Vanrolleghem, P.A. (2004). Towards a common benchmark for long-term process control and monitoring performance evaluation. *Water Science and Technology*, **50**(11), 41-49.

Takács, I., Patry, G.G. & Nolasco, D. (1991) A dynamic model of the clarification thickening process. *Water Research*, **25**(10), 1263-1271.

APPENDIX

A1. Corrections to ASM3 implementation in BSM1

NOTE: Highlighted in grey are the errors in BSM1 ASM3 (model B1: pre-existing implementation) while statements in blue are the corrected statements.

stateset.m

```
SI_5 =  settler(m,131);
SS_5 =  settler(m,132);
SNH_5 = settler(m,133);
SN2_5 = settler(m,134);
SNO3_5 = settler(m,135);
SALK_5 = settler(m,136);
XI_5 =  settler(m,127);
XS_5 =  settler(m,138);
XBH_5 = settler(m,139);
XSTO_5 =settler(m,140);
XBA_5 = settler(m,141);
TSS_5 = settler(m,142);
T_5 =   settler(m,143);
SD1_5 = settler(m,144);
SD2_5 = settler(m,145);
SD3_5 = settler(m,146);
XD4_5 = settler(m,147);
XD5_5 = settler(m,148);
```

```
SI_5 =  settler(m,131);
SS_5 =  settler(m,132);
SNH_5 = settler(m,133);
SN2_5 = settler(m,134);
SNO3_5 = settler(m,135);
SALK_5 = settler(m,136);
XI_5 =  settler(m,137);
XS_5 =  settler(m,138);
XBH_5 = settler(m,139);
XSTO_5 =settler(m,140);
XBA_5 = settler(m,141);
TSS_5 = settler(m,142);
T_5 =   settler(m,143);
SD1_5 = settler(m,144);
SD2_5 = settler(m,145);
SD3_5 = settler(m,146);
XD4_5 = settler(m,147);
XD5_5 = settler(m,148);
```

perf_plant.m

```
BOD5evec2 =  (0.65*(SSevec+XSevec+(1-f_P)*(XBHevec+XBAevec + XSTOevec ))./Qevec;
```
```
BOD5evec2 =  (0.25*(SSevec+XSevec+(1-f_P)*(XBHevec+XBAevec + XSTOevec ))./Qevec;
```

```
BOD5eload =  (0.65*(SSeload+XSeload+(1-f_P)*(XBHeload+XBAeload + XSTOeload)));
```
```
BOD5eload =  (0.25*(SSeload+XSeload+(1-f_P)*(XBHeload+XBAeload + XSTOeload)));
```

```
BOD5e = 0.65*(settlerpart(:,24)+settlerpart(:,30)+(1-f_P)*(settlerpart(:,31)
+settlerpart(:,32)+ settlerpart(:,33) ));
```
```
BOD5e = 0.25*(settlerpart(:,24)+settlerpart(:,30)+(1-f_P)*(settlerpart(:,31)
+settlerpart(:,32)+ settlerpart(:,33) ));
```

```
SNKjin=     inpart(:,4)+ i_NSI*(inpart(:,1)) + i_NSS*(inpart(:,2)) +
i_NXI*(inpart(:,8)) + i_NXS*(inpart(:,9)) + i_NBM*( inpart(:,10) + inpart(:,11));
```
```
SNKjin=     inpart(:,4)+ i_NSI*(inpart(:,2)) + i_NSS*(inpart(:,3)) +
i_NXI*(inpart(:,8)) + i_NXS*(inpart(:,9)) + i_NBM*( inpart(:,10) + inpart(:,11));
```

```
SNOe=       settlerpart(:,29);
```
```
SNOe=       settlerpart(:,27);
```

```
IQvec=(BSS*SSin+BCOD*CODin+BNKj*SNKjin+BNO*SNOin+BBOD5*BOD5in).*Qevec;
```
```
IQvec=(BSS*SSin+BCOD*CODin+BNKj*SNKjin+BNO*SNOin+BBOD5*BOD5in).*Qinvec;
```

```
disp(['Effluent average BOD5 conc = ',num2str(0.65*(SSeav+XSeav+(1-
f_P)*(XBHeav+XBAeav + XBAeav))),' mg/l (limit = 10 mg/l)'])
```
```
disp(['Effluent average BOD5 conc = ',num2str(0.25*(SSeav+XSeav+(1-
f_P)*(XBHeav+XBAeav + XSTOeav))),' mg/l (limit = 10 mg/l)'])
```

asm3init.m

```
X_SS_ASin = (0.75*X_S_ASin + 0.75*X_S_ASin + 0.90*X_H_ASin + 0.90*X_STO_ASin +
0.90*X_A_ASin);
```

```
X_SS_ASin = (0.75*X_S_ASin + 0.75*X_S_ASin + 0.90*X_H_ASin + 0.60*X_STO_ASin +
0.90*X_A_ASin) ;
```

asm3.c

```
KLa_temp = u[20]*pow(1.024, (u[14]-15.0));
```

```
KLa_temp = u[20]*pow(1.024, (u[14]-20.0));
```

A2. Corrections to ASM3 bioP

NOTE: Highlighted in grey are the errors in ASM3 bioP (model A1: pre-existing implementation) while statements in blue are the corrected statements.

asm3_bioP.c

```
proc11 = b_A_Temp * S_O_MonodTerm_X_H * xtemp[15];
```

```
proc11 = b_A_Temp * S_O_MonodTerm_X_A * xtemp[15];
```

```
proc12 = b_A_Temp * n_NO_end_A * S_O_InhibitionTerm_X_H * S_NO_MonodTerm_X_H *
xtemp[15];
```

```
proc12 = b_A_Temp * n_NO_end_A * S_O_InhibitionTerm_X_A * S_NO_MonodTerm_X_H *
xtemp[15];
```

A3. Matlab code for ASM3 bioP

```
#define S_FUNCTION_NAME asm3_bioP

#include "simstruc.h"
#include <math.h>

#define XINIT       ssGetArg(S,0)
#define PAR         ssGetArg(S,1)
#define V           ssGetArg(S,2)
#define SOSAT       ssGetArg(S,3)
#define TEMPMODEL   ssGetArg(S,4)
#define ACTIVATE    ssGetArg(S,5)


// mdlInitializeSizes - initialize the sizes array
static void mdlInitializeSizes(SimStruct *S)

{
    ssSetNumContStates      (S, 24);        // number of continuous states
    ssSetNumDiscStates      (S, 0);         // number of discrete states
    ssSetNumInputs          (S, 25);        // number of inputs
    ssSetNumOutputs         (S, 24);        // number of outputs
    ssSetDirectFeedThrough  (S, 1);         // direct feedthrough flag
    ssSetNumSampleTimes     (S, 1);         // number of sample times
    ssSetNumSFcnParams      (S, 6);         // number of input arguments
    ssSetNumRWork           (S, 0);         // number of real work vector elements
    ssSetNumIWork           (S, 0);         // number of integer work vector elements
    ssSetNumPWork           (S, 0);         // number of pointer work vector elements
}

// mdlInitializeSampleTimes - initialize the sample times array
static void mdlInitializeSampleTimes(SimStruct *S)

{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


// mdlInitializeConditions - initialize the states
static void mdlInitializeConditions(double *x0, SimStruct *S)

{
int i;

for (i = 0; i < 24; i++) {
   x0[i] = mxGetPr(XINIT)[i];
}
}

// mdlOutputs - compute the outputs

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S, int tid)
{

  double tempmodel, activate;
  int i;

  tempmodel = mxGetPr(TEMPMODEL)[0];
  activate  = mxGetPr(ACTIVATE)[0];

  for (i = 0; i < 17; i++) {
      y[i] = x[i];
  }

      y[17] = u[17];                    // Flow rate

  if (tempmodel < 0.5)
      y[18] = u[18];                    // Temp
  else
      y[18] = x[18];

  // dummy states, only give outputs if ACTIVATE = 1
  if (activate > 0.5) {
      y[19] = x[19];
      y[20] = x[20];
      y[21] = x[21];
      y[22] = x[22];
      y[23] = x[23];
      }
  else if (activate < 0.5) {
      y[19] = 0.0;
      y[20] = 0.0;
      y[21] = 0.0;
      y[22] = 0.0;
      y[23] = 0.0;
      }
}

// mdlUpdate - perform action at major integration time step
static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

// mdlDerivatives - compute the derivatives
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct *S, int tid)

{
double f_SI, Y_H_O2, Y_H_NO, Y_STO_O2, Y_STO_NO, f_XI, Y_PAO_O2, Y_PAO_NO, Y_PHA, Y_PO4,
Y_AUT;
double iN_SS, iN_SI, iN_XI, iN_XS, iN_BM, iP_SS, iP_SI, iP_XI, iP_XS, iP_BM, iTSS_XI, iTSS_XS,
iTSS_XSTO, iTSS_BM, iTSS_XPP, iNOX_N2, iCOD_NOX, iCOD_N2, icharge_NHX, icharge_NOX,
icharge_XPAO_PP, icharge_PO4;
double k_H, K_X, k_STO, mu_H, h_NO_H, K_SS_H, K_STO_H, b_H, h_NO_end_H, b_STO, K_O_H, K_NO_H,
K_NH_H, K_PO4_H, K_HCO_H;
double q_PHA, q_PP, K_PP_PAO, K_max_PAO, K_iPP_PAO, mu_PAO, h_NO_PAO, K_PHA, b_PAO,
h_NO_end_PAO, b_PP, h_NO_lys_PP, b_PHA, h_NO_resp_PHA, K_SS_PAO, K_O_PAO, K_NO_PAO, K_NH_PAO,
K_PO4_PP, K_PO4_PAO, K_HCO_PAO;
double mu_A, b_A, h_NO_A, K_O_A, K_NH_A, K_PO4_A, K_HCO_A;
double proc1, proc2, proc3, proc4, proc5, proc6, proc7, proc8, proc9, proc10, proc11, proc12;
double procP1, procP2, procP3, procP4, procP5, procP6, procP7, procP8, procP9, procP10,
procP11;
double reac1, reac2, reac3, reac4, reac5, reac6, reac7, reac8_1, reac8_2, reac8_3, reac8_4,
reac8, reac9, reac10, reac11, reac12, reac13, reac14, reac15, reac16, reac17_1, reac17_2,
reac17, reac18, reac19, reac20, reac21, reac22;
double vol, SO_sat, SO_sat_temp, KLa_temp;
double xtemp[24];
double tempmodel;
double activate;
double k_H_T, k_STO_T, mu_H_T, b_H_T, b_STO_T, mu_A_T, b_A_T, q_PHA_T, q_PP_T, mu_PAO_T,
b_PAO_T, b_PP_T, b_PHA_T;

int i;


// Stoichiometric parameters
f_SI            = mxGetPr(PAR)[0];
Y_H_O2          = mxGetPr(PAR)[1];
Y_H_NO          = mxGetPr(PAR)[2];
Y_STO_O2        = mxGetPr(PAR)[3];
Y_STO_NO        = mxGetPr(PAR)[4];
f_XI            = mxGetPr(PAR)[5];
Y_PAO_O2        = mxGetPr(PAR)[6];
Y_PAO_NO        = mxGetPr(PAR)[7];
Y_PHA           = mxGetPr(PAR)[8];
Y_PO4           = mxGetPr(PAR)[9];
Y_AUT           = mxGetPr(PAR)[10];
iN_SS           = mxGetPr(PAR)[11];
iN_SI           = mxGetPr(PAR)[12];
iN_XI           = mxGetPr(PAR)[13];
iN_XS           = mxGetPr(PAR)[14];
iN_BM           = mxGetPr(PAR)[15];
iP_SS           = mxGetPr(PAR)[16];
iP_SI           = mxGetPr(PAR)[17];
iP_XI           = mxGetPr(PAR)[18];
iP_XS           = mxGetPr(PAR)[19];
iP_BM           = mxGetPr(PAR)[20];
iTSS_XI         = mxGetPr(PAR)[21];
iTSS_XS         = mxGetPr(PAR)[22];
iTSS_XSTO       = mxGetPr(PAR)[23];
iTSS_BM         = mxGetPr(PAR)[24];
iTSS_XPP        = mxGetPr(PAR)[25];
iNOX_N2         = mxGetPr(PAR)[26];
```

```
iCOD_NOX        = mxGetPr(PAR)[27];
iCOD_N2         = mxGetPr(PAR)[28];
icharge_NHX     = mxGetPr(PAR)[29];
icharge_NOX     = mxGetPr(PAR)[30];
icharge_XPAO_PP = mxGetPr(PAR)[31];
icharge_PO4     = mxGetPr(PAR)[32];

// Kinetic parameters (with process numbers in which they are included)
k_H             = mxGetPr(PAR)[33]; /* 1 */
K_X             = mxGetPr(PAR)[34]; /* 1 */
k_STO           = mxGetPr(PAR)[35]; /* 2,3,4 */
mu_H            = mxGetPr(PAR)[36]; /* 4,5 */
h_NO_H          = mxGetPr(PAR)[37]; /* 3,5 */
K_SS_H          = mxGetPr(PAR)[38]; /* 2,3 */
K_STO_H         = mxGetPr(PAR)[39]; /* 5 */
b_H             = mxGetPr(PAR)[40]; /* 6,7 */
h_NO_end_H      = mxGetPr(PAR)[41]; /* 7,9 */
b_STO           = mxGetPr(PAR)[42]; /* 8,9 */
K_O_H           = mxGetPr(PAR)[43]; /* 2,3,4,5,6,7,8,9 */
K_NO_H          = mxGetPr(PAR)[44]; /* 3,5,7,9,12 */
K_NH_H          = mxGetPr(PAR)[45]; /* 4,5 */
K_PO4_H         = mxGetPr(PAR)[46]; /* 4,5 */
K_HCO_H         = mxGetPr(PAR)[47]; /* 4,5 */
q_PHA           = mxGetPr(PAR)[48]; /* P1 */
q_PP            = mxGetPr(PAR)[49]; /* P2,P3 */
K_PP_PAO        = mxGetPr(PAR)[50]; /* P1 */
K_max_PAO       = mxGetPr(PAR)[51]; /* P2,P3 */
K_iPP_PAO       = mxGetPr(PAR)[52]; /* P2,P3 */
mu_PAO          = mxGetPr(PAR)[53]; /* P4,P5 */
h_NO_PAO        = mxGetPr(PAR)[54]; /* P3,P5 */
K_PHA           = mxGetPr(PAR)[55]; /* P2,P3,P4,P5 */
b_PAO           = mxGetPr(PAR)[56]; /* P6,P7 */
h_NO_end_PAO    = mxGetPr(PAR)[57]; /* P7 */
b_PP            = mxGetPr(PAR)[58]; /* P8,P9 */
h_NO_lys_PP     = mxGetPr(PAR)[59]; /* P9 */
b_PHA           = mxGetPr(PAR)[60]; /* P10,P11 */
h_NO_resp_PHA   = mxGetPr(PAR)[61]; /* P11 */
K_SS_PAO        = mxGetPr(PAR)[62]; /* P1 */
K_O_PAO         = mxGetPr(PAR)[63]; /* P2,P3,P4,P5,P6,P7,P8,P9,P10,P11 */
K_NO_PAO        = mxGetPr(PAR)[64]; /* P3,P5,P7,P9,P11 */
K_NH_PAO        = mxGetPr(PAR)[65]; /* P4,P5 */
K_PO4_PP        = mxGetPr(PAR)[66]; /* P3 */
K_PO4_PAO       = mxGetPr(PAR)[67]; /* P2,P4,P5 */
K_HCO_PAO       = mxGetPr(PAR)[68]; /* P1,P2,P3,P4,P5,P8,P9 */
mu_A            = mxGetPr(PAR)[69]; /* 10 */
b_A             = mxGetPr(PAR)[70]; /* 11,12 */
h_NO_A          = mxGetPr(PAR)[71]; /* 12 */
K_O_A           = mxGetPr(PAR)[72]; /* 10,11,12 */
K_NH_A          = mxGetPr(PAR)[73]; /* 10 */
K_PO4_A         = mxGetPr(PAR)[74]; /* 10 */
K_HCO_A         = mxGetPr(PAR)[75]; /* 10 */


vol             = mxGetPr(V)[0];
SO_sat          = mxGetPr(SOSAT)[0];

tempmodel       = mxGetPr(TEMPMODEL)[0];
activate        = mxGetPr(ACTIVATE)[0];



// temperature compensation
if (tempmodel < 0.5) {
    // Compensation from the temperature at the influent of the reactor
    k_H_T         = k_H   *exp(0.04 *(u[18]-20));
    k_STO_T       = k_STO *exp(0.07 *(u[18]-20));
    mu_H_T        = mu_H  *exp(0.07 *(u[18]-20));
    b_H_T         = b_H   *exp(0.07 *(u[18]-20));
    b_STO_T       = b_STO *exp(0.07 *(u[18]-20));
    mu_A_T        = mu_A  *exp(0.105*(u[18]-20));
    b_A_T         = b_A   *exp(0.105*(u[18]-20));

    q_PHA_T       = q_PHA *exp(0.04 *(u[18]-20));
    q_PP_T        = q_PP  *exp(0.04 *(u[18]-20));
    mu_PAO_T      = mu_PAO*exp(0.07 *(u[18]-20));
    b_PAO_T       = b_PAO *exp(0.07 *(u[18]-20));
    b_PP_T        = b_PP  *exp(0.07 *(u[18]-20));
    b_PHA_T       = b_PHA *exp(0.07 *(u[18]-20));

    SO_sat_temp = 0.9997743214*8.0/10.5*(56.12*6791.5*exp(-66.7354 +
87.4755/((u[18]+273.15)/100.0) + 24.4526*log((u[18]+273.15)/100.0))); /* van't Hoff equation
*/
    KLa_temp    = u[24]*pow(1.024,(u[18]-20));
}
else {
    // Compensation from the current temperature in the reactor
    k_H_T         = k_H   *exp(0.04 *(x[18]-20));
    k_STO_T       = k_STO *exp(0.07 *(x[18]-20));
    mu_H_T        = mu_H  *exp(0.07 *(x[18]-20));
    b_H_T         = b_H   *exp(0.07 *(x[18]-20));
    b_STO_T       = b_STO *exp(0.07 *(x[18]-20));
    mu_A_T        = mu_A  *exp(0.105*(x[18]-20));
    b_A_T         = b_A   *exp(0.105*(x[18]-20));

    q_PHA_T       = q_PHA *exp(0.04 *(x[18]-20));
    q_PP_T        = q_PP  *exp(0.04 *(x[18]-20));
    mu_PAO_T      = mu_PAO*exp(0.07 *(x[18]-20));
    b_PAO_T       = b_PAO *exp(0.07 *(x[18]-20));
    b_PP_T        = b_PP  *exp(0.07 *(x[18]-20));
    b_PHA_T       = b_PHA *exp(0.07 *(x[18]-20));

    SO_sat_temp = 0.9997743214*8.0/10.5*(56.12*6791.5*exp(-66.7354 +
87.4755/((x[18]+273.15)/100.0) + 24.4526*log((x[18]+273.15)/100.0))); /* van't Hoff equation
*/
    KLa_temp    = u[24]*pow(1.024,(x[18]-20));
}


for (i = 0; i < 24; i++) {
   if (x[i] < 0.0)
     xtemp[i] = 0.0;
   else
     xtemp[i] = x[i];
}

if (u[24] < 0.0)
      x[0] = fabs(u[24]);       // u[24]=KLa value




// (Reference: Rieger et al., 2001; Hauduc et al., 2010)

proc1    = k_H_T*((xtemp[9]/xtemp[10])/(K_X+xtemp[9]/xtemp[10]))*xtemp[10];
proc2    = k_STO_T*(xtemp[0]/(K_O_H+xtemp[0]))*(xtemp[1]/(K_SS_H+xtemp[1]))*xtemp[10];
proc3    =
k_STO_T*h_NO_H*(K_O_H/(K_O_H+xtemp[0]))*(xtemp[4]/(K_NO_H+xtemp[4]))*(xtemp[1]/(K_SS_H+xtemp[1
]))*xtemp[10];
proc4    =
mu_H_T*(xtemp[0]/(K_O_H+xtemp[0]))*(xtemp[3]/(K_NH_H+xtemp[3]))*(xtemp[7]/(K_HCO_H+xtemp[7]))*
(xtemp[6]/(K_PO4_H+xtemp[6]))*((xtemp[11]/xtemp[10])/(K_STO_H+xtemp[11]/xtemp[10]))*xtemp[10];
proc5    =
mu_H_T*h_NO_H*(K_O_H/(K_O_H+xtemp[0]))*(xtemp[4]/(K_NO_H+xtemp[4]))*(xtemp[3]/(K_NH_H+xtemp[3]
))*(xtemp[7]/(K_HCO_H+xtemp[7]))*(xtemp[6]/(K_PO4_H+xtemp[6]))*((xtemp[11]/xtemp[10])/(K_STO_H
+xtemp[11]/xtemp[10]))*xtemp[10];
proc6    = b_H_T*(xtemp[0]/(K_O_H+xtemp[0]))*xtemp[10];
proc7    = b_H_T*h_NO_end_H*(K_O_H/(K_O_H+xtemp[0]))*(xtemp[4]/(K_NO_H+xtemp[4]))*xtemp[10];
proc8    = b_STO_T*(xtemp[0]/(K_O_H+xtemp[0]))*xtemp[11];
proc9    = b_STO_T*h_NO_end_H*(K_O_H/(K_O_H+xtemp[0]))*(xtemp[4]/(K_NO_H+xtemp[4]))*xtemp[11];
proc10   =
mu_A_T*(xtemp[0]/(K_O_A+xtemp[0]))*(xtemp[3]/(K_NH_A+xtemp[3]))*(xtemp[7]/(K_HCO_A+xtemp[7]))*
(xtemp[6]/(K_PO4_A+xtemp[6]))*xtemp[15];
proc11   = b_A_T*(xtemp[0]/(K_O_A+xtemp[0]))*xtemp[15];
proc12   = b_A_T*h_NO_A*(K_O_A/(K_O_A+xtemp[0]))*(xtemp[4]/(K_NO_H+xtemp[4]))*xtemp[15];

procP1   =
q_PHA_T*(xtemp[1]/(K_SS_PAO+xtemp[1]))*(xtemp[7]/(K_HCO_PAO+xtemp[7]))*((xtemp[13]/xtemp[12])/
(K_PP_PAO+xtemp[13]/xtemp[12]))*xtemp[12];
procP2   =
q_PP_T*(xtemp[0]/(K_O_PAO+xtemp[0]))*(xtemp[6]/(K_PO4_PP+xtemp[6]))*(xtemp[7]/(K_HCO_PAO+xtemp
[7]))*((xtemp[14]/xtemp[12])/(K_PHA+xtemp[14]/xtemp[12]))*((K_max_PAO-
(xtemp[13]/xtemp[12]))/(K_iPP_PAO+K_max_PAO-(xtemp[13]/xtemp[12])))*xtemp[12];
procP3   =
q_PP_T*h_NO_PAO*(K_O_PAO/(K_O_PAO+xtemp[0]))*(xtemp[4]/(K_NO_PAO+xtemp[4]))*(xtemp[6]/(K_PO4_P
P+xtemp[6]))*(xtemp[7]/(K_HCO_PAO+xtemp[7]))*((xtemp[14]/xtemp[12])/(K_PHA+xtemp[14]/xtemp[12]
))*((K_max_PAO-(xtemp[13]/xtemp[12]))/(K_iPP_PAO+K_max_PAO-(xtemp[13]/xtemp[12])))*xtemp[12];
procP4  =
mu_PAO_T*(xtemp[0]/(K_O_PAO+xtemp[0]))*(xtemp[3]/(K_NH_PAO+xtemp[3]))*(xtemp[6]/(K_PO4_PAO+xte
mp[6]))*(xtemp[7]/(K_HCO_PAO+xtemp[7]))*((xtemp[14]/xtemp[12])/(K_PHA+xtemp[14]/xtemp[12]))*xt
emp[12];
procP5  =
mu_PAO_T*h_NO_PAO*(K_O_PAO/(K_O_PAO+xtemp[0]))*(xtemp[4]/(K_NO_PAO+xtemp[4]))*(xtemp[3]/(K_NH_
PAO+xtemp[3]))*(xtemp[6]/(K_PO4_PAO+xtemp[6]))*(xtemp[7]/(K_HCO_PAO+xtemp[7]))*((xtemp[14]/xte
mp[12])/(K_PHA+xtemp[14]/xtemp[12]))*xtemp[12];
procP6  = b_PAO_T*(xtemp[0]/(K_O_PAO+xtemp[0]))*xtemp[12];
procP7  =
b_PAO_T*h_NO_end_PAO*(K_O_PAO/(K_O_PAO+xtemp[0]))*(xtemp[4]/(K_NO_PAO+xtemp[4]))*xtemp[12];
procP8  = b_PP_T*(xtemp[0]/(K_O_PAO+xtemp[0]))*(xtemp[7]/(K_HCO_PAO+xtemp[7]))*xtemp[13];
procP9  =
b_PP_T*h_NO_lys_PP*(K_O_PAO/(K_O_PAO+xtemp[0]))*(xtemp[4]/(K_NO_PAO+xtemp[4]))*(xtemp[7]/(K_HC
O_PAO+xtemp[7]))*xtemp[13];
procP10 = b_PHA_T*(xtemp[0]/(K_O_PAO+xtemp[0]))*xtemp[14];
procP11 =
b_PHA_T*h_NO_resp_PHA*(K_O_PAO/(K_O_PAO+xtemp[0]))*(xtemp[4]/(K_NO_PAO+xtemp[4]))*xtemp[14];


reac1   = (Y_STO_O2-1)*proc2 + (1-1/Y_H_O2)*proc4 + (f_XI-1)*(proc6+proc11+procP6) + (-
1)*(proc8+procP10) + (1+iCOD_NOX/Y_AUT)*proc10 + (-Y_PHA)*procP2 + (1-1/Y_PAO_O2)*procP4;
reac2   = (1-f_SI)*proc1 + (-1)*(proc2+proc3+procP1);
reac3   = (f_SI)*proc1;
reac4   = ((f_SI-1)*iN_SS-f_SI*iN_SI+iN_XS)*proc1 + (iN_SS)*(proc2+proc3+procP1) + (-
iN_BM)*(proc4+proc5+procP4+procP5) + (iN_BM-
f_XI*iN_XI)*(proc6+proc7+proc11+proc12+procP6+procP7) + (-1/Y_AUT-iN_BM)*proc10;
reac5   = ((Y_STO_NO-1)/iNOX_N2)*proc3 + ((1-1/Y_H_NO)/iNOX_N2)*proc5 + ((f_XI-
1)/iNOX_N2)*(proc7+proc12+procP7) + (-1/iNOX_N2)*(proc9+procP11) + (1/Y_AUT)*proc10 + (-
Y_PHA/iNOX_N2)*procP3 + ((1-1/Y_PAO_NO)/iNOX_N2)*procP5;
reac6   = ((1-Y_STO_NO)/iNOX_N2)*proc3 + ((1/Y_H_NO-1)/iNOX_N2)*proc5 + ((1-
f_XI)/iNOX_N2)*(proc7+proc12+procP7) + (1/iNOX_N2)*(proc9+procP11) + (Y_PHA/iNOX_N2)*procP3 +
((1/Y_PAO_NO-1)/iNOX_N2)*procP5;
reac7   = ((f_SI-1)*iP_SS-f_SI*iP_SS+iP_XS)*proc1 + (iP_SS)*(proc2+proc3) + (-
iP_BM)*(proc4+proc5+proc10+procP4+procP5) + (iP_BM-
f_XI*iP_XI)*(proc6+proc7+proc11+proc12+procP6+procP7) + (Y_PO4+iP_SS)*procP1 + (-
1)*(procP2+procP3) + (1)*(procP8+procP9);
reac8_1 = (((f_SI-1)*iN_SS-f_SI*iN_SI+iN_XS)*icharge_NHX+((f_SI-1)*iP_SS-
f_SI*iP_SI+iP_XS)*icharge_PO4)*proc1 + (iN_SS*icharge_NHX+iP_SS*icharge_PO4)*proc2 +
(iN_SS*icharge_NHX+(Y_STO_NO-1)/iNOX_N2*icharge_NOX+iP_SS*icharge_PO4)*proc3 + (-
iN_BM*icharge_NHX+(-iP_BM)*icharge_PO4)*proc4 + (-iN_BM*icharge_NHX+(1-
1/Y_H_NO)/iNOX_N2*icharge_NOX+(-iP_BM)*icharge_PO4)*proc5;
reac8_2 = ((iN_BM-f_XI*iN_XI)*icharge_NHX+(iP_BM-f_XI*iP_XI)*icharge_PO4)*proc6 + ((iN_BM-
f_XI*iN_XI)*icharge_NHX+(f_XI-1)/iNOX_N2*icharge_NOX+(iP_BM-f_XI*iP_XI)*icharge_PO4)*proc7 +
(-1/iNOX_N2*icharge_NOX)*proc9 + ((-1/Y_AUT-iN_BM)*icharge_NHX+(1/Y_AUT)*icharge_NOX+(-
iP_BM)*icharge_PO4)*proc10 + ((iN_BM-f_XI*iN_XI)*icharge_NHX+(iP_BM-
f_XI*iP_XI)*icharge_PO4)*proc11 + ((iN_BM-f_XI*iN_XI)*icharge_NHX+(f_XI-
1)/iNOX_N2*icharge_NOX+(iP_BM-f_XI*iP_XI)*icharge_PO4)*proc12;
reac8_3 = (iN_SS*icharge_NHX+(Y_PO4+iP_SS)*icharge_PO4+(-Y_PO4)*icharge_XPAO_PP)*procP1 + (-
1*icharge_PO4+1*icharge_XPAO_PP)*procP2 + (-Y_PHA/iNOX_N2*icharge_NOX+(-
1)*icharge_PO4+1*icharge_XPAO_PP)*procP3 + (-iN_BM*icharge_NHX+(-iP_BM)*icharge_PO4)*procP4 +
(-iN_BM*icharge_NHX+(1-1/Y_PAO_NO)/iNOX_N2*icharge_NOX+(-iP_BM)*icharge_PO4)*procP5;
reac8_4 = ((iN_BM-f_XI*iN_XI)*icharge_NHX+(iP_BM-f_XI*iP_XI)*icharge_PO4)*procP6 + ((iN_BM-
f_XI*iN_XI)*icharge_NHX+(f_XI-1)/iNOX_N2*icharge_NOX+(iP_BM-f_XI*iP_XI)*icharge_PO4)*procP7 +
(1*icharge_PO4+(-1)*icharge_XPAO_PP)*(procP8+procP9) + (-1/iNOX_N2*icharge_NOX)*procP11;
reac8   = reac8_1+reac8_2+reac8_3+reac8_4;
reac9   = (f_XI)*(proc6+proc7+proc11+proc12+procP6+procP7);
reac10  = (-1)*proc1;
reac11  = (1)*(proc4+proc5) + (-1)*(proc6+proc7);
reac12  = (Y_STO_O2)*proc2 + (Y_STO_NO)*proc3 + (-1/Y_H_O2)*proc4 + (-1/Y_H_NO)*proc5 + (-
1)*(proc8+proc9);
reac13  = (1)*(procP4+procP5) + (-1)*(procP6+procP7);
reac14  = (-Y_PO4)*procP1 + (1)*(procP2+procP3) + (-1)*(procP8+procP9);
reac15  = (1)*procP1 + (-Y_PHA)*(procP2+procP3) + (-1/Y_PAO_O2)*procP4 + (-1/Y_PAO_NO)*procP5
+ (-1)*(procP10+procP11);
reac16  = (1)*proc10 + (-1)*(proc11+proc12);
reac17_1= (-iTSS_XS)*proc1 + (Y_STO_O2*iTSS_XSTO)*proc2 + (Y_STO_NO*iTSS_XSTO)*proc3 +
(iTSS_BM-iTSS_XSTO/Y_H_O2)*proc4 + (iTSS_BM-iTSS_XSTO/Y_H_NO)*proc5 + (f_XI*iTSS_XI-
iTSS_BM)*(proc6+proc7+proc11+proc12+procP6+procP7) + (-
iTSS_XSTO)*(proc8+proc9+procP10+procP11);
reac17_2= (iTSS_BM)*proc10 + (iTSS_XSTO-iTSS_XPP*Y_PO4)*procP1 + (iTSS_XPP-
Y_PHA*iTSS_XSTO)*(procP2+procP3) + (iTSS_BM-iTSS_XSTO/Y_PAO_O2)*procP4 + (iTSS_BM-
iTSS_XSTO/Y_PAO_NO)*procP5 + (-iTSS_XPP)*(procP8+procP9);
reac17  = reac17_1+reac17_2;
```

```
reac18  = 0.0;      // for dummy states
reac19  = 0.0;
reac20  = 0.0;
reac21  = 0.0;
reac22  = 0.0;


if (u[24] < 0.0)
    dx[0] = 0.0;
else
    dx[0] = 1.0/vol*(u[17]*(u[0]-x[0])) + reac1 + KLa_temp*(SO_sat_temp-x[0]);

dx[1]   = 1.0/vol*(u[17]*(u[1] -x[1]))  + reac2;
dx[2]   = 1.0/vol*(u[17]*(u[2] -x[2]))  + reac3;
dx[3]   = 1.0/vol*(u[17]*(u[3] -x[3]))  + reac4;
dx[4]   = 1.0/vol*(u[17]*(u[4] -x[4]))  + reac5;
dx[5]   = 1.0/vol*(u[17]*(u[5] -x[5]))  + reac6;
dx[6]   = 1.0/vol*(u[17]*(u[6] -x[6]))  + reac7;
dx[7]   = 1.0/vol*(u[17]*(u[7] -x[7]))  + reac8;
dx[8]   = 1.0/vol*(u[17]*(u[8] -x[8]))  + reac9;
dx[9]   = 1.0/vol*(u[17]*(u[9] -x[9]))  + reac10;
dx[10]  = 1.0/vol*(u[17]*(u[10]-x[10])) + reac11;
dx[11]  = 1.0/vol*(u[17]*(u[11]-x[11])) + reac12;
dx[12]  = 1.0/vol*(u[17]*(u[12]-x[12])) + reac13;
dx[13]  = 1.0/vol*(u[17]*(u[13]-x[13])) + reac14;
dx[14]  = 1.0/vol*(u[17]*(u[14]-x[14])) + reac15;
dx[15]  = 1.0/vol*(u[17]*(u[15]-x[15])) + reac16;
dx[16]  = 1.0/vol*(u[17]*(u[16]-x[16])) + reac17;

dx[17]  = 0.0;                      // Flowrate

if (tempmodel < 0.5)                // Temperature
   dx[18] = 0.0;
else
   dx[18] = 1.0/vol*(u[17]*(u[18]-x[18]));

// dummy states
dx[19]  = 1.0/vol*(u[17]*(u[19]-x[19])) + reac18;
dx[20]  = 1.0/vol*(u[17]*(u[20]-x[20])) + reac19;
dx[21]  = 1.0/vol*(u[17]*(u[21]-x[21])) + reac20;
dx[22]  = 1.0/vol*(u[17]*(u[22]-x[22])) + reac21;
dx[23]  = 1.0/vol*(u[17]*(u[23]-x[23])) + reac22;

}


// mdlTerminate - called when the simulation is terminated.

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
```

A4. Matlab code for ASM3

```
#define S_FUNCTION_NAME asm3

#include "simstruc.h"
#include <math.h>

#define XINIT   ssGetArg(S,0)
#define PAR ssGetArg(S,1)
#define V   ssGetArg(S,2)
#define SOSAT   ssGetArg(S,3)
#define TEMPMODEL  ssGetArg(S,4)
#define ACTIVATE  ssGetArg(S,5)


/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 20);  /* number of continuous states           */
    ssSetNumDiscStates(    S, 0);   /* number of discrete states             */
    ssSetNumInputs(        S, 21);  /* number of inputs                      */
    ssSetNumOutputs(       S, 24);  /* number of outputs                     */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag               */
    ssSetNumSampleTimes(   S, 1);   /* number of sample times                */
    ssSetNumSFcnParams(    S, 6);   /* number of input arguments             */
    ssSetNumRWork(         S, 0);   /* number of real work vector elements   */
    ssSetNumIWork(         S, 0);   /* number of integer work vector elements*/
    ssSetNumPWork(         S, 0);   /* number of pointer work vector elements*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
int i;

for (i = 0; i < 20; i++) {
   x0[i] = mxGetPr(XINIT)[i];
}
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S, int tid)
{
  double i_NSI, i_NSS, i_NXI, i_NXS, i_NBM, i_SSXI, i_SSXS, i_SSBM, i_SSSTO;
  double tempmodel, activate;

  int i;

i_NSI = mxGetPr(PAR)[28];
i_NSS = mxGetPr(PAR)[29];
i_NXI = mxGetPr(PAR)[30];
i_NXS = mxGetPr(PAR)[31];
i_NBM = mxGetPr(PAR)[32];
i_SSXI = mxGetPr(PAR)[33];
i_SSXS = mxGetPr(PAR)[34];
i_SSBM = mxGetPr(PAR)[35];
i_SSSTO = mxGetPr(PAR)[36]; /* not properly defined in ASM3 report */

tempmodel = mxGetPr(TEMPMODEL)[0];
activate = mxGetPr(ACTIVATE)[0];
```

```
   for (i = 0; i < 13; i++) {
      y[i] = x[i];
  }

  y[13] = u[13];  /*  Q */

  if (tempmodel < 0.5)                            /* Temp */
     y[14] = u[14];
  else
    y[14] = x[14];


  /* dummy states, only give outputs if ACTIVATE = 1 */
  if (activate > 0.5) {
      y[15] = x[15];
      y[16] = x[16];
      y[17] = x[17];
      y[18] = x[18];
      y[19] = x[19];
      }
  else if (activate < 0.5) {
      y[15] = 0.0;
      y[16] = 0.0;
      y[17] = 0.0;
      y[18] = 0.0;
      y[19] = 0.0;
      }

     y[20] = -x[0]+x[1]+x[2]-24.0/14.0*x[4]-64.0/14.0*x[5]+x[7]+x[8]+x[9]+x[10]+x[11];
     y[21] =
i_NSI*x[1]+i_NSS*x[2]+x[3]+x[4]+x[5]+i_NXI*x[7]+i_NXS*x[8]+i_NBM*x[9]+i_NBM*x[11];
     y[22] = 1.0/14.0*x[3]-1.0/14.0*x[5]-x[6];
     y[23] = i_SSXI*x[7]+i_SSXS*x[8]+i_SSBM*x[9]+i_SSSTO*x[10]+i_SSBM*x[11];

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct *S, int tid)
{

double k_H, K_X, k_STO, ny_NOX, K_O2, K_NOX, K_S, K_STO, mu_H, K_NH4, K_ALK, b_HO2, b_HNOX,
b_STOO2, b_STONOX, mu_A, K_ANH4, K_AO2, K_AALK, b_AO2, b_ANOX;
double f_SI, Y_STOO2, Y_STONOX, Y_HO2, Y_HNOX, Y_A, f_XI, i_NSI, i_NSS, i_NXI, i_NXS, i_NBM,
i_SSXI, i_SSXS, i_SSBM, i_SSSTO;
double x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12;
double y1, y2, y3, y4, y5, y6, y7, y10, y11, y12;
double z1, z2, z3, z4, z5, z6, z7, z9, z10, z11, z12;
double t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12;
double proc1, proc2, proc3, proc4, proc5, proc6, proc7, proc8, proc9, proc10, proc11, proc12;
double reac1, reac2, reac3, reac4, reac5, reac6, reac7, reac8, reac9, reac10, reac11, reac12,
reac13, reac15 ,reac16, reac17, reac18, reac19;
double vol, SO_sat, T;
double xtemp[20];
double tempmodel;
double SO_sat_temp, KLa_temp;

int i;

k_H = mxGetPr(PAR)[0];
K_X = mxGetPr(PAR)[1];
k_STO = mxGetPr(PAR)[2];
ny_NOX = mxGetPr(PAR)[3];
K_O2 = mxGetPr(PAR)[4];
K_NOX = mxGetPr(PAR)[5];
K_S = mxGetPr(PAR)[6];
```

```
K_STO = mxGetPr(PAR)[7];
mu_H = mxGetPr(PAR)[8];
K_NH4 = mxGetPr(PAR)[9];
K_ALK = mxGetPr(PAR)[10];
b_HO2 = mxGetPr(PAR)[11];
b_HNOX = mxGetPr(PAR)[12];
b_STOO2 = mxGetPr(PAR)[13];
b_STONOX = mxGetPr(PAR)[14];
mu_A = mxGetPr(PAR)[15];
K_ANH4 = mxGetPr(PAR)[16];
K_AO2 = mxGetPr(PAR)[17];
K_AALK = mxGetPr(PAR)[18];
b_AO2 = mxGetPr(PAR)[19];
b_ANOX = mxGetPr(PAR)[20];
f_SI = mxGetPr(PAR)[21];
Y_STOO2 = mxGetPr(PAR)[22];
Y_STONOX = mxGetPr(PAR)[23];
Y_HO2 = mxGetPr(PAR)[24];
Y_HNOX = mxGetPr(PAR)[25];
Y_A = mxGetPr(PAR)[26];
f_XI = mxGetPr(PAR)[27];
i_NSI = mxGetPr(PAR)[28];
i_NSS = mxGetPr(PAR)[29];
i_NXI = mxGetPr(PAR)[30];
i_NXS = mxGetPr(PAR)[31];
i_NBM = mxGetPr(PAR)[32];
i_SSXI = mxGetPr(PAR)[33];
i_SSXS = mxGetPr(PAR)[34];
i_SSBM = mxGetPr(PAR)[35];
i_SSSTO = mxGetPr(PAR)[36]; /* not properly defined in ASM3 report */
vol = mxGetPr(V)[0];
SO_sat = mxGetPr(SOSAT)[0];

tempmodel = mxGetPr(TEMPMODEL)[0];

/* temperature compensation */
if (tempmodel < 0.5) {
   /* Compensation from the temperature at the influent of the reactor */
   k_H = k_H*exp((log(k_H/2)/10.0)*(u[14]-20.0));

   k_STO = k_STO *exp((log(k_STO /2.5)/10.0)*(u[14]-20.0));
   mu_H = mu_H*exp((log(mu_H/1.0)/10.0)*(u[14]-20.0));
   b_HO2 = b_HO2*exp((log(b_HO2/0.1)/10.0)*(u[14]-20.0));
   b_HNOX = b_HNOX*exp((log(b_HNOX/0.05)/10.0)*(u[14]-20.0));
   b_STOO2 = b_STOO2*exp((log(b_STOO2/0.1)/10.0)*(u[14]-20.0));
   b_STONOX = b_STONOX*exp((log(b_STONOX/0.05)/10.0)*(u[14]-20.0));

   mu_A = mu_A*exp((log(mu_A/0.35)/10.0)*(u[14]-20.0));
   b_AO2 = b_AO2*exp((log(b_AO2/0.05)/10.0)*(u[14]-20.0));
   b_ANOX = b_ANOX*exp((log(b_ANOX/0.02)/10.0)*(u[14]-20.0));

   SO_sat_temp = 0.9997743214*8.0/10.5*(56.12*6791.5*exp(-66.7354 +
87.4755/((u[14]+273.15)/100.0) + 24.4526*log((u[14]+273.15)/100.0))); /* van't Hoff equation
*/
   KLa_temp = u[20]*pow(1.024, (u[14]-20.0));
}
else {
   /* Compensation from the current temperature in the reactor */

   k_H = k_H*exp((log(k_H/2)/10.0)*(x[14]-20.0));

   k_STO = k_STO *exp((log(k_STO /2.5)/10.0)*(x[14]-20.0));
   mu_H = mu_H*exp((log(mu_H/1.0)/10.0)*(x[14]-20.0));
   b_HO2 = b_HO2*exp((log(b_HO2/0.1)/10.0)*(x[14]-20.0));
   b_HNOX = b_HNOX*exp((log(b_HNOX/0.05)/10.0)*(x[14]-20.0));
   b_STOO2 = b_STOO2*exp((log(b_STOO2/0.1)/10.0)*(x[14]-20.0));
   b_STONOX = b_STONOX*exp((log(b_STONOX/0.05)/10.0)*(x[14]-20.0));

   mu_A = mu_A*exp((log(mu_A/0.35)/10.0)*(x[14]-20.0));
   b_AO2 = b_AO2*exp((log(b_AO2/0.05)/10.0)*(x[14]-20.0));
   b_ANOX = b_ANOX*exp((log(b_ANOX/0.02)/10.0)*(x[14]-20.0));

   SO_sat_temp = 0.9997743214*8.0/10.5*(56.12*6791.5*exp(-66.7354 +
87.4755/((x[14]+273.15)/100.0) + 24.4526*log((x[14]+273.15)/100.0))); /* van't Hoff equation
*/
   KLa_temp = u[20]*pow(1.024, (x[14]-20.0));
}


/* note 64.0/14.0 is not exactly equal to 4.57 */
/* note 24.0/14.0 is not exactly equal to 1.71 */

x1 = 1.0-f_SI;
x2 = -1.0+Y_STO02;
x3 = (-1.0+Y_STONOX)/(64.0/14.0-24.0/14.0);
x4 = 1.0-1.0/Y_HO2;
x5 = (+1.0-1.0/Y_HNOX)/(64.0/14.0-24.0/14.0);
x6 = -1.0+f_XI;
x7 = (f_XI-1.0)/(64.0/14.0-24.0/14.0);
x8 = -1.0;
x9 = -1.0/(64.0/14.0-24.0/14.0);
x10 = -(64.0/14.0)/Y_A+1.0;
x11 = f_XI-1.0;
x12 = (f_XI-1.0)/(64.0/14.0-24.0/14.0);

y1 = -f_SI*i_NSI-(1.0-f_SI)*i_NSS+i_NXS;
y2 = i_NSS;
y3 = i_NSS;
y4 = -i_NBM;
y5 = -i_NBM;
y6 = -f_XI*i_NXI+i_NBM;
y7 = -f_XI*i_NXI+i_NBM;
y10 = -1.0/Y_A-i_NBM;
y11 = -f_XI*i_NXI+i_NBM;
y12 = -f_XI*i_NXI+i_NBM;

z1 = y1/14.0;
z2 = y2/14.0;
z3 = y3/14.0-x3/14.0;
z4 = y4/14.0;
z5 = y5/14.0-x5/14.0;
z6 = y6/14.0;
z7 = y7/14.0-x7/14.0;
z9 = -x9/14.0;
z10 = y10/14.0-1.0/(Y_A*14.0);
z11 = y11/14.0;
z12 = y12/14.0-x12/14.0;

t1 = -i_SSXS;
t2 = Y_STO02*i_SSSTO;
t3 = Y_STONOX*i_SSSTO;
t4 = i_SSBM-1.0/Y_HO2*i_SSSTO;
t5 = i_SSBM-1.0/Y_HNOX*i_SSSTO;
t6 = f_XI*i_SSXI-i_SSBM;
t7 = f_XI*i_SSXI-i_SSBM;
t8 = -i_SSSTO;
t9 = -i_SSSTO;
t10 = i_SSBM;
t11 = f_XI*i_SSXI-i_SSBM;
t12 = f_XI*i_SSXI-i_SSBM;

T = 0.0001; /* time constant = 0.0001 d = 8.64 s */

for (i = 0; i < 20; i++) {
   if (x[i] < 0)
     xtemp[i] = 0;
   else
     xtemp[i] = x[i];
}


if (u[20] < 0)
     x[0] = fabs(u[20]);


proc1 = k_H*(xtemp[8]/xtemp[9])/(K_X+xtemp[8]/xtemp[9])*xtemp[9];
proc2 = k_STO*xtemp[0]/(K_O2+xtemp[0])*xtemp[2]/(K_S+xtemp[2])*xtemp[9];
proc3 =
k_STO*ny_NOX*K_O2/(K_O2+xtemp[0])*xtemp[5]/(K_NOX+xtemp[5])*xtemp[2]/(K_S+xtemp[2])*xtemp[9];
proc4 =
mu_H*xtemp[0]/(K_O2+xtemp[0])*xtemp[3]/(K_NH4+xtemp[3])*xtemp[6]/(K_ALK+xtemp[6])*(xtemp[10]/x
temp[9])/(K_STO+xtemp[10]/xtemp[9])*xtemp[9];
```

```
proc5 =
mu_H*ny_NOX*K_O2/(K_O2+xtemp[0])*xtemp[5]/(K_NOX+xtemp[5])*xtemp[3]/(K_NH4+xtemp[3])*xtemp[6]/
(K_ALK+xtemp[6])*(xtemp[10]/xtemp[9])/(K_STO+xtemp[10]/xtemp[9])*xtemp[9];
proc6 = b_HO2*xtemp[0]/(K_O2+xtemp[0])*xtemp[9];
proc7 = b_HNOX*K_O2/(K_O2+xtemp[0])*xtemp[5]/(K_NOX+xtemp[5])*xtemp[9];
proc8 = b_STOO2*xtemp[0]/(K_O2+xtemp[0])*xtemp[10];
proc9 = b_STONOX*K_O2/(K_O2+xtemp[0])*xtemp[5]/(K_NOX+xtemp[5])*xtemp[10];
proc10 =
mu_A*xtemp[0]/(K_AO2+xtemp[0])*xtemp[3]/(K_ANH4+xtemp[3])*xtemp[6]/(K_AALK+xtemp[6])*xtemp[11]
;
proc11 = b_AO2*xtemp[0]/(K_AO2+xtemp[0])*xtemp[11];
proc12 = b_ANOX*K_AO2/(K_AO2+xtemp[0])*xtemp[5]/(K_NOX+xtemp[5])*xtemp[11];

reac1 = x2*proc2+x4*proc4+x6*proc6+x8*proc8+x10*proc10+x11*proc11;
reac2 = f_SI*proc1;
reac3 = x1*proc1-proc2-proc3;
reac4 =
y1*proc1+y2*proc2+y3*proc3+y4*proc4+y5*proc5+y6*proc6+y7*proc7+y10*proc10+y11*proc11+y12*proc1
2;
reac5 = -x3*proc3-x5*proc5-x7*proc7-x9*proc9-x12*proc12;
reac6 = x3*proc3+x5*proc5+x7*proc7+x9*proc9+1.0/Y_A*proc10+x12*proc12;
reac7 =
z1*proc1+z2*proc2+z3*proc3+z4*proc4+z5*proc5+z6*proc6+z7*proc7+z9*proc9+z10*proc10+z11*proc11+
z12*proc12;
reac8 = f_XI*proc6+f_XI*proc7+f_XI*proc11+f_XI*proc12;
reac9 = -proc1;
reac10 = proc4+proc5-proc6-proc7;
reac11 = Y_STOO2*proc2+Y_STONOX*proc3-1.0/Y_HO2*proc4-1.0/Y_HNOX*proc5-proc8-proc9;
reac12 = proc10-proc11-proc12;
reac13 =
t1*proc1+t2*proc2+t3*proc3+t4*proc4+t5*proc5+t6*proc6+t7*proc7+t8*proc8+t9*proc9+t10*proc10+t1
1*proc11+t12*proc12;

reac15 = 0.0;
reac16 = 0.0;
reac17 = 0.0;
reac18 = 0.0;
reac19 = 0.0;


/* SO2 */ if (u[20] < 0)
          //if (KLa_temp < 0)
             dx[0] = 0;
         else
//              dx[0] = 1.0/vol*(u[13]*(u[0]-x[0]))+reac1+u[20]*(SO_sat-x[0]);
             dx[0] = 1.0/vol*(u[13]*(u[0]-x[0]))+reac1+KLa_temp*(SO_sat_temp-x[0]);

/* Si */     dx[1] = 1.0/vol*(u[13]*(u[1]-x[1]))+reac2;
/* Ss */     dx[2] = 1.0/vol*(u[13]*(u[2]-x[2]))+reac3;
/* Snh4 */   dx[3] = 1.0/vol*(u[13]*(u[3]-x[3]))+reac4;
/* Sn2 */    dx[4] = 1.0/vol*(u[13]*(u[4]-x[4]))+reac5;
/* Snox */   dx[5] = 1.0/vol*(u[13]*(u[5]-x[5]))+reac6;
/* Salk */   dx[6] = 1.0/vol*(u[13]*(u[6]-x[6]))+reac7;
/* Xi */     dx[7] = 1.0/vol*(u[13]*(u[7]-x[7]))+reac8;
/* Xs */     dx[8] = 1.0/vol*(u[13]*(u[8]-x[8]))+reac9;
/* Xbh */    dx[9] = 1.0/vol*(u[13]*(u[9]-x[9]))+reac10;
/* Xsto */   dx[10] = 1.0/vol*(u[13]*(u[10]-x[10]))+reac11;
/* Xba */    dx[11] = 1.0/vol*(u[13]*(u[11]-x[11]))+reac12;
/* TSS */    dx[12] = 1.0/vol*(u[13]*(u[12]-x[12]))+reac13;

/* Flow */   dx[13] = (u[13]-x[13])/T;   //low pass filter for flow, avoid algebraic loops */

/* Flow */   //dx[13] = 0.0;

/* Temp */   if (tempmodel < 0.5)
             dx[14] = 0.0;
             else
             dx[14] = 1.0/vol*(u[13]*(u[14]-x[14]));


/* dummy states, only dilution at this point */
dx[15] = 1.0/vol*(u[13]*(u[15]-x[15]))+reac15;
dx[16] = 1.0/vol*(u[13]*(u[16]-x[16]))+reac16;
dx[17] = 1.0/vol*(u[13]*(u[17]-x[17]))+reac17;
dx[18] = 1.0/vol*(u[13]*(u[18]-x[18]))+reac18;
dx[19] = 1.0/vol*(u[13]*(u[19]-x[19]))+reac19;
```

```
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
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
```

A5. Initialization file for ASM3 bioP

```
%% The following represent concentrations in different AS reactors (1 to 5)

Qin0     = 18446;
Qintr    = 3*Qin0;
Qw       = 385;
Qr       = Qin0;

% -------------------------------------------------------------------------
if (SETTLER < 0.5)
% ANAEROBIC
S_O_1    = 0.0038;
S_S_1    = 5.1;
S_I_1    = 30;
S_NH_1   = 22.7;
S_NO_1   = 0.27;
S_N2_1   = 20.6;
S_PO4_1  = 5.7;
S_HCO_1  = 5.8;
X_I_1    = 1525;
X_S_1    = 122;
X_H_1    = 1089;
X_STO_1  = 33.7;
X_PAO_1  = 684;
X_PP_1   = 100;
X_PHA_1  = 176;
X_A_1    = 133;
X_TSS_1  = 3715;

Q1       = Qin0 + Qr;
T1       = 15;

S_D1_1   = 0;
S_D2_1   = 0;
S_D3_1   = 0;
X_D4_1   = 0;
X_D5_1   = 0;


% -------------------------------------------------------------------------
% ANAEROBIC
S_O_2    = 0.00002;
S_S_2    = 3.4;
S_I_2    = 30;
S_NH_2   = 23.4;
S_NO_2   = 0.01;
S_N2_2   = 20.9;
S_PO4_2  = 12;
S_HCO_2  = 5.8;
X_I_2    = 1525;
X_S_2    = 103;
X_H_2    = 1089;
X_STO_2  = 34;
X_PAO_2  = 684;
X_PP_2   = 100;
X_PHA_2  = 195;
X_A_2    = 133;
X_TSS_2  = 3693;

Q2       = Qin0 + Qr;
T2       = 15;

S_D1_2   = 0;
S_D2_2   = 0;
S_D3_2   = 0;
X_D4_2   = 0;
X_D5_2   = 0;

% -------------------------------------------------------------------------
% ANOXIC
S_O_3    = 0.018;
S_S_3    = 0.56;
S_I_3    = 30;
S_NH_3   = 10.9;
S_NO_3   = 4;
S_N2_3   = 30;
```

```
S_PO4_3 = 0.36;
S_HCO_3 = 4.8;
X_I_3   = 1530;
X_S_3   = 62.5;
X_H_3   = 1096;
X_STO_3 = 25.1;
X_PAO_3 = 694;
X_PP_3  = 100;
X_PHA_3 = 170;
X_A_3   = 135;
X_TSS_3 = 3700;

Q3      = Qin0 + Qr + Qintr;
T3      = 15;

S_D1_3  = 0;
S_D2_3  = 0;
S_D3_3  = 0;
X_D4_3  = 0;
X_D5_3  = 0;

% ------------------------------------------------------------------------
% HYDDELAY
% values in gram/day
S_O_0   = 115710;
S_S_0   = 140500;
S_I_0   = 2766900;
S_NH_0  = 1022800;
S_NO_0  = 556620;
S_N2_0  = 2524000;
S_PO4_0 = 445780;
S_HCO_0 = 421120;
X_I_0   = 14113000;
X_S_0   = 6199500;
X_H_0   = 100830000;
X_STO_0 = 2508500;
X_PAO_0 = 63808000;
X_PP_0  = 18352000;
X_PHA_0 = 15988000;
X_A_0   = 12442000;
X_TSS_0 = 340240000;

Q0      = Q3;
T0      = 15;

S_D1_0  = 0;
S_D2_0  = 0;
S_D3_0  = 0;
X_D4_0  = 0;
X_D5_0  = 0;

% ------------------------------------------------------------------------
% ANOXIC
S_O_4   = 0.00028;
S_S_4   = 0.47;
S_I_4   = 30;
S_NH_4  = 10.9;
S_NO_4  = 2.4;
S_N2_4  = 31.1;
S_PO4_4 = 0.030;
S_HCO_4 = 4.89;
X_I_4   = 1531;
X_S_4   = 58;
X_H_4   = 1097;
X_STO_4 = 23.9;
X_PAO_4 = 695;
X_PP_4  = 100;
X_PHA_4 = 168;
X_A_4   = 135;
X_TSS_4 = 3699;

Q4      = Qin0 + Qr + Qintr;
T4      = 15;

S_D1_4  = 0;
S_D2_4  = 0;
S_D3_4  = 0;
```

```
X_D4_4   = 0;
X_D5_4   = 0;

% ------------------------------------------------------------------------
% AEROBIC
S_O_5    = 1.37;
S_S_5    = 0.32;
S_I_5    = 30;
S_NH_5   = 8.1;
S_NO_5   = 5.0;
S_N2_5   = 31;
S_PO4_5 = 0.010;
S_HCO_5 = 4.5;
X_I_5    = 1532;
X_S_5    = 52.7;
X_H_5    = 1097;
X_STO_5 = 23.6;
X_PAO_5 = 696;
X_PP_5   = 100;
X_PHA_5 = 165;
X_A_5    = 135;
X_TSS_5 = 3695;

Q5       = Qin0 + Qr + Qintr;
T5       = 15;

S_D1_5   = 0;
S_D2_5   = 0;
S_D3_5   = 0;
X_D4_5   = 0;
X_D5_5   = 0;

% ------------------------------------------------------------------------
% AEROBIC
S_O_6    = 1.73;
S_S_6    = 0.29;
S_I_6    = 30;
S_NH_6   = 5.4;
S_NO_6   = 7.6;
S_N2_6   = 31.5;
S_PO4_6 = 0.0093;
S_HCO_6 = 4.12;
X_I_6    = 1532;
X_S_6    = 47.8;
X_H_6    = 1096;
X_STO_6 = 23.1;
X_PAO_6 = 696;
X_PP_6   = 100;
X_PHA_6 = 162;
X_A_6    = 136;
X_TSS_6 = 3690;

Q6       = Qin0 + Qr + Qintr;
T6       = 15;

S_D1_6   = 0;
S_D2_6   = 0;
S_D3_6   = 0;
X_D4_6   = 0;
X_D5_6   = 0;

% ------------------------------------------------------------------------
% AEROBIC
S_O_7    = 2.1;
S_S_7    = 0.26;
S_I_7    = 30;
S_NH_7   = 2.89;
S_NO_7   = 10;
S_N2_7   = 31.7;
S_PO4_7 = 0.0089;
S_HCO_7 = 3.77;
X_I_7    = 1533;
X_S_7    = 43.4;
X_H_7    = 1096;
X_STO_7 = 22.5;
X_PAO_7 = 697;
X_PP_7   = 100;
```

```
X_PHA_7 = 159;
X_A_7   = 136;
X_TSS_7 = 3986;

Q7      = Qin0 + Qr + Qintr;
T7      = 15;

S_D1_7  = 0;
S_D2_7  = 0;
S_D3_7  = 0;
X_D4_7  = 0;
X_D5_7  = 0;

elseif (SETTLER > 0.5)

% ANAEROBIC
S_O_1   = 0.0038;
S_S_1   = 5.1;
S_I_1   = 30;
S_NH_1  = 22.7;
S_NO_1  = 0.27;
S_N2_1  = 20.6;
S_PO4_1 = 5.7;
S_HCO_1 = 5.8;
X_I_1   = 1525;
X_S_1   = 122;
X_H_1   = 1089;
X_STO_1 = 33.7;
X_PAO_1 = 684;
X_PP_1  = 100;
X_PHA_1 = 176;
X_A_1   = 133;
X_TSS_1 = 3715;

Q1      = Qin0 + Qr;
T1      = 15;

S_D1_1  = 0;
S_D2_1  = 0;
S_D3_1  = 0;
X_D4_1  = 0;
X_D5_1  = 0;


% -----------------------------------------------------------------------
% ANAEROBIC
S_O_2   = 0.00002;
S_S_2   = 3.4;
S_I_2   = 30;
S_NH_2  = 23.4;
S_NO_2  = 0.01;
S_N2_2  = 20.9;
S_PO4_2 = 12;
S_HCO_2 = 5.8;
X_I_2   = 1525;
X_S_2   = 103;
X_H_2   = 1089;
X_STO_2 = 34;
X_PAO_2 = 684;
X_PP_2  = 100;
X_PHA_2 = 195;
X_A_2   = 133;
X_TSS_2 = 3693;

Q2      = Qin0 + Qr;
T2      = 15;

S_D1_2  = 0;
S_D2_2  = 0;
S_D3_2  = 0;
X_D4_2  = 0;
X_D5_2  = 0;

% -----------------------------------------------------------------------
% ANOXIC
S_O_3   = 0.018;
S_S_3   = 0.56;
```

```
S_I_3   = 30;
S_NH_3  = 10.9;
S_NO_3  = 4;
S_N2_3  = 30;
S_PO4_3 = 0.36;
S_HCO_3 = 4.8;
X_I_3   = 1530;
X_S_3   = 62.5;
X_H_3   = 1096;
X_STO_3 = 25.1;
X_PAO_3 = 694;
X_PP_3  = 100;
X_PHA_3 = 170;
X_A_3   = 135;
X_TSS_3 = 3700;

Q3      = Qin0 + Qr + Qintr;
T3      = 15;

S_D1_3  = 0;
S_D2_3  = 0;
S_D3_3  = 0;
X_D4_3  = 0;
X_D5_3  = 0;

% -------------------------------------------------------------------------
% HYDDELAY
% values in gram/day
S_O_0   = 115710;
S_S_0   = 140500;
S_I_0   = 2766900;
S_NH_0  = 1022800;
S_NO_0  = 556620;
S_N2_0  = 2524000;
S_PO4_0 = 445780;
S_HCO_0 = 421120;
X_I_0   = 14113000;
X_S_0   = 6199500;
X_H_0   = 100830000;
X_STO_0 = 2508500;
X_PAO_0 = 63808000;
X_PP_0  = 18352000;
X_PHA_0 = 15988000;
X_A_0   = 12442000;
X_TSS_0 = 340240000;

Q0      = Q3;
T0      = 15;

S_D1_0  = 0;
S_D2_0  = 0;
S_D3_0  = 0;
X_D4_0  = 0;
X_D5_0  = 0;

% -------------------------------------------------------------------------
% ANOXIC
S_O_4   = 0.00028;
S_S_4   = 0.47;
S_I_4   = 30;
S_NH_4  = 10.9;
S_NO_4  = 2.4;
S_N2_4  = 31.1;
S_PO4_4 = 0.030;
S_HCO_4 = 4.89;
X_I_4   = 1531;
X_S_4   = 58;
X_H_4   = 1097;
X_STO_4 = 23.9;
X_PAO_4 = 695;
X_PP_4  = 100;
X_PHA_4 = 168;
X_A_4   = 135;
X_TSS_4 = 3699;

Q4      = Qin0 + Qr + Qintr;
T4      = 15;
```

```
S_D1_4  = 0;
S_D2_4  = 0;
S_D3_4  = 0;
X_D4_4  = 0;
X_D5_4  = 0;

% ------------------------------------------------------------------------
% AEROBIC
S_O_5   = 1.37;
S_S_5   = 0.32;
S_I_5   = 30;
S_NH_5  = 8.1;
S_NO_5  = 5.0;
S_N2_5  = 31;
S_PO4_5 = 0.010;
S_HCO_5 = 4.5;
X_I_5   = 1532;
X_S_5   = 52.7;
X_H_5   = 1097;
X_STO_5 = 23.6;
X_PAO_5 = 696;
X_PP_5  = 100;
X_PHA_5 = 165;
X_A_5   = 135;
X_TSS_5 = 3695;

Q5      = Qin0 + Qr + Qintr;
T5      = 15;

S_D1_5  = 0;
S_D2_5  = 0;
S_D3_5  = 0;
X_D4_5  = 0;
X_D5_5  = 0;

% ------------------------------------------------------------------------
% AEROBIC
S_O_6   = 1.73;
S_S_6   = 0.29;
S_I_6   = 30;
S_NH_6  = 5.4;
S_NO_6  = 7.6;
S_N2_6  = 31.5;
S_PO4_6 = 0.0093;
S_HCO_6 = 4.12;
X_I_6   = 1532;
X_S_6   = 47.8;
X_H_6   = 1096;
X_STO_6 = 23.1;
X_PAO_6 = 696;
X_PP_6  = 100;
X_PHA_6 = 162;
X_A_6   = 136;
X_TSS_6 = 3690;

Q6      = Qin0 + Qr + Qintr;
T6      = 15;

S_D1_6  = 0;
S_D2_6  = 0;
S_D3_6  = 0;
X_D4_6  = 0;
X_D5_6  = 0;

% ------------------------------------------------------------------------
% AEROBIC
S_O_7   = 2.1;
S_S_7   = 0.26;
S_I_7   = 30;
S_NH_7  = 2.89;
S_NO_7  = 10;
S_N2_7  = 31.7;
S_PO4_7 = 0.0089;
S_HCO_7 = 3.77;
X_I_7   = 1533;
X_S_7   = 43.4;
```

```
X_H_7   = 1096;
X_STO_7 = 22.5;
X_PAO_7 = 697;
X_PP_7  = 100;
X_PHA_7 = 159;
X_A_7   = 136;
X_TSS_7 = 3986;

Q7      = Qin0 + Qr + Qintr;
T7      = 15;

S_D1_7  = 0;
S_D2_7  = 0;
S_D3_7  = 0;
X_D4_7  = 0;
X_D5_7  = 0;

end;
% ------------------------------------------------------------------------

XINITDELAY  = [ S_O_0  S_S_0  S_I_0  S_NH_0  S_NO_0  S_N2_0  S_PO4_0  S_HCO_0  X_I_0  X_S_0
X_H_0  X_STO_0  X_PAO_0  X_PP_0  X_PHA_0  X_A_0  X_TSS_0  Q0  T0  S_D1_0  S_D2_0  S_D3_0
X_D4_0  X_D5_0 ];
XINIT1      = [ S_O_1  S_S_1  S_I_1  S_NH_1  S_NO_1  S_N2_1  S_PO4_1  S_HCO_1  X_I_1  X_S_1
X_H_1  X_STO_1  X_PAO_1  X_PP_1  X_PHA_1  X_A_1  X_TSS_1  Q1  T1  S_D1_1  S_D2_1  S_D3_1
X_D4_1  X_D5_1 ];
XINIT2      = [ S_O_2  S_S_2  S_I_2  S_NH_2  S_NO_2  S_N2_2  S_PO4_2  S_HCO_2  X_I_2  X_S_2
X_H_2  X_STO_2  X_PAO_2  X_PP_2  X_PHA_2  X_A_2  X_TSS_2  Q2  T2  S_D1_2  S_D2_2  S_D3_2
X_D4_2  X_D5_2 ];
XINIT3      = [ S_O_3  S_S_3  S_I_3  S_NH_3  S_NO_3  S_N2_3  S_PO4_3  S_HCO_3  X_I_3  X_S_3
X_H_3  X_STO_3  X_PAO_3  X_PP_3  X_PHA_3  X_A_3  X_TSS_3  Q3  T3  S_D1_3  S_D2_3  S_D3_3
X_D4_3  X_D5_3 ];
XINIT4      = [ S_O_4  S_S_4  S_I_4  S_NH_4  S_NO_4  S_N2_4  S_PO4_4  S_HCO_4  X_I_4  X_S_4
X_H_4  X_STO_4  X_PAO_4  X_PP_4  X_PHA_4  X_A_4  X_TSS_4  Q4  T4  S_D1_4  S_D2_4  S_D3_4
X_D4_4  X_D5_4 ];
XINIT5      = [ S_O_5  S_S_5  S_I_5  S_NH_5  S_NO_5  S_N2_5  S_PO4_5  S_HCO_5  X_I_5  X_S_5
X_H_5  X_STO_5  X_PAO_5  X_PP_5  X_PHA_5  X_A_5  X_TSS_5  Q5  T5  S_D1_5  S_D2_5  S_D3_5
X_D4_5  X_D5_5 ];
XINIT6      = [ S_O_6  S_S_6  S_I_6  S_NH_6  S_NO_6  S_N2_6  S_PO4_6  S_HCO_6  X_I_6  X_S_6
X_H_6  X_STO_6  X_PAO_6  X_PP_6  X_PHA_6  X_A_6  X_TSS_6  Q6  T6  S_D1_6  S_D2_6  S_D3_6
X_D4_6  X_D5_6 ];
XINIT7      = [ S_O_7  S_S_7  S_I_7  S_NH_7  S_NO_7  S_N2_7  S_PO4_7  S_HCO_7  X_I_7  X_S_7
X_H_7  X_STO_7  X_PAO_7  X_PP_7  X_PHA_7  X_A_7  X_TSS_7  Q7  T7  S_D1_7  S_D2_7  S_D3_7
X_D4_7  X_D5_7 ];

% used to avoid stiff solver from getting stuck when running steady state for long time
periods and starting to close to correct value
% multiply by random number between 0.95 and 1.05
XINIT1      = XINIT1.*      (((rand(1, 24) - 0.5)/10) + 1);
XINIT2      = XINIT2.*      (((rand(1, 24) - 0.5)/10) + 1);
XINIT3      = XINIT3.*      (((rand(1, 24) - 0.5)/10) + 1);
XINIT4      = XINIT4.*      (((rand(1, 24) - 0.5)/10) + 1);
XINIT5      = XINIT5.*      (((rand(1, 24) - 0.5)/10) + 1);
XINIT6      = XINIT5.*      (((rand(1, 24) - 0.5)/10) + 1);
XINIT7      = XINIT5.*      (((rand(1, 24) - 0.5)/10) + 1);


%% ASM3_Bio-P model parameters (Reference: Hauduc et al.,2010; 20 deg.C)

% Stoichiometric parameters
f_SI            = 0;          % Fraction of inert COD generated in hydrolysis; Fraction of
S_I generated in X_PAO decay

Y_H_O2          = 0.8;        % Yield for X_H growth per X_STO (Aerobic)
Y_H_NO          = 0.65;       % Yield for X_H growth per X_STO (Anoxic)
Y_STO_O2        = 0.8;        % Yield for X_STO formation per S_S (Aerobic)
Y_STO_NO        = 0.7;        % Yield for X_STO formation per S_S (Anoxic)

f_XI            = 0.2;        % Fraction of X_I generated in heterotrophic biomass decay

Y_PAO_O2        = 0.6;        % Yield for X_PAO growth per X_PHA (Aerobic)
Y_PAO_NO        = 0.5;        % Yield for X_PAO growth per X_PHA (Anoxic)
Y_PHA           = 0.2;        % Yield for X_PP storage (S_PO4 uptake) per X_PHA utilized
Y_PO4           = 0.35;       % Yield for X_PP requirement (S_PO4 release) per X_PHA stored

Y_AUT           = 0.24;       % Yield of X_A growth per S_NO
```

```
iN_SS           = 0.03;         % N content of S_S
iN_SI           = 0.01;         % N content of S_I
iN_XI           = 0.03;         % N content of X_I
iN_XS           = 0.035;        % N content of X_S
iN_BM           = 0.07;         % N content of biomass (X_H, X_PAO, X_A)

iP_SS           = 0;            % P content of S_S
iP_SI           = 0;            % P content of S_I
iP_XI           = 0.01;         % P content of X_I
iP_XS           = 0.005;        % P content of X_S
iP_BM           = 0.014;        % P content of biomass (X_H, X_PAO, X_A)

iTSS_XI         = 0.75;         % TSS to COD ratio for X_I
iTSS_XS         = 0.75;         % TSS to COD ratio for X_S
iTSS_XSTO       = 0.6;          % TSS to COD ratio for X_STO
iTSS_BM         = 0.9;          % TSS to COD ratio for biomass
iTSS_XPP        = 3.23;         % TSS to COD ratio for X_PP

iNOX_N2         = 40/14;        % Conversion factor for NO3 reduction to N2
iCOD_NOX        = -64/14;       % Conversion factor for NO3 in COD
iCOD_N2         = -24/14;       % Conversion factor for N2 in COD
icharge_NHX     = 1/14;         % Conversion factor for NH in charge
icharge_NOX     = -1/14;        % Conversion factor for NO3 in charge
icharge_XPAO_PP = -1/31;        % Conversion factor for X_PP (K0.33Mg0.33PO3)n in charge
icharge_PO4     = -1.5/31;      % Conversion factor for PO4 in charge

% Kinetic parameters
k_H             = 9;            % Maximum specific hydrolysis rate
K_X             = 1;            % Half saturation parameter for X_S/X_H

k_STO           = 12.5;         % Rate constant for X_STO storage
mu_H            = 3;            % Maximum growth rate of X_H
h_NO_H          = 0.8;          % Reduction factor for anoxic growth of X_H (0.6 in ASM3)
K_SS_H          = 10;           % Half saturation parameter for S_S
K_STO_H         = 0.1;          % Half saturation parameter X_STO/X_H
b_H             = 0.3;          % Endogenous respiration rate of X_H
h_NO_end_H      = 0.33;         % Reduction factor for Anoxic endogenous respiration of
heterotrophs
b_STO           = 0.3;          % Endogenous respiration rate of X_STO
K_O_H           = 0.2;          % Half saturation parameter for S_O
K_NO_H          = 0.5;          % Half saturation parameter for S_NO
K_NH_H          = 0.01;         % Half saturation parameter for S_NH
K_PO4_H         = 0.011;         % Half saturation parameter for S_PO4
K_HCO_H         = 0.1;          % Half saturation parameter for S_HCO

q_PHA           = 6;            % Rate constant for S_Ac uptake rate (X_PHA storage)
q_PP            = 1.5;          % Rate constant for storage of X_PP
K_PP_PAO        = 0.05;         % Maximum ratio of X_PP/X_PAO
K_max_PAO       = 0.2;          % Half saturation parameter for X_PP/X_PAO
K_iPP_PAO       = 0.05;         % Half Inhibition parameter for X_PP/X_PAO
mu_PAO          = 1;            % Maximum growth rate of X_PAO
h_NO_PAO        = 0.6;          % Reduction factor for anoxic growth of X_PAO
K_PHA           = 0.1;          % Saturation constant for X_PHA/X_PAO
b_PAO           = 0.2;          % Endogenous respiration rate of X_PAO
h_NO_end_PAO    = 0.33;         % Reduction factor for anoxic endogenous respiration of X_PAO
b_PP            = 0.2;          % Rate constant for lysis of X_PP
h_NO_lys_PP     = 0.33;         % Reduction factor for anoxic lysis of X_PP
b_PHA           = 0.2;          % Rate constant for respiration of X_PHA
h_NO_resp_PHA   = 0.33;         % Reduction factor for anoxic respiration of X_PHA
K_SS_PAO        = 10;           % Half saturation parameter for S_S
K_O_PAO         = 0.2;          % Half saturation parameter for S_O
K_NO_PAO        = 0.5;          % Half saturation parameter for S_NO
K_NH_PAO        = 0.05;         % Half saturation parameter for S_NH
K_PO4_PP        = 0.2;          % Half saturation parameter for S_PO4 uptake (X_PP storage)
K_PO4_PAO       = 0.01;         % Half saturation parameter for S_PO4 as nutrient (X_PAO
growth)
K_HCO_PAO       = 0.1;          % Half saturation parameter for S_HCO

mu_A            = 1;            % Maximum growth rate of X_A (0.9-1.8)
b_A             = 0.2;          % Decay rate for X_A; Endogenous respiration rate for X_A
h_NO_A          = 0.5;          % Reduction factor for anoxic endogenous respiration of X_A
K_O_A           = 0.5;          % Half saturation parameter for S_O
K_NH_A          = 1;            % Half saturation parameter for S_NH (1-2)
K_PO4_A         = 0.01;         % Half saturation parameter for S_PO4
K_HCO_A         = 0.5;          % Half saturation parameter for S_HCO
```

```
% Identical parameter set in all reactors
PAR1     = [ f_SI Y_H_O2 Y_H_NO Y_STO_O2 Y_STO_NO f_XI Y_PAO_O2 Y_PAO_NO Y_PHA Y_PO4 Y_AUT ...
            iN_SS iN_SI iN_XI iN_XS iN_BM iP_SS iP_SI iP_XI iP_XS iP_BM iTSS_XI iTSS_XS
iTSS_XSTO iTSS_BM iTSS_XPP iNOX_N2 iCOD_NOX iCOD_N2 icharge_NHX icharge_NOX icharge_XPAO_PP
icharge_PO4 ...
            k_H K_X k_STO mu_H h_NO_H K_SS_H K_STO_H b_H h_NO_end_H b_STO K_O_H K_NO_H K_NH_H
K_PO4_H K_HCO_H ...
            q_PHA q_PP K_PP_PAO K_max_PAO K_iPP_PAO mu_PAO h_NO_PAO K_PHA b_PAO h_NO_end_PAO
b_PP h_NO_lys_PP b_PHA h_NO_resp_PHA K_SS_PAO K_O_PAO K_NO_PAO K_NH_PAO K_PO4_PP K_PO4_PAO
K_HCO_PAO ...
            mu_A b_A h_NO_A K_O_A K_NH_A K_PO4_A K_HCO_A ];
PAR2     = PAR1;
PAR3     = PAR1;
PAR4     = PAR1;
PAR5     = PAR1;
PAR6     = PAR1;
PAR7     = PAR1;

% Reactor volumes
VOL1     = 1000;
VOL2     = VOL1;
VOL3     = 1000;
VOL4     = VOL3;
VOL5     = 1333;
VOL6     = VOL5;
VOL7     = VOL5;

% Oxygen saturation concentration at 15 deg.C (based on BSM1)
SOSAT1   = 8;
SOSAT2   = SOSAT1;
SOSAT3   = SOSAT1;
SOSAT4   = SOSAT1;
SOSAT5   = SOSAT1;
SOSAT6   = SOSAT1;
SOSAT7   = SOSAT1;

% KLa values
KLa1     = 0;
KLa2     = 0;
KLa3     = 0;
KLa4     = 0;
KLa5     = 240;
KLa6     = 240;
KLa7     = 240;

% External carbon flow rate
carb1    = 0;
carb2    = 0;
carb3    = 0;
carb4    = 0;
carb5    = 0;
carb6    = 0;
carb7    = 0;

CARBONSOURCECONC = 400000;      % external carbon source concentration = 400000 mg COD/l

T = 0.0001;
```

A6. Initialization file for ASM3

```
Qin0 = 18446;
Qintr = 3*Qin0;
Qin = Qin0;
Qw = 385;

% _ASin represents mass (g/d), flow is still m3/d, used by hydraulic delay

S_O2_ASin = 0.0001*(Qin+Qin0+Qintr);
S_I_ASin = 30*(Qin+Qin0+Qintr);
S_S_ASin = 69.50*(Qin+Qin0+Qintr);
S_NH4_ASin = 34.716*(Qin+Qin0+Qintr);
S_N2_ASin =0.0001* (Qin+Qin0+Qintr);
S_NOX_ASin = 0.0001*(Qin+Qin0+Qintr);
S_ALK_ASin = 7*(Qin+Qin0+Qintr);
X_I_ASin = 51.20*(Qin+Qin0+Qintr);
X_S_ASin = 202.32*(Qin+Qin0+Qintr);
X_H_ASin = 28.17*(Qin+Qin0+Qintr);
X_STO_ASin = 0.0001*(Qin+Qin0+Qintr);
X_A_ASin = 0.0001*(Qin+Qin0+Qintr);
X_SS_ASin = (0.75*X_S_ASin + 0.75*X_S_ASin + 0.90*X_H_ASin + 0.60*X_STO_ASin + 0.90*X_A_ASin)
;
Q_ASin = Qin+Qin0+Qintr;
T_ASin = 14.8581;
S_D1_ASin = 0;
S_D2_ASin = 0;
S_D3_ASin = 0;
X_D4_ASin = 0;
X_D5_ASin = 0;

S_O2_1 = 2;
S_I_1 = 30;
S_S_1 = 2;
S_NH4_1 = 20;
S_N2_1 = 0;
S_NOX_1 = 0;
S_ALK_1 = 5;
X_I_1 = 100;
X_S_1 = 40;
X_H_1 = 100;
X_STO_1 = 40;
X_A_1 = 1;
X_SS_1 = 200;
Q_1 = Qin+Qin0+Qintr;
T1 = 14.8581;
S_D1_1 = 0;
S_D2_1 = 0;
S_D3_1 = 0;
X_D4_1 = 0;
X_D5_1 = 0;

S_O2_2 = 2;
S_I_2 = 30;
S_S_2 = 2;
S_NH4_2 = 20;
S_N2_2 = 0;
S_NOX_2 = 0;
S_ALK_2 = 5;
X_I_2 = 100;
X_S_2 = 40;
X_H_2 = 100;
X_STO_2 = 40;
X_A_2 = 1;
X_SS_2 = 200;
Q_2 = Qin+Qin0+Qintr;
T2 = 14.8581;
S_D1_2 = 0;
S_D2_2 = 0;
S_D3_2 = 0;
X_D4_2 = 0;
X_D5_2 = 0;

S_O2_3 = 2;
S_I_3 = 30;
S_S_3 = 2;
S_NH4_3 = 20;
```

```
S_N2_3 = 0;
S_NOX_3 = 0;
S_ALK_3 = 5;
X_I_3 = 100;
X_S_3 = 40;
X_H_3 = 100;
X_STO_3 = 40;
X_A_3 = 1;
X_SS_3 = 200;
Q_3 = Qin+Qin0+Qintr;
T3 = 14.8581;
S_D1_3 = 0;
S_D2_3 = 0;
S_D3_3 = 0;
X_D4_3 = 0;
X_D5_3 = 0;

S_O2_4 = 2;
S_I_4 = 30;
S_S_4 = 2;
S_NH4_4 = 20;
S_N2_4 = 0;
S_NOX_4 = 0;
S_ALK_4 = 5;
X_I_4=  100;
X_S_4 = 40;
X_H_4 = 100;
X_STO_4 = 40;
X_A_4 = 1;
X_SS_4 = 200;
Q_4 = Qin+Qin0+Qintr;
T4 = 14.8581;
S_D1_4 = 0;
S_D2_4 = 0;
S_D3_4 = 0;
X_D4_4 = 0;
X_D5_4 = 0;

S_O2_5 = 2;
S_I_5 = 30;
S_S_5 = 2;
S_NH4_5 = 20;
S_N2_5 = 0;
S_NOX_5 = 0;
S_ALK_5 = 5;
X_I_5 = 100;
X_S_5 = 40;
X_H_5 = 100;
X_STO_5 = 40;
X_A_5 = 1;
X_SS_5 = 200;
Q_5 = Qin+Qin0+Qintr;
T5 = 14.8581;
S_D1_5 = 0;
S_D2_5 = 0;
S_D3_5 = 0;
X_D4_5 = 0;
X_D5_5 = 0;

XINITDELAY = [ S_O2_ASin S_I_ASin S_S_ASin S_NH4_ASin S_N2_ASin S_NOX_ASin S_ALK_ASin X_I_ASin
X_S_ASin X_H_ASin X_STO_ASin X_A_ASin X_SS_ASin Q_ASin T_ASin S_D1_ASin S_D2_ASin S_D3_ASin
X_D4_ASin X_D5_ASin ];
XINIT1 = [ S_O2_1 S_I_1 S_S_1 S_NH4_1 S_N2_1 S_NOX_1 S_ALK_1 X_I_1 X_S_1 X_H_1 X_STO_1 X_A_1
X_SS_1 Q_1 T1 S_D1_1 S_D2_1 S_D3_1 X_D4_1 X_D5_1];
XINIT2 = [ S_O2_2 S_I_2 S_S_2 S_NH4_2 S_N2_2 S_NOX_2 S_ALK_2 X_I_2 X_S_2 X_H_2 X_STO_2 X_A_2
X_SS_2 Q_2 T2 S_D1_2 S_D2_2 S_D3_2 X_D4_2 X_D5_2];
XINIT3 = [ S_O2_3 S_I_3 S_S_3 S_NH4_3 S_N2_3 S_NOX_3 S_ALK_3 X_I_3 X_S_3 X_H_3 X_STO_3 X_A_3
X_SS_3 Q_3 T3 S_D1_3 S_D2_3 S_D3_3 X_D4_3 X_D5_3];
XINIT4 = [ S_O2_4 S_I_4 S_S_4 S_NH4_4 S_N2_4 S_NOX_4 S_ALK_4 X_I_4 X_S_4 X_H_4 X_STO_4 X_A_4
X_SS_4 Q_4 T4 S_D1_4 S_D2_4 S_D3_4 X_D4_4 X_D5_4];
XINIT5 = [ S_O2_2 S_I_5 S_S_5 S_NH4_5 S_N2_5 S_NOX_5 S_ALK_5 X_I_5 X_S_5 X_H_5 X_STO_5 X_A_5
X_SS_5 Q_5 T5 S_D1_5 S_D2_5 S_D3_5 X_D4_5 X_D5_5];

% XINIT1 = XINIT1.*(rand(1, 20)/2);
% XINIT2 = XINIT2.*(rand(1, 20)/2);
% XINIT3 = XINIT3.*(rand(1, 20)/2);
% XINIT4 = XINIT4.*(rand(1, 20)/2);
% XINIT5 = XINIT5.*(rand(1, 20)/2);

%XINIT5(1) = 2; % just to avoid antiwindupeffect the first sample time

k_H = 3;
K_X = 1;
k_STO = 5;
ny_NOX = 0.6;
K_O2 = 0.2;
K_NOX = 0.5;
K_S = 2;
K_STO = 1;
mu_H = 2;
K_NH4 = 0.01;
K_ALK = 0.1;
b_HO2 = 0.2;
b_HNOX = 0.1;
b_STOO2 = 0.2;
b_STONOX = 0.1;
mu_A = 1.0;
K_ANH4 = 1;
K_AO2 = 0.5;
K_AALK = 0.5;
b_AO2 = 0.15;
b_ANOX = 0.05;
f_SI = 0;
Y_STOO2 = 0.85;
Y_STONOX = 0.80;
Y_HO2 = 0.63;
Y_HNOX = 0.54;
Y_A = 0.24;
f_XI = 0.20;
i_NSI = 0.01;
i_NSS = 0.03;
i_NXI = 0.02;
i_NXS = 0.04;
i_NBM = 0.07;
i_SSXI = 0.75;
i_SSXS = 0.75;
i_SSBM = 0.90;
i_SSSTO = 0.60;  %not properly defined in ASM3 report
f_P = 0.08;

PAR1 = [ k_H K_X k_STO ny_NOX K_O2 K_NOX K_S K_STO mu_H K_NH4 K_ALK b_HO2 b_HNOX b_STOO2
b_STONOX mu_A K_ANH4 K_AO2 K_AALK b_AO2 b_ANOX f_SI Y_STOO2 Y_STONOX Y_HO2 Y_HNOX Y_A f_XI
i_NSI i_NSS i_NXI i_NXS i_NBM i_SSXI i_SSXS i_SSBM i_SSSTO ];
PAR2 = PAR1;
PAR3 = PAR1;
PAR4 = PAR1;
PAR5 = PAR1;


VOL1 = 1000;
VOL2 = VOL1;
VOL3 = 1333;
VOL4 = VOL3;
VOL5 = VOL3;


SOSAT1 = 8;
SOSAT2 = SOSAT1;
SOSAT3 = SOSAT1;
SOSAT4 = SOSAT1;
SOSAT5 = SOSAT1;


KLa1 = 0;
KLa2 = 0;
KLa3 = 240;
KLa4 = 240;
KLa5 = 240;


carb1 = 0; % external carbon flow rate to reactor 1
carb2 = 0; % external carbon flow rate to reactor 2
carb3 = 0; % external carbon flow rate to reactor 3
carb4 = 0; % external carbon flow rate to reactor 4
```

```
carb5 = 0; % external carbon flow rate to reactor 5
CARBONSOURCECONC = 400000; % external carbon source concentration = 400000 mg COD/l


T = 0.0001; % used by hydraulic delays
QintrT = T*10;
```

A7. Matlab code for Settler

```
#define S_FUNCTION_NAME settler1d_asm3_bioP

#include "simstruc.h"
#include <math.h>

#define XINIT     ssGetArg(S,0)         // initial values
#define SEDPAR    ssGetArg(S,1)         // parameters sedimentation model
#define DIM       ssGetArg(S,2)         //
#define LAYER     ssGetArg(S,3)         //
#define ASMPAR    ssGetArg(S,4)         // parameters activated sludge model
#define SETTLER   ssGetArg(S,5)         // reactive settler
#define ACTIVATE  ssGetArg(S,6)         // dummy states
#define TEMPMODEL ssGetArg(S,7)         // temperature propagation

// mdlInitializeSizes - initialize the sizes array
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates      (S, 230);   // number of continuous states, 17 ASM3_bioP
components for each settler layer + T + 5 dummy states
    ssSetNumDiscStates      (S, 0);     // number of discrete states
    ssSetNumInputs          (S, 26);    // number of inputs, 17 components + Q + T +  5 dummy
states + Qr + Qw
    ssSetNumOutputs         (S, 291);   // number of outputs
    ssSetDirectFeedThrough  (S, 1);     // direct feedthrough flag
    ssSetNumSampleTimes     (S, 1);     // number of sample times
    ssSetNumSFcnParams      (S, 8);     // number of input arguments
    ssSetNumRWork           (S, 0);     // number of real work vector elements
    ssSetNumIWork           (S, 0);     // number of integer work vector elements
    ssSetNumPWork           (S, 0);     // number of pointer work vector elements
}

// mdlInitializeSampleTimes - initialize the sample times array
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}


// mdlInitializeConditions - initialize the states
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
int i;

for (i = 0; i < 230; i++) {
   x0[i] = mxGetPr(XINIT)[i];
}
}

// mdlOutputs - compute the outputs

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S, int tid)
{
  double gamma, gamma_eff;
  double tempmodel;
  int i;

  gamma     = x[169]/u[16]; // factor describing underflow vs. inflow TSS concentration ratio
  gamma_eff = x[160]/u[16]; // factor describing effluent vs. inflow TSS concentration ratio

  tempmodel = mxGetPr(TEMPMODEL)[0];

    // layers as separate mixed tanks for soluble components

    // UNDERFLOW: concentrations in underflow equal concentrations in bottom layer settler
    y[0]     = x[9];          // S_O
    y[1]     = x[19];         // S_S
    y[2]     = x[29];         // S_I
    y[3]     = x[39];         // S_NH
    y[4]     = x[49];         // S_NO
    y[5]     = x[59];         // S_N2
    y[6]     = x[69];         // S_PO4
    y[7]     = x[79];         // S_HCO
    y[8]     = x[89];         // X_I
    y[9]     = x[99];         // X_S
```

```
y[10]   = x[109];       // X_H
y[11]   = x[119];       // X_STO
y[12]   = x[129];       // X_PAO
y[13]   = x[139];       // X_PP
y[14]   = x[149];       // X_PHA
y[15]   = x[159];       // X_A
y[16]   = x[169];       // X_TSS

y[17]   = u[24];        // Q_r

if (tempmodel < 0.5)    // Temp
    y[18]   = u[18];
else
    y[18]   = x[179];

y[19]   = x[189];       // Dummy 1
y[20]   = x[199];       // Dummy 2
y[21]   = x[209];       // Dummy 3
y[22]   = x[219];       // Dummy 4
y[23]   = x[229];       // Dummy 5

y[24]   = u[25];        // Q_w


// EFFLUENT: concentrations in effluent equal concentrations in top layer settler
y[25]   = x[0];        // S_O
y[26]   = x[10];       // S_S
y[27]   = x[20];       // S_I
y[28]   = x[30];       // S_NH
y[29]   = x[40];       // S_NO
y[30]   = x[50];       // S_N2
y[31]   = x[60];       // S_PO4
y[32]   = x[70];       // S_HCO
y[33]   = x[80];       // X_I
y[34]   = x[90];       // X_S
y[35]   = x[100];      // X_H
y[36]   = x[110];      // X_STO
y[37]   = x[120];      // X_PAO
y[38]   = x[130];      // X_PP
y[39]   = x[140];      // X_PHA
y[40]   = x[150];      // X_A
y[41]   = x[160];      // X_TSS

y[42]   = u[17] - u[24] - u[25];  // effluent Q of settler = Qin_settler - Q_r - Q_w

if (tempmodel < 0.5)    // Temp
    y[43]   = u[18];
else
    y[43]   = x[170];

y[44]   = x[180];       // Dummy 1
y[45]   = x[190];       // Dummy 2
y[46]   = x[200];       // Dummy 3
y[47]   = x[210];       // Dummy 4
y[48]   = x[220];       // Dummy 5


// Internal TSS states
y[49]   = x[160];       // top layer
y[50]   = x[161];
y[51]   = x[162];
y[52]   = x[163];
y[53]   = x[164];
y[54]   = x[165];
y[55]   = x[166];
y[56]   = x[167];
y[57]   = x[168];
y[58]   = x[169];       // bottom layer

y[59]   = gamma;
y[60]   = gamma_eff;


//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx//
//        states for each layer of the secondary clarifier       //

// LAYER 1
```

```
y[61]   = x[0];          // S_O
y[62]   = x[10];         // S_S
y[63]   = x[20];         // S_I
y[64]   = x[30];         // S_NH
y[65]   = x[40];         // S_NO
y[66]   = x[50];         // S_N2
y[67]   = x[60];         // S_PO4
y[68]   = x[70];         // S_HCO
y[69]   = x[80];         // X_I
y[70]   = x[90];         // X_S
y[71]   = x[100];        // X_H
y[72]   = x[110];        // X_STO
y[73]   = x[120];        // X_PAO
y[74]   = x[130];        // X_PP
y[75]   = x[140];        // X_PHA
y[76]   = x[150];        // X_A
y[77]   = x[160];        // X_TSS

if (tempmodel < 0.5)     // Temp
    y[78]   = u[18];
else
    y[78]   = x[170];

y[79]   = x[180];        // Dummy 1
y[80]   = x[190];        // Dummy 2
y[81]   = x[200];        // Dummy 3
y[82]   = x[210];        // Dummy 4
y[83]   = x[220];        // Dummy 5

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx//

// LAYER 2
y[84]   = x[1];          // S_O
y[85]   = x[11];         // S_S
y[86]   = x[21];         // S_I
y[87]   = x[31];         // S_NH
y[88]   = x[41];         // S_NO
y[89]   = x[51];         // S_N2
y[90]   = x[61];         // S_PO4
y[91]   = x[71];         // S_HCO
y[92]   = x[81];         // X_I
y[93]   = x[91];         // X_S
y[94]   = x[101];        // X_H
y[95]   = x[111];        // X_STO
y[96]   = x[121];        // X_PAO
y[97]   = x[131];        // X_PP
y[98]   = x[141];        // X_PHA
y[99]   = x[151];        // X_A
y[100]  = x[161];        // X_TSS

if (tempmodel < 0.5)     // Temp
    y[101]  = u[18];
else
    y[101]  = x[171];

y[102]  = x[181];        // Dummy 1
y[103]  = x[191];        // Dummy 2
y[104]  = x[201];        // Dummy 3
y[105]  = x[211];        // Dummy 4
y[106]  = x[221];        // Dummy 5

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx//

// LAYER 3
y[107]  = x[2];          // S_O
y[108]  = x[12];         // S_S
y[109]  = x[22];         // S_I
y[110]  = x[32];         // S_NH
y[111]  = x[42];         // S_NO
y[112]  = x[52];         // S_N2
y[113]  = x[62];         // S_PO4
y[114]  = x[72];         // S_HCO
y[115]  = x[82];         // X_I
y[116]  = x[92];         // X_S
y[117]  = x[102];        // X_H
y[118]  = x[112];        // X_STO
y[119]  = x[122];        // X_PAO
```

```
y[120]  = x[132];       // X_PP
y[121]  = x[142];       // X_PHA
y[122]  = x[152];       // X_A
y[123]  = x[162];       // X_TSS

if (tempmodel < 0.5)    // Temp
    y[124]  = u[18];
else
    y[124]  = x[172];

y[125]  = x[182];       // Dummy 1
y[126]  = x[192];       // Dummy 2
y[127]  = x[202];       // Dummy 3
y[128]  = x[212];       // Dummy 4
y[129]  = x[222];       // Dummy 5

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx//

// LAYER 4
y[130]  = x[3];         // S_O
y[131]  = x[13];        // S_S
y[132]  = x[23];        // S_I
y[133]  = x[33];        // S_NH
y[134]  = x[43];        // S_NO
y[135]  = x[53];        // S_N2
y[136]  = x[63];        // S_PO4
y[137]  = x[73];        // S_HCO
y[138]  = x[83];        // X_I
y[139]  = x[93];        // X_S
y[140]  = x[103];       // X_H
y[141]  = x[113];       // X_STO
y[142]  = x[123];       // X_PAO
y[143]  = x[133];       // X_PP
y[144]  = x[143];       // X_PHA
y[145]  = x[153];       // X_A
y[146]  = x[163];       // X_TSS

if (tempmodel < 0.5)    // Temp
    y[147]  = u[18];
else
    y[147]  = x[173];

y[148]  = x[183];       // Dummy 1
y[149]  = x[193];       // Dummy 2
y[150]  = x[203];       // Dummy 3
y[151]  = x[213];       // Dummy 4
y[152]  = x[223];       // Dummy 5

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx//

// LAYER 5
y[153]  = x[4];         // S_O
y[154]  = x[14];        // S_S
y[155]  = x[24];        // S_I
y[156]  = x[34];        // S_NH
y[157]  = x[44];        // S_NO
y[158]  = x[54];        // S_N2
y[159]  = x[64];        // S_PO4
y[160]  = x[74];        // S_HCO
y[161]  = x[84];        // X_I
y[162]  = x[94];        // X_S
y[163]  = x[104];       // X_H
y[164]  = x[114];       // X_STO
y[165]  = x[124];       // X_PAO
y[166]  = x[134];       // X_PP
y[167]  = x[144];       // X_PHA
y[168]  = x[154];       // X_A
y[169]  = x[164];       // X_TSS

if (tempmodel < 0.5)    // Temp
    y[170]  = u[18];
else
    y[170]  = x[174];

y[171]  = x[184];       // Dummy 1
y[172]  = x[194];       // Dummy 2
y[173]  = x[204];       // Dummy 3
```

```
y[174]  = x[214];        // Dummy 4
y[175]  = x[224];        // Dummy 5

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx//

// LAYER 6
y[176]  = x[5];          // S_O
y[177]  = x[15];         // S_S
y[178]  = x[25];         // S_I
y[179]  = x[35];         // S_NH
y[180]  = x[45];         // S_NO
y[181]  = x[55];         // S_N2
y[182]  = x[65];         // S_PO4
y[183]  = x[75];         // S_HCO
y[184]  = x[85];         // X_I
y[185]  = x[95];         // X_S
y[186]  = x[105];        // X_H
y[187]  = x[115];        // X_STO
y[188]  = x[125];        // X_PAO
y[189]  = x[135];        // X_PP
y[190]  = x[145];        // X_PHA
y[191]  = x[155];        // X_A
y[192]  = x[165];        // X_TSS

if (tempmodel < 0.5)     // Temp
    y[193]  = u[18];
else
    y[193]  = x[175];

y[194]  = x[185];        // Dummy 1
y[195]  = x[195];        // Dummy 2
y[196]  = x[205];        // Dummy 3
y[197]  = x[215];        // Dummy 4
y[198]  = x[225];        // Dummy 5

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx//

// LAYER 7
y[199]  = x[6];          // S_O
y[200]  = x[16];         // S_S
y[201]  = x[26];         // S_I
y[202]  = x[36];         // S_NH
y[203]  = x[46];         // S_NO
y[204]  = x[56];         // S_N2
y[205]  = x[66];         // S_PO4
y[206]  = x[76];         // S_HCO
y[207]  = x[86];         // X_I
y[208]  = x[96];         // X_S
y[209]  = x[106];        // X_H
y[210]  = x[116];        // X_STO
y[211]  = x[126];        // X_PAO
y[212]  = x[136];        // X_PP
y[213]  = x[146];        // X_PHA
y[214]  = x[156];        // X_A
y[215]  = x[166];        // X_TSS

if (tempmodel < 0.5)     // Temp
    y[216]  = u[18];
else
    y[216]  = x[176];

y[217]  = x[186];        // Dummy 1
y[218]  = x[196];        // Dummy 2
y[219]  = x[206];        // Dummy 3
y[220]  = x[216];        // Dummy 4
y[221]  = x[226];        // Dummy 5

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx//

// LAYER 8
y[222]  = x[7];          // S_O
y[223]  = x[17];         // S_S
y[224]  = x[27];         // S_I
y[225]  = x[37];         // S_NH
y[226]  = x[47];         // S_NO
y[227]  = x[57];         // S_N2
y[228]  = x[67];         // S_PO4
```

```
y[229]  = x[77];        // S_HCO
y[230]  = x[87];        // X_I
y[231]  = x[97];        // X_S
y[232]  = x[107];       // X_H
y[233]  = x[117];       // X_STO
y[234]  = x[127];       // X_PAO
y[235]  = x[137];       // X_PP
y[236]  = x[147];       // X_PHA
y[237]  = x[157];       // X_A
y[238]  = x[167];       // X_TSS

if (tempmodel < 0.5)    // Temp
    y[239]  = u[18];
else
    y[239]  = x[177];

y[240]  = x[187];       // Dummy 1
y[241]  = x[197];       // Dummy 2
y[242]  = x[207];       // Dummy 3
y[243]  = x[217];       // Dummy 4
y[244]  = x[227];       // Dummy 5

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx//

// LAYER 9
y[245]  = x[8];         // S_O
y[246]  = x[18];        // S_S
y[247]  = x[28];        // S_I
y[248]  = x[38];        // S_NH
y[249]  = x[48];        // S_NO
y[250]  = x[58];        // S_N2
y[251]  = x[68];        // S_PO4
y[252]  = x[78];        // S_HCO
y[253]  = x[88];        // X_I
y[254]  = x[98];        // X_S
y[255]  = x[108];       // X_H
y[256]  = x[118];       // X_STO
y[257]  = x[128];       // X_PAO
y[258]  = x[138];       // X_PP
y[259]  = x[148];       // X_PHA
y[260]  = x[158];       // X_A
y[261]  = x[168];       // X_TSS

if (tempmodel < 0.5)    // Temp
    y[262]  = u[18];
else
    y[262]  = x[178];

y[263]  = x[188];       // Dummy 1
y[264]  = x[198];       // Dummy 2
y[265]  = x[208];       // Dummy 3
y[266]  = x[218];       // Dummy 4
y[267]  = x[228];       // Dummy 5

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx//

// LAYER 10
y[268]  = x[9];         // S_O
y[269]  = x[19];        // S_S
y[270]  = x[29];        // S_I
y[271]  = x[39];        // S_NH
y[272]  = x[49];        // S_NO
y[273]  = x[59];        // S_N2
y[274]  = x[69];        // S_PO4
y[275]  = x[79];        // S_HCO
y[276]  = x[89];        // X_I
y[277]  = x[99];        // X_S
y[278]  = x[109];       // X_H
y[279]  = x[119];       // X_STO
y[280]  = x[129];       // X_PAO
y[281]  = x[139];       // X_PP
y[282]  = x[149];       // X_PHA
y[283]  = x[159];       // X_A
y[284]  = x[169];       // X_TSS

if (tempmodel < 0.5)    // Temp
    y[285]  = u[18];
    else
        y[285]  = x[179];

    y[286]  = x[189];       // Dummy 1
    y[287]  = x[199];       // Dummy 2
    y[288]  = x[209];       // Dummy 3
    y[289]  = x[219];       // Dummy 4
    y[290]  = x[229];       // Dummy 5
}

// mdlUpdate - perform action at major integration time step

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

// Sedimentation velocity: no need to update this part because similar for ASM1, ASM3 and
ASM2d

// mdlDerivatives - compute the derivatives

static void mdlDerivatives(double *dx, double *x, double *u, SimStruct *S, int tid)
{

double v0_max, v0, r_h, r_p, f_ns, X_t, area, h, feedlayer, volume;
double Q_f, Q_e, Q_u, v_up, v_dn, v_in, eps;
int    i;
double vs[10];
double Js[11];
double Jstemp[10];
double Jflow[11];

double f_SI, Y_H_O2, Y_H_NO, Y_STO_O2, Y_STO_NO, f_XI, Y_PAO_O2, Y_PAO_NO, Y_PHA, Y_PO4,
Y_AUT;
double iN_SS, iN_SI, iN_XI, iN_XS, iN_BM, iP_SS, iP_SI, iP_XI, iP_XS, iP_BM, iTSS_XI, iTSS_XS,
iTSS_XSTO, iTSS_BM, iTSS_XPP, iNOX_N2, iCOD_NOX, iCOD_N2, icharge_NHX, icharge_NOX,
icharge_XPAO_PP, icharge_PO4;
double k_H, K_X, k_STO, mu_H, h_NO_H, K_SS_H, K_STO_H, b_H, h_NO_end_H, b_STO, K_O_H, K_NO_H,
K_NH_H, K_PO4_H, K_HCO_H;
double q_PHA, q_PP, K_PP_PAO, K_max_PAO, K_iPP_PAO, mu_PAO, h_NO_PAO, K_PHA, b_PAO,
h_NO_end_PAO, b_PP, h_NO_lys_PP, b_PHA, h_NO_resp_PHA, K_SS_PAO, K_O_PAO, K_NO_PAO, K_NH_PAO,
K_PO4_PP, K_PO4_PAO, K_HCO_PAO;
double mu_A, b_A, h_NO_A, K_O_A, K_NH_A, K_PO4_A, K_HCO_A;

double proc1[10], proc2[10], proc3[10], proc4[10], proc5[10], proc6[10], proc7[10], proc8[10],
proc9[10], proc10[10], proc11[10], proc12[10];
double procP1[10], procP2[10], procP3[10], procP4[10], procP5[10], procP6[10], procP7[10],
procP8[10], procP9[10], procP10[10], procP11[10];
double reac1[10], reac2[10], reac3[10], reac4[10], reac5[10], reac6[10], reac7[10],
reac8_1[10], reac8_2[10], reac8_3[10], reac8_4[10], reac8[10], reac9[10], reac10[10],
reac11[10], reac12[10], reac13[10], reac14[10], reac15[10], reac16[10], reac17_1[10],
reac17_2[10], reac17[10], reac18[10], reac19[10], reac20[10], reac21[10], reac22[10];

double k_H_T[10], k_STO_T[10], mu_H_T[10], b_H_T[10], b_STO_T[10], mu_A_T[10], b_A_T[10],
q_PHA_T[10], q_PP_T[10], mu_PAO_T[10], b_PAO_T[10], b_PP_T[10], b_PHA_T[10];

double xtemp[250];

double reactive_settler;
double tempmodel;


v0_max      = mxGetPr(SEDPAR)[0];                // Takacs parameter
v0          = mxGetPr(SEDPAR)[1];                // Takacs parameter
r_h         = mxGetPr(SEDPAR)[2];                // Takacs parameter
r_p         = mxGetPr(SEDPAR)[3];                // Takacs parameter
f_ns        = mxGetPr(SEDPAR)[4];                // Takacs parameter
X_t         = mxGetPr(SEDPAR)[5];                // Takacs parameter
area        = mxGetPr(DIM)[0];                   // settler area
h           = mxGetPr(DIM)[1]/mxGetPr(LAYER)[1]; // height of a layer in the settler =
height settler/number of layers
feedlayer   = mxGetPr(LAYER)[0];                 // indicate which layer is feed layer
volume      = area*mxGetPr(DIM)[1];              // settler volume


// ASM3_Bio-P model parameters (Reference: Hauduc et al.,2010; 20 deg.C)
// Stoichiometric parameters
```

```
f_SI            = mxGetPr(ASMPAR)[0];

Y_H_O2          = mxGetPr(ASMPAR)[1];
Y_H_NO          = mxGetPr(ASMPAR)[2];
Y_STO_O2        = mxGetPr(ASMPAR)[3];
Y_STO_NO        = mxGetPr(ASMPAR)[4];

f_XI            = mxGetPr(ASMPAR)[5];

Y_PAO_O2        = mxGetPr(ASMPAR)[6];
Y_PAO_NO        = mxGetPr(ASMPAR)[7];
Y_PHA           = mxGetPr(ASMPAR)[8];
Y_PO4           = mxGetPr(ASMPAR)[9];

Y_AUT           = mxGetPr(ASMPAR)[10];

iN_SS           = mxGetPr(ASMPAR)[11];
iN_SI           = mxGetPr(ASMPAR)[12];
iN_XI           = mxGetPr(ASMPAR)[13];
iN_XS           = mxGetPr(ASMPAR)[14];
iN_BM           = mxGetPr(ASMPAR)[15];

iP_SS           = mxGetPr(ASMPAR)[16];
iP_SI           = mxGetPr(ASMPAR)[17];
iP_XI           = mxGetPr(ASMPAR)[18];
iP_XS           = mxGetPr(ASMPAR)[19];
iP_BM           = mxGetPr(ASMPAR)[20];

iTSS_XI         = mxGetPr(ASMPAR)[21];
iTSS_XS         = mxGetPr(ASMPAR)[22];
iTSS_XSTO       = mxGetPr(ASMPAR)[23];
iTSS_BM         = mxGetPr(ASMPAR)[24];
iTSS_XPP        = mxGetPr(ASMPAR)[25];

iNOX_N2         = mxGetPr(ASMPAR)[26];
iCOD_NOX        = mxGetPr(ASMPAR)[27];
iCOD_N2         = mxGetPr(ASMPAR)[28];
icharge_NHX     = mxGetPr(ASMPAR)[29];
icharge_NOX     = mxGetPr(ASMPAR)[30];
icharge_XPAO_PP = mxGetPr(ASMPAR)[31];
icharge_PO4     = mxGetPr(ASMPAR)[32];

// Kinetic parameters
k_H             = mxGetPr(ASMPAR)[33];
K_X             = mxGetPr(ASMPAR)[34];

k_STO           = mxGetPr(ASMPAR)[35];
mu_H            = mxGetPr(ASMPAR)[36];
h_NO_H          = mxGetPr(ASMPAR)[37];
K_SS_H          = mxGetPr(ASMPAR)[38];
K_STO_H         = mxGetPr(ASMPAR)[39];
b_H             = mxGetPr(ASMPAR)[40];
h_NO_end_H      = mxGetPr(ASMPAR)[41];
b_STO           = mxGetPr(ASMPAR)[42];
K_O_H           = mxGetPr(ASMPAR)[43];
K_NO_H          = mxGetPr(ASMPAR)[44];
K_NH_H          = mxGetPr(ASMPAR)[45];
K_PO4_H         = mxGetPr(ASMPAR)[46];
K_HCO_H         = mxGetPr(ASMPAR)[47];

q_PHA           = mxGetPr(ASMPAR)[48];
q_PP            = mxGetPr(ASMPAR)[49];
K_PP_PAO        = mxGetPr(ASMPAR)[50];
K_max_PAO       = mxGetPr(ASMPAR)[51];
K_iPP_PAO       = mxGetPr(ASMPAR)[52];
mu_PAO          = mxGetPr(ASMPAR)[53];
h_NO_PAO        = mxGetPr(ASMPAR)[54];
K_PHA           = mxGetPr(ASMPAR)[55];
b_PAO           = mxGetPr(ASMPAR)[56];
h_NO_end_PAO    = mxGetPr(ASMPAR)[57];
b_PP            = mxGetPr(ASMPAR)[58];
h_NO_lys_PP     = mxGetPr(ASMPAR)[59];
b_PHA           = mxGetPr(ASMPAR)[60];
h_NO_resp_PHA   = mxGetPr(ASMPAR)[61];
K_SS_PAO        = mxGetPr(ASMPAR)[62];
K_O_PAO         = mxGetPr(ASMPAR)[63];
K_NO_PAO        = mxGetPr(ASMPAR)[64];
```

```
K_NH_PAO         = mxGetPr(ASMPAR)[65];
K_PO4_PP         = mxGetPr(ASMPAR)[66];
K_PO4_PAO        = mxGetPr(ASMPAR)[67];
K_HCO_PAO        = mxGetPr(ASMPAR)[68];

mu_A             = mxGetPr(ASMPAR)[69];
b_A              = mxGetPr(ASMPAR)[70];
h_NO_A           = mxGetPr(ASMPAR)[71];
K_O_A            = mxGetPr(ASMPAR)[72];
K_NH_A           = mxGetPr(ASMPAR)[73];
K_PO4_A          = mxGetPr(ASMPAR)[74];
K_HCO_A          = mxGetPr(ASMPAR)[75];


// Switch
reactive_settler    = mxGetPr(SETTLER)[0];
tempmodel           = mxGetPr(TEMPMODEL)[0];


eps     = 0.01;
v_in    = u[17]/area;
Q_f     = u[17];            // influent flow rate
Q_u     = u[24] + u[25];    // underflow flow rate
Q_e     = u[17] - Q_u;      // effluent flow rate
v_up    = Q_e/area;         // upstream flow velocity
v_dn    = Q_u/area;         // downstream flow velocity


for (i = 0; i < 230; i++) {
   if (x[i] < 0.0)
     xtemp[i] = 0.0;
   else
     xtemp[i] = x[i];
}



// Calculation of the sedimentation velocity for each of the layers

for (i = 0; i < 10; i++) {
   vs[i] = v0*(exp(-r_h*(xtemp[160+i]-f_ns*u[16]))-exp(-r_p*(xtemp[160+i]-f_ns*u[16])));   //
u[16] = influent SS concentration
   if (vs[i] > v0_max)
      vs[i] = v0_max;
   else if (vs[i] < 0.0)
      vs[i] = 0.0;
}

// Calculation of the sludge flux due to sedimentation for each layer

for (i = 0; i < 10; i++) {
   Jstemp[i] = vs[i]*xtemp[160+i];
}

// Calculation of the sludge flux due to the liquid flow (upflow or downflow, depending on
layer)

for (i = 0; i < 11; i++) {
   if (i < (feedlayer-eps))
      Jflow[i] = v_up*xtemp[160+i];
   else
      Jflow[i] = v_dn*xtemp[160+i-1];
}

Js[0]  = 0.0;
Js[10] = 0.0;
for (i = 0; i < 9; i++) {
   if ((i < (feedlayer-1-eps)) && (xtemp[160+i+1] <= X_t))
      Js[i+1] = Jstemp[i];
   else if (Jstemp[i] < Jstemp[i+1])
      Js[i+1] = Jstemp[i];
   else
      Js[i+1] = Jstemp[i+1];
}

// Reaction rates ASM3_bioP
```

```
for (i = 0; i < 10; i++) {

if (tempmodel < 0.5) {
    // Compensation from the temperature at the influent of the reactor
    k_H_T[i]     = k_H   *exp(0.04 *(u[18]-20));
    k_STO_T[i]   = k_STO *exp(0.07 *(u[18]-20));
    mu_H_T[i]    = mu_H  *exp(0.07 *(u[18]-20));
    b_H_T[i]     = b_H   *exp(0.07 *(u[18]-20));
    b_STO_T[i]   = b_STO *exp(0.07 *(u[18]-20));
    mu_A_T[i]    = mu_A  *exp(0.105*(u[18]-20));
    b_A_T[i]     = b_A   *exp(0.105*(u[18]-20));

    q_PHA_T[i]   = q_PHA *exp(0.04 *(u[18]-20));
    q_PP_T[i]    = q_PP  *exp(0.04 *(u[18]-20));
    mu_PAO_T[i]  = mu_PAO*exp(0.07 *(u[18]-20));
    b_PAO_T[i]   = b_PAO *exp(0.07 *(u[18]-20));
    b_PP_T[i]    = b_PP  *exp(0.07 *(u[18]-20));
    b_PHA_T[i]   = b_PHA *exp(0.07 *(u[18]-20));

}
else {
    // Compensation from the current temperature at each layer
    k_H_T[i]     = k_H   *exp(0.04 *(x[i+170]-20));
    k_STO_T[i]   = k_STO *exp(0.07 *(x[i+170]-20));
    mu_H_T[i]    = mu_H  *exp(0.07 *(x[i+170]-20));
    b_H_T[i]     = b_H   *exp(0.07 *(x[i+170]-20));
    b_STO_T[i]   = b_STO *exp(0.07 *(x[i+170]-20));
    mu_A_T[i]    = mu_A  *exp(0.105*(x[i+170]-20));
    b_A_T[i]     = b_A   *exp(0.105*(x[i+170]-20));

    q_PHA_T[i]   = q_PHA *exp(0.04 *(x[i+170]-20));
    q_PP_T[i]    = q_PP  *exp(0.04 *(x[i+170]-20));
    mu_PAO_T[i]  = mu_PAO*exp(0.07 *(x[i+170]-20));
    b_PAO_T[i]   = b_PAO *exp(0.07 *(x[i+170]-20));
    b_PP_T[i]    = b_PP  *exp(0.07 *(x[i+170]-20));
    b_PHA_T[i]   = b_PHA *exp(0.07 *(x[i+170]-20));
}

if (reactive_settler > 0.5) {

proc1[i]    =
k_H_T[i]*((xtemp[i+90]/xtemp[i+100])/(K_X+xtemp[i+90]/xtemp[i+100]))*xtemp[i+100];
proc2[i]    =
k_STO_T[i]*(xtemp[i]/(K_O_H+xtemp[i]))*(xtemp[i+10]/(K_SS_H+xtemp[i+10]))*xtemp[i+100];
proc3[i]    =
k_STO_T[i]*h_NO_H*(K_O_H/(K_O_H+xtemp[i]))*(xtemp[i+40]/(K_NO_H+xtemp[i+40]))*(xtemp[i+10]/(K_
SS_H+xtemp[i+10]))*xtemp[i+100];
proc4[i]    =
mu_H_T[i]*(xtemp[i]/(K_O_H+xtemp[i]))*(xtemp[i+30]/(K_NH_H+xtemp[i+30]))*(xtemp[i+70]/(K_HCO_H
+xtemp[i+70]))*(xtemp[i+60]/(K_PO4_H+xtemp[i+60]))*((xtemp[i+110]/xtemp[i+100])/(K_STO_H+xtemp
[i+110]/xtemp[i+100]))*xtemp[i+100];
proc5[i]    =
mu_H_T[i]*h_NO_H*(K_O_H/(K_O_H+xtemp[i]))*(xtemp[i+40]/(K_NO_H+xtemp[i+40]))*(xtemp[i+30]/(K_N
H_H+xtemp[i+30]))*(xtemp[i+70]/(K_HCO_H+xtemp[i+70]))*(xtemp[i+60]/(K_PO4_H+xtemp[i+60]))*((xt
emp[i+110]/xtemp[i+100])/(K_STO_H+xtemp[i+110]/xtemp[i+100]))*xtemp[i+100];
proc6[i]    = b_H_T[i]*(xtemp[i]/(K_O_H+xtemp[i]))*xtemp[i+100];
proc7[i]    =
b_H_T[i]*h_NO_end_H*(K_O_H/(K_O_H+xtemp[i]))*(xtemp[i+40]/(K_NO_H+xtemp[i+40]))*xtemp[i+100];
proc8[i]    = b_STO_T[i]*(xtemp[i]/(K_O_H+xtemp[i]))*xtemp[i+110];
proc9[i]    =
b_STO_T[i]*h_NO_end_H*(K_O_H/(K_O_H+xtemp[i]))*(xtemp[i+40]/(K_NO_H+xtemp[i+40]))*xtemp[i+110]
;
proc10[i]   =
mu_A_T[i]*(xtemp[i]/(K_O_A+xtemp[i]))*(xtemp[i+30]/(K_NH_A+xtemp[i+30]))*(xtemp[i+70]/(K_HCO_A
+xtemp[i+70]))*(xtemp[i+60]/(K_PO4_A+xtemp[i+60]))*xtemp[i+150];
proc11[i]   = b_A_T[i]*(xtemp[i]/(K_O_A+xtemp[i]))*xtemp[i+150];
proc12[i]   =
b_A_T[i]*h_NO_A*(K_O_A/(K_O_A+xtemp[i]))*(xtemp[i+40]/(K_NO_H+xtemp[i+40]))*xtemp[i+150];

procP1[i]   =
q_PHA_T[i]*(xtemp[i+10]/(K_SS_PAO+xtemp[i+10]))*(xtemp[i+70]/(K_HCO_PAO+xtemp[i+70]))*((xtemp[
i+130]/xtemp[i+120])/(K_PP_PAO+xtemp[i+130]/xtemp[i+120]))*xtemp[i+120];
procP2[i]   =
q_PP_T[i]*(xtemp[i]/(K_O_PAO+xtemp[i]))*(xtemp[i+60]/(K_PO4_PP+xtemp[i+60]))*(xtemp[i+70]/(K_H
CO_PAO+xtemp[i+70]))*((xtemp[i+140]/xtemp[i+120])/(K_PHA+xtemp[i+140]/xtemp[i+120]))*((K_max_P
AO-(xtemp[i+130]/xtemp[i+120]))/(K_iPP_PAO+K_max_PAO-
(xtemp[i+130]/xtemp[i+120])))*xtemp[i+120];
```

```
procP3[i]  =
q_PP_T[i]*h_NO_PAO*(K_O_PAO/(K_O_PAO+xtemp[i]))*(xtemp[i+40]/(K_NO_PAO+xtemp[i+40]))*(xtemp[i+
60]/(K_PO4_PP+xtemp[i+60]))*(xtemp[i+70]/(K_HCO_PAO+xtemp[i+70]))*((xtemp[i+140]/xtemp[i+120])
/(K_PHA+xtemp[i+140]/xtemp[i+120]))*((K_max_PAO-
(xtemp[i+130]/xtemp[i+120]))/(K_iPP_PAO+K_max_PAO-(xtemp[i+140]/xtemp[i+120])))*xtemp[i+120];
procP4[i]  =
mu_PAO_T[i]*(xtemp[i]/(K_O_PAO+xtemp[i]))*(xtemp[i+30]/(K_NH_PAO+xtemp[i+30]))*(xtemp[i+60]/(K
_PO4_PAO+xtemp[i+60]))*(xtemp[i+70]/(K_HCO_PAO+xtemp[i+70]))*((xtemp[i+140]/xtemp[i+120])/(K_P
HA+xtemp[i+140]/xtemp[i+120]))*xtemp[i+120];
procP5[i]  =
mu_PAO_T[i]*h_NO_PAO*(K_O_PAO/(K_O_PAO+xtemp[i]))*(xtemp[i+40]/(K_NO_PAO+xtemp[i+40]))*(xtemp[
i+30]/(K_NH_PAO+xtemp[i+30]))*(xtemp[i+60]/(K_PO4_PAO+xtemp[i+60]))*(xtemp[i+70]/(K_HCO_PAO+xt
emp[i+70]))*((xtemp[i+140]/xtemp[i+120])/(K_PHA+xtemp[i+140]/xtemp[i+120]))*xtemp[i+120];
procP6[i]  = b_PAO_T[i]*(xtemp[i]/(K_O_PAO+xtemp[i]))*xtemp[i+120];
procP7[i]  =
b_PAO_T[i]*h_NO_end_PAO*(K_O_PAO/(K_O_PAO+xtemp[i]))*(xtemp[i+40]/(K_NO_PAO+xtemp[i+40]))*xtem
p[i+120];
procP8[i]  =
b_PP_T[i]*(xtemp[i]/(K_O_PAO+xtemp[i]))*(xtemp[i+70]/(K_HCO_PAO+xtemp[i+70]))*xtemp[i+130];
procP9[i]  =
b_PP_T[i]*h_NO_lys_PP*(K_O_PAO/(K_O_PAO+xtemp[i]))*(xtemp[i+40]/(K_NO_PAO+xtemp[i+40]))*(xtemp
[i+70]/(K_HCO_PAO+xtemp[i+70]))*xtemp[i+130];
procP10[i] = b_PHA_T[i]*(xtemp[i]/(K_O_PAO+xtemp[i]))*xtemp[i+140];
procP11[i] =
b_PHA_T[i]*h_NO_resp_PHA*(K_O_PAO/(K_O_PAO+xtemp[i]))*(xtemp[i+40]/(K_NO_PAO+xtemp[i+40]))*xte
mp[i+140];

reac1[i]   = (Y_STO_O2-1)*proc2[i] + (1-1/Y_H_O2)*proc4[i] + (f_XI-
1)*(proc6[i]+proc11[i]+procP6[i]) + (-1)*(proc8[i]+procP10[i]) + (1+iCOD_NOX/Y_AUT)*proc10[i]
+ (-Y_PHA)*procP2[i] + (1-1/Y_PAO_O2)*procP4[i];
reac2[i]   = (1-f_SI)*proc1[i] + (-1)*(proc2[i]+proc3[i]+procP1[i]);
reac3[i]   = (f_SI)*proc1[i];
reac4[i]   = ((f_SI-1)*iN_SS-f_SI*iN_SI+iN_XS)*proc1[i] +
(iN_SS)*(proc2[i]+proc3[i]+procP1[i]) + (-iN_BM)*(proc4[i]+proc5[i]+procP4[i]+procP5[i]) +
(iN_BM-f_XI*iN_XI)*(proc6[i]+proc7[i]+proc11[i]+proc12[i]+procP6[i]+procP7[i]) + (-1/Y_AUT-
iN_BM)*proc10[i];
reac5[i]   = ((Y_STO_NO-1)/iNOX_N2)*proc3[i] + ((1-1/Y_H_NO)/iNOX_N2)*proc5[i] + ((f_XI-
1)/iNOX_N2)*(proc7[i]+proc12[i]+procP7[i]) + (-1/iNOX_N2)*(proc9[i]+procP11[i]) +
(1/Y_AUT)*proc10[i] + (-Y_PHA/iNOX_N2)*procP3[i] + ((1-1/Y_PAO_NO)/iNOX_N2)*procP5[i];
reac6[i]   = ((1-Y_STO_NO)/iNOX_N2)*proc3[i] + ((1/Y_H_NO-1)/iNOX_N2)*proc5[i] + ((1-
f_XI)/iNOX_N2)*(proc7[i]+proc12[i]+procP7[i]) + (1/iNOX_N2)*(proc9[i]+procP11[i]) +
(Y_PHA/iNOX_N2)*procP3[i] + ((1/Y_PAO_NO-1)/iNOX_N2)*procP5[i];
reac7[i]   = ((f_SI-1)*iP_SS-f_SI*iP_SS+iP_XS)*proc1[i] + (iP_SS)*(proc2[i]+proc3[i]) + (-
iP_BM)*(proc4[i]+proc5[i]+proc10[i]+procP4[i]+procP5[i]) + (iP_BM-
f_XI*iP_XI)*(proc6[i]+proc7[i]+proc11[i]+proc12[i]+procP6[i]+procP7[i]) +
(Y_PO4+iP_SS)*procP1[i] + (-1)*(procP2[i]+procP3[i]) + (1)*(procP8[i]+procP9[i]);
reac8_1[i] = (((f_SI-1)*iN_SS-f_SI*iN_SI+iN_XS)*icharge_NHX+((f_SI-1)*iP_SS-
f_SI*iP_SI+iP_XS)*icharge_PO4)*proc1[i] + (iN_SS*icharge_NHX+iP_SS*icharge_PO4)*proc2[i] +
(iN_SS*icharge_NHX+(Y_STO_NO-1)/iNOX_N2*icharge_NOX+iP_SS*icharge_PO4)*proc3[i] + (-
iN_BM*icharge_NHX+(-iP_BM)*icharge_PO4)*proc4[i] + (-iN_BM*icharge_NHX+(1-
1/Y_H_NO)/iNOX_N2*icharge_NOX+(-iP_BM)*icharge_PO4)*proc5[i];
reac8_2[i] = ((iN_BM-f_XI*iN_XI)*icharge_NHX+(iP_BM-f_XI*iP_XI)*icharge_PO4)*proc6[i] +
((iN_BM-f_XI*iN_XI)*icharge_NHX+(f_XI-1)/iNOX_N2*icharge_NOX+(iP_BM-
f_XI*iP_XI)*icharge_PO4)*proc7[i] + (-1/iNOX_N2*icharge_NOX)*proc9[i] + ((-1/Y_AUT-
iN_BM)*icharge_NHX+(1/Y_AUT)*icharge_NOX+(-iP_BM)*icharge_PO4)*proc10[i] + ((iN_BM-
f_XI*iN_XI)*icharge_NHX+(iP_BM-f_XI*iP_XI)*icharge_PO4)*proc11[i] + ((iN_BM-
f_XI*iN_XI)*icharge_NHX+(f_XI-1)/iNOX_N2*icharge_NOX+(iP_BM-
f_XI*iP_XI)*icharge_PO4)*proc12[i];
reac8_3[i] = (iN_SS*icharge_NHX+(Y_PO4+iP_SS)*icharge_PO4+(-Y_PO4)*icharge_XPAO_PP)*procP1[i]
+ (-1*icharge_PO4+1*icharge_XPAO_PP)*procP2[i] + (-Y_PHA/iNOX_N2*icharge_NOX+(-
1)*icharge_PO4+1*icharge_XPAO_PP)*procP3[i] + (-iN_BM*icharge_NHX+(-
iP_BM)*icharge_PO4)*procP4[i] + (-iN_BM*icharge_NHX+(1-1/Y_PAO_NO)/iNOX_N2*icharge_NOX+(-
iP_BM)*icharge_PO4)*procP5[i];
reac8_4[i] = ((iN_BM-f_XI*iN_XI)*icharge_NHX+(iP_BM-f_XI*iP_XI)*icharge_PO4)*procP6[i] +
((iN_BM-f_XI*iN_XI)*icharge_NHX+(f_XI-1)/iNOX_N2*icharge_NOX+(iP_BM-
f_XI*iP_XI)*icharge_PO4)*procP7[i] + (1*icharge_PO4+(-
1)*icharge_XPAO_PP)*(procP8[i]+procP9[i]) + (-1/iNOX_N2*icharge_NOX)*procP11[i];
reac8[i]   = reac8_1[i]+reac8_2[i]+reac8_3[i]+reac8_4[i];
reac9[i]   = (f_XI)*(proc6[i]+proc7[i]+proc11[i]+proc12[i]+procP6[i]+procP7[i]);
reac10[i]  = (-1)*proc1[i];
reac11[i]  = (1)*(proc4[i]+proc5[i]) + (-1)*(proc6[i]+proc7[i]);
reac12[i]  = (Y_STO_O2)*proc2[i] + (Y_STO_NO)*proc3[i] + (-1/Y_H_O2)*proc4[i] + (-
1/Y_H_NO)*proc5[i] + (-1)*(proc8[i]+proc9[i]);
reac13[i]  = (1)*(procP4[i]+procP5[i]) + (-1)*(procP6[i]+procP7[i]);
reac14[i]  = (-Y_PO4)*procP1[i] + (1)*(procP2[i]+procP3[i]) + (-1)*(procP8[i]+procP9[i]);
reac15[i]  = (1)*procP1[i] + (-Y_PHA)*(procP2[i]+procP3[i]) + (-1/Y_PAO_O2)*procP4[i] + (-
1/Y_PAO_NO)*procP5[i] + (-1)*(procP10[i]+procP11[i]);
```

```
reac16[i]  = (1)*proc10[i] + (-1)*(proc11[i]+proc12[i]);
reac17_1[i]= (-iTSS_XS)*proc1[i] + (Y_STO_O2*iTSS_XSTO)*proc2[i] +
(Y_STO_NO*iTSS_XSTO)*proc3[i] + (iTSS_BM-iTSS_XSTO/Y_H_O2)*proc4[i] + (iTSS_BM-
iTSS_XSTO/Y_H_NO)*proc5[i] + (f_XI*iTSS_XI-
iTSS_BM)*(proc6[i]+proc7[i]+proc11[i]+proc12[i]+procP6[i]+procP7[i]) + (-
iTSS_XSTO)*(proc8[i]+proc9[i]+procP10[i]+procP11[i]);
reac17_2[i]= (iTSS_BM)*proc10[i] + (iTSS_XSTO-iTSS_XPP*Y_PO4)*procP1[i] + (iTSS_XPP-
Y_PHA*iTSS_XSTO)*(procP2[i]+procP3[i]) + (iTSS_BM-iTSS_XSTO/Y_PAO_O2)*procP4[i] + (iTSS_BM-
iTSS_XSTO/Y_PAO_NO)*procP5[i] + (-iTSS_XPP)*(procP8[i]+procP9[i]);
reac17[i]  = reac17_1[i]+reac17_2[i];

reac18[i]  = 0.0;
reac19[i]  = 0.0;
reac20[i]  = 0.0;
reac21[i]  = 0.0;
reac22[i]  = 0.0;
}
else if (reactive_settler  < 0.5) {
reac1[i]   = 0.0;
reac2[i]   = 0.0;
reac3[i]   = 0.0;
reac4[i]   = 0.0;
reac5[i]   = 0.0;
reac6[i]   = 0.0;
reac7[i]   = 0.0;
reac8_1[i] = 0.0;
reac8_2[i] = 0.0;
reac8_3[i] = 0.0;
reac8_4[i] = 0.0;
reac8[i]   = 0.0;
reac9[i]   = 0.0;
reac10[i]  = 0.0;
reac11[i]  = 0.0;
reac12[i]  = 0.0;
reac13[i]  = 0.0;
reac14[i]  = 0.0;
reac15[i]  = 0.0;
reac16[i]  = 0.0;
reac17_1[i]= 0.0;
reac17_2[i]= 0.0;
reac17[i]  = 0.0;

reac18[i]  = 0.0;
reac19[i]  = 0.0;
reac20[i]  = 0.0;
reac21[i]  = 0.0;
reac22[i]  = 0.0;

}
}

// ASM3_bioP model component balances over the layers

// S_O
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i]     = (-v_up*xtemp[i]+v_up*xtemp[i+1])/h+reac1[i];                  // above feed
layer
   else if (i > (feedlayer-eps))
      dx[i]     = (v_dn*xtemp[i-1]-v_dn*xtemp[i])/h+reac1[i];                   // below feed
layer
   else
      dx[i]     = (v_in*u[0]-v_up*xtemp[i]-v_dn*xtemp[i])/h+reac1[i];           // feed layer
}

// S_S
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+10]  = (-v_up*xtemp[i+10]+v_up*xtemp[i+1+10])/h+reac2[i];
   else if (i > (feedlayer-eps))
      dx[i+10]  = (v_dn*xtemp[i-1+10]-v_dn*xtemp[i+10])/h+reac2[i];
   else
      dx[i+10]  = (v_in*u[1]-v_up*xtemp[i+10]-v_dn*xtemp[i+10])/h+reac2[i];
}

// S_I
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+20]  = (-v_up*xtemp[i+20]+v_up*xtemp[i+1+20])/h+reac3[i];
   else if (i > (feedlayer-eps))
      dx[i+20]  = (v_dn*xtemp[i-1+20]-v_dn*xtemp[i+20])/h+reac3[i];
   else
      dx[i+20]  = (v_in*u[2]-v_up*xtemp[i+20]-v_dn*xtemp[i+20])/h+reac3[i];
}

// S_NH
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+30]  = (-v_up*xtemp[i+30]+v_up*xtemp[i+1+30])/h+reac4[i];
   else if (i > (feedlayer-eps))
      dx[i+30]  = (v_dn*xtemp[i-1+30]-v_dn*xtemp[i+30])/h+reac4[i];
   else
      dx[i+30]  = (v_in*u[3]-v_up*xtemp[i+30]-v_dn*xtemp[i+30])/h+reac4[i];
}

// S_NO
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+40]  = (-v_up*xtemp[i+40]+v_up*xtemp[i+1+40])/h+reac5[i];
   else if (i > (feedlayer-eps))
      dx[i+40]  = (v_dn*xtemp[i-1+40]-v_dn*xtemp[i+40])/h+reac5[i];
   else
      dx[i+40]  = (v_in*u[4]-v_up*xtemp[i+40]-v_dn*xtemp[i+40])/h+reac5[i];
}

// S_N2
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+50]  = (-v_up*xtemp[i+50]+v_up*xtemp[i+1+50])/h+reac6[i];
   else if (i > (feedlayer-eps))
      dx[i+50]  = (v_dn*xtemp[i-1+50]-v_dn*xtemp[i+50])/h+reac6[i];
   else
      dx[i+50]  = (v_in*u[5]-v_up*xtemp[i+50]-v_dn*xtemp[i+50])/h+reac6[i];
}

// S_PO4
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+60]  = (-v_up*xtemp[i+60]+v_up*xtemp[i+1+60])/h+reac7[i];
   else if (i > (feedlayer-eps))
      dx[i+60]  = (v_dn*xtemp[i-1+60]-v_dn*xtemp[i+60])/h+reac7[i];
   else
      dx[i+60]  = (v_in*u[6]-v_up*xtemp[i+60]-v_dn*xtemp[i+60])/h+reac7[i];
}

// S_HCO
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+70]  = (-v_up*xtemp[i+70]+v_up*xtemp[i+1+70])/h+reac8[i];
   else if (i > (feedlayer-eps))
      dx[i+70]  = (v_dn*xtemp[i-1+70]-v_dn*xtemp[i+70])/h+reac8[i];
   else
      dx[i+70]  = (v_in*u[7]-v_up*xtemp[i+70]-v_dn*xtemp[i+70])/h+reac8[i];
}

// X_I
    dx[80] = ((x[80]/x[160])*(-Jflow[0]-Js[1])+(x[81]/x[161])*Jflow[1])/h +reac9[0];
for (i = 0; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+80] = ((xtemp[i+80]/xtemp[i+160])*(-Jflow[i]-Js[i+1])+(xtemp[i-1+80]/xtemp[i-
1+160])*Js[i]+(xtemp[i+1+80]/xtemp[i+1+160])*Jflow[i+1])/h+reac9[i];
   else if (i > (feedlayer-eps))
      dx[i+80] = ((xtemp[i+80]/xtemp[i+160])*(-Jflow[i+1]-Js[i+1])+(xtemp[i-1+80]/xtemp[i-
1+160])*(Jflow[i]+Js[i]))/h+reac9[i];
   else
      dx[i+80] = ((xtemp[i+80]/xtemp[i+160])*(-Jflow[i]-Jflow[i+1]-Js[i+1])+(xtemp[i-
1+80]/xtemp[i-1+160])*Js[i]+v_in*u[8])/h+reac9[i];
}

// X_S
      dx[90] = ((x[90]/x[160])*(-Jflow[0]-Js[1])+(x[91]/x[161])*Jflow[1])/h +reac10[0];
for (i = 1; i < 10; i++) {
   if (i < (feedlayer-1-eps))
```

```
      dx[i+90] = ((xtemp[i+90]/xtemp[i+160])*(-Jflow[i]-Js[i+1])+(xtemp[i-1+90]/xtemp[i-
1+160])*Js[i]+(xtemp[i+1+90]/xtemp[i+1+160])*Jflow[i+1])/h+reac10[i];
   else if (i > (feedlayer-eps))
      dx[i+90] = ((xtemp[i+90]/xtemp[i+160])*(-Jflow[i+1]-Js[i+1])+(xtemp[i-1+90]/xtemp[i-
1+160])*(Jflow[i]+Js[i]))/h+reac10[i];
   else
      dx[i+90] = ((xtemp[i+90]/xtemp[i+160])*(-Jflow[i]-Jflow[i+1]-Js[i+1])+(xtemp[i-
1+90]/xtemp[i-1+160])*Js[i]+v_in*u[9])/h+reac10[i];
}

// X_H
      dx[100] = ((x[100]/x[160])*(-Jflow[0]-Js[1])+(x[101]/x[161])*Jflow[1])/h +reac11[0];
for (i = 1; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+100] = ((xtemp[i+100]/xtemp[i+160])*(-Jflow[i]-Js[i+1])+(xtemp[i-1+100]/xtemp[i-
1+160])*Js[i]+(xtemp[i+1+100]/xtemp[i+1+160])*Jflow[i+1])/h+reac11[i];
   else if (i > (feedlayer-eps))
      dx[i+100] = ((xtemp[i+100]/xtemp[i+160])*(-Jflow[i+1]-Js[i+1])+(xtemp[i-1+100]/xtemp[i-
1+160])*(Jflow[i]+Js[i]))/h+reac11[i];
   else
      dx[i+100] = ((xtemp[i+100]/xtemp[i+160])*(-Jflow[i]-Jflow[i+1]-Js[i+1])+(xtemp[i-
1+100]/xtemp[i-1+160])*Js[i]+v_in*u[10])/h+reac11[i];
}

// X_STO
      dx[110] = ((x[110]/x[160])*(-Jflow[0]-Js[1])+(x[111]/x[161])*Jflow[1])/h +reac12[0];
for (i = 1; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+110] = ((xtemp[i+110]/xtemp[i+160])*(-Jflow[i]-Js[i+1])+(xtemp[i-1+110]/xtemp[i-
1+160])*Js[i]+(xtemp[i+1+110]/xtemp[i+1+160])*Jflow[i+1])/h+reac12[i];
   else if (i > (feedlayer-eps))
      dx[i+110] = ((xtemp[i+110]/xtemp[i+160])*(-Jflow[i+1]-Js[i+1])+(xtemp[i-1+110]/xtemp[i-
1+160])*(Jflow[i]+Js[i]))/h+reac12[i];
   else
      dx[i+110] = ((xtemp[i+110]/xtemp[i+160])*(-Jflow[i]-Jflow[i+1]-Js[i+1])+(xtemp[i-
1+110]/xtemp[i-1+160])*Js[i]+v_in*u[11])/h+reac12[i];
}

// X_PAO
      dx[120] = ((x[120]/x[160])*(-Jflow[0]-Js[1])+(x[121]/x[161])*Jflow[1])/h +reac13[0];
for (i = 1; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+120] = ((xtemp[i+120]/xtemp[i+160])*(-Jflow[i]-Js[i+1])+(xtemp[i-1+120]/xtemp[i-
1+160])*Js[i]+(xtemp[i+1+120]/xtemp[i+1+160])*Jflow[i+1])/h+reac13[i];
   else if (i > (feedlayer-eps))
      dx[i+120] = ((xtemp[i+120]/xtemp[i+160])*(-Jflow[i+1]-Js[i+1])+(xtemp[i-1+120]/xtemp[i-
1+160])*(Jflow[i]+Js[i]))/h+reac13[i];
   else
      dx[i+120] = ((xtemp[i+120]/xtemp[i+160])*(-Jflow[i]-Jflow[i+1]-Js[i+1])+(xtemp[i-
1+120]/xtemp[i-1+160])*Js[i]+v_in*u[12])/h+reac13[i];
}

// X_PP
      dx[130] = ((x[130]/x[160])*(-Jflow[0]-Js[1])+(x[131]/x[161])*Jflow[1])/h +reac14[0];
for (i = 1; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+130] = ((xtemp[i+130]/xtemp[i+160])*(-Jflow[i]-Js[i+1])+(xtemp[i-1+130]/xtemp[i-
1+160])*Js[i]+(xtemp[i+1+130]/xtemp[i+1+160])*Jflow[i+1])/h+reac14[i];
   else if (i > (feedlayer-eps))
      dx[i+130] = ((xtemp[i+130]/xtemp[i+160])*(-Jflow[i+1]-Js[i+1])+(xtemp[i-1+130]/xtemp[i-
1+160])*(Jflow[i]+Js[i]))/h+reac14[i];
   else
      dx[i+130] = ((xtemp[i+130]/xtemp[i+160])*(-Jflow[i]-Jflow[i+1]-Js[i+1])+(xtemp[i-
1+130]/xtemp[i-1+160])*Js[i]+v_in*u[13])/h+reac14[i];
}

// X_PHA
      dx[140] = ((x[140]/x[160])*(-Jflow[0]-Js[1])+(x[141]/x[161])*Jflow[1])/h +reac15[0];
for (i = 1; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+140] = ((xtemp[i+140]/xtemp[i+160])*(-Jflow[i]-Js[i+1])+(xtemp[i-1+140]/xtemp[i-
1+160])*Js[i]+(xtemp[i+1+140]/xtemp[i+1+160])*Jflow[i+1])/h+reac15[i];
   else if (i > (feedlayer-eps))
      dx[i+140] = ((xtemp[i+140]/xtemp[i+160])*(-Jflow[i+1]-Js[i+1])+(xtemp[i-1+140]/xtemp[i-
1+160])*(Jflow[i]+Js[i]))/h +reac15[i];
   else
```

```
        dx[i+140] = ((xtemp[i+140]/xtemp[i+160])*(-Jflow[i]-Jflow[i+1]-Js[i+1])+(xtemp[i-
1+140]/xtemp[i-1+160])*Js[i]+v_in*u[14])/h+reac15[i];
}

// X_A
        dx[150] = ((x[150]/x[160])*(-Jflow[0]-Js[1])+(x[151]/x[161])*Jflow[1])/h +reac16[0];
for (i = 1; i < 10; i++) {
    if (i < (feedlayer-1-eps))
        dx[i+150] = ((xtemp[i+150]/xtemp[i+160])*(-Jflow[i]-Js[i+1])+(xtemp[i-1+150]/xtemp[i-
1+160])*Js[i]+(xtemp[i+1+150]/xtemp[i+1+160])*Jflow[i+1])/h+reac16[i];
    else if (i > (feedlayer-eps))
        dx[i+150] = ((xtemp[i+150]/xtemp[i+160])*(-Jflow[i+1]-Js[i+1])+(xtemp[i-1+150]/xtemp[i-
1+160])*(Jflow[i]+Js[i]))/h+reac16[i];
    else
        dx[i+150] = ((xtemp[i+150]/xtemp[i+160])*(-Jflow[i]-Jflow[i+1]-Js[i+1])+(xtemp[i-
1+150]/xtemp[i-1+160])*Js[i]+v_in*u[15])/h+reac16[i];
}

// X_TSS
        dx[160] = ((x[160]/x[160])*(-Jflow[0]-Js[1])+(x[161]/x[161])*Jflow[1])/h +reac17[0];
for (i = 1; i < 10; i++) {
    if (i < (feedlayer-1-eps))
        dx[i+160] = ((xtemp[i+160]/xtemp[i+160])*(-Jflow[i]-Js[i+1])+(xtemp[i-1+160]/xtemp[i-
1+160])*Js[i]+(xtemp[i+1+160]/xtemp[i+1+160])*Jflow[i+1])/h+reac17[i];
    else if (i > (feedlayer-eps))
        dx[i+160] = ((xtemp[i+160]/xtemp[i+160])*(-Jflow[i+1]-Js[i+1])+(xtemp[i-1+160]/xtemp[i-
1+160])*(Jflow[i]+Js[i]))/h+reac17[i];
    else
        dx[i+160] = ((xtemp[i+160]/xtemp[i+160])*(-Jflow[i]-Jflow[i+1]-Js[i+1])+(xtemp[i-
1+160]/xtemp[i-1+160])*Js[i]+v_in*u[16])/h+reac17[i];
}


// Temperature
for (i = 0; i < 10; i++) {
    if (i < (feedlayer-1-eps))
        dx[i+170]  = (-v_up*xtemp[i+170]+v_up*xtemp[i+1+170])/h;
    else if (i > (feedlayer-eps))
        dx[i+170]  = (v_dn*xtemp[i-1+170]-v_dn*xtemp[i+170])/h;
    else
        dx[i+170]  = (v_in*u[18]-v_up*xtemp[i+170]-v_dn*xtemp[i+170])/h;
}

// S1
for (i = 0; i < 10; i++) {
    if (i < (feedlayer-1-eps))
        dx[i+180]  = (-v_up*xtemp[i+180]+v_up*xtemp[i+1+180])/h+reac18[i];
    else if (i > (feedlayer-eps))
        dx[i+180]  = (v_dn*xtemp[i-1+180]-v_dn*xtemp[i+180])/h+reac18[i];
    else
        dx[i+180]  = (v_in*u[19]-v_up*xtemp[i+180]-v_dn*xtemp[i+180])/h+reac18[i];
}

// S2
for (i = 0; i < 10; i++) {
    if (i < (feedlayer-1-eps))
        dx[i+190]  = (-v_up*xtemp[i+190]+v_up*xtemp[i+1+190])/h+reac19[i];
    else if (i > (feedlayer-eps))
        dx[i+190]  = (v_dn*xtemp[i-1+190]-v_dn*xtemp[i+190])/h+reac19[i];
    else
        dx[i+190]  = (v_in*u[20]-v_up*xtemp[i+190]-v_dn*xtemp[i+190])/h+reac19[i];
}

// S3
for (i = 0; i < 10; i++) {
    if (i < (feedlayer-1-eps))
        dx[i+200]  = (-v_up*xtemp[i+200]+v_up*xtemp[i+1+200])/h+reac20[i];
    else if (i > (feedlayer-eps))
        dx[i+200]  = (v_dn*xtemp[i-1+200]-v_dn*xtemp[i+200])/h+reac20[i];
    else
        dx[i+200]  = (v_in*u[21]-v_up*xtemp[i+200]-v_dn*xtemp[i+200])/h+reac20[i];
}


// X4
        dx[210] = ((x[210]/x[160])*(-Jflow[0]-Js[1])+(x[211]/x[161])*Jflow[1])/h+reac21[0];
for (i = 1; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+210] = ((xtemp[i+210]/xtemp[i+160])*(-Jflow[i]-Js[i+1])+(xtemp[i-1+210]/xtemp[i-
1+160])*Js[i]+(xtemp[i+1+210]/xtemp[i+1+160])*Jflow[i+1])/h+reac21[i];
   else if (i > (feedlayer-eps))
      dx[i+210] = ((xtemp[i+210]/xtemp[i+160])*(-Jflow[i+1]-Js[i+1])+(xtemp[i-1+210]/xtemp[i-
1+160])*(Jflow[i]+Js[i]))/h+reac21[i];
   else
      dx[i+210] = ((xtemp[i+210]/xtemp[i+160])*(-Jflow[i]-Jflow[i+1]-Js[i+1])+(xtemp[i-
1+210]/xtemp[i-1+160])*Js[i]+v_in*u[22])/h+reac21[i];
}

// X5
      dx[220] = ((x[220]/x[160])*(-Jflow[0]-Js[1])+(x[221]/x[161])*Jflow[1])/h+reac22[0];
for (i = 1; i < 10; i++) {
   if (i < (feedlayer-1-eps))
      dx[i+220] = ((xtemp[i+220]/xtemp[i+160])*(-Jflow[i]-Js[i+1])+(xtemp[i-1+220]/xtemp[i-
1+160])*Js[i]+(xtemp[i+1+220]/xtemp[i+1+160])*Jflow[i+1])/h+reac22[i];
   else if (i > (feedlayer-eps))
      dx[i+220] = ((xtemp[i+220]/xtemp[i+160])*(-Jflow[i+1]-Js[i+1])+(xtemp[i-1+220]/xtemp[i-
1+160])*(Jflow[i]+Js[i]))/h+reac22[i];
   else
      dx[i+220] = ((xtemp[i+220]/xtemp[i+160])*(-Jflow[i]-Jflow[i+1]-Js[i+1])+(xtemp[i-
1+220]/xtemp[i-1+160])*Js[i]+v_in*u[23])/h+reac22[i];
}

}

// mdlTerminate - called when the simulation is terminated.

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
```

A8. Evaluation script for BSM1_ASM3 bioP

```
starttime    = 7;
stoptime     = 14;
startindex   = max(find(t <= starttime));
stopindex    = min(find(t >= stoptime));

time         = t(startindex:stopindex);

sampletime   = time(2)   - time(1);
totalt       = time(end) - time(1);

% -------------------------------------------------------------------------

totalCODemax= 100;
totalNemax   = 18;
SNHemax      = 4;
TSSemax      = 30;
BOD5emax     = 10;
totalPemax   = 2;               % ASM2d specific

BSS          = 2;
BCOD         = 1;
BNKj         = 20;              % original BSM1
BNO          = 20;              % original BSM1
BBOD5        = 2;
BNKj_new     = 30;              % updated BSM TG meeting no 8
BNO_new      = 10;              % updated BSM TG meeting no 8

BPorg        = 100;             % new value (old value = 20)
BPinorg      = 100;             % new value (old value = 20)

pumpfactor   = 0.04;            % original BSM1, same for all pumped flows
PF_Qintr     = 0.004;           % kWh/m3, pumping energy factor, internal AS recirculation
PF_Qr        = 0.008;           % kWh/m3, pumping energy factor, AS sludge recycle
PF_Qw        = 0.05;            % kWh/m3, pumping energy factor, AS wastage flow

% -------------------------------------------------------------------------
% Cut out the parts of the files to be used

inpart       = in(startindex:(stopindex-1),:);
reac1part    = reac1(startindex:(stopindex-1),:);
reac2part    = reac2(startindex:(stopindex-1),:);
reac3part    = reac3(startindex:(stopindex-1),:);
reac4part    = reac4(startindex:(stopindex-1),:);
reac5part    = reac5(startindex:(stopindex-1),:);
reac6part    = reac6(startindex:(stopindex-1),:);
reac7part    = reac7(startindex:(stopindex-1),:);

settlerpart = settler(startindex:(stopindex-1),:);
recpart      = rec(startindex:(stopindex-1),:);

%% Effluent concentrations

timevector   = time(2:end) - time(1:(end-1));

Qevec        = settlerpart(:,43).*timevector;        % effluent flow rate of settler
Qinvec       = inpart(:,18).*timevector;             % influent flow rate

SO2evec      = settlerpart(:,26).*Qevec;             % SO2 in settler effluent
SSevec       = settlerpart(:,27).*Qevec;             % SS in settler effluent
SIevec       = settlerpart(:,28).*Qevec;             % SI in settler effluent
SNHevec      = settlerpart(:,29).*Qevec;             % SNH in settler effluent
SNOevec      = settlerpart(:,30).*Qevec;             % SNO in settler effluent
SN2evec      = settlerpart(:,31).*Qevec;             % SN2 in settler effluent
SPO4evec     = settlerpart(:,32).*Qevec;             % SPO4 in settler effluent
SHCOevec     = settlerpart(:,33).*Qevec;             % SHCO in settler effluent
XIevec       = settlerpart(:,34).*Qevec;             % XI in settler effluent
XSevec       = settlerpart(:,35).*Qevec;             % XS in settler effluent
XHevec       = settlerpart(:,36).*Qevec;             % XH in settler effluent
XSTOevec     = settlerpart(:,37).*Qevec;             % XSTO in settler effluent
XPAOevec     = settlerpart(:,38).*Qevec;             % XPAO in settler effluent
XPPevec      = settlerpart(:,39).*Qevec;             % XPP in settler effluent
XPHAevec     = settlerpart(:,40).*Qevec;             % XPHA in settler effluent
XAevec       = settlerpart(:,41).*Qevec;             % XA in settler effluent
XTSSevec     = settlerpart(:,42).*Qevec;             % XTSS in settler effluent
```

```
Qetot       = sum(Qevec);
Qeav        = Qetot/totalt;

SO2eload    = sum(SO2evec);
SSeload     = sum(SSevec);
SIeload     = sum(SIevec);
SNHeload    = sum(SNHevec);
SNOeload    = sum(SNOevec);
SN2eload    = sum(SN2evec);
SPO4eload   = sum(SPO4evec);
SHCOeload   = sum(SHCOevec);
XIeload     = sum(XIevec);
XSeload     = sum(XSevec);
XHeload     = sum(XHevec);
XSTOeload   = sum(XSTOevec);
XPAOeload   = sum(XPAOevec);
XPPeload    = sum(XPPevec);
XPHAeload   = sum(XPHAevec);
XAeload     = sum(XAevec);
XTSSeload   = sum(XTSSevec);


SO2eav      = SO2eload/Qetot;
SSeav       = SSeload/Qetot;
SIeav       = SIeload/Qetot;
SNHeav      = SNHeload/Qetot;
SNOeav      = SNOeload/Qetot;
SN2eav      = SN2eload/Qetot;
SPO4eav     = SPO4eload/Qetot;
SHCOeav     = SHCOeload/Qetot;
XIeav       = XIeload/Qetot;
XSeav       = XSeload/Qetot;
XHeav       = XHeload/Qetot;
XSTOeav     = XSTOeload/Qetot;
XPAOeav     = XPAOeload/Qetot;
XPPeav      = XPPeload/Qetot;
XPHAeav     = XPHAeload/Qetot;
XAeav       = XAeload/Qetot;
XTSSeav     = XTSSeload/Qetot;


totalNKjevec2   = (iN_SS*SSevec + iN_SI*SIevec + SNHevec + iN_XI*XIevec + iN_XS*XSevec +
iN_BM*(XHevec+XPAOevec+XAevec))./Qevec;
totalNevec2     = (SNOevec + SNHevec + iN_SS*SSevec + iN_SI*SIevec + iN_XI*XIevec +
iN_XS*XSevec + iN_BM*(XHevec+XPAOevec+XAevec))./Qevec;          % SN2 not included
totalPorgevec2  = (iP_SS*SSevec + iP_SI*SIevec + iP_XI*XIevec + iP_XS*XSevec + XPPevec+
iP_BM*(XHevec+XPAOevec+XAevec))./Qevec;
totalPevec2     = (iP_SS*SSevec + iP_SI*SIevec + SPO4evec + XPPevec + iP_XI*XIevec +
iP_XS*XSevec + iP_BM*(XHevec+XPAOevec+XAevec))./Qevec;
totalCODevec2   = (SSevec + SIevec + XIevec + XSevec + XHevec +  XSTOevec + XPAOevec +
XPHAevec + XAevec)./Qevec;
SNHevec2        = SNHevec./Qevec;
SPO4evec2       = SPO4evec./Qevec;
XTSSevec2       = XTSSevec./Qevec;
BOD5evec2       = (0.25*(SSevec + (1-f_SI)*XSevec + (1-f_XI)*(XHevec + XPAOevec + XPHAevec +
XAevec)))./Qevec;

totalNKjeload   = iN_SS*SSeload + iN_SI*SIeload + SNHeload +  iN_XI*XIeload + iN_XS*XSeload +
iN_BM*(XHeload+XPAOeload+XAeload);
totalNeload     = SNOeload + totalNKjeload;
% SN2 not included
totalPorgeload  = (iP_SS*SSeload +  iP_SI*SIeload + iP_XI*XIeload + iP_XS*XSeload + XPPeload +
iP_BM*(XHeload+XPAOeload+XAeload));
totalPeload     = SPO4eload + totalPorgeload;
totalCODeload   = (SSeload + SIeload + XIeload + XSeload + XHeload + XSTOeload + XPAOeload +
XPHAeload + XAeload);
BOD5eload       = (0.25*(SSeload + (1-f_SI)*XSeload + (1-f_XI)*(XHeload + XPAOeload +
XPHAeload + XAeload)));


%% Influent and Effluent Quality Index

SSin        = inpart(:,17);             % Influent suspended solids
CODin       = inpart(:,2) + inpart(:,3) + inpart(:,9) + inpart(:,10) + inpart(:,11) +
inpart(:,13) + inpart(:,15) + inpart(:,16);
SNKjin      = inpart(:,4) + iN_SS*inpart(:,2) + iN_SI*inpart(:,3) + iN_XI*inpart(:,9) +
iN_XS*inpart(:,10) + iN_BM*(inpart(:,11) + inpart(:,15) + inpart(:,16));
```

```
SNOXin       = inpart(:,5);                % Influent nitrate
SPorgin      = inpart(:,14) + iP_SS*inpart(:,2) + iP_SI*inpart(:,3) + iP_XI*inpart(:,9) +
iP_XS*inpart(:,10) + iP_BM*(inpart(:,11) + inpart(:,13) + inpart(:,16));
SPinorgin    = inpart(:,7);                % Influent inorganic P
BOD5in       = 0.65*(inpart(:,2) + (1-f_SI)*inpart(:,10) + (1-f_XI)*(inpart(:,11) +
inpart(:,13) + inpart(:,15) + inpart(:,16)));

SSe          = settlerpart(:,42);          % Effluent suspended solids
CODe         = settlerpart(:,27) + settlerpart(:,28) + settlerpart(:,34) + settlerpart(:,35) +
settlerpart(:,36) + settlerpart(:,38) + settlerpart(:,40) + settlerpart(:,41);
SNKje        = settlerpart(:,29) + iN_SS*settlerpart(:,27) + iN_SI*settlerpart(:,28) +
iN_XI*settlerpart(:,34) + iN_XS*settlerpart(:,35) + iN_BM*(settlerpart(:,36) +
settlerpart(:,38) + settlerpart(:,41));
SNOXe        = settlerpart(:,30);
SPorge       = settlerpart(:,39) + iP_SS*settlerpart(:,27) + iP_SI*settlerpart(:,28) +
iP_XI*settlerpart(:,34) + iP_XS*settlerpart(:,35) + iP_BM*(settlerpart(:,36) +
settlerpart(:,38) + settlerpart(:,41));
SPinorge     = settlerpart(:,32);          % Effluent inorganic P
BOD5e        = 0.25*(settlerpart(:,27) + (1-f_SI)*settlerpart(:,35) + (1-
f_XI)*(settlerpart(:,36) + settlerpart(:,38) + settlerpart(:,40) + settlerpart(:,41)));


EQvecinst        = (BSS*SSe + BCOD*CODe + BNKj*SNKje + BNO*SNOXe +
BBOD5*BOD5e).*settlerpart(:,43);
EQvecinst_new    = (BSS*SSe + BCOD*CODe + BNKj_new*SNKje + BNO_new*SNOXe +
BBOD5*BOD5e).*settlerpart(:,43);                                    % updated BSM TG
meeting no 8
EQvecinst_P      = (BSS*SSe + BCOD*CODe + BNKj_new*SNKje + BNO_new*SNOXe + BBOD5*BOD5e +
BPorg*SPorge + BPinorg*SPinorge).*settlerpart(:,43);        % ASM2d

IQvec        = (BSS*SSin + BCOD*CODin + BNKj*SNKjin + BNO*SNOXin + BBOD5*BOD5in).*Qinvec;
IQvec_new    = (BSS*SSin + BCOD*CODin + BNKj_new*SNKjin + BNO_new*SNOXin +
BBOD5*BOD5in).*Qinvec;                                              % updated BSM TG
meeting no 8
IQvec_P      = (BSS*SSin + BCOD*CODin + BNKj_new*SNKjin + BNO_new*SNOXin + BBOD5*BOD5in +
BPorg*SPorgin + BPinorg*SPinorgin).*Qinvec;                 % ASM2d

IQ           = sum(IQvec)/(totalt*1000);
IQ_new       = sum(IQvec_new)/(totalt*1000);
IQ_P         = sum(IQvec_P)/(totalt*1000);

EQvec        = (BSS*SSe + BCOD*CODe + BNKj*SNKje + BNO*SNOXe + BBOD5*BOD5e).*Qevec;
EQvec_new    = (BSS*SSe + BCOD*CODe + BNKj_new*SNKje + BNO_new*SNOXe + BBOD5*BOD5e).*Qevec;
% updated BSM TG meeting no 8
EQvec_P      = (BSS*SSe + BCOD*CODe + BNKj_new*SNKje + BNO_new*SNOXe + BBOD5*BOD5e +
BPorg*SPorge + BPinorg*SPinorge).*Qevec;

EQ           = sum(EQvec)/(totalt*1000);
EQ_new       = sum(EQvec_new)/(totalt*1000);
% updated BSM TG meeting no 8
EQ_P         = sum(EQvec_P)/(totalt*1000);
% ASM2d


%% Cost factors for operation

% Sludge production
TSSreactors_start   = (reac1part(1,17)*VOL1     + reac2part(1,17)*VOL2    +
reac3part(1,17)*VOL3    + reac4part(1,17)*VOL4      + reac5part(1,17)*VOL5      +
reac6part(1,17)*VOL6      + reac7part(1,17)*VOL7)/1000;
TSSreactors_end     = (reac1part(end,17)*VOL1    + reac2part(end,17)*VOL2  +
reac3part(end,17)*VOL3  + reac4part(end,17)*VOL4     + reac5part(end,17)*VOL5    +
reac6part(end,17)*VOL6    + reac7part(end,17)*VOL7)/1000;

TSSsettler_start    = (settlerpart(1,50)*DIM3_bioP(1)*DIM3_bioP(2)/10   +
settlerpart(1,51)*DIM3_bioP(1)*DIM3_bioP(2)/10    +
settlerpart(1,52)*DIM3_bioP(1)*DIM3_bioP(2)/10    +
settlerpart(1,53)*DIM3_bioP(1)*DIM3_bioP(2)/10    +
settlerpart(1,54)*DIM3_bioP(1)*DIM3_bioP(2)/10    +
settlerpart(1,55)*DIM3_bioP(1)*DIM3_bioP(2)/10    +
settlerpart(1,56)*DIM3_bioP(1)*DIM3_bioP(2)/10    +
settlerpart(1,57)*DIM3_bioP(1)*DIM3_bioP(2)/10    +
settlerpart(1,58)*DIM3_bioP(1)*DIM3_bioP(2)/10    +
settlerpart(1,59)*DIM3_bioP(1)*DIM3_bioP(2)/10)/1000;
TSSsettler_end      = (settlerpart(end,50)*DIM3_bioP(1)*DIM3_bioP(2)/10 +
settlerpart(end,51)*DIM3_bioP(1)*DIM3_bioP(2)/10  +
settlerpart(end,52)*DIM3_bioP(1)*DIM3_bioP(2)/10  +
settlerpart(end,53)*DIM3_bioP(1)*DIM3_bioP(2)/10  +
settlerpart(end,54)*DIM3_bioP(1)*DIM3_bioP(2)/10  +
settlerpart(end,55)*DIM3_bioP(1)*DIM3_bioP(2)/10  +
settlerpart(end,56)*DIM3_bioP(1)*DIM3_bioP(2)/10  +
settlerpart(end,57)*DIM3_bioP(1)*DIM3_bioP(2)/10  +
settlerpart(end,58)*DIM3_bioP(1)*DIM3_bioP(2)/10  +
settlerpart(end,59)*DIM3_bioP(1)*DIM3_bioP(2)/10)/1000;

TSSwasteconc        = settlerpart(:,59)/1000;       % kg/m3
Qwasteflow          = settlerpart(:,25);            % m3/d
TSSuvec             = TSSwasteconc.*Qwasteflow.*timevector;
TSSproduced         = sum(TSSuvec) + TSSreactors_end + TSSsettler_end - TSSreactors_start -
TSSsettler_start;
TSSproducedperd     = TSSproduced/totalt;           % for OCI

% Aeration energy, original BSM1
kla1vec = kla1in(startindex:(stopindex-1),:);
kla2vec = kla2in(startindex:(stopindex-1),:);
kla3vec = kla3in(startindex:(stopindex-1),:);
kla4vec = kla4in(startindex:(stopindex-1),:);
kla5vec = kla5in(startindex:(stopindex-1),:);
kla6vec = kla6in(startindex:(stopindex-1),:);
kla7vec = kla7in(startindex:(stopindex-1),:);

kla1newvec = 0.0007*(VOL1/1333)*(kla1vec.*kla1vec)+0.3267*(VOL1/1333)*kla1vec;
kla2newvec = 0.0007*(VOL2/1333)*(kla2vec.*kla2vec)+0.3267*(VOL2/1333)*kla2vec;
kla3newvec = 0.0007*(VOL3/1333)*(kla3vec.*kla3vec)+0.3267*(VOL3/1333)*kla3vec;
kla4newvec = 0.0007*(VOL4/1333)*(kla4vec.*kla4vec)+0.3267*(VOL4/1333)*kla4vec;
kla5newvec = 0.0007*(VOL5/1333)*(kla5vec.*kla5vec)+0.3267*(VOL5/1333)*kla5vec;
kla6newvec = 0.0007*(VOL6/1333)*(kla6vec.*kla6vec)+0.3267*(VOL6/1333)*kla6vec;
kla7newvec = 0.0007*(VOL7/1333)*(kla7vec.*kla7vec)+0.3267*(VOL7/1333)*kla7vec;

airenergyvec    = 24*(kla1newvec + kla2newvec + kla3newvec + kla4newvec + kla5newvec +
kla6newvec + kla7newvec);
airenergy       = sum(airenergyvec.*timevector);
airenergyperd   = airenergy/totalt; % for OCI

% Aeration energy, updated BSM1 (and also for BSM2)
kla1newvec_new = SOSAT1*VOL1*kla1vec;
kla2newvec_new = SOSAT2*VOL2*kla2vec;
kla3newvec_new = SOSAT3*VOL3*kla3vec;
kla4newvec_new = SOSAT4*VOL4*kla4vec;
kla5newvec_new = SOSAT5*VOL5*kla5vec;
kla6newvec_new = SOSAT6*VOL6*kla6vec;
kla7newvec_new = SOSAT7*VOL7*kla7vec;

airenergyvec_new    = (kla1newvec_new + kla2newvec_new + kla3newvec_new + kla4newvec_new +
kla5newvec_new + kla6newvec_new + kla7newvec_new)/(1.8*1000);
airenergy_new       = sum(airenergyvec_new.*timevector);
airenergy_newperd   = airenergy_new/totalt; % for OCI

% Mixing energy (calculated as kWh consumed for the complete evaluation period), same as for
BSM2
mixnumreac1 = length(find(kla1vec<20));
mixnumreac2 = length(find(kla2vec<20));
mixnumreac3 = length(find(kla3vec<20));
mixnumreac4 = length(find(kla4vec<20));
mixnumreac5 = length(find(kla5vec<20));
mixnumreac6 = length(find(kla6vec<20));
mixnumreac7 = length(find(kla7vec<20));

mixenergyunitreac = 0.005; % kW/m3

mixenergyreac1 = mixnumreac1*mixenergyunitreac*VOL1;
mixenergyreac2 = mixnumreac2*mixenergyunitreac*VOL2;
mixenergyreac3 = mixnumreac3*mixenergyunitreac*VOL3;
mixenergyreac4 = mixnumreac4*mixenergyunitreac*VOL4;
mixenergyreac5 = mixnumreac5*mixenergyunitreac*VOL5;
mixenergyreac6 = mixnumreac6*mixenergyunitreac*VOL6;
mixenergyreac7 = mixnumreac7*mixenergyunitreac*VOL7;

mixenergy       = 24*(mixenergyreac1 + mixenergyreac2 + mixenergyreac3 + mixenergyreac4 +
mixenergyreac5 + mixenergyreac6 + mixenergyreac7)*sampletime;
mixenergyperd   = mixenergy/totalt; % for OCI

% Pumping energy, original BSM1
Qintrflow       = recpart(:,18);
```

```
Qrflow           = settlerpart(:,18);
pumpenergyvec    = pumpfactor*(Qwasteflow + Qintrflow + Qrflow);
pumpenergy       = sum(pumpenergyvec.*timevector);
pumpenergyperd   = pumpenergy/totalt; % for OCI

% Pumping energy (based on BSM2 principles)
Qintrflow_new        = recpart(:,18);
Qrflow_new           = settlerpart(:,18);
Qwflow_new           = settlerpart(:,25);
pumpenergyvec_new    = PF_Qintr*Qintrflow_new + PF_Qr*Qrflow_new + PF_Qw*Qwflow_new;
pumpenergy_new       = sum(pumpenergyvec_new.*timevector);
pumpenergy_newperd   = pumpenergy_new/totalt; % for OCI

% Carbon source addition
carbon1vec = carbon1in(startindex:(stopindex-1),:);
carbon2vec = carbon2in(startindex:(stopindex-1),:);
carbon3vec = carbon3in(startindex:(stopindex-1),:);
carbon4vec = carbon4in(startindex:(stopindex-1),:);
carbon5vec = carbon5in(startindex:(stopindex-1),:);
carbon6vec = carbon6in(startindex:(stopindex-1),:);
carbon7vec = carbon7in(startindex:(stopindex-1),:);

Qcarbonvec       = (carbon1vec + carbon2vec + carbon3vec + carbon4vec + carbon5vec + carbon6vec
+ carbon7vec);
carbonmassvec    = Qcarbonvec*CARBONSOURCECONC/1000;
Qcarbon          = sum(Qcarbonvec.*timevector); % m3
carbonmass       = sum(carbonmassvec.*timevector); % kg COD
carbonmassperd   = carbonmass/totalt; % for OCI


% Operational Cost Index for BSM1
TSScost              = 5*TSSproducedperd;
airenergycost        = 1*airenergyperd;      % original BSM1
airenergy_newcost    = 1*airenergy_newperd;  % updated BSM1
mixenergycost        = 1*mixenergyperd;      % based on BSM2
pumpenergycost       = 1*pumpenergyperd;     % original BSM1
pumpenergy_newcost   = 1*pumpenergy_newperd; % based on BSM2
carbonmasscost       = 3*carbonmassperd;
% metalmasscost        = 1.5*metalmassperd;

OCI     = TSScost + airenergycost + mixenergycost + pumpenergycost + carbonmasscost;
OCI_new = TSScost + airenergy_newcost + mixenergycost + pumpenergy_newcost + carbonmasscost;


%% Calculate 95% percentiles for effluent SNH, TN, and TSS (BSM2 standard criteria)

SNHeffprctile   = prctile(SNHevec2,95);
SPO4effprctile  = prctile(SPO4evec2,95);
TNeffprctile    = prctile(totalNevec2,95);
TPeffprctile    = prctile(totalPevec2,95);
TSSeffprctile   = prctile(XTSSevec2,95);


disp(' ')
disp(['Overall plant performance during time ',num2str(time(1)),' to ',num2str(time(end)),'
days'])
disp('**************************************************')
disp(' ')
disp('Effluent average concentrations based on load')
disp('---------------------------------------------')
disp(['Effluent average flow rate  = ',num2str(Qeav),' m3/d'])
disp(['Effluent average SO2 conc   = ',num2str(SO2eav),' mg (-COD)/l'])
disp(['Effluent average SS conc    = ',num2str(SSeav),' mg COD/l'])
disp(['Effluent average SI conc    = ',num2str(SIeav),' mg COD/l'])
disp(['Effluent average SNH conc   = ',num2str(SNHeav),' mg N/l'])
disp(['Effluent average SNO conc   = ',num2str(SNOeav),' mg N/l'])
disp(['Effluent average SN2 conc   = ',num2str(SN2eav),' mg N/l'])
disp(['Effluent average SPO4 conc  = ',num2str(SPO4eav),' mg P/l'])
disp(['Effluent average SHCO conc  = ',num2str(SHCOeav),' mol HCO3/m3'])
disp(['Effluent average XI conc    = ',num2str(XIeav),' mg COD/l'])
disp(['Effluent average XS conc    = ',num2str(XSeav),' mg COD/l'])
disp(['Effluent average XH conc    = ',num2str(XHeav),' mg COD/l'])
disp(['Effluent average XSTO conc  = ',num2str(XSTOeav),' mg COD/l'])
disp(['Effluent average XPAO conc  = ',num2str(XPAOeav),' mg COD/l'])
disp(['Effluent average XPP conc   = ',num2str(XPPeav),' mg P/l'])
disp(['Effluent average XPHA conc  = ',num2str(XPHAeav),' mg COD/l'])
disp(['Effluent average XA conc    = ',num2str(XAeav),' mg COD/l'])
disp(['Effluent average XTSS conc  = ',num2str(XTSSeav),' mg SS/l  (limit = 35 mg SS/l)'])
disp(' ')
disp(['Effluent average Kjeldahl N conc = ',num2str(SNHeav + iN_SS*SSeav + iN_SI*SIeav +
iN_XI*XIeav + iN_XS*XSeav + iN_BM*(XHeav + XPAOeav +  XAeav)),' mg N/l'])
disp(['Effluent average total N conc     = ',num2str(SNOeav + SNHeav + iN_SS*SSeav +
iN_SI*SIeav + iN_XI*XIeav + iN_XS*XSeav + iN_BM*(XHeav + XPAOeav + XAeav)),' mg N/l  (limit =
15 mg N/l)'])
disp(['Effluent average total P conc     = ',num2str(SPO4eav + XPPeav + iP_SS*SSeav +
iP_SI*SIeav + iP_XI*XIeav + iP_XS*XSeav + iP_BM*(XHeav + XPAOeav + XAeav)),' mg P/l  (limit =
2 mg P/l)'])
disp(['Effluent average total COD conc  = ',num2str(SSeav + SIeav + XIeav + XSeav + XHeav +
XPAOeav + XPHAeav + XAeav),' mg COD/l  (limit = 125 mg COD/l)'])
disp(['Effluent average BOD5 conc        = ',num2str(0.25*(SSeav + (1-f_SI)*XSeav + (1-
f_XI)*(XHeav + XPAOeav + XPHAeav + XAeav))),' mg/l  (limit = 25 mg/l)'])
disp(' ')

disp('Effluent average load')
disp('---------------------')
disp(['Effluent average SO2 load   = ',num2str(SO2eload/(1000*totalt)),' kg (-COD)/day'])
disp(['Effluent average SS load    = ',num2str(SSeload/(1000*totalt)),' kg COD/day'])
disp(['Effluent average SI load    = ',num2str(SIeload/(1000*totalt)),' kg COD/day'])
disp(['Effluent average SNH load   = ',num2str(SNHeload/(1000*totalt)),' kg N/day'])
disp(['Effluent average SNO load   = ',num2str(SNOeload/(1000*totalt)),' kg N/day'])
disp(['Effluent average SN2 load   = ',num2str(SN2eload/(1000*totalt)),' kg N/day'])
disp(['Effluent average SPO4 load  = ',num2str(SPO4eload/(1000*totalt)),' kg P/day'])
disp(['Effluent average SHCO load  = ',num2str(SHCOeload/(1000*totalt)),' kmol HCO3/day'])
disp(['Effluent average XI load    = ',num2str(XIeload/(1000*totalt)),' kg COD/day'])
disp(['Effluent average XS load    = ',num2str(XSeload/(1000*totalt)),' kg COD/day'])
disp(['Effluent average XH load    = ',num2str(XHeload/(1000*totalt)),' kg COD/day'])
disp(['Effluent average XSTO load  = ',num2str(XSTOeload/(1000*totalt)),' kg COD/day'])
disp(['Effluent average XPAO load  = ',num2str(XPAOeload/(1000*totalt)),' kg COD/day'])
disp(['Effluent average XPP load   = ',num2str(XPPeload/(1000*totalt)),' kg P/day'])
disp(['Effluent average XPHA load  = ',num2str(XPHAeload/(1000*totalt)),' kg COD/day'])
disp(['Effluent average XA load    = ',num2str(XAeload/(1000*totalt)),' kg COD/day'])
disp(['Effluent average XTSS load  = ',num2str(XTSSeload/(1000*totalt)),' kg SS/day'])

disp(' ')
disp(['Effluent average Kjeldahl N load = ',num2str(totalNKjeload/(1000*totalt)),' kg N/d'])
disp(['Effluent average total N load     = ',num2str(totalNeload/(1000*totalt)),' kg N/d'])
disp(['Effluent average organic P load  = ',num2str(totalPorgeload/(1000*totalt)),' kg P/d'])
disp(['Effluent average total P load     = ',num2str(totalPeload/(1000*totalt)),' kg P/d'])
disp(['Effluent average total COD load  = ',num2str(totalCODeload/(1000*totalt)),' kg COD/d'])
disp(['Effluent average BOD5 load        = ',num2str(BOD5eload/(1000*totalt)),' kg/d'])

disp(' ')
disp('Other effluent quality variables')
disp('--------------------------------')
disp(['Influent Quality (I.Q.) index = ',num2str(IQ),' kg poll.units/d (original BSM1
version)'])
disp(['Effluent Quality (E.Q.) index = ',num2str(EQ),' kg poll.units/d (original BSM1
version)'])
disp(['Influent Quality (I.Q.) index = ',num2str(IQ_new),' kg poll.units/d (updated BSM1
version)'])
disp(['Effluent Quality (E.Q.) index = ',num2str(EQ_new),' kg poll.units/d (updated BSM1
version)'])
disp(['Influent Quality (I.Q.) index = ',num2str(IQ_P),' kg poll.units/d (updated ASM3 bioP
version)'])
disp(['Effluent Quality (E.Q.) index = ',num2str(EQ_P),' kg poll.units/d (updated ASM3 bioP
version)'])

disp(' ')
disp(['Sludge production for disposal = ',num2str(TSSproduced),' kg SS'])
disp(['Average sludge production for disposal per day = ',num2str(TSSproduced/totalt),' kg
SS/d'])
disp(['Sludge production released into effluent = ',num2str(XTSSeload /1000),' kg SS'])
disp(['Average sludge production released into effluent per day = ',num2str(XTSSeload
/(1000*totalt)),' kg SS/d'])
disp(['Total sludge production = ',num2str(TSSproduced+XTSSeload /1000),' kg SS'])
disp(['Total average sludge production per day =
',num2str(TSSproduced/totalt+XTSSeload/(1000*totalt)),' kg SS/d'])

disp(' ')
disp(['Total aeration energy = ',num2str(airenergy),' kWh (original BSM1 version)'])
disp(['Average aeration energy per day = ',num2str(airenergy/totalt),' kWh/d (original BSM1
version)'])
disp(['Total aeration energy = ',num2str(airenergy_new),' kWh (updated BSM1 version)'])
disp(['Average aeration energy per day = ',num2str(airenergy_new/totalt),' kWh/d (updated BSM1
version)'])
disp(' ')
disp(['Total pumping energy (for Qintr, Qr and Qw) = ',num2str(pumpenergy),' kWh (original
BSM1 version)'])
disp(['Average pumping energy per day (for Qintr, Qr and Qw) = ',num2str(pumpenergy/totalt),'
kWh/d (original BSM1 version)'])
disp(['Total pumping energy (for Qintr, Qr and Qw) = ',num2str(pumpenergy_new),' kWh (based on
BSM2 principles)'])
disp(['Average pumping energy per day (for Qintr, Qr and Qw) =
',num2str(pumpenergy_new/totalt),' kWh/d (based on BSM2 principles)'])
disp(' ')
disp(['Total mixing energy = ',num2str(mixenergy),' kWh (based on BSM2 principles)'])
disp(['Average mixing energy per day = ',num2str(mixenergy/totalt),' kWh/d (based on BSM2
principles)'])
disp(' ')
disp(['Total added carbon volume = ',num2str(Qcarbon),' m3'])
disp(['Average added carbon flow rate = ',num2str(Qcarbon/totalt),' m3/d'])
disp(['Total added carbon mass = ',num2str(carbonmass),' kg COD'])
disp(['Average added carbon mass per day = ',num2str(carbonmass/totalt),' kg COD/d'])
disp(' ')
% disp(['Total added metal volume = ',num2str(Qmetal),' m3'])
% disp(['Average added metal flow rate = ',num2str(Qmetal/totalt),' m3/d'])
% disp(['Total added metal mass = ',num2str(metalmass),' kg COD'])
% disp(['Average added metal mass per day = ',num2str(metalmass/totalt),' kg COD/d'])
disp(' ')
disp('Operational Cost Index')
disp('----------------------')
disp(['Sludge production cost index = ',num2str(TSScost),' (using weight 5 for BSM1)'])
disp(['Aeration energy cost index = ',num2str(airenergycost),' (original BSM1 version)'])
disp(['Updated aeration energy cost index = ',num2str(airenergy_newcost),' (updated BSM1
version)'])
disp(['Pumping energy cost index = ',num2str(pumpenergycost),' (original BSM1 version)'])
disp(['Updated pumping energy cost index = ',num2str(pumpenergy_newcost),' (based on BSM2
principles)'])
disp(['Carbon source addition cost index = ',num2str(carbonmasscost)])
% disp(['Metal source addition cost index = ',num2str(metalmasscost)])
disp(['Mixing energy cost index = ',num2str(mixenergycost),' (based on BSM2 principles)'])
disp(['Total Operational Cost Index (OCI) = ',num2str(OCI),' (original BSM1 version)'])
disp(['Updated Total Operational Cost Index (OCI) = ',num2str(OCI_new),' (using new aeraration
and pumping costs)'])
disp(' ')
%
Nviolation=find(totalNevec2>totalNemax);
Pviolation=find(totalPevec2>totalPemax);
CODviolation=find(totalCODevec2>totalCODemax);
SNHviolation=find(SNHevec2>SNHemax);
TSSviolation=find(XTSSevec2>TSSemax);
BOD5violation=find(BOD5evec2>BOD5emax);
%
noofNviolation = 1;
noofPviolation = 1;
noofCODviolation = 1;
noofSNHviolation = 1;
noofTSSviolation = 1;
noofBOD5violation = 1;

disp('Effluent violations')
disp('-------------------')
disp(['95% percentile for effluent SNH (Ammonia95) = ',num2str(SNHeffprctile),' g N/m3'])
disp(['95% percentile for effluent SPO4(Phosphate95) = ',num2str(SPO4effprctile),' g N/m3'])
disp(['95% percentile for effluent TN (TN95) = ',num2str(TNeffprctile),' g N/m3'])
disp(['95% percentile for effluent TP (TP95) = ',num2str(TPeffprctile),' g N/m3'])
disp(['95% percentile for effluent TSS (TSS95) = ',num2str(TSSeffprctile),' g SS/m3'])
disp(' ')
%
if not(isempty(Pviolation))
  disp('The maximum effluent total phosphorus level (2 mg N/l) was violated')
  disp(['during ',num2str(min(totalt,length(Pviolation)*sampletime)),' days, i.e.
',num2str(min(100,length(Pviolation)*sampletime/totalt*100)),'% of the operating time.'])
  for i=2:length(Pviolation)
    if Pviolation(i-1)~=(Pviolation(i)-1)
      noofPviolation = noofPviolation+1;
    end
  end
  disp(['The limit was violated at ',num2str(noofPviolation),' different occasions.'])
  disp(' ')
```

```
end

if not(isempty(Nviolation))
  disp('The maximum effluent total nitrogen level (18 mg N/l) was violated')
  disp(['during ',num2str(min(totalt,length(Nviolation)*sampletime)),' days, i.e.
',num2str(min(100,length(Nviolation)*sampletime/totalt*100)),'% of the operating time.'])
  for i=2:length(Nviolation)
    if Nviolation(i-1)~=(Nviolation(i)-1)
      noofNviolation = noofNviolation+1;
    end
  end
  disp(['The limit was violated at ',num2str(noofNviolation),' different occasions.'])
  disp(' ')
end

if not(isempty(CODviolation))
  disp('The maximum effluent total COD level (100 mg COD/l) was violated')
  disp(['during ',num2str(min(totalt,length(CODviolation)*sampletime)),' days, i.e.
',num2str(min(100,length(CODviolation)*sampletime/totalt*100)),'% of the operating time.'])
  for i=2:length(CODviolation)
    if CODviolation(i-1)~=(CODviolation(i)-1)
      noofCODviolation = noofCODviolation+1;
    end
  end
  disp(['The limit was violated at ',num2str(noofCODviolation),' different occasions.'])
  disp(' ')
end

if not(isempty(SNHviolation))
  disp('The maximum effluent ammonia nitrogen level (4 mg N/l) was violated')
  disp(['during ',num2str(min(totalt,length(SNHviolation)*sampletime)),' days, i.e.
',num2str(min(100,length(SNHviolation)*sampletime/totalt*100)),'% of the operating time.'])
  for i=2:length(SNHviolation)
    if SNHviolation(i-1)~=(SNHviolation(i)-1)
      noofSNHviolation = noofSNHviolation+1;
    end
  end
  disp(['The limit was violated at ',num2str(noofSNHviolation),' different occasions.'])
  disp(' ')
end

if not(isempty(TSSviolation))
  disp('The maximum effluent total suspended solids level (30 mg SS/l) was violated')
  disp(['during ',num2str(min(totalt,length(TSSviolation)*sampletime)),' days, i.e.
',num2str(min(100,length(TSSviolation)*sampletime/totalt*100)),'% of the operating time.'])
  for i=2:length(TSSviolation)
    if TSSviolation(i-1)~=(TSSviolation(i)-1)
      noofTSSviolation = noofTSSviolation+1;
    end
  end
  disp(['The limit was violated at ',num2str(noofTSSviolation),' different occasions.'])
  disp(' ')
end

if not(isempty(BOD5violation))
  disp('The maximum effluent BOD5 level (10 mg/l) was violated')
  disp(['during ',num2str(min(totalt,length(BOD5violation)*sampletime)),' days, i.e.
',num2str(min(100,length(BOD5violation)*sampletime/totalt*100)),'% of the operating time.'])
  for i=2:length(BOD5violation)
    if BOD5violation(i-1)~=(BOD5violation(i)-1)
      noofBOD5violation = noofBOD5violation+1;
    end
  end
  disp(['The limit was violated at ',num2str(noofBOD5violation),' different occasions.'])
  disp(' ')
end
```

A9. Evaluation script for BSM1_ASM3

```
starttime = 7;
stoptime = 14;
startindex=max(find(t <= starttime));
stopindex=min(find(t >= stoptime));

time=t(startindex:stopindex);

sampletime = time(2)-time(1);
totalt=time(end)-time(1);

totalCODemax = 100;
totalNemax = 18;
SNHemax = 4;
TSSemax = 30;
BOD5emax = 10;

BSS=2;
BCOD=1;
BNKj=20; % original BSM1
BNO=20; % original BSM1
BBOD5=2;
BNKj_new = 30; % updated BSM TG meeting no 8
BNO_new = 10; % updated BSM TG meeting no 8

pumpfactor = 0.04; % original BSM1, same for all pumped flows
PF_Qintr = 0.004;  % kWh/m3, pumping energy factor, internal AS recirculation
PF_Qr = 0.008;  % kWh/m3, pumping energy factor, AS sludge recycle
PF_Qw = 0.05;  % kWh/m3, pumping energy factor, AS wastage flow

%cut out the parts of the files to be used
inpart=in(startindex:(stopindex-1),:);
reac1part=reac1(startindex:(stopindex-1),:);
reac2part=reac2(startindex:(stopindex-1),:);
reac3part=reac3(startindex:(stopindex-1),:);
reac4part=reac4(startindex:(stopindex-1),:);
reac5part=reac5(startindex:(stopindex-1),:);
settlerpart=settler(startindex:(stopindex-1),:);
recpart=rec(startindex:(stopindex-1),:);

% Effluent concentrations
timevector=time(2:end)-time(1:(end-1));

Qevec = settlerpart(:,35).*timevector;
Qinvec=inpart(:,14).*timevector;
SOevec = settlerpart(:,22).*Qevec;
SIevec = settlerpart(:,23).*Qevec;
SSevec = settlerpart(:,24).*Qevec;
SNHevec = settlerpart(:,25).*Qevec;
SN2evec = settlerpart(:,26).*Qevec;
SNOevec = settlerpart(:,27).*Qevec;
SALKevec = settlerpart(:,28).*Qevec;
XIevec = settlerpart(:,29).*Qevec;
XSevec = settlerpart(:,30).*Qevec;
XBHevec = settlerpart(:,31).*Qevec;
XSTOevec = settlerpart(:,32).*Qevec;
XBAevec = settlerpart(:,33).*Qevec;
TSSevec = settlerpart(:,34).*Qevec;

Qetot = sum(Qevec);
Qeav = Qetot/totalt;

SOeload = sum(SOevec );
SIeload = sum(SIevec);
SSeload = sum(SSevec);
SNHeload = sum(SNHevec);
SN2eload = sum(SN2evec);
SNOeload = sum(SNOevec);
SALKeload = sum(SALKevec);
XIeload = sum(XIevec);
XSeload = sum(XSevec);
XBHeload = sum(XBHevec);
XSTOeload = sum(XSTOevec);
XBAeload = sum(XBAevec);
TSSeload = sum(TSSevec );
```

```
SOeav = SOeload /Qetot;
SIeav = SIeload/Qetot;
SSeav = SSeload/Qetot;
SNHeav = SNHeload/Qetot;
SN2eav = SN2eload/Qetot;
SNOeav = SNOeload/Qetot;
SALKeav = SALKeload/Qetot;
XIeav = XIeload /Qetot;
XSeav = XSeload /Qetot;
XBHeav = XBHeload /Qetot;
XSTOeav = XSTOeload /Qetot;
XBAeav = XBAeload /Qetot;
TSSeav = TSSeload /Qetot;

totalNKjevec2= (SNHevec+ i_NSI*(SIevec) + i_NSS*(SSevec) + i_NXI*(XIevec) + i_NXS*(XSevec) +
i_NBM*( XBHevec + XBAevec))./Qevec;
totalNevec2=   (SNOevec + SNHevec+ i_NSI*(SIevec) + i_NSS*(SSevec) + i_NXI*(XIevec) +
i_NXS*(XSevec) + i_NBM*( XBHevec + XBAevec))./Qevec;
totalCODevec2= (SIevec+SSevec+XIevec+XSevec+XBHevec+XBAevec+XSTOevec)./Qevec;
SNHevec2=       SNHevec./Qevec;
TSSevec2=       TSSevec./Qevec;
BOD5evec2=     (0.25*(SSevec+XSevec+(1-f_P)*(XBHevec+XBAevec + XSTOevec )))./Qevec;

totalNKjeload= SNHeload+ i_NSI*(SIeload) + i_NSS*(SSeload) + i_NXI*(XIeload) + i_NXS*(XSeload)
+ i_NBM*( XBHeload + XBAeload);
totalNeload=   SNOeload+totalNKjeload;
totalCODeload=(SIeload+SSeload+XIeload+XSeload+XBHeload+XBAeload+XSTOeload);
BOD5eload=     (0.25*(SSeload+XSeload+(1-f_P)*(XBHeload+XBAeload + XSTOeload)));


% Influent and Effluent quality index
SSin=       inpart(:,13);
CODin=      inpart(:,2)+ inpart(:,3)+ inpart(:,8)+inpart(:,9)+inpart(:,10)+inpart(:,11) +
inpart(:,12);
SNKjin=     inpart(:,4)+ i_NSI*(inpart(:,2)) + i_NSS*(inpart(:,3)) + i_NXI*(inpart(:,8)) +
i_NXS*(inpart(:,9)) + i_NBM*( inpart(:,10) + inpart(:,11));
SNOin=      inpart(:,6);
BOD5in=0.65*(inpart(:,3)+inpart(:,9)+(1-f_P)*(inpart(:,10)+inpart(:,11)+ inpart(:,12) ));

SSe=       settlerpart(:,34);
CODe=      settlerpart(:,23)+ settlerpart(:,24)+
settlerpart(:,29)+settlerpart(:,30)+settlerpart(:,31)+settlerpart(:,32) + settlerpart(:,33);
SNKje=     settlerpart(:,25)+ i_NSI*(settlerpart(:,23)) + i_NSS*(settlerpart(:,24)) +
i_NXI*(settlerpart(:,29)) + i_NXS*(settlerpart(:,30)) + i_NBM*( settlerpart(:,31) +
settlerpart(:,33));
SNOe=      settlerpart(:,27);
BOD5e=     0.25*(settlerpart(:,24)+settlerpart(:,30)+(1-
f_P)*(settlerpart(:,31)+settlerpart(:,32)+ settlerpart(:,33) ));

EQvecinst=(BSS*SSe+BCOD*CODe+BNKj*SNKje+BNO*SNOe+BBOD5*BOD5e).*settlerpart(:,35);
EQvecinst_new=(BSS*SSe+BCOD*CODe+BNKj_new*SNKje+BNO_new*SNOe+BBOD5*BOD5e).*settlerpart(:,35);
%updated BSM TG meeting no 8

IQvec=(BSS*SSin+BCOD*CODin+BNKj*SNKjin+BNO*SNOin+BBOD5*BOD5in).*Qinvec;
IQvec_new=(BSS*SSin+BCOD*CODin+BNKj_new*SNKjin+BNO_new*SNOin+BBOD5*BOD5in).*Qinvec; %updated
BSM TG meeting no 8

IQ=sum(IQvec)/(totalt*1000);
IQ_new=sum(IQvec_new)/(totalt*1000);

EQvec=(BSS*SSe+BCOD*CODe+BNKj*SNKje+BNO*SNOe+BBOD5*BOD5e).*Qevec;
EQvec_new=(BSS*SSe+BCOD*CODe+BNKj_new*SNKje+BNO_new*SNOe+BBOD5*BOD5e).*Qevec; %updated BSM TG
meeting no 8

EQ=sum(EQvec)/(totalt*1000);
EQ_new=sum(EQvec_new)/(totalt*1000); %updated BSM TG meeting no 8

% Costfactors for operation

% Sludge production
TSSreactors_start =
(reac1part(1,13)*VOL1+reac2part(1,13)*VOL2+reac3part(1,13)*VOL3+reac4part(1,13)*VOL4+reac5part
(1,13)*VOL5)/1000;
TSSreactors_end =
(reac1part(end,13)*VOL1+reac2part(end,13)*VOL2+reac3part(end,13)*VOL3+reac4part(end,13)*VOL4+r
eac5part(end,13)*VOL5)/1000;
```

```
TSSsettler_start=(settlerpart(1,42)*DIM(1)*DIM(2)/10+settlerpart(1,43)*DIM(1)*DIM(2)/10+settlerpart(1,44)*DIM(1)*DIM(2)/10+settlerpart(1,45)*DIM(1)*DIM(2)/10+settlerpart(1,46)*DIM(1)*DIM(2)/10+settlerpart(1,47)*DIM(1)*DIM(2)/10+settlerpart(1,48)*DIM(1)*DIM(2)/10+settlerpart(1,49)*DIM(1)*DIM(2)/10+settlerpart(1,50)*DIM(1)*DIM(2)/10+settlerpart(1,51)*DIM(1)*DIM(2)/10)/1000;
TSSsettler_end=(settlerpart(end,42)*DIM(1)*DIM(2)/10+settlerpart(end,43)*DIM(1)*DIM(2)/10+settlerpart(end,44)*DIM(1)*DIM(2)/10+settlerpart(end,45)*DIM(1)*DIM(2)/10+settlerpart(end,46)*DIM(1)*DIM(2)/10+settlerpart(end,47)*DIM(1)*DIM(2)/10+settlerpart(end,48)*DIM(1)*DIM(2)/10+settlerpart(end,49)*DIM(1)*DIM(2)/10+settlerpart(end,50)*DIM(1)*DIM(2)/10+settlerpart(end,51)*DIM(1)*DIM(2)/10)/1000;

TSSwasteconc=settlerpart(:,51)/1000;  %kg/m3
Qwasteflow=settlerpart(:,21);         %m3/d
TSSuvec=TSSwasteconc.*Qwasteflow.*timevector;
TSSproduced=sum(TSSuvec)+TSSreactors_end+TSSsettler_end-TSSreactors_start-TSSsettler_start;
TSSproducedperd = TSSproduced/totalt; %for OCI


% Aeration energy, original BSM1
kla1vec = kla1in(startindex:(stopindex-1),:);
kla2vec = kla2in(startindex:(stopindex-1),:);
kla3vec = kla3in(startindex:(stopindex-1),:);
kla4vec = kla4in(startindex:(stopindex-1),:);
kla5vec = kla5in(startindex:(stopindex-1),:);

kla1newvec = 0.0007*(VOL1/1333)*(kla1vec.*kla1vec)+0.3267*(VOL1/1333)*kla1vec;
kla2newvec = 0.0007*(VOL2/1333)*(kla2vec.*kla2vec)+0.3267*(VOL2/1333)*kla2vec;
kla3newvec = 0.0007*(VOL3/1333)*(kla3vec.*kla3vec)+0.3267*(VOL3/1333)*kla3vec;
kla4newvec = 0.0007*(VOL4/1333)*(kla4vec.*kla4vec)+0.3267*(VOL4/1333)*kla4vec;
kla5newvec = 0.0007*(VOL5/1333)*(kla5vec.*kla5vec)+0.3267*(VOL5/1333)*kla5vec;
airenergyvec = 24*(kla1newvec+kla2newvec+kla3newvec+kla4newvec+kla5newvec);
airenergy = sum(airenergyvec.*timevector);
airenergyperd = airenergy/totalt; % for OCI

% Aeration energy, updated BSM1 (and also for BSM2)
kla1newvec_new = SOSAT1*VOL1*kla1vec;
kla2newvec_new = SOSAT2*VOL2*kla2vec;
kla3newvec_new = SOSAT3*VOL3*kla3vec;
kla4newvec_new = SOSAT4*VOL4*kla4vec;
kla5newvec_new = SOSAT5*VOL5*kla5vec;

airenergyvec_new =
(kla1newvec_new+kla2newvec_new+kla3newvec_new+kla4newvec_new+kla5newvec_new)/(1.8*1000);
airenergy_new = sum(airenergyvec_new.*timevector);
airenergy_newperd = airenergy_new/totalt; % for OCI

% Mixing energy (calculated as kWh consumed for the complete evaluation
% period), same as for BSM2
mixnumreac1 = length(find(kla1vec<20));
mixnumreac2 = length(find(kla2vec<20));
mixnumreac3 = length(find(kla3vec<20));
mixnumreac4 = length(find(kla4vec<20));
mixnumreac5 = length(find(kla5vec<20));

mixenergyunitreac = 0.005; %kW/m3

mixenergyreac1 = mixnumreac1*mixenergyunitreac*VOL1;
mixenergyreac2 = mixnumreac2*mixenergyunitreac*VOL2;
mixenergyreac3 = mixnumreac3*mixenergyunitreac*VOL3;
mixenergyreac4 = mixnumreac4*mixenergyunitreac*VOL4;
mixenergyreac5 = mixnumreac5*mixenergyunitreac*VOL5;

mixenergy =
24*(mixenergyreac1+mixenergyreac2+mixenergyreac3+mixenergyreac4+mixenergyreac5)*sampletime;
mixenergyperd = mixenergy/totalt; % for OCI


% Pumping energy, original BSM1
Qintrflow = recpart(:,14);
Qrflow = settlerpart(:,14);
pumpenergyvec = pumpfactor*(Qwasteflow+Qintrflow+Qrflow);
pumpenergy = sum(pumpenergyvec.*timevector);
pumpenergyperd = pumpenergy/totalt; %for OCI

% Pumping energy (based on BSM2 principles)
Qintrflow_new = recpart(:,14);
Qrflow_new = settlerpart(:,14);
Qwflow_new = settlerpart(:,21);
```

```
pumpenergyvec_new = PF_Qintr*Qintrflow_new+PF_Qr*Qrflow_new+PF_Qw*Qwflow_new;
pumpenergy_new = sum(pumpenergyvec_new.*timevector);
pumpenergy_newperd = pumpenergy_new/totalt; % for OCI


% % Carbon source addition
carbon1vec = carbon1in(startindex:(stopindex-1),:);
carbon2vec = carbon2in(startindex:(stopindex-1),:);
carbon3vec = carbon3in(startindex:(stopindex-1),:);
carbon4vec = carbon4in(startindex:(stopindex-1),:);
carbon5vec = carbon5in(startindex:(stopindex-1),:);
Qcarbonvec = (carbon1vec+carbon2vec+carbon3vec+carbon4vec+carbon5vec);
carbonmassvec = Qcarbonvec*CARBONSOURCECONC/1000;
Qcarbon = sum(Qcarbonvec.*timevector); %m3
carbonmass = sum(carbonmassvec.*timevector); %kg COD
carbonmassperd = carbonmass/totalt; %for OCI


% Operational Cost Index for BSM1
TSScost = 5*TSSproducedperd;
airenergycost = 1*airenergyperd; %original BSM1
airenergy_newcost = 1*airenergy_newperd; %updated BSM1
mixenergycost = 1*mixenergyperd; %based on BSM2
pumpenergycost = 1*pumpenergyperd; % original BSM1
pumpenergy_newcost = 1*pumpenergy_newperd; % based on BSM2
carbonmasscost = 3*carbonmassperd;

OCI = TSScost + airenergycost + mixenergycost + pumpenergycost + carbonmasscost;
OCI_new = TSScost + airenergy_newcost + mixenergycost + pumpenergy_newcost + carbonmasscost;

% Calculate 95% percentiles for effluent SNH, TN and TSS; BSM2 standard criteria
SNHeffprctile=prctile(SNHevec2,95);
TNeffprctile=prctile(totalNevec2,95);
TSSeffprctile=prctile(TSSevec2,95);


disp(' ')
disp(['Overall plant performance during time ',num2str(time(1)),' to ',num2str(time(end)),'
days'])
disp('**************************************************')
disp(' ')
disp('Effluent average concentrations based on load')
disp('---------------------------------------------')
disp(['Effluent average flow rate = ',num2str(Qeav),' m3/d'])
disp(['Effluent average SO conc = ',num2str(SOeav),' mg (-COD)/l'])
disp(['Effluent average SI conc = ',num2str(SIeav),' mg COD/l'])
disp(['Effluent average SS conc = ',num2str(SSeav),' mg COD/l'])
disp(['Effluent average SNH conc = ',num2str(SNHeav),' mg N/l'])
disp(['Effluent average SN2 conc = ',num2str(SN2eav),' mg N/l'])
disp(['Effluent average SNO conc = ',num2str(SNOeav),' mg COD/l'])
disp(['Effluent average SALK conc = ',num2str(SALKeav),' mol HCO3/l'])
disp(['Effluent average XI conc = ',num2str(XIeav),' mg COD/l'])
disp(['Effluent average XS conc = ',num2str(XSeav),' mg COD/l'])
disp(['Effluent average XBH conc = ',num2str(XBHeav),' mg COD/l'])
disp(['Effluent average XSTO conc = ',num2str(XSTOeav),' mg COD/l'])
disp(['Effluent average XBA conc = ',num2str(XBAeav),' mg COD/l'])
disp(['Effluent average TSS conc = ',num2str(TSSeav),' mg TSS/m3'])
disp(' ')
disp(['Effluent average Kjeldahl N conc = ',num2str(SNHeav+ i_NSI*(SIeav) + i_NSS*(SSeav) +
i_NXI*(XIeav) + i_NXS*(XSeav) + i_NBM*( XBHeav + XBAeav)),' mg N/l'])
disp(['Effluent average total N conc = ',num2str(SNOeav + SNHeav+ i_NSI*(SIeav) +
i_NSS*(SSeav) + i_NXI*(XIeav) + i_NXS*(XSeav) + i_NBM*( XBHeav + XBAeav)),' mg N/l (limit = 18
mg COD/l) '])
disp(['Effluent average total COD conc =
',num2str(SIeav+SSeav+XIeav+XSeav+XBHeav+XBAeav+XSTOeav),' mg COD/l (limit = 100 mg COD/l)'])
disp(['Effluent average BOD5 conc = ',num2str(0.25*(SSeav+XSeav+(1-f_P)*(XBHeav+XBAeav +
XSTOeav))),' mg/l (limit = 10 mg/l)'])
disp(' ')
disp('Effluent average load')
disp('---------------------')
disp(['Effluent average SO load = ',num2str(SOeload/(1000*totalt)),' kg (-COD)/day'])
disp(['Effluent average SI load = ',num2str(SIeload/(1000*totalt)),' kg COD/day'])
disp(['Effluent average SS load = ',num2str(SSeload/(1000*totalt)),' kg COD/day'])
disp(['Effluent average SNH load = ',num2str(SNHeload/(1000*totalt)),' kg N/day'])
disp(['Effluent average SN2 load = ',num2str(SN2eload/(1000*totalt)),' kg N/day'])
disp(['Effluent average SNO load = ',num2str(SNOeload/(1000*totalt)),' kg N/day'])
disp(['Effluent average SALK load = ',num2str(SALKeload/(1000*totalt)),' kmol HCO3 /day'])
disp(['Effluent average XI load = ',num2str(XIeload/(1000*totalt)),' kg COD)/day'])
disp(['Effluent average XS load = ',num2str(XSeload/(1000*totalt)),' kg COD/day'])
disp(['Effluent average XBH load = ',num2str(XBHeload/(1000*totalt)),' kg COD/day'])
disp(['Effluent average XSTO load = ',num2str(XSTOeload/(1000*totalt)),' kg COD/day'])
disp(['Effluent average XBA load = ',num2str(XBAeload/(1000*totalt)),' kg COD/day'])
disp(['Effluent average TSS load = ',num2str(TSSeload/(1000*totalt)),' kg TSS/day'])
disp(' ')

disp(['Effluent average Kjeldahl N load = ',num2str(totalNKjeload/(1000*totalt)),' kg N/d'])
disp(['Effluent average total N load = ',num2str(totalNeload/(1000*totalt)),' kg N/d'])
disp(['Effluent average total COD load = ',num2str(totalCODeload/(1000*totalt)),' kg COD/d'])
disp(['Effluent average BOD5 load = ',num2str(BOD5eload/(1000*totalt)),' kg/d'])
disp(' ')

disp('Other effluent quality variables')
disp('--------------------------------')
disp(['Influent Quality (I.Q.) index = ',num2str(IQ),' kg poll.units/d (original BSM1
version)'])
disp(['Effluent Quality (E.Q.) index = ',num2str(EQ),' kg poll.units/d (original BSM1
version)'])
disp(['Influent Quality (I.Q.) index = ',num2str(IQ_new),' kg poll.units/d (updated BSM1
version)'])
disp(['Effluent Quality (E.Q.) index = ',num2str(EQ_new),' kg poll.units/d (updated BSM1
version)'])

disp(' ')
disp(['Sludge production for disposal = ',num2str(TSSproduced),' kg SS'])
disp(['Average sludge production for disposal per day = ',num2str(TSSproduced/totalt),' kg
SS/d'])
disp(['Sludge production released into effluent = ',num2str(TSSeload/1000),' kg SS'])
disp(['Average sludge production released into effluent per day =
',num2str(TSSeload/(1000*totalt)),' kg SS/d'])
disp(['Total sludge production = ',num2str(TSSproduced+TSSeload/1000),' kg SS'])
disp(['Total average sludge production per day =
',num2str(TSSproduced/totalt+TSSeload/(1000*totalt)),' kg SS/d'])

disp(' ')
disp(['Total aeration energy = ',num2str(airenergy),' kWh (original BSM1 version)'])
disp(['Average aeration energy per day = ',num2str(airenergy/totalt),' kWh/d (original BSM1
version)'])
disp(['Total aeration energy = ',num2str(airenergy_new),' kWh (updated BSM1 version)'])
disp(['Average aeration energy per day = ',num2str(airenergy_new/totalt),' kWh/d (updated BSM1
version)'])
disp(' ')
disp(['Total pumping energy (for Qintr, Qr and Qw) = ',num2str(pumpenergy),' kWh (original
BSM1 version)'])
disp(['Average pumping energy per day (for Qintr, Qr and Qw) = ',num2str(pumpenergy/totalt),'
kWh/d (original BSM1 version)'])
disp(['Total pumping energy (for Qintr, Qr and Qw) = ',num2str(pumpenergy_new),' kWh (based on
BSM2 principles)'])
disp(['Average pumping energy per day (for Qintr, Qr and Qw) =
',num2str(pumpenergy_new/totalt),' kWh/d (based on BSM2 principles)'])
disp(' ')
disp(['Total mixing energy = ',num2str(mixenergy),' kWh (based on BSM2 principles)'])
disp(['Average mixing energy per day = ',num2str(mixenergy/totalt),' kWh/d (based on BSM2
principles)'])
disp(' ')
disp(['Total added carbon volume = ',num2str(Qcarbon),' m3'])
disp(['Average added carbon flow rate = ',num2str(Qcarbon/totalt),' m3/d'])
disp(['Total added carbon mass = ',num2str(carbonmass),' kg COD'])
disp(['Average added carbon mass per day = ',num2str(carbonmass/totalt),' kg COD/d'])
disp(' ')
disp('Operational Cost Index')
disp('----------------------')
disp(['Sludge production cost index = ',num2str(TSScost),' (using weight 5 for BSM1)'])
disp(['Aeration energy cost index = ',num2str(airenergycost),' (original BSM1 version)'])
disp(['Updated aeration energy cost index = ',num2str(airenergy_newcost),' (updated BSM1
version)'])
disp(['Pumping energy cost index = ',num2str(pumpenergycost),' (original BSM1 version)'])
disp(['Updated pumping energy cost index = ',num2str(pumpenergy_newcost),' (based on BSM2
principles)'])
disp(['Carbon source addition cost index = ',num2str(carbonmasscost)])
disp(['Mixing energy cost index = ',num2str(mixenergycost),' (based on BSM2 principles)'])
disp(['Total Operational Cost Index (OCI) = ',num2str(OCI),' (original BSM1 version)'])
disp(['Updated Total Operational Cost Index (OCI) = ',num2str(OCI_new),' (using new aeraration
and pumping costs)'])
disp(' ')
```

```
Nviolation=find(totalNevec2>totalNemax);
CODviolation=find(totalCODevec2>totalCODemax);
SNHviolation=find(SNHevec2>SNHemax);
TSSviolation=find(TSSevec2>TSSemax);
BOD5violation=find(BOD5evec2>BOD5emax);

noofNviolation = 1;
noofCODviolation = 1;
noofSNHviolation = 1;
noofTSSviolation = 1;
noofBOD5violation = 1;

disp('Effluent violations')
disp('--------------------')
disp(['95% percentile for effluent SNH (Ammonia95) = ',num2str(SNHeffprctile),' g N/m3'])
disp(['95% percentile for effluent TN (TN95) = ',num2str(TNeffprctile),' g N/m3'])
disp(['95% percentile for effluent TSS (TSS95) = ',num2str(TSSeffprctile),' g SS/m3'])
disp(' ')

if not(isempty(Nviolation))
  disp('The maximum effluent total nitrogen level (18 mg N/l) was violated')
  disp(['during ',num2str(min(totalt,length(Nviolation)*sampletime)),' days, i.e.
',num2str(min(100,length(Nviolation)*sampletime/totalt*100)),'% of the operating time.'])
  for i=2:length(Nviolation)
    if Nviolation(i-1)~=(Nviolation(i)-1)
      noofNviolation = noofNviolation+1;
    end
  end
  disp(['The limit was violated at ',num2str(noofNviolation),' different occasions.'])
  disp(' ')
end

if not(isempty(CODviolation))
  disp('The maximum effluent total COD level (100 mg COD/l) was violated')
  disp(['during ',num2str(min(totalt,length(CODviolation)*sampletime)),' days, i.e.
',num2str(min(100,length(CODviolation)*sampletime/totalt*100)),'% of the operating time.'])
  for i=2:length(CODviolation)
    if CODviolation(i-1)~=(CODviolation(i)-1)
      noofCODviolation = noofCODviolation+1;
    end
  end
  disp(['The limit was violated at ',num2str(noofCODviolation),' different occasions.'])
  disp(' ')
end

if not(isempty(SNHviolation))
  disp('The maximum effluent ammonia nitrogen level (4 mg N/l) was violated')
  disp(['during ',num2str(min(totalt,length(SNHviolation)*sampletime)),' days, i.e.
',num2str(min(100,length(SNHviolation)*sampletime/totalt*100)),'% of the operating time.'])
  for i=2:length(SNHviolation)
    if SNHviolation(i-1)~=(SNHviolation(i)-1)
      noofSNHviolation = noofSNHviolation+1;
    end
  end
  disp(['The limit was violated at ',num2str(noofSNHviolation),' different occasions.'])
  disp(' ')
end

if not(isempty(TSSviolation))
  disp('The maximum effluent total suspended solids level (30 mg SS/l) was violated')
  disp(['during ',num2str(min(totalt,length(TSSviolation)*sampletime)),' days, i.e.
',num2str(min(100,length(TSSviolation)*sampletime/totalt*100)),'% of the operating time.'])
  for i=2:length(TSSviolation)
    if TSSviolation(i-1)~=(TSSviolation(i)-1)
      noofTSSviolation = noofTSSviolation+1;
    end
  end
  disp(['The limit was violated at ',num2str(noofTSSviolation),' different occasions.'])
  disp(' ')
end

if not(isempty(BOD5violation))
  disp('The maximum effluent BOD5 level (10 mg/l) was violated')
  disp(['during ',num2str(min(totalt,length(BOD5violation)*sampletime)),' days, i.e.
',num2str(min(100,length(BOD5violation)*sampletime/totalt*100)),'% of the operating time.'])
  for i=2:length(BOD5violation)
```

```
    if BOD5violation(i-1)~=(BOD5violation(i)-1)
      noofBOD5violation = noofBOD5violation+1;
    end
  end
  disp(['The limit was violated at ',num2str(noofBOD5violation),' different occasions.'])
  disp(' ')
end
```

A10. Results of BSM1_ASM3 bioP simulations (non-reactive settler)

Steady state values of ASM3 bioP state variables for the last anerobic, last anoxic and last aerobic tanks (TEMPMODEL = 1, SETTLER = 0, ss for 1000 d).

ASM3 bioP STATE VARIABLES		TANK 2 (last anaerobic tank)	TANK 4 (last anoxic tank)	TANK 7 (last aerobic tank)
1	S_O	0.0000	0.0004	2.0011
2	S_S	4.4790	0.4404	0.2416
3	S_I	30.0000	30.0000	30.0000
4	S_NH	23.6310	11.3594	3.3615
5	S_NO	0.0146	3.4103	10.8290
6	S_N2	20.5537	29.5416	30.1799
7	S_PO4	18.1696	9.1634	4.9482
8	S_HCO	5.6752	4.7051	3.6728
9	X_I	1544.0258	1549.7378	1552.6661
10	X_S	102.1717	57.4929	42.8849
11	X_H	1254.3786	1263.9107	1264.5986
12	X_STO	30.6301	18.0363	13.1720
13	X_PAO	578.0629	586.8573	590.1193
14	X_PP	77.6381	86.5302	90.7151
15	X_PHA	67.1265	45.8998	35.8728
16	X_A	129.6490	131.3450	132.6844
17	X_TSS	3309.8871	3307.1116	3307.6947

Performance evaluation of ASM3 bioP plant during dynamic simulation (TEMPMODEL = 1, SETTLER = 0, dry weather data):

```
Overall plant performance during time 7 to 14 days
**************************************************

Effluent average concentrations based on load
---------------------------------------------
Effluent average flow rate  = 18061.3485 m3/d
Effluent average SO2 conc   = 2.24 mg (-COD)/l
Effluent average SS conc    = 0.25533 mg COD/l
Effluent average SI conc    = 30 mg COD/l
Effluent average SNH conc   = 6.013 mg N/l
Effluent average SNO conc   = 9.6037 mg N/l
Effluent average SN2 conc   = 28.7342 mg N/l
Effluent average SPO4 conc  = 4.6271 mg P/l
Effluent average SHCO conc  = 3.9563 mol HCO3/m3
Effluent average XI conc    = 6.1224 mg COD/l
Effluent average XS conc    = 0.17391 mg COD/l
Effluent average XH conc    = 4.8165 mg COD/l
Effluent average XSTO conc  = 0.052315 mg COD/l
Effluent average XPAO conc  = 2.5284 mg COD/l
Effluent average XPP conc   = 0.38017 mg P/l
Effluent average XPHA conc  = 0.1702 mg COD/l
Effluent average XA conc    = 0.49167 mg COD/l
Effluent average XTSS conc  = 13.1366 mg SS/l  (limit = 35 mg SS/l)

Effluent average Kjeldahl N conc = 7.0589 mg N/l
Effluent average total N conc    = 16.6626 mg N/l  (limit = 15 mg N/l)
Effluent average total P conc    = 5.179 mg P/l  (limit = 2 mg P/l)
Effluent average total COD conc  = 44.5584 mg COD/l  (limit = 125 mg COD/l)
Effluent average BOD5 conc       = 1.7087 mg/l  (limit = 25 mg/l)
Effluent average load
---------------------
Effluent average SO2 load   = 40.4569 kg (-COD)/day
Effluent average SS load    = 4.6117 kg COD/day
Effluent average SI load    = 541.8405 kg COD/day
Effluent average SNH load   = 108.6022 kg N/day
Effluent average SNO load   = 173.4555 kg N/day
Effluent average SN2 load   = 518.9789 kg N/day
Effluent average SPO4 load  = 83.5711 kg P/day
Effluent average SHCO load  = 71.4565 kmol HCO3/day
Effluent average XI load    = 110.5783 kg COD/day
Effluent average XS load    = 3.1411 kg COD/day
Effluent average XH load    = 86.9921 kg COD/day
Effluent average XSTO load  = 0.94487 kg COD/day
Effluent average XPAO load  = 45.6672 kg COD/day
Effluent average XPP load   = 6.8665 kg P/day
Effluent average XPHA load  = 3.074 kg COD/day
Effluent average XA load    = 8.8802 kg COD/day
Effluent average XTSS load  = 237.2644 kg SS/day

Effluent average Kjeldahl N load = 127.494 kg N/d
Effluent average total N load    = 300.9495 kg N/d
Effluent average organic P load  = 9.9695 kg P/d
Effluent average total P load    = 93.5406 kg P/d
Effluent average total COD load  = 805.7298 kg COD/d
Effluent average BOD5 load       = 30.8609 kg/d

Other effluent quality variables
--------------------------------
Influent Quality (I.Q.) index = 41596.1442 kg poll.units/d (original BSM1 version)
Effluent Quality (E.Q.) index = 7360.0257 kg poll.units/d (original BSM1 version)
Influent Quality (I.Q.) index = 51373.9769 kg poll.units/d (updated BSM1 version)
Effluent Quality (E.Q.) index = 6900.411 kg poll.units/d (updated BSM1 version)
Influent Quality (I.Q.) index = 71527.2296 kg poll.units/d (updated ASM3 bioP version)
Effluent Quality (E.Q.) index = 16254.4724 kg poll.units/d (updated ASM3 bioP version)

Sludge production for disposal = 17502.8893 kg SS
Average sludge production for disposal per day = 2500.4128 kg SS/d
Sludge production released into effluent = 1660.8509 kg SS
Average sludge production released into effluent per day = 237.2644 kg SS/d
Total sludge production = 19163.7402 kg SS
Total average sludge production per day = 2737.6772 kg SS/d

Total aeration energy = 59838.912 kWh (original BSM1 version)
Average aeration energy per day = 8548.416 kWh/d (original BSM1 version)
Total aeration energy = 29859.2 kWh (updated BSM1 version)
Average aeration energy per day = 4265.6 kWh/d (updated BSM1 version)

Total pumping energy (for Qintr, Qr and Qw) = 20767.32 kWh (original BSM1 version)
Average pumping energy per day (for Qintr, Qr and Qw) = 2966.76 kWh/d (original BSM1 version)
Total pumping energy (for Qintr, Qr and Qw) = 2717.19 kWh (based on BSM2 principles)
Average pumping energy per day (for Qintr, Qr and Qw) = 388.17 kWh/d (based on BSM2
principles)

Total mixing energy = 3360 kWh (based on BSM2 principles)
Average mixing energy per day = 480 kWh/d (based on BSM2 principles)

Total added carbon volume = 0 m3
Average added carbon flow rate = 0 m3/d
Total added carbon mass = 0 kg COD
Average added carbon mass per day = 0 kg COD/d


Operational Cost Index
----------------------
Sludge production cost index = 12502.0638 (using weight 5 for BSM1)
```

```
Aeration energy cost index = 8548.416 (original BSM1 version)
Updated aeration energy cost index = 4265.6 (updated BSM1 version)
Pumping energy cost index = 2966.76 (original BSM1 version)
Updated pumping energy cost index = 388.17 (based on BSM2 principles)
Carbon source addition cost index = 0
Mixing energy cost index = 480 (based on BSM2 principles)
Total Operational Cost Index (OCI) = 24497.2398 (original BSM1 version)
Updated Total Operational Cost Index (OCI) = 17635.8338 (using new aeraration and pumping
costs)

Effluent violations
-------------------
95% percentile for effluent SNH (Ammonia95) = 11.041 g N/m3
95% percentile for effluent SPO4(Phosphate95) = 9.2827 g N/m3
95% percentile for effluent TN (TN95) = 21.0123 g N/m3
95% percentile for effluent TP (TP95) = 9.7343 g N/m3
95% percentile for effluent TSS (TSS95) = 16.0398 g SS/m3

The maximum effluent total phosphorus level (2 mg N/l) was violated
during 6.2708 days, i.e. 89.5833% of the operating time.
The limit was violated at 6 different occasions.

The maximum effluent total nitrogen level (18 mg N/l) was violated
during 2.3125 days, i.e. 33.0357% of the operating time.
The limit was violated at 9 different occasions.

The maximum effluent ammonia nitrogen level (4 mg N/l) was violated
during 4.6458 days, i.e. 66.369% of the operating time.
The limit was violated at 7 different occasions.
```

A11. Results of BSM1_ASM3 bioP simulations (reactive settler)

Steady state values of ASM3 bioP state variables for the last anerobic, last anoxic and last aerobic tanks (TEMPMODEL = 1, SETTLER = 1, ss for 1000 d).

ASM3 bioP STATE VARIABLES		TANK 2 (last anaerobic tank)	TANK 4 (last anoxic tank)	TANK 7 (last aerobic tank)
1	S_O	0.0000	0.0004	1.9849
2	S_S	2.7092	0.4562	0.2758
3	S_I	30.0000	30.0000	30.0000
4	S_NH	24.3652	12.4145	4.5809
5	S_NO	0.0037	3.7804	10.9843
6	S_N2	20.0599	28.2950	28.9065
7	S_PO4	26.2179	9.9214	2.4338
8	S_HCO	5.5978	4.7406	3.7886
9	X_I	1541.1673	1546.3153	1548.9488
10	X_S	93.7016	53.1669	39.8585
11	X_H	776.0617	778.2545	777.6382
12	X_STO	9.4853	8.0879	7.2168
13	X_PAO	1042.9234	1057.1942	1062.3527
14	X_PP	130.2812	146.4748	153.9205
15	X_PHA	109.4747	74.5025	58.1713
16	X_A	120.5132	122.2027	123.5235
17	X_TSS	3463.8171	3484.0981	3495.0971

Performance evaluation of ASM3 bioP plant during dynamic simulation (TEMPMODEL = 1, SETTLER = 1, dry weather data):

```
Overall plant performance during time 7 to 14 days
**************************************************

Effluent average concentrations based on load
---------------------------------------------
Effluent average flow rate  = 18061.3483 m3/d
Effluent average SO2 conc   = 0.22591 mg (-COD)/l
Effluent average SS conc    = 0.26539 mg COD/l
Effluent average SI conc    = 30 mg COD/l
Effluent average SNH conc   = 6.9619 mg N/l
Effluent average SNO conc   = 9.7402 mg N/l
Effluent average SN2 conc   = 27.7427 mg N/l
Effluent average SPO4 conc  = 2.4995 mg P/l
Effluent average SHCO conc  = 4.048 mol HCO3/m3
Effluent average XI conc    = 5.9856 mg COD/l
Effluent average XS conc    = 0.096443 mg COD/l
Effluent average XH conc    = 2.9829 mg COD/l
Effluent average XSTO conc  = 0.021404 mg COD/l
Effluent average XPAO conc  = 4.1758 mg COD/l
Effluent average XPP conc   = 0.57655 mg P/l
Effluent average XPHA conc  = 0.18776 mg COD/l
Effluent average XA conc    = 0.45029 mg COD/l
Effluent average XTSS conc  = 13.3974 mg SS/l  (limit = 35 mg SS/l)

Effluent average Kjeldahl N conc = 7.9854 mg N/l
Effluent average total N conc    = 17.7256 mg N/l  (limit = 15 mg N/l)
Effluent average total P conc    = 3.2429 mg P/l  (limit = 2 mg P/l)
Effluent average total COD conc  = 44.1442 mg COD/l  (limit = 125 mg COD/l)
Effluent average BOD5 conc       = 1.6498 mg/l  (limit = 25 mg/l)
Effluent average load
---------------------
Effluent average SO2 load   = 4.0802 kg (-COD)/day
Effluent average SS load    = 4.7933 kg COD/day
Effluent average SI load    = 541.8404 kg COD/day
Effluent average SNH load   = 125.7414 kg N/day
Effluent average SNO load   = 175.9206 kg N/day
Effluent average SN2 load   = 501.0709 kg N/day
Effluent average SPO4 load  = 45.1446 kg P/day
Effluent average SHCO load  = 73.1129 kmol HCO3/day
Effluent average XI load    = 108.1082 kg COD/day
Effluent average XS load    = 1.7419 kg COD/day
Effluent average XH load    = 53.8754 kg COD/day
Effluent average XSTO load  = 0.38659 kg COD/day
Effluent average XPAO load  = 75.4208 kg COD/day
Effluent average XPP load   = 10.4132 kg P/day
Effluent average XPHA load  = 3.3912 kg COD/day
Effluent average XA load    = 8.1328 kg COD/day
Effluent average XTSS load  = 241.9744 kg SS/day

Effluent average Kjeldahl N load = 144.2279 kg N/d
Effluent average total N load    = 320.1485 kg N/d
Effluent average organic P load  = 13.427 kg P/d
Effluent average total P load    = 58.5716 kg P/d
Effluent average total COD load  = 797.6908 kg COD/d
Effluent average BOD5 load       = 29.7979 kg/d

Other effluent quality variables
--------------------------------
Influent Quality (I.Q.) index = 41596.1442 kg poll.units/d (original BSM1 version)
Effluent Quality (E.Q.) index = 7743.8191 kg poll.units/d (original BSM1 version)
Influent Quality (I.Q.) index = 51373.9769 kg poll.units/d (updated BSM1 version)
Effluent Quality (E.Q.) index = 7426.8918 kg poll.units/d (updated BSM1 version)
Influent Quality (I.Q.) index = 71527.2296 kg poll.units/d (updated ASM3 bioP version)
Effluent Quality (E.Q.) index = 13284.0516 kg poll.units/d (updated ASM3 bioP version)

Sludge production for disposal = 18198.0032 kg SS
Average sludge production for disposal per day = 2599.7147 kg SS/d
Sludge production released into effluent = 1693.8211 kg SS
Average sludge production released into effluent per day = 241.9744 kg SS/d
Total sludge production = 19891.8243 kg SS
Total average sludge production per day = 2841.6892 kg SS/d

Total aeration energy = 59838.912 kWh (original BSM1 version)
Average aeration energy per day = 8548.416 kWh/d (original BSM1 version)
Total aeration energy = 29859.2 kWh (updated BSM1 version)
Average aeration energy per day = 4265.6 kWh/d (updated BSM1 version)

Total pumping energy (for Qintr, Qr and Qw) = 20767.32 kWh (original BSM1 version)
Average pumping energy per day (for Qintr, Qr and Qw) = 2966.76 kWh/d (original BSM1 version)
Total pumping energy (for Qintr, Qr and Qw) = 2717.19 kWh (based on BSM2 principles)
Average pumping energy per day (for Qintr, Qr and Qw) = 388.17 kWh/d (based on BSM2
principles)

Total mixing energy = 3360 kWh (based on BSM2 principles)
Average mixing energy per day = 480 kWh/d (based on BSM2 principles)

Total added carbon volume = 0 m3
Average added carbon flow rate = 0 m3/d
Total added carbon mass = 0 kg COD
Average added carbon mass per day = 0 kg COD/d


Operational Cost Index
---------------------
Sludge production cost index = 12998.5737 (using weight 5 for BSM1)
```

```
Aeration energy cost index = 8548.416 (original BSM1 version)
Updated aeration energy cost index = 4265.6 (updated BSM1 version)
Pumping energy cost index = 2966.76 (original BSM1 version)
Updated pumping energy cost index = 388.17 (based on BSM2 principles)
Carbon source addition cost index = 0
Mixing energy cost index = 480 (based on BSM2 principles)
Total Operational Cost Index (OCI) = 24993.7497 (original BSM1 version)
Updated Total Operational Cost Index (OCI) = 18132.3437 (using new aeraration and pumping
costs)

Effluent violations
-------------------
95% percentile for effluent SNH (Ammonia95) = 11.9649 g N/m3
95% percentile for effluent SPO4(Phosphate95) = 6.8865 g N/m3
95% percentile for effluent TN (TN95) = 22.0896 g N/m3
95% percentile for effluent TP (TP95) = 7.6008 g N/m3
95% percentile for effluent TSS (TSS95) = 16.3518 g SS/m3

The maximum effluent total phosphorus level (2 mg N/l) was violated
during 3.8646 days, i.e. 55.2083% of the operating time.
The limit was violated at 7 different occasions.

The maximum effluent total nitrogen level (18 mg N/l) was violated
during 3.7917 days, i.e. 54.1667% of the operating time.
The limit was violated at 8 different occasions.

The maximum effluent ammonia nitrogen level (4 mg N/l) was violated
during 5.125 days, i.e. 73.2143% of the operating time.
The limit was violated at 8 different occasions.
```