HELMHOLTZ CENTRE FOR ENVIRONMENTAL RESEARCH – UFZ

Optimizer

Description of Usage

Contributors:
Burkhard Oelschlägel, Uwe Franko, Eric Bönecke, Katharina Meurer, Nadia Prays, Felix Witing

Optimizer Version 2015

Introduction

Optimizer (optimizer.exe) is a program which can be used to fit model parameters to observed data and calculate sensitivities of model parameters. Although originally designed for the models CANDY and CCB, optimizer.exe can be used for calibration and optimization of any model which uses a database structure.

Generally a model can be seen as a composite of several inputs (constants, drivers, and initial values), a calculating kernel, and an output dataset (Figure 1).

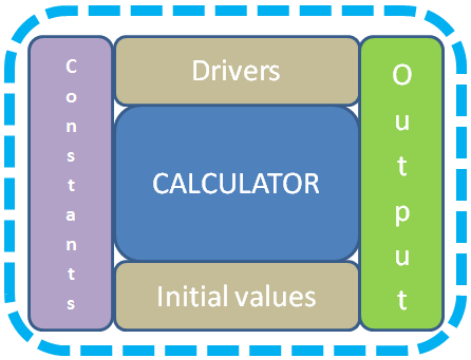

Figure 1: General model scheme.

The optimizer has a slightly different view point: all parts of the original model from Figure 1 are wrapped in one model unit (Figure 2, dashed blue box). This model unit is surrounded by a 'parameter' dataset (which can be related to any model input, (Figure 2, brown boxes)) and by a result dataset (containing actual model outputs together with specific destination values – the target (Figure 2, green boxes). The optimizer calls this more abstract model composition (Figure 2, red dashed box) and makes an assessment of the results (error function value (Figure 2, right blue arrow)) to decide about changing the problem-specific parameters by applying the simplex algorithm (Figure 2, left blue arrow).

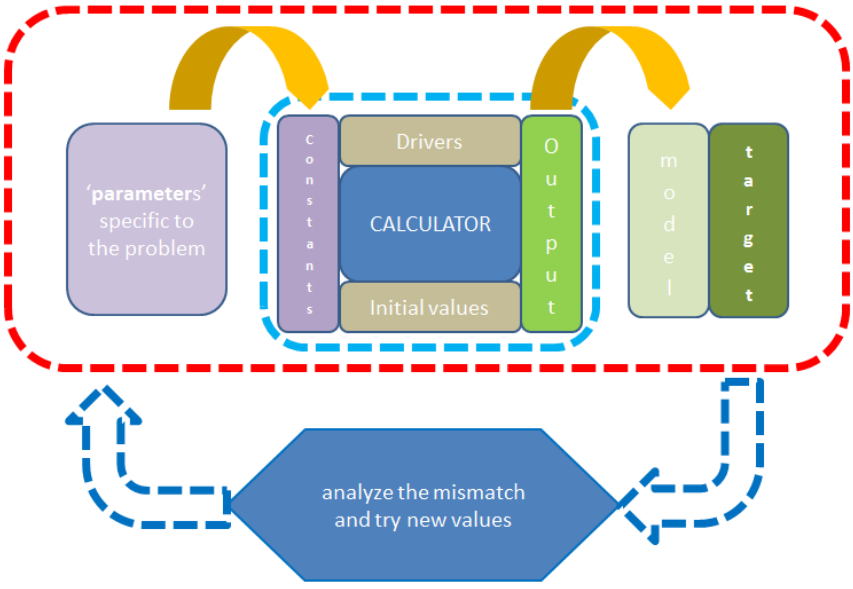

Figure 2: Optimizer framework.

Using the optimizer requires (Figure 3):

- the selection of items (parameters of the problem)
- specification of the objective (values that should match with the model outputs)
- the model call

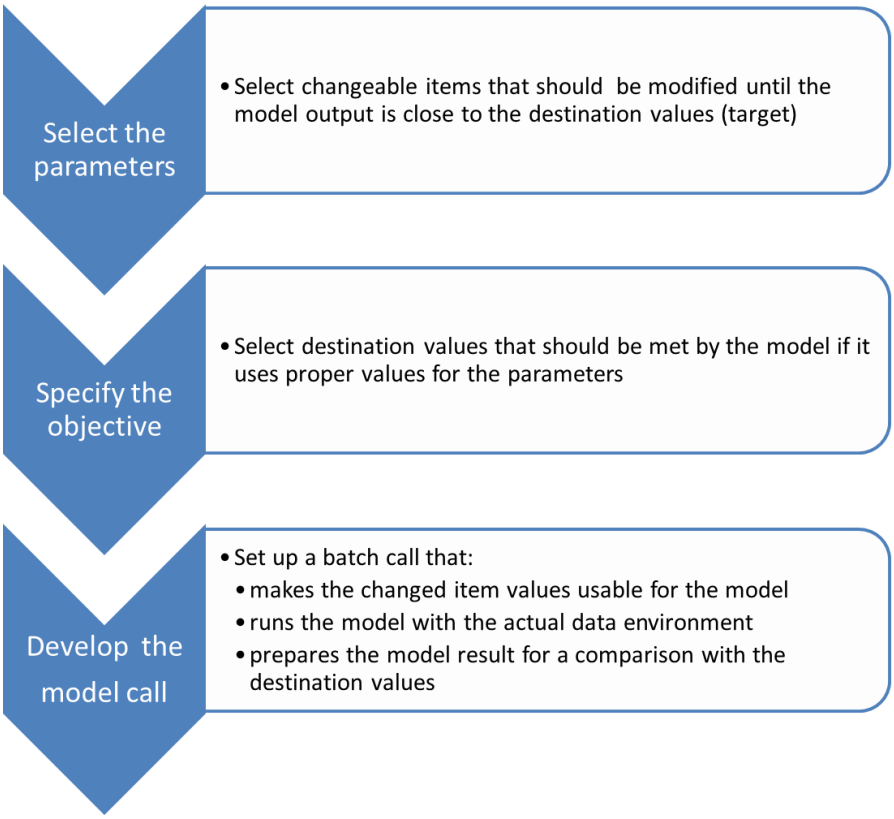

Figure 3: Requirements of the optimization set up.

To solve an optimization problem the optimizer.exe requires an ACCESS database with one table describing the parameters of the problem (here called "parameter interface") and another table containing the destination values (further on called "result table"). Furthermore a BATCH program is required containing a code that includes the following steps:

a) preprocessing: transfer the actual parameters to the proper datasets of the inner model

b) model call: run the inner model with this changed data environment

c) postprocessing: transfer the required outputs to the result table

The optimizer changes the parameter values following a special algorithm until the best fit between both datasets (model & target) is found or the maximum number of iterations is reached. The quality of the fit is calculated for each iteration step. It is given by the value of an error function (usually the sum of squared deviations).

Good modeling practice demands that the modeler provides an evaluation of the confidence in the model. This requires i) a quantification of the uncertainty in any model results (uncertainty analysis) and ii) an evaluation of how much each input is contributing to the output uncertainty. Sensitivity analysis addresses the second of these issues, performing the role of ordering inputs by importance. It is analyzing the impact of the inputs on the variation of the output. Model calibration is typically done using only the most sensitive model parameters.

The sensitivity analysis of optimizer.exe is a local sensitivity analysis, which is analyzing the effect of a single parameter change on the model results. It is designed to give an overview on parameter sensitivities while having only little computational demand. Local methods of sensitivity analysis are addressing sensitivity relative to point estimates of parameter values. They do typically not attempt to fully explore the input space. Methods of global sensitivity analysis examine sensitivity with regard to the entire parameter distribution.

There are several other tools that might support the optimization process:

- **parmadapt.exe** (used to select data from the database that is relevant for the optimization and write the data into model-specific input files.)
- **resimp.exe** (used to import simulation results from model-specific output files into the database and to compare the results with measured data (objective function))
- **oma.exe** (master call for sequential optimization of multiple tasks)
- **sqlpro.exe** (used for post processing of the results. Executes calculations within the database via SQL queries)

Theory

Simplex algorithm

The optimization procedure of the optimizer.exe is based on the approach of Nelder and Mead (1965), also known as downhill simplex method. This numerical method is used to find a minimum of a nonlinear function having more than one independent variable (Press et al. 1992). During a series of steps the process estimates a local optimum of a problem until it possesses a unique value. The concept uses the geometric figure of a simplex, which, in n dimensions, consists of n + 1 vertices (points) and interconnecting line segments. Hence, in one dimension the simplex is a line, in two dimensions a triangle and a polyhedron for three dimensions. Each point corresponds to a parameter set for which a functional value can be calculated and compared with the other points of the simplex. A decision process selects for each iteration step if the "worst" point W is replaced by a new and (hopefully) better point, while the "best" point is kept. The movement to the optimum is realized by three specific operations: reflection, contraction, and expansion.

For minimizing the function y_i at P_i (with y_w = max(y_i) and y_B = min(y_i)) the procedure is as follows:

1. Ordering of points according to their function values as in $y_B < y_i < \cdots < y_N < y_W$ (1:

$$y_B < y_i < \cdots < y_N < y_W \quad (1)$$

Please note that the index "i" is the general index and "W", "N" and "B" describe the worst, next worst and best points.

2. Calculation of the centroid of all points (except for the worst point) ($M = \frac{1}{n}\sum_{i=1}^{i\neq w} P_i$ (2):

$$M = \frac{1}{n}\sum_{i=1}^{i\neq w} P_i \quad (2)$$

3. Reflection of worst point W as in $R = (1 + \alpha)M - \alpha W$ (3:

$$R = (1 + \alpha)M - \alpha W \quad (3)$$

4. If the function value y_R of the reflected point R lies between y_B and y_W, the actual worst point W is replaced by R If y_R is lower than y_B (the current best) there is a new minimum and the point R is expanded to E ($E = \gamma R + (1 - \gamma)M$ (4):

$$E = \gamma R + (1 - \gamma)M \quad (4)$$

If $y_E < y_B$, W is replaced by E; else the expansion has failed a W is replaced by R.

5. In the case that the result for the previous reflected point R is between the worst and the second worst case $y_W > y_R > y_N$ a new W is calculated by contracting W using $C_R = \beta R + (1-\beta)M$
(5:

$$C_R = \beta R + (1-\beta)M \quad (5)$$

If the reflection was not successful ($y_R \geq y_W$) this one dimensional contraction is performed from the initially identified worst point ($C_R = \beta W + (1-\beta)M$
(6):

$$C_R = \beta W + (1-\beta)M \quad (6)$$

Assuming the all previous trials for improvement failed ($y_E > y_W$), new points are calculated by a n-dimensional contraction (shrinkage) towards the best point B ($N_{new} = \frac{1}{2}(P_i + \mathrm{N})$ and $W_{new} = \frac{1}{2}(P_i + \mathrm{W})$ *(7)*
and the process is restarted.

$$N_{new} = \frac{1}{2}(P_i + \mathrm{N}) \quad \text{and} \quad W_{new} = \frac{1}{2}(P_i + \mathrm{W}) \quad (7)$$

Within the next iteration step N_{new}, W_{new} and B will be re-ordered according to $yB < y_i < \cdots < y_N < y_W$ (1
and labelled accordingly.

The parameter values used for reflection (α), expansion (γ) and contraction (β) are $\alpha = 1$, $\beta = 0.5$ and $\gamma = 2$.

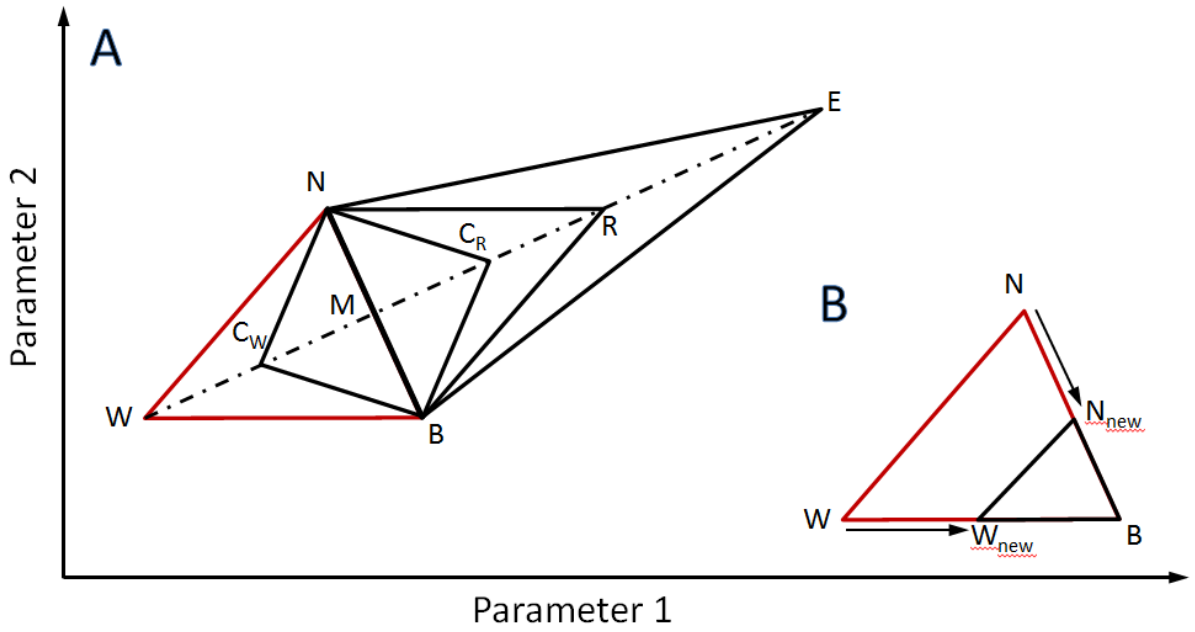

Figure 4 A: Basic triangle simplex is BNW (red) with the worst point W, next best point N and best point B. M is the centroid of all P_i without W, thus of B and N. R is a reflected W. E is reflected and expanded W. C_R and C_w are contraction points from R or W depending on the result of the reflection. Adapted from Bezerra et al. (2016) and Nelder and Mead (1965). B: If the direction is wrong and reflection, expansion and contraction failed, a multiple contraction of the basic triangle BNW towards the best point B takes place and results in a new triangle $BN_{new}W_{new}$.

Uncertainty calculation with Fisher Information Matrix

The Fisher Information Matrix (FIM) is calculated to get additional uncertainty information about the optimized parameters for the best fit. It requires the sensitivity of the model output f and the variance of the observed values at each observation point.

The sensitivity S is calculated for each observation point o following $S(i,o) = \frac{\Delta f_o}{\Delta f_i} = \frac{f_o\left(p_i^{opt}\right) - f_o\left(1.01 * p_i^{opt}\right)}{-0.01 * p_i^{opt}}$ (8:

$$S(i,o) = \frac{\Delta f_o}{\Delta f_i} = \frac{f_o\left(p_i^{opt}\right) - f_o\left(1.01 * p_i^{opt}\right)}{-0.01 * p_i^{opt}} \quad (8)$$

Every parameter p_i is increased by 1% keeping the optimized values for all other parameters.

The sensitivity describes the importance of each observation point for the parameter estimation.

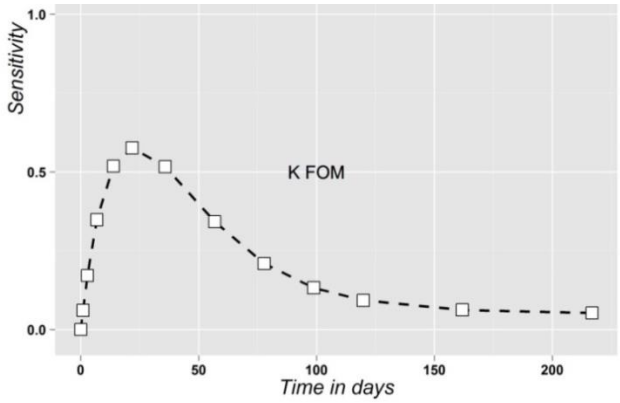

Figure 5: Example of the parameter sensitivity over time, showing the decreasing importance of the observations after 50 days.

Knowing the sensitivities, the FIM elements for all parameter combinations can be calculated ($[\mathrm{FIM}]_{ij} = \sum_o \frac{1}{\sigma_o^2} \frac{\partial f_o}{\partial p_i} \frac{\partial f_o}{\partial p_j} \approx \sum_o \frac{1}{\sigma_o^2} \frac{\Delta f_o}{\Delta p_i} \frac{\Delta f_o}{\Delta p_j}$ (9). FIM is a symmetric matrix ($[\mathrm{FIM}]_{ij} = [\mathrm{FIM}]_{ji}$).

$$[\mathrm{FIM}]_{ij} = \sum_o \frac{1}{\sigma_o^2} \frac{\partial f_o}{\partial p_i} \frac{\partial f_o}{\partial p_j} \approx \sum_o \frac{1}{\sigma_o^2} \frac{\Delta f_o}{\Delta p_i} \frac{\Delta f_o}{\Delta p_j} \quad (9)$$

The inversion of FIM provides the covariance matrix [COV] ($[FIM]^{-1} = [COV] = \begin{bmatrix} \sigma_1^2 & \cdots & \sigma_{1n} \\ \vdots & \ddots & \vdots \\ \sigma_{1n} & \cdots & \sigma_n^2 \end{bmatrix}$ (10):

$$[FIM]^{-1} = [COV] = \begin{bmatrix} \sigma_1^2 & \cdots & \sigma_{1n} \\ \vdots & \ddots & \vdots \\ \sigma_{1n} & \cdots & \sigma_n^2 \end{bmatrix} \quad (10)$$

[COV] contains the relevant statistical information about the optimized parameters. The main diagonal consists of the variances σ_i^2 and the other elements represent the covariance σ_{ij} between parameter i and j.

The mean parameter value including the standard deviation is: $p_i \pm \sqrt{\sigma_i^2}$. The correlation between two parameters is: $r_{i,j} = \frac{\sigma_{ij}}{\sigma_i \sigma_j}$.

Handling

After opening the optimizer.exe, the user has the possibility to

- select the parameters, which have to be optimized,
- define the result table, which includes the measurement data on which the parameters shall be optimized
- define the batch call for the model run
- start a sensitivity analysis of a specified dataset and/or directly start the optimization process
- calculate the uncertainty of the parameters after successful optimization

Definition and selection of parameters

The sheet [parameters] of the optimizer user interface enables the selection of parameters, which should be integrated in the optimization routine. Therefore, they have to be listed in the PARM_INT table of the database. If not yet existing, the PARM_INT table will be incepted in the database after starting the optimizer.exe and opening a database. Otherwise the [change database] button can be used to select a new or another database. Selection of parameters can be made over the user interface via [add parameter record] (Figure 6) or directly in the PARM_INT table using i.e. Microsoft ACCESS. If PARM_INT is created over the interface of optimizer.exe and more than one parameter is added, additional attributes for the correlation between parameters are inserted.

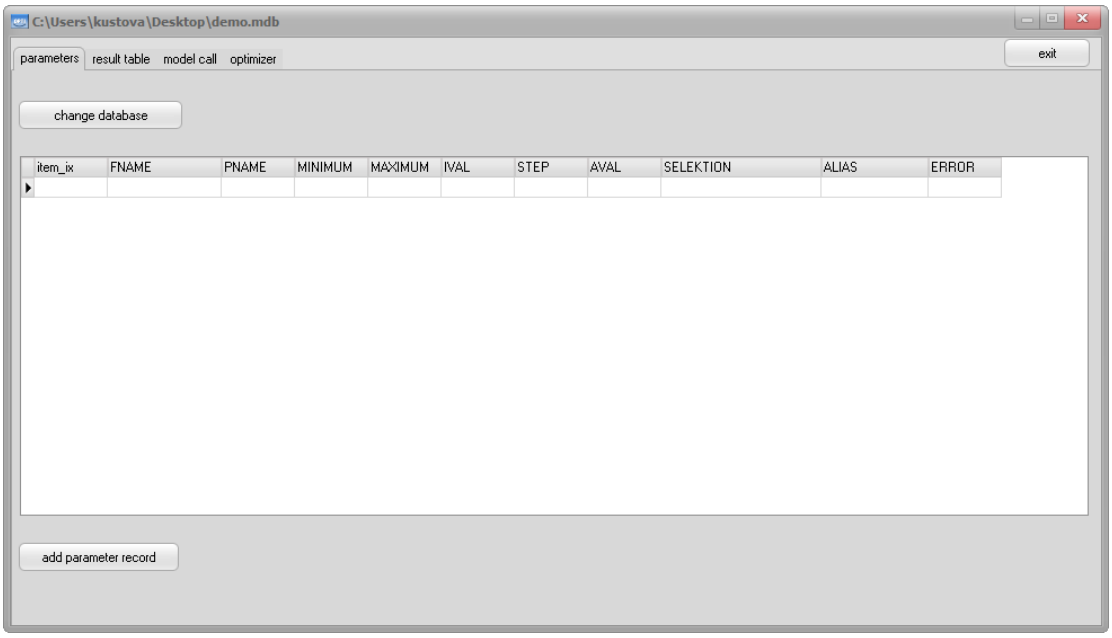

Figure 6: User interface for the parameter definition.

The definition of the parameters can be done by clicking the [add parameter record] button. In the parameter definition dialog (Figure 7) you may select the required parameters from the specific table (double-click). All columns of the table are listed in the window "selectable parameters". Select the required parameter and the corresponding record in the "selectable records" window as well as the "index field" for a unique identification of this record. Insert the selection into the PARM_INT table by clicking the [use selection] button. Proceed in the same manner for additional parameters.

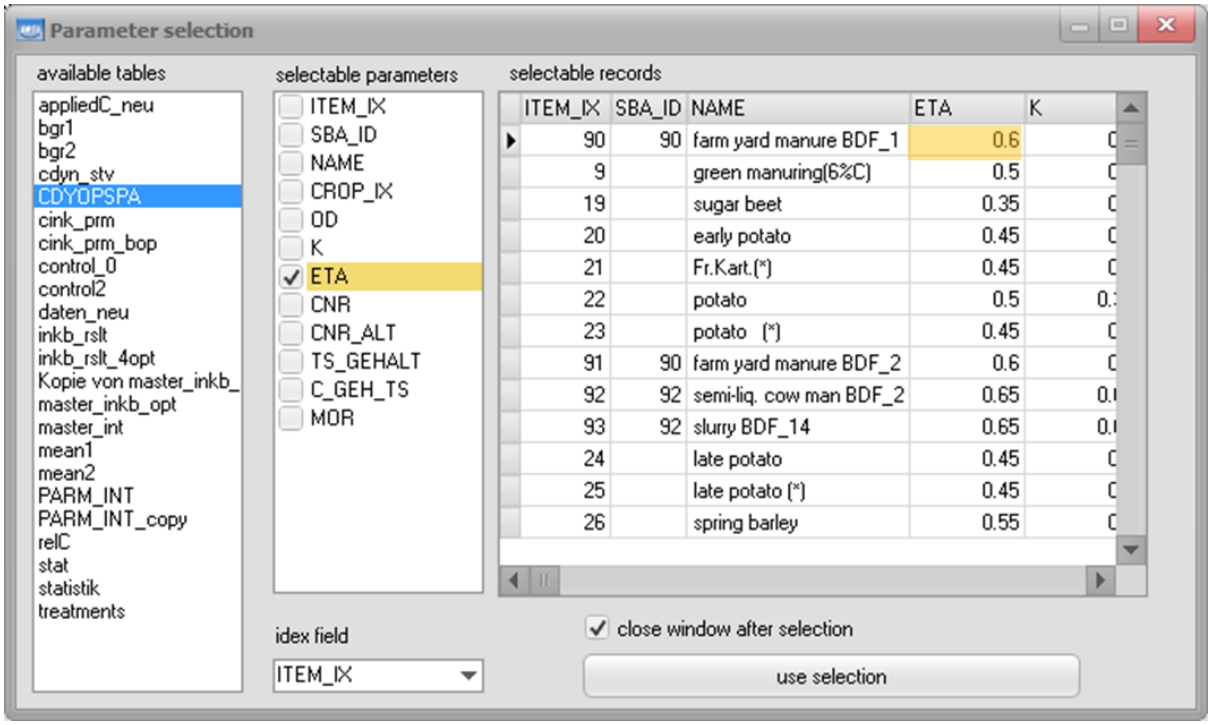

Figure 7: Parameter selection dialog for adding a new parameter record.

The PARM_INT table now contains closer specifications about the origin table (*fname*), the column (*pname*), and selection criterion (*selection*) of each parameter (Figure 6).
Standard values for e.g. maximum and minimum values will be inserted automatically, after selecting a record with the [add parameter record] button. The minimum value is set as a tenth and the maximum value is the threefold of the current parameter value. However, it is important to ensure that minimum and maximum values are in a scientifically appropriate range in order to get reasonable parameter estimation.
Before starting an optimization, it is recommended to test the sensitivity of the error function for each parameter. This is automatically accomplished by increasing the initial value IVAL for each parameter by a defined step width (STEP). The current parameter value of each optimization step, as well as the final optimized value, is stored in the AVAL column.
If you detect a non-reasonable record in the PARM_INT table (Figure 6) you can delete it by using Ctrl + Del.

The structure of the PARM_INT table is as follows:

attribute	**Meaning**
item_ix	unique index
fname	table containing the parameter
pname	attribute name (column) of the parameter
selection	unique selection criteria to define a special record in *fname*
minimum	lower limit of the parameter used for optimization
maximum	upper limit of the parameter used for optimization
ival	initial value; starting point of optimization
step	step width for the first optimization run
aval	currently optimal value
error	standard deviation of the parameter
alias	symbolic name for the parameter (user specific)

Definition and selection of results

By clicking on the sheet [result table], the user can select the database table that contains the data, which shall be used to calculate the distance between the model and the target within the error function. Selection of the specific table can be made via the drop down menu (Figure 8, red box). With the filter condition (Figure 8, blue box), data can be further limited to select only meaningful records and leave out records that are not sensitive to parameter variation. If no limitation is required, the input has to be a true condition, e.g. “1 = 1”. It ensures that all the data from the selected file are shown in the window below (by clicking the [open … as result table] button) and, consequently, taken into account for the optimization process.

Information about the model and observed value are mandatory. The specification of an index field (required for grouping observations) and observed variance (required

for uncertainty analysis) are optional. All selections are made via the drop down menus at the bottom of the window (Figure 8, green box).

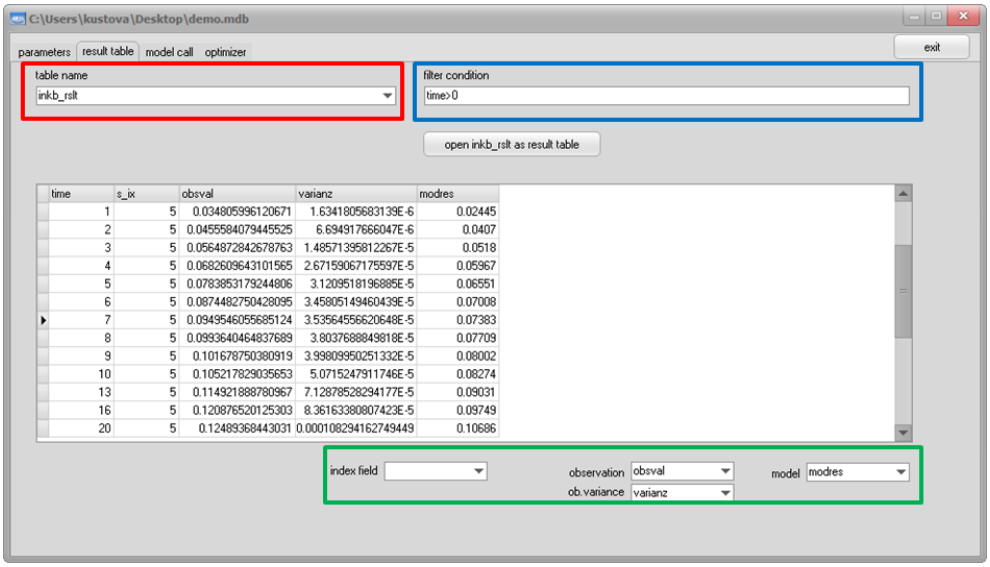

Figure 8: Specification of the objective function: Selection of table including measurement values (red box), filter conditions (blue box), and indication of observed and simulated values within the data table (green box).

Model Call

To specify the model call click on [open runfile] within the [model call] sheet and select the right batch file which contains the model algorithm and possible pre- and postprocessing steps for storing the results (Figure 9). The content of the batch file will be visible in the window below. Changes can be made by the [edit runfile] button. It is recommended to test the model run by using the option [check model run] before starting the optimization. The model run was successful if it didn't crash and the result file is updated.

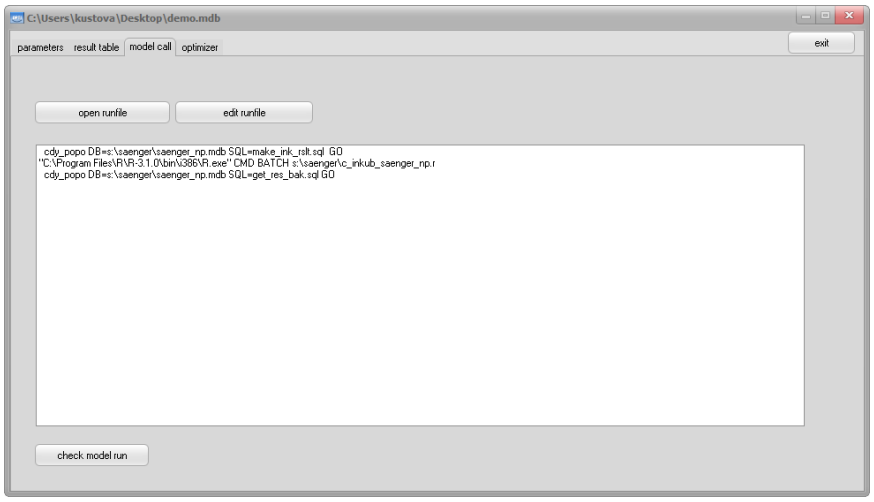

Figure 9: Example for a model call.

Optimizer

In the "optimizer" sheet you can start the optimization process as well as a couple of supporting procedures. For different purposes it is possible to select the preferred error assessment (Figure 10, yellow box), running a sensitivity analysis (Figure 10, green box), as well as an uncertainty analysis (Figure 10, blue box), and start the optimization process within limited settings (Figure 10, red box).

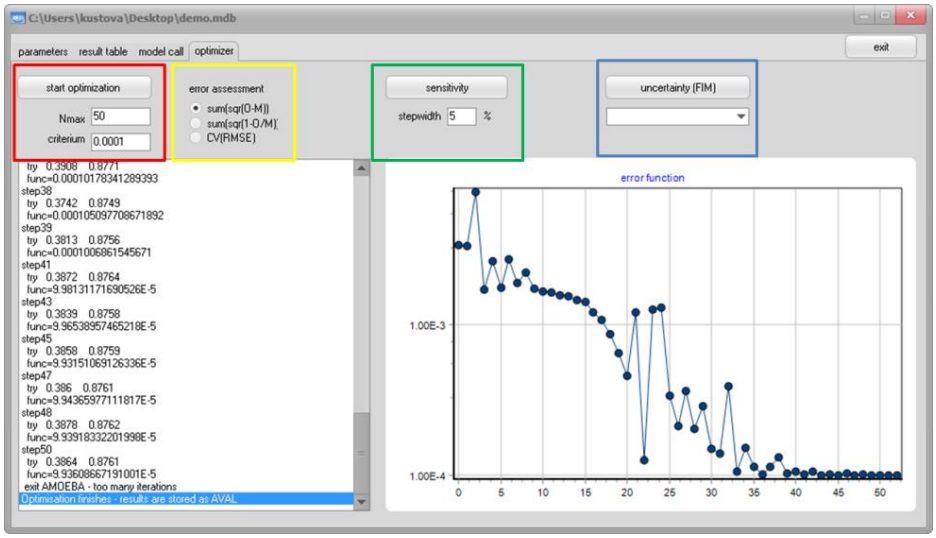

Figure 10: CANDY parameter optimization.

Error Assessment

The area "error assessment" provides the user the possibility to choose between three types of error functions/objective functions (Figure 10, yellow box):

- root mean square error: $RMSE = \sqrt{\frac{\sum(Obs-Mod)^2}{n}}$
- sum of normalized RMSE's for different observation types (groups): $NRMSE = \sum \frac{RMSE_i}{range(Obs_i)}$
- sum coefficient of variation of the root mean squared error for different observation types (groups): $CV(RMSE) = \sum \frac{RMSE_i}{\overline{Obs_i}}$

The last two options are recommended if the set of observations is not homogeneous (i.e. the observations have different units). In this case the index field of the result table (figure 8) is used to identify the groups.

Sensitivity analysis

Before starting the optimization process, it is strongly recommended to test the sensitivity of the parameters (Figure 10, green box), in order to avoid an unnecessary optimization of parameters that do not have a strong impact on the values within the objective function.

During the sensitivity analysis each parameter is only once increased by a given factor. The sensitivity analysis can be adapted by changing the parameter "step size", given as percentage of the specific parameter value (default: 5 %).

Adaptation of optimization effort

Depending on the task the optimization may take some time. Therefore the user can define two restrictions (Figure 10, red box). With "Nmax" you can define the maximum number of iteration steps the optimizer can run for finding the optimal parameter value. The "criterion" defines the condition to finish the minimum search of the error. The iteration will stop if the relative difference of the extreme values (max and min) of the error function at the corresponding simplex points is lower than the given tolerance criterion C_{tol} (here 0.0001):

$$2 * \frac{|Y_{max} - Y_{min}|}{|Y_{max} + Y_{min}|} < C_{tol} \qquad (11)$$

Results are stored in the PARM_INT table within the *aval* column, but can also be found in an extra text file (opti_result.txt), which is stored in the same directory as the optimizer.exe.

Finishing and results evaluation

The finishing window of the optimization provides two options (Figure 11): First, to use result values for e.g. uncertainty analysis with Fisher Matrix Information or, second, to exit using the initial parameter values (Figure 11, green box).

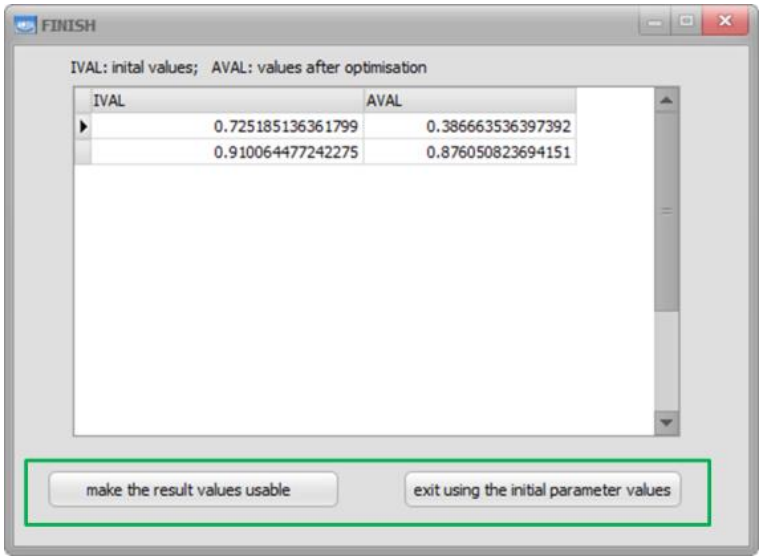

Figure 11: Finishing window of the optimization.

To check your results you have to pay attention to a few things.

1. Check in opti_result.txt if the number of iteration steps was sufficient and if the optimization process has finished successfully.

2. Did you choose the right range for minimum and maximum parameter values?
3. The Nelder-Mead-method is sensitive to the initial values. Change the starting point of optimization (IVAL) to make sure, that the global minimum of the error function was found instead of a local minimum.

Uncertainty analysis

This feature is only available if the result table contains information about the variance of the observations. It will be skipped if the "ob.variance" field (Figure 8) is left empty.

"Uncertainty (FIM)" (Figure 3, blue box) means uncertainty calculation from the Fisher Information Matrix. This process includes additional model runs. The first run uses the optimal values of the parameters, followed by model runs where each parameter is changed by 1%.

You can find the result of the uncertainty calculation in the text file (opti_result.txt). There optimized parameters with mean value, standard deviation (±), and correlation between the parameters are stored. The results are also kept in the PARM_INT table containing a record for each parameter:

- *error*: standard deviation of this parameter
- *r_n*: correlation coefficient between this parameter and the *n*th parameter.

With the drop down selection each parameter can be chosen to check the sensitivity of each observation. Names are visible when parameters have an alias name in the PARM_INT table. The graphic can be helpful for the proper selection of the dataset and for future experiment design (compare with Figure 5).

Batch call of the optimizer

Calling the optimizer.exe is possible via BATCH call using the following parameters:

DB=<database file>

RT=<result table with observations and model results>

CD=<selection condition if not all records of the result table shall be included>

OB=<field name (column) of the observation data>

OV=<field name of the variance data>

MO=<field name of the model results>

IX=<field name of the index that separates different data types>

EF=<error function: RMSE/NRMSE/CV(RMSE)>

NX=<maximum of iteration steps>

KR=<tolerance criterion for the optimal solution>

RF=<run file (batch) for the model call>

RS_<dataset name> : textfile with the optimization protocol

GO : start the optimization process

Example:

```
optimizer.exe DB=CCB_bdfsa.mdb RT=cn_result CD="id = 5 and year = 2015"
OB=meas_value OV=? MO=sim_value IX=M_IX EF=CV(RMSE) NX=100
KR=0.0001 RS_Clossa_17 RF=modcall.bat GO
```

Additional Tools

In case that the model input and/or output are only available in *.txt files, the tools parmadapt.exe and resimp.exe might be helpful to integrate the data into the optimization scheme (Figure 12).

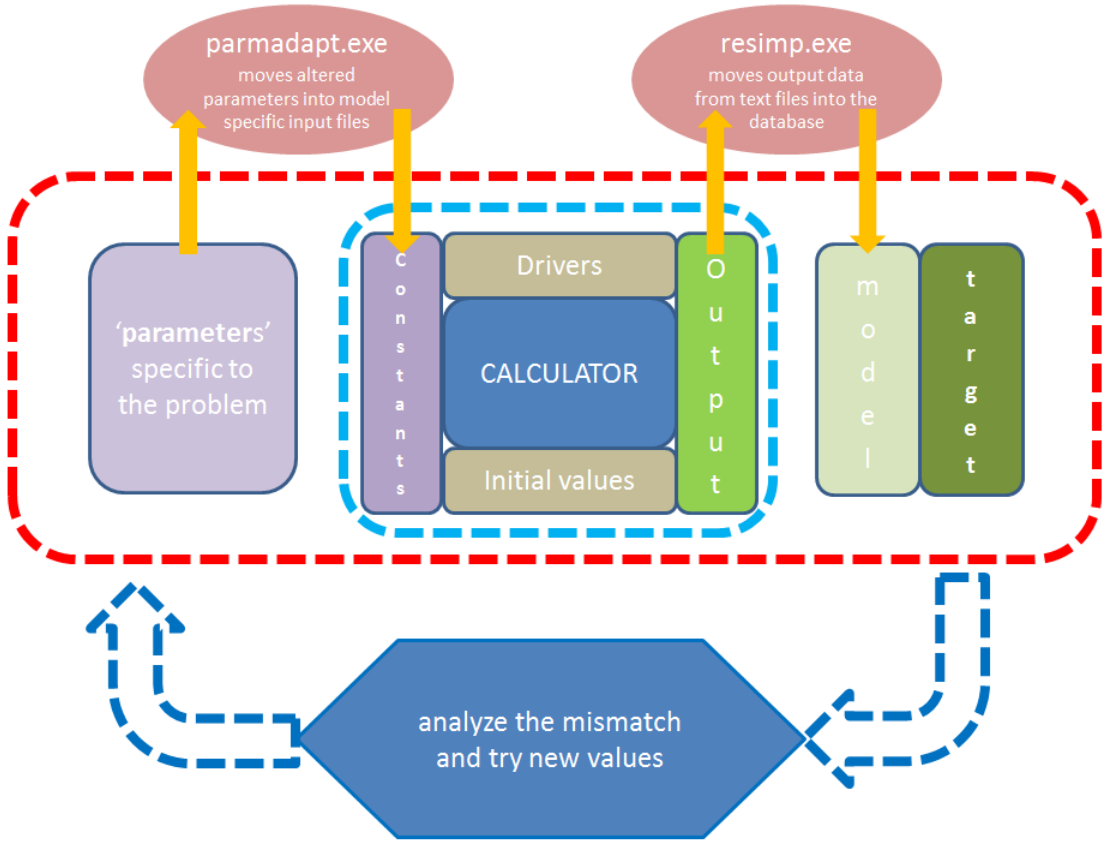

Figure 12: Interface connection of parmadapt.exe and resimp.exe within the optimizer framework to enable the work with text files.

Moving data into text files using parmadapt.exe

The parmadapt.exe provides the opportunity to automatically update data from the database into *.txt files. This can be a helpful tool when using the optimizer.exe with models that do not have a database interface. In this way parameters that are to be optimized can be updated within the input files of the model before each model run. Each data item is considered as a list of the elements: '0'..'9','.','-'.

The name of the *.txt file, as well as exact information about the position (line and column) of the parameters have to be given. The interaction between the elements is visualized in Figure 13.

Description of the specific parameters has to be given in a new table within the database:

attribute	**meaning**	**type**
parameter	name of the parameter (not mandatory)	Text
fname	name of the original *.txt file into which the parameters have to be updated (has to be in the same directory as the database)	Text
line	line number of the parameter within the *.txt file	Integer
item	column number of the parameter within the *.txt file	Integer
cntnt	value of the parameter which has to be updated	Double
comment	space for your own information (not mandatory)	Text
hrz	selection criteria of the parameter	Text

Calling the parmadapt.exe is possible via BATCH call by the following parameters:

DB=<database file>

CL=<table name containing the control list>

T=<lag time in ms> (to synchronize the system cache on slow PC)

GO process the parameters and wait for start

! start after processing the parameters (no wait)

Example:

```
parmadapt.exe  db=demo.mdb  t=40  cl=parmsoils  go  !
```

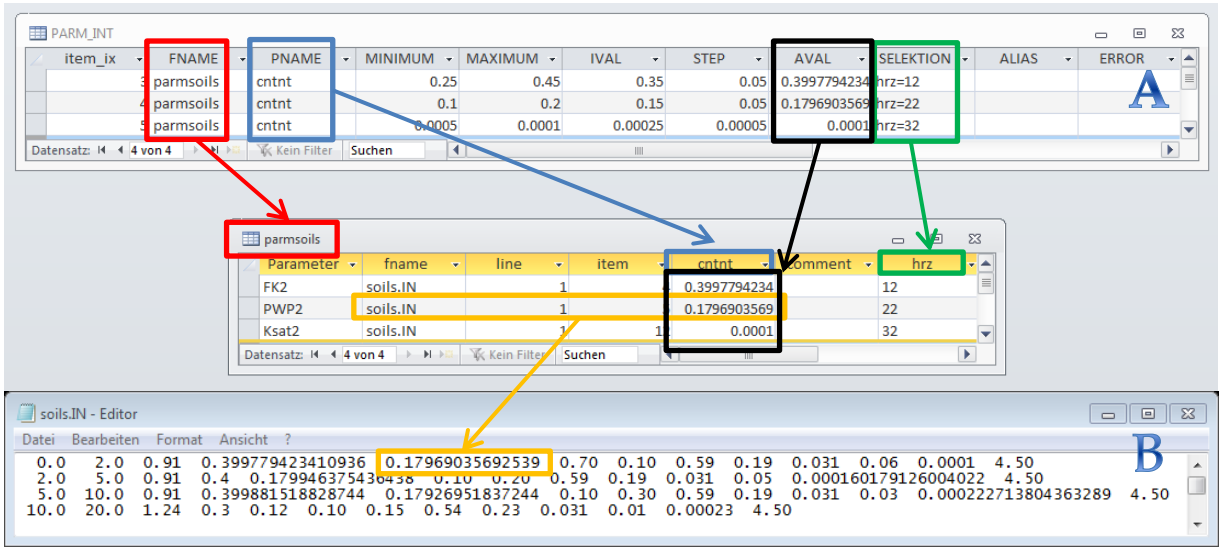

Figure 13: Links between A) the parameter interface (PARM_INT) of the optimizer and the mandatory parameter table (here "parmsoils") within the database, and B) the transfer from the database into a text file.

If parmadapt.exe is called without the "!" command within the batch call the interface appears as shown in Figure 14.

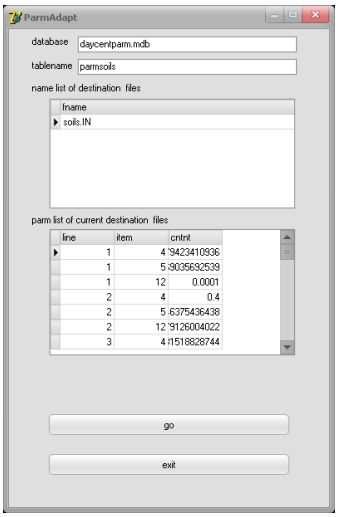

Figure 14: Interface of parmadapt.exe.

Extraction of data from text files using resimp.exe

With the resimp.exe the user has the possibility to load values of selected columns from *.txt files (e.g. output/result files) into the database. In this context a value is a list of the elements: '0'..'9', '.', '-', '/'. Selected data is stored in an automatically new created database table. It has to be noted that the resimp.exe additionally loads the first and second column of the *.txt file by default, since these columns contain date information in most cases.

Calling the resimp.exe via BATCH call requires information about the *.txt file and the column that has to be extracted. The following parameters are used:

DB=<database name>

RT=<result table> (if the call contains a "*" the table will be re-created)

HD=<number of headlines to be skipped of the text file to reach the required table without header>

I*x*=<item number x(x <=3) to be included in a list of variables (table column Wx) with type double>

or

IN=<number of a single input column from the text file additionally to the first two columns, that are supposed to contain year and day number >

Example:

```
resimp.exe  db=demo.mdb  rt=vwc2  in=3  fn=vswc.out  go  !
```

If resimp.exe is called without the "!" command within the batch call the interface appears as shown in Figure 15Figure 14.

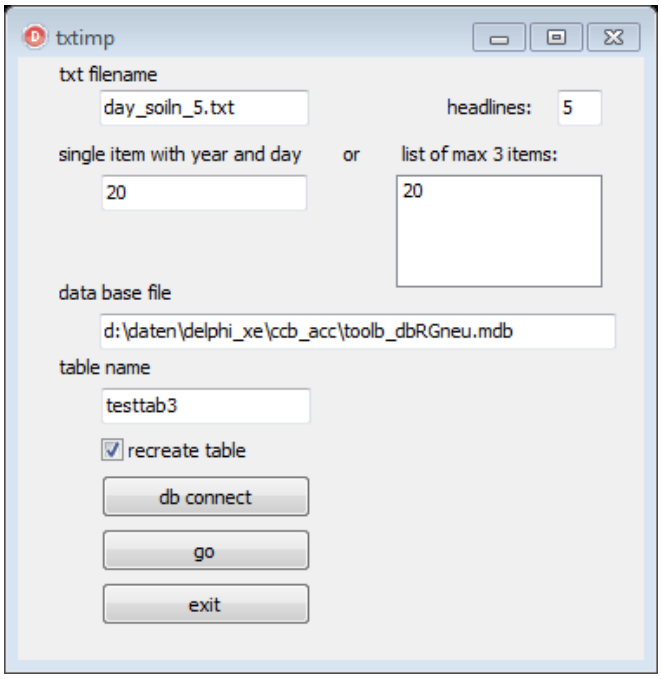

Figure 15: Interface of the resimp.exe.

The interaction between the elements is visualized in Figure 16Figure 13.

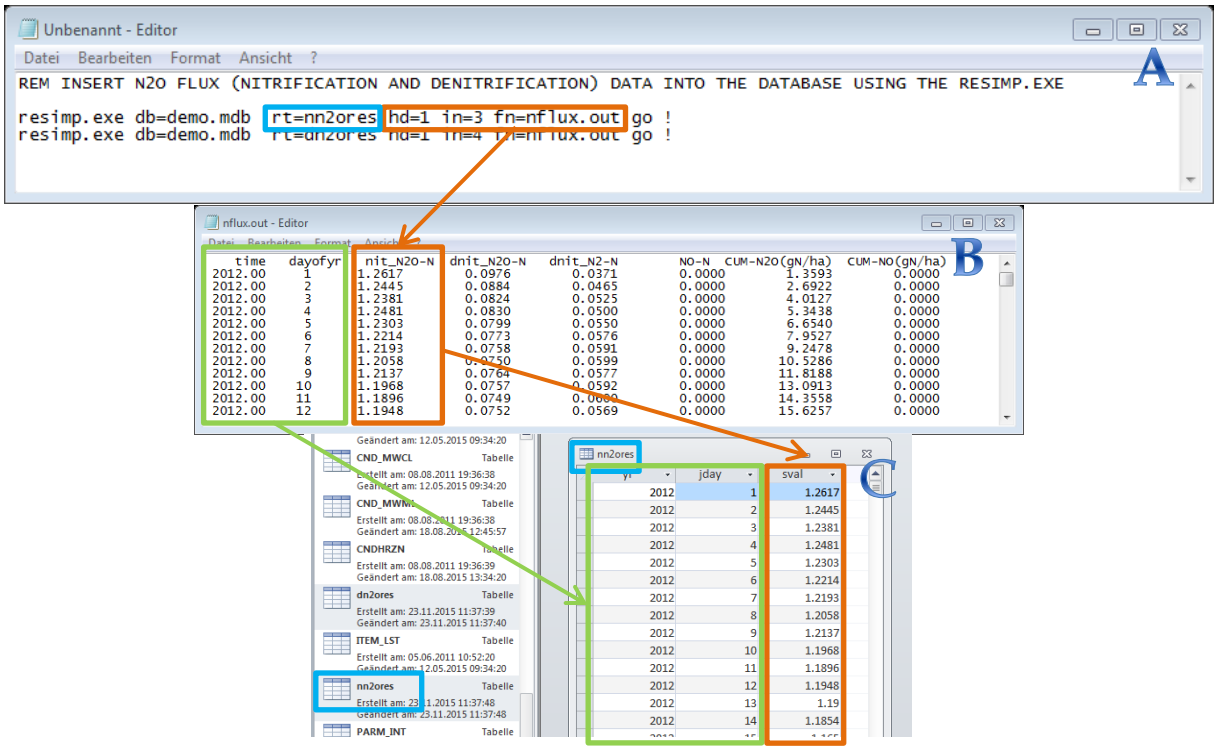

Figure 16: Batch call (A) for inserting specific rows of a model output file (here "nflux.out" - B) into the database (C).

Besides the optimization purpose the application of resimp.exe can be helpful to obtain more clarity when working with large text files.

Database processing using sqlpro.exe

To make the data suitable for the optimization process, it might be necessary to apply the sqlpro.exe to bring data into the result table. The sql processor sqlpro.exe is designed to execute scripts of several sql statements (Figure 17).

It can be called with the following parameters:

DB=<name of the ACCESS database>

SQL=<name of the text file containing the sql statements>

GO (optional)

$*xyz*=<value> (optional)

Remarks:

Without specifying "GO" in the program call, you see a window comparable to Figure 9. Here, it is possible to edit the script, check the parameters, and finally execute the script. If the last statement provides a dataset, it will be shown on the sheet [result data set].

Any string starting with "$" is interpreted as parameter in the SQL script. This is helpful to make sql scripts more flexible, for example when different treatments with changing IDs have to be addressed. All appearances of $*xyz* will be replaced by <value> prior execution of the script. The number of parameters is not limited. The use of meaningful names (other than $xyz) is strongly recommended.

Example:

```
sqlpro.exe  DB=demo.mdb  SQL=anything.sql  $snr=17  GO
```

Supported sql statements are:

- Select
- Insert
- Update
- Delete
- Drop

Remarks:

An additional (non sql) statement is the term "edit_table" that opens a table in an edit mode to change values.

Furthermore, it is possible to insert a dataset into a *.txt file using the keyword "totxt", followed by the desired name of a text file instead of "into" with a table name.

For a better documentation of more elaborated scripts it is possible to include comments by starting the line with the "#".

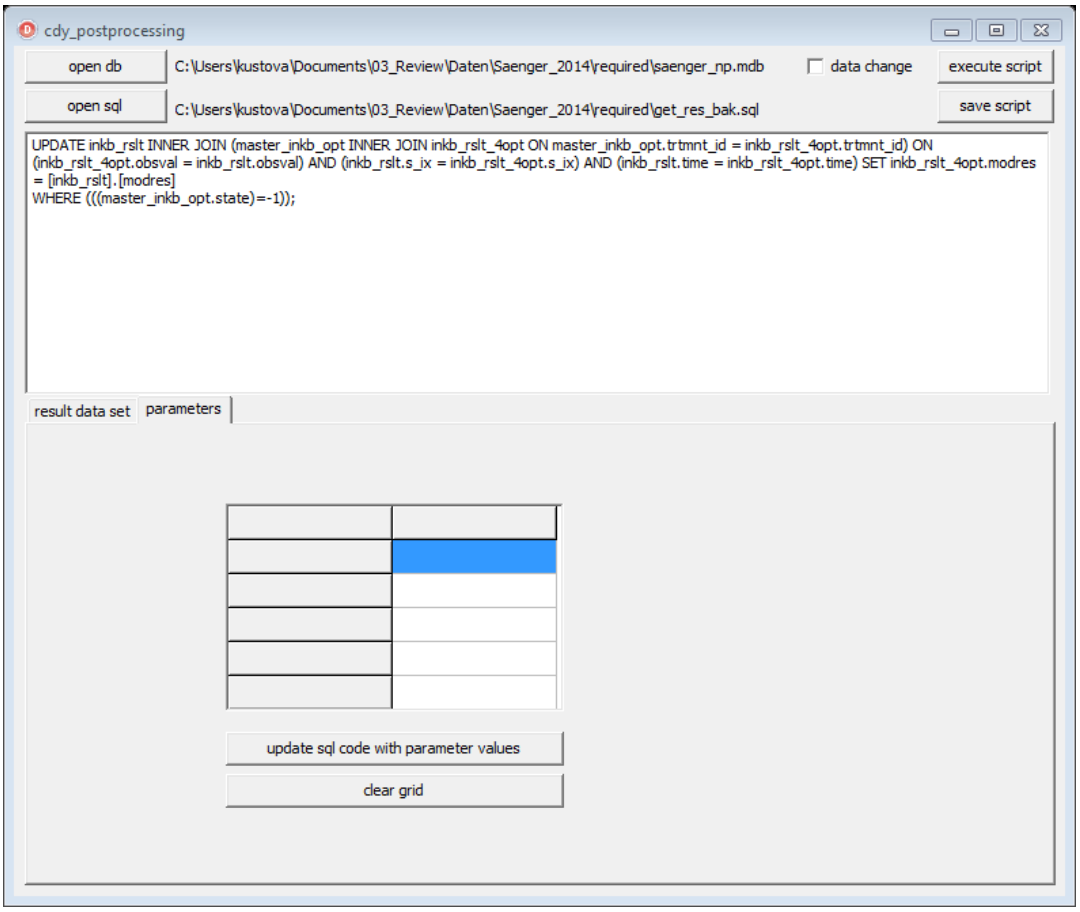

Figure 17: Interface for the sql processor.

Managing serial treatments using oma.exe

Oma.exe is an optimization master, which can be used for e.g. optimization of a series of treatments with almost one click. The user has to provide a template for the model call which includes preprocessing, model call, and postprocessing of the data similar to the optimizer. The call may contain a replaceable parameter symbol, which relates to the actual treatment as described in **Fehler! Verweisquelle konnte nicht gefunden werden.**, where "##" is used.

It is required to prepare the master table (Figure 18) with details about the infrastructure and the table "master_int", which is a collection of the parameter interfaces for each optimization problem.

master_opt

trtmnt	trtmnt_id	restab	condition	obs_field	mod_field	state	var_field	idx_field	errfunc
	110	tmp_cn_result	1=1	meas_value	sim_value	0	?	M_IX	NRMSE
	100	tmp_cn_result	1=1	meas_value	sim_value	0	?	M_IX	NRMSE
	130	tmp_cn_result	1=1	meas_value	sim_value	0	?	M_IX	NRMSE
	20	tmp_cn_result	1=1	meas_value	sim_value	0	?	M_IX	NRMSE
	30	tmp_cn_result	1=1	meas_value	sim_value	0	?	M_IX	NRMSE
	120	tmp_cn_result	1=1	meas_value	sim_value	0	?	M_IX	NRMSE
	140	tmp_cn_result	1=1	meas_value	sim_value	0	?	M_IX	NRMSE
	160	tmp_cn_result	1=1	meas_value	sim_value	0	?	M_IX	NRMSE
	90	tmp_cn_result	1=1	meas_value	sim_value	0	?	M_IX	NRMSE
	190	tmp_cn_result	1=1	meas_value	sim_value	0	?	M_IX	NRMSE
	170	tmp_cn_result	1=1	meas_value	sim_value	0	?	M_IX	NRMSE
	10	tmp_cn_result	1=1	meas_value	sim_value	-99	?	M_IX	NRMSE
	220	tmp_cn_result	1=1	meas_value	sim_value	0	?	M_IX	NRMSE
	50	tmp_cn_result	1=1	meas_value	sim_value	-99	?	M_IX	NRMSE
	250	tmp_cn_result	1=1	meas_value	sim_value	-99	?	M_IX	NRMSE

Figure 18 : Example for the master table. Here, the condition is set to 1=1 (true) for each treatment; therefore all records in the result table will be used with the error function NRMSE (errfunc) for separating the different observation types by means of the field M_IX in tmp_cn_result. No Variance field is specified (var_field: ?); therefore the results will contain no error information.

The structure of the master table is equal to PARM_INT with an additional attribute that is used as control field (trtmnt_id in Figure 19):

attribute	**meaning**	**type**
trtmnt_id	treatment index	integer
trtmnt	name of the treatment	text
restab		text
condition		text
obs_field		text
mod_field		text
state	-99 excluded record, -1 record to be processed, 0 included record	integer
Var_field	(optional)	text
Idx_field	(optional)	text
Errfunc	one of : RMSE / NRMSE / CV(RMSE) (optional)	text

master_int

trtmnt_id	FNAME	PNAME	MINIMUM	MAXIMUM	IVAL	STEP	AVAL	SELECTION	ERROR	alias
10	soilproperties	cif	0.001	0.99	0.5	0.01	0.21143104977	soil_id=19		cif
10	measurements	meas_value	0.0001	4.95	0.08	0.001	0.08973154271	MEAS_ID=681		nini
10	measurements	meas_value	0.165	4.95	0.77928	0.005	0.84597023293	meas_ID=31		cini
20	soilproperties	cif	0.001	0.99	0.5	0.01	0.99	soil_id=8		cif
20	measurements	meas_value	0.165	4.95	0.7	0.005	0.73256260759	meas_ID=41		cini
20	measurements	meas_value	0.0001	4.95	0.08	0.001	0.07304604600	MEAS_ID=636		nini
30	soilproperties	cif	0.001	0.99	0.5	0.01	0.01677889100	soil_id=10		cif
30	measurements	meas_value	0.165	4.95	0.56959	0.005	0.65670317951	meas_ID=608		cini
30	measurements	meas_value	0.0001	4.95	0.056	0.001	0.06604768591	MEAS_ID=672		nini
50	soilproperties	cif	0.001	0.99	0.5	0.01	0.00100000000	soil_id=22		cif
50	measurements	meas_value	0.165	4.95	1.4	0.005	1.59650899271	meas_ID=61		cini
50	measurements	meas_value	0.0001	4.95	0.14	0.001	0.15673803196	MEAS_ID=682		nini
70	soilproperties	cif	0.001	0.99	0.5	0.01	-99	soil_id=9		cif
70	measurements	meas_value	0.165	4.95	6.53004	0.005	-99	meas_ID=627		cini

Figure 19 : Master_int table: structure and example of a record set.

After a blank start of oma.exe you have to specify the database that shall be used. This automatically selects the directory of the database as working directory. From this database you have to select the master table (Figure 18) with all required information for the optimization run (see chapters "Definition and selection of parameters" and "Definition and selection of results"). This table should specify at least the names for columns with e.g. treatment ID as control field, treatment name for the result file, selection condition, observed data, simulated data, and variance of observed data (compare with Figure 8, green box). Then select the control field which acts as a key. Furthermore you have to limit the maximum iteration steps and define a value for the tolerance criterion at which the optimization is finished. After choosing the model call, click on [go] and start the serial optimization. The batch file will be called by the optimizer and needs to prepare the result table at each model call. Results will be saved in *.txt files in the database folder. After each model run the result table (here tmp_cn_result) and the columns obs_field and mod_field are used by the optimizer.

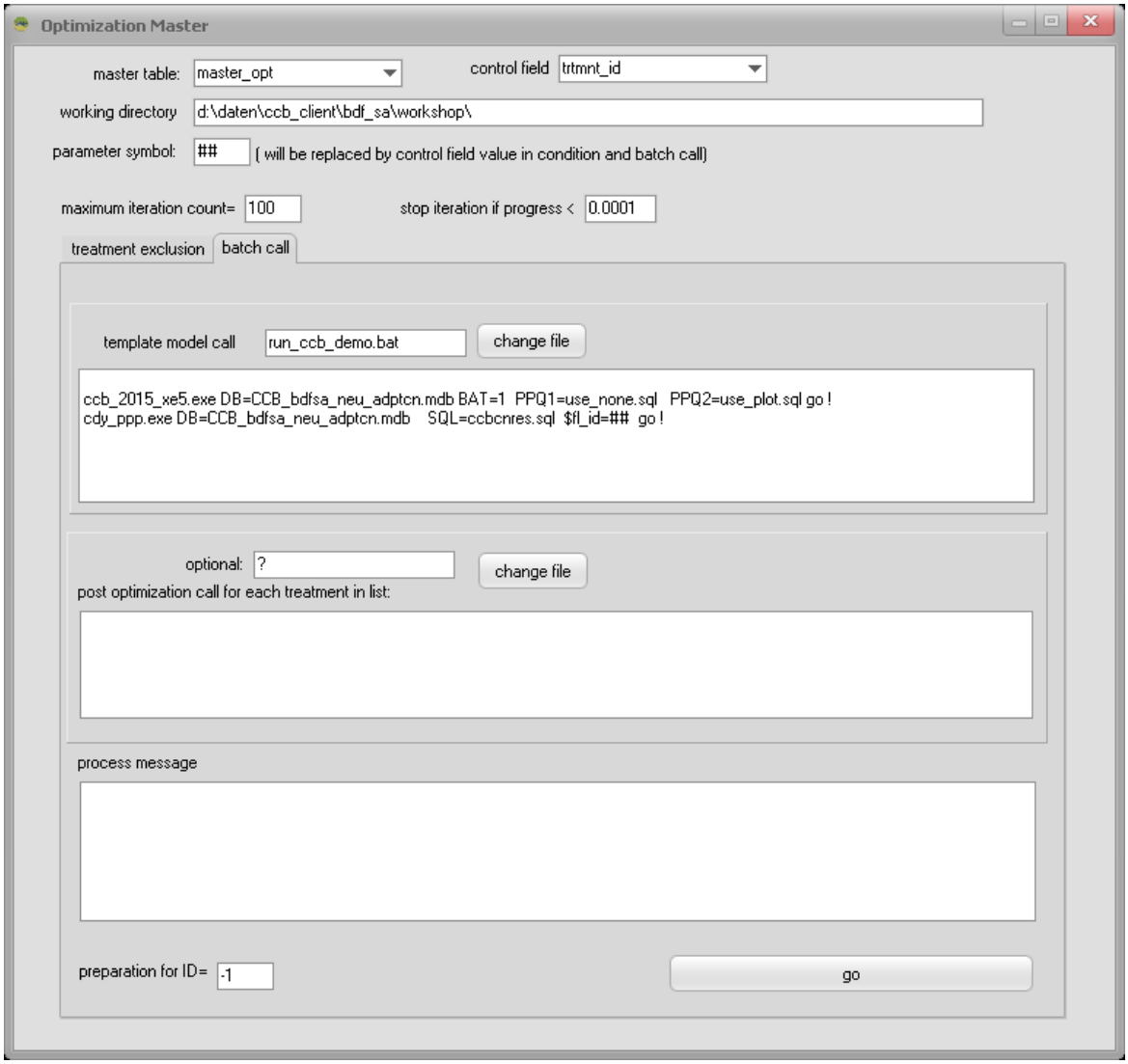

Figure 20: Optimization master for an example where "##" is used as the replaceable parameter.

For each record of the master table oma.exe will prepare and execute a temporary batch call for the optimizer (tmp_opti.bat) and a temporary model call (tmp_modcall.bat) that will be used by the optimizer. Furthermore, it is possible to have an additional batch file that is called after the optimizer has finished one task. These batch calls are based on the specified templates where the optional parameter string is replaced by the current value of the control field. If the optimization has finished, the result values of the parameters are transferred into the master_int table and a post optimization batch will be executed if prepared in advance.

Oma.exe itself can be called in a batch program using the following parameters:

DB=<database file>

MT=<name of the master table>

MF=<name of the control field (master field)>

RF=<template for the model call (may contain a parameter symbol)>

PF=<batch file that is executed after each finished optimization (optional)>

PRS=<parameter symbol that shall be replaced in batch files (optional)>

GO process is started automatically

Example:

```
oma.exe  DB=ccb_data.mdb  MT=master_opt MF=trtmnt_id  RF=run_ccb.bat  GO
```

Extended example using OMA

Problem: a model calibration has to be performed for a number of treatments where always the same parameters have to be adapted. Here we use the CCB model and want to find the optimal pool distribution at the beginning for eight different bare fallow plots.

The master_opt table looks like this (Figure 21: content fo the master_opt table in this exampleFigure 21):

master_opt

ID	trtmnt	trtmnt_id	restab	condition	obs_field	mod_field	state	var_field	idx_field	errfunc
1	Askov B3	1	simres	1=1	meas_value	sim_value	0	vrnz	fl_id	RMSE
2	Askov B4	2	simres	1=1	meas_value	sim_value	0	vrnz	fl_id	RMSE
3	Grignon	3	simres	1=1	meas_value	sim_value	-99	vrnz	fl_id	RMSE
4	Kursk	4	simres	1=1	meas_value	sim_value	-99	vrnz	fl_id	RMSE
5	Rothamsted	5	simres	1=1	meas_value	sim_value	-99	vrnz	fl_id	RMSE
6	Ultuna	6	simres	1=1	meas_value	sim_value	-99	vrnz	fl_id	RMSE
7	Versailles	7	simres	1=1	meas_value	sim_value	-99	vrnz	fl_id	RMSE
8	Lauchstädt Fex	8	simres	1=1	meas_value	sim_value	-99	vrnz	fl_id	RMSE

Figure 21: content fo the master_opt table in this example

We have specified where the system will find the observed values (obs_field) the simulation results (mod_field) and the variance of the observations that are available in this case. As an index field (idx_field) we use the fl_id despite it is not necessary as the error function (errfunc) is set to RMSE and anyway, we will optimize each treatment separately. To test the approach we have selected only the treatments 1 and 2.

The master_int contains all the information for the optimizer- here already containing the results for the first two treatments (Figure 22).

master_int

trtmnt_id	PNAME	FNAME	MINIMUM	MAXIMUM	IVAL	STEP	AVAL	SELECTION	ALIAS	ERROR
1	K_DEG	CDYAPARM	##########	##########	##########	##########	##########	ITEM_IX=1	kdeg	##########
1	c_am	som_state	2000	100000	16000	200	##########	fl_id=1	caos	##########
1	c_sm	som_state	4000	120000	28700	400	##########	fl_id=1	csos	##########
1	c_lts	som_state	500	150000	16000	50	##########	fl_id=1	clts	##########
2	K_DEG	CDYAPARM	##########	##########	##########	##########	##########	ITEM_IX=1	kdeg	##########
2	c_am	som_state	2000	100000	16000	200	##########	fl_id=2	caos	##########
2	c_sm	som_state	4000	120000	28700	400	##########	fl_id=2	csos	##########
2	c_lts	som_state	500	150000	16000	50	##########	fl_id=2	clts	##########
3	K_DEG	CDYAPARM	##########	##########	##########	##########	##########	ITEM_IX=1	kdeg	
3	c_am	som_state	2000	100000	16000	200	##########	fl_id=3	caos	
3	c_sm	som_state	4000	120000	28700	400	##########	fl_id=3	csos	
3	c_lts	som_state	500	150000	16000	50	##########	fl_id=3	clts	
4	K_DEG	CDYAPARM	##########	##########	##########	##########	##########	ITEM_IX=1	kdeg	
4	c_am	som_state	2000	100000	16000	200	##########	fl_id=4	caos	
4	c_sm	som_state	4000	120000	28700	400	##########	fl_id=4	csos	
4	c_lts	som_state	500	150000	16000	50	##########	fl_id=4	clts	
5	K_DEG	CDYAPARM	##########	##########	##########	##########	##########	ITEM_IX=1	kdeg	
5	c_am	som_state	2000	100000	16000	200	##########	fl_id=5	caos	
5	c_sm	som_state	4000	120000	28700	400	##########	fl_id=5	csos	
5	c_lts	som_state	500	150000	16000	50	##########	fl_id=5	clts	

Figure 22: abbreviated content of the master_int table

The batch call for oma.exe is:

oma.exe DB=d:\..\ccb_bf.mdb MT=master_opt MF=trtmnt_id RF=ccb_run.bat PF=post_opt.bat PRS=##

The batch files that are specified to run the model and for the post optimization procedure are ccb_run.bat:

ccb.exe DB=d:\..\ccb_bf.mdb ID=## go !
sql_pro.exe DB=d:\..\ccb_bf.mdb SQL=d:\..\get_res.sql go !
copy ccb_report.txt ccb_report##.txt

and post_opt.bat:

sql_pro.exe DB=d:\..\ccb_b.mdb SQL=d:\..\postopt.sql $id=## go !
copy ccb_report.txt ccb_report##.txt

In each batch file is included the execution of one sql script that prepares the results for the optimizer get_res.sql:

```
DELETE * from simres
go

INSERT INTO simres ( sim_value, meas_value, fl_id, [year], m_ix ,vrnz)
SELECT sim_result.sim_value, measurements.meas_value, sim_result.fl_id,
measurements.year, sim_result.m_ix,measurements.vrnz
FROM (sim_result INNER JOIN measurements ON (sim_result.year_num =
measurements.year_number) AND (sim_result.m_ix = measurements.M_IX) AND
(sim_result.fl_id = measurements.FL_ID)) INNER JOIN site_state ON sim_result.fl_id =
site_state.FL_ID WHERE (((sim_result.m_ix)=7) AND ((site_state.status)=1));
go
```

And collects the final resulst of the optimization including the uncertainty analysis postopt.sql:

```
DELETE from opti_result where fl_id=$id
go

INSERT INTO opti_result ( parm, optval, ERROR, r_2, r_3, r_4, fl_id )
SELECT parm_int.alias AS parm, parm_int.aval AS optval, parm_int.error, parm_int..r_2,
parm_int.r_3, parm_int.r_4, $id AS fl_id FROM parm_int
go

DELETE from sim_result_all where fl_id=$id
go

INSERT INTO sim_result_all ( sim_value, meas_value, fl_id, [year], m_ix ,vrnz)
```

```
SELECT    sim_result.sim_value, measurements.meas_value, sim_result.fl_id,
measurements.year, sim_result.m_ix,measurements.vrnz
FROM (sim_result INNER JOIN measurements ON (sim_result.year_num =
measurements.year_number) AND (sim_result.m_ix = measurements.M_IX) AND
(sim_result.fl_id = measurements.FL_ID)) INNER JOIN site_state ON sim_result.fl_id =
site_state.FL_ID
WHERE (((sim_result.m_ix)=7) AND ((site_state.status)=1))
go
```

Obviously, the master process will replace the symbol ## in both batchfiles with the current treatment ID. This is then used as parameter for the included program calls and as parameter for the postopt.sql with the construction $id=##

References

Bezerra, M. A., Q. O. dos Santos, A. G. Santos, C. G. Novaes, S. L. C. Ferreira and V. S. de Souza (2016). "Simplex optimization: A tutorial approach and recent applications in analytical chemistry." Microchemical Journal 124: 45-54.

Nelder, J. A. and R. Mead (1965). "A Simplex Method for Function Minimization." The Computer Journal 7(4): 308-313.

Press, W. H., S. A. Teukolsky, W. T. Vetterling and B. P. Flannery (1992). Downhill simplex method in multidimensions. Numerical recipes in C: the art of scientific computing. Cambridge University Press**:** 408-412.