Anaerobic Digester Model #1 Fast (ADM1F)

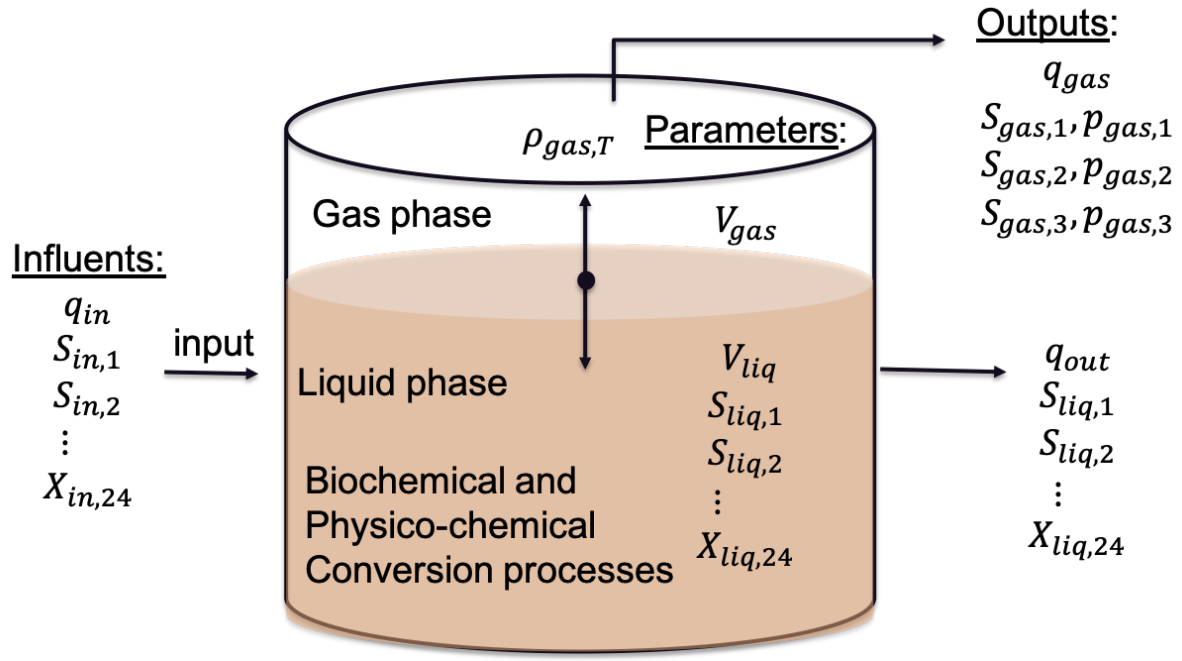
0.1

Elchin Jafarov, Satish Karra, Kuang Zhu, Wenjuan Zhang, and Kurt Solander

Jun 21, 2021

User Guide:

Anaerobic digestion (AD) process converts organic wastes into biogas. Biogas can generate heat and electricity through a cascade of biochemical reactions and has been adapted by various facilities and industries to treat and recover energy from high-strength liquid or solid waste streams. Anaerobic Digestion Model 1 (ADM1) is a mathematical model that describes the stoichiometry and kinetics of the essential biochemical reactions in AD. This repository includes C++ version of the Matlab/Simulink[1] version of the ADM1 model and the solid retention time (SRT) version[2]. The C++ version of the model is computationally more efficient than its Matlab/Simulink predecessor. We called this version of the model Anaerobic Digestion Model 1 Fast (ADM1F).



[1] Rosen C., Vrecko D., Gernaey K.V., Pons M.-N. and Jeppsson U. (2006). Implementing ADM1 for plant-wide benchmark simulations in Matlab/Simulink. Water Sci. Technol., 54(4), 11-19.

[2] *Zhu et al, in prep. A Novel Core-shell ADM1 model allows rapid optimization of membrane anaerobic digestion processes*

Chapter 1

Compile ADM1F

1. ADM1F uses external numerical library package PETSc. First download PETSc:

```
$ cd build; git clone -b release https://gitlab.com/petsc/petsc.git petsc
```

```
$ cd petsc; git checkout v3.14
```

2. Set **PETSC_DIR** and **PETSC_ARCH** in your environmental variables. We suggest to put these lines in your *~/.bashrc* or similar files (*~/.bash_profile* on Mac OS X). Once you add it into the bash file, run *source ~/.bash_profile*:

```
$ export PETSC_DIR=/path-to-my-ADM1F-folder/build/petsc
```

   and:

```
$ export PETSC_ARCH=macx-debug
```

   Make sure that 'adolc-utils' folder is in the 'build' folder.

3. Configure PETSC:

```
$ ./configure --download-mpich --with-cc=clang --with-fc=gfortran --with-debugging=0 --
↪download-adolc PETSC_ARCH=macx-debug --with-cxx-dialect=C++11 --download-colpack
```

NOTE: that these are for Mac OSX. If you are installing on a linux machine, then replace **clang** with **gcc**. Also, sometimes turning off *–with-fc=0* could help with compilation. This step will take awhile.

4. If configuration goes well, you can then compile. This step will take awhile too.:

```
$ make PETSC_DIR=/path-to-my-ADM1F-folder/build/petsc PETSC_ARCH=macx-debug all
```

5. After compilation, PETSc will show you how to test your installation (testing is optional).

6. Navigate back to the *build* folder (*cd ../*) and compile adm1f:

```
$ make adm1f
```

   or:

```
$ make
```

7. Set **ADM1F_EXE** in your environmental variable. Add this line in your *~/.bashrc* or similar files (*~/.bash_profile* on Mac OS X). Once you add it into the bash file, do not forget to *source ~/.bash_profile*:

```
$ export ADM1F_EXE=path-to-my-ADM1F-folder/build/adm1f
```

8. **NOTE**: There are two versions of the ADMF1: the original version (adm1f.cxx), and the modified version of the model (adm1f_srt.cxx, see *ADM1F SRT*).

Chapter 2

Running ADM1F

1. Make sure that **ADM1F_EXE:** is not empty (see step 7 from the previous section).:

```
$ echo $ADM1F_EXE
```

2. Navigate to the *simulations* folder and run the model:

```
$ $ADM1F_EXE

or using command-line options (see 4 and 5):

$ $ADM1F_EXE -ts_monitor -steady
```

3. Note that adm1f will look for three files *ic.dat*, *params.dat*, and *influent.dat*, which contain the initial conditions (45 values), parameters (100 values), and influent values (28 values), see *Inputs/Outputs*.
4. The command-line options are:
    - -Cat [val] - mass of Cat+ added [kmol/m3]
    - -Vliq [val] - volume of liquid [m3]
    - -Vgas [val] - volume of liquid [m3]
    - -t_resx [val] -SRT adjustment: t_resx = SRT-HRT, [d] (works only for adm1f_srt.cxx)
    - -params_file [filename] - specify params filename (default is params.dat)
    - -ic_file [filename] - specify initial conditions filename (default is ic.dat)
    - -influent_file [filename] - specify influent filename (default is influent.dat)
    - -ts_monitor - shows the timestep and time information on screen
    - -steady - run as steady state else runs as transient
    - -debug - gives out more details on the screen
5. More command-line options can be found here.

Chapter 3

ADM1F SRT

The adm1f_srt.cxx version includes solid retention time (SRT) and other modifications described below. To switch to the SRT version of the model change 'EXAMPLESC = adm1f_srt.cxx' , 'OBJECTS_PF = adm1f_srt.o', and 'adm1f: adm1f_srt.o' in the *build/makefile*. Then recompile the model (Compile ADM1F, step 6).

- Includes a term (T_resx) in the mass balance to separate the solids retention time from hydraulic retention time.
- Uses the empirical Hill function that describes the inhibition of acetogenesis and hydrogenotrophic methanogenesis by acetic acid with the noncompetitive inhibition model[1][2].
- Describes the inhibition of acetic acid on acetoclastic methanogenesis with the Haldane equation[3][4].
- Includes a adsorption-inhibition term describing the long-chain fatty acid[6] (LCFA) inhibition of LCFA degradation and methanogenesis.
- Includes Arrhenius equations describing the effect of temperature on bioreaction kinetics[5].
- Includes a cation term to simulate the addition of NaOH for pH adjustment.

References

[1] Love, N. G., R. J. Smith, K. R. Gilmore, and C. W. Randall. 1999. Oxime inhibition of nitrification during treatment of an ammonia-containing industrial waste. Water Environment Research 71:418–26.

[2] Oslislo, A., and Z. Lewandowski. 1985. Inhibition of nitrification in the packed bed reactors by selected organic compounds. Water Research 19:423–26.

[3] Haldane, J. B. S. 1930. Enzymes. London: Longmans.

[4] Andrews, J. F. 1968. A mathematical model for the continuous culture of microorganisms utilizing inhibitory substrates. Biotechnology and Bioengineering 10:707–23.

[6] Palatsi, J., Illa, J., Prenafeta-Boldú, F.X., Laureni, M., Fernandez, B., Angelidaki, I., Flotats. X. 2010. Long-chain fatty acids inhibition and adaptation process in anaerobic thermophilic digestion: Batch tests, microbial community structure and mathematical modelling. Bioresource Technology. 101, 7, 2243-2251.

[5] Novak, J. T. 1974. Temperature-substrate interactions in biological treatment. Journal, Water Pollution Control Federation 46:1984–94.

Chapter 4

Inputs/Outputs

4.1 Initial Conditions [ic.dat]

Index	Notation	Unit	Description
1	S_su	kgCOD/m3	soluble monosaccharides
2	S_aa	kgCOD/m3	soluble amino acids
3	S_fa	kgCOD/m3	soluble total LCFA
4	S_va	kgCOD/m3	soluble total valerate
5	S_bu	kgCOD/m3	soluble total butyrate
6	S_pro	kgCOD/m3	soluble total propionate
7	S_ac	kgCOD/m3	soluble acetate
8	S_h2	kgCOD/m3	hydrogen gas
9	S_ch4	kgCOD/m3	methane gas
10	S_IC	kmoleC/m3	soluble inorganic carbon
11	S_IN	kmoleC/m3	soluble inorganic nitrogen
12	S_I	kgCOD/m3	soluble inert materials
13	X_c_biom	kgCOD/m3	particulate of composites
14	X_ch_biom	kgCOD/m3	particulate of carbohydrate
15	X_pr_biom	kgCOD/m3	particulate of proteins
16	X_li_biom	kgCOD/m3	particulate of lipids
17	X_su	kgCOD/m3	monosaccharides degraders (microorganisms)
18	X_aa	kgCOD/m3	amino acids degraders (microorganisms)
19	X_fa	kgCOD/m3	LCFA degraders (microorganisms)
20	X_c4	kgCOD/m3	valerate and butyrate degraders (microorganisms)
21	X_pro	kgCOD/m3	propionate degraders (microorganisms)
22	X_ac	kgCOD/m3	acetate degraders (microorganisms)
23	X_h2	kgCOD/m3	hydrogen degraders (microorganisms)
24	X_I	kgCOD/m3	particulate of inerts
25	S_cation	kmole/m3	cations (strong base)
26	S_anion	kmole/m3	anions (strong acid)
27	S_hva	kgCOD/m3	soluble valerate acid
28	S_hbu	kgCOD/m3	soluble butyratic acid
29	S_hpro	kgCOD/m3	soluble propionatic acid
30	S_hac	kgCOD/m3	soluble acetatic acid
31	S_hco3	kmole/m3	soluble bicarbonate
32	S_nh3	kmole/m3	soluble ammonia
33	S_gas_h2	kgCOD/m3	soluble hydrogen gas
34	S_gas_ch4	kgCOD/m3	soluble methane gas

continues on next page

Table 1 – continued from previous page

Index	Notation	Unit	Description
35	S_gas_co2	kmole/m3	soluble carbon dioxide gas
36	Q	m3/d	flow rate
37	Temp	°C	temperature
38	S_D1_D	unitless	Dummy
39	S_D2_D	unitless	Dummy
40	S_D3_D	unitless	Dummy
41	X_D4_D	unitless	Dummy
42	X_D5_D	unitless	Dummy
43	S_H_ion	kmoleH+/m3	soluble hydrogen ion
44	S_co2	kmoleC/m3	soluble carbon dioxide
45	S_nh4	kmoleN/m3	soluble ammonium

4.2 Influent Data [influent.dat]

Index	Notation	Unit	Description
1	S_su_in	kgCOD/m3	soluble input monosaccharides
2	S_aa_in	kgCOD/m3	soluble input amino acids
3	S_fa_in	kgCOD/m3	soluble input total LCFA
4	S_va_in	kgCOD/m3	soluble input total valerate
5	S_bu_in	kgCOD/m3	soluble input total butyrate
6	S_pro_in	kgCOD/m3	soluble input total propionate
7	S_ac_in	kgCOD/m3	soluble input acetate
8	S_h2_in	kgCOD/m3	hydrogen gas
9	S_ch4_in	kgCOD/m3	methane gas
10	S_IC_in	kmoleC/m3	soluble input inorganic carbon
11	S_IN_in	kmoleC/m3	soluble input inorganic nitrogen
12	S_I_in	kgCOD/m3	soluble input inert materials
13	X_c_biom_in	kgCOD/m3	particulate input of composites
14	X_ch_biom_in	kgCOD/m3	particulate input of carbohydrate
15	X_pr_biom_in	kgCOD/m3	particulate input of proteins
16	X_li_biom_in	kgCOD/m3	particulate input of lipids
17	X_su_in	kgCOD/m3	monosaccharides degraders (microorganisms)
18	X_aa_in	kgCOD/m3	amino acids degraders (microorganisms)
19	X_fa_in	kgCOD/m3	LCFA degraders (microorganisms)
20	X_c4_in	kgCOD/m3	valerate and butyrate degraders (microorganisms)
21	X_pro_in	kgCOD/m3	propionate degraders (microorganisms)
22	X_ac_in	kgCOD/m3	acetate degraders (microorganisms)
23	X_h2_in	kgCOD/m3	hydrogen degraders (microorganisms)
24	X_I_in	kgCOD/m3	particulate input of inerts
25	S_cation_in	kmole/m3	input cations
26	S_anion_in	kmole/m3	input anions
27	Q	m3/d	flow rate
28	Temp	°C	temperature

4.3 Parameters [params.dat]

Index	Notation	Unit	Description
1	f_sI_xc	kgCOD/kg COD	fraction of composites (substrate) disintegrate to soluble inerts (product)
2	f_xI_xc	kgCOD/kg COD	fraction of composites (substrate) disintegrate to particulate inerts (product)
3	f_ch_xc	kgCOD/kg COD	fraction of composites (substrate) disintegrate to carbohydrates (product)
4	f_pr_xc	kgCOD/kg COD	fraction of composites (substrate) disintegrate to proteins (product)
5	f_li_xc	kgCOD/kg COD	fraction of composites (substrate) disintegrate to lipids (product)
6	N_xc	kmole N/(kg COD)	nitrogen content of composites
7	N_I	kmole N/(kg COD)	nitrogen content of inerts
8	N_aa	kmole N/(kg COD)	nitrogen content of amino acids
9	C_xc	kmole C/(kg COD)	carbon content of composites
10	C_sI	kmole C/(kg COD)	carbon content of soluble inerts
11	C_ch	kmole C/(kg COD)	carbon content of carbohydrates
12	C_pr	kmole C/(kg COD)	carbon content of proteins
13	C_li	kmole C/(kg COD)	carbon content of lipids
14	C_xI	kmole C/(kg COD)	carbon content of particulate inerts
15	C_su	kmole C/(kg COD)	carbon content of monosaccharides
16	C_aa	kmole C/(kg COD)	carbon content of amino acids
17	f_fa_li	kgCOD/kg COD	fraction of lipids (substrate) degrade to LCFA (product)
18	C_fa	kmole C/(kg COD)	carbon content of total LCFA
19	f_h2_su	kgCOD/kg COD	fraction of monosaccharides (substrate) degrade to hydrogen (product)
20	f_bu_su	kgCOD/kg COD	fraction of monosaccharides (substrate) degrade to butyrate (product)
21	f_pro_su	kgCOD/kg COD	fraction of monosaccharides (substrate) degrade to propionate (product)
22	f_ac_su	kgCOD/kg COD	fraction of monosaccharides (substrate) degrade to acetate (product)
23	N_bac	kmole N/(kg COD)	nitrogen content of synthesized into bacteria
24	C_bu	kmole C/(kg COD)	carbon content of total butyrate
25	C_pro	kmole C/(kg COD)	carbon content of total propionate
26	C_ac	kmole C/(kg COD)	carbon content of total acetate
27	C_bac	kmole C/(kg COD)	carbon content of synthesized into bacteria
28	Y_su	kgCOD_X/kg COD_S	yield of biomass on monosaccharides (substrate)
29	f_h2_aa	kgCOD/kg COD	fraction of amino acids (substrate) degrade to hydrogen (product)
30	f_va_aa	kgCOD/kg COD	fraction of amino acids (substrate) degrade to valerate (product)
31	f_bu_aa	kgCOD/kg COD	fraction of amino acids (substrate) degrade to butyrate (product)
32	f_pro_aa	kgCOD/kg COD	fraction of amino acids (substrate) degrade to propionate (product)
33	f_ac_aa	kgCOD/kg COD	fraction of amino acids (substrate) degrade to acetate (product)
34	C_va	kmole C/(kg COD)	carbon content of total valerate
35	Y_aa	kgCOD_X/kg COD_S	yield of biomass on amino acids (substrate)

continues on next page

Table 2 – continued from previous page

Index	Notation	Unit	Description
36	Y_fa	kgCOD_X/kg COD_S	yield of biomass on total LCFA (substrate)
37	Y_c4	kgCOD_X/kg COD_S	yield of biomass on butyrate (substrate)
38	Y_pro	kgCOD_X/kg COD_S	yield of biomass on total propionate (substrate)
39	C_ch4	kmole C/(kg COD)	carbon content of methane
40	Y_ac	kgCOD_X/kg COD_S	yield of biomass on total acetate (substrate)
41	Y_h2	kgCOD_X/kg COD_S	yield of biomass on hydrogen (substrate)
42	k_dis	1/d	disintegration rate
43	k_hyd_ch	1/d	hydrolysis rate of carbohydrates (carbs to simple sugars)
44	k_hyd_pr	1/d	hydrolysis rate of proteins
45	k_hyd_li	1/d	hydrolysis rate of lipids
46	K_S_IN	kgCOD_S/m3	half saturation value of inorganic nitrogen
47	k_m_su	kgCOD_S/kgCOD_X/d	Monod maximum specific uptake rate for monosaccharides
48	K_S_su	kgCOD_S/m3	half saturation value of monosaccharides
49	pH_UL_acidacet	unitless	upper pH limit of acidic acetate
50	pH_LL_acidacet	unitless	lower pH limit of acidic acetate
51	k_m_aa	kgCOD_S/kgCOD_X/d	Monod maximum specific uptake rate for amino acids
52	K_S_aa	kgCOD_S/m3	half saturation value of amino acids
53	k_m_fa	kgCOD_S/kgCOD_X/d	Monod maximum specific uptake rate for total LCFA
54	K_S_fa	kgCOD_S/m3	half saturation value of total LCFA
55	K_Ih2_fa	kgCOD/m3	inhibition constant LCFA (substrate) degradation by hydrogen (inhibitor)
56	k_m_c4	kgCOD_S/kgCOD_X/d	Monod maximum specific uptake rate for butyrate and valerate
57	K_S_c4	kgCOD_S/m3	half saturation value of butyrate
58	K_Ih2_c4	kgCOD/m3	inhibition constant for butyrate and valerate (substrate) degradation by hydrogen (inhibitor)
59	k_m_pro	kgCOD_S/kgCOD_X/d	Monod maximum specific uptake rate for total propionate
60	K_S_pro	kgCOD_S/m3	half saturation value of total propionate
61	K_Ih2_pro	kgCOD/m3	inhibition constant for propionate (substrate) degradation by hydrogen (inhibitor)
62	k_m_ac	kgCOD_S/kgCOD_X/d	Monod maximum specific uptake rate for total acetate
63	K_S_ac	kgCOD_S/m3	half saturation value of total acetate
64	K_I_nh3	kmol N/m3	inhibition constant by ammonia (inhibitor)
65	pH_UL_ac	unitless	upper pH limit for total acetate
66	pH_LL_ac	unitless	lower pH limit for total acetate
67	k_m_h2	kgCOD_S/kgCOD_X/d	Monod maximum specific uptake rate for hydrogen
68	K_S_h2	kgCOD_S/m3	half saturation value of hydrogen
69	pH_UL_h2	unitless	upper pH limit for hydrogen
70	pH_LL_h2	unitless	lower pH limit for hydrogen
71	k_dec_Xsu	1/d	first order decay rate for the monosaccharide degraders

continues on next page

Table 2 – continued from previous page

Index	Notation	Unit	Description
72	k_dec_Xaa	1/d	first order decay rate for the amino acids degraders
73	k_dec_Xfa	1/d	first order decay rate for the LCFA degraders
74	k_dec_Xc4	1/d	first order decay rate for the butyrate and valerate
75	k_dec_Xpro	1/d	first order decay rate for the propionate degraders
76	k_dec_Xac	1/d	first order decay rate for the acetate degraders
77	k_dec_Xh2	1/d	first order decay rate for the hydrogen degraders
78	R	bar m3 kmole-1 K-1	gas law constant (8.314e-2)
79	T_base	°C	base temperature
80	T_op	°C	operating temperature
81	pK_w_base	unitless	pKa of water
82	pK_a_va_base	unitless	pKa of total valerate
83	pK_a_bu_base	unitless	pKa of total butyrate
84	pK_a_pro_base	unitless	pKa of total propionate
85	pK_a_ac_base	unitless	pKa of total acetate
86	pK_a_co2_base	unitless	pKa of carbon dioxide
87	pK_a_IN_base	unitless	pKa of inorganic nitrogen
88	pK_a_hco3_base	unitless	pKa of bicarbonate
89	k_A_Bbu	1/M/d	acid base kinetic parameter for total butyrate
90	k_A_Bpro	1/M/d	acid base kinetic parameter for total propionate
91	k_A_Bac	1/M/d	acid base kinetic parameter for total acetate
92	k_A_Bco2	1/M/d	acid base kinetic parameter for carbon dioxide
93	k_A_BIN	1/M/d	acid base kinetic parameter for inhibitors
94	P_atm	bar	atmospheric pressure
95	kLa	1/d	gas-liquid transfer coefficient
96	K_H_h2o_base	M(liq)/bar	Henry's law coefficient of water
97	K_H_co2_base	M(liq)/bar	Henry's law coefficient of carbon dioxide
98	K_H_ch4_base	M(liq)/bar	Henry's law coefficient of methane
99	K_H_h2_base	M(liq)/bar	Henry's law coefficient of hydrogen
100	k_P	m2/d/bar	proportional gain

4.4 Outputs [indicator***.out]

Index	Notation	Unit	Description
1	Ssu	mg COD/L	soluble monosaccharides
2	Saa	mg COD/L	soluble amino acids
3	Sfa	mg COD/L	soluble total LCFA
4	Sva	mg COD/L	soluble total valerate
5	Sbu	mg COD/L	soluble total butyrate
6	Spro	mg COD/L	soluble total propionate
7	Sac	mg COD/L	soluble total acetate
8	Sh2	mg COD/L	soluble hydrogen
9	Sch4	mg COD/L	soluble methane
10	Sic	mg C/L	soluble inorganic carbon
11	Sin	mg N/L	soluble inorganic nitrogen
12	Si	mg COD/L	soluble inerts
13	Xc	mg COD/L	particulate composites
14	Xch	mg COD/L	particulate carbohydrates
15	Xpr	mg COD/L	particulate proteins
16	Xli	mg COD/L	particulate lipids
17	Xsu	mg COD/L	monosaccharides degraders (microorganisms)

continues on next page

Table 3 – continued from previous page

Index	Notation	Unit	Description
18	Xaa	mg COD/L	amino acids degraders (microorganisms)
19	Xfa	mg COD/L	LCFA degraders (microorganisms)
20	Xc4	mg COD/L	butyrate and valerate degraders (microorganisms)
21	Xpro	mg COD/L	propionate degraders (microorganisms)
22	Xac	mg COD/L	acetate degraders (microorganisms)
23	Xh2	mg COD/L	hydrogen degraders (microorganisms)
24	Xi	mg COD/L	particulate inerts
25	scat+	mmol/L	cations
26	san-	mmol/L	anions
27	pH	unitless	a scale used to specify how acidic or basic a water-based solution is
28	S_H+	mol/L	soluble hydrogen cation
29	Sva-	mg COD/L	soluble total valerate anion
30	Sbu-	mg COD/L	soluble total butyrate anion
31	Spro-	mg COD/L	soluble total propionate anion
32	Sac-	mg COD/L	soluble total acetate anion
33	Shco3-	mmol C/L	soluble bicarbonate anion
34	Sco2	mmol C/L	soluble carbon dioxide
35	Snh3	mg N/L	soluble ammonia
36	Snh4+	mg N/L	soluble ammonia cation (ammonium)
37	Sgas,h2	mg COD/L	soluble hydrogen gas
38	Sgas,ch4	mg COD/L	soluble methane gas
39	Sgas,co2	mmol C/L	soluble carbon dioxide gas
40	pgas,h2	atm	partial pressure of gas hydrogen
41	pgas,ch4	atm	partial pressure of gas methane
42	pgas,co2	atm	partial pressure of gas carbon dioxide
43	pgas,total	atm	partial pressure of gas all gases
44	pgas	m3/d	flow rate of gas
45	Si	mg COD/L	soluble inert organics
46	Ss	mg COD/L	readily biodegradable substrate
47	Xi	mg COD/L	particulate inert organics
48	Xs	mg COD/L	slowly biodegradable substrate
49	Xd	mg COD/L	particulate arising from biomass decay decay
50	Snh	mg N/L	ammonia and ammonium nitrogen, soluble the ammonia produced during ammonification process from soluble organic nitrogen
51	Sns	mg N/L	soluble biodegradable organic nitrogen generated during hydrolysis of particulate biodegradable organic nitrogen, suggesting it is the concentration of soluble organic hydrogen generated during hydrolysis
52	Xns	mg N/L	particulate biodegradable organic nitrogen generated during hydrolysis of particulate biodegradable organic nitrogen, suggesting it is the concentration of soluble organic hydrogen generated during hydrolysis
53	Salk	mg C/L	charge balance
54	TSS	mg TSS/L	total suspended solids
55	VFA_C2toC5	mg COD/L	volatile fatty acid from C2 to C5
56	mass_Sac	mg Hac/L	acetic acid
57	PAratio	kg acetate/ kg acetate equivalent of propionate	acetate propionate ratio

continues on next page

Table 3 – continued from previous page

Index	Notation	Unit	Description
58	Alk	mg/L CaCO3	alkalinity
59	NH3	mg N/L	ammonia
60	NH4	mg N/L	ammonium
61	LCFA	mgCOD_LCFA/L	long chain fatty acid
62	percentch4	%	biogas methane content, methane percentage output, percent by volume
63	energych4	%	energy content of CH4 gas, methane energy output, methane converted to COD, percentage of input that's converted to CH4 energy wise
64	efficiency	%	COD removal
65	VFA/ALK	g acetate eq./g CaCO3	volatile fatty acid to alkalinity ratio
66	ACN	kg COD/m3/d	acetate capacity number, the ratio between the maximum acetate utilization rate and the average acetate production rate
67	SampleT	d	

Chapter 5

Examples

5.1 Jupyter Notebook

All demo cases are documented in jupyter notebooks

5.1.1 ADM1F: Steady State

Here we run the steady state case and comparing it with the Matlab results. Make sure to compile `build/adm1f.cxx`.

Author: Elchin Jafarov

1. Steady State Run

[1]:
```
import os
import numpy as np
import pandas as pd
import subprocess
import sklearn.metrics as sklm
import xlrd
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
# navigate to simulations folder
os.chdir('../../simulations')
```

[3]:
```
# check the path to the executable
!echo $ADM1F_EXE
```

```
/Users/elchin/project/ADM1F_WM/build/adm1f
```

[4]:
```
# running the executable in the cell
!$ADM1F_EXE -steady
```

```
Vliq [m3] is: 3400.000000
Vgas [m3] is: 300.000000
Reading parameters in file: params.dat
Reading influent values in file: influent.dat
Reading initial condition values in file: ic.dat
Running as steady state problem.
Solving.
Done!
```

[5]:
```
# remove the output files
!sh clean.sh
```

[6]:
```
# or run using subprocess
subprocess.Popen('$ADM1F_EXE -ts_monitor -steady', shell=True)
```

[6]:
```
<subprocess.Popen at 0x7f84a4974040>
```

If the run is successful then `indicator-***.out` should be saved in the `simulations` folder. Here take the last time step saved in the last indicator file (`indicator-062.out`). The accending numeration of the output files corresponds to the time iterations taken towards the steady state condition.

2. Comparison of the ADM1F (C++) with ADM1 (Matlab)

The ADM1F runs much faster than the corresponding Matlab version. The main difference between C++ and the Matlab versions of the model is that ADM1F uses optimized solvers from the PETCS package to solve the corresponding mass balance equations. The ADM1F allows usage of the diffrent solvers. The ADM1(Matlab) is using ode45 nonstiff differential equation solver that cannot be changed. Below we benchmark ADM1F(C++) outputs with the ADM1(Matlab).

[7]:
```
# read the output produced by ADM1(Matlab) from the xls file
wb = xlrd.open_workbook('../docs/jupyter_notebook/out_sludge.xls')
sheet = wb.sheet_by_index(1)
results_matlab = [sheet.cell_value(4,i) for i in range(66)]
```

load the last file from the steady runs and compare it with the Matlab output.

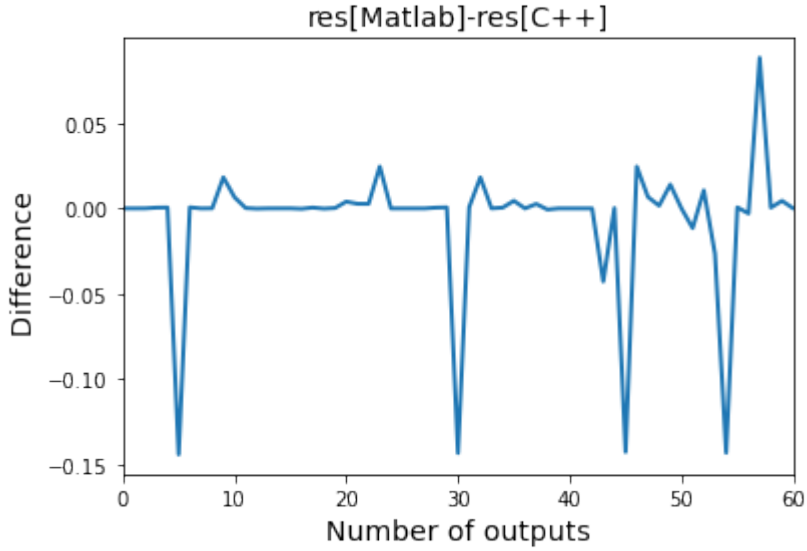
[8]:
```
results_c=np.loadtxt('indicator-062.out', skiprows=2, unpack=True)
```

[9]:
```
res = pd.DataFrame({
    "Matlab": np.asarray(results_matlab),
    "C++": np.asarray(results_c[:-1])})
res.head()
```

[9]:

	Matlab	C++
0	10.116210	10.11620
1	4.529384	4.52938
2	83.095031	83.09500
3	9.171951	9.17149
4	11.824647	11.82410

[10]:
```
plt.plot(res['Matlab']-res['C++'],linewidth=2)
plt.xlabel('Number of outputs',fontsize=14)
plt.ylabel('Difference ',fontsize=14);
plt.title('res[Matlab]-res[C++]',fontsize=14)
plt.xlim([0,60]);
```



[11]:
```
rmse=sklm.mean_squared_error(res['Matlab'],res['C++'])
mae=sklm.mean_absolute_error(res['Matlab'],res['C++'])
r2=sklm.r2_score(res['Matlab'],res['C++'])
print('MAE:',round(mae,4))
print('RMSE:',round(rmse,4))
print('R2 Score:',round(r2,4))
```

```
MAE: 0.0136
RMSE: 0.0014
R2 Score: 1.0
```

5.1.2 ADM1F: Execution time

Here we calculate the execution time for a sample of size 100. We perturb a certain number of elements in one of the inputs files (e.g. influent.dat, ic.dat, params.dat) by some 'percent' value. We sample perturbed elements 100 times in a non-repeatable fashion using latin hypercube 'lhs' or 'uniform' sampling methods. Then we calculate the execution time 100 times. Note, if you do not have any of the packages used in this script, use `pip install package_name`.

Authors: Wenjuan Zhang and Elchin Jafarov

[1]:
```
import adm1f_utils as adm1fu
import os
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
# navigate to simulations folder
os.chdir('../../simulations')
```

1. Let's vary elements of the influent.dat

[3]:
```
#Set the path to the ADM1F executable
ADM1F_EXE = '/Users/elchin/project/ADM1F_WM/build/adm1f'

# Set the value of percentage and sample size for lhs
percent = 0.1 # NOTE: for params percent should be <= 0.05
sample_size = 100
variable = 'influent'    # influent/params/ic
method = 'lhs'    #'uniform' or 'lhs'
```

[4]:
```
#use help command to learn more about create_a_sample_matrix function
#help(adm1fu.create_a_sample_matrix)
```

[5]:
```
index=adm1fu.create_a_sample_matrix(variable,method,percent,sample_size)
print ()
print ('Number of elements participated in the sampling:',len(index))
```

```
Saves a sampling matrix [sample_size,array_size] into var_influent.csv
sample_size,array_size:  (100, 11)
Each column of the matrix corresponds to a variable perturbed 100 times around its original␣
↪value
var_influent.csv SAVED!

Number of elements participated in the sampling: 11
```

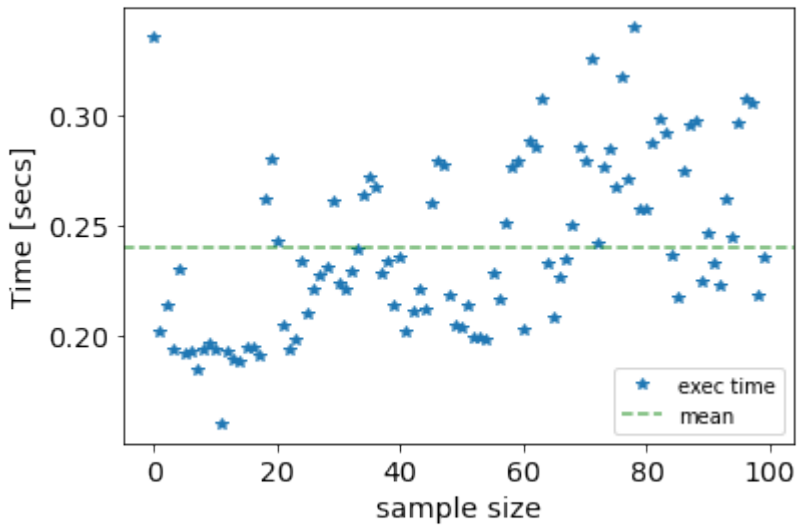
[6]:
```
exe_time=adm1fu.adm1f_output_sampling(ADM1F_EXE,variable,index)
```

```
All 100 runs were successfully computed
outputs_influent.csv SAVED!
```

Note: Depending on the computer system configuration, the computational time might vary.

[7]:
```
def plot_exec_time(exe_time):
    plt.plot(exe_time,'*')
    plt.axhline(exe_time.mean(),linestyle='--', alpha=0.6,color='green')
    plt.xlabel('sample size',fontsize=14)
    plt.ylabel('Time [secs]',fontsize=14)
    plt.legend(['exec time','mean'])
    print('cumulative time:',round(exe_time.sum(),2),'seconds',)
    print('mean time:',round(exe_time.mean(),2),'seconds')
    print('min time:',round(exe_time.min(),2),'seconds')
    print('max time:',round(exe_time.max(),2),'seconds')
    ax = plt.gca()
    ax.tick_params(axis = 'both', which = 'major', labelsize = 14)

plot_exec_time(exe_time)
```

```
cumulative time: 23.97 seconds
mean time: 0.24 seconds
min time: 0.16 seconds
max time: 0.34 seconds
```



2. Let's vary the param.dat elements and compute the execution time.

[8]:
```
# Set the value of percentage and sample size for lhs
percent = 0.05 # NOTE: for params percent should be <= 0.05
sample_size = 100
variable = 'params'     # influent/params/ic
method = 'lhs'     #'uniform' or 'lhs'
```

[9]:
```
index=adm1fu.create_a_sample_matrix(variable,method,percent,sample_size)
print ()
print ('Number of elements participated in the sampling:',len(index))
```

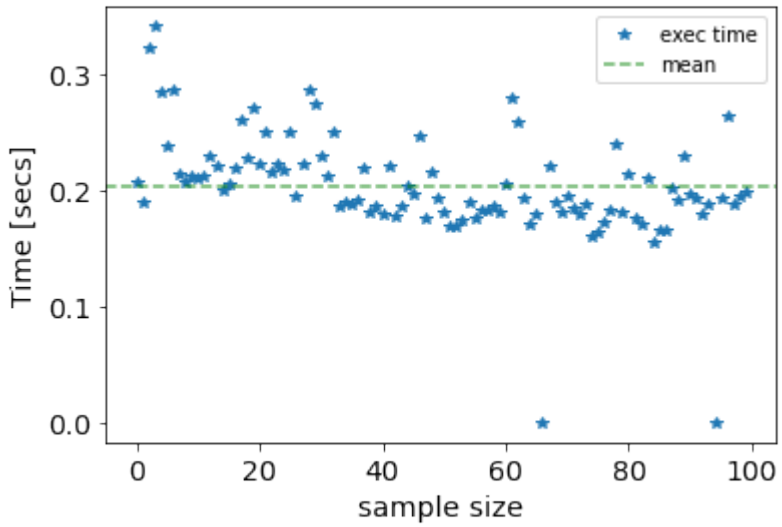
```
Saves a sampling matrix [sample_size,array_size] into var_params.csv
sample_size,array_size:  (100, 92)
Each column of the matrix corresponds to a variable perturbed 100 times around its original␣
↪value
var_params.csv SAVED!

Number of elements participated in the sampling: 92
```

[10]:
```
exe_time=adm1fu.adm1f_output_sampling(ADM1F_EXE,variable,index)
```

```
All 100 runs were successfully computed
outputs_params.csv SAVED!
```

[11]:
```
plot_exec_time(exe_time)
```

```
cumulative time: 20.32 seconds
mean time: 0.2 seconds
min time: 0.0 seconds
max time: 0.34 seconds
```



[ ]:

5.1.3 ADM1F SRT: single tank and two-phase anaerobic dynamic membrane bioreactor

This script is used to simulate a single tank suspended anaerobic dynamic membrane digester and a novel two-phase anaerobic dynamic membrane bioreator with separated SRT and HRT. In the two-phase reactor the effluent (model output) from the first phase dynamic membrane bioreactor is converted to the influent (model input) for the second-phase anaerobic dynamic membrane bioreactor as shown in the figure.

Note: Before running the ADM1F SRT simulations see intructions on how to compile `adm1f_srt.cxx` in the User Guide.

Mass balance equation with SRT:

The ADM1 and ADM1F solve the mass balance equation (i.e. mass_change = mass_in – mass_out + reaction). ADM1F_SRT version of the model includes solid retention time ($t_{res,X}$) as shown in the equation below.

$$\frac{d\,S_{\mathrm{liq,i}}}{dt} = \frac{qS_{\mathrm{in,i}}}{V_{\mathrm{liq}}} - \frac{qS_{\mathrm{liq,i}}}{V_{\mathrm{liq}}} + \sum_{j=1}^{12} \rho_j v_{i,j}$$

$$\frac{d\,X_{\mathrm{liq,i}}}{dt} = \frac{qX_{\mathrm{in,i}}}{V_{\mathrm{liq}}} - \frac{X_{\mathrm{liq,i}}}{t_{\mathrm{res,X}} + V_{\mathrm{liq}}/q} + \sum_{j=13}^{24} \rho_j v_{i,j}$$

Authors: Wenjuan Zhang, Elchin Jafarov, Kuang Zhu

[1]:
```
# Load packages
import numpy as np
import pandas as pd
import os
import seaborn as sns
import adm1f_utils as adm1fu
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
# navigate to simulations folder
os.chdir('../../simulations')
```

[3]:
```
# Grab the names and unit of all the outputs
(output_name,output_unit)=adm1fu.get_output_names()
```

[4]:
```
#check the path to the executable
!echo $ADM1F_EXE
```

```
/Users/elchin/project/ADM1F_WM/build/adm1f
```

Single tank anaerobic dynamic membrane bioreactor (AnDMBR)

To simulate the single tank AnDMBR that has separated SRT and HRT, We can call the function`reactor1` with different Q (flow-rate), Vliq (reactor volume), t_resx (SRT-HRT) values, the function will return the corresponding output

Usage example: reactor1(Q=100, t_resx=30, Vliq=300)

(Unit) Q: [m3/d], t_resx: [day], Vliq: [m3]

[5]:
```
# testing different SRTs=[0,1,2,...9] on the one-phase reactor
resx_list = [i for i in range(10)]
# setup the matrix with columns correspoding SRTs and rows to outputs
output1_resx = np.zeros((len(resx_list), 67))
```

(continues on next page)

(continued from previous page)

```
# here we utilize back euler solver and adataptive time step
# for more command options see "User Guide/Running ADM1F/step 5"
options='-ts_type beuler -ts_adapt_type basic -ts_max_snes_failures -1 -steady'

for i in range(len(resx_list)):
    output1_resx[i] = adm1fu.reactor1(opt=options, Vliq=300, Q=600, t_resx=resx_list[i])

np.savetxt('output_1phase.csv',output1_resx,delimiter=',',fmt='%1.4e')
```

```
Reactor run, phase-one:
$ADM1F_EXE -ts_type beuler -ts_adapt_type basic -ts_max_snes_failures -1 -steady -Vliq 300 -t_
↪resx 0 -influent_file influent_cur.dat
indicator-228.out
Reactor run, phase-one:
$ADM1F_EXE -ts_type beuler -ts_adapt_type basic -ts_max_snes_failures -1 -steady -Vliq 300 -t_
↪resx 1 -influent_file influent_cur.dat
indicator-307.out
Reactor run, phase-one:
$ADM1F_EXE -ts_type beuler -ts_adapt_type basic -ts_max_snes_failures -1 -steady -Vliq 300 -t_
↪resx 2 -influent_file influent_cur.dat
indicator-437.out
Reactor run, phase-one:
$ADM1F_EXE -ts_type beuler -ts_adapt_type basic -ts_max_snes_failures -1 -steady -Vliq 300 -t_
↪resx 3 -influent_file influent_cur.dat
indicator-563.out
Reactor run, phase-one:
$ADM1F_EXE -ts_type beuler -ts_adapt_type basic -ts_max_snes_failures -1 -steady -Vliq 300 -t_
↪resx 4 -influent_file influent_cur.dat
indicator-522.out
Reactor run, phase-one:
$ADM1F_EXE -ts_type beuler -ts_adapt_type basic -ts_max_snes_failures -1 -steady -Vliq 300 -t_
↪resx 5 -influent_file influent_cur.dat
indicator-517.out
Reactor run, phase-one:
$ADM1F_EXE -ts_type beuler -ts_adapt_type basic -ts_max_snes_failures -1 -steady -Vliq 300 -t_
↪resx 6 -influent_file influent_cur.dat
indicator-534.out
Reactor run, phase-one:
$ADM1F_EXE -ts_type beuler -ts_adapt_type basic -ts_max_snes_failures -1 -steady -Vliq 300 -t_
↪resx 7 -influent_file influent_cur.dat
indicator-542.out
Reactor run, phase-one:
$ADM1F_EXE -ts_type beuler -ts_adapt_type basic -ts_max_snes_failures -1 -steady -Vliq 300 -t_
↪resx 8 -influent_file influent_cur.dat
indicator-556.out
Reactor run, phase-one:
$ADM1F_EXE -ts_type beuler -ts_adapt_type basic -ts_max_snes_failures -1 -steady -Vliq 300 -t_
↪resx 9 -influent_file influent_cur.dat
indicator-576.out
```

[6]:
```
df1_resx = pd.read_csv('output_1phase.csv', sep=',', header=None)
df1_resx.columns = output_name
df1_resx.insert(0,"T_resx",resx_list)
```

Relation between t_resx and output when Vliq=300m3, Q=600m3/d

[7]:
```
# check the results with increasing T_resx, we should expect decrease in Ssu
df1_resx
```

[7]:

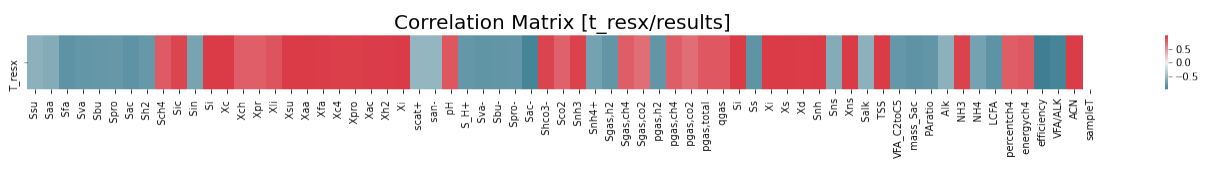
	T_resx	Ssu	Saa	Sfa	Sva	Sbu	Spro	Sac	\
0	0	2152.800	324.2700	7382.70	2381.700	3667.300	2542.700	8263.5	
1	1	123.060	49.1670	7751.20	2574.400	4202.900	3244.500	9600.7	

(continues on next page)

(continued from previous page)

```
2       2    63.057   26.6380  7839.00  2610.000  4268.600  3305.000  9742.9
3       3    39.313   17.3450   908.19    41.129    54.118    53.411  3339.7
4       4    30.634   13.5750   457.19    30.592    39.989    35.895  3066.6
5       5    25.261   11.2240   309.53    24.543    31.958    27.254  2856.4
6       6    21.606    9.6181   236.08    20.615    26.775    22.111  2685.8
7       7    18.960    8.4513   192.13    17.859    23.153    18.702  2543.7
8       8    16.955    7.5652   162.87    15.818    20.479    16.276  2423.2
9       9    15.383    6.8691   141.96    14.244    18.422    14.462  2319.3

        Sh2          Sch4   ...      Alk       NH3      NH4     LCFA   \
0  25.643000 -8.963700e-82   ...  44957.0  0.053067  1074.00  7382.70
1  30.352000  3.859800e-12   ...  59182.0  0.047701  1135.60  7751.20
2  30.772000  1.129100e-09   ...  61421.0  0.048208  1150.90  7839.00
3   0.000662  2.574900e+02   ...  16289.0  2.513100   963.03   908.19
4   0.000515  2.653300e+02   ...  18846.0  3.157300   963.70   457.19
5   0.000424  2.695900e+02   ...  21320.0  3.693300   966.73   309.53
6   0.000362  2.726300e+02   ...  23721.0  4.147800   970.17   236.08
7   0.000317  2.750200e+02   ...  26050.0  4.540600   973.69   192.13
8   0.000284  2.770000e+02   ...  28311.0  4.885500   977.21   162.87
9   0.000257  2.786900e+02   ...  30505.0  5.192300   980.68   141.96

    percentch4    energych4   efficiency   VFA/ALK           ACN  sampleT
0 -1.133000e-81       83.808       3.9774  0.351490 -3.203400e-72      0.5
1  1.599000e-12      100.630     -76.4800  0.310840  1.255000e-08      0.5
2  4.600500e-10      102.210    -155.0200  0.304150  2.832600e-07      0.5
3  5.579800e+01      209.370    -218.4400  0.200760  9.097200e+00      0.5
4  5.623900e+01      214.510    -304.2100  0.157850  1.324200e+01      0.5
5  5.648800e+01      217.230    -390.0200  0.129290  1.778600e+01      0.5
6  5.666900e+01      219.140    -475.6200  0.108900  2.278900e+01      0.5
7  5.681200e+01      220.630    -560.8900  0.093693  2.825000e+01      0.5
8  5.692900e+01      221.870    -645.8400  0.081984  3.416600e+01      0.5
9  5.702800e+01      222.920    -730.4900  0.072728  4.053400e+01      0.5

[10 rows x 68 columns]
```

[8]:
```
# correlation between SRT and output
plt.figure(figsize=(24,1))
corr1=df1_resx.corr()
sns.heatmap(corr1.iloc[0:1,-67:], xticklabels=df1_resx.columns[-67:], yticklabels=df1_resx.
↪columns[0:1], cmap=sns.diverging_palette(220, 10, as_cmap=True))
plt.title('Correlation Matrix [t_resx/results]',fontsize=20);
```



Configurations

Configuration	Vliq (m^3)	t_resx (d)	Q (m^3/d)
Sherri's	3400	0	134
Phase 1	340	1.5	618
Phase 2	3400	700	618/—

where t_resx = SRT - HRT

[9]:
```
config_default = {'Vliq':3400, 't_resx':0, 'Q':134}
config1 = {'Vliq':340, 't_resx':1.5, 'Q':618}
config2 = {'Vliq':3400, 't_resx':700, 'Q':618}
```

Default Configuration

[10]:
```
# output using default configuration
ls_default = adm1fu.reactor1(**config_default).tolist()
df_default = pd.DataFrame(data = [ls_default],columns=output_name, index=['Sherri\'s␣
↪Configuration'])
df_default
```

```
Reactor run, phase-one:
$ADM1F_EXE  -Vliq 3400 -t_resx 0 -influent_file influent_cur.dat
indicator-034.out
```

[10]:

	Ssu	Saa	Sfa	Sva	Sbu	Spro	\
Sherri's Configuration	7.17674	3.21796	54.9446	6.39778	8.23228	6.08339	

	Sac	Sh2	Sch4	Sic	...	Alk	\
Sherri's Configuration	1963.6	0.00012	48.3303	639.764	...	8438.52	

	NH3	NH4	LCFA	percentch4	energych4	\
Sherri's Configuration	9.32395	1026.69	54.9446	56.905	65.228	

	efficiency	VFA/ALK	ACN	sampleT
Sherri's Configuration	53.3712	0.220453	109.164	25.3731

[1 rows x 67 columns]

Configuration 1

[11]:
```
# output using configuration 1
ls_config1 = adm1fu.reactor1(**config1).tolist()
df_config1 = pd.DataFrame(data = [ls_config1],columns=output_name, index=['Phase 1␣
↪Configuration'])
df_config1
```

```
Reactor run, phase-one:
$ADM1F_EXE  -Vliq 340 -t_resx 1.5 -influent_file influent_cur.dat
indicator-051.out
```

[11]:

	Ssu	Saa	Sfa	Sva	Sbu	Spro	\
Phase 1 Configuration	80.5616	33.4704	7808.75	2598.14	4246.87	3285.25	

	Sac	Sh2	Sch4	Sic	...	Alk	\
Phase 1 Configuration	9696.05	28.5069	9.600970e-30	147.214	...	60316.4	

	NH3	NH4	LCFA	percentch4	energych4	\
Phase 1 Configuration	0.048011	1145.6	7808.75	4.182560e-30	104.203	

	efficiency	VFA/ALK	ACN	sampleT
Phase 1 Configuration	-104.737	0.308161	1.722550e-17	0.550162

[1 rows x 67 columns]

Configuration 2

[12]:
```
# output using configuration 2
ls_config2 = adm1fu.reactor1(**config2).tolist()
df_config2 = pd.DataFrame(data = [ls_config2],columns=output_name, index=['Phase 2␣
↪Configuration'])
df_config2
```

```
Reactor run, phase-one:
$ADM1F_EXE  -Vliq 3400 -t_resx 700 -influent_file influent_cur.dat
indicator-025.out
```

[12]:
```
                          Ssu      Saa      Sfa      Sva      Sbu     Spro   \
Phase 2 Configuration  2.70068  1.20723  19.0168  2.34856  3.01652  2.16597

                          Sac       Sh2     Sch4      Sic   ...      Alk   \
Phase 2 Configuration  1125.18  0.000045  75.1774  952.376  ...  21574.7

                          NH3      NH4     LCFA    percentch4    energych4   \
Phase 2 Configuration  14.7549  1186.64  19.0168        58.194      255.569

                      efficiency   VFA/ALK      ACN  sampleT
Phase 2 Configuration    -899.833  0.049221  3302.96  5.50162

[1 rows x 67 columns]
```

Two-Phase anaerobic dynamic membrane bioreactor

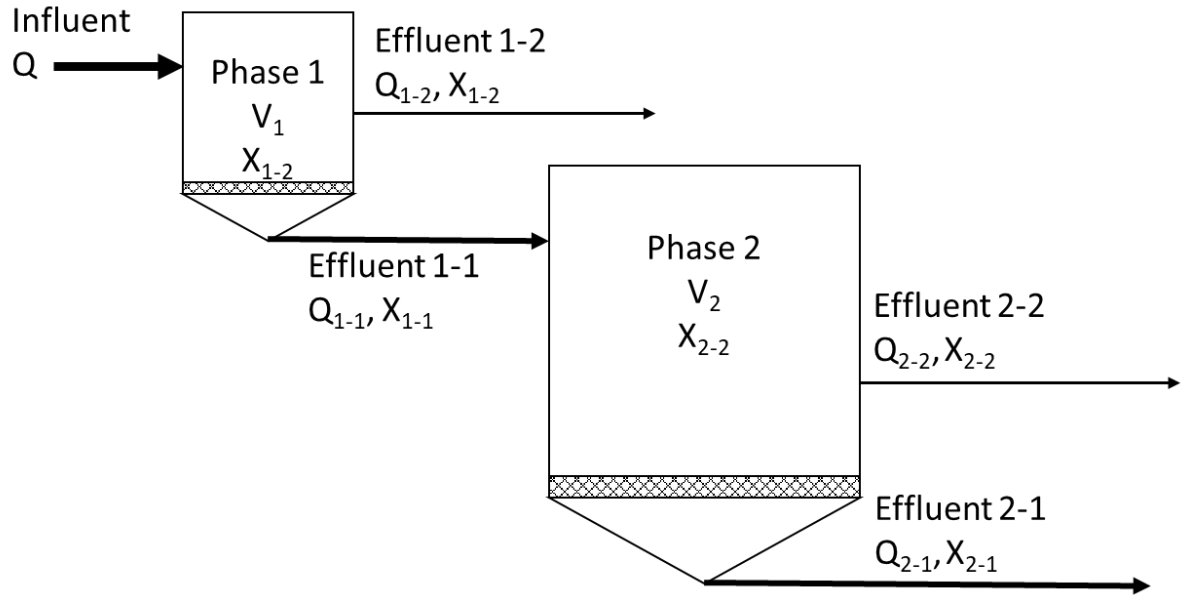
To simulate a novel two-phase AnDMBR, We can call the function `reactor2` with different Q1 (phase 1 flow rate), Q2 (phase 2 flow rate, normally equal to Q1), Vliq1 (phase 1 volume), Vliq2 (phase 2 volume), t_resx1 (SRT-HRT for phase 1), and t_resx2 (SRT-HRT for phase 1) values. In this function, the model output from phase 1 will be extracted and converted to the model input for phase 2. The function will return phase1 output and phase2 output

Usage example: reactor2(Q1=100, t_resx1=30, t_resx2=100, Vliq1=300, Vliq2=3000)

[13]:
```
from IPython.display import Image
print ('Schematics of the two-phase reactor')
Image(filename='../docs/_static/images/2phase_reactor.png')
```

```
Schematics of the two-phase reactor
```

[13]:



Config12

Configurations

Configuration	Vliq (m^3)	t_resx (d)	Q (m^3/d)
Sherri's	3400	0	134
Phase 1	340	1.5	618
Phase 2	3400	700	618/—

where t_resx = SRT - HRT

[14]:
```
# config12 = {"Vliq1":340, "Vliq2":3400, "t_resx1":1.5, "t_resx2":700, "Q1":618, "Q2":618}
config12 = {"Vliq1":340, "Vliq2":3400, "t_resx1":1.5, "t_resx2":700, "Q1":618}
```

[15]:
```
result_config12 = adm1fu.reactor2(**config12)
```

```
Reactor run, phase-one:
$ADM1F_EXE  -Vliq 340 -t_resx 1.5 -influent_file influent_cur.dat
indicator-051.out
Reactor run, phase-two:
$ADM1F_EXE -Vliq 3400 -t_resx 700 -influent_file influent_cur.dat
indicator-024.out
```

[16]:
```
# output using config12
result_config12 = adm1fu.reactor2(**config12)
ls1_config12 = result_config12[0].tolist()
ls2_config12 = result_config12[1].tolist()

df_config12 = pd.DataFrame(data = [ls1_config12],columns=output_name,index=['Phase1'])
df_config12.loc['Phase2'] = ls2_config12
df_config12
```

```
Reactor run, phase-one:
$ADM1F_EXE  -Vliq 340 -t_resx 1.5 -influent_file influent_cur.dat
indicator-051.out
Reactor run, phase-two:
```

(continues on next page)

(continued from previous page)

```
$ADM1F_EXE -Vliq 3400 -t_resx 700 -influent_file influent_cur.dat
indicator-024.out
```

[16]:
```
             Ssu        Saa        Sfa         Sva         Sbu        Spro   \
Phase1  80.561600  33.470400  7808.7500  2598.14000  4246.87000  3285.25000
Phase2   0.355681   0.132504    19.2757     2.37729     3.05196     2.19399

           Sac        Sh2          Sch4      Sic   ...      Alk        NH3   \
Phase1  9696.05  28.506900  9.371060e-31  147.214  ...  60316.4   0.048011
Phase2  1248.96   0.000046  7.114040e+01  867.356  ...  17101.8  15.289200

          NH4       LCFA    percentch4    energych4   efficiency   VFA/ALK  \
Phase1  1145.6  7808.7500  4.082390e-31      104.203    -104.737  0.308161
Phase2  1139.4    19.2757  6.198330e+01      192.277    -590.307  0.068884

                 ACN   sampleT
Phase1  5.381270e-18  0.550162
Phase2 -2.807000e+01  6.241160

[2 rows x 67 columns]
```

[ ]:

5.1.4 ADM1F_SRT: Input/output sensitivity

Here we explore the relationships between inputs and outputs. In the `ADM1F: Execution time` example we showed how to run the models with the perturbed input values from `influent.dat` and `param.dat` files. Assuming that you run the `ADM1F: Execution time` example and produced the `outputs_influent.csv` and `outputs_params.csv` files, we use these outputs here to study the relationship between influents and outputs, and params and outputs. If not just uncomment lines 5 and 18 and re-run the simulations.

Authors: Wenjuan Zhang and Elchin Jafarov

[1]:
```
import adm1f_utils as adm1fu
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

1. Influent/Output sensitivity

[2]:
```
# navigate to simulations folder
os.chdir('../../simulations')
```

[3]:
```
#Set the path to the ADM1F executable
ADM1F_EXE = '/Users/elchin/project/ADM1F_WM/build/adm1f'

# Set the value of percentage and sample size for lhs
percent = 0.1 # NOTE: for params percent should be <= 0.05
sample_size = 100
variable = 'influent'    # influent/params/ic
method = 'uniform'    #'uniform' or 'lhs'
```

[4]:
```
index=adm1fu.create_a_sample_matrix(variable,method,percent,sample_size)
print ()
print ('Number of elements participated in the sampling:',len(index))
```

```
Saves a sampling matrix [sample_size,array_size] into var_influent.csv
sample_size,array_size:  (100, 11)
Each column of the matrix corresponds to a variable perturbed 100 times around its original␣
↪value
var_influent.csv SAVED!

Number of elements participated in the sampling: 11
```

[5]:
```
#exe_time=adm1fu.adm1f_output_sampling(ADM1F_EXE,variable,index)
```

[6]:
```
[output_name,output_unit]=adm1fu.get_output_names()
alloutputs = pd.read_csv('outputs_influent.csv', sep=',', header=None)
alloutputs.columns = output_name
```

[7]:
```
alloutputs
```

[7]:

	Ssu	Saa	Sfa	Sva	Sbu	Spro	Sac	Sh2	\
0	7.37820	3.30795	56.7230	6.60272	8.46032	6.26942	1908.38	0.000123	
1	7.28460	3.26614	55.8953	6.52078	8.34621	6.18289	2033.37	0.000121	
2	6.85866	3.07582	52.1661	6.11321	7.84990	5.79147	1926.30	0.000114	
3	6.72726	3.01709	51.0289	5.97094	7.70738	5.67153	1931.76	0.000112	
4	7.55763	3.38810	58.3212	6.83699	8.63179	6.43593	2242.26	0.000126	
..	...	...	...	...	...	...	...	...	
95	7.65859	3.43319	59.2237	6.78604	8.84032	6.52986	1911.45	0.000128	
96	7.40606	3.32040	56.9708	6.57190	8.52763	6.29525	2049.40	0.000123	
97	7.06582	3.16840	53.9710	6.30301	8.09601	5.98132	1778.88	0.000118	
98	6.85456	3.07399	52.1312	6.12749	7.83407	5.78776	2146.27	0.000114	
99	7.44704	3.33870	57.3346	6.67541	8.53678	6.33321	1983.08	0.000124	

	Sch4	Sic	...	Alk	NH3	NH4	LCFA	percentch4	\
0	48.7774	615.799	...	8148.98	8.64671	987.952	56.7230	57.2720	
1	48.4068	645.849	...	8602.48	9.61694	1048.750	55.8953	56.8728	
2	47.7048	651.060	...	8442.25	9.57599	1031.740	52.1661	56.7510	
3	47.6768	660.852	...	8558.31	9.84058	1044.000	51.0289	56.6698	
4	49.1049	657.957	...	9061.95	10.49270	1109.360	58.3212	57.2318	
..	...	...	...	...	...	...	...	...	
95	48.9244	616.258	...	8197.36	8.37148	985.507	59.2237	56.5753	
96	48.6287	649.584	...	8699.15	9.58421	1054.650	56.9708	56.5357	
97	48.2397	605.950	...	7819.47	8.22490	949.405	53.9710	57.3804	
98	48.1629	685.484	...	9144.20	11.07250	1119.840	52.1312	56.8556	
99	48.5798	627.253	...	8349.28	9.04090	1016.750	57.3346	57.0513	

	energych4	efficiency	VFA/ALK	ACN	sampleT
0	66.2091	52.7615	0.222004	116.5620	24.3594
1	65.9860	53.8907	0.223891	92.1431	24.8179
2	61.6157	52.8110	0.216106	125.4410	27.1630
3	61.9098	53.9655	0.213730	116.0810	27.9786
4	68.8331	51.4620	0.234238	77.3666	23.5162
..	...	...	...	...	...
95	71.6467	52.7621	0.221140	125.5700	23.0752
96	69.7230	54.2485	0.223168	102.1390	24.2222
97	61.6582	52.9718	0.215719	129.6520	25.9740
98	64.3136	53.8121	0.222069	88.1808	27.1834
99	66.1387	53.8114	0.225090	103.0960	24.0295

[100 rows x 67 columns]

[8]:
```
[influent_name,influent_index]=adm1fu.get_influent_names()
```

[9]:
```
# since we did not use all the columns in the influent.dat (see create_a_sample_matrix)
# we use index to select used headers for used values
header=[]
for i in index:
    header.append(influent_name[i])

influent_inputs = pd.read_csv('var_influent.csv', sep=',', header=None)
influent_inputs.columns = header
influent_inputs.head()
```

[9]:
```
   S_su_in  S_aa_in  S_fa_in  S_ac_in  S_IN_in  X_ch_biom_in  X_pr_biom_in  \
0  2.41679  4.54049  3.26551  1.06715  0.00794       8.21103       7.65897
1  2.71197  4.44197  2.94117  0.97991  0.00798       8.47247       8.44312
2  2.37594  4.05372  3.09329  1.10619  0.00801       8.84280       8.55680
3  2.70155  4.55292  3.31319  1.00561  0.00784       9.14260       8.30096
4  2.35939  4.30043  3.00319  1.05070  0.00860       8.26193       9.19056

   X_li_biom_in    X_I_in          Q      Temp
0       5.45471  18.04704  139.57635  31.64409
1       5.01331  16.95071  136.99766  32.47646
2       4.62146  18.06979  125.17005  31.95536
3       4.69829  17.67225  121.52161  37.86524
4       5.36216  19.24420  144.58137  35.68530
```

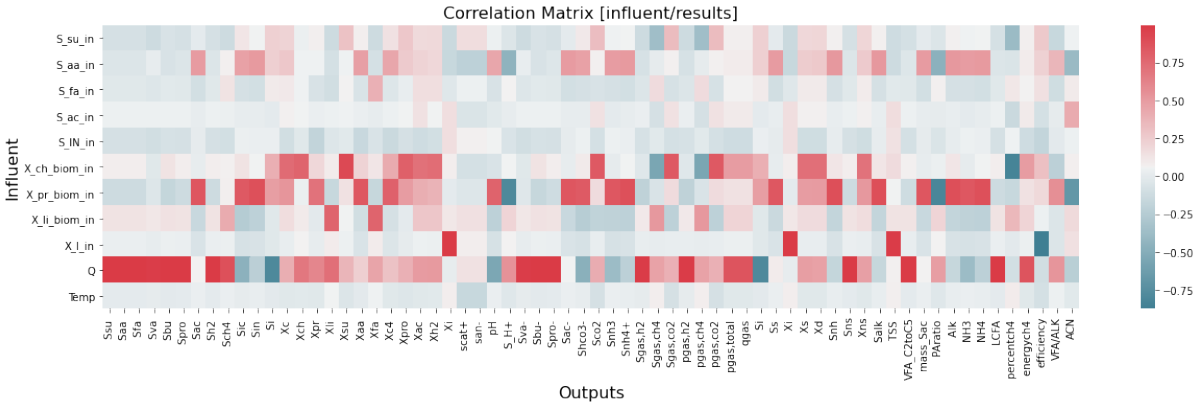
[10]:
```
# merge influent and output datasets
inout=pd.concat([influent_inputs,alloutputs], axis=1)
inout.head()
```

[10]:
```
   S_su_in  S_aa_in  S_fa_in  S_ac_in  S_IN_in  X_ch_biom_in  X_pr_biom_in  \
0  2.41679  4.54049  3.26551  1.06715  0.00794       8.21103       7.65897
1  2.71197  4.44197  2.94117  0.97991  0.00798       8.47247       8.44312
2  2.37594  4.05372  3.09329  1.10619  0.00801       8.84280       8.55680
3  2.70155  4.55292  3.31319  1.00561  0.00784       9.14260       8.30096
4  2.35939  4.30043  3.00319  1.05070  0.00860       8.26193       9.19056

   X_li_biom_in    X_I_in          Q  ...     Alk       NH3       NH4   \
0       5.45471  18.04704  139.57635  ...  8148.98   8.64671   987.952
1       5.01331  16.95071  136.99766  ...  8602.48   9.61694  1048.750
2       4.62146  18.06979  125.17005  ...  8442.25   9.57599  1031.740
3       4.69829  17.67225  121.52161  ...  8558.31   9.84058  1044.000
4       5.36216  19.24420  144.58137  ...  9061.95  10.49270  1109.360

     LCFA    percentch4    energych4   efficiency   VFA/ALK       ACN  sampleT
0  56.7230       57.2720      66.2091      52.7615  0.222004  116.5620  24.3594
1  55.8953       56.8728      65.9860      53.8907  0.223891   92.1431  24.8179
2  52.1661       56.7510      61.6157      52.8110  0.216106  125.4410  27.1630
3  51.0289       56.6698      61.9098      53.9655  0.213730  116.0810  27.9786
4  58.3212       57.2318      68.8331      51.4620  0.234238   77.3666  23.5162

[5 rows x 78 columns]
```

The correlation heat map matrix below shows that four influents have the highest impact on the results: `X_ch_biom_in`, `X_pr_biom_in`, `X_li_biom_in`, and `Q`.

[11]:
```
corr=inout.corr()
plt.figure(figsize=(21,5))
sns.heatmap(corr.iloc[0:11,11:-1], cmap=sns.diverging_palette(220, 10, as_cmap=True))
plt.title('Correlation Matrix [influent/results]',fontsize=16);
plt.ylabel('Influent',fontsize=16)
plt.xlabel('Outputs',fontsize=16)
```

[11]:
```
Text(0.5, 23.09375, 'Outputs')
```

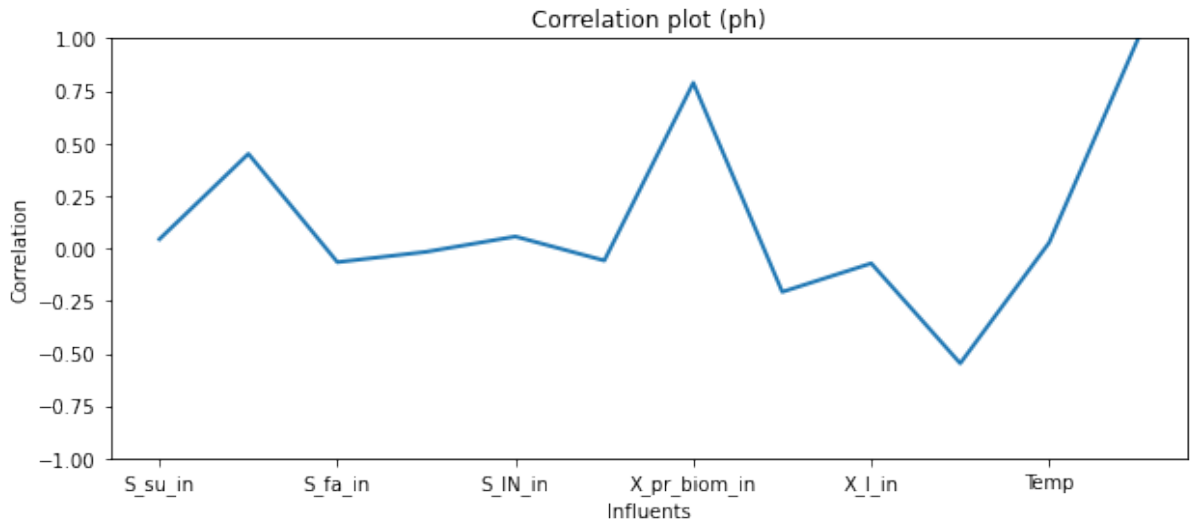


Let's merge ph values from the results with the influents and explore the correlations.

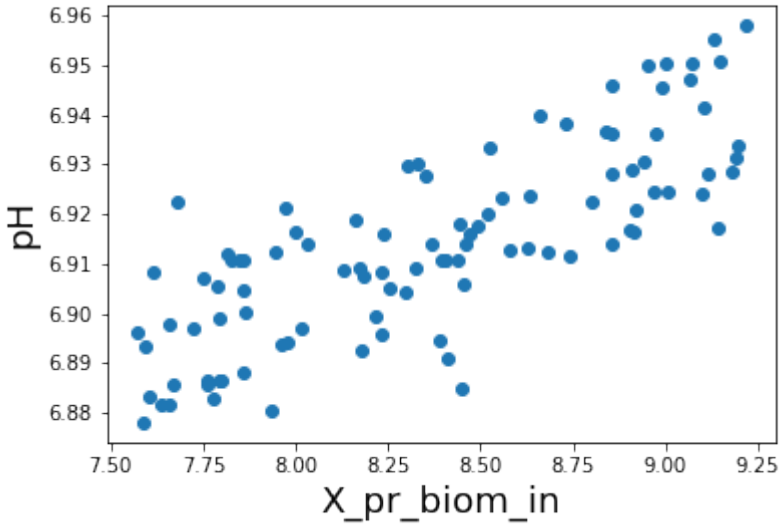
[12]:
```
influent_ph=pd.concat([influent_inputs,alloutputs[' pH ']], axis=1)
```

[13]:
```
plt.figure(figsize=(10,4))
influent_ph.corr().iloc[-1].plot(linewidth=2)
plt.title('Correlation plot (ph)')
plt.xlabel('Influents')
plt.ylabel('Correlation')
plt.ylim([-1,1])
```

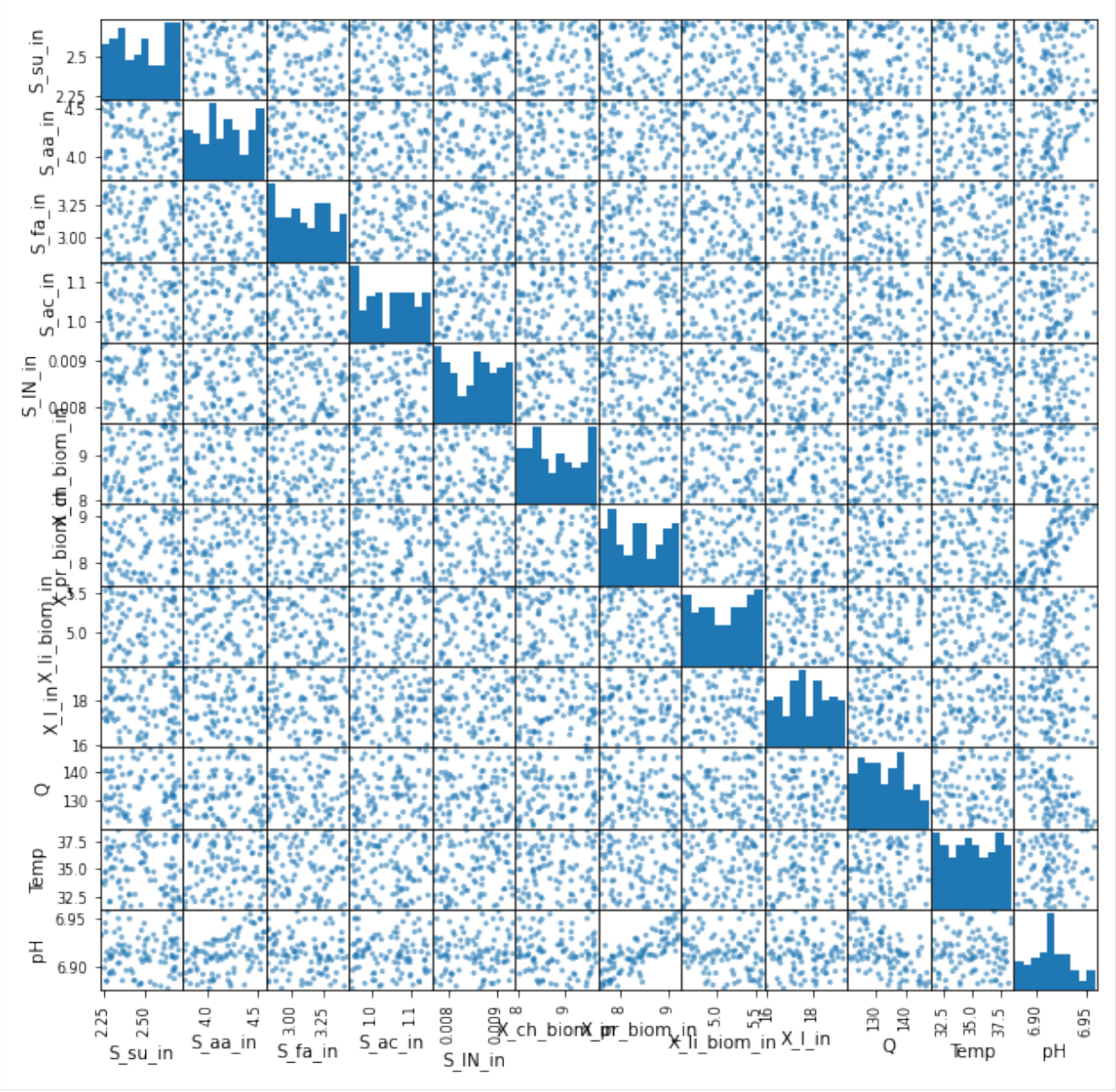
[13]:
```
(-1.0, 1.0)
```



[14]:
```
plt.scatter( influent_ph['X_pr_biom_in'],influent_ph[' pH '])
plt.xlabel('X_pr_biom_in',fontsize=18)
plt.ylabel('pH',fontsize=18);
```



[15]:
```
from pandas.plotting import scatter_matrix
scatter_matrix(influent_ph, alpha=0.6,figsize=(10,10));
```



2. Params/Output sensitivity

[16]:
```
# Set the value of percentage and sample size for lhs
percent = 0.1 # NOTE: for params percent should be <= 0.05
sample_size = 100
variable = 'params'     # influent/params/ic
method = 'uniform'    #'uniform' or 'lhs'
```

[17]:
```
index=adm1fu.create_a_sample_matrix(variable,method,percent,sample_size)
print ()
print ('Number of elements participated in the sampling:',len(index))
```

```
Saves a sampling matrix [sample_size,array_size] into var_params.csv
sample_size,array_size:  (100, 92)
Each column of the matrix corresponds to a variable perturbed 100 times around its original␣
↪value
var_params.csv SAVED!

Number of elements participated in the sampling: 92
```

[18]:
```
#exe_time=adm1fu.adm1f_output_sampling(ADM1F_EXE,variable,index)
```

[19]:
```
[output_name,output_unit]=adm1fu.get_output_names()
alloutputs = pd.read_csv('outputs_params.csv', sep=',', header=None)
alloutputs.columns = output_name
```

[20]:
```
[param_name,param_index]=adm1fu.get_param_names()
```

[21]:
```
# since we did not use all the columns in the influent.dat (see create_a_sample_matrix)
# we use index to select used headers for used values
header=[]
for i in index:
    header.append(param_name[i])

param_inputs = pd.read_csv('var_params.csv', sep=',', header=None)
param_inputs.columns = header
param_inputs.head()
```

[21]:
```
   f_sI_xc  f_xI_xc  f_ch_xc  f_pr_xc  f_li_xc     N_xc      N_I     N_aa  \
0  0.09116  0.27346  0.21535  0.21711  0.27475  0.00251  0.00420  0.00736
1  0.09692  0.25605  0.18183  0.21486  0.27367  0.00293  0.00450  0.00648
2  0.09653  0.25089  0.18351  0.19403  0.22666  0.00246  0.00420  0.00649
3  0.09891  0.25074  0.19437  0.20372  0.23318  0.00263  0.00468  0.00666
4  0.10742  0.26412  0.20268  0.20954  0.26893  0.00263  0.00414  0.00723

      C_xc     C_sI  ...      k_A_Bpro       k_A_Bac      k_A_Bco2  \
0  0.02895  0.02792  ...  1.026524e+10  1.086406e+10  9.205019e+09
1  0.02930  0.02715  ...  9.961014e+09  9.209860e+09  9.484090e+09
2  0.02824  0.03114  ...  1.043165e+10  1.081807e+10  9.359366e+09
3  0.02873  0.02895  ...  9.431350e+09  1.031777e+10  9.787729e+09
4  0.02958  0.03157  ...  1.090281e+10  1.020322e+10  1.063838e+10

        k_A_BIN        kLa  K_H_h2o_base  K_H_co2_base  K_H_ch4_base  \
0  1.087446e+10  207.51543       0.02859       0.03361       0.00146
1  1.097333e+10  185.69982       0.03129       0.03583       0.00146
2  9.475087e+09  218.85580       0.02930       0.03748       0.00140
3  1.030247e+10  184.26372       0.03229       0.03850       0.00127
4  1.076841e+10  189.12319       0.02950       0.03578       0.00138

   K_H_h2_base          k_P
0      0.00071  50821.70460
1      0.00079  45097.70847
2      0.00074  53707.49901
3      0.00086  49069.07961
4      0.00083  54000.23123

[5 rows x 92 columns]
```

[22]:
```
# merge influent and output datasets
inout=pd.concat([param_inputs,alloutputs], axis=1)
inout.head()
```

[22]:
```
   f_sI_xc  f_xI_xc  f_ch_xc  f_pr_xc  f_li_xc     N_xc      N_I     N_aa  \
0  0.09116  0.27346  0.21535  0.21711  0.27475  0.00251  0.00420  0.00736
1  0.09692  0.25605  0.18183  0.21486  0.27367  0.00293  0.00450  0.00648
2  0.09653  0.25089  0.18351  0.19403  0.22666  0.00246  0.00420  0.00649
3  0.09891  0.25074  0.19437  0.20372  0.23318  0.00263  0.00468  0.00666
4  0.10742  0.26412  0.20268  0.20954  0.26893  0.00263  0.00414  0.00723

      C_xc     C_sI  ...     Alk       NH3       NH4     LCFA    percentch4   \
0  0.02895  0.02792  ...  8909.63   9.53919  1060.990  61.5561       56.0945
```

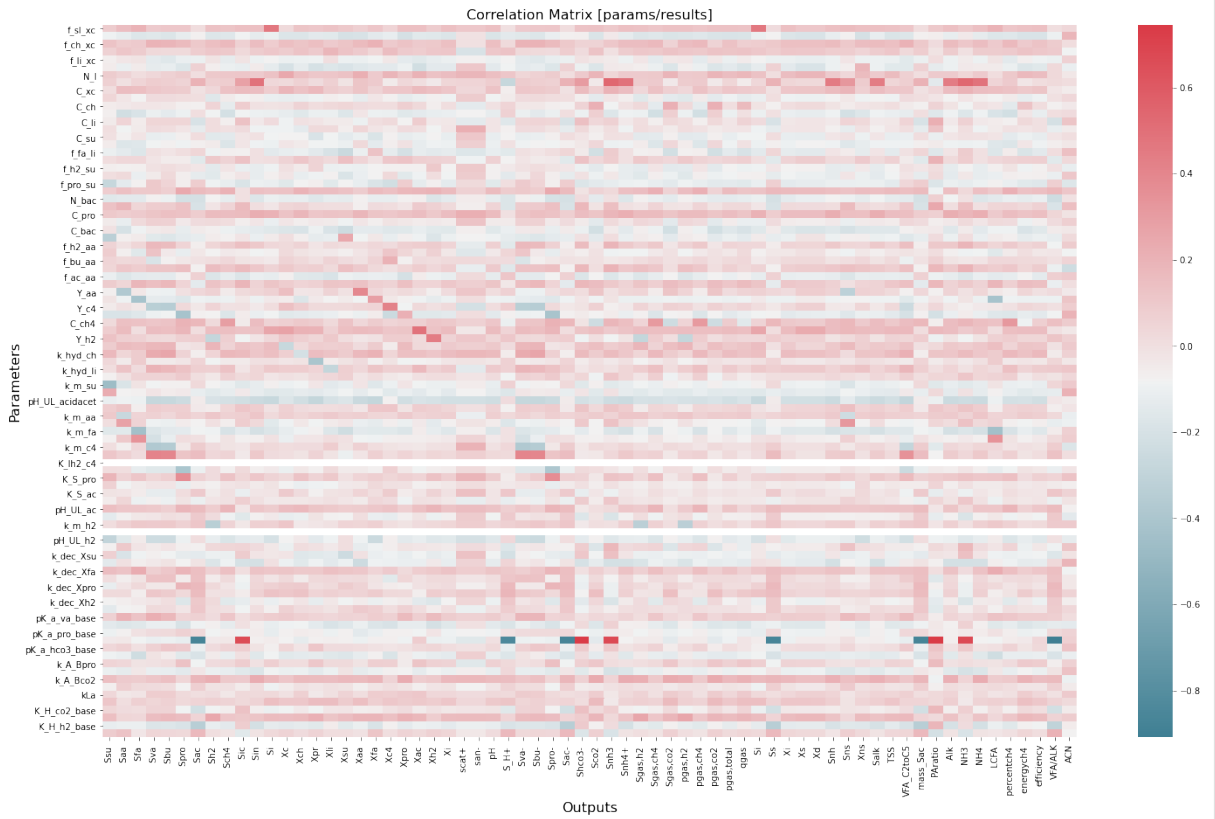
(continues on next page)

(continued from previous page)

```
1  0.02930  0.02715  ...  7707.69  10.88420   927.238  56.1139       62.1148
2  0.02824  0.03114  ...  7873.10   8.67396   927.390  68.2248       58.8253
3  0.02873  0.02895  ...  7961.45  10.83540   953.650  64.3854       57.4965
4  0.02958  0.03157  ...  8755.37  10.43910  1081.950  67.7978       56.9557

    energych4   efficiency   VFA/ALK        ACN  sampleT
0     65.7436      52.8869  0.222452    87.1484  25.3731
1     62.3272      55.0303  0.147138   654.5290  25.3731
2     64.9817      54.1026  0.191081   196.3670  25.3731
3     66.4942      55.2003  0.118815 -1663.6400  25.3731
4     64.9350      53.1639  0.222974    94.6821  25.3731

[5 rows x 159 columns]
```

[23]:
```
corr=inout.corr()
plt.figure(figsize=(25,15))
sns.heatmap(corr.iloc[0:92,92:-1], cmap=sns.diverging_palette(220, 10, as_cmap=True))
plt.title('Correlation Matrix [params/results]',fontsize=16);
plt.ylabel('Parameters',fontsize=16)
plt.xlabel('Outputs',fontsize=16)
```

[23]:
```
Text(0.5, 113.09375, 'Outputs')
```



[ ]:

5.1.5 ADM1F SRT: Synthetic Model-Data Calibration

Authors: Wenjuan Zhang and Elchin Jafarov

Before starting model clibration some additional softwares needs to be installed

1. Install Julia
2. Install Mads
   - "import Pkg; Pkg.add("Mads")" -> this command should be enough
   - Mads github page is https://github.com/madsjulia/Mads.jl
   - Mads documentation documentation page is http://madsjulia.github.io/Mads.jl/
3. To use jupyter notebook to display Julia, add JuliaHub, check this page https://juliahub.com/ui/Packages/IJulia/nfu7T/1.23.1

1. Sensitivity of the model output to parameter changes

Before starting model clibration we need to make sure that results are senstive to the parameter changes.

[2]:
```
import Mads
import DelimitedFiles
import OrderedCollections
```

[3]:
```
# check if you are in the right folder
pwd()
```

[3]:
```
"/Users/elchin/project/ADM1F_WM/calibration"
```

[4]:
```
# navigate to the calibration folder
cd("/Users/elchin/project/ADM1F_WM/calibration")
```

[5]:
```
# passing "tim.mads" to the Mads. "tim.mads" includes the "tim.jl" that executes the
# ADM1F model and saves seven outputs of interest.
filename = "tim.mads"
md = Mads.loadmadsfile(filename)
```

[5]:
```
Dict{String,Any} with 4 entries:
  "Parameters"    => OrderedCollections.OrderedDict{String,OrderedCollections.O...
  "Observations"  => OrderedCollections.OrderedDict{String,OrderedCollections.O...
  "Julia command" => "./tim.jl"
  "Filename"      => "tim.mads"
```

[6]:
```
# run the model with the default pamaters from "tim.mads"
# we denote the synthetic truth/observations
output_truth=Mads.forward(md)
```

[6]:
```
OrderedCollections.OrderedDict{Any,Float64} with 7 entries:
  "o2" => 145.362
  "o3" => 48.578
  "o1" => 363.333
  "o4" => 35.1163
  "o5" => 11.6628
  "o6" => 6.51573
  "o7" => 3.04116e-312
```

[7]:
```
# the variables in the dictinary that have "type" equal to "opt" will participate in the
# model-data calibration
md["Parameters"]["p48"]#["init"]
```

[7]:
```
OrderedCollections.OrderedDict{String,Any} with 4 entries:
  "init" => 0.5
  "max"  => 0.6
  "min"  => 0.4
  "type" => "opt"
```

[8]:
```
Mads.showobservations(md)
```

```
o1         target =          43.524 weight =               1
o2         target =         19.1588 weight =               1
o3         target =         6.23771 weight =               1
o4         target =         0.00779 weight =               1
o5         target =               0 weight =               1
o6         target =         6.87726 weight =               1
o7         target =         0.63904 weight =               1
Number of observations is 7
```

[10]:
```
# set observations equal to model runs with md["Parameters"]["p48"][init]=0.5
md["Observations"]["o1"]["target"]=output_truth["o1"]
md["Observations"]["o2"]["target"]=output_truth["o2"]
md["Observations"]["o3"]["target"]=output_truth["o3"]
md["Observations"]["o4"]["target"]=output_truth["o4"]
md["Observations"]["o5"]["target"]=output_truth["o5"]
md["Observations"]["o6"]["target"]=output_truth["o6"]
md["Observations"]["o7"]["target"]=output_truth["o7"]
```

[10]:
```
3.04116075262e-312
```

[11]:
```
# new synthetic observations
Mads.showobservations(md)
```

```
o1         target =         363.333 weight =               1
o2         target =         145.362 weight =               1
o3         target =          48.578 weight =               1
o4         target =         35.1163 weight =               1
o5         target =         11.6628 weight =               1
o6         target =         6.51573 weight =               1
o7         target =    3.04116e-312 weight =               1
Number of observations is 7
```

[27]:
```
#find out how many parameters are opt-in
for i in 1:100
    s1=string.(i)
    s2="p"
    if md["Parameters"][s2*s1]["type"]=="opt"
        println(i)
        println(md["Parameters"][s2*s1])
    end
end
```

```
48
OrderedCollections.OrderedDict{String,Any}("init" => 0.41,"max" => 0.6,"min" => 0.4,"type" =>
↪"opt")
54
OrderedCollections.OrderedDict{String,Any}("init" => 0.031,"max" => 0.045599999999999995,"min"␣
↪=> 0.0304,"type" => "opt")
60
OrderedCollections.OrderedDict{String,Any}("init" => 0.013,"max" => 0.018,"min" => 0.012,"type
↪" => "opt")
```

Two paramaters are opt-in from the tim.mads. This file can be changed manually or here by chaning the initial value (`init`) to a slightly different one.

[20]:
```
#let's opt-in a new parameter
md["Parameters"]["p54"]["type"]="opt" #K_S_fa
md["Parameters"]["p54"]
```

[20]:
```
OrderedCollections.OrderedDict{String,Any} with 4 entries:
  "init" => 0.032
  "max"  => 0.0456
  "min"  => 0.0304
  "type" => "opt"
```

[21]:
```
println(md["Parameters"]["p48"])
println(md["Parameters"]["p60"])
println(md["Parameters"]["p54"])
```

```
OrderedCollections.OrderedDict{String,Any}("init" => 0.55,"max" => 0.6,"min" => 0.4,"type" =>
↪"opt")
OrderedCollections.OrderedDict{String,Any}("init" => 0.017,"max" => 0.018,"min" => 0.012,"type
↪" => "opt")
OrderedCollections.OrderedDict{String,Any}("init" => 0.032,"max" => 0.045599999999999995,"min"␣
↪=> 0.0304,"type" => "opt")
```

[31]:
```
#change to new initial values
md["Parameters"]["p48"]["init"]=0.58
md["Parameters"]["p60"]["init"]=0.0165 #K_S_pro
md["Parameters"]["p54"]["init"]=0.045
println(md["Parameters"]["p48"])
println(md["Parameters"]["p60"])
println(md["Parameters"]["p54"])
```

```
OrderedCollections.OrderedDict{String,Any}("init" => 0.58,"max" => 0.6,"min" => 0.4,"type" =>
↪"opt")
OrderedCollections.OrderedDict{String,Any}("init" => 0.0165,"max" => 0.018,"min" => 0.012,"type
↪" => "opt")
OrderedCollections.OrderedDict{String,Any}("init" => 0.045,"max" => 0.045599999999999995,"min"␣
↪=> 0.0304,"type" => "opt")
```

[9]:
```
#we can also change the boundary of the allowed interval
#md["Parameters"]["p48"]["init"]=0.59
#md["Parameters"]["p48"]["max"]=0.7
#md["Parameters"]["p48"]["min"]=0.35
```

[9]:
```
0.35
```

[32]:
```
#rerun the model and compare the results with the previous forward run
output1=Mads.forward(md)
```

[32]:
```
OrderedCollections.OrderedDict{Any,Float64} with 7 entries:
  "o2" => 145.06
  "o3" => 48.3842
  "o1" => 363.488
  "o4" => 35.0174
  "o5" => 11.6574
  "o6" => 6.51656
  "o7" => -1.14602e-312
```

“o2” => 145.362 “o3” => 48.578 “o1” => 363.333 “o4” => 35.1163 “o5” => 11.6628 “o6” => 6.51573 “o7” => 3.04116e-312

2. Model Calibration

Once the sensitivity of the model outputs to changes in model parameters are established we can start the calibration. The elaborated sensitivy analysis is required, for an in-depth model calibration.

[34]:
```
p, r = Mads.calibraterandom(md)
```

[34]:
```
(OrderedCollections.OrderedDict("p1" => 0.09060657861205099,"p2" => 0.2964692632944874,"p3" =>␣
↪0.18538822521329718,"p4" => 0.21021796952211014,"p5" => 0.20993757892890663,"p6" => 0.
↪0025810430070856297,"p7" => 0.004153144361053321,"p8" => 0.006812701647314224,"p9" => 0.
↪029624259152454625,"p10" => 0.028993549342676592...), OptimBase.
↪MultivariateOptimizationResults{LsqFit.LevenbergMarquardt,Float64,1}(LsqFit.
↪LevenbergMarquardt(), [-0.4889181593485484, 1.1927438925052574, -0.3739490066652115, 0.
↪2583123288342893, -0.9293788161320726, -0.8970848356758778, -0.6686810831789546, -0.
↪5646712759835035, 0.3221739693428631, -0.16853854003002233  ...  -0.515933804055272, -0.
↪9755102180590636, -1.1189154560307426, 1.0593944723620046, 0.1384639308865269, -0.
↪5026261086108612, -0.012128162836436718, 0.28788576425004025, -0.0029143278533535384, -1.
↪1138366569543616], [-0.4889181593485484, 1.1927438925052574, -0.3739490066652115, 0.
↪2583123288342893, -0.9293788161320726, -0.8970848356758778, -0.6686810831789546, -0.
↪5646712759835035, 0.3221739693428631, -0.16853854003002233  ...  -0.515933804055272, -0.
↪9755102180590636, -1.1189154560307426, 1.0593944723620046, 0.1384639308865269, -0.
↪5026261086108612, -0.012128162836436718, 0.28788576425004025, -0.0029143278533535384, -1.
↪1138366569543616], 150.99483823310115, 10, true, false, 0.0001, 0.0, false, 0.001, 0.0,␣
↪false, 1.0e-6, 0.0, false, Iter     Function value   Gradient norm
------   --------------   --------------
, 1101, 10, 0))
```

[35]:
```
Mads.showobservations(md)
```

```
o1         target =         363.333 weight =               1
o2         target =         145.362 weight =               1
o3         target =          48.578 weight =               1
o4         target =         35.1163 weight =               1
o5         target =         11.6628 weight =               1
o6         target =         6.51573 weight =               1
o7         target =    3.04116e-312 weight =               1
Number of observations is 7
```

[52]:
```
println("initial guess:")
println("p48=",md["Parameters"]["p48"]["init"])
println("p60=",md["Parameters"]["p60"]["init"])
println("p54=",md["Parameters"]["p54"]["init"])
println(" ")
println("calibrated parameters:")
println("p48=",round(p["p48"],digits=3))
println("p60=",round(p["p60"],digits=3))
println("p54=",round(p["p54"],digits=3))
println(" ")
println("true parameters:")
println("p48=",0.41)
println("p60=",0.013)
println("p54=",0.031)
```

```
initial guess:
p48=0.58
p60=0.0165
p54=0.045

calibrated parameters:
p48=0.486
p60=0.013
p54=0.039
```

(continues on next page)

(continued from previous page)

```
true parameters:
p48=0.41
p60=0.013
p54=0.031
```

[51]:
```
#Only plot the outputs which participate in this match

est = Mads.forward(md, p);
obs = Mads.getobstarget(md);

idx_out = collect(1:1:7)
key_out = [string('o',i) for i in idx_out]

est_filt = [est[i] for i in key_out];
obs_filt = [obs[i] for i in idx_out];
Mads.plotseries([obs_filt est_filt]; names=["Truth", "Est."])
```

[ ]:
```
```

5.1.6 PH Control

The two-phase AnDMBR simulates a rumen environment to enhance the rates of hydrolysis and acidogenesis. A large amount of volatile fatty acids that decreases PH is produced, which could lead to digester failure. In real-life conditions, sodium hydroxide is added into the first-phase reactor. This maintains a PH around 6.3 and ensures optimal microbial activities, which is similar to the rumen reactor of a cow. Similarly, the PH in the second-phase methane-producing AnDMBR is maintained at 7.2 to provide optimal reactor functionality. In the AnDMBR model, we retain the appropriate levels of PH within each reactor phase by adding corresponding cation mass, which is estimated via the data consistent inversion method[1]. The cation amount can then be used to calculate the amount of sodium hydroxide in real implementation. The method and more in-depth mathematical description of the *ph-control* methods can be found in Dr. Zhang's Ph.D. thesis.

Data Consistent Inversion Method

Data consistent inversion method is used here to determine the cation needed to adjust the PH level in the reactor. It is first introduced for uncertainty quantification in inverse problems in[?] where the ideas and derivations behind it were discussed in great detail. Since the goal here is to solve the inverse problem where target PH, μ^*, can be achieved, we can apply this method to our model. Our goal here is to control the PH in the reactor within an acceptable range. We denote the target PH value by μ^*, a small deviation is allowed around μ^* to meet real-life conditions.

Gaussian (or normal) distribution denoted by $N(\mu, \sigma^2)$ has a property that there is a probability of 99.73% that its observation denoted by X will lie within three standard deviations (3σ) of the mean (μ). This is the so-called three-sigma rule of thumb, to write it in mathematical notation cite{3sigma},

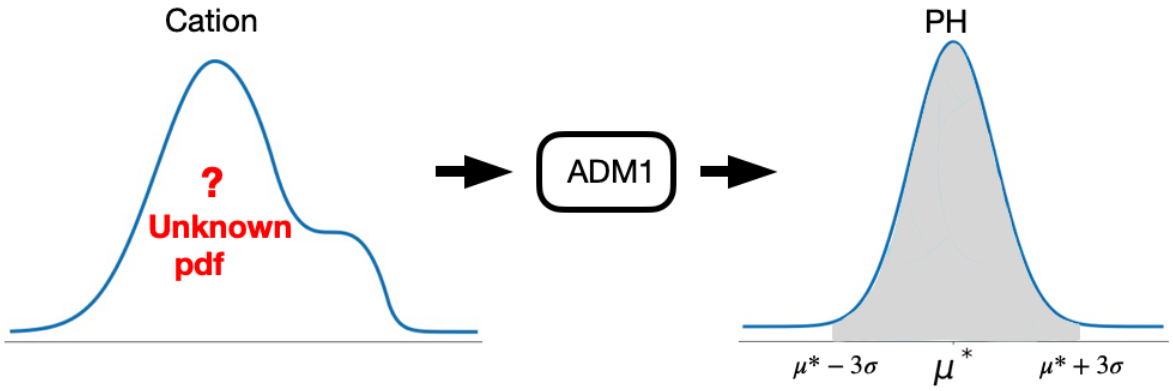
$$P(\mu - 3\sigma \leq X \leq \mu + 3\sigma) \approx 99.73\%.$$

We would like to use this special property of Gaussian distribution to achieve a high chance of predicted PH being in the three standard deviations of μ^*. Therefore we assume that our predicted PH value follows a Gaussian distribution[2], $N(\mu^*, \sigma^2)$. We first propose an initial probability density function (pdf) of cationfootnote{A uniform distribution $U(0, 0.2)$ is assumed in our model.}, then the data consistent inversion method is applied to update the pdf of cation such that the pdf of $N(\mu^*, \sigma^2)$ is returned when the updated pdf is propagated through the model.

[1] T. Butler and J. Jakeman and T. Wildey, Combining Push-Forward Measures and Bayes' Rule to Construct Consistent Solutions to Stochastic Inverse Problems, SIAM Journal on Scientific Computing, 40, A984-A1011 (2018).

[2] We can also assume different distributions for PH.

The following figure describes the relation between cation density, model and PH density.



The methodology of data consistent inversion[3] guarantees that when a sample of the updated pdf of cation is put in the model input, there is a probability of 99.73% that the predicted PH will be within three standard deviations around μ^* (grey area). It is straightforward to get the conclusion that the smaller σ we choose in $N(\mu^*, \sigma^2)$, the more accurate the PH will be.

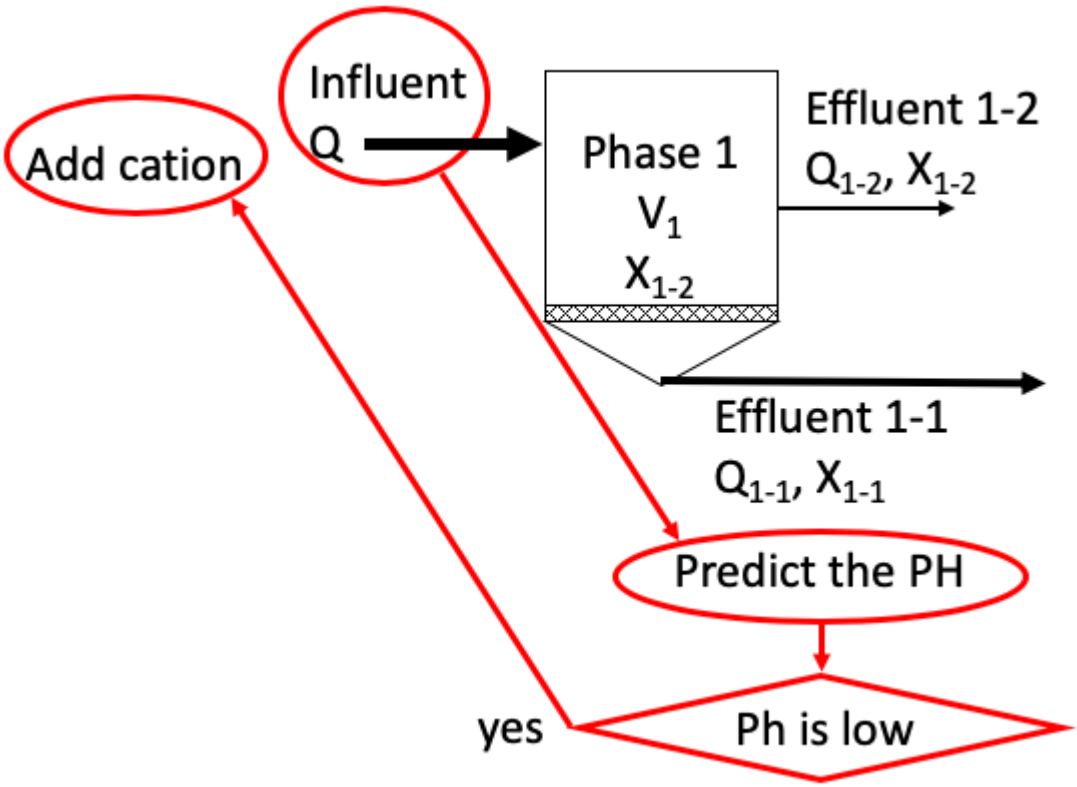
PH Control for One-Phase and Two-Phase Reactor

Phase 1 of the digester represents the condition where membrane blocks the solid materials that are part of the effluent. Here we setup the corresponding SRT to define the time required for the solids to dilute in phase 1. As mentioned above, during this phase VFAs are produced, which reduces the ph of the digester.

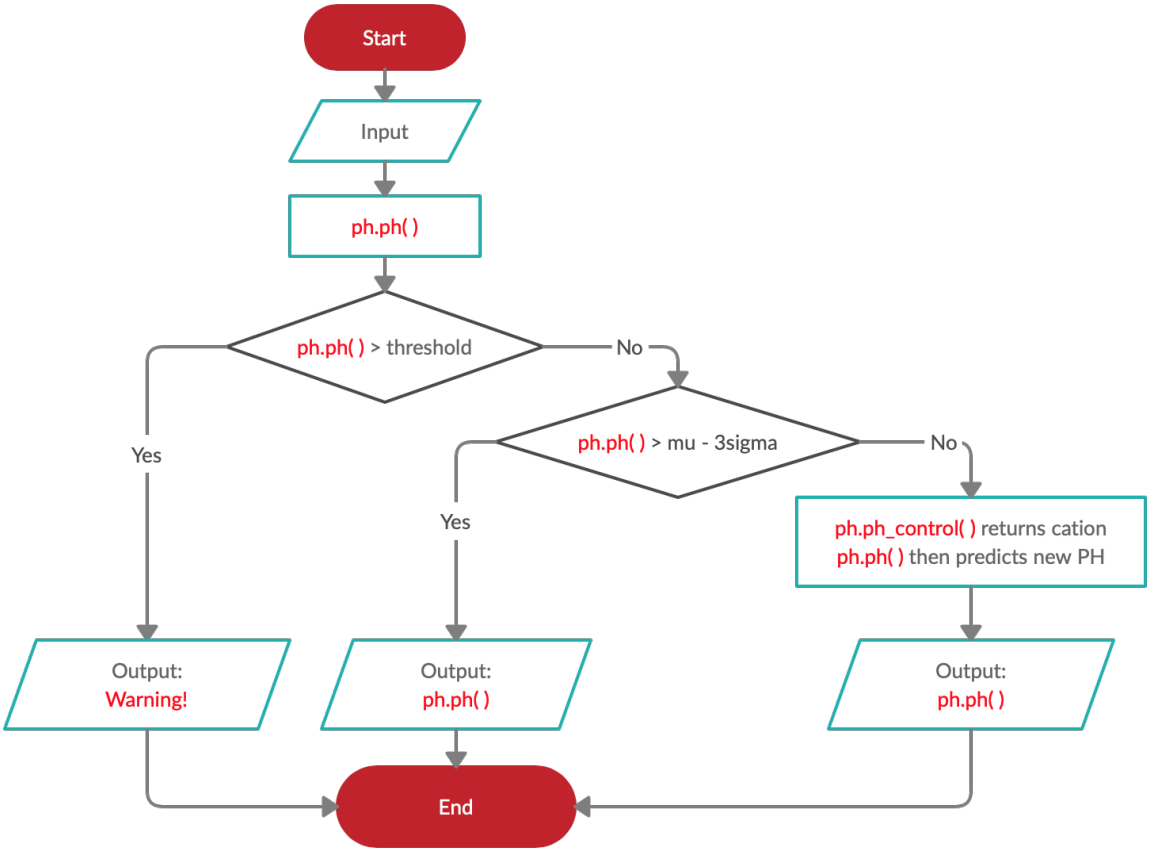
To prevent the ph from further decreasing, we developed the ph-control algorithm that allows us to adjust the ph at the beginning of each phase. Implemented in this study ph-control method (see `ph_control` function in `adm1f_utils.py`) is designed to implement data consistent inversion method in PH control (described in the section below). The `ph` method can be used to predict the PH value. The `ph_control` method is used to return the cation concentration to be changed in the input in order to reach the target PH value (μ^*). σ is used to denote the standard deviation we allow the PH value to vary around μ^*.

The following figure describes the basic strategy used for PH control

[3] Assumptions are not discussed here for the ease of explanation.



The idea can also be explained in the following flow chart



Similar idea can then be applied to the two-phase reactor. We can configure the two-phase reactor where

volume, flow rate, t_resx (t_resx = SRT-HRT) of each phase can be set manually to simulate the real reactor. The implementations in both phase 1 and 2 are similar to the above implementation in the one-phase reactor, the only difference is that the output of phase 1 is now the input of phase 2.

In the *Ipython notebook*, `reactor_cat(target_1, target_2, **kwargs)` function is used to calculate the corresponding cation for each phase in order to control the PH level for each phase. Note that the target PH for phase 1 and 2 (target_1, target_2) can be set different when using this function under the corresponding configurations.

ADM1F_SRT: Ph control method

The `ph control` method was developed by Wenjuan Zhang and uses Data Consistent Inversion Method.

Authors: Wenjuan Zhang and Elchin Jafarov

[1]:
```
import os
import adm1f_utils as adm1fu
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
%matplotlib inline
```

1. Relation between cation and PH

Here we explore the cation/ph relationships using diffrerent configurations. Note, based in the results will be different based on the ADM1F code version (i.e. original or SRT).

[2]:
```
# navigate to simulations folder
os.chdir('../../simulations')
```

[3]:
```
# Configuration of the one-phase reactor
config_default = {'Vliq':3400, 't_resx':0, 'Q':134}
config1 = {'Vliq':340, 't_resx':1.5, 'Q':618}
config2 = {'Vliq':3400, 't_resx':700, 'Q':618}
```

Configurations

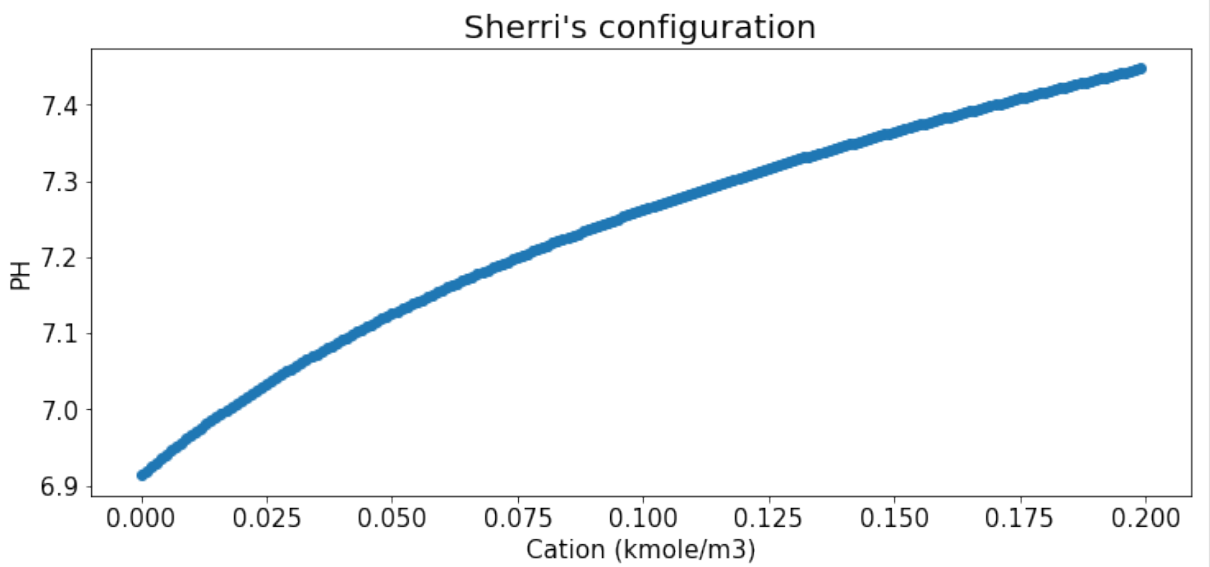
Configuration	Vliq (m^3)	t_resx (d)	Q (m^3/d)
Sherri's (default)	3400	0	134
Phase 1	340	1.5	618
Phase 2	3400	700	618/—

where t_resx = SRT - HRT

[4]:
```
# check if file exsits read from file
# otherwise run the simulations with different cations `cat_test`
cat_test = [i*0.001 for i in range(200)]

filename='data/no-configuration.dat'
if adm1fu.check_filename(filename):
    ph_test = np.loadtxt(filename)
else:
    ph_test = [adm1fu.ph(i,verbose='off',**config_default)[0] for i in cat_test]
    np.savetxt(filename, ph_test, fmt='%5.6f')
```

Relation b/t cation and Ph under Sherri's config

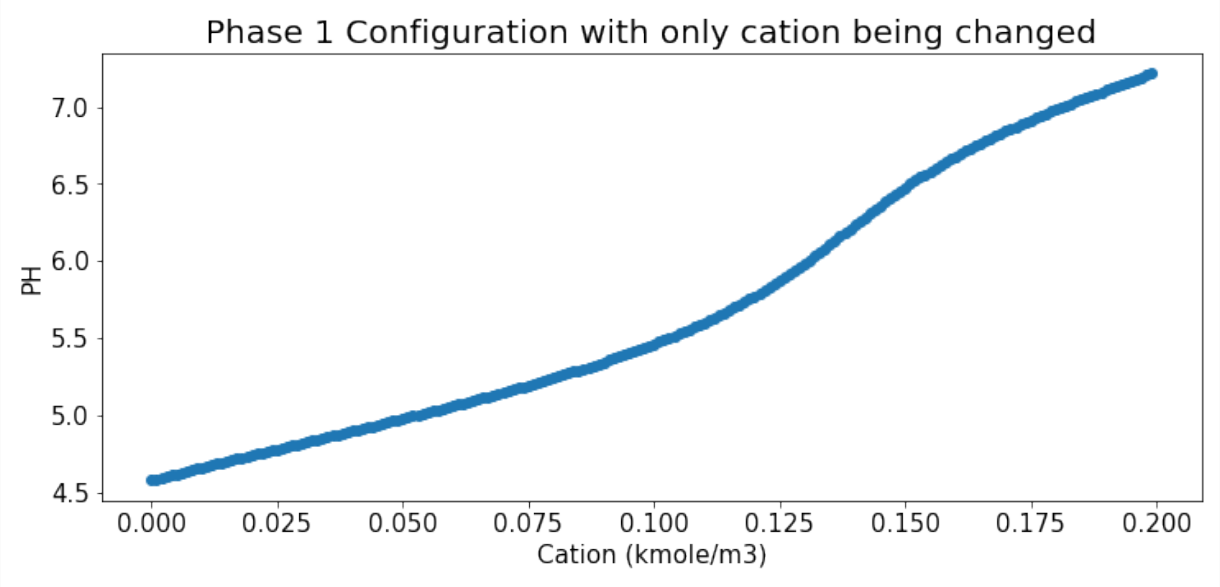
[5]:
```
# Relation b/t cation and Ph under Sherri's config
plt.figure(figsize=(12,5))
plt.scatter(cat_test, ph_test)
plt.ylabel('PH',fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.xlabel('Cation (kmole/m3)',fontsize=15)
plt.title("Sherri's configuration",fontsize=20);
```



Relation b/t cation and Ph under Phase 1 config

[6]:
```
filename='data/configuration1.dat'
if adm1fu.check_filename(filename):
    ph_test_config1 = np.loadtxt(filename)
else:
    ph_test_config1 = [adm1fu.ph(i, verbose='off', **config1)[0] for i in cat_test]
    np.savetxt(filename, ph_test_config1, fmt='%5.6f')
```

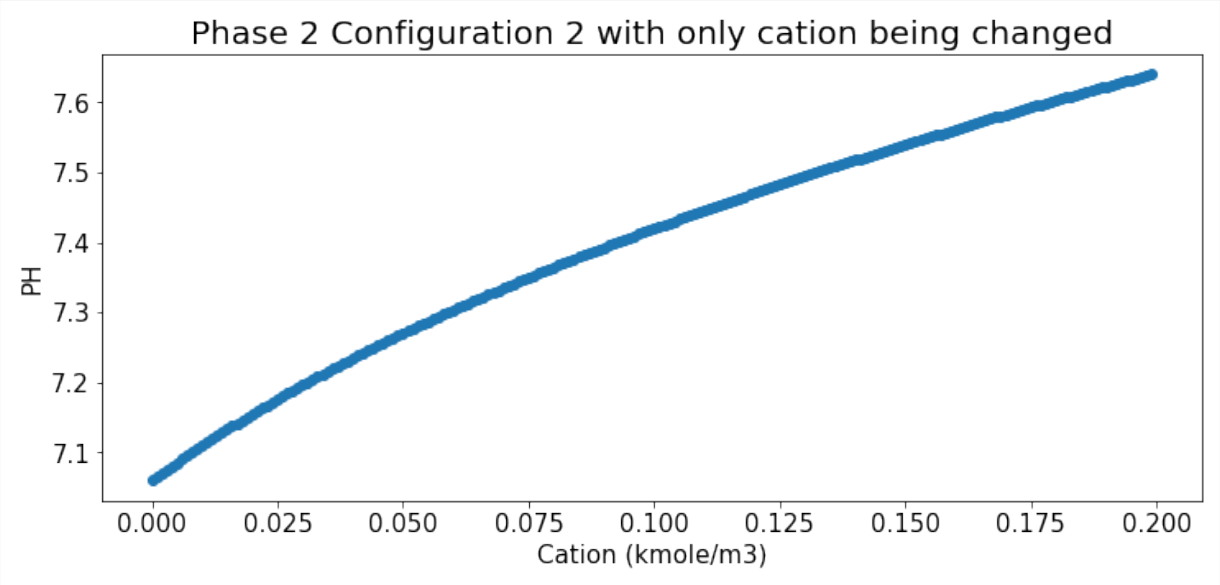
[7]:
```
plt.figure(figsize=(12,5))
plt.scatter(cat_test, ph_test_config1)
plt.ylabel('PH',fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.xlabel('Cation (kmole/m3)',fontsize=15)
plt.title("Phase 1 Configuration with only cation being changed",fontsize=20);
```



Relation b/t cation and Ph under Phase 2 config

[8]:
```
filename='data/configuration2.dat'
if adm1fu.check_filename(filename):
    ph_test_config2 = np.loadtxt(filename)
else:
    ph_test_config2 = [adm1fu.ph(i, verbose='off', **config2)[0] for i in cat_test]
    np.savetxt(filename, ph_test_config2, fmt='%5.6f')
```

[9]:
```
plt.figure(figsize=(12,5))
plt.scatter(cat_test, ph_test_config2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.ylabel('PH',fontsize=15)
plt.xlabel('Cation (kmole/m3)',fontsize=15)
plt.title("Phase 2 Configuration 2 with only cation being changed",fontsize=20);
```



2. PH: one-phase reactor

[10]:
```
old_ph0 = adm1fu.ph(0)
print('Predicted PH is {} if using the original cation value {}'.format(old_ph0[0], old_
↪ph0[1]))
```

```
Reactor run, ph phase-one:
$ADM1F_EXE -steady -influent_file influent_cur.dat
Predicted PH is 6.91367 if using the original cation value 0
```

[11]:
```
old_ph1 = adm1fu.ph(0, **config1)
print('Predicted PH is {} if using the original cation value {}'.format(old_ph1[0], old_
↪ph1[1]))
```

```
Reactor run, ph phase-one:
$ADM1F_EXE -steady -influent_file influent_cur.dat -Vliq 340 -t_resx 1.5
Predicted PH is 4.57781 if using the original cation value 0
```

[12]:
```
old_ph2 = adm1fu.ph(0, **config2)
print('Predicted PH is {} if using the original cation value {}'.format(old_ph2[0], old_
↪ph2[1]))
```

```
Reactor run, ph phase-one:
$ADM1F_EXE -steady -influent_file influent_cur.dat -Vliq 3400 -t_resx 700
Predicted PH is 7.05983 if using the original cation value 0
```

Set Target: Let's calculate the amount of cation needed by the one-phase reactor to match required `ph` targets using Data Consistent Inversion method.

[13]:
```
class target:
    def __init__(self,ph,sig):
        self.ph = ph
        self.sig = sig
    def pdf(self,x):
        return norm.pdf(x,self.ph,self.sig)
```

[14]:
```
# Give the necessary information
# target_ph = 6.5       # target_ph: target PH value, target_sig: allow some variations around␣
↪target PH
target_sig = 0.01     # The smaller this value is, the more accurate we will get in the end
sample_size = 100

infl_path = 'influent.dat'
params_path = 'params.dat'
ic_path = 'ic.dat'

## Use data consistent inversion method to return the needed cation to get the target PH
init_sample = np.random.uniform(0,0.2,sample_size)   #the more samples we generate, the more␣
↪accurate we will get in the end
```

[15]:
```
target72 = target(7.2,target_sig)
target73 = target(7.3,target_sig)
target75 = target(7.5,target_sig)
```

Target 1: target_ph=7.2 with Sherri's (default) configuration

[16]:
```
## ph_control accepts target, initial sample, number of cation values and file path of each␣
↪input file
## ph_control return the needed cation to get the target PH
cat_tar72_dc = adm1fu.ph_control(target72,init_sample,1,infl_path,params_path,ic_path,verbose=
↪'off', **config_default)
```

(continues on next page)

(continued from previous page)

```
# Print out the Needed Cation value!!
print('The amount of cation in the reactor should be:', cat_tar72_dc[0], 'kmole/m3')
```

```
Predicted PH is 6.91367
The amount of cation in the reactor should be: 0.07135066533871785 kmole/m3
```

[17]:
```
[adm1fu.ph(i, **config_default) for i in cat_tar72_dc]
```

```
Reactor run, ph phase-one:
$ADM1F_EXE -steady -influent_file influent_cur.dat -Vliq 3400 -t_resx 0
```

[17]:
```
[(7.18975, 0.07135066533871785)]
```

Target 2: target_ph=7.2 with configuration 1

[18]:
```
## ph_control accepts target, initial sample, number of cation values and file path of each␣
↪input file
## pph_control return the needed cation to get the target PH
cat_tar72_c1 = adm1fu.ph_control(target72,init_sample,1,infl_path,params_path,ic_path,verbose=
↪'off', **config1)

# Print out the Needed Cation value!!
print('The amount of cation in the reactor should be:', cat_tar72_c1[0], 'kmole/m3')
```

```
Predicted PH is 4.57781
The amount of cation in the reactor should be: 0.19737738732010346 kmole/m3
```

[19]:
```
[adm1fu.ph(i, **config1) for i in cat_tar72_c1]
```

```
Reactor run, ph phase-one:
$ADM1F_EXE -steady -influent_file influent_cur.dat -Vliq 340 -t_resx 1.5
```

[19]:
```
[(7.19608, 0.19737738732010346)]
```

Target 3: target_ph=7.2 with configuration 2

[20]:
```
## ph_control accepts target, initial sample, number of cation values and file path of each␣
↪input file
## ph_control return the needed cation to get the target PH
cat_tar72_c2 = adm1fu.ph_control(target72,init_sample,1,infl_path,params_path,ic_path,verbose=
↪'off', **config2)

# Print out the Needed Cation value!!
print('The amount of cation in the reactor should be:', cat_tar72_c2[0], 'kmole/m3')
```

```
Predicted PH is 7.05983
The amount of cation in the reactor should be: 0.031198904067240532 kmole/m3
```

[21]:
```
[adm1fu.ph(i, **config2) for i in cat_tar72_c2]
```

```
Reactor run, ph phase-one:
$ADM1F_EXE -steady -influent_file influent_cur.dat -Vliq 3400 -t_resx 700
```

[21]:
```
[(7.20101, 0.031198904067240532)]
```

3. PH: two-phase reactor

PH control for both phase 1 and phase 2

reactor_cat(target_1=target1, target_2=target2, Q1=1, Vliq1=1, t_resx1=1, Q2=1, Vliq2=1, t_res2=1)

PH control for just phase 1 in two-phase reactor

reactor_cat(target_1=target1, Q1=1, Vliq1=1, t_resx1=1, Q2=1, Vliq2=1, t_res2=1)

[22]:
```
## Configuration of two-phase reacotr
# config12 = {"Vliq1":340, "Vliq2":3400, "t_resx1":1.5, "t_resx2":700, "Q1":618, "Q2":618}
config12 = {"Vliq1":340, "Vliq2":3400, "t_resx1":1.5, "t_resx2":700, "Q1":618}
```

target_ph1=7.5, target_ph2=7.2 with default configuration12

[23]:
```
config12 = {"Vliq1":340, "Vliq2":3400, "t_resx1":1.5, "t_resx2":700, "Q1":618}
adm1fu.reactor2_cat(init_sample,target_1=target75,target_2=target72,verbose='off',**config12)
```

```
verbose: off
Predicted PH is 4.57781
$ADM1F_EXE -steady -Vliq 340 -t_resx 1.5 -influent_file influent_cur.dat
Phase 1, after changing cation to 0.197377 kmole/m3, new PH = 7.19608
Predicted PH is 7.10724

$ADM1F_EXE -steady -Vliq 3400 -t_resx 700 -influent_file influent_cur.dat
Phase 2, after changing cation to 0.017699 kmole/m3, new PH = 7.18887
```

target_ph1=7.5, target_ph2=None with default configuration12

[24]:
```
adm1fu.reactor2_cat(init_sample,target_1=target75,verbose='off',**config12)
```

```
verbose: off
Predicted PH is 4.57781
$ADM1F_EXE -steady -Vliq 340 -t_resx 1.5 -influent_file influent_cur.dat
Phase 1, after changing cation to 0.197377 kmole/m3, new PH = 7.19608

$ADM1F_EXE -steady -Vliq 3400 -t_resx 700 -influent_file influent_cur.dat
Phase 2, without changing cation, predicted PH = 7.65471
```

References

Footnote

Chapter 6

Join Us on Slack

If you have any questions or would like to be a developer, feel free to join our slack channel.

References

Acknowledgements

The research was supported by the U.S. Department of Energy, Office of Energy Efficiency and Renewable Energy, Bioenergy Technologies Office, under contract DE-AC02-06CH11357.