TECHNISCHE UNIVERSITÄT
CHEMNITZ

AUTOMATIC CONTROL &
SYSTEM DYNAMICS

Faculty of Electrical Engineering and Information Technology
Automatic Control and System Dynamics Lab

Software documentation

for a nutrient mixing optimization algorithm

Authors:
Patrick Nestler
Alexander Kobelski
Stefan Streif

Contents

1 Prerequisites

This documentation servers a detailed documentation for an algorithm that calculates optimal inputs to mix a desired nutrient solution. The algorithm is being published with the article Kobelski et. al 2024 (currently under review; doi follows as soon as available). When using the algorithm, please cite this article. References made to sections, tables, and equations are related to this article. The aim is to facilitate the usage of the program and to demonstrate how to adapt it to other systems.

In the first part of the documentation it is explained what prerequisites are needed to run the program. Then, a more detailed explanation of the program-structure is given. Afterwards the xml-files, which contain the parameters of the optimal control problem, are described in detail. It is also explained which parameters are mandatory and which are optional to run the program. The last part of the documentation then shows how the given parameters are used to set up the optimization.

1.1 Running and debugging the program code

Prerequisites The program is implemented in the Python file `Static_NutrientMixing.py` and can be run like any other `.py` file. First of all, a Python interpreter has to be installed. For the simulation study in the article Python version 3.11 was used. It is recommended to use the same or a newer version for the interpreter.

For the code-implementation several Python packages are used, see Tab. 1. These libraries contain pre-built functions for reading and writing of, e.g., xml-files as well as the framework for the optimal control problem presented in Sec. 2.2. For easier access to these libraries a package-manager may need to be installed.

There are multiple ways to provide a Python interpreter and the needed packages, but one free to use software which combines both is the distribution **Anaconda**. It has an in-built Python interpreter of version 3.11 and also contains all required packages.

For editing and testing/debugging the code an additional Python editor is needed. Free software **Visual Studio Code** was used for the program implementation and therefore the descriptions in this documentation are based on this IDE.

Running the code With Anaconda installed, running the program can be done with the windows command prompt. The necessary steps are:

1. in the command prompt, change to directory where Anaconda is installed `...\anaconda3`
2. move further into directory `...\anaconda3\condabin`
3. run batch file `conda_hook.bat`
4. activate conda base environment with `conda activate ...\anaconda3`

Table 1: Packages used for implementing the program.

Package	Use in the program
numpy	easily generate arrays of ones and arrays of zeros
matplotlib	generate plots to visualize the results of the optimization
math	use `isnan()` function to check if given parameter is set in xml-files
pandas	read parameters from xml-files and convert data to Python dictionaries; write the optimization results in a xml-file
casadi	set up optimization problem symbolically; provide interface to NLP solver, e.g. IPOPT
tkinter	generate interactive error messages

5. change to directory where program code of `Static_NutrientMixing.py` is saved
6. type `python Static_NutrientMixing.py`.

The first four steps may be skipped when using the built in Anaconda command prompt.

Creating a new Conda environment Note, that after step 4 the program will be run in the base environment of Anaconda which contains all the needed packages listed in Tab. 1, but also many more libraries. To run the program in a more compact virtual environment, that only contains the needed packages, a new environment can be created via the Anaconda prompt:

1. in Anaconda prompt, change to directory where new environment should be created
2. type `conda create -p <path>\<env_name> Python=3.11 numpy matplotlib pandas casadi`

(opt) if CasADi is not found in the default channel, type `conda config --add channels conda-forge`

The `math` and `tkinter` libraries do not have to be added in a new environment since they are already part of the base installation of Python.

Editing the code Editing the code can be done with the IDE Visual Studio Code (VSC). Open the file using VSC or, using VSC, navigate to its directory. Afterwards, choose the virtual environment by opening the command palette (`ctrl+shift+p`) and typing "Python: Select Interpreter". Visual Studio Code then lists all available virtual environments; choose one that contains the packages of Tab. 1. The third and last step is to then open the code-file `Static_NutrientMixing.py` and click on run.

2 Structure of the program

The different steps of the program are presented in Fig. 1. In the first step, the program

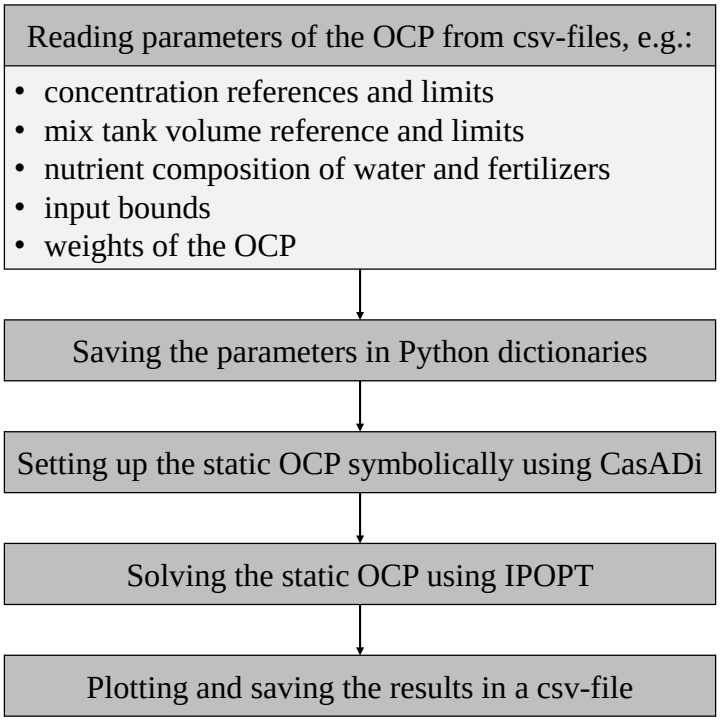

Figure 1: Schema of the program.

reads the parameters from the xml-files and saves them in Python dictionaries with the help of functions from *pandas*. Afterwards, the optimization problem is set up symbolically with CasADi and solved with IPOPT. At the end, the results are visualized and saved in a separate xml-file. In the following subsections all these parts will be explained in detail according to the order in Fig. 1.

2.1 Contents of the input xml-files

The optimization parameters are stored as float-type numbers with a **point "."** as the decimal separator in four different tables that are each realized as xml-file:

- `Mix_Tank_Parameters.xml`
- `Water_Use.xml`
- `Nutrient_Concentrations.xml`
- `Fertilizers.xml`.

In the xml-files the rows and columns of the table are specified by tags. Each row of the table starts with the tag `<row>` and ends with `</row>`. In each row the parameters are separately encapsulated in the tags `<header_name>` and `</hander_name>`. These tags are the same in every row for their respective parameter and in that way the columns of the table are realized. If one parameter cannot be stated, the space between the start and end tag can be left empty. For an easier notation,they are referred to as rows and columns when describing the contents of the xml-files in the further documentation.

Since the program accesses the data of every column of the tables with the associated name of the header, it is important to not change the headers of existing columns of the tables. In contrast to that, the order of columns within one file does not matter. In the following paragraphs the contents of every xml-file will be explained as well as which inputs are mandatory and which are optional.

All the parameters that are stated in the files are by default assumed to have units L, g and $\mathrm{g\,L^{-1}}$. In general, the user can use other units as well, but by doing so the program code has to be adjusted accordingly. The balancing weights ω_{bal} (see Sec. 2.2.1) assume that the volumes are given in L and the fertilizer masses are given in g. There are fallback values `V_max_fallback` that are used for constraining the maximum value that is stored in the tank after mixing a new solution. This value is given in L as well and has to be adjusted when changing the units of the parameters.

Another important mark is that the units of the parameters have to be consistent for every input file, e.g. if the user changes a unit to mL and the unit for masses is kg, the concentrations in the file `Nutrient_Concentrations.xml` has to be given in $\mathrm{kg\,mL^{-1}}$.

Note that in the article, many of the contents of the xml-files were left out for a more compact and easier to follow presentation of the contents.

The xml-file Mix_Tank_Parameters.xml The first file `Mix_Tank_Parameters.xml` contains parameters that are associated with the water volume that is filled inside the mixing tank. By default all values of this file are assumed to be given in L. The contents of this file are given in List. 1.

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <row>
    <Current_Value>10</Current_Value>
    <Target_Value></Target_Value>
    <Max_Value>900</Max_Value>
    <Min_Value>600</Min_Value>
    <Max_Tank_Volume>1000</Max_Tank_Volume>
  </row>
</root>
```

Listing 1: One row of content of Mix_Tank_Parameters.xml

Of the parameters given in this file, only the entry of the `Current_Value` is mandatory, because the program has to know the initial state $V_{\mathrm{MW,pre}}$ of the system. The other parameters are used to set the constraints of the water volume that is present in the mixing tank after realizing a new nutrient solution. How the bounds of these constraints are set depending on the input of parameters is explained in Sec. 2.3. For both the maximum and minimum constraints, a fallback value is given in the program. Thus the entries for parameters `Target_Value`, `Max_Value`, `Min_Value` and `Max_Tank_Volume` are optional.

The xml-file Water_Use.xml The file `Water_Use.xml` contains data of the water sources that are used for mixing as well as parameters that affect the use of the different water sources.

By default all values of this file are assumed to be given in L. The contents of this file are given in List. 2.

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <PlantDrain>
    <Use_Source>1</Use_Source>
    <Max_Value>400</Max_Value>
    <Min_Value>0</Min_Value>
    <Current_Value>2000</Current_Value>
    <MaxTank_Value>4000</MaxTank_Value>
    <Source_Weight>-1</Source_Weight>
  <PlantDrain>
</root>
```

Listing 2: One row of content of Water_Use.xml

In the first column under the header `Water_Source` the identifiers of the water sources are given (usually as a string). It is mandatory to enter an identifier for every source. This allows the program to link the concentrations of the inputs that are given in the file `Nutrient_Concentrations.xml` to the correct water source.

The second parameter `Use_Source` (string) determines if the given water source should be used in the optimization. Value "1" means the source is used. Best practice is to set to "0" if it should not be used, but any other value works as well.

The following two parameters `Max_Value` and `Min_Value` realize the constraints for the optimization variables that are given by the water sources. Both values are optional, since the program uses fallback values if they are not entered. The fallback values of these constraints are again given in Sec. 2.3.

The remaining three parameters `Current_Value`, `MaxTank_Value` and `Source_Weight` are used to calculate the individual weights $\omega_{\text{water},i}$ of every water source in the error-term $e_{\text{water}}(u_{\text{V}})$ (Eq. (14)) of the objective function. These parameters are optional. If there is no weight parameter entered for a specific source, the corresponding weight is set equal to one. The weights $\omega_{\text{water},i}$ are furthermore amplified by the `Current_Value` and `MaxTank_Value` parameters. These parameters describe the state of the storage tanks where the water sources are stored. If the use of a water source is rewarded in the objective function (by setting the parameter `Source_Weight` as a negative number), that reward gets increased when a high percentage of the tank is filled. In contrary, if the use of a water source is penalized in the cost-function, that penalization gets decreased when a high percentage of the tank is filled. The mathematical formulation of this is stated in Sec. 2.3. If the water source is not stored in a tank, e.g. tap water or if the water sources should not be penalized depending on the fill status of the respective storage tank, these two parameters can be left blank. The weight $\omega_{\text{water},i}$ is then equal to the value given for the parameter `Source_Weight`.

The xml-file Nutrient_Concentrations.xml The file `Nutrient_Concentrations.xml` contains all the values related to nutrient concentrations as well as the identifiers of the considered nutrients, e.g., their chemical symbols. By default all values of this file are assumed to be given in $\mathrm{g\,L^{-1}}$. The contents of the file are given in List. 3.

In the first column the identifiers of the nutrients are stated. These identifiers are important to combine the nutrient masses realized by the water sources and fertilizers and thus correctly calculate the resulting nutrient masses of in the mixing tank. It is mandatory to enter an identifier for every nutrient that is to be used in the optimization.

In the second column the parameter `Nutrient_Weight` is stored. This parameter sets the weights $\omega_{\text{N}_{\text{ref}},q}$ if the reference value is greater then zero or $\omega_{\text{N}_{\text{min}},q}$ if the reference value equals

zero. Entering this parameter is optional. If no value is given for one nutrient, the corresponding weight is set equal to one.

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <row>
    <Nutrient>NO3</Nutrient>
    <Nutrient_Weight>1</Nutrient_Weight>
    <Start_Concentration>0.1225</Start_Concentration>
    <Target_Concentration>0.1225</Target_Concentration>
    <Max_Concentration></Max_Concentration>
    <Min_Concentration>0.11025</Min_Concentration>
    <FishDrain_Concentration>0.094</FishDrain_Concentration>
    <Rain_Concentration>0.0004</Rain_Concentration>
    <Tap_Concentration>0.0063</Tap_Concentration>
    <PlantDrain_Concentration>0.037</PlantDrain_Concentration>
  </row>
</root>
```

Listing 3: One row of content of Nutrient_Concentrations.xml

The third parameter `Start_Concentration` is the concentration of nutrients that is given in the tank before mixing a new solution. It is mandatory to state a value as it is necessary for the initialization of the optimization.

Parameter `Target_Concentration` is used to calculate the desired mass of nutrients that should be present in the tank after mixing a new solution (see Eq. (12)). It is therefore the basis of calculating the error terms $e_{N_{MW,ref}}(u)$ and $e_{N_{MW,min}}(u)$ in the objective function. Nevertheless, the parameter is optional and if the target concentration is not given for a specific nutrient it will simply not be considered in the objective function.

The next two parameters `Max_Concentration` and `Min_Concentration` realize the bounds of the nutrient concentrations after mixing a new solution. Both parameters are optional. If no `Max_Concentration` is given and `Target_Concentration` is not greater than zero, i.e. it is a harmful ion, a maximum concentration of $200\,\mathrm{mg\,L^{-1}}$ is assumed to calculate weights.

In the last columns of this xml-file the nutrient compositions of the water sources are stored. If one value of these concentrations is left out, the program assumes the concentration in the respective water source to be equal to zero. Entering these parameters is therefore optional. One **important** thing to note is that for every water source that is stated in `Water_Source.xml` there has to be one corresponding column of concentrations in this file. Furthermore, if the identifier given for the water source is `<source_name>`, the header of the column that stores the concentrations of this source must be `<source_name>_Concentration`.

The xml-file Fertilizers.xml The last file containing the input parameters of the program is `Fertilizers.xml`. The contents of the file are given in List. 4.

In the first column of the file once again the identifiers of the fertilizers are stated. These identifiers do not have a use in the program but make the xml-table way more clearly to read and thus it is recommended to set meaningful identifiers for every fertilizer.

The next parameter `Use_Source` determines if the corresponding fertilizer should be used in the optimization. If the values is set to "1" the fertilizer will considered and for any other value it won't be used.

The Parameter in the third column `Max_Value` sets the upper bound of the fertilizer consumption in the optimization. Stating these parameters is optional, because the program uses a fallback value (see Sec. 2.3) if no parameter is given in the xml-file. By default this parameter is assumed to be given in g.

The parameters in `Fertilizer_Weight` set the individual weights $\omega_{\text{fert},f}$ of every fertilizer in the error-term $e_{\text{fert}}(u_{\text{N}})$ (Eq. (15)) of the objective function. All the parameters are optional. If there is no weight parameter entered for a specific fertilizer, the corresponding weight is set equal to one.

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <row>
    <Fertilizers>Ca_NO3_2</Fertilizers>
    <Use_Source>1</Use_Source>
    <Max_Value>500</Max_Value>
    <Fertilizer_Weight>1</Fertilizer_Weight>
    <NO3>15.5</NO3>
    <NH4>0</NH4>
    <P>0</P>
    <K>0</K>
    <Ca>19.9</Ca>
    <Mg>0</Mg>
    <S>0</S>
    <Fe>0</Fe>
  </row>
</root>
```

Listing 4: One row of content of Fertilizers.xml

The remaining parameters of the file describe the nutrient composition of each fertilizer. This composition is given as the percentage of nutrient mass that is added to the nutrient solution after putting a specific mass of fertilizers into the solution. The values are therefore given in %. The unit of these parameter should not be changed. To assign the nutrient mass contributed by the fertilizers to the correct nutrient, the headers of the composition parameters must be equal to the nutrient identifiers stated in `Nutrient_Concentrations.xml`. However not all stated nutrient must be present as header. If a nutrient is missing in the `Fertilizers.xml` file, its composition parameter is assumed to be equal to zero.

2.2 Reading the parameters with pandas

In the first part of the program the parameters that are stored in the xml-files are transformed to Python dictionaries with the help of the Python library pandas. This can be done with the `pandas.read_xml()` function, that reads all data of the xml-file directly into a dataframe, where in case of the files `Water_Use.xml`, `Nutrient_Concentrations.xml` and `Fertilizers.xml` the identifiers are used as the labels of the data. Furthermore, the identifiers are stored in a list of strings. In the remaining program these labels are then used to access the parameters by iterating through the list of identifiers and for every entry search the corresponding label in the dictionary.

The first thing that is done after reading the data is to iterate over the `"Use_Source"` column of the dataframes of `Water_Use.xml` and `Fertilizers.xml` and check if any given source should be used in the optimization. If any element of `"Use_Source"` is not equal to one, the corresponding row in the frame gets deleted from the frame with the use of the pandas function `drop()` and by specifying the row-label.

After the frames are cleaned up one column with the key `"Indexes"` is added to the dataframes of `Water_Use.xml` and `Fertilizers.xml`. In this column an index is assigned to every label and by that the water sources and fertilizers can easily be linked to the respective element in the vector of optimization variables.

Converting the dataframes to Python dictionaries can be done by calling the method `to_dict()` of the dataframe. As a result four dictionaries are obtained that each contain all the parameters that were stated in the xml-files.

2.3 Setting up and solving the optimization problem

The main part of the program is setting up the optimization problem described in section 2.2. The whole problem formulation is done symbolically using the `SX` datatype of CasADi to initialize the optimization variables `u_opt`. Every element of this list of symbolic values can be linked to one input of the mixing tank system with the indexes that are established in the data frames of `Water_Use.xml` and `Fertilizers.xml`.

With the optimization variables initialized, the next step is to symbolically implement equations (8) to (23) to create the objective function as well as the constraints of the optimization. This is done in the following two paragraphs, first for the objective and afterwards for the constraints. In the second paragraph it is also explained in detail how the boundaries of the constraints are set depending on the inputs in the xml-files. A third paragraph at the end of this chapter then describes how the solver IPOPT is initialized and called to solve the generated optimization problem.

Symbolically generating the objective function The first step is to symbolically calculate the volume $V_{\mathrm{MW,post}}$ that is stored in the mixing tank after the solution is prepared. The calculation is done in a for-loop that is shown in the List. 5 below.

```
1    V_post = V_0
2    for i from 0 to i_max-1
3      ind_water = WaterSource_dict["Indexes"][WaterSource_list[i]]
4      V_post = V_post + u_opt[ind_water]
5    end
```

Listing 5: For-loop for calculating V_post

Post-mixing volume $V_{\mathrm{MW,post}}$ is set equal to $V_{\mathrm{MW,pre}}$, which is the water volume that is currently stored in the mixing tank. The corresponding float-type value is located in the dictionary of `Mixing_Tank_Parameters.xml` under the header `Current_Volume`. In the for-loop the symbolic optimization variables are then added cumulatively. If for example $i_{\max} = 3$ and $V_{\mathrm{MW,pre}} = 10\,\mathrm{L}$, the resulting expression for $V_{\mathrm{MW,post}}$ will be

`V_post=10+u_opt_0+u_opt_1+u_opt2`.

The next step is to generate a symbolic expression for $N_{\mathrm{MW,post}}$. For this a for-loop iterates over every nutrient q and by that the elements of $N_{\mathrm{MW,post}}$ are calculated separately for every single nutrient. Inside this loop the symbolic expressions of nutrient masses $N_{\mathrm{MW,q,post}}$ are then again generated cumulatively via for-loops. The corressponding code is presented in List. 6 as pseudocode.

```
1    for q from 0 to q_max-1
2
3      N_post_water = V_0 * Nut_dict["Start_Concentration"][Nut_list[q]]
4      for i from 0 to i_max-1
5        ind_water = WaterSource_dict["Indexes"][WaterSource_list[ind_water]]
6        c_iq = Nut_dict[WaterSource_list[ind_water]+"_Concentration"][Nut_list[
       q]]
7        if isnan(c_iq) == 0
8          N_post_water = N_post_water + u_opt[ind_water] * c_iq
```

```
 9         end
10       end
11
12       N_post_fertil = 0
13       for f from 0 to f_max-1
14         ind_fert = Fertilizer_dict["Indexes"][Fertilizer_list[ind_fert]]
15         k_fq = Fertilizer_dict[Nut_list[q]][Fertilizer_list[ind_fert]]
16         if Nut_list[q] in Fertilizer_dict and isnan(k_fq) == 0
17           N_post_fertil = N_post_fertil + u_opt[ind_fert] * k_fq/100
18         end
19       end
20       N_post_nut = N_post_water + N_post_fertil
21
                                                    ⋮
```

Listing 6: For-loop for calculating N_post

The nutrient mass realized by mixing the fertilizers and water sources are calculated separately via two for-loops and are then added together to the resulting nutrient mass. Note, that an optimization variable and thus the corresponding water source or fertilizer only gets added to the resulting expression if the value `c_iq` – that is the concentration of nutrient i in water source q – or `k_fq` – that is the mass fraction of nutrient i in fertilizer f – is a number-type data. The resulting symbolic expression of $N_{\mathrm{MW,q,post}}$ is then pretty similar to that of $V_{\mathrm{MW,post}}$. The only difference is that the optimization variables are multiplied by either the factor `c_iq` or `k_fq`.

The for-loop over the nutrients doesn't end with the calculation of $N_{\mathrm{MW,q,post}}$. The resulting expression is immediately afterwards used to symbolically calculate the error $e_{N_{MW,q,ref}}(u)$ or $e_{N_{MW,q,min}}(u)$. This is shown in List. 7.

```
23
                                                    ⋮
24       if isnan(Nut_dict["Target_Concentration"][Nut_list[q]]) == 0:
25         N_ref = Nut_dict["Target_Concentration"][Nut_list[q]] * V_post
26         w_Nut = 1
27         if isnan(Nut_dict["Nutrient_Weight"][Nut_list[q]]) == 0
28           w_Nut = Nut_dict["Nutrient_Weight"][Nut_list[q]]
29         end
30         if Nut_dict["Target_Concentration"][Nut_list[q]] != 0
31           e_NutRef = e_NutRef + w_Nut * (N_post_nut-N_ref)^2/N_ref^2
32         else
33           e_NutMin = e_NutMin + w_Nut * (N_post_nut/V_post)^2
34         end
35       end
36
                                                    ⋮
37     end
```

Listing 7: For-loop for calculating e_NutRef

Before calculating the error term $e_{\mathrm{N_{MW,ref},}q}(u)$ or $e_{\mathrm{N_{MW,min},}q}(u)$, the program checks if a reference concentration for the respective nutrient is given in `Nutrient_Concentrations.xml`. The corresponding header is `Target_Concentration`. Only then the error terms of one nutrient are calculated and added to the objective function. Ìf the parameter is given, the for-loop proceeds by calculating the reference mass via Eq. (12).

Afterwards, the weights for each individual nutrient are set. If no parameter for the weight is stated in `Nutrient_Concentrations.xml`, the default value $\omega_{\mathrm{N_{MW,ref},}q} = 1$ is used in the further computation.

At the end of the for-loop either $e_{\mathrm{N_{MW,ref},}q_{\mathrm{nut}}}(u)$ is computed by Eq. (11) or $e_{\mathrm{N_{MW,min},}q_{\mathrm{harm}}}(u)$ is computed by Eq.(13) – depending on the reference concentration being equal to zero or not.

The error terms of each nutrient are then again added cumulatively to construct the resulting error $e_{\mathrm{N_{MW,ref}}}(u)$ and $e_{\mathrm{N_{MW,min}}}(u)$.

After the error terms of the resulting nutrient concentrations are generated, the same is done for $e_{\mathrm{water}}(u_{\mathrm{V}})$ and $e_{\mathrm{fertil}}(u_{\mathrm{N}})$.

The corresponding code is presented in List. 8.

```
1    for i from 0 to i_max-1
2      w_water = 1
3      if isnan(WaterSource_dict["Source_Weight"][WaterSource_list[i]]) == 0:
4        w_water = WaterSource_dict["Source_Weight"][WaterSource_list[i]]
5        V_max = WaterSource_dict["MaxTank_Value"][WaterSource_list[i]]
6        V_curr = WaterSource_dict["Current_Value"][WaterSource_list[i]]
7        if isnan(V_max)==0 and isnan(V_curr)==0:
8        if w_water > 0:
9          w_water = w_water * (2*(V_max-V_curr)/V_max)
10       if w_water < 0:
11         w_water = w_water * (2 - 2*(V_max-V_curr)/V_max)
12       ind_water = WaterSource_dict["Indexes"][WaterSource_list[i]]
13       e_Water = e_Water + w_water * u_opt[ind_water]
14   end
```

Listing 8: For-loop for calculating e_water

First, the weights $\omega_{\mathrm{water},i}$ for each water source are set. If no parameter for a weight is given in `Water_Use.xml`, the program uses the default value $\omega_{\mathrm{water},i} = 1$. Then, the weights can be adjusted based on the percentage the respective storage tank is filled if the parameters `MaxTank_Value` and `Current_Value` are given. If the weight of a water source is positive, it increases the value of the cost function and is therefore penalized. The equation for adjusting the weights based on the ratio of current and maximum volume is

$$\omega_{\mathrm{water},i,\mathrm{new}} = \omega_{\mathrm{water},i,\mathrm{old}} \cdot 2 \cdot \frac{V_{\max} - V_{\mathrm{curr}}}{V_{\max}}. \tag{1}$$

Thereby $\omega_{\mathrm{water},i}$, and thus the penalty of the corresponding water source, is doubled if the storage tank is empty, and is set to zero if the storage tank is full. If $\omega_{\mathrm{water},i}$ is negative, its use is rewarded via the objective function. The equation for adjusting the weight in this case is

$$\omega_{\mathrm{water},i,\mathrm{new}} = \omega_{\mathrm{water},i,\mathrm{old}} \cdot (2 - 2 \cdot \frac{V_{\max} - V_{\mathrm{curr}}}{V_{\max}}). \tag{2}$$

By that the reward for using the water source is doubled if the storage tank is full and is set to zero if it is empty.

With the weights given, the symbolic expression for the error $e_{\mathrm{water}}(u_{\mathrm{V}})$ can easily be generated by multiplying all optimization variables of the water sources with the respective weights and then summing up the terms with a for-loop.

Computing $e_{\mathrm{fertil}}(u_{\mathrm{N}})$ is done in the same way. The weights are first set to their default value $\omega_{\mathrm{fert},f} = 1$, and if a parameter for the weight is given in `Fertilizers.xml` it is adjusted accordingly. The error $e_{\mathrm{fert}}(u_{\mathrm{N}})$ is then symbolically generated by multiplying the weights with the optimization variable of the respective fertilizer and summing up the terms, again with a for-loop. Since this section of the code is very similar to that of the water error, the code is not listed.

The last step for generating the objective function is to set the balancing weights according to Eq. (9) and generate the objective function as stated in Eq. (8).

Symbolically generating the constraints The second component of the static optimization problem are the constraints. In total there are three sets of constraints that need to be implemented in the program code.

The Bounds of optimization variables The first set of constraints are the lower and upper bounds of the optimization variables, i.e., the maximum and minimum allowed values of fertilizers and water sources for mixing. The values for the bounds of water sources are given in `Water_Use.xml` in the columns `Max_Value` and `Min_Value`. If one of these parameters is not given the program uses the default values $u_{\text{max,def}} = \infty$ and $u_{\text{min,def}} = 0$. For the fertilizers, only the maximum bound can be specified in `Fertilizers.xml` in the column `Max_Value`. If no parameter is stated, the program again uses $u_{\text{max,def}} = \infty$ as the upper limit. The upper and lower bounds are realized separately from the other constraints as two one-dimensional numpy arrays `u_max` and `u_min`. The reason for the separate statement is that the solver IPOPT treats bounds of variables different than other algebraic constraints. Variables `u_max` and `u_min` are initialized as arrays of zeros with

`u_max=np.zeros(Number_of_WaterSources+Number_of_Fertilizers)`.

The program then iterates over all water sources and afterwards over all fertilizers via a for-loop. If the parameters for the maximum or minimum bound are given, the respective element of `u_max` is set accordingly. Otherwise the default value is inserted.

Constraints on the Water Volume inside the mixing tank The rest of the algebraic constraints are expressed symbolically in the vector `h_mix`. The vector is initialized as an empty list. All algebraic expressions of the constraints are then one by one added to the vector with the Python function `append()`. All constraints are stated such that the lower bound of the resulting algebraic expression is always equal to zero and that the upper bound is infinite.

The first algebraic constraint is the water volume $V_{\text{MW,post}}$ in the mixing tank after the solution is prepared. In the program code this volume is expressed by the symbolic variable `V_post`, which was introduced in the previous paragraph. For the he mathematical formulation of the constraints see Eq. (20) and Eq. (21), where the latter equation is multiplied by minus one, such that both constraints have the lower limit of zero and no upper limit. The parameters $V_{\text{MW,post,max}}$ and $V_{\text{MW,post,min}}$ are specified by the user through the file `Mixing_Tank_Parameters.xml`. The program first checks if an input is given in the `Target_Value` column. If this is the case, $V_{\text{MW,post,max}}$ and $V_{\text{MW,post,min}}$ are set to:

$$V_{\text{MW,post,max}} = 1.01 \cdot V_{\text{MW,target}}, \tag{3}$$

$$V_{\text{MW,post,min}} = 0.99 \cdot V_{\text{MW,target}}. \tag{4}$$

In this case the upper and lower bounds guarantee that the resulting water volume in the mixing tank is very close to the reference value. If the target volume is not stated, the program checks if there is an input given in the columns `Max_Value` and `Min_Value`. If a parameter is available in `Min_Value`, $V_{\text{MW,post,min}}$ is set accordingly. If this input is also not given, the program uses $V_{\text{MW,pre}}$ as default value. The same goes for the maximum bound. If `Max_Value` is given, $V_{\text{MW,post,min}}$ is set to this value. Else, the program checks for a parameter in the column `Max_Tank_Volume`, and if this also does not exist, $V_{\text{MW,post,max}}$ is by default set to

$$V_{\text{MW,post,max}} = \max\left(V_{\text{MW,pre}} + 100\,\text{L}, 1000\,\text{L}\right). \tag{5}$$

Constraints on the nutrient concentrations inside the mixing tank The last set of constraints are the limits of the post-mixing nutrient concentrations in the mixing tank. In the program code, these constraints are generated at the end of the for-loop of List. 7, because $N_{\text{MW},q,\text{post}}$ is already available after computing $e_{N_{\text{MW,ref}}}(u)$. The constraints of the concentrations are given in Eq. (16) and Eq. (17), where the latter equation is again multiplied by minus one. Both are only implemented for a specific nutrient if the parameters for the maximum and minimum concentration are given in `Nutrient_Concentrations.xml` in the row of the corresponding nutrient identifier. The respective columns are `Max_Concentration` and `Min_Concentration`.

Solving the generated optimization problem For solving the stated optimization problem with IPOPT through the CasADi interface, the components of the optimization, i.e. the objective function `J_mix`, constraints `h_mix` and optimization variables `u_opt` first have to be combined in one Python dictionary `nlp_mix`. Then, some basic solver-options `opts_sol` are modified. Both variables are then used to initialize the solver IPOPT with the CasADi function `nlpsol()`. The solver can then be run to solve the symbolically stated optimization problem. Variables that are passed to the solver are the upper and lower bounds of the optimization variables `u_max` and `u_min`, upper and lower bounds of the constraints `h_max` and `h_min` and the starting values of the optimization variables `u_0` that initialize the iterative solving process. For the optimization variables that are associated with the fertilizers, the starting values are chosen to be equal to zero. For the water sources the starting values are set as defined by Eq. (6)

$$u_{0,i} = \frac{V_{\mathrm{MW,post,max}}}{i_{\mathrm{max}}}. \tag{6}$$

2.4 Saving the parameters in an output xml-file with pandas

Saving the solution the solver provides in a xml-file is done with functions of the pandas library. The optimization variables and the identifiers are combined in a Python dictionary, which is then transformed to a Pandas-dataframe. By using the function `to_xml()` a xml-file `Output_Values.xml` is created, which contains all data that was previously stored in the frame.

3 Additional notes

3.1 Error messages

There are two parts in the program that lead to an exception which triggers an error warning and an immediate exit out of the program. The first exception happens when the starting concentration is equal to the reference concentration and the stated maximum volume is less or equal to the current volume. In this case the desired nutrient solution is already present in the mixing tank and a new solution does not have to be mixed. The program then stops the execution at this case, because it also leads to a division by zero when initializing the balancing weights and therefore to an error message of the solver.

The second exception occurs when the optimization problem is infeasible. This can for example happen if the sum of the upper bounds of water sources added to the current volume of the mixing tank is less than the minimum water volume that should be in the tank after mixing the solution. Another example is that by constraining the use of certain water sources the minimum realizable nutrient concentration is larger than the stated `Max_Value` in `Nutrient_Concentrations.xml`. If the corresponding error message shows up, the user is advised to check the inputs in the xml-files for plausibility.

3.2 Deactivating plots

At the end of the program the simulation results are visually presented as bar charts. These plots were also used in Sec. 3.1. However, keeping the plot open also interrupts the program execution and a new run of the program can only be done after the plot is closed. If the visualization of the results is not of interest, the respective line `plot.show()` at the end of the Python script can be commented or deleted.

3.3 Copyright and License

This disclaimer of warranty is an essential part of the Licence and a condition for the grant of any rights to the Work.
