Kgs. Lyngby 30/6 – 2020

# NOAH Tool User Manual and Documentation

## Output 3.3 of Interreg Baltic Sea Region project NOAH

*Protecting Baltic Sea from untreated wastewater spillages during flood events in urban areas*

**Authors:** Jonas Wied Pedersen, Magnus Bjerrum Johansen, Morten Borup

*Technical University of Denmark, Department of Environmental Engineering (DTU Environment), Urban Water Systems section, 2800 Kgs. Lyngby Denmark.*

# Contents

# Introduction

The NOAH Tool is an assisting tool that enables the user to explore the potential in implementing Real Time Control (RTC) solutions in urban drainage systems based on an existing SWMM model of the system. The tool can furthermore calibrate a SWMM model based on observations from the system. The basic functionality of the tool can be used via its Graphical User Interface (GUI) while more options are available in the tools code repository for the user that is accustomed with coding in Python.

## Estimating Real Time Control settings

To estimate RTC settings, the user has to define what should be controlled (such as a weir, gate or orifice), what sensor the control is based on, and what the control should try to minimize. The tool then automatically finds close to optimal control settings based on a limited number of simulations. The tool is meant to provide the user with a solid idea about how much can be gained with a specific control setup for a default version of the given type of actuator. The final minor adjustment or tuning of RTC parameters so that they fit exactly to the specific actuator will have to be a manual process.

## Calibration and data validation

Since the RTC parameters are found using a SWMM model, this model needs to be accurate in representing the system dynamics. For this reason, the tool includes a model calibration feature that automates the process of calibrating the SWMM model based on downstream flow or water level measurements from the drainage system. This is done running a limited number of simulations in order to ensure that calibration can be performed on large, computationally expensive real life models.

If the calibration process cannot make the model and observations fit, this can be due to errors in the overall model structure that need to be fixed, but it can also be due to errors in the observational data. Furthermore, if a calibrated model suddenly starts to perform worse in terms of matching the observations this can be due to either a change in the system not included in the model or a suddenly occurring error related to the sensor data. Therefore calibration and data validation goes hand in hand, and the NOAH tool can both calibrate a model as well as quantify the model to data fit for a user defined period which enables model based data validation.

## Open source, collaborative development

The NOAH Tool 1.0 is the first version of the tool. In time its functionality will increase as the university partners of the NOAH project as well as others not connected to the project, add functionality through the collaborative, open source nature of the code that can be accessed and improved by anyone using the repository available at https://github.com/mbjjo/NOAH

# Estimation of RTC settings

The RTC feature in the NOAH Tool helps the user with finding good RTC settings. This is a task that most often is performed by manual running of numerous model simulations, which besides being waste of the engineer's time, also does not ensure that a result close to optimum is obtained. The NOAH Tool does this automatically meanwhile quantifying the improvement of the system performance due to the RTC. The real time control is computed as simple rule-based control. If the water level in a node exceeds a certain level an actuator is activated.

## Prerequisites

The only prerequisites for using the tool is a calibrated SWMM model that represent the main dynamics of the urban drainage system and a rainfall time series that represent the types of events for which the RTC should improve the performance of the network

## Methodology

The estimation procedure is illustrated in Figure 1. It is based on repeated model runs with different RTC settings. The performance of each model run is quantified in an objective function based on the scope of the RTC (e.g. reducing overflow). First the method samples the relevant RTC parameter uniformly within the user defined parameter space. The best performing of these parameter sets is then fine-tuned using a simplex algorithm.

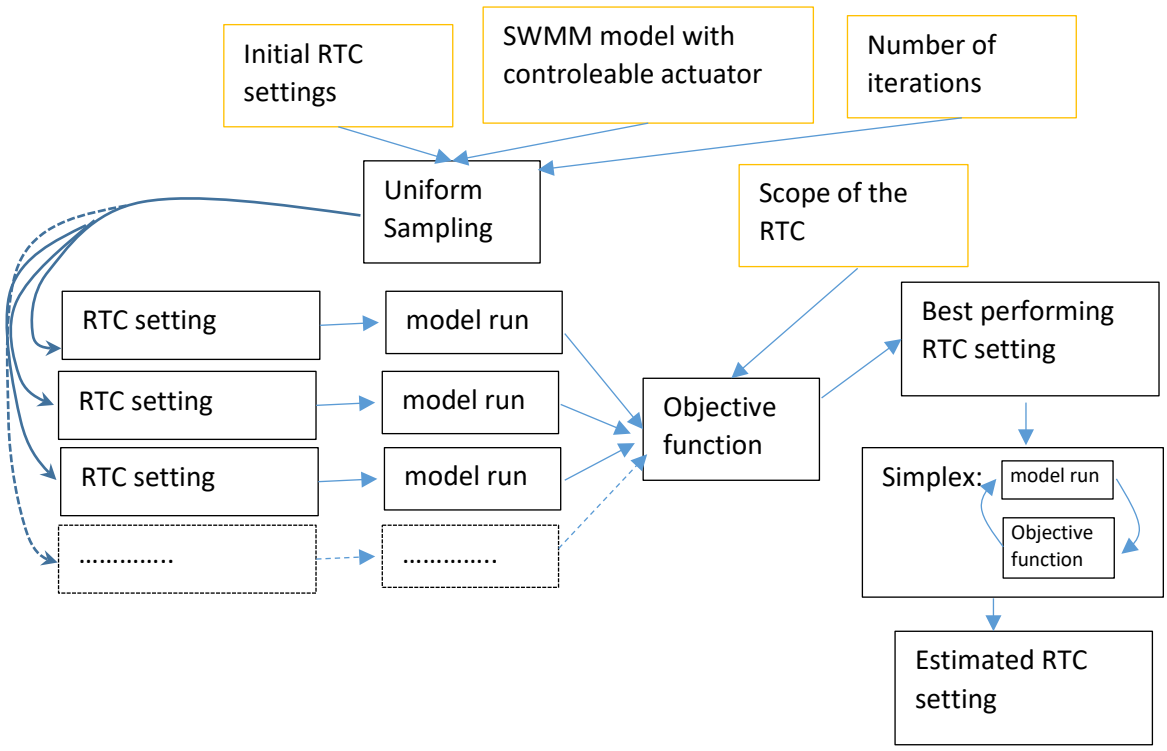



*Figure 1: Overview of the process of estimating RTC settings.*

## Using the GUI

The RTC estimation can be initiated through the GUI of the NOAH Tool. Figure 2 shows the RTC tab in the tool.

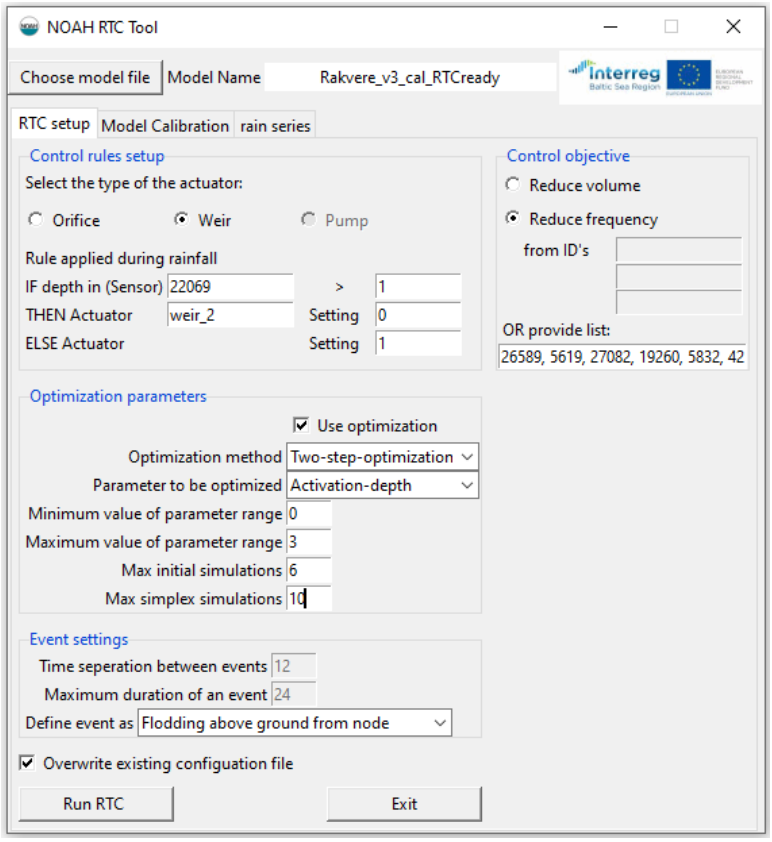
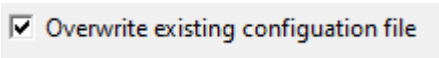



*Figure 2: The NOAH Tool GUI with the RTC pane selected.*

The *Overwrite existing config file* check box in the button on the window allows the user to run simulations without typing in all the required fields in the GUI. If this is checked the data in the GUI is saved to a new configuration file that is used for the computation. If not, the existing configuration file that matches the model name is used. This feature allows the user to easily run similar computations with various changes in the model such as basins with different dimensions.




A folder with the timestamp of the beginning of the computation is created in *NOAH_RTC_Tool\output*. (e.g. NOAH_RTC_Tool\output\2020-06-11_10-19-18). This folder contains the results and plots from the computation.

The model generated both after the calibration and RTC optimization process is saved in the same folder as the original model.

Rain data and other external files must be defined as a Timeseries in SWMM. If this is not the case (e.g. if rain is defined directly in the Rain gages) a copy of the timeseries must be placed in the *\lib* folder.

For more information about the different fields and elements in the GUI, hover the mouse over the widget and a text with small explanations will appear.

## Actuators and sensors

Control always needs at least one sensor and one actuator.

All nodes in a SWMM model can be used as virtual water level sensors since this only require the depth in the node while all links can be used as virtual flow sensors.

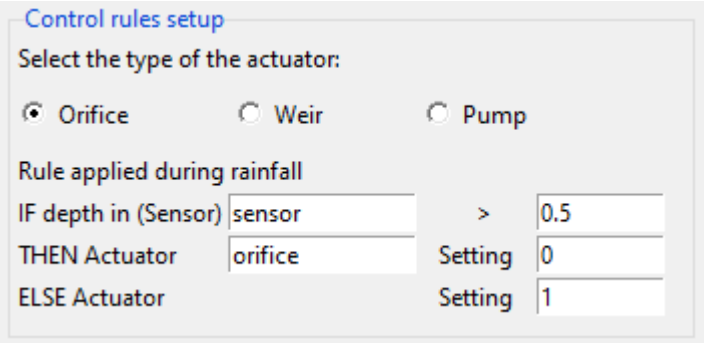
The required actuators are expected to be implemented in the model before the RTC tool is used. The following is an explanation on how different actuators are applied.

### Orifice

An orifice is a structure that can be either open or closed and thus allowing water to pass towards the downstream part of the system.

The setting that is defined in the interface is the fraction open (i.e. 0 means closed and 1 means open) and should be in the range between 0 and 1.

Example:




If the depth in the sensor exceeds 0.5 meters the orifice will close. If the depth is below 0.5 the orifice will be open.

Other settings between 0 and 1 can be applied in order to let smaller amounts of water through the orifice.

Note that the time that it takes for the orifice to change position should be defined beforehand in the SWMM model.

### Weir

Weirs in SWMM can act as moveable weirs where the height of the crest can be controlled dynamically. The *setting* that is defined corresponds to the fraction that the crest of the weir is lowered compared to its highest position.

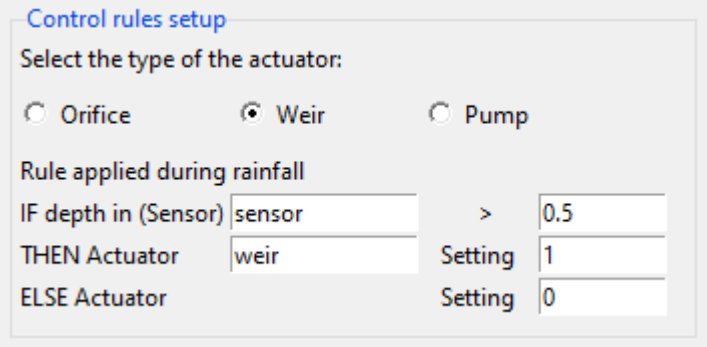
The height of the weir is then:

$$weir\ height = \ Inlet\ offset + (1 - X) * height$$

where X is the setting that is inputted in the GUI.

Thus, a setting of 1 means that the weir is fully open (the lowest point) while a setting of 0 means that the weir is closed (lifted to its maximum height).

Example:




If the depth in the node that is named *sensor* exceeds 0.5 meters, the weir will be fully open and else it will be at maximum elevation.

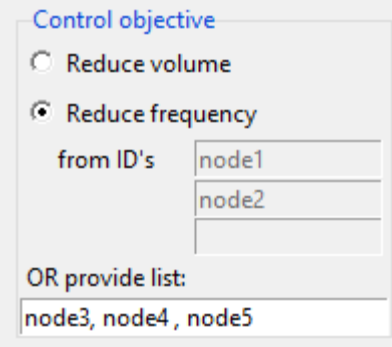
In the example the sensor is located upstream of the weir. This means that a high water level will lower the weir and let water flow further down in the system.

If the sensor is located downstream of the weir the setting should typically be swapped, such that a high water level causes the weir to close and prevent water from flowing down in the system.

Other settings between 0 and 1 can be applied. This could be to lift the weir in case of rising water level but still allow water to flow over the weir when the water level reaches a certain height.

## Defining the objective

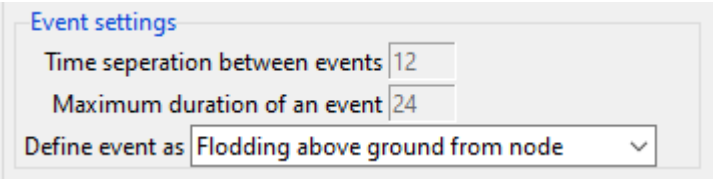
The scope of the optimization is to minimize either the total volume or the frequency (number of events) where flooding or CSO occur. This is computed from a selected number of nodes defined by their node ID.
The node ID's can be defined in three fields or provided as a list of nodes separated by "," if more than three nodes are to be computed. If nodes are typed in the list then the upper three fields will be ignored.




In this case the number of flooding events are to be reduced from node3, node 4 and node5. Node1 and node2 are not included in the computation.

Depending on the selected optimization target (volume or frequency) the optimized results might differ. It is therefore recommended to run the optimization for both targets and compare the results. This will also reveal any conflicts between the two targets that might appear in some cases.

The Event settings frame is used to define how an event is computed. These are predefined




If CSO's or flooding occur within the "separation time" they are counted as one event.
If one event lasts longer than "maximum duration" they are counted as several events.
The event definition depends on the model. If the CSO's are computed as outflow from an outlet, the "Outflow from CSO structure" should be selected whereas "Flooding above ground" should be selected if the flooding is computed as flooding from nodes.

## Optimization of Real Time Control

The optimization of the real time control is defined in the window with optimization parameters.

The variable that is optimized in Figure 3 is the "activation depth". This is the depth that should be measured in the sensor node before the actuator (orifice, weir or pump) reacts and changes its position.

Optimization is only applied if the "Use Optimization" field is checked. If this is not the case the simulation will just run with the setup that is defined in the Control rule setup.

The optimization is done by a “two-step-optimization” which first makes a certain number of simulations within the entire parameter space and afterwards runs a simplex algorithm with starting point as the minimum value from the initial step.

Example:

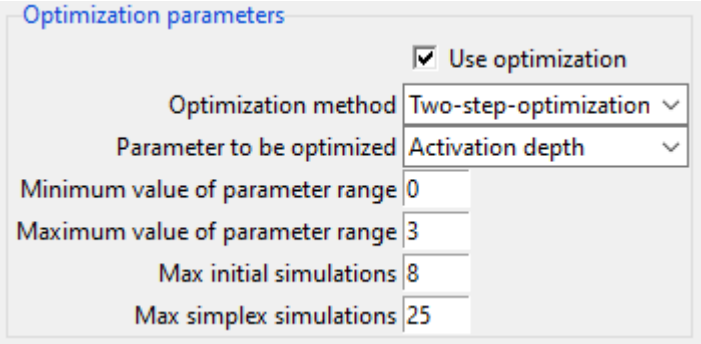



*Figure 3: Settings for the optimization of RTC parameters.*

The parameter range in Figure 3 is defined to be between 0 and 3 meters. No setting outside this range can be found. This should typically be the lowest and highest depth of the sensor node.
8 Simulations are run within this range and afterwards a simplex algorithm is initialized from the lowest point and runs either 25 iterations or until the algorithm has converged and reached the minimum value.

A SWMM model with the optimal RTC setting is saved in the same folder as the original model. Furthermore a text file as well as a plot of the initial step of the optimization is saved in the output folder.

## Example case: Estimating RTC settings for Rakvere pilot case

The NOAH Tool was applied at the pilot case of Rakvere. The GUI was used as described below, see Figure 4.

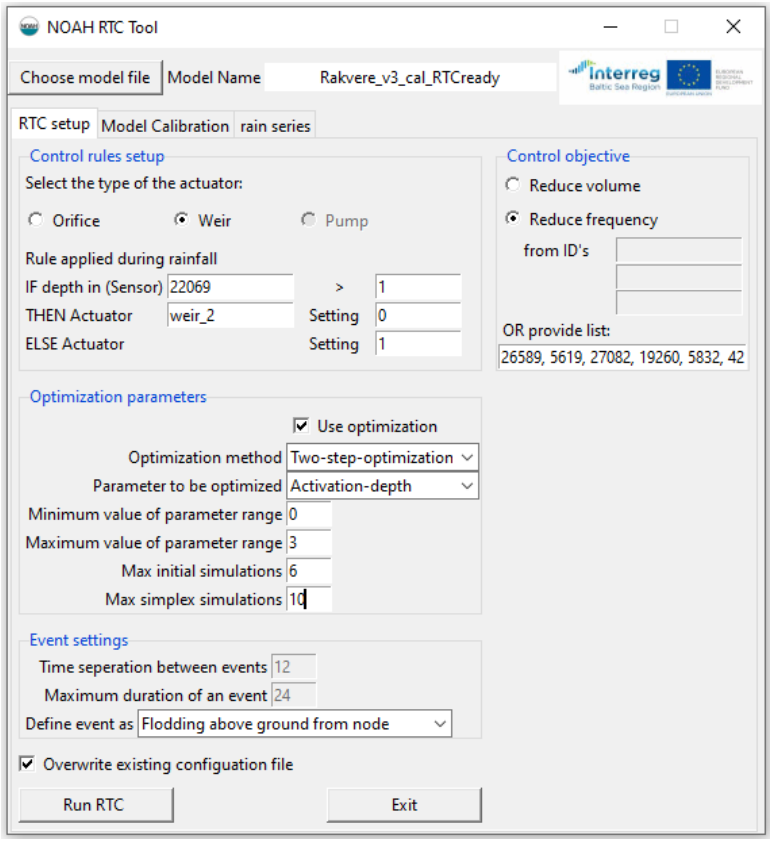



*Figure 4: Setting up the estimation of the RTC settings for the weir controlling the flow from the upstream lake in Rakvere.*

The model was prepared by adding a weir in SWMM. This is the one that is to be activated based on values from the sensor which is located at the node with ID 22069.

When the water level in node 22069 exceeds a certain depth the weir is set to a setting of 0 (i.e. maximum elevation). The weir lowers again when the water level drops below this level. A wide static weir is installed in parallel with the moveable weir to ensure the water level in the lake does not become too high.

The flooding is monitored from 8 different nodes with ID's: 22065, 14, 26589, 5619, 27082, 19260, 5832 and 42. The objective of the optimization is to reduce the number of flooding events from these nodes. Note that one rain event can be counted as several events if it causes flooding in several nodes.

The parameter range is set to be between 0 and 3 as the water level at the sensor cannot exceed 3 meters. Initially 6 simulations are run within this range, see results in Figure 5.

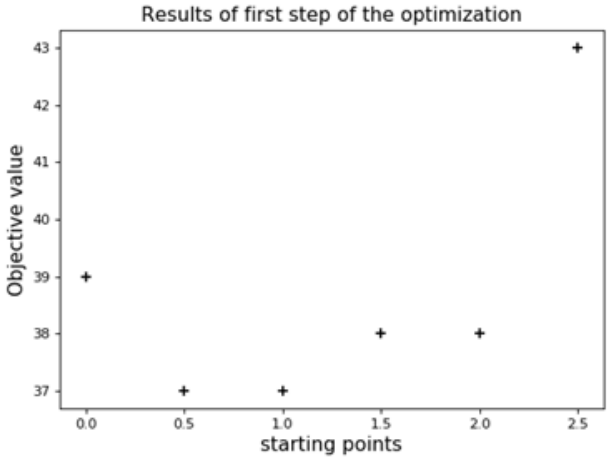



*Figure 5: Objective function values for the first step of the optimization. This indicate that the optimum is somewhere between 0 m and 1.5 m. Hereafter the simplex routine finds the optimum.*

It is seen that the lowest number of flooding events is 37. This is reached at an activation depth of 0.5 meters. The simplex now uses this setting as a starting point and evaluates nearby steps and thus finding the minimum value.
In this case an activation depth of 0.5125 will result in 36 flooding events which is the minimum.

This result means that if the weir is raised up when the sensor depth exceeds 0.51 meters, the number of flooding events can be reduced.

# Model Calibration

The NOAH tool allows you to calibrate a SWMM model against in-sewer observations of flow and water levels. This chapter describes the calibration features available through the NOAH Tool GUI as well as listing the expanded functionality available through the tools in the Python code repository.

## Prerequisites

The tool requires three inputs to be prepared by the user before a calibration can be performed:

1. A functioning SWMM model
2. In-sewer observation time series of either water level or flow data
3. An observed rainfall time series for the catchment area for the same period as the in-sewer observations

The output of running the calibration procedure is a new SWMM .inp file, where the selected parameters have been calibrated against the user-provided data.

## Calibration Methodology

The automatic calibration is performed by running multiple SWMM model simulations with different parameters and comparing the results from these simulations with sensor data. The comparison is done with an objective function, which quantifies the difference between model and observations. The optimization is divided into two steps: First the overall parameter space is explored in a cost effective way using Latin Hypercube Sampling[1]. Hereafter, in an iterative process the next parameter values to be tested is found by analysing the objective function values of the previous simulations and in the end the parameter set resulting in the best objective function value is chosen as the calibrated model. In version 1.0, the latter is done with the commonly used Nelder-Mead simplex routine[2], but this will probably be supplemented in later version of the tool with a newly developed method that is tailor-made for calibration of urban drainage models. The procedure is illustrated in Figure 6.

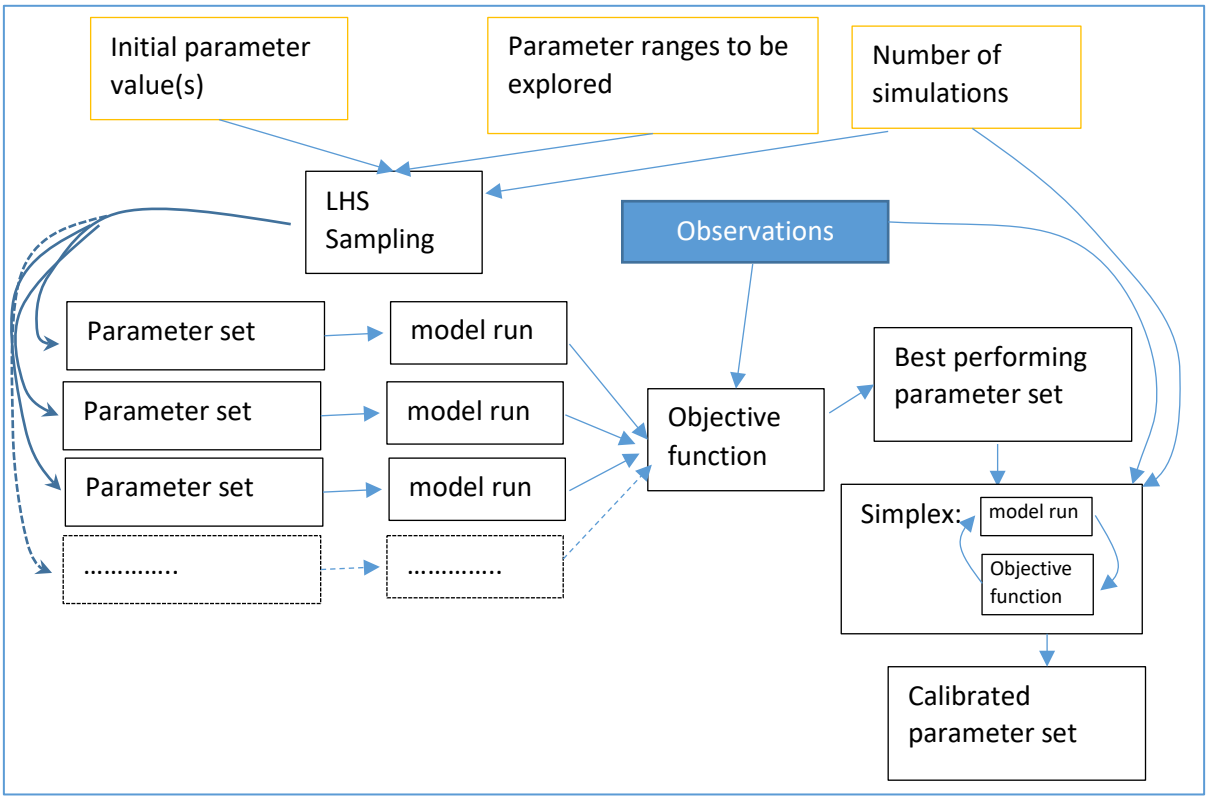



*Figure 6: Conceptual overview of the calibration process. The yellow boxes indicate user inputs.*

[1] Iman, R. L., Helton, J. C., & Campbell, J. E. (1981). An approach to sensitivity analysis of computer models: Part I—Introduction, input variable selection and preliminary variable assessment. Journal of quality technology, 13(3), 174-183.
Python implementation: https://pythonhosted.org/pyDOE/randomized.html#latin-hypercube [2020-06-30]

[2] Gao, F., & Han, L. (2012). Implementing the Nelder-Mead simplex algorithm with adaptive parameters. Computational Optimization and Applications, 51(1), 259-277.

# Using the GUI

## Preparing the model

Before the calibration tool is used it is important that the user-specified SWMM model is fully functioning, in the sense that the user should be able to run simulations from the standard SWMM GUI. It is important to acknowledge that calibration is the LAST step after the modeller has ensured that the asset data from physical properties of the system has been determined, such as pipe diameters, invert levels, approximate areas of the various sub-catchments etc. The parameter values in the user-specified model do not need to be well-defined, but they should be relatively reasonable from an engineering point of view.

Any additional .dat or .ini files (e.g. rain data, boundary conditions, or hotstart files) that are needed to run the model must be located in the same folder as the .inp file.

The first step in the calibration procedure is to select the model that needs calibration, see Figure 7.

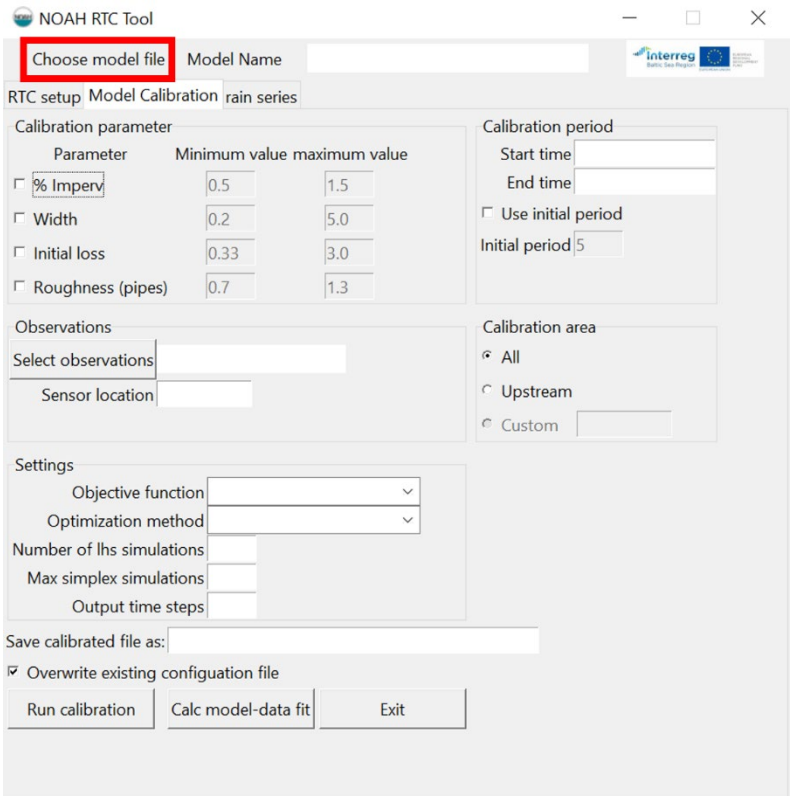


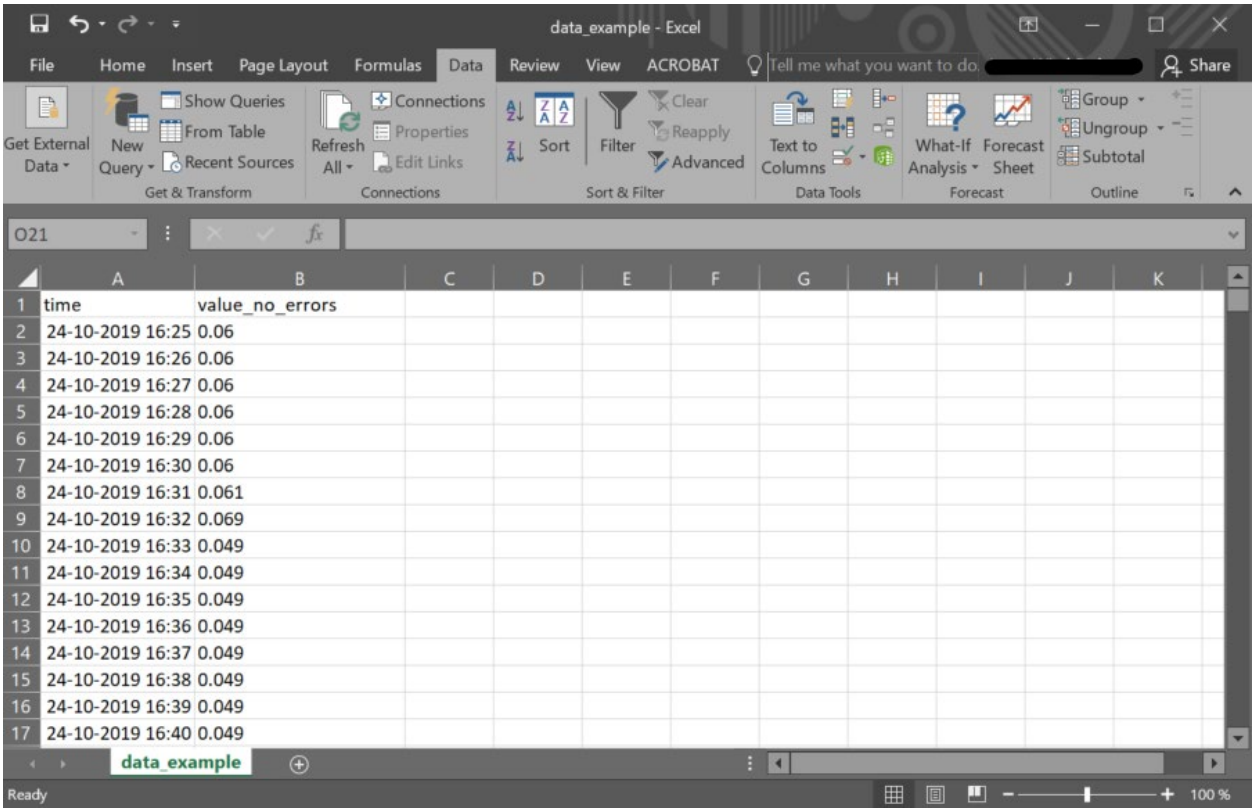

*Figure 7: Selection of the SWMM model that is to be calibrated is done using the same dialog as when selecting model for RTC (highlighted with a red square).*

## Preparing the data

The current version of the tool is able to calibrate a model against observations from a single point in the sewer system. This can either be measurements of water levels or flows. The data needs to be provided as a comma separated file (.csv) with two columns.

- The first column should have the headline “time” and contain time stamps in the format “YYYY-mm-dd HH:MM:SS” (e.g. 2018-08-01 00:00:00).
- The second column should have the headline “value_no_errors” and contain numeric values.

The units of the data should be the same as the user-specified units in the SWMM model, e.g. meters for water depths and $m^3/s$ for flows.



## Parameter selection

The current version of the tool is able to calibrate the four main SWMM parameters that relate to the impervious surfaces and the pipe network:

1. "% Imp": Imperviousness of the sub-catchments. Affects the magnitude of runoff and peaks during rain events.
2. "Width": Width of the overland flow path. This parameter determines how fast the water run of a sub-catchment.
3. "Dstore": Depth of the depression storage on the impervious areas (also known as the initial loss/abstraction in the sub-catchment). This determines how much rainfall that is needed before runoff happens from impervious areas.
4. "n_pipe": Manning's roughness coefficient for the conduits. Determines the capacity of the pipes and to some extent the velocity of the water in the pipes and thus affects the shape of the hydrograph from the system.

The user can choose one or more of the four parameters for calibration by ticking of the boxes next to each parameter, see Figure 8.

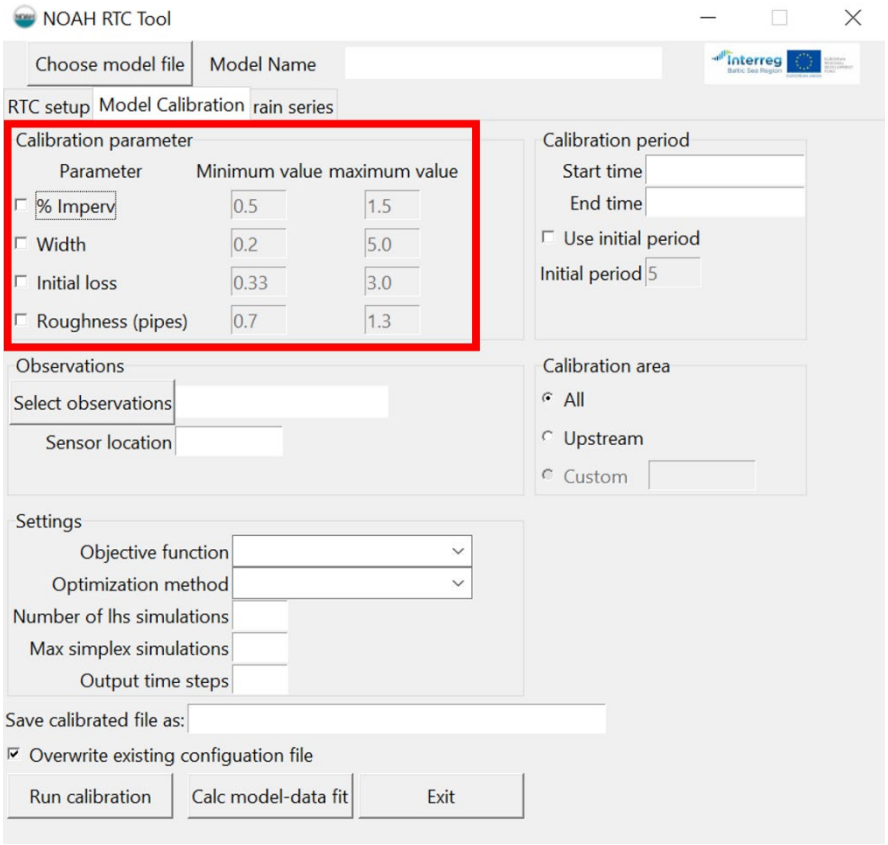



*Figure 8: Choosing what parameter(s) to calibrate.*

The calibration needs a user-specified minimum and maximum value for each parameter that determine the range of values that will be investigated during the calibration routine. The min and max values are multiplicative factors that are applied to the original parameter values in the uncalibrated SWMM model. As an example, a min value of 0.5 and a max value of 2 means that the calibration procedure will search for the best possible parameter value within a range of half to double the original parameter value.

## Calibration period

The period of time that the calibration should take place over is also a user-specified value. Here, the user chooses a Start time and an End time for the period in the format "yyyy-mm-dd HH:MM", see Figure 9.

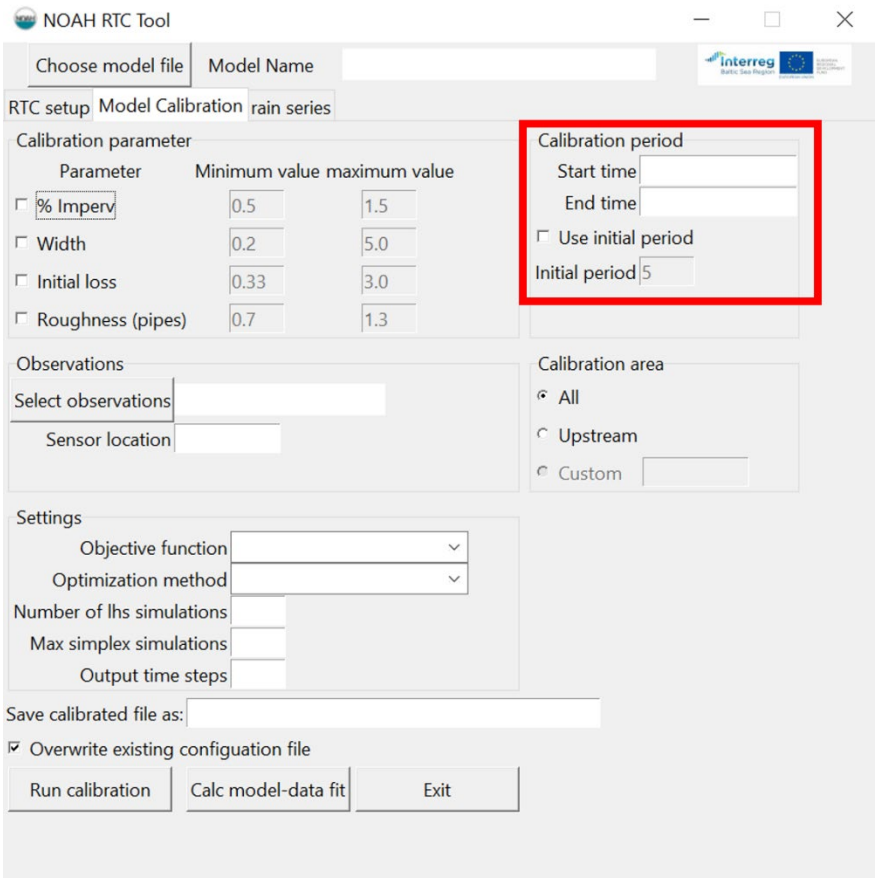



*Figure 9: Setting the calibration period.*

There is also an option of using an initial period for “warming up” the model before the calibration starts to quantify model performance with the objective function. This feature starts the simulation a user-specified number of hours before the specified Start time, which provides the model with better initial conditions. The user should choose an initial period that is large enough to “wash out” all effects of the initial conditions in the system. Five hours is given as a default value for the initial period. It is important that the input data (e.g. rainfall data and other boundary conditions) is available also for the initial period.

## Calibration area

It is often the case that the location of the sensor device that measures the calibration data is not located at the outlet of the SWMM model. It might therefore not be appropriate to calibrate the parameters of all parts of the model with data from a single point in the system (especially when large parts of the model is downstream of the sensor location). The NOAH tool gives the users a couple of options for choosing which parts of the model that should be calibrated, see Figure 10.

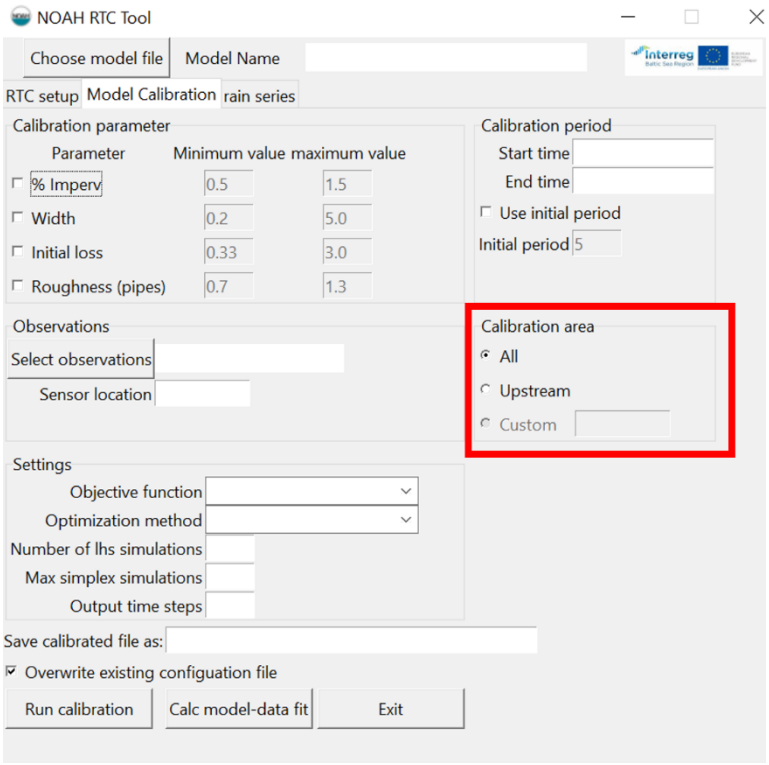



*Figure 10: Selecting for what parts of the system the parameters should be changed during the calibration.*

- All: All elements in the SWMM model (i.e. all sub-catchments or conduits) are calibrated against the data.

- Upstream: Only the model elements that are upstream of the sensor location are part of the calibration. Downstream model elements will maintain their original parameter values. This functionality only works if conduits are drawn from upstream to downstream nodes in the original SWMM model.
- Custom: This function is not fully implemented yet. The plan is to implement an option so the user can specify exactly which sub-catchments and conduits that need calibration.

## Optimization settings

A calibrated model is not just a calibrated model. The objective function, that is the function that is used by the calibration routine to decide which parameter set gives the best model performance when comparing to the observations, does have an influence on the final calibration result. Therefore the user has to choose between different objective functions, see Figure 11.

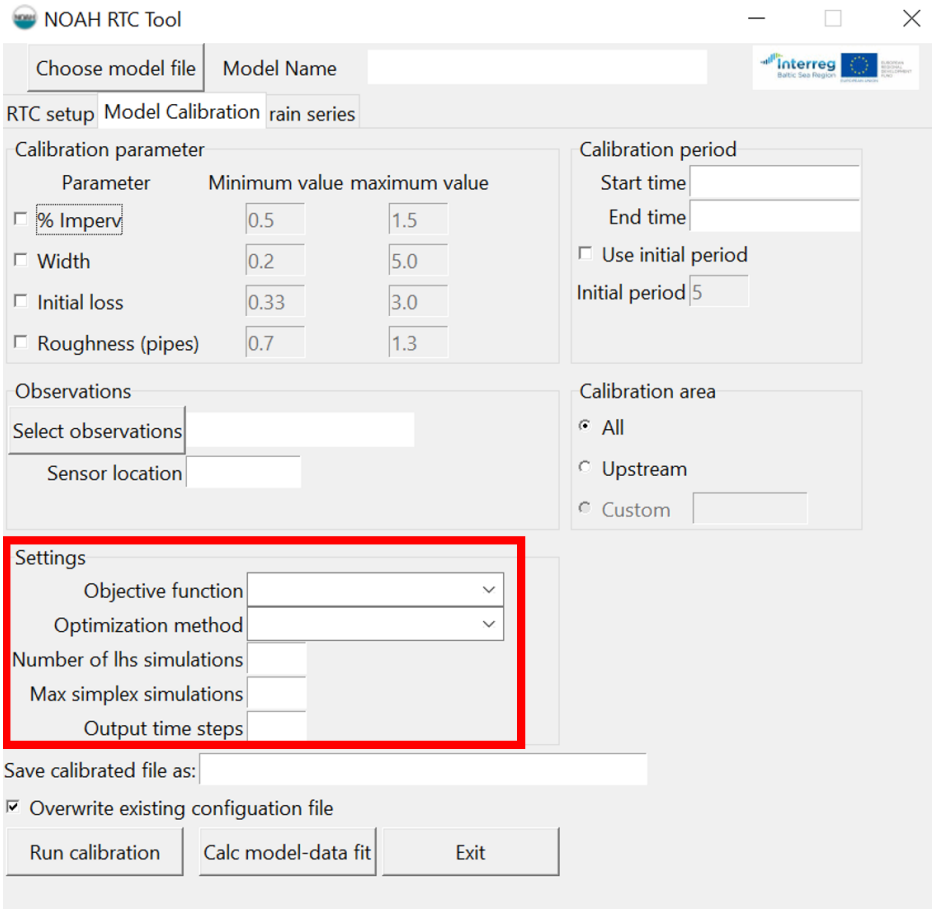



*Figure 11: Choosing the objective function and setting the optimization settings.*

Five different objective functions can be used during the calibration:

- Root Mean Squared Error (RMSE)
- Mean Absolute Error (MAE)
- Nash-Sutcliffe Efficiency (NSE, during optimization the routine is minimizing the negative NSE)
- Absolute Relative Peak Error (ARPE)
- User-defined objective function

The first four objective functions are predefined as:

$$RMSE = \sqrt{\sum_{i=1}^{n}(M_i - O_i)^2}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|M_i - O_i|$$

$$NSE = 1 - \frac{\sum_{i=1}^{n}(M_i - O_i)^2}{\sum_{i=1}^{n}(O_i - \bar{O})^2}$$

$$ARPE = \left|\frac{\max(M) - \max(O)}{\max(O)}\right|$$

Where $M$ is the simulated model output, $O$ is the observed time series, $n$ is the number of time steps in the observed time series within the simulation period (from Start time to End time), and $\bar{O}$ is the mean of the observed time series.

The last objective function “User-defined objective function” gives the user the possibility of defining his only objective function by coding a few lines of Python code. This is done by locating the Python code file User_defined_objective_function.py and editing it with the appropriate function. The default content of this file is a mixed objective function that calculates its output value as an average between the max peak value and the correlation of the time series. Note that the user defined objective function needs to be increasing for worse match between model and observations.

The user can choose between three different optimization methods for the calibration:

1. **LHS** (Latin Hypercube Sampling): This method performs an efficient search of the parameter space with Latin Hypercube Sampling. The method simulates a number of model runs with different parameter sets that obey the rules of LHS and calculates the objective function value for each simulation. The parameter set that returns the lowest objective function value is the final, calibrated parameter set. The user specifies how many SWMM simulations the optimization method is allowed to perform before it has to deliver a calibrated parameter set.
2. **Simplex**: This method uses a standard Nelder-Mead simplex routine as provided in the "optimize" function of the Python package "Scipy". The optimization method starts with the initial parameter set in the SWMM model and progressively searches for new parameter sets that improve the objective function value. The search strategy is a standard simplex-style search. The user defines the number of SWMM simulations the optimization method is allowed to run before it has to return the parameter set that so far has provided the best objective function value. In case the algorithm converges on a parameter set with fewer simulations than the user has specified, the routine will simply end and return the best parameter set. It should be noted that simplex-style optimizations are sensitive to local minima, which can be an issue for objective functions with complex response surfaces.
3. **Combined**: This method runs a user-defined number of LHS simulations before it starts a simplex routine. The advantage of this is that the initial LHS simulations search the parameter space globally before the simplex routine is started. This will reduce the risk of the simplex routine ending in a local minima although it does not guarantee that this cannot happen. This method is the preferred and default method among the three options.

Finally, the user should specify the "Output time step" that is needed in the simulation output. An appropriate choice is to use an output time step that equals the time steps in the observations. If the observations and simulation do not have the exact same time stamps, the NOAH tool will linearly interpolate the simulated values to fit the observed time stamps.

## Output from running the calibration

To run the calibration simply press the *Run calibration* button, Figure 12. The output of running the calibration tool is a new .inp file where the selected parameters have been optimized against the provided data. This file will be located in the same folder as the original file and have the name that is specified in the GUI. Plots that show the objective values during the various simulations as well as convergence of the simplex routine are saved in a folder with the timestamp of the simulation in *NOAH_RTC_Tool**output*. (e.g. NOAH_RTC_Tool\output\2020-06-11_10-19-18).

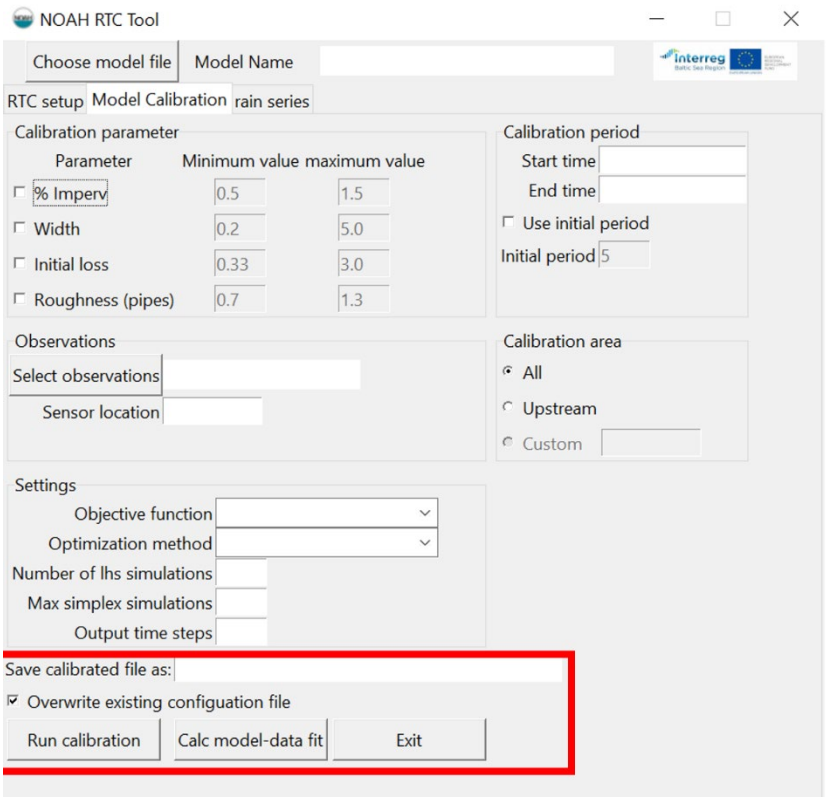



*Figure 12: The part of the Model Calibration tab that allows for starting a calibration and saving the calibrated SWMM file, as well as simply doing a single run and calculating the objective function using the "Calc model-data fit" button.*

## Calculate model-data fit

It is possible to evaluate the fit between a SWMM simulation and an observed time series. If the model has previously been calibrated towards a sensor, this function can be used to do integrated model-data validation. By running this function from time to time with the newest data included, the user can see if the objective function is getting worse, in which case there is likely to be errors in either the model or the data. The inputs to the functionality is similar to that of the calibration routine, see Figure 12.

The user must specify:

1. "Choose model file".
2. "Start time", "End time", and warm-up period settings.
3. Objective function

## Functionality in the Python code library relevant for model calibration

The Python library includes more functionality than what can be accessed through the GUI. The following shows a list of python functions and what they do.

| Name | Description | Input | Output |
|---|---|---|---|
| Swmm_model_inventory() | Load properties of nodes, links, subcatchments and rain gauges from the SWMM model into Python. | • A SWMM .inp file. | Four pandas data frames, one for each type of data:<br>• Nodes<br>• Links<br>• Subcatchments<br>• Rain gauges |
| Backwards_network_trace() | Identify model elements that are upstream of a specific node. Requires that the links are properly drawn (in the flow direction) in the original SWMM model. | Loaded model element properties from swmm_model_inventory function). | Three lists of model element ID's for each type of system element:<br>• Nodes<br>• Links<br>• Subcatchments |
| Swmm_simulate() | Simulate a SWMM model within a given period and collect model output for selected elements. The variables that can be collected for each element is:<br>• Nodes: Water depths<br>• Links: Flows<br>• Subcatchments: Runoff | • .inp file<br>• Start and end time stamps for simulation period<br>• Lists of selected elements (nodes, links, subcatchments)<br>• Desured output time step given in seconds<br>• Whether a hotstart/warm-up period should be used<br>• The length of this warm-up period in hours | Dictionaries with outputs for all selected model elements. If multiple kinds of elements are chosen, there is one dictionary for each type (nodes, links, subcatchments. |

| | | | |
|---|---|---|---|
| Change_model_property() | Change almost any kind of property in a SWMM .inp file (e.g. a parameter value, pipe diameter, weir type, dry weather flow, etc.). The user can choose whether to provide a new value to a model element, or to modify the current value with a multiplicative factor or an additive factor. A resulting new model can then be saved as a new .inp file if desired. | • .inp file<br>• The "section" of the model element that needs to be changed (see outcommented code above the function for options).<br>• A list of model element ID's that needs a property changed.<br>• Name of parameter/property to be changed.<br>• Optional: a new replacement value for the property.<br>• Optional: a multiplicative factor.<br>• Optional: an additive factor.<br>• Optional: Name of path for a new .inp that can be created. | A new SWMM model where the desired property has been changed. |
| Simulate_objective() | Run a SWMM simulation, compare with observed data, and return an objective function value. | • .inp file.<br>• Simulation start and end time stamps.<br>• Selected model elements (the model location where the data has been observed).<br>• Desired output time step.<br>• Whether a hotstart/warm-up period should be used.<br>• The length of this warm-up period in hours.<br>• Pandas data frame with observated data.<br>• A function that contains the desired objective function. | A calculated objection function value. |

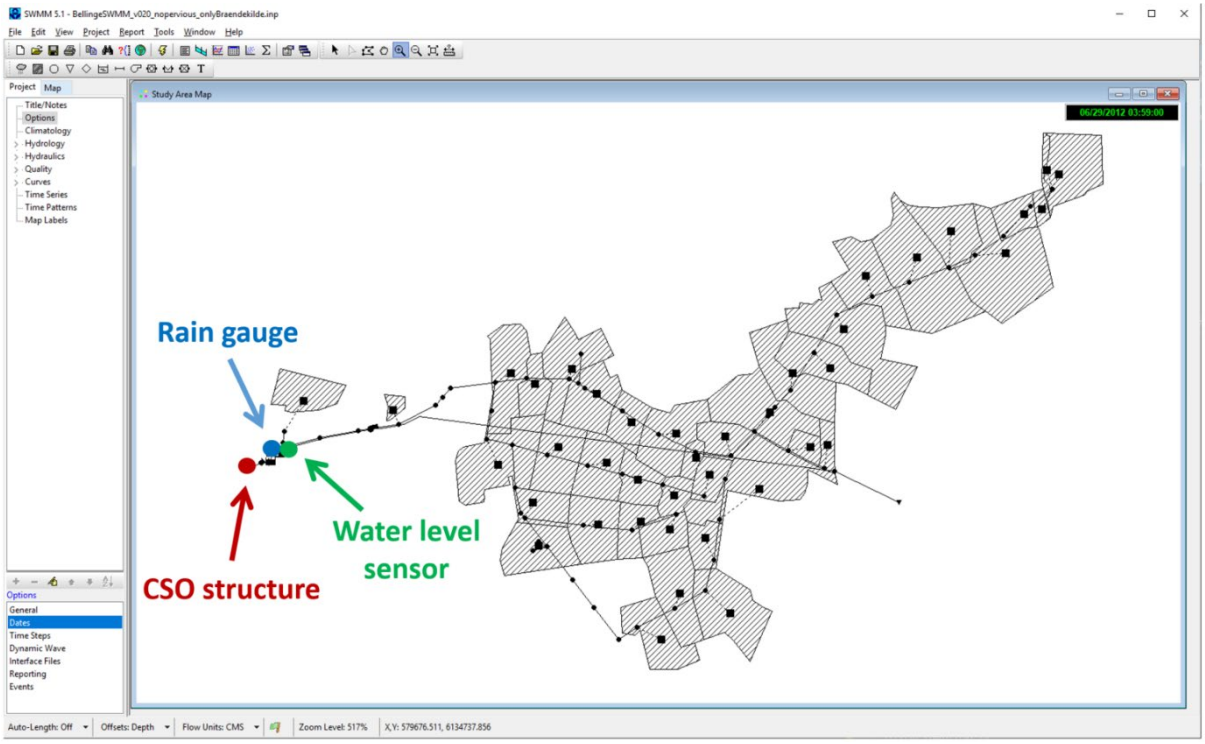
| | | | |
|---|---|---|---|
| Create_new_model() | Change multiple parameter values in a model by multiplying a factor to the original parameter values. | • List of multiplicative factors for each parameter that is being changed.<br>• .inp file<br>• Name of path for a new .inp that is created.<br>• "Sections" of the parameters that are being changed.<br>• "Names" of the parameters that are being changed.<br>• Number of parameters that are being changed.<br>• List of model elements that needs new parameters vales (nodes, links, subcatchments). | A new .inp file. |
| Create_runobjective_delete_model() | Specify settings for a model with changed parameter values, and simulate an objective function with this model. | • All inputs required for the create_new_model() function<br>• All inputs for the simulate_objective() function. | An objective function value. |
| Generate_run_lhs() | Generate parameter sets that follow LHS principles and run SWMM simulations with those sets. The user can specify whether results from all sets or just the best set should be outputted. | • Number of LHS parameter sets that are sampled.<br>• Ranges for all parameters for constraining the sampling space.<br>• All inputs required for the create_runobjective_delete_model() function. | Objective function values for all tested parameter sets, or objective function value for just the best parameter set. |
| Run_simplex_objective() | Run a simplex routine based on a user-specified maximum | • Initial parameter values.<br>• Path to a temporary folder where models can be created and | Objective function values for all tested parameter sets, |

| | | | |
|---|---|---|---|
| | number of simulation iterations. The user can specify whether results from all sets or just the best set should be outputted. | simulated (this folder is deleted after the function is finished).<br>• All inputs to the create_runobjective_delete_model() function. | or objective function value for just the best parameter set. |

# Example case: Calibrating SWMM model for Brændekilde, Denmark

## Bellinge case: system, actuators and sensors

An example of calibration setup is given here for the small town of Brændekilde, Denmark. The town has a combined sewer system, which collects wastewater from local residents and surface stormwater. At the outlet of the sewer catchment, the water is pumped further downstream towards the local wastewater treatment plant. There is a combined sewer overflow (CSO) structure next to the pumping station where wastewater is discharged into the natural environment during heavy rainfall events. There is a flow control structure immediately upstream of the pump sump that governs whether water flows to the pumping station or to the CSO structure. At this point, there is a sensor that measures water levels, and which will be used to calibrate the model against.




## Calibration setup and GUI settings

For this example, a calibration will be performed with one month of water level observations (March 1-31, 2020). Figure 13 shows a screenshot of the GUI settings for this calibration example.

All four parameters that can be calibrated with the tool are used here. The parameter ranges are left at the default values except for the “% Imperv”’s minimum value, which is lowered to 0.3 as the local utility company that operates the sewer system suspects that the imperviousness might be overestimated in the original model. All model elements will be calibrated, not just the upstream parts.

The objective function is chosen as RMSE, while the optimization method will use the “Combined” LHS and Simplex approach. The number of initial LHS simulations is set to 10, while the Simplex

routine is allowed to fine tune the parameter values with 130 simulations. The number of simplex simulations is here deliberately set high with the purpose of illustrating how performance continually improves during the fine tuning. However, in real cases the number of simplex simulation can be set much lower while still obtaining a decent fit (see below). Simulating this one month period with SWMM on a standard HP Elitebook laptop with an i5 processor takes approximately two minutes. A total of 140 simulations thus means that the full calibration will take between four to five hours to complete.

The water level data is recorded at one-minute time steps and the "Output time steps" is therefore set to 60 seconds for the simulations.

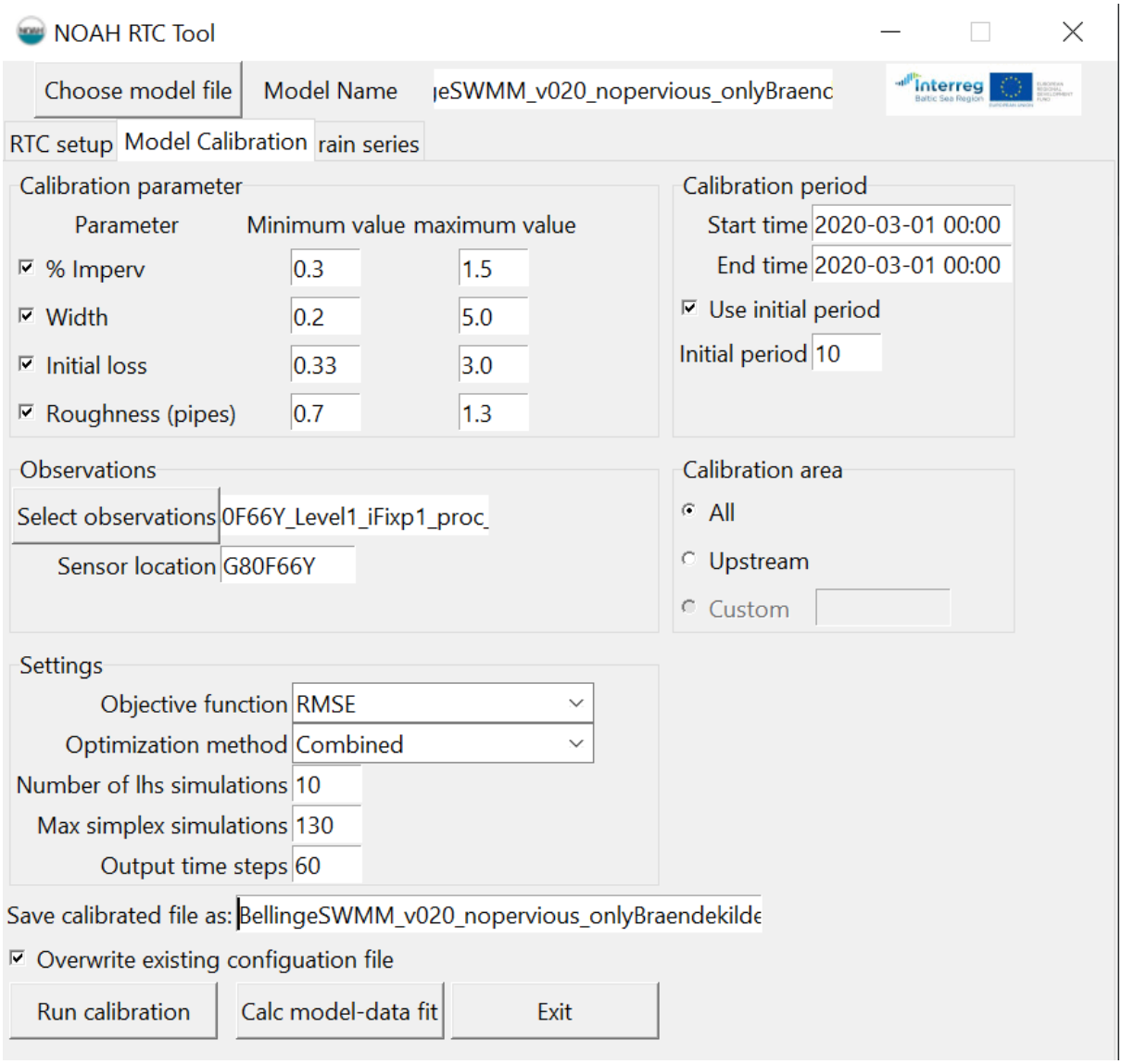



*Figure 13: A screenshot of the settings for the calibration example.*

## Results

After completion of the calibration routine, two figures are produced. The first figure (Figure 14) shows the objective function value for all tested parameter sets where the best sets of the initial LHS screening and the overall best parameter set after the full combined calibration run are highlighted. From the figure it is visible that the %Imperv parameter has a large effect on the value of the objective function, while the other parameters are less clearly defined. It is also seen that the difference between the best LHS parameter set and the best simplex set (130 simulations later) is rather small, which shows that a few initial simulations by themselves can substantially improve model performance. The result of the calibration confirmed the utility company's suspicion that the imperviousness was overestimated in the original model as the resulting %Imperv parameter is approximately half the original estimate. The width parameter is increased by a factor of four leading to a faster response to rainfall, while the initial loss ("S-Imperv" in the figure) and pipe roughness ("ManningN" in the figure) are close to their original estimates.

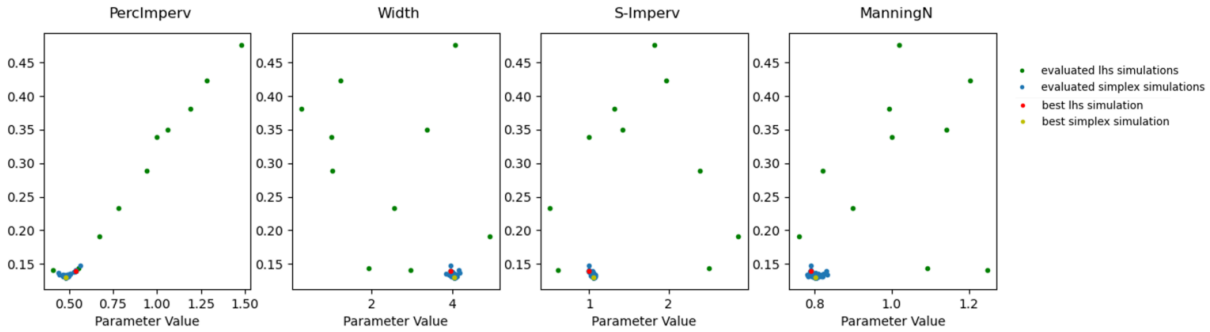



*Figure 14: All LHS and Simplex simulations for the test case and their resulting RMSE objective function value (y-axis).*

The second figure (Figure 15) shows the objective function value as the simplex algorithms converges towards the final calibrated parameter values. After 25 model simulations, the algorithm has already identified a parameter set that it very close to the performance of the final calibrated parameter set. Beyond 50 simulations there is no substantial improvement in the objective function. Normally, the simplex algorithm has found well-performing parameter sets within the first 25-30 simulations.

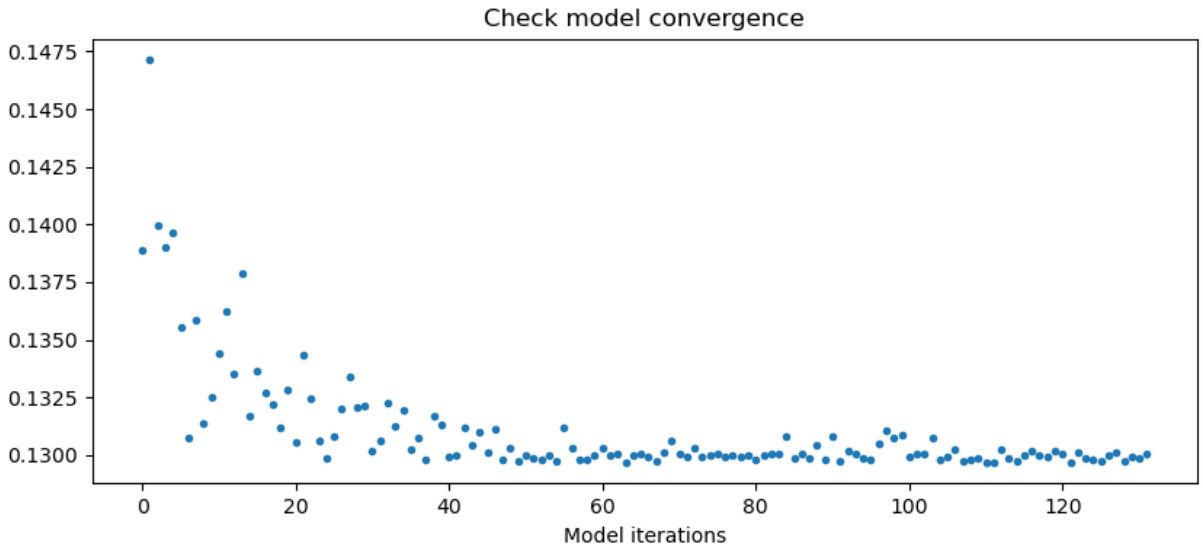



*Figure 15: RMSE objective function values (y-axis) against the number of simulations in the simplex algorithm.*

Figure 16 shows the observed water levels against simulations with the original model and the calibrated model. Here, it is clearly seen that the original model overestimates the amounts of water in the system during rainfall events. The calibrated model shows much better agreements with the observations. In fact, during the largest rain event on March 11$^{th}$ in the figure, the original model simulates a CSO event in the nearby CSO structure, which the observations show did not occur in reality. The calibrated model does not make this error.

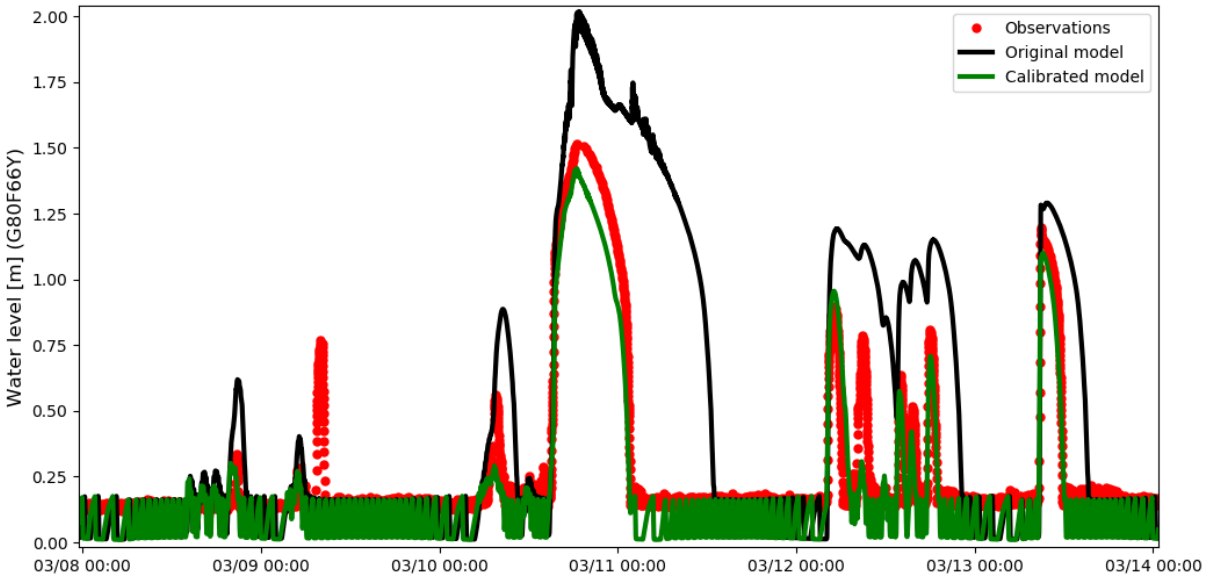
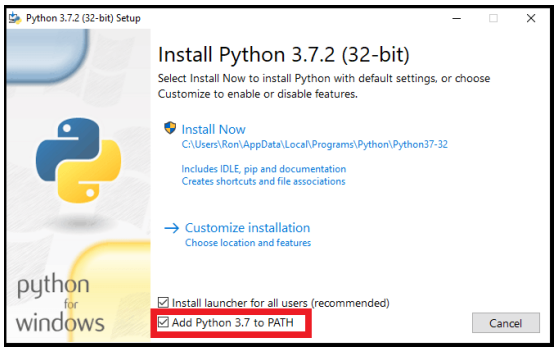



*Figure 16: Observed water levels against the simulated equivalents with the original model and the calibrated model.*

# Installation of NOAH Tool

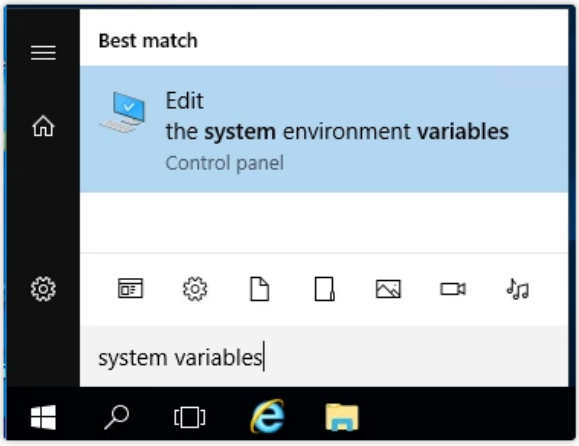
The following are required before you can install and run the program.

- Installation of Python version 3.7.3 (other versions might work but this one is tested.)
- Python should be set in System environment variables
    - This can either be done when installing Python, by checking the box at the bottom.

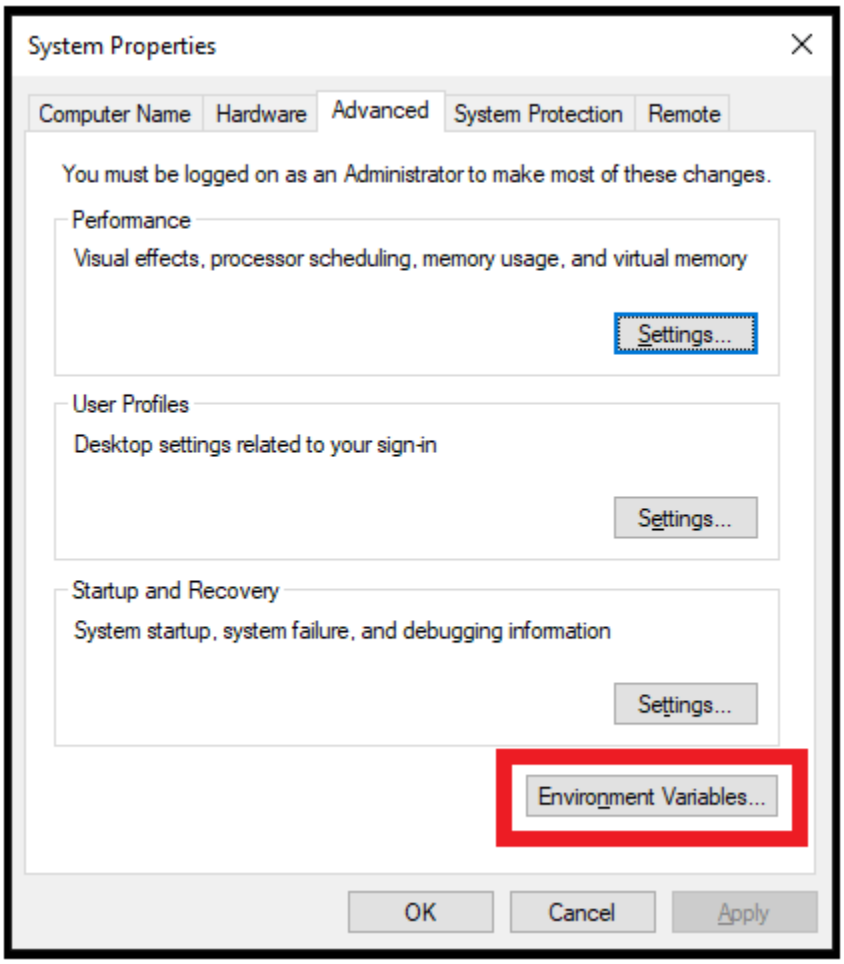



    - Or manually afterwards by following the steps below: (Pictures taken from https://datatofish.com/add-python-to-windows-path/):

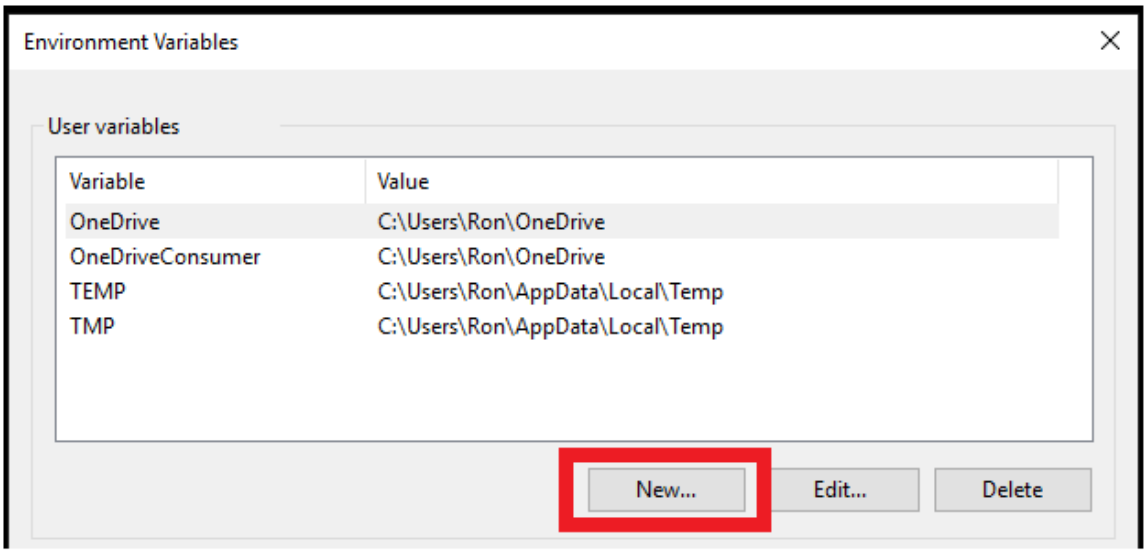
Search for system environment variables from the Windows menu.



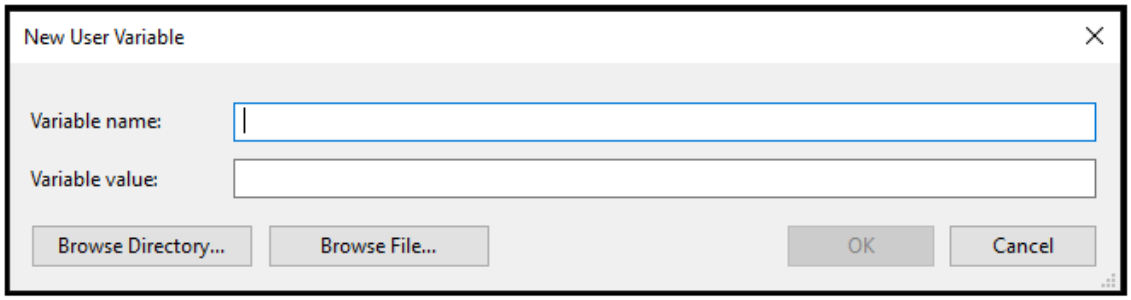

Open Environment Variables


System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...
User Profiles
Desktop settings related to your sign-in
Settings...
Startup and Recovery
System startup, system failure, and debugging information
Settings...
Environment Variables...
OK
Cancel
Apply

Select New

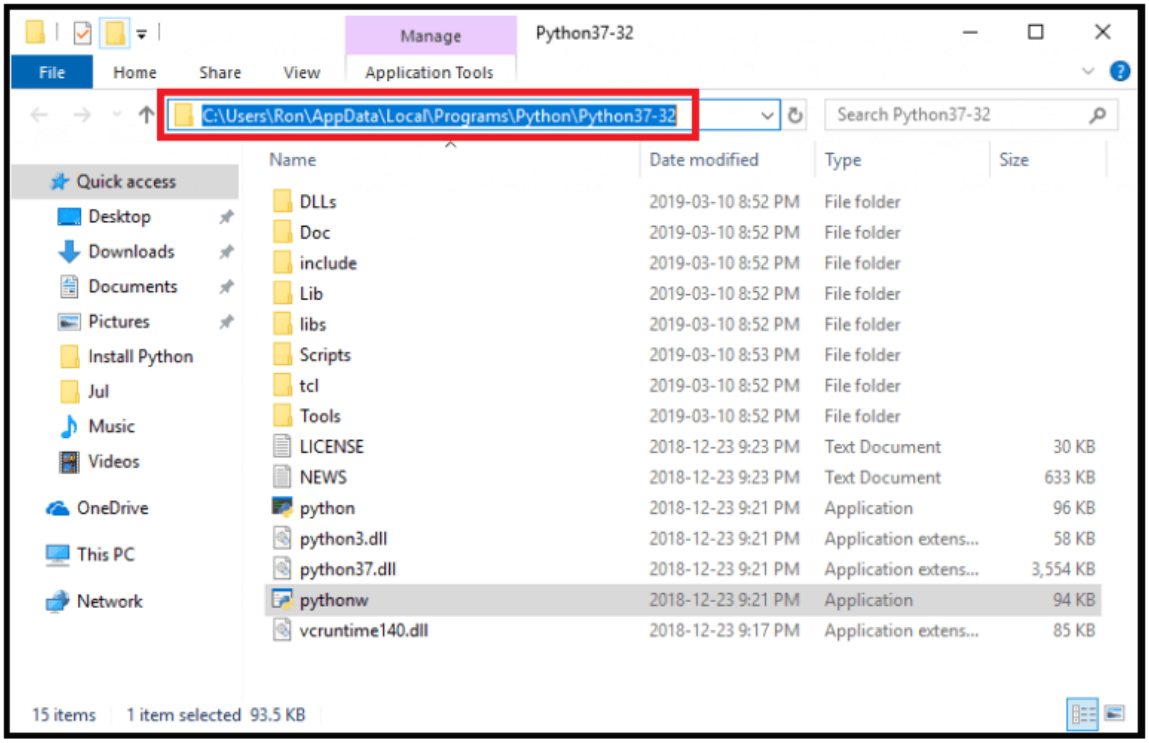

Environment Variables
User variables
Variable
Value
OneDrive
C:\Users\Ron\OneDrive
OneDriveConsumer
C:\Users\Ron\OneDrive
TEMP
C:\Users\Ron\AppData\Local\Temp
TMP
C:\Users\Ron\AppData\Local\Temp
New...
Edit...
Delete

The following window will appear



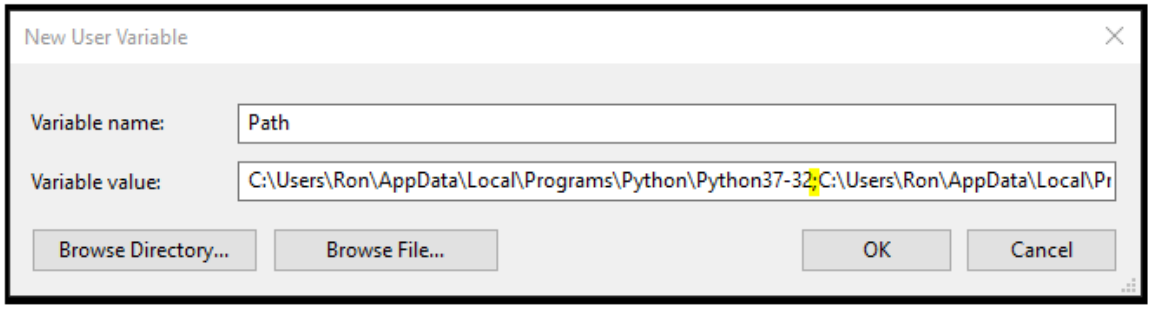

In the New User Variable window type in a name for the Variable (e.g Path).
In the “Variable value” field two paths are required.

The first is the path of your python installation:
Locate your Python installation and copy the path. The window should look like below

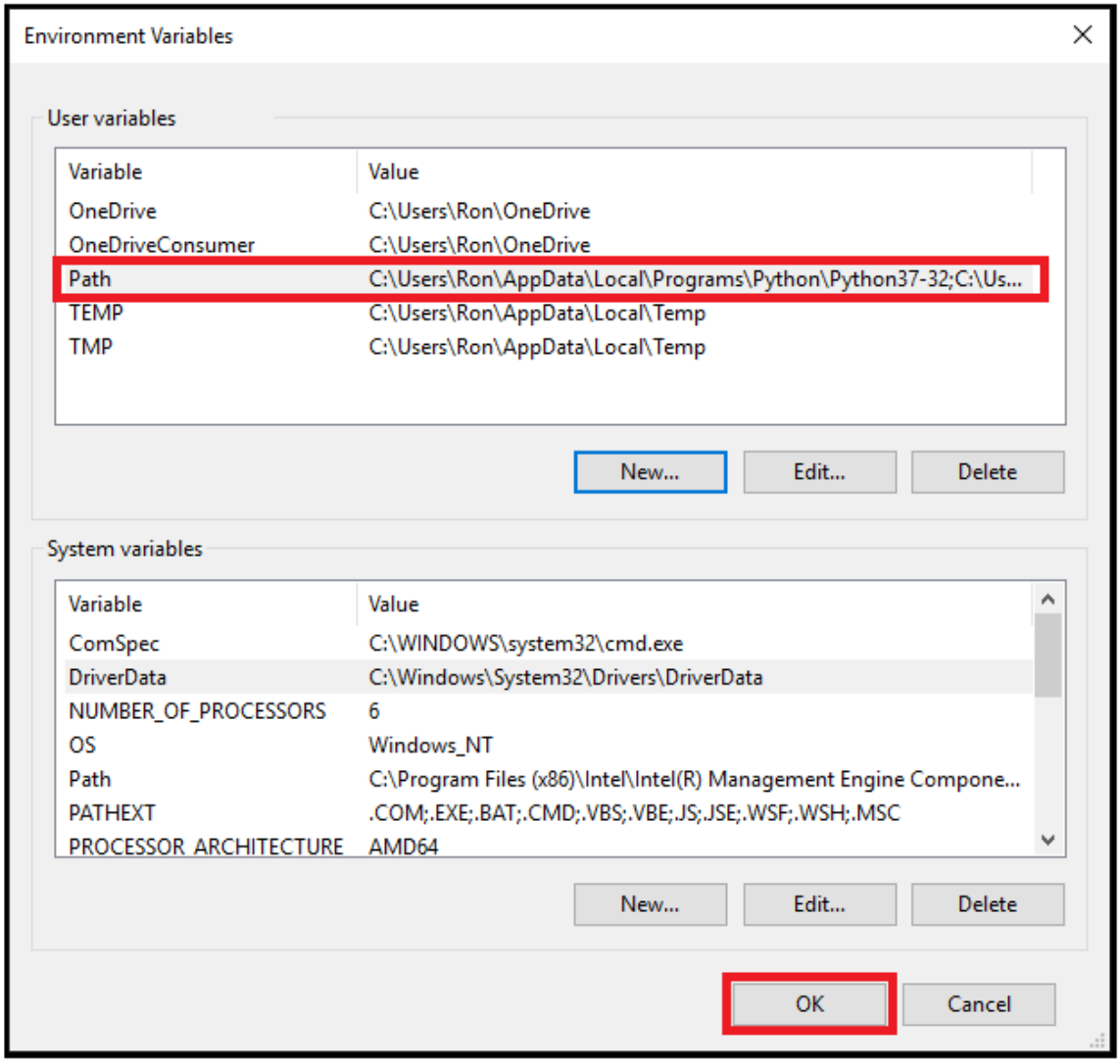



and insert the copied path as the first variable value.

Put a ; and insert the path again followed by “\Scripts”. This is the second path.
The “Variable value” should look like: *copied_path*;*copied_path*\Scripts as the picture below.




The new variable can now be seen.
Click OK and you are done.




- Clone the NOAH repository to a location on your computer. (https://github.com/mbjjo/NOAH)

To install the program, double click the *install* batchfile.

To execute the program, double click the *Run* batchfile.

All *print* statements and user massages will be shown in the windows console that appear in the background. If warnings occur or the program crashes the errors will be shown here.

Note that when the program is running it will be “not responding”. This means that simulations are running in the background. The status can be seen in the windows console.

# Appendix A: GUI and RTC functionality in the Python code library

This description applies to the version committed to github on June 30th, 2020. Features added after this day may not be included in this document

Link:

## Pyswmm_GUI

This module contains all widgets of the Graphical User Interface (i.e. the physical and visible components of the GUI). Run this script to run the NOAH Tool from python.

The structure is the following:

### *__init__:*

Creates the window and settings such as size, title, iconbitmap (the small icon in the top corner) etc. Also executes all subsequent functions.

### *create_widgets():*

Here all visible components are defined.
First the top part is defined. This includes model selection and logo.
Then the notebook frame (the various tabs) are defined.
To add tabs it should be included here and afterwards content can be written.
The content of each tab is created in the following part. This makes up the majority of the script.

### *if __name__ == "__main__":*

In the very end of the document
Execute the program by starting the mainloop.

## Gui_elements

This module contains all functions that are not visible in the GUI and not related to the actual computation of the simulations.

### *Run():*

This is executed when the *Run* button is activated. Either the configuration file will be written, or the simulation will be run with the existing configuration file.

### *Write_config():*

Write all parameters that are given in the GUI to a configuration file with the same name as the model. This file is saved in *\NOAH_RTC_Tool\config\saved_configs* and can be edited with any text editor. If *Overwrite existing configuration file* is not checked the simulation will run with the most recent version of this file and most input in the GUI is ignored.

If the *save config* button is pushed the configuration file is written without running the simulation.

### *Parameters():*

This class contain all parameters that are used before the actual computation. This includes parameters that defines what kind of simulations that are run, that are shown in the GUI or changes the appearance of the GUI.

Also, parameters that are defined via radio buttons or checkboxes needs to be defined here because these set an existing variable to a given value or string.

*Read config parameters():*

Reads the parameters from the corresponding configuration file and saves them to be used in the Python code.

*Calibrate_with_config():*

Wrapper function for the Calibrate() function. Loads the configuration file.

*Calibrate():*

Computes all functions related to the calibration tab with the input specified in the GUI.

The functions that are used here are the ones written in *noah_calibration_tools.py* and *model_structure_analysis.py.*

Objective functions for calibration:

Four redefined objective functions are defined here.

This includes Root Mean Square Error, Nash Sutcliffe Efficiency, Mean Annual Error, and absolute relative peak error.

*Tooltip/create_Tooltip():*

Creates a small text that appears when the cursor is above a widget. Useful when explanations are required.

*Small functions:*

A range of small functions that are used when changing states in the GUI based on input to the GUI.
These are typically specific and only used on a few widgets each.

*OpenFile():*

Opens a dialog that allow the user to choose a SWMM model.

*Select_obs():*

Opens a dialog that allow the user to choose the data for the calibration.

*User_msg():*

The function is activated every time a simulation is run. Gives useful information about status of simulation, computation time etc.

*Results_plot():*

Makes a plot with some data depending on the input.
Also saves the plot as a .png in the output folder.

*First_step_optimization_plot():*

Creates a plot of the first step of the optimization. Calls *Reults_plot()*.

NB. The plots for the results are in GUI_emelents.py while the text is in the *Optimizer* class in pyswmm_Simulation.py.

The functions related to the results are not very intuitively structured. Ideally a class is created with all functions related to the results so that these are streamlined and easier to edit. This is part of the future work.

## Pyswmm_Simulation

This module contains all code that is related to the computation of SWMM models and processing of the results.

### *Optimizer:*

This class contains all functions related to the optimization of the RTC.
All following functions are methods within this class.

### *Init():*

Initializes the class and parameters required before the optimization can begin. This includes creating an output directory for the results and choosing the correct optimizer.

### *read_config():*

This method reads the parameters from the most recent configuration file with the same name as the model.

The try/except ValueError: clause is only applied where the type is required to be float. This ensures that spaces can be left blank without causing an error. However, if the parameter is needed for the computation the error occurs at a later stage. This can be troublesome if it for instance occur after the simulation when the results are to be written since this will cause loss of results.

A way of validating the configuration file and checking that all required parameters are correct should be implemented in the future.

### *write_SWMM_controls(x,filename):*

The SWMM file is copied with the filename as suffix and controls are inserted before simulation. The setting (X) is what is being optimized.

### *Redefine_Timeseries():*

This method changes the Timeseries in the SWMM model to the correct directory since it would else require that all external files are in the *lib* directory.

### *Two_step_optimizer():*

This is the actual optimizer. It contains two parts:

Part 1 is an initial screening of the parameter space defined by input in the Optimization tab in the GUI.
This is used to save time in the optimization since the starting point is chosen more accurate. Also, this part allows the user to easily get an overview of whether RTC shows a potential or not.

Part 2 is the "finetuning" of the optimization setting. It begins at the lowest point of the screening and uses a build in python optimizer to find the lowest point. This typically requires more simulations and computation time.

Part 2 will only be computed if the "Maximum simplex simulations" is greater than 0.

*simulation(x):*

This method computes each simulation. The input parameter x is the one that is optimized in the Two-step-optimizer and that is to be determined. If optimization is not applied the activation depth from the GUI is used.

*Objective functions:*

The result processing that returns the objective value is computed in these functions.

This is either number of events or volume either from CSO structures or flooding

*write_optimal_result():*

This writes a text file that is shown after the optimization. As mentioned, the structure of the results is not intuitive and should be improved.

## Adding parameters to the code

When new parameters are to be added to the python code the following steps must be done.

- Add the visible widgets where the parameters must be typed in in pyswmm_GUI.py. (E.g. entry, radiobutton, Checkbutton etc.)
- Add the parameter in the *write_config()* function in GUI_elements.py. This ensures that the configuration file contains the parameter.
- Add the parameter in the *read_config()* function in the *Optimizer* class in pyswmm_Simulation.py and in the *Read_Config_Parameters* class in GUI_elements.py. This ensures that the parameter can be used in the simulation.
- Add the parameter to the *parameters* class in GUI_elements.py. Only necessary if the parameter is used before the actual computation. I.e. if it defines what kind of simulation that should be run, if it is shown somewhere in the GUI or if it changes the appearance of the GUI. (Examples are model name, model directory, whether optimization is applied.)
- Add the parameter in the actual computation in pyswmm_Simulation.py.