UNIVERSITY OF AGDER

Automatic calibration of numerical model using artificial intelligence based techniques

Roger Tjosås, Tom Sverre Hageland

Thesis submitted in Partial Fulfillment of the Requirements for the Degree Master of Science in Information and Communication Technology

Faculty of Engineering and Science
University of Agder

Grimstad
May 2010

Abstract

Many energy companies rely on natural resources to produce energy. They use advanced models to estimate how much of those resources they have access to, but if a model is to make an accurate estimation it needs to be accurately calibrated.

There is little agreement in the science literature about what automatic calibration method is the best one to use on numerical model. The Shuffled Complex Evolution (SCE-UA) method is considered state of the art, and while it has been over 20 years since it was developed it is still in use both for commercial purposes and research.

We compared the SCE-UA method to three other methods that can potentially be used for parameter optimization; Continuous Action Learning Automata(CALA), Genetic Algorithms(GA) and a Monte Carlo Scheme. We implemented and configured these methods to run an implementation of the HBV hydrological model. The purpose of this was to see if the SCE-UA method was still the best one to use compared to these more general methods.

We designed a test protocol and an evaluation method to compare the methods on a level playing field. To be able to do this we had to research the characteristics of the methods and how to configure them to work with the HBV model.

Our results conclusively showed the SCE-UA and Genetic Algorithm methods giving the most accurate and efficient results. However, both their results were so similar that we could not make a decisive conclusion of which one of them was the best. We concluded that with our evaluation and test procedures they produced roughly equal results. The CALA method came out worse than any of the other methods.

Preface

This master thesis is submitted to fulfill the requirements for the degree Master of Science in Information and Communication Technology at the University of Agder,Faculty of Engineering and Science in Grimstad, Norway. The work carried out was under the supervision and guidance of associate professor Ole-Christoffer Granmo at the university of Agder, Norway.

First we would like to thank our supervisor Ole-Christoffer for the guidance and tips he gave us. He supportd and assisted us throughout the entire thesis.

Secondly we would like to thank Bernt Viggo Mattheussen and Jarand Røynstrand from Agder Energi. They provided the project and gave us valuable feedback whenever we asked for an input.

Grimstad, May 2010. Roger Tjosås, Tom Sverre Hageland.

Contents

List of Figures

1 Introduction

In this chapter we will discuss what a numerical model does, what its parameters are, and their importance to the model. We further discuss automatic model calibration and its importance to our thesis. We will also discuss some principles of Genetic Algorithms, CALA (Continous Action Learning Automata) and the Shuffled Complex Evolution approach described in the SCE-UA paper, and we show our contribution to the problem area.

1.1 Background

The hydrological power production industry uses models to estimate the runoff from snowpack in their water catchments. The model uses precipitation, air-temperature and geological data to produce an estimate of the daily runoff of a catchment. In order for the model to accurately estimate the water runoff of the catchment it has to be calibrated to that catchment. Because calibrating the model takes a lot of adjustments, it is necessary for the power industry to use automatic methods to do these adjustments. There is little concencus in the scientific literature on what kind of automatic calibration of hydrological model is the best. This project attempts to adress this field of knowledge by comparing some existing methods of automatic model calibration.

1.2 Importance of research

A numerical model attempts to mathematically reproduce the effects or state of an observed system. The complexity of the model is dependent on the complexity of the observed system, as well as the need for accurate results, by which we mean that the output of the model closely resembles observations of the modelled system. As more accurate results are required, the model of a complex system must become more complex.

The model produces an output by applying a mathematical formula to measured data. The characteristics of the modeled system is typically represented by a group of static parameters. The more complex the system, the larger the amount of parameters needed to characterize the different factors in the system.

If we think of a numerical model as a mathematical function, a parameter can be described as a single variable in that mathematical function. Modifying a variable will have a certain effect on the outcome of the function depending on what relation the variable has to the rest of the function. When we modify this value, we calibrate the model. If, after modifying the value the modelled output is better than before, according to some arbitrary measure such as comparison with historical observed values, we have calibrated the model to better represent the modelled system, even though the mathematics in the function remain unchanged.

The goal of calibrating the parameter set in this way is to find a parameter set with which the model produces an output that is a close fit to an observed or arbitrarily pregenerated dataset [11] [6]. Certain parameters within a hydrological watershed model can be learned by studying maps of the area, but usually complex hydrological models will have a large amount of parameters that due to issues such as spatial variability o measurement error may not be exactly known[15]. In this case a model is calibrated to determine their value.

The problem when calibrating a numerical model is that the evaluation of each calibrated parameter set can be very time consuming or expensive. The evaluation of the parameter set is computationally expensive because the model must first be run using the parameter set before the output of the model can be evaluated. And since getting an good model output often requires a lot of evaluation runs, the cost in time or computational power can be quite high. This presents a dilemma for someone designing a numerical model. The more accurate the model, the more time consuming or expensive the model will be, but reducing time consumption and resource use may make the model less accurate. The modeller wants the mode to be as accurate as possible, but doesnt́ want the model to be too time consuming or expensive (two factors that are usually linked). [10]

The general solution to this dilemma has usually been to use an algorithm that can with an acceptable speed find an accurate result, that is, a set of parameters that when used on the model will give a model output that closely reproduces observed values. Automatic calibration of parameters is not a new concept, there are several known and documented approaches. In this thesis we will explore several approaches to automatic parameter calibration using learning systems and genetic algorithms to automatically calibrate a model chosen by the thesis supplier (Agder Energi), the OpenHBV model, and explore the weaknesses and strengths of these approaches. This knowledge is a contribution to the field of automatic calibration, and may reinforce knowledge of the efficiency of different real world applications of automatic parameter calibration techniques. [20] [10] [9] [18]

We intend to benchmark and compare four different approaches to automatic parameter calibration.

- Shuffled Complex Ecolution (SCE-UA)
- Monte Carlo Scheme (MCS)
- Continuous Action Learning Automata (CALA)
- Genetic Algorithms (GA)

Because the SCE-UA method is thought to be a state of the art automatic parameter calibration method, we will primarily compare the other methods to the SCE-UA method. [18] [17]

1.3 Thesis description

In order to optimize income from production of power, it is important for power manufacturers to possess accurate predictions of how much water is coming into their reservoirs at any given time. To provide such predictions, hydrological stream flow models are typically applied. Calibrating such numerical models is an arduous and important task due to the amounts of variables and involved. The purpose of this thesis is to compare several approaches to automatic numerical model parameter calibration, using a simplified water table model. Whereas practical experience has shown that no single error measure can adequately capture the ways in which a model fails to match characteristics of observed data, genetic algorithms have proved effective in numerous automatic model parameter calibration applications.Genetic algorithms have been used for a wide range of models. In this thesis, four automatic parameter calibration approaches will be compared; a Monte Carlo scheme, Learning Automata using a CALA implementation, Genetic Algorithms and SCE-UA. It is expected that the SCE-UA approach will produce better results than other approaches because it is considered the state of the art of watershed model calibration algorithms. [18]

1.4 Research questions

In this thesis we intend to answer the following research questions.

Which method is best at calibrating the model? In order to answer this question we split it in two parts. First, the calibration method must find an output that closely matches observed historical values. Second, the method must find a parameter set that produces this output efficiently. We will use each method to calibrate the parameter set and evaluate the modelled output for an arbitrarily chosen amount of times. Each method will be able to run the model and evaluate the model output a set number of times

How does each method calibrate the parameter set when the modelled output is compared to a pregenerated output from the same model, compared to when the modelled output is compared to historical measurements? We find this question interesting because the SCE-UA paper [18] uses a pre-generated model output to evaluate modelled output of the calibrated parameter set. The SCE-UA paper indicates that their model implementation was deterministic, and as such using the SCE-UA method to calibrate a method against observed historical values may not produce the same results as calibrating against an output the model can perfectly reproduce. Since the OHBV model is deterministic[15], given a pregenerated model output produced from a model run on a given set of parameter values, the automatic calibration methods should be able to reproduce the modelled output in a way that comparing modelled output to historical measurements may not. How does this influence the different automatic parameter calibration methods?

1.5 Contributions

The purpose of this project is to show the theory behind each approach and elucidate through testing whether there are any clear and distinct advantage to using one specific approach over another.

We assume that by using a solid reward and punishment feedback system for the reinforcement algorithms that is also usable as a criterion value for genetic/evolution algorithms our research can be applied to most models that need to be calibrated. The reward and punishment/criterion value system is based on the results from the simulation. The simulated results are compared to a measured result; we measure the accuracy using various techniques like mean root square error and correlation.

We then present the results and discuss their importance, and decide based on these results which of the algorithms is the best one to use for calibration in our case.

Finally, we propose areas for further research with regards to our work in the field of automatic model calibration.

1.6 Report outline

In chapter 2 we will give detailed descriptions of the different technologies used and some details of how they were implemented in our project. In chapter 3 we present and discuss the results we have obtained. And we end the thesis in chapter 4 with a conclusion and recommendations for further work.

2 Methods

In this chapter we discuss and explain what methods we used. In the OHBV model chapter we explain what the OHBV model is and how it works, including the parameters, as well as some implementation details. In evaluation we show the methods we used to create a feedback that we could use for our configuration methods. SCE-UA, Monte Carlo Scheme, CALA and Genetic Algorithms is the methods we used to autocalibrate the OHBV model. In Each chapter we discuss how the method works and how we configured the algorithms and their parameters.

2.1 OHBV Model

In our project, we calibrated the parameters of the OHBV model, or Open HBV model, which as supplied by Agder Energi and is based on the HBV model. In this chapter we describe the HBV model and the parameters it uses, and which parameters we calibrated. The information and figures in this chapter are largely gathered from Hydrology[15].

2.1.1 The HBV model

"The HBV model, developed by Dr. Sten Bergström at the Swedish Meteorological and Hydrological Institute, is a conceptual precipitation-runoff model which is used to simulate the runoff process in a catchment based on data for precipitation, air temperature and potential evapotranspiration (Evapotranspiration (ET) is a term used to describe the sum of evaporation and plant transpiration from the Earth's land surface to atmosphere). The model computes snow accumulation, snow melt, actual evapotranspiration, storage in soil moisture and groundwater and runoff from the catchment. "[15]

The HBV model is a mathematical, and to some extent a linear model(here meaning that one state of the model is closely dependent on an earlier state of the model) of the hydrological processes in a catchment. While some parts of it, like the soil moisture routine, is non-linear, most mathematical expressions in the model are linear. The HBV model is based on conceptual considerations of the physical structure and processes in the catchment, because of this the model must be calibrated for a catchment before it can be used to model the runoff of that catchment.
The HBV model is deterministic. Two equal sets of input will always yield the same output if run through the model from identical start conditions and with identical model parameters.

2.1.2 HBV model routines and parameters

The HBV model uses several parameters to describe the conceptual workings of a catchment. Several parameters describe the physicality of the catchment, its size, and the size of natural and regulated lakes within the catchment. The HBV model uses these parameters to split the catchment into several elevation zones. This is important, as elevation influences factors like air temperature and relative precipitation.

2.1.3 The snow routine

The snow routine computes snow melt or refreeze, precipitation and precipitation type within each elevation zone. according to the area-elevation curve. At each zone the model computes the air temperature for that zone according to its elevation relative to the elevation of the air temperature measurement station, the amount of precipitation in the zone based on observed precipitation values and precipitation type (rain or snow) based on the zones air temperature.
The main results of these computations give three main variables that are computed for each elevation zone and time step: snow storage, free or liquid water contents in snow and snow melt per time step. All these variables denote the water-equivalents in millimeters.[15]

If there is more liquid water in the snow than a certain threshold, that water is passed to the soil moisture routine.

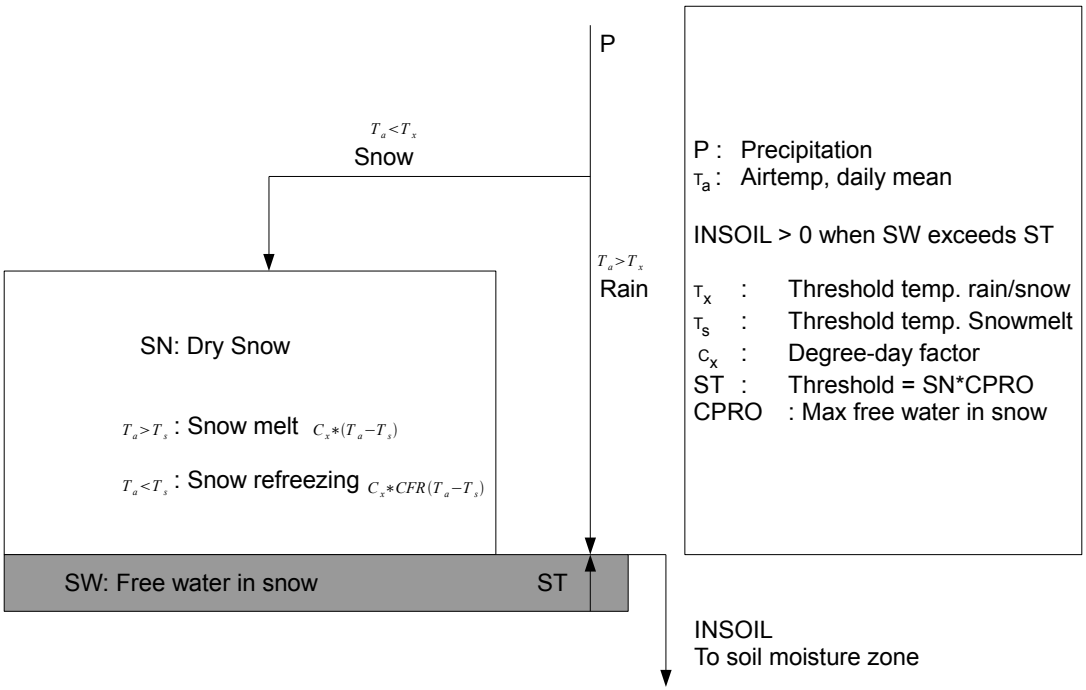


Figure 1: The snow routine in the HBV model

Figure 1 shows how incoming precipitation is categorized by the model as either snow or rain using a threshold temperature T_x. Another threshold temperature T_s is used to determine whether snow is melting or refreezing. Using this routine the model computes a conceptual water storage component. We can also see that rain is either stored in the snow, or passed to the soil moisture routine if the water in the snow, SW exceeds the threshold ST. Figure from Hydrology[15].

2.1.4 Snow cover distribution

Usually more precipitation falls in areas of high elevation, and areas of high elevation usually also has a lower air temperature. Because the catchment area is divided into elevation zones, the model can simulate different amounts of snow storage in different zones. However, precipitation is not the only form in which snow storage in each zone is impacted. Local variability and wind patterns may move existing snow around, causing some zones to have areas with large amounts of snow that may last beyond normal time spans, and some areas may remain free of snow all through the winter. Helicopter based radar measurements by Agder Energi Produksjon have shown that 3-8% of random areas may be free of snow in winter. In the OpenHBV model this is addressed by splitting the precipitation into smaller units within each elevation zone. The precipitation of each sub-snowunit is then scaled by a factor N_i. This factor is

dependent on the index of the sub-snowunit og the value of the snow-distribution parameter b.

b= $SD_{min} + (1 - FF) * (SD_max - SD - min)$
SD_{min} = Minimum snow distribution parameter
SD_{max} = Maximum snow distribution parameter
FF = Forest fraction in the elevation zone (Generated from vegetation maps)

Figure 2: How the snow-distribution factor b is found.

2.1.5 The soil moisture routine

> "The soil moisture routine receives rainfall or snow melt as input from the snow routine and computes the storage of water in soil moisture, actual evapo-transpiration and what may be called the net runoff generating precipitation as output to the runoff response routine."[15]

The soil moisture routine uses two simple equations with three empirical parameters; β, FC and LP, to model water content in the soil. β controls the contribution to the runoff response routine(dUZ), this is usually non-linear, but will be linear if β is equal to 1. FC is the field capacity, that is, how much water can be stored in the soil. If the soil moisture storage is filled to FC, input from the snow routine is transformed directly to runoff. The soil moisture storage is depleted by evapotranspiration. Evapotranspiration is determined by the evaporation equation in figure 3 and by LP, the Potential Evapotranspiration threshold. When the amount of water in the soil exceeds LP, the amount of Evapotranspiration will be equal Potential Evapotranspiration. Evapotranspiration in the model is only computed from the snow-free part of a catchment.

> "Both β, LP and FC are free parameters and must be determined by model calibration, they can not be determined directly from maps or field surveys."[15]

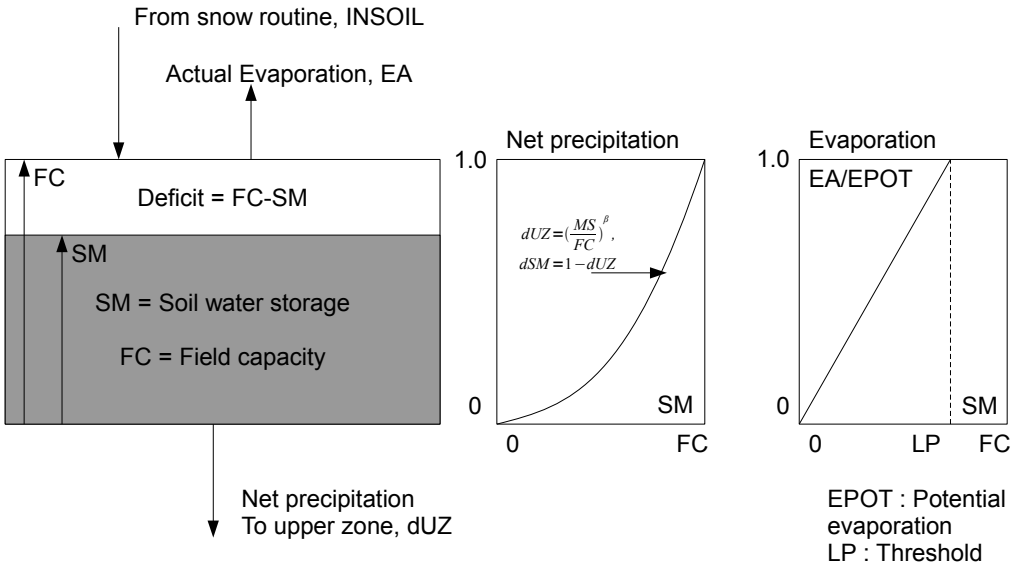


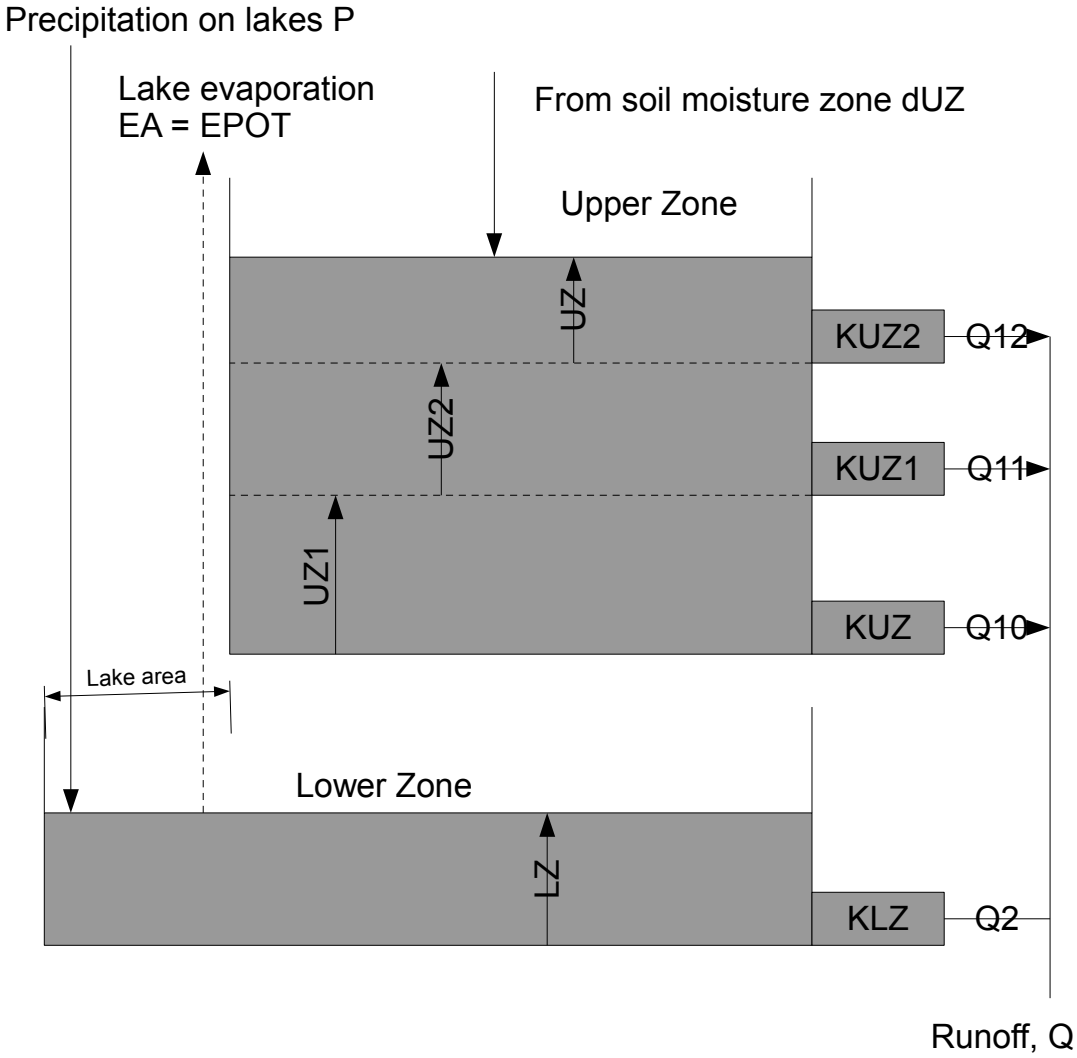
Figure 3: The soil routine in the HBV model. As can be seen from the figure, water is input into the system from the snow routine, and leaves the system by precipitation to upper zone, dUZ or by Actual Evaporation EA. Figure from Hydrology [15]

2.1.6 The Runoff Response Routine

"The runoff response routine transforms the net precipitation produced in the soil moisture routine into runoff. The runoff response function in the HBV model consist of two linear reservoirs, the Upper zone and the Lower zone. This routine also includes the effect of direct precipitation and evaporation from rivers and lakes in the catchment."[15]

"The upper zone conceptually represents the quick runoff components, both from overland flow and from groundwater drained through more superficial channels, interflow."[15]

"The lower zone conceptually represents the groundwater and lake storage that contributes to base flow in the catchment. The drainage speed is controlled by one recession parameter. The lower zone gets water input by percolation from upper zone (percolation is the movement of water through the pores in soil or permeable rock), and by direct precipitation on lakes and rivers. It is depleted through base flow runoff and also through evaporation from lakes an rivers. This evaporation always equals potential evaporation as long as there is water in the lower zone storage."[15]



Parameters in the response function:
KLZ : Time constant, LZ
KUZ : Time constant, UZ
KUZ1 : Time constant, UZ
KUZ2 : Time constant, UZ
UZ1 : Threshold for quick flow, mm
UZ2 : Thresholdfor v. quick flow
PERC: Percolation to lower zone, mm/day
UZ, LZ : Upper Zone, Lower Zone

Runoff Components

Q= Q10+Q11+Q12+Q2
Q10=MIN(UZ,UZ1)*KUZ
Q11=MAX(0,((MIN(UZ,UZ2)-UZ1)*KUZ1))
Q12=MAX(0,(UZ-UZ2))*KUZ2
Q2 = KLZ*LZ

Figure 4: The runoff routine in the HBV model. As can be seen from the figure, precipitation and evaporation that takes place on lakes is excluded from the snow and soil routines, and instead impacts the Lower Zone in the runoff response routine.

Water is input from the soil moisture routine, dUZ. The total runoff Q is computed from several other runoffs from the upper and lower Zones. The runoffs are computed from the slower flow from the Lower Zone, and the quick and very quick flow of the Upper Zone. The different flows in the upper zones are computed using thresholds UZ1 and UZ2. The input dUZ is continually passed to the Lower Zone through percolation PERC. The upper zone is filled when dUZ exceeds a percolation capacity, and is simultaneously drained by the lower outlet KUZ.

2.1.7 Model Input

The different inputs of the model are used for specific tasks. Air temperature is used to determine the type of precipitation received in the model, it determines snow melt and snow refreezing and in some model implementations potential evapotranspiration. Precipitation determines the total amount of water input into the model. Observed precipitation is processed in the following steps before the areal precipitation values can be entered into the model:

- "Observed precipitation is corrected for gage catch errors to obtain true precipitation at each precipitation station."
- "Data for several precipitation stations may have to e combined to obtain average or areal precipitation for the catchment."
- "The amount of precipitation within each elevation level is determined based on precipitation lapse rate and elevation."

[15]

Wind, air humidity and air temperature and radiation balance is used to determine Potential evapotranspiration. It is usually computed from standard meteorological data, however in order to increase the complexity of the parametersets in our project we let the calibration methods calibrate these values.

2.1.8 Model calibration

The model produces several different outputs. The catchment runoff is what we used to calibrate the model. In order to correctly calibrate the model to a given catchment, it is important to have high quality runoff observations for that catchment. We used historical observations an area and values from a pre-generated parameter set. In model calibration there are two types of parameters, free and confined parameters. Confined parameters are parameters that are physically dependent on the catchment area. Once these parameters are found, they no longer need to be changed, and thus can be ignored by an automatic model calibration scheme. Free parameters are determined through model calibration.

We here list the confined OHBV parameters that our methods did not calibrate on.

lake_fraction The lake fraction parameter is a physical description of the Skjerka catchment. Because of this we decided not to calibrate this parameter. See figure 4

zone There are 10 zones described in the parameters. These are also determined by the physicality of the Skjerka catchment and was therefore used for calibration. See figure 1

We here list the free OHBV parameters that our methods used for calibration.

rain_corr, snow_corr A correction factor applied to precipitation data to account for measurement error and adjustments if the measurement station is not representative for the zone. rain_corr is applied to precipitation in the form of rain, while snow_corr is applied to precipitation in the form of snow.

max_liquid_in_snow Max liquid in snow, or SWSee figure 1.

threshold_rain_snow Threshold temp between rain or snow T_xSee figure 1.

degree_day_factor Describes how much snow melts or refreezes per day. Described as C_x in the HBV model snow routine. See figure 1

threshold_melt The threshold for when snow melts into water. See figure 1

threshold_freeze The threshold for when water in SN freezes into snow. The melt and freeze threshold are not shown as separate in the HBV model snow routine figure, but in the OHBV model they are separate values because they are not necessarily equal. See figure 1

refreeze_efficiency This is the efficiency at which water freezes into snow. This value is again separate from the one value shown in the HBV model snow routine figure See figure 1 because it is not necessarily equal to the melt-per-day factor.

precip_grad Describes how precipitation varies in the different elevation zones, ie; how precipitation changes according to the zones elevation above sea level compared to the measurement station. Denoted in percent per meter.

temp_grad_clear, temp_grad_precip Describes how air temperature changes according to the zone elevation, like precip_grad. Clear refers to no precipitation in the air, while precip refers to with precipiation in the air.

field_capacity Describes how much water can be held by the soil. Denoted as FC. See figure 3.

lp This is the potential evapotranspiration threshold. Once SM exceeds LP. See figure 3, EA equals EPOT.

pot_evap This is the potential evaporation. This parameter consists of 12 internal parameters, each describing the potential evaporation of each month in a year. Usually, this would be considered a constricted parameter, but in order to increase the complexity of the parameter calibration scheme, we used these parameters for calibration as well. See figure 3

beta This is used to determine the net precipitation passed from the soil routine to the runoff response routine. See figures 3 and 4

max_infil_soil This describes how much water the soil can take up in one day. If the amount of water from snowmelt or rain that comes out of the snow routine is greater than this value, that water is passed through the soil routine automatically.

kuz2,kuz1,klz These parameters describe the flow of runoff from the upper and lower zones over time. See figure 3

uz2, uz1 Thresholds describing when runoff should flow from kuz2 and kuz1. See figure 3

snow_dist_min, snow_dist_max Parameters used to describe the distribution of snow within elevation zones. See figure 2

percolation This parameter describes the flow of water from the upper zone to the lower zone. See figure 3

annual_et This parameter describes the annual rainfall in the catchment over a year. This can usually be a constricted variable, but was calibrated on for increased complexity.

2.2 Evaluation Methods

We want the model to produce an output that as closely as possible fits the observed values. In order to automatically calibrate the parametersets to do this we need a criterion value that describes how closely the modelled output fits the observed values.
The criterion we used is a combination of several different methods for data series comparison that is sendt to the methods as a feedback. From here we refer to the criterion value as the feedback for the rest of the report, as we feel that feedback more generally describes both the reinforcement feedback of the CALA method, and the criterion value of the GA, SCE-UA and Monte Carlo methods. To generate this feedback, we compared the output of the model, when run with parameter sets generated by our algorithms, with outputs from a pre-generated model and observed historical values.

2.2.1 Correlation

$$ymid = \frac{\sum_{i=1}^{n} observed_i}{n}, xmid = \frac{\sum_{i=1}^{n} modelled_i}{n},$$
$$sumxy = \frac{\sum_{i=1}^{n} n(modelled_i - xmid) * (observed_i - ymid)}{n},$$
$$sumx = \sqrt{\frac{\sum_{i=1}^{n} modelled_i - xmid^2}{n}}, sumy = \sqrt{\frac{\sum_{i=1}^{n} observed_i - ymid^2}{n}},$$
$$correlation = \frac{sumxy}{sumx * sumy},$$
$$returncorel = \frac{correlation+1}{2}$$

The purpose of the correlation method is to find how closely the curve generated by the model compares with the observed or pre-generated values. This is an important factor since it shows how well the model parameters describes the effect of temperature, freeze and melt threshold levels in the model, as well as other less obvious effect like ground saturation, ie; when the observed values go up, the modelled values should go up at the same time and vice versa. Because the correlation evaluation method is insensitive to changes in terms of magnitudes, a second evaluation method was needed to deal with this. (Explained in Mean Square Error method.) The correlation is found using a standard statistical procedure for finding the correlation between two dataseries. The correlation will be in the span between -1 and 1. Because it is more useful to us in the span between 0 and 1, we add 1 to the correlation and divide the sum by 2, and return the new value.

2.2.2 Mean Square Error

$mse = \sqrt{\sum_{i=1}^{n} \frac{(modelled_i - observed_i)^2}{n}}$,
$returnmse = e^{-\frac{mse}{200}}$

The Mean Square Error (MSE) method finds the general amount of error in the modelled values. The purpose of the MSE method is to find whether the model generates outflow values correctly in terms of their respective magnitudes, which the correlation evaluation method is insensitive to. For example, if the model shows a run of values $[1, 2, 4, 8, 4, 2, 1]$ and the measured values are $[3, 6, 12, 24, 12, 6, 3]$, the correlation evaluation will give a perfect result because these dataseries correlate perfectly. But for the model, the amount of outflow generated by the model in this case is a significant underestimation. The MSE method was introduced to deal with this specific problem, but because the MSE method is sensitive to noise, we used a modified version of it in our output evaluations.

2.2.3 Mean Square Root Error

$mrse = (\frac{\sum_{i=1}^{n} \sqrt{|modelled_i - observed_i|}}{n})^2$,
$returnmrse = e^{-\frac{mse}{200}}$

The Mean Square Root Error (MSRE) method works like the MSE method, but better accounts for noise problems like measurement error or freak weather that the model cannot account for. Using this scheme instead of the standard MSE, pronounced differences in modelled output and observed values (noise) have a less pronounced effect on the output, while all errors will still impact the evaluation.

2.2.4 Bias

$bias = \frac{\sum_{i=1}^{n} modelled_i - observed_i}{n}$,
$returnbias = e^{-\frac{|bias|}{50}}$

This method estimates how much the model over- or undershoots the observed values in general. Testing showed that this method caused problems for the CALA algorithm. We therefore stopped using this method to evaluate model outputs.

2.2.5 Feedback

In order to create a useful feedback, we decided on a key requirement being that all feedback values must be usable on all algorithms. A further requirement was that the feedback had to be interchangable and combinable. The MSRE method returns the mean square root error, which can be anywhere from $0 \rightarrow \infty$, albeit unlikely. The correlation method returns a value ranging between -1 and 1, 1 being perfect correlation, 0 being no correlation and -1 being inverse correlation. (Modelled output does the opposite of measured values.) It would be difficult to combine these into a single feedback value in this form. By normalizing these feedback values, they can be multiplied together into a single feedback. This allows both feedbacks to have equal value in relation to their relative ranges.

2.3 SCE-UA

There are two main components to the SCE-UA algorithm. The outer algorithm that sorts and handles the parametersets and the inner CCE (Competitive Complex Evolution) algorithm based on the Nelder and Mead Simplex downhill search scheme[14][18]. The SCE-UA scheme works by sampling a large number of points in the parameterspace. Each time the algorithm tries to move the worst point closer to a presumed optimal point, it is either successful, in which case the algorithm is one step closer to the optimal point, or it replaces the worst point with a new point randomly generated within the parameter space, thus providing further sampling. Because it is the worst point that is replaced each time, we eventually end up with a collection of points gathered in the optimal zones of the parameter space, once these points have been gathered, the algorithm slowly inches in on the optimal point, while moving points from areas that are a local optima to the area that is the global optima. Our implemenation of the SCE-UA is based on [18]. This reference is from now on called ”the SCE-UA paper”.

2.3.1 Algorithm

Here follows a description of our implementation of the SCE-UA algorithm.

STEP 1:

Generate a population of parameter sets according to a uniform probability distribution. Find the criterion value of each parameter set. The size of the population is dependent on the number of complexes used and the size of each complex.

STEP 2:

Rank and sort the parameter sets according to their criterion values, best sets to worst sets.

STEP 3:

Partition the parametersets into complexes of size m, where m = 2*h +1 and h is the number of parameters in each parameterset. Partition the s parametersets into p complexes such that the first complex contains every p(k-1)+p ranked point, where k = 1,2,...,m and p = 1,2,... etc. Using this scheme, if you had 3 complexes and 9 points, the first complex would contain points [1,4,7] the second would contain [2,5,8] and the third would contain [3,6,9].

STEP 4:

Evolve each complex using the CCE-Algorithm. (Explained below)

STEP 5:

Combine the parametersets in the complexes into a single population. Sort the population by criterion value, best to worst. Repartition the population into p complexes according to the procedure in STEP 3.

STEP 6:

This step usually checks for convergence, however, we are comparing different methods using 10 000 model iterations as a baseline for comparison. The convergence step therefore checks whether the model has been run, in total, 10 000 times or more and if it has the algorithm is stopped.

STEP 7:

In some testing, using a large number of complexes appeared to give better results, and to improve efficency within 10 000 runs, the number of complexes was be reduced once certain thresholds of criterion values were reached. In other tests, we used a small or medium number of complexes and the reduction part was not used. The reduction step removes the complexes with the lowest ranked points. Our tests showed that using this reduction scheme did not significantly improve the efficiency of the SCE-UA algorithm compared to using fewer complexes from the start. (See figure 45, 44 in the appendix)

2.3.2 The CCE algorithm

The CCE algorithm runs once for each complex.

STEP I:

Construct a subcomplex by selecting q points from the complex according to a trapezoidal probability distribution, where q = h+1 and h is the number of parameters in each parameter set. The trapezoidal probability distribution is similar to a roulette wheel selection scheme in that each parameterset is given a probability of being selected based on its criterion value compared to the criterion values of all other parametersets in the complex.
Sort the subcomplex according to its criterion values, best to worst.

STEP II:

Identify the worst point P_l in the subcomplex of size n+1. Find the centroid $\overline{P}$ of the subcomplex while ignoring the worst point.

$\overline{P} = \frac{\sum Points_i}{n}$, where $P_l \notin Points$

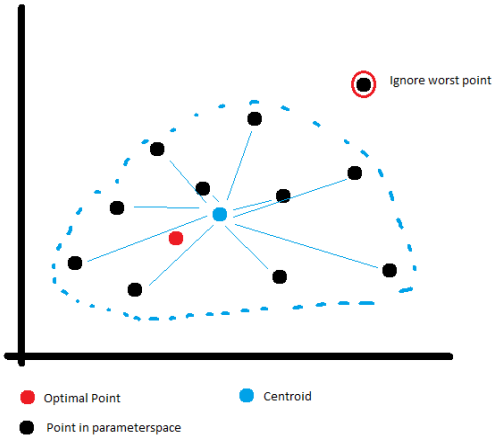


Figure 5: Ignore the worst point in the subcomplex, and find the $\overline{P}$ of all other points.

STEP III:

Attempt a reflection step by mirroring P_l through $\overline{P}$. If the mirrored point P^* is within the feasible parameter space, go to STEP IV, otherwise, go to step VI.

$P^* = (1+\alpha)\overline{P} - \alpha P_l$

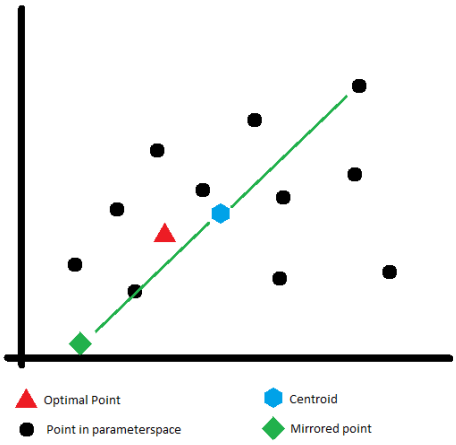


Figure 6: Mirror P_l through $\overline{P}$. α = 1

STEP IV:

If P^* has a better criterion value then P_l, replace P_l with P^* and return to step I. Otherwise, go to step V.

STEP V:

Attempt a contraction step by computing a point P^{**} halfway between $\overline{P}$ and P_l. If P^{**} is better than P_l, replace P_l with P^{**} and go to STEP VII. Otherwise, go to STEP VI.

$$P^{**} = \beta P_l + (1 - \beta)\overline{P}$$

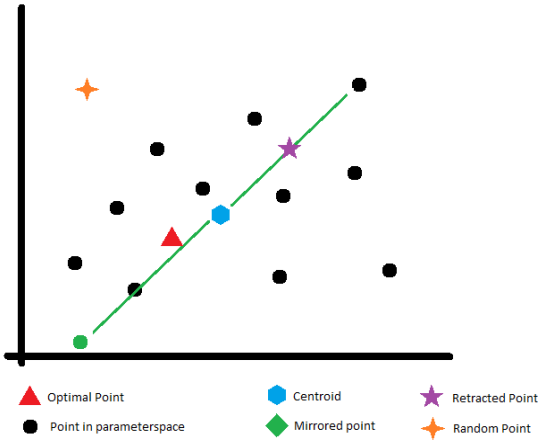


Figure 7: Find the retracted point P^{**} halfway between P_l and $\overline{P}$. β = 0.5

STEP VI:

Replace P_l with a point randomly generated within the feasible parameter space.

STEP VII:

Repeat steps I to VI $2 * m + 1$ times, where m is the size of each complex.

2.3.3 Algorithm Parameters

There are a few key parameters in the algorithm. There's α, the reflection constant. In the SCE-UA paper it was set as 1.0, however, our testing seemed to show that the SCE-UA algorithm was far more efficient within 10 000 iterations of the OHBV model with an α of 0.5(Figure 9). While setting α to 0.5 may increase the chance of finding local optima, rather than global optima, we consider the increase in efficiency to be more important, because our test will attempt to fit the model output to observed values, rather than trying to find a set of parameter values, which is what was done in the SCE-UA paper. This means that the step shown in figure 6 will now look like this:

$$P^* = (1+\alpha)\overline{P} - \alpha P_l$$

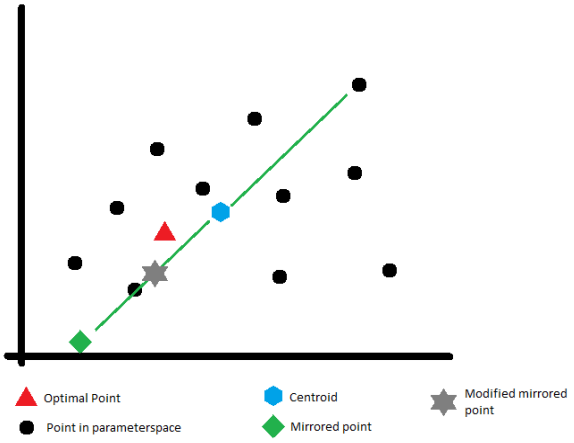


Figure 8: Mirror P_l through $\overline{P}$. α = 0.5

The difference in efficiency can be seen in figure 9.

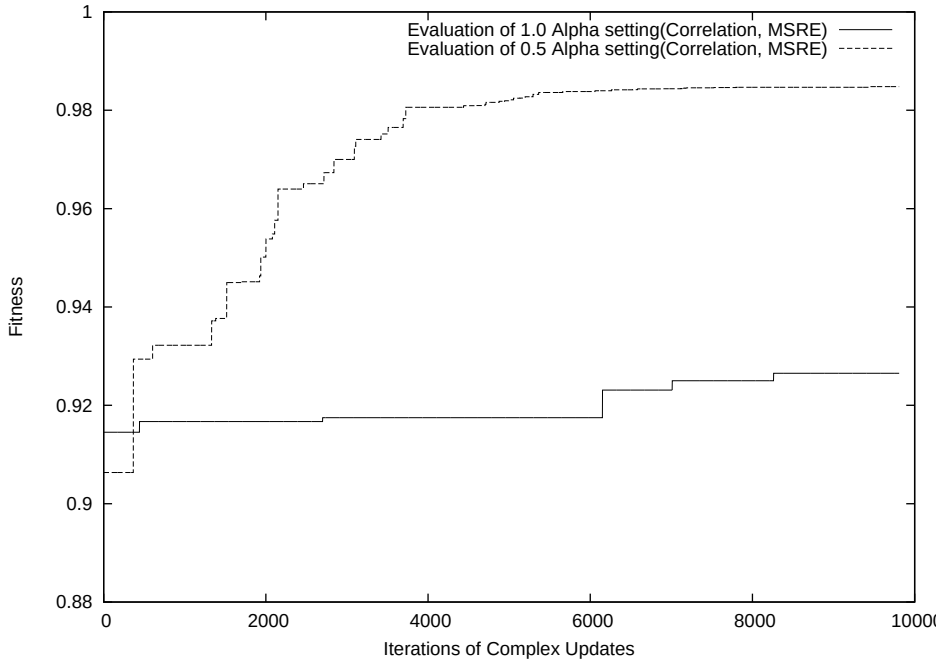


Figure 9: Our testing consistenly showed a better efficiency when using an α of 0.5. Here is a comparison between two runs of 4 complexes, one with an α of 1.0, and one with an α of 0.5. The graph shows average values of 30 runs.

After running the algorithm with the two different αs several times, and seeing little difference from figure 9, we concluded that using an α of 0.5 was the best choice in our case.

β is the contraction constant, we kept this at 0.5, as per the SCE-UA paper. The number of offspring produced by each subcomplex was kept at 1, as the SCE-UA paper specifically stated that this was the optimal solution. We tested several combinations of two parameters: The number of times a subcomplex is generated from a complex per run of the CCE-Algorithm and the number of complexes used by the SCE-UA algorithm. The SCE-UA paper stated that the suggested that the default number of subcomplexes generated per complex each CCE-pass should be 2n+1 where n is the number of parameters to be optimized on. We tested these with an α of 0.5 in these combinations;

Number of complexes	Number of subcomplexes generated
4	73
8	73
10	10
10(with reduction)	10
10	73
10(with reduction)	73

We ran these tests because we are using the SCE-UA algorithm to optimize a parameterset of 36 parameters, whereas the SCE-UA paper used the algorithm

to optimize a maximum of 13 parameters. However, in the tests seemed to show that using 4 complexes and generating 73 subcomplexes gave the best results, whereas using 10 complexes with 73 subcomplexes produced barely any improvement at all. Because the number of OHBV iterations run each time the 10 complexes with 73 subcomplex generations are run this method proved to be too inefficient (37 parameter sets run 73 times for 10 complexes for a total of 27 000 OHBV iterations per iteration of the SCE-UA algorithm, which means 4 total SCE-UA iterations to calibrate the parameter set.) to warrant further testing. See figures 44, 45, 43 in the appendix to see a comparison of the 8-73, 10-10 and 10-10 with reduction implementations.
From these tests we decided to use the SCE-UA algorithm with these algorithm parameters:

Number of complexes:	4
Size of complex:	73
Subcomplexes generated:	73
α:	0.5
β:	0.5
Reduction:	No reduction in number of complexes.

2.4 Monte Carlo Scheme

In the Monte Carlo method, a point is generated within the feasible parameter space, where each parameter is randomly generated according to a uniform probability distribution. These parameters are used to generate an output by the model, which is compared to pre-generated or observed values. If the parameter set produced a better model output previous best, it is kept. If it is not better, it is discarded.

2.5 CALA

Continuous Action Learning Automata (CALA) are used when one is searching for an optimum value in a real-world parameter set, such as gain in a control system, or a weight in a neural network. An LA is able optimize a value to produce the best possible feedback in a random environment through reinforcement learning [19].

2.5.1 Theory

For a single CALA the basic idea is that it constantly modifies a variable by either subtracting or adding a small amount to it, moving it towards a value in that returns the highest. It determines if it has to add or subtract from the variable by generating a random number and see how close to the optimum value the random number is. Cala does this by comparing the feedback for the variable and for the random number, the closer a number is to the optimum value the higher the feedback is. From the comparison CALA will know if it has to add or subtract [19].

These pictures show what happens when CALA is used on one parameter:

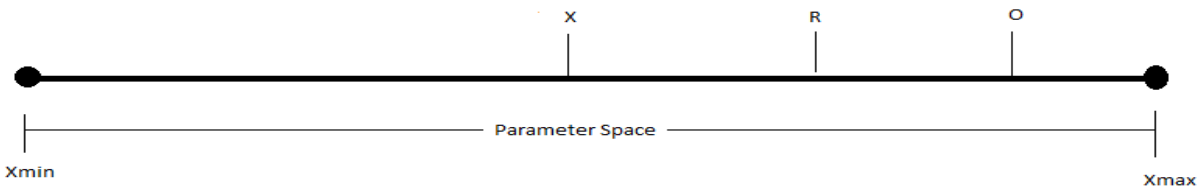


Figure 10: X is the current parameter value, R is the random value and O is the optimal value

As the figure shows, X is the number that we want to change to the optimum value. O is the optimum value and R is the random number. In the picture R is closer to O than X, therefore R will get a better feedback then X. xmin and xmax is the lower and upper bound of the parameter.

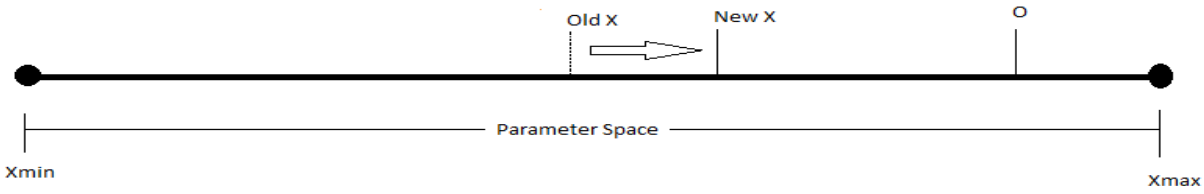


Figure 11: X moves closer to O

As a result the CALA algorithm will know in what direction O is and will update X moving it closer to O. If R had been on the other side of X, the CALA algorithm would know that it would have to move away from R to get closer to O. This is a general description that could be applied to several different Learning Automatas. CALAs are designed to work as a team sharing a single feedback to find the optimal value for a set of variables. In our case we will assign a CALA to each parameter we intend to calibrate on in the model.

When using teams of CALA you can expect to have an extended "learning period". The larger the team, the longer the learning period. The learning period occurs at the beginning of a run and is the time it takes for the algorithm to stop acting randomly and start giving consistent feedback. This can be seen very clearly in figure 48

2.5.2 Algorithm

$$\mu(k+1) = \mu(k) + \lambda \frac{(\beta x(k) - \beta\mu(k))}{(\varphi(\sigma(k)))} \frac{(x(k) - \mu(k))}{(\varphi(\sigma(k)))}$$

$$\sigma(k+1) = \sigma(k) + \lambda \frac{(\beta x(k) - \beta\mu(k))}{(\varphi(\sigma(k)))} \left[\left(\frac{(x(k) - \mu(k))}{(\varphi(\sigma(k)))} \right)^2 - 1 \right] - \lambda K(\sigma(k) - \sigma L)$$

where,

$$\begin{aligned} \varphi(\sigma) \quad &= \sigma L \text{ for } \sigma \leq \sigma L \\ &= \sigma \text{ for } \sigma > \sigma L > 0 \end{aligned}$$

[19]
The CALA algorithm has a number μ(k) and at each iteration k it generates a random number x(k). Based on the feedback for the random number βx(k) it updates its number μ(k) either to or away from x(k).

X(k) is generated based on the current action probability distribution, which is a normal distribution with mean μ(k) and standard deviation σ(k). The algorithm updates μ(k) and σ(k) at each iteration based on the feedback.

The Objective of CALA is to learn the value of x where the feedback is highest, and move μ(k) towards that value. It does this by having σ get smaller as the feedback gets bigger. When μ(k) is close to the optimum value, sigma will be very small, the normal distribution N(μ(k), σ(k)) will generate a number close to μ(k) because σ(k) is so low[19].

This means that the CALA will not be completely accurate, but will "hover" around the optimum value. This is because the CALA will constantly try to get a better feedback with x(k), but since we don't know the optimum value of x(k), we try to slow down the spread of numbers x(k) can be when it is close to the optimum value [19].

The accuracy of the algorithm is based on two things, λ and σL. λ is the learning speed of the algorithm, the larger λ is the faster the algorithm is, and likewise, the smaller λ is the more accurate it is. The Size of λ determines the size of the steps μ(k) takes to or away from x(k). the size of λ is 0 < λ > 1. σL is the lowest value sigma can be, this is to stop sigma from being 0, and also to indicate that the algorithm is very close to the optimum value. β denominates a feedback and all feedbacks are between 0 and 1 [19].

2.5.3 Pseudo Code

The following is a pseudo code of the CALA Algorithm.

Create CALA with value X

Get feedback for value X

Loop:

    Generate random value R

    Get feedback for value R

    Compare feedback for X and R and decide if to increment or decrement based on comparison

    Generate new X based on comparison, Lambda and Sigma

    Get feedback for value X

2.5.4 Testing

The CALA algorithm was tested on the Shubert function [7] and on the model using a limited number of parameters.

The shubert function [7] is a mathematical formula that has a number of local and global maximas and minimas. Shubert function is a good tool for testing optimization algorithms because an optimization algorithm can be set to look for these maximas and minimas. Figur 52 shows a graph of the shubert function. In the book [19]; that our implementation of CALA was based on, they search for the minimas in the shubert function to verify their implementation, and we did do the same with ours.

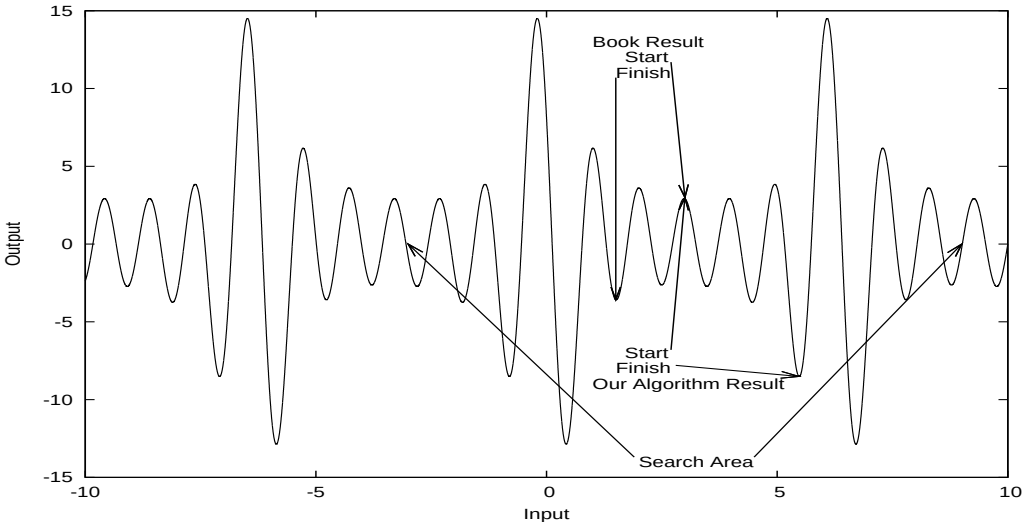


Figure 12: Comparing the Shubert function results form the book and from our own testing. μ = 3 σ = 6 on both algorithms

Limiting the Shubert function input to the space of -10 to 10 we gave the Cala algorithm a start point μ and a search spread of σ. The algorithm would then try and find a minimum value. When compared to the tests done in the book[19] we see that we usually get a different result, in fact on several runs we can get different results with the same settings on the same algorithm. This is because the search area given by σ can encompass several minimas and the CALA may end up in several of these.

As we can see from figure 12 which compares the result from the book with our implementation. The figure shows that even if the algorithms start with the same value (μ = 3) they end up with a different result. This is because of the search area determined by σ. And as we can see from figur 12 that the search area encompasses several minimums. I should be noted that the search area given in figur 12 is not a litteral representation of the actual search area, but only serves to visualize that a search area encompasses several optimum values for the algorithm. If the random number go beyond the borders we have set for the shubert function, we set the random value to be the same as the border it crossed.

We also tested CALA against a pre-generated test set. the test set was a simulated result from the OHBV model based on parameters we had written. We then let the CALA algorithm try and configure four of the thirty-six parameters and see if it could optimize the parameters. The parameters that were not tested were set to the optimum value The parameters that we tested on were: percolation - rain_corr - snow_corr - field_capacity. We chose these because they represented different aspects of the model and had different sizes to their parameter spread. By this we mean the size of the minimum and maximum values differed between each parameter.

We tested several combinations of Sigma and Lambda values, and the difference between having the parameters start at one end versus having them starting at random points

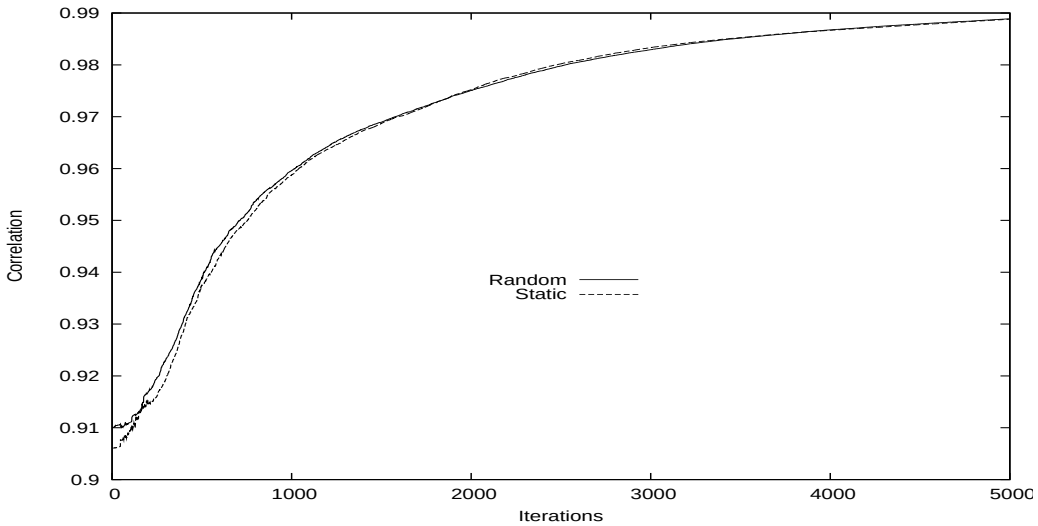


Figure 13: Comparison of Random and static parameter start

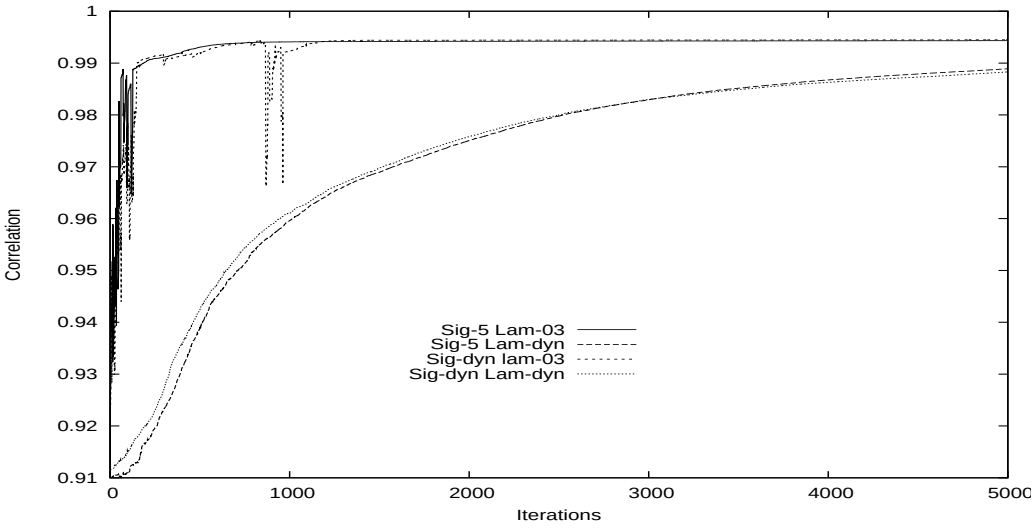


Figure 14: Comparison of the different configurations. all lines are the average value of 5 tests

As the results in figure 13 show, there is little difference between the parameter starting at one end and it being randomly generated. The first 200-250 runs are what we call the learning runs. Because the algorithm has not had any runs to learn from, the first runs will be more or less random so it can learn what the response from the enviroment is to the different actions. The algorithm seems to learn what to do and will consistently on all configurations reach a high feedback. It also shows that there seems to be a difference between a dynamic λ and a static λ.

2.5.5 Configuration

CALA relies on the two variables σ and λ when it comes to determine speed and accuracy. We did several tests to see what gave the best result. What we was mainly wondering about when it came to the configuration of the algorithm was the difference between a static variable that was the same on all the parameters, or a dynamic variable that was a certain percent of the parameter and was individual to each parameter. The static value for the σ was 5 and λ was 03. We also tested several different combinations of dynamic and static variables for σ and λ.

Dynamic value for λ and σ means that their value is dependant on the size of the spread the parameter has. If a parameter goes from -500 to 500 it's spread is 1000 and λ and σ will be set to a fraction of that. The size of the fraction is different for both λ and σ and is decided by us. Static means that the value for λ and σ is the same for all parameters no matter how large the spread is.

We also tested the difference in starting values. In one test the parameters started at the low end of the spectrum they are searching, and had to "search up". In the other the parameters start at complete random places and have to learn which way to search. We decided to use the four parameters that were used to measure the performance of the algortihm in chapter 2 to configure it. This is because we knew that they would after a known number of iterations find a high optimum, and we where only interested in finding the difference in speed and accuracy.

Each configuration was run for 5000 iterations 5 times on both random parameter start and static parameter start. The results shown are the average value for each configuration. We only used the correlation value as feedback for the configuration tests because these tests were meant to show the different characteristics of the different configurations, and therefore it was not important what we used for feedback as long as we used it consistently on the other tests.
The parameters used in the following figures are: Percolation, Rain-corr, Snow-corr and Field-capacity. And the plot's are an average of six runs.

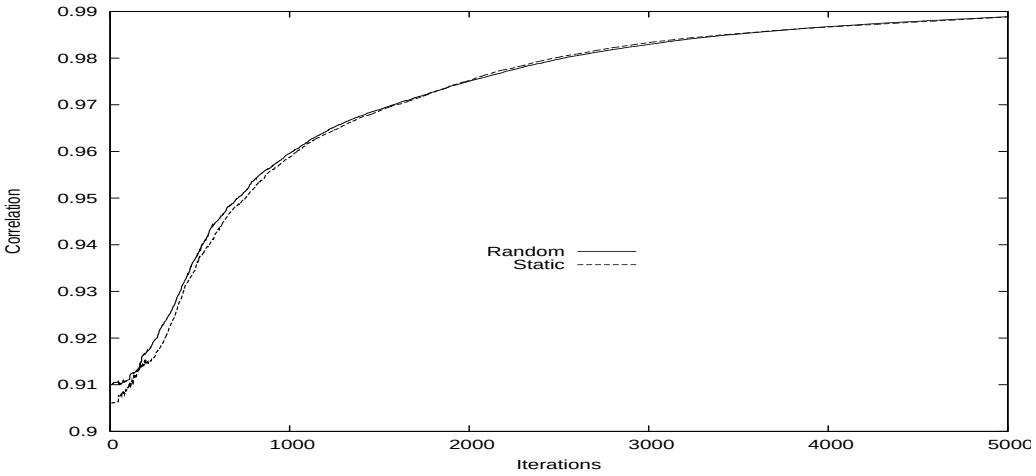


Figure 15: Shows the difference between the parameters starting at one end (the lower end) and parameters starting at random places. The configuration is σ-5 λ-Dynamic

Figur 15 shows the difference between a static and a random parameter start. We can read from the plot that there is very little difference based on where the parameters start, the only obvious difference is in the correlation value at the start. There the random value will fluctuate a little on the different runs while the static will always start with the same correlation value. The overall shapes of the plot are also quite similar. One can see that the random plot rises a little faster than the static, but seem to have the same top value. We believe that the differences are natural occurring variations that will happen because of the nature of the algorithm.

Figures 48, 49, 50 and 51 shows that the difference between a random-parameter start and a static-parameter start is small and almost neglible. They show no conlcusive evidence for one beeing better then the other. Based on this we decied to go with a static-parameter start. the reason for this is that the optimum parameters in an observed historical pattern is not known, and to reduce the search time as much as possible we want to set the start-parameters as close as possible to the optimum-parameters. by setting all the parameters at the center of the parameter space we effectively halv the area the algorithm has to search. Reasoning that the optimum-parameter can then never be more than half the parameter space away.

2.5.6 Configuration Test results

When comparing the four configurations we get this:

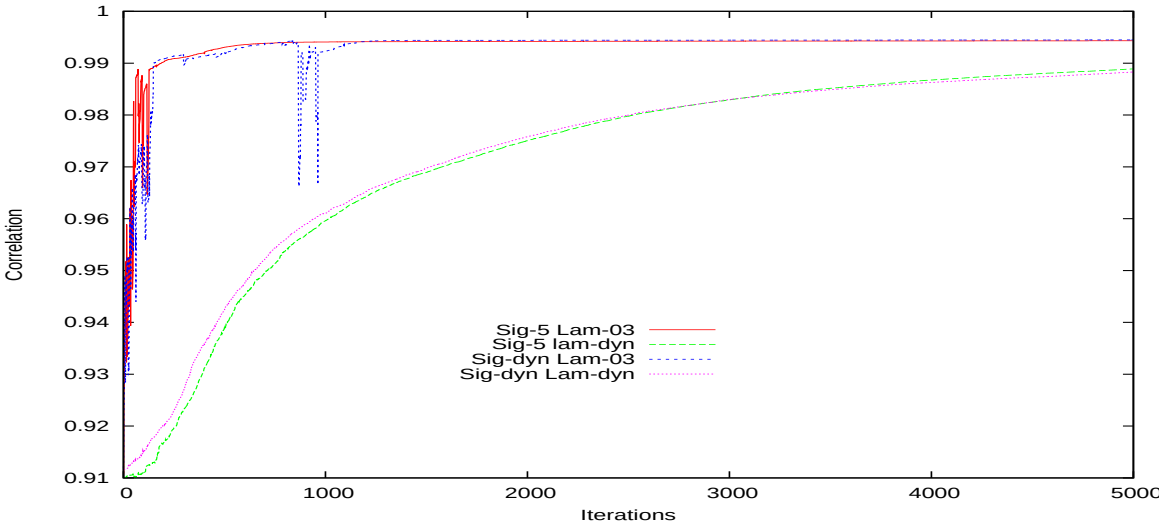


Figure 16: Comparing the four configuration plots

The overall shape seems to suggest that Lambda has the most impact on both speed and accuracy. It also shows that a static value on Lambda makes it not only faster, but more accurate than a dynamic value. These results somewhat confirms what we expected. We had expected that Lambda had a greater influence on the speed then Sigma, but we did not expect it to be so severe. However, the results show that we will be using a dynamic Sigma and a static Lambda for our comparison tests.

2.5.7 Error

We decided to use a dynamic σ and a static λ for the full parameter test. The setup was similar to the configuration setup, except we used all the parameters and the full evaluation routine. The setups was that CALA would run for 5000 iterations a total of 30 times. as we explained at the start of this chapter that we believe this is sufficient conclusive result.

When we analyzed the first runs we had done we saw that something was wrong. The plot in figure 46 clearly show that the CALA was not functioning properly. We had excpected some variations on the feedback when using all parameters, especially at the beginning as the algorithm would be learning. But after learning it should display a steadily rising graph.

We believed that CALA might have a problem with the amount of parameters involved so we tested them. We suspected that the parameters had different 'weight'. By 'weight' we mean that the impact the parameters have on the outcome is different. Some parameters affect on the outcome is almost negligible, whilst others can alter the entire outcome completely. To test this theory used the configuration setup. We started with the four parameters we had used for the CALA configurations and kept adding more and more parameters until the algorithm started showing signs that it couldn‘t handle anymore. If the algorithm suddenly showed signs that it it had reached its limits, we would scale back and try a different parameter and see if it still couldn‘t handle it.

The idea behind this approach was that if it was the weight of the parameters the algorithm couldn‘t handle, after getting a new parameter it would show a drastically different result than the previous run. If it wasn‘t the weight but the amount of parameters the result would gradually degrade with each new run with a new parameter. When running the tests we decided to add two and two parameters each time, and if there was a sudden change of the result we would scale back the two latest additions and test them separately to find the culprit. The result with six parameters (the four used in the configurations pluss two new) showed a result that didn‘t seem to learn nor would it converge towards an optimum. We had not not expected such result with so few parameters, especially since we got so good results with four. We decided to test the original four parameters from the configuration tests.The result echoed the previous test.

We wondered if the problem might not be with the parameters but with the evaluation method. We tested correlation, bias and Mean Root Square Error separately. We used the configurations setup and tested them on the same four parameters, the only difference was that the feedback was one of the three evaluation methods. Correlation gave us the same results we had gotten in the configuration tests, and Mean square root error also gave us results indicating it was learning and converge towards an optimum. Bias however seemed to be what caused the problem. The plot in figure 47 is the feedback we got when we only used bias for evaluation. The plot clearly shows that CALA could not handle using bias as a feedback. The graph shows no signs of convergence or learning. Bias was removed from the evaluation method.

2.6 Genetic Algorithms

Genetic Algorithms (GA) is based on 'Darwins Theory Of Evolution' [4]. The theory of evolution states that all species of life has descended from the same ancestor, and the individuals that exist in newer generations are better fitt than previous generations.

Evolution has a much known concept, "Survival of the fittest". Survival of the fittest means that in a group of animals; the fittest individuals will be more likely to reproduce, thus there is a higher probability that the genes of a fit individual will be passed to the next generation than the genes of a less fit individual. The fitness of an individual, and what constitutes fitness, depends on the environment the individual exists in [5].

GAs work by seeding a population of individuals, each with a finite set of chromosomes. For each generation (iteration) the algorithm will remove the individuals with less fitness, and combine the surviving individuals into a new generation and repeat the process. For Genetic Algorithms to work it must be possible to calculate a fitness for each individual [12].

2.6.1 Theory

A chromosome is one of the parameters that will be used to calculate the fitness. The fitness in our algorithm is the feedback from the evaluation methods. An individual is a collection of parameters, and the population is a collection of individuals. This means that the amount of chromosomes each individual has will correspond to the amount of parameters there are. In other words, each individual is one parameter set.

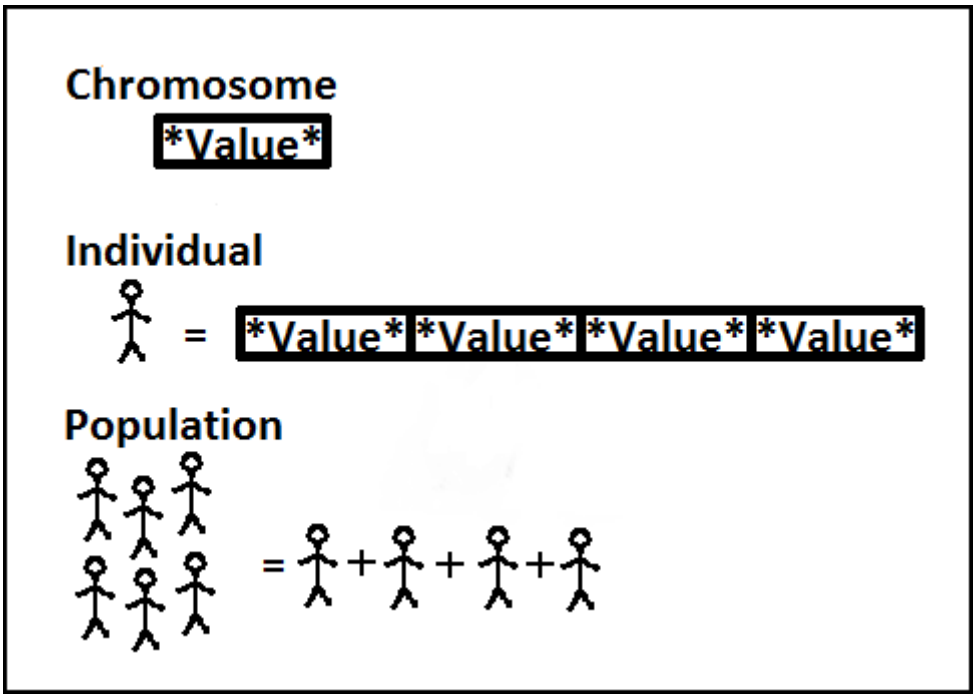


Figure 17: Showing the realationships between chromosome, individual and population

2.6.2 Implementation

GA starts by seeding an initial population; this is often done by random number generation.Then the population enters a loop, where each iteration represents a new generation of the population. The population is evaluated to determine fitness. In the loop it selects a few individuals from the population that will be allowed to reproduce while the rest of the population is discarded. Then the selected individuals are "reproduced". The reproduction method uses several techniques to combine or mutate individuals into new individuals. After the reproduction methods have built up a new population, the population gets evaluated and the individuals get their fitness and the cycle is continued [13].

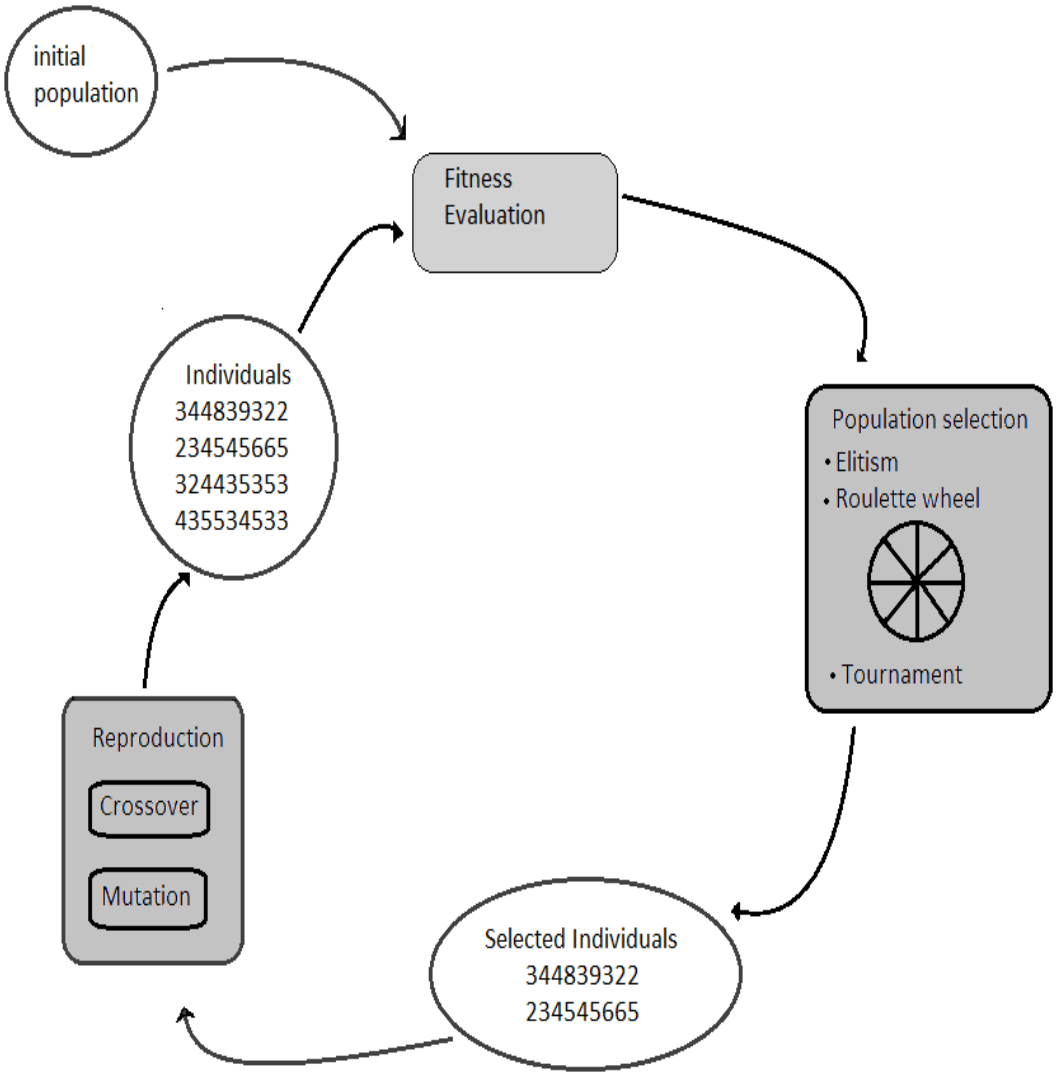


Figure 18: Illustrates the Genetic Algorithm cycle

2.6.3 Selection

The selection method selects what individuals are allowed to reproduce. To evade a premature convergence several different methods are used to select the new population. Premature convergence means that the algorithm has found a local optimum, and is unable to effectively explore further to find a better global optimum. When selecting what individuals to reproduce, there is a struggle between speed and accuracy. By focusing on speed, the speed the algorithm uses to find the optimum is fast (as indicated by the word speed), but the danger of getting a premature convergence is also increased. By focusing on accuracy, the chance for a premature convergence is reduced, but this means an increase in time used because of the larger sets that needs to be processed. Some algorithms (Elitism) focuses only on one of those aspects; ignoring the other. While others attempt to find an acceptable compromise between the two.

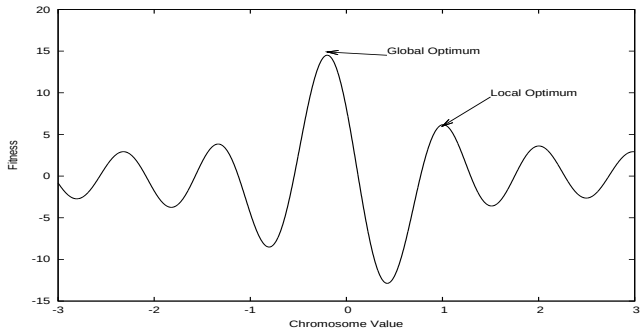


Figure 19: Plot showing Local and Global optimums

To avoid a premature convergence, other algorithms also selects individuals with a lower fitness. Because a lower fit individual may be closer to the global optima and when the higher fitness individuals max out on the local optima fitness, the 'lower' fitness individual may get a better fitness and direct the algorithm in on a better curve. Algorithms like roulette wheel selection uses a combination of random numbers and fitness when selecting which individuals that get to reproduce, giving the individuals with lower fitness a chance to be selected.

Elitism
Elitism selects the new population solely based on their fitness rating; this is a very effective method, but also one which has the most chance of getting a premature convergence. Elitism is generally used to secure that the best genes of the previous generation is preserved. Therefore elitism only selects a small number of individuals and usually works with other selection algorithms. The selection process looks at the fitness of all the individuals and selects the ones with the highest fitness.

Roulette wheel selection
In Roulette wheel selection, the chance of being selected is

$$\frac{fi}{\sum j = 1^n fj}$$

where n is the number of individuals in the population. This could be imagined similar to a roulette wheel in a casino. Each of the slots on the wheel is an individual, and the size of the slot is based on the individuals' fitness.

When you "spin" the wheel, the chance of each individual to be selected is based on the size of the area on the wheel they occupy. This means that the individuals with higher fitness has a larger chance of being selected, but is not absolutely certain that they will be selected [1].

With this method a number of mediocre fitness and a few lower fitness individuals will most likely be selected along with the higher fitness individuals. By selecting less than optimum fitness individuals the algorithm counteracts the chance of a premature convergence by having a population pool represent a larger spread of the search area.

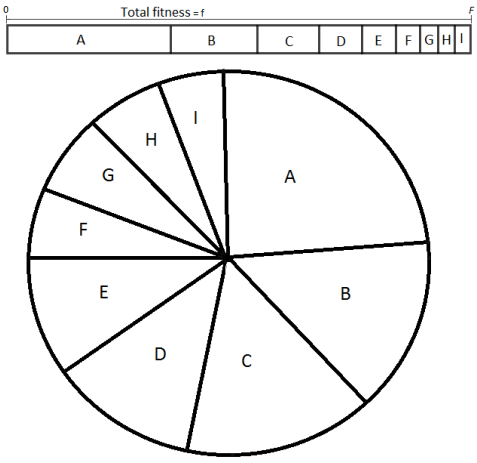


Figure 20: Showing the relationship between a roulette wheel and fitness

Tournament selection
Tournament selection stages tournaments between individuals and selects the winner of the tournament, in our case the winner is the individual with the best fitness. The pseudo code for our selection is:

Randomly select K individuals from population N
Compare and select individual with best fitness from K
Remove selected individual from N
repeat

As With the roulette wheel selection, this process does not ensure that the fittest individuals gets selected, but that a mixed group of fitness' with an emphasis on the better fitness' gets selected [16].

2.6.4 Reproduction

A reproduction process in GA means taking one or more individuals, and making a new individual. Some reproduction methods only mutate part of the chromosomes, while others combine two individuals. In many GA‘s the location of the parameters are unimportant, but since ours has to be static, many reproduction techniques like Order Xover and Position based Xover cannot be used.

Mutation
Mutation usually makes a small change in one individual, like changing one parameter. This is because mutation is meant to be a "Dark horse" of the population. Instead of relying on the chromosomes that already exists, it does a random mutation on a chromosome. This is because the population may have gotten stuck in a premature convergence, and the mutation may break out of that convergence and find the way to a better optimum, and if it doesn‘t find a better optimum or fitness, it will most likely not be selected for reproduction in the next selection cycle, causing no harm. Because of the random nature of a mutation, the chance of a mutation happening is very low [8].

Since our parameter locations are static (ie, the position of the chromosomes in the DNA are static), we only use insertions for mutation. Insertions takes and randomly changes on parameter, we also use a technique that changes several parameters, but this technique has an even smaller chance than the normal mutation of happening.

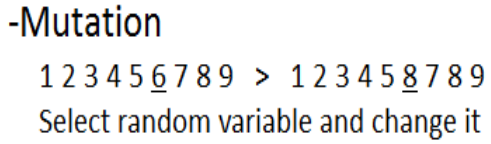


Figure 21: An illustration of what mutation does

Crossover
Crossover basically lets individuals switch parameters with each-other. The crossover we use is partly based on the PMX (Partially Mapped Xover) method. Our method divides two individuals into sections and exchange them making two new individuals consisting of parts wholly from the two selected individuals. We have three different variations of this technique, each varying the size and amount of sections used.

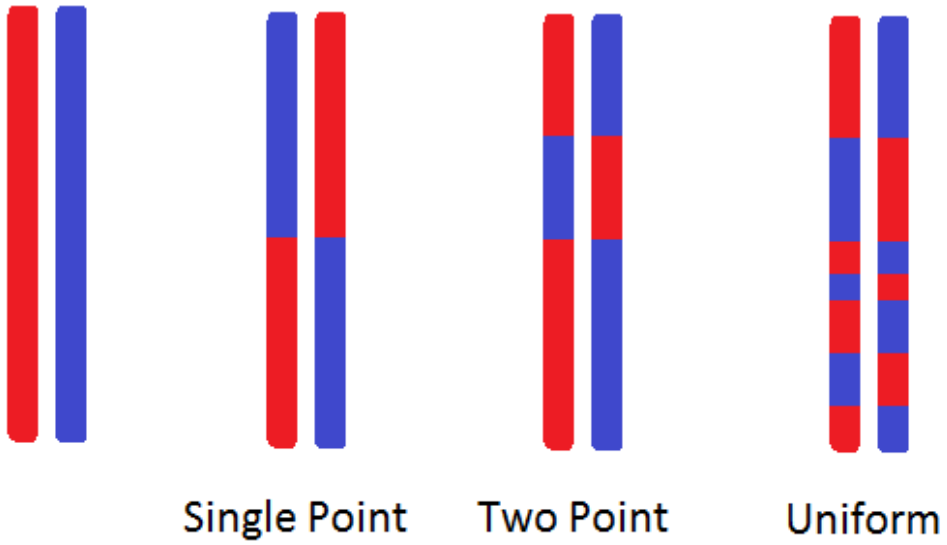


Figure 22: An illustration of what crossover does

Another crossover method we use is to randomly select the chromosomes form two individuals and make only one offspring. This method eliminates half of the total chromosomes from the parents.

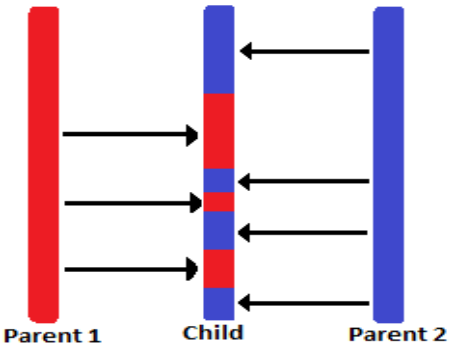


Figure 23: An illustration of how a 'child' is made in Genetic Algorithms

2.6.5 Fitness

After the new population has been created we get the fitness of each individual and start the selection process all over again.

2.6.6 Testing

When testing the Genetic Algorithm we used the traveling salesman problem (TSP)[3]. The TSP is described as such. Given a list of cities and their locations, the task is to find the shortest route to visit each city exactly once [2].

Figure 24: Traveling Salesman Problem

Figur 1 shows a good example of the problem. The salesman has to visit each city once and he would like to know the shortest route. For the six cities shown the solution is quite simple for a computer.

Figure 25: TSP solved.

A computer can find the shortest route by using bruteforce to test all possibilities. As the complexity for such problems are O(n!), that leaves only a relative small number of 720 possibilities for the computer to check.

But if the number of cities needed to visit rises it gets difficult to use this technique. If one has to visit 500 cities, the number of possibilities is just too many to effectively test.

Figure 26: 500 cities

Problems like these are ideal to test optimizing algorithms on, and we used it to check our Genetic Algorithm. The Algorithm did solve the TSP, thus confirming to us that it works. We did have to remove a few xover methods when we started to use the algorithm.

3 Results and discussion

In this chapter we show the results from testing the different methods. As we explained in the evaluation part in chapter 2, we decided on the following procedure for the tests. The methods would run the OHBV model for 10 000 iterations. This would be done 30 times. Each method was first tested on a pre-generated dataset. This showed whether the method converged towards a good output, and thus a good parameter set. If the method converged against a feedback of 1.0, that is, the correlation and mean squared root error (MSRE) of the modelled output compared to the pregenerated values were both close to 1.0, the method would likely be able to converged when the modelled output was compared to the historical values. Each run of the method was repeated 30 times so the behaviour of the methods would be better represented, and the impact of less characteristical runs would be decreased, while at the same time making it more apparent whether runs that might be considered uncharacteristic actually were uncharacteristic. We arbitrarily decided on 10 000 iterations after testing the SCE-UA on both pre-generated values and historical observations, and found that within 10 000 iterations the SCE-UA would show little discernable improvement in convergence or feedback.

When evaluating the data we look at the structure generated by the data runs. For each iteration of the OHBV method, we plot the median, second best and second worst feedback out of all 30 runs. We use the second best and second worst as a precaution against exreme outliers that may not be characteristical to the method. By plotting these three datasets for each method, we get a visual representation of the expected feedback distribution for the method. Feedback variance is the gap between the second best and the second worst feedback. This gap represents what results to expect when you run the model. The smaller the gap is the more consistend the feedback will be for each run, the large the gap is the less consisten the feedback will be for each run. We show the plots where we compare the pregenerated and the historical outputs with the second best each method. We only show the two last years and the last year. This is becuse the OHBV model needs a few years to normalize the water level in the OHBV model, the later years would be more representative for the accuracy of the modelled output.

We compared the methods based on the number of iterations of the OHBV model rather than explicit time units, since the time it takes to run the model is for all our methods far greater than the time it takes to run the code in the actual calibrating method.

3.1 Monte Carlo scheme

3.1.1 Monte Carlo scheme test results: Versus pre-generated values

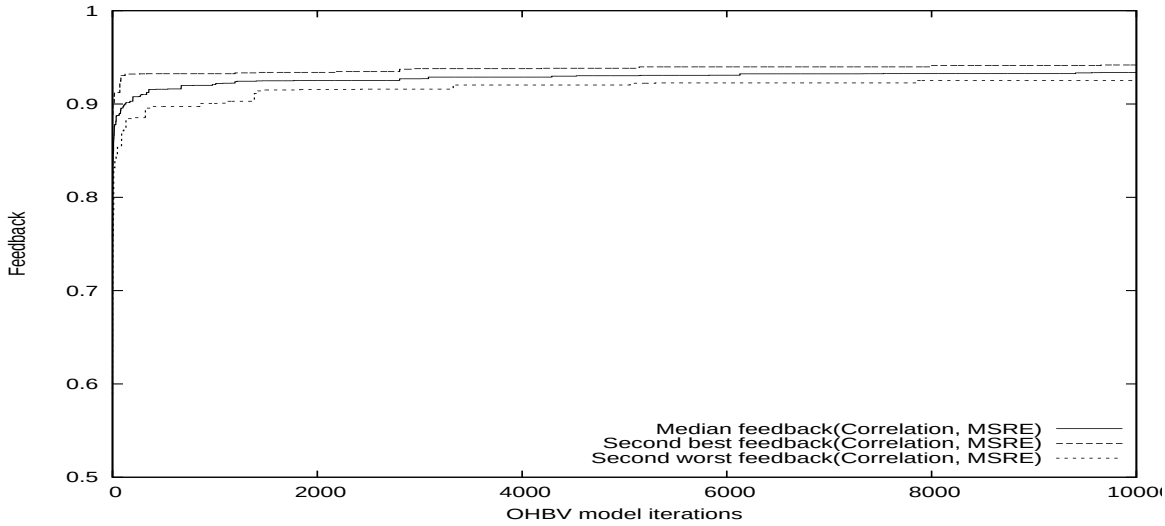


Figure 27: 30 runs of 10000 OHBV iterations. Model outputs are compared against a pregenerated model output.

When using the Monte Carlo scheme against a pregenerated model output, we see from figure 27 that a near best solution is found within 3-4000 iterations, quite close to figure 29, while the best solution produced was found within 6-8000 iterations, also quite close to figure 29. The Monte Carlo scheme starts at a value of 0.756, and all evaluations slightly converge until around 4000 iterations, at a median value of 0.929, an increase of 0.173, at an average 0.00004325 per iteration. The highest median value found is 0.934, an increase of 0.005 over 6000 iterations, an increase that is microscopic. At 10 000 iterations the second best feedback was 0.942.

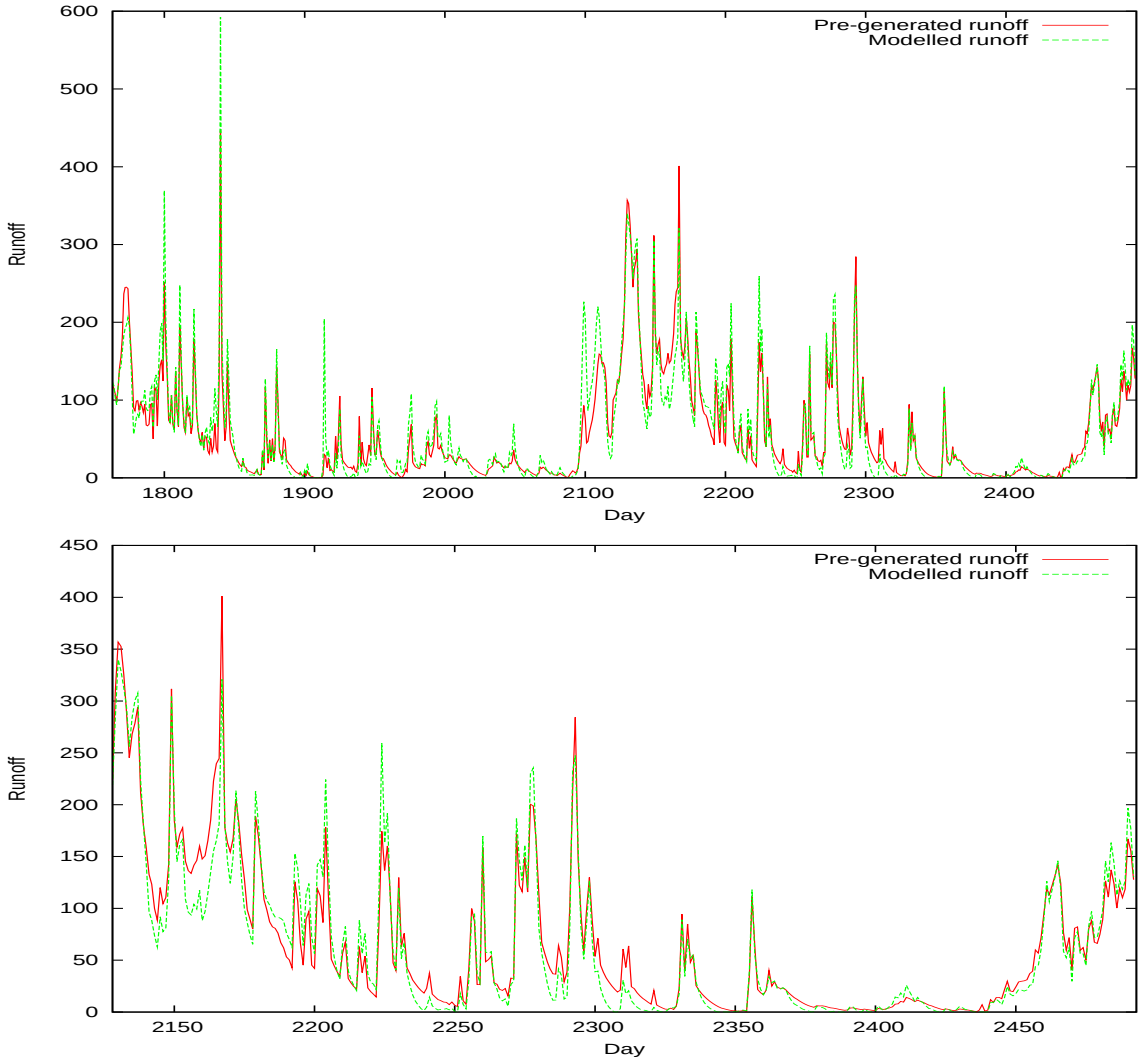


Figure 28: Second best output of 30 runs vs pregenerated runoff found by the Monte Carlo scheme. The top graph shows model output for the last two years, the bottom graph shows output for the last year.

This shows the second best ouput of the Monte Carlo scheme when compared against pregenerated values. The modelled output follows the pregenerated output quite closely rin regards to correlation, however there are some extremes in the pregenerated values that the modelled output does not match in terms of amplitude.

3.1.2 Monte Carlo scheme test results: Versus observed historical values.

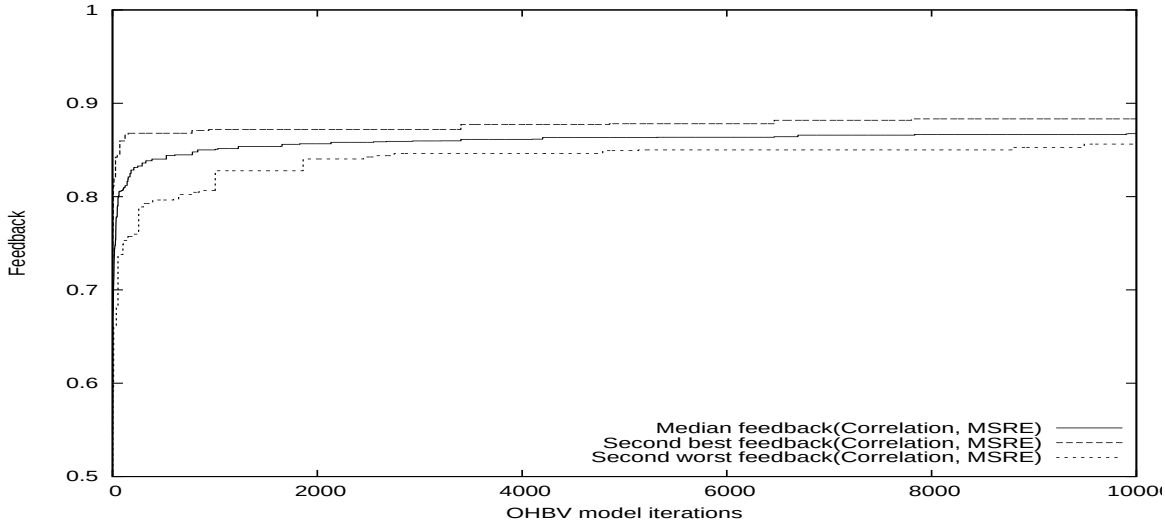


Figure 29: 30 runs of 10000 OHBV iterations. Model outputs are compared against observed historical values.

When using the Monte Carlo scheme against observed values, we see from figure 29 that a near best solution is found within 4000 iterations, while the best solution in total from the figure and data was found within 7-8000 iterations. The Monte Carlo scheme starts at a median value of 0.54. The evaluations do not appear to significantly converge, but there is a consistent improvement in values. The largest improvements are made within 2000 iterations, at which iteration the median value is 0.858, an improvement of 0.318, and an average improvement of 0.000159 per iteration. The highest median value found is 0.871, an improvement over 8000 iterations of 0.013, an average of 0.000001625 per iteration. After 10 000 iterations the second best feedback value was 0.883.

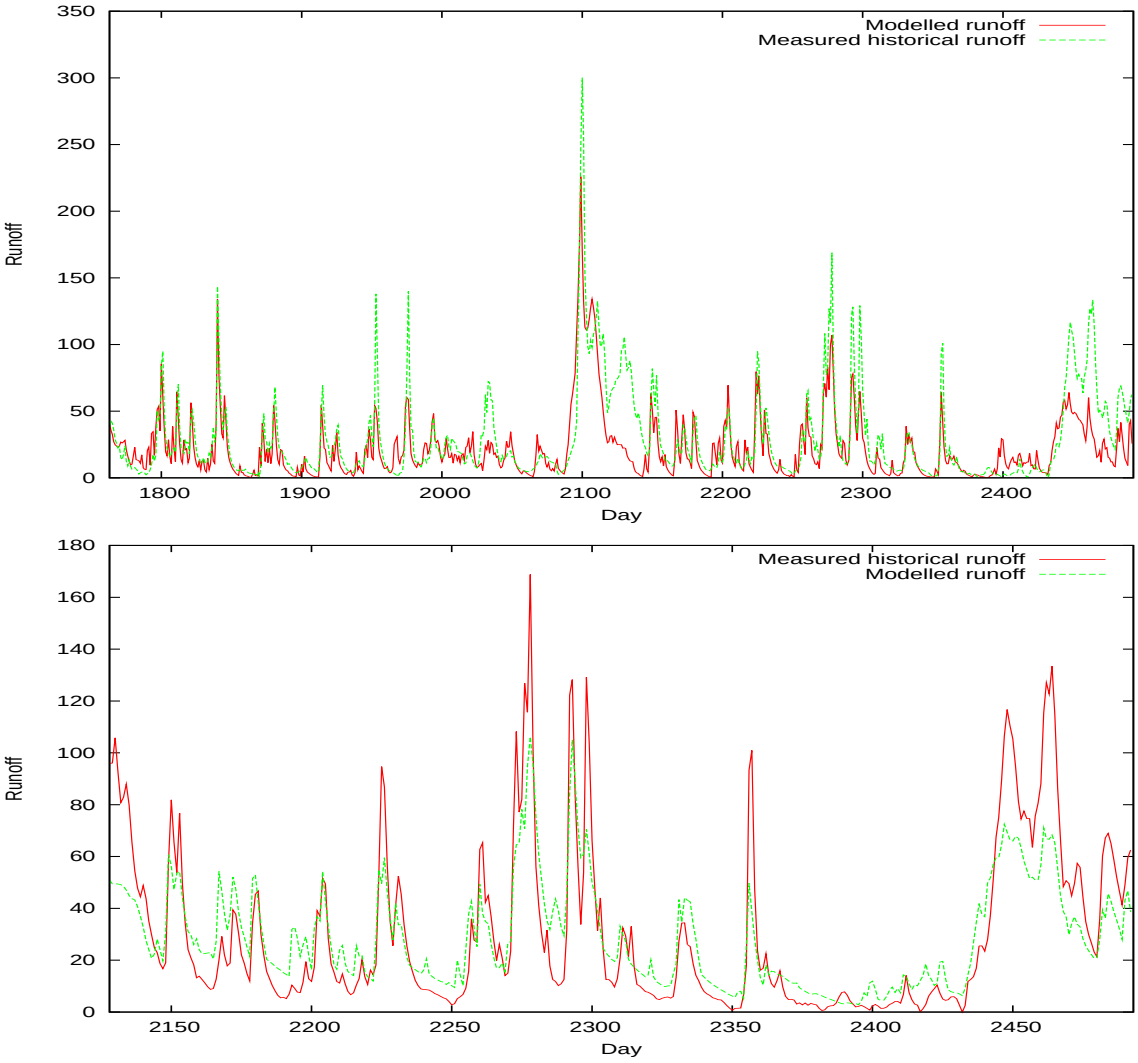


Figure 30: Second best output of 30 runs vs observed runoff found by the Monte Carlo scheme. Top graph shows output of the last two years, bottom graph shows model output for the last year.

This shows the second bets modelled output of the MCS compared to observed historical values. Here again there is an overall tendency in the modelled output to follow the observed values in terms of correlation, however around day 2100 we an see an inverse correlation between historical measurements and modelled output.

3.2 SCE-UA

The SCE-UA implementation effectively found a good parameter set within a short time. The SCE-UA method will not produce a full 10 000 outputs because it must first construct a parameterset population. In these cases the population consists of 4 complexes with 73 parametersets each. This means that the OHBV model must run for 292 iteratons before the method starts. Every time the algorithm runs the OHBV model, the parameterset with the highest criterion value is printed.

3.2.1 SCE-UA test results: Versus pre-generated values

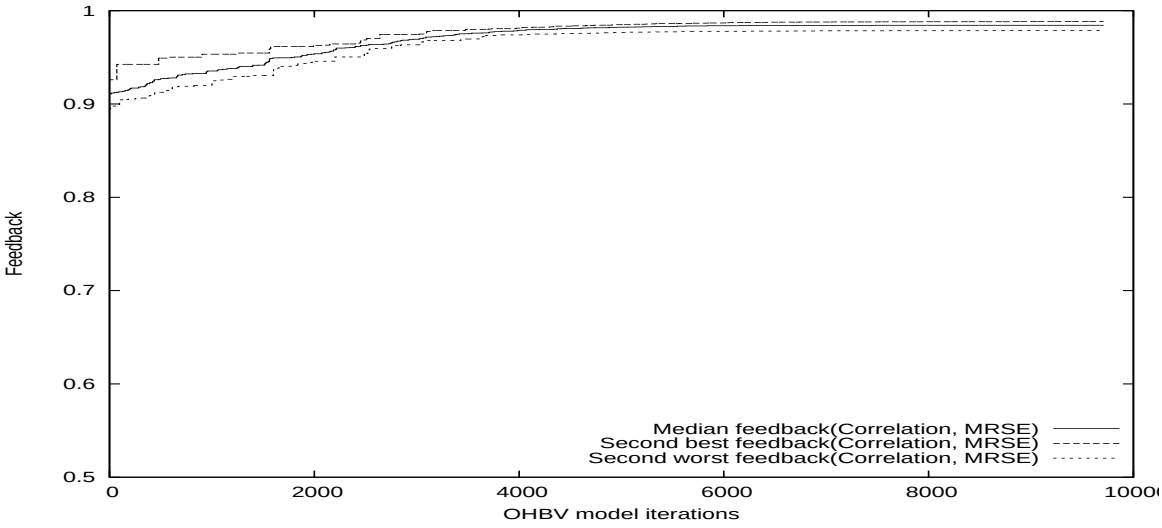


Figure 31: 30 runs of 10 000 OHBV iterations, plot shows median, second worst and second best values. Model outputs are compared against pre-generated values.

As can be seen in figure 33, the SCE-UA method produced a parameterset that allowed the OHBV model produce an output that was very close to the observed values. The output feedback appears to start very high, but we expect this from the method because it samples about 300 parametersets beginning its actual function, after which it writes the highest values to the plots. As such, we expect the SCE-UA to have a comparable distribution of fitness at the start that we see from the best ouputs of the Monte Carlo scheme after 300 iterations. Within the first 4000 iterations the median feedback goes from 0.91 to 0.979, an improvement of 0.069, and an average of 0.0001725 per OHBV model iteration. From 4000 iterations to 10 000 iterations, the SCE-UA output improves from 0.979 to 0.984, an improvement of 0.05, a very small improvement.

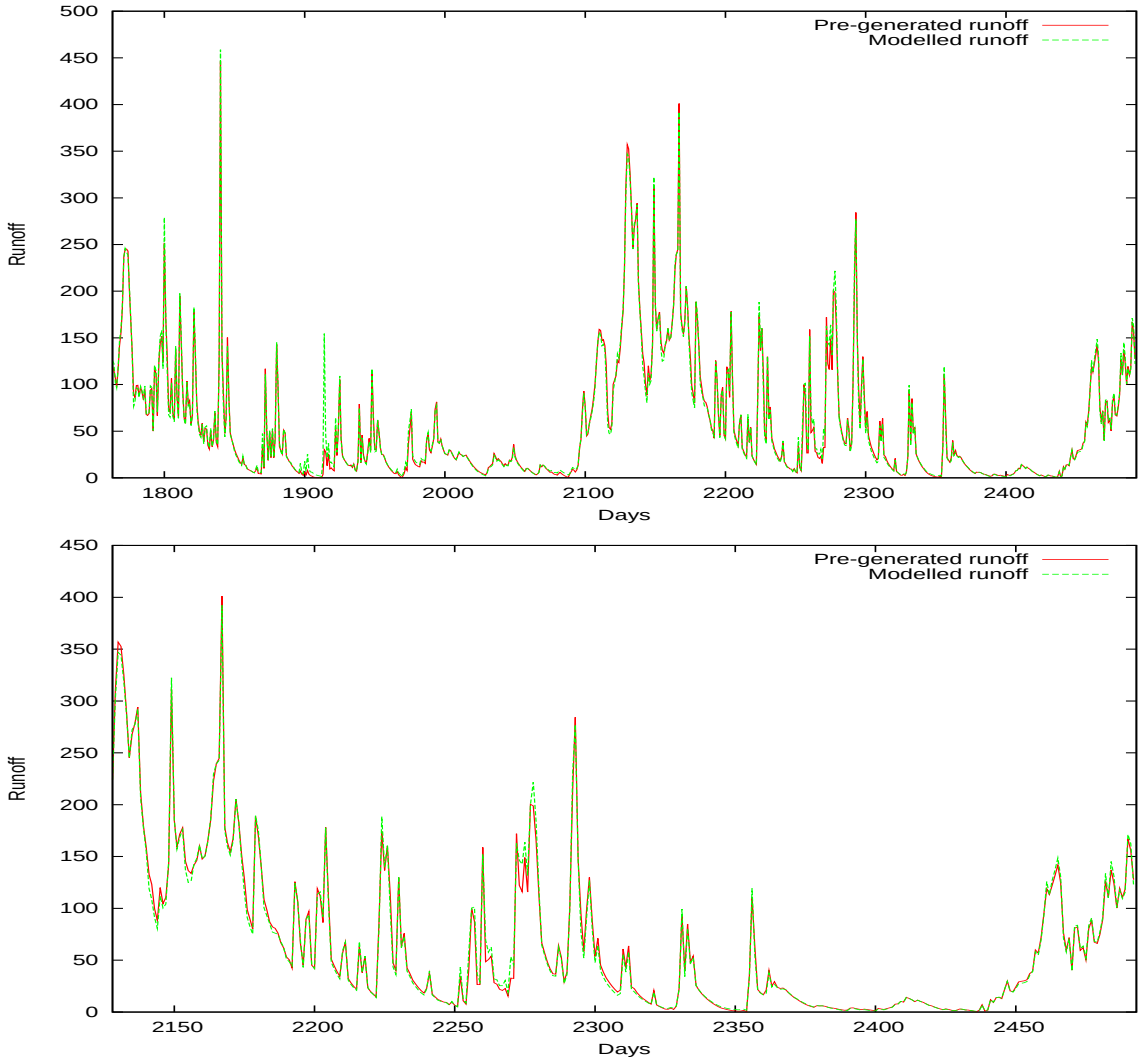


Figure 32: Shows the plot for the pre-generated runoff and the second best result from the SCE-UA method. Top graph shows modelled output for the last two years, and bottom graph shows modelled output for the last year.

This shows the second best output of the SCE-UA method against pregenerated values. As can be seen from the graphs the modelled output fits very closely to the pregenerated output. This is expected as a deterministic model will produce the same output from identical inputs.

3.2.2 SCE-UA test results: Versus observed historical values.

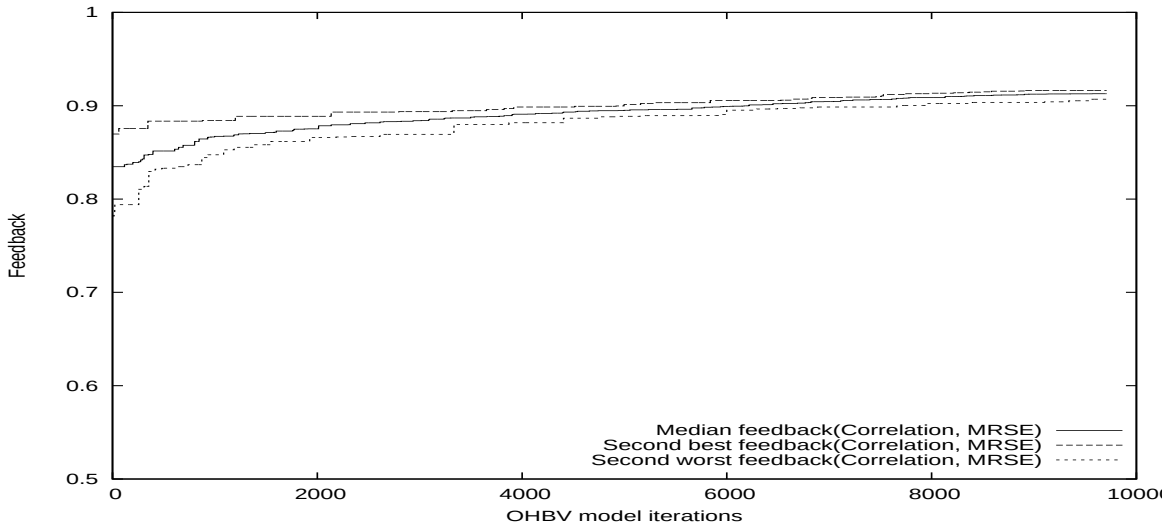


Figure 33: 30 runs of 10 000 OHBV iterations, plot shows median, second worst and second best values. Model outputs are compared against observed historical values.

As can be seen in figure 33, the SCE-UA quickly produces an ouput that's very close to 0.92. A close convergence between each dataset usually starts at about 2000 iterations of the OHBV model. Within the first 2000 iterations the median feedback goes from 0.835 to 0.875, an improvement of 0.04, and an average of 0.00002 per OHBV model iteration. From 2000 iterations to 10 000 iterations, the SCE-UA output improves from 0.875 to 0.913, an improvement of 0.038, and an average of 0.00000475 per OHBV iteration. The shape of the curves show a steady rate of improvement, up to 10 000 iterations.

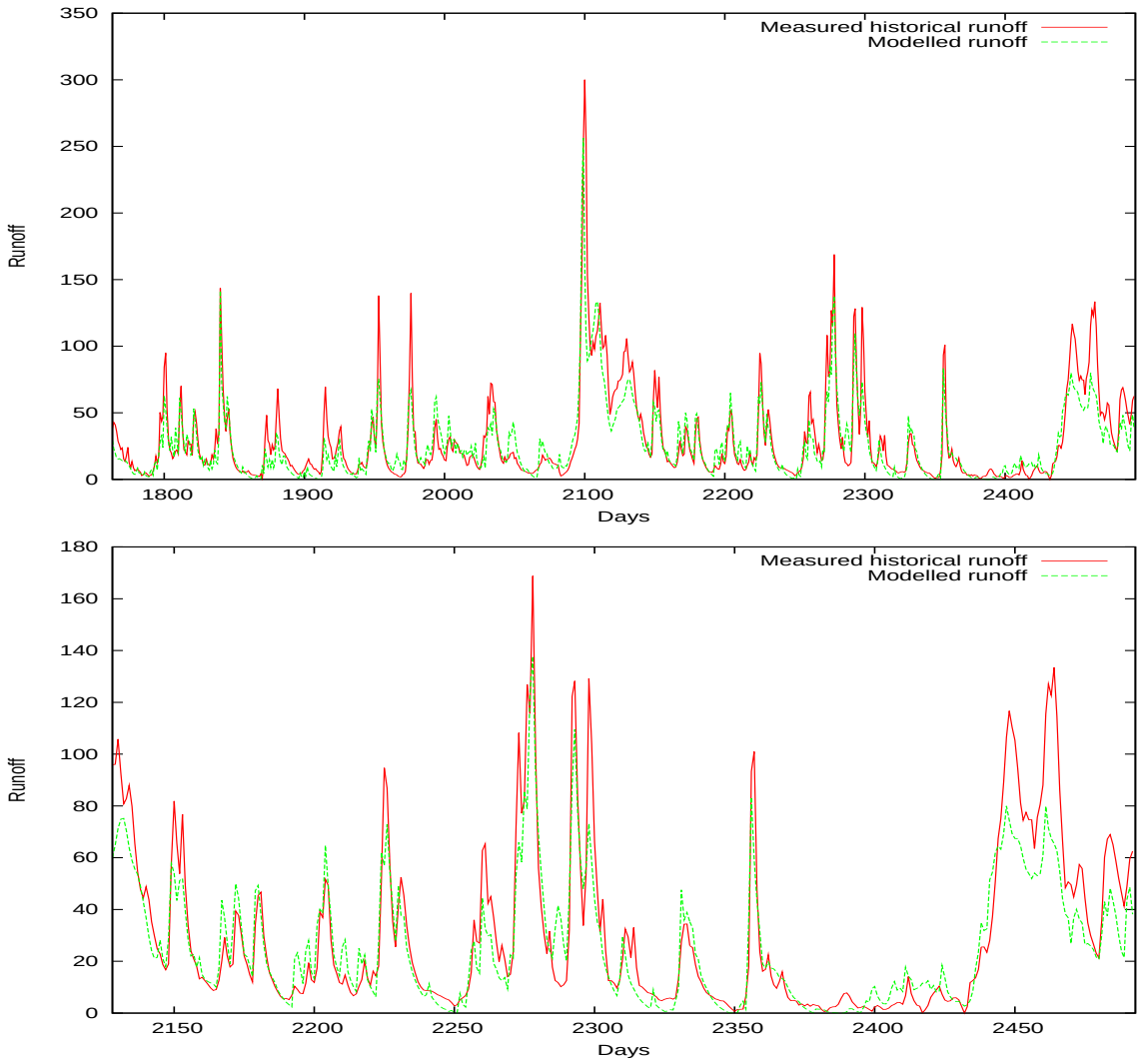


Figure 34: Shows the plot for the observed historical result and the second best result from the SCE-UA method. The top graph is modelled output from the last two years and the bottom graph is modelled output from the last year.

As can be seen from Figure 34, the modelled runoff closely follows the observed runoff, especially with regards to correlation. This means that the model, and by extension, the parameter sets, now simulate a situation that's quite close to the physical properties of the water catchment. The differences in the curves can stem from imperfect models, to our methods not being able to generate better parameter sets, to measurement errors in the historical data to freak weather.

3.3 CALA

There are two separate tests we did for the CALA. The two tests was against the pre-generated model output and the historical data. in the figures 35 and 37 we have modified the plots to only show the increase in values.

3.3.1 CALA test results: Versus pre-generated values

As the described at the start of the chapter, each test set were constructed to run the OHBV model 10 000 times and there where 30 tests done. Beacause CALA needs feedback for the value it is trying to optimize and its randomly generated value. Means that for each iteration in the CALA loop it does runs the OHBV model two times. Because we are only interseted in the feedback for the value it is trying to optimize we only get 5000 feedbacks. The test where done against the pre-generated model output and gave the following result:

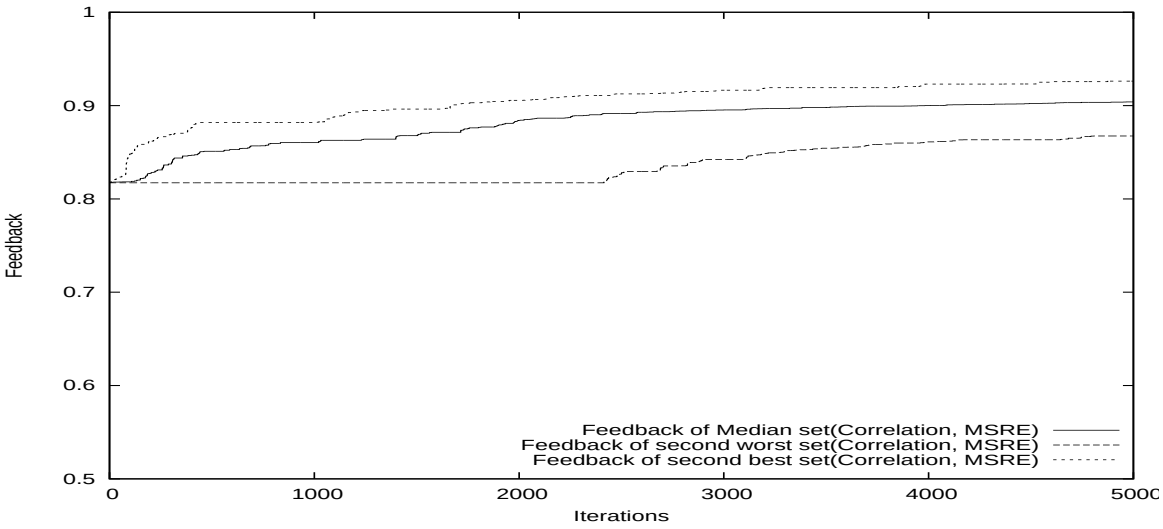


Figure 35: The graph shows the second highest and lowest value for each iteration and the median value for each iteration.

The plots show that the variance in the feedback is quite large compared to what it is in SCE-UA and Genetic Algorithms, and the feedback is also lower. We see that the plots start at the same feedback, 0.8172. This is because we can decide the start value of a CALA and had decided that they all should start with middle value a parameter can be. We see from the median plot that it has a small rise until it reaches 2500 iterations and then levels out. This gives us a rise from 0.8172 to 0.8912, an increase of 0.074 and an average of 0.0000296. From 2500 to 5000 iterations it rises from 0.8912 to 0.9038, an increase of 0.0126 and an average of 0.00000504. The second best feedback are 0.9243 and the second worst are 0.8515. The feedback variance at iteration 5000 is 0.0728.

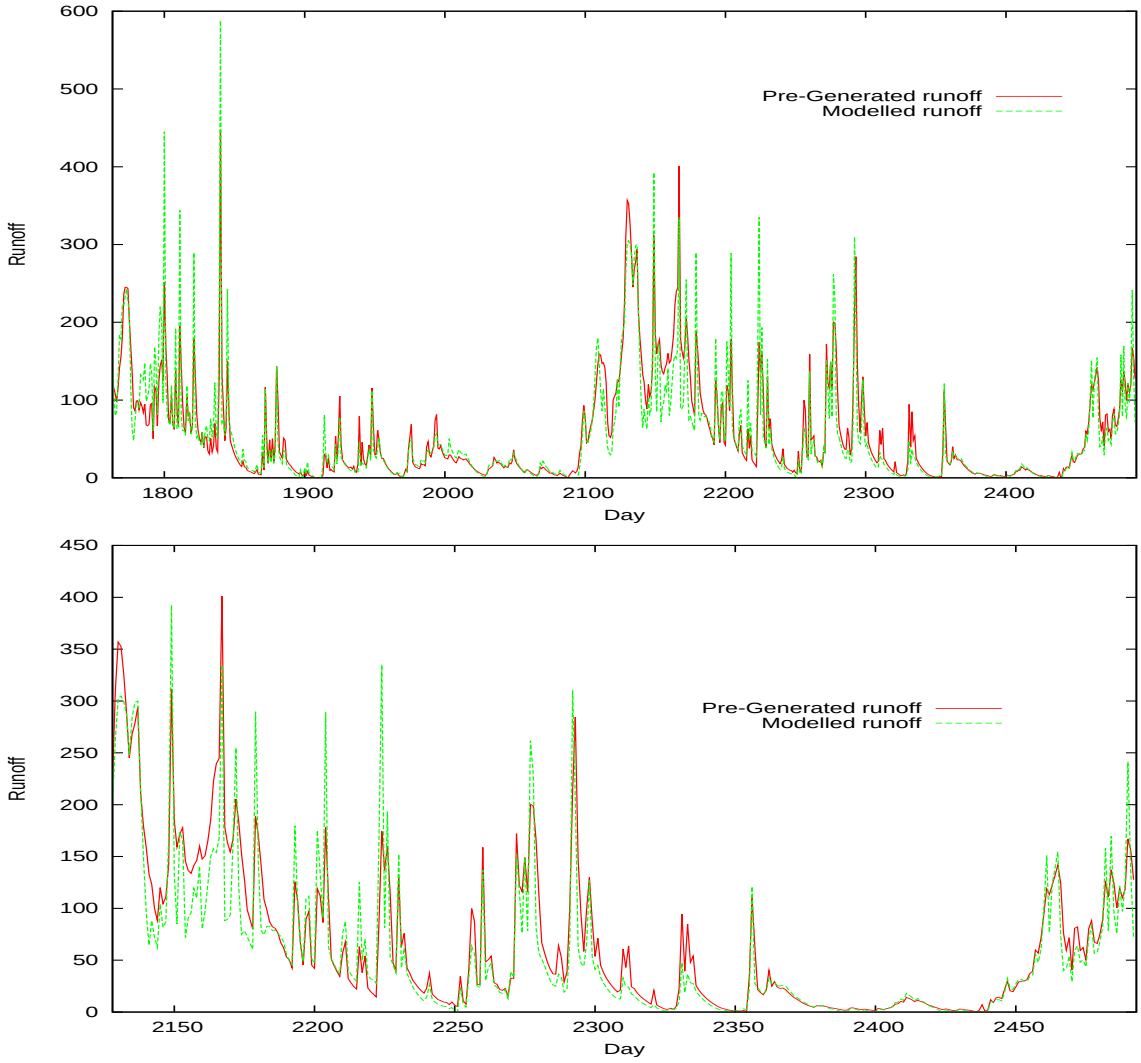


Figure 36: Shows the graphs for the pre-generated result and the second best result CALA managed. Top graph shows model output from the last two years and bottom graph shows output from last year.

The plots in figure 38 shows that the modelled runoff follows the plot of the pre-generated runoff, but in general has a lower value and has a higher number of peaks and lows. In comparison to the plots for SCE-UA or Genetic Algoriths (figure 40 and 32) we see that the plot for the CALA model very inaccurate.

3.3.2 CALA test results: Versus observed historical values

The second test were run against observed historical values and gave the following result

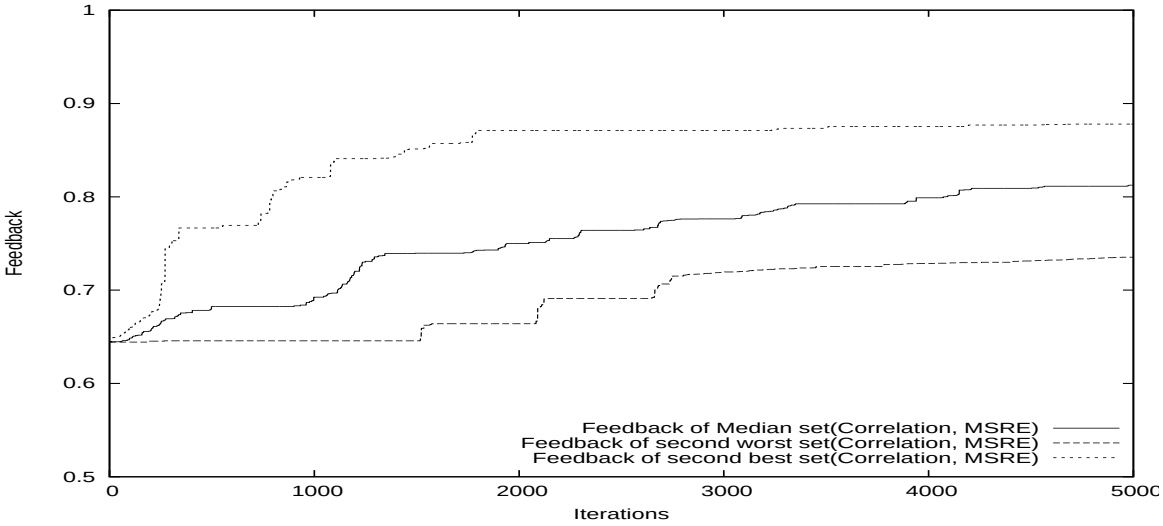


Figure 37: The graph shows the second highest and lowest value for each iteration and the median value for each iteration.

The median value starts at around 0.6442 and rises to its highest feedback of 0.8077 at 5000 iterations. Thats an improvement of 0.1635 and an average of 0.0000327. The second best got an feedback of 0.8598 at its best and a second worst feedback at 0.7351. These feedback results are barely better then the Monte Carlo Scheme. The median plot never seems to level out, it follow a steady rise. The variance in feedback is also the largest of the methods at the size of 0.1247.
By plotting out the result from the second best parameter set and the observed historical runoff set we compared it to we get a visual representation of how accurate the results are.

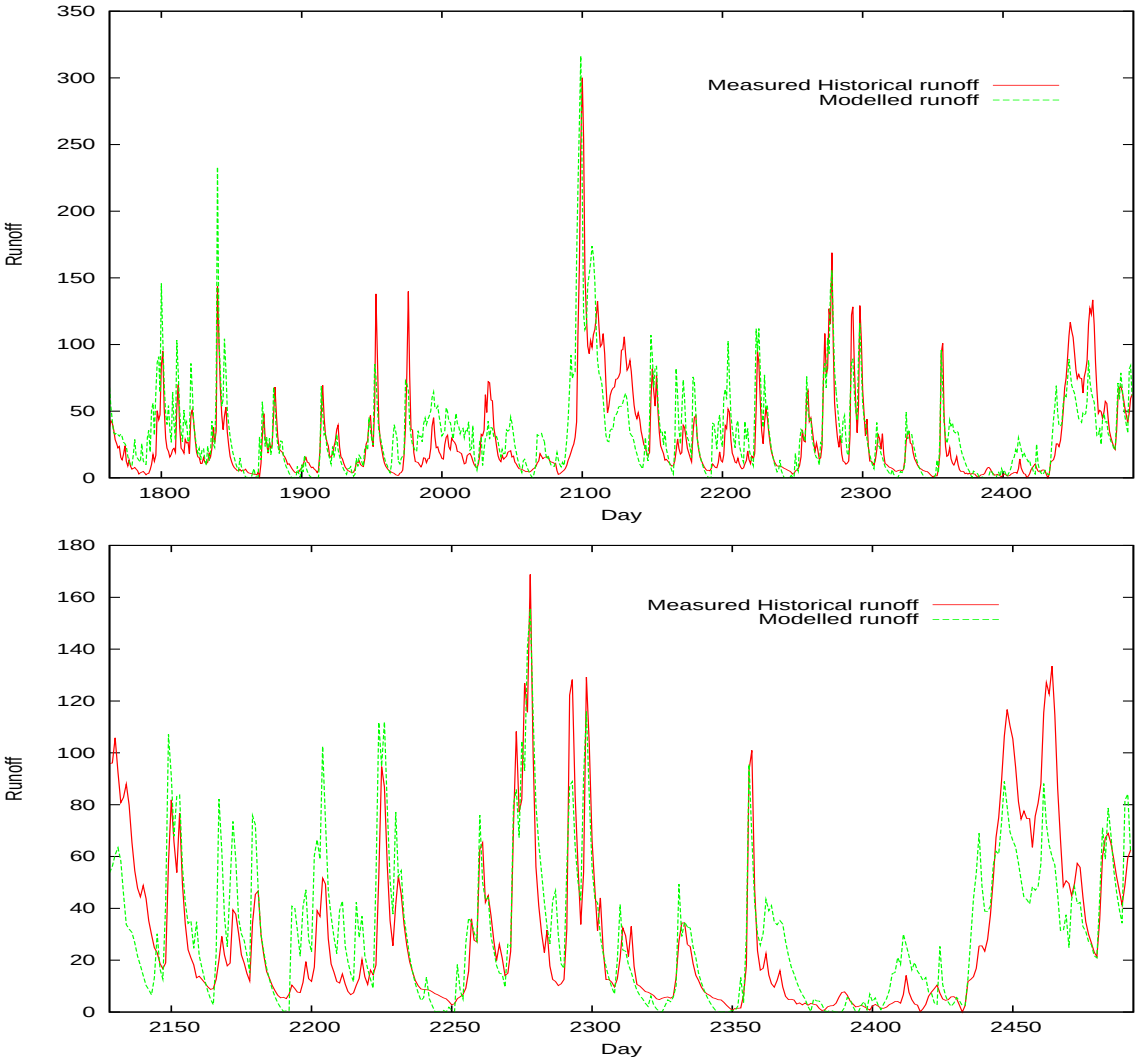


Figure 38: Shows the graphs for the observed historical result and the second best result CALA managed. Top graph shows modelled output for the last two years, bottom graph shows modelled output for the last year.

From looking at the plots in figure 38 we se that the modelled runoff follows the contours of the observed historical runoff, but the values are very incorrect often showing more runoff then what has been observed.

3.4 Genetic Algorithms

The were two test done for the Genetic Algorithms(GA). One aginst a pregenerated modeled output and on against a historical data set. As described in the evaluation part, each test set consisted of thirty tests with the OHBV model run for 10 000 iterations. GA was setup with a population of 30 and needed only 333 iterations for each test.

The GA relied on several different values for configuration. Specifically three values for the amount of individuals the selection algorithms should select (Elitism, roulette, tournament), an how large the new population should be. After running several test we decided to use a new population size of 30, and a size of 4,8,8, for the size of elitism,roulette,tournament. Because GA needs to check the fitness of each individual, the ohbv model will be run 30 times for each iteration of GA, therefor we only need to run it 333 times.

3.4.1 GA test results: Versus pre-generated values

The results from the pre-determined model output was this:

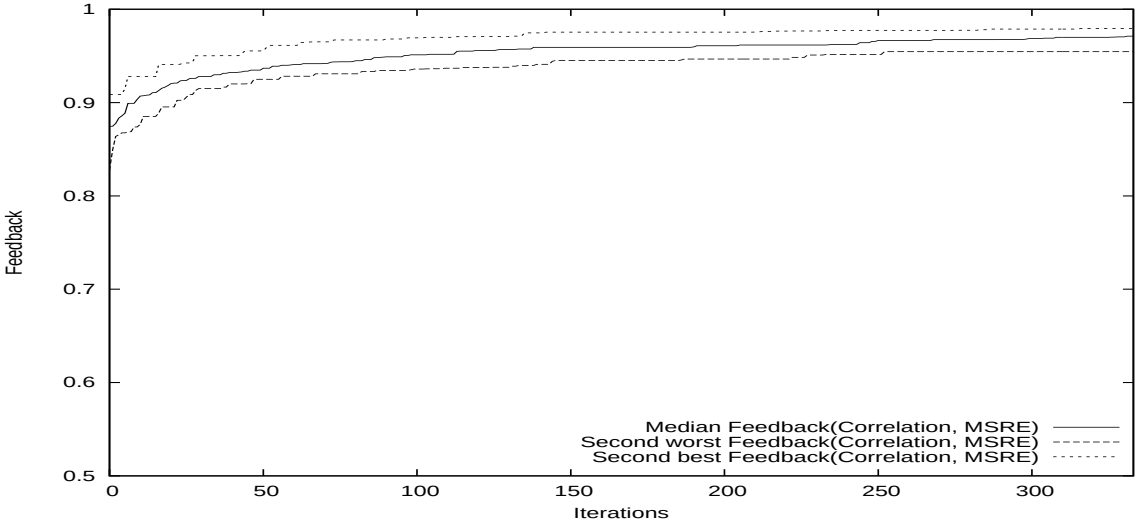


Figure 39: The graph shows the second highest and lowest value for each iteration and the median value for each iteration.

The Algorithm starts off quite high, between 0.85 and 0.90. And that may be the reason we dont see a characteristical fast rise at the start as we see in figure 42 where it is compared to the historical data. It seems to rise abit faster before the 50 iteration mark and then mellow out; and as mentioned before, since it already started at such a high feedback the improvements over each iteration would be lower than if it had started at a lower feedback, as evident by looking at figure 42. From 0 to 50 iterations it rises from o.8741 to 0.9345, an improvement of 0.0604 and an average of 0.001208 each iteration. From 50 iterations to 300 iterations it rises from 0.9345 to 0.9677, an improvement of 0.0332 and an average of 0.000166. The shape of the distribution is consistent, but the variance of the feedback distribution is alittle wider than what it is on the historical results. The size of the variance is 0.0156.

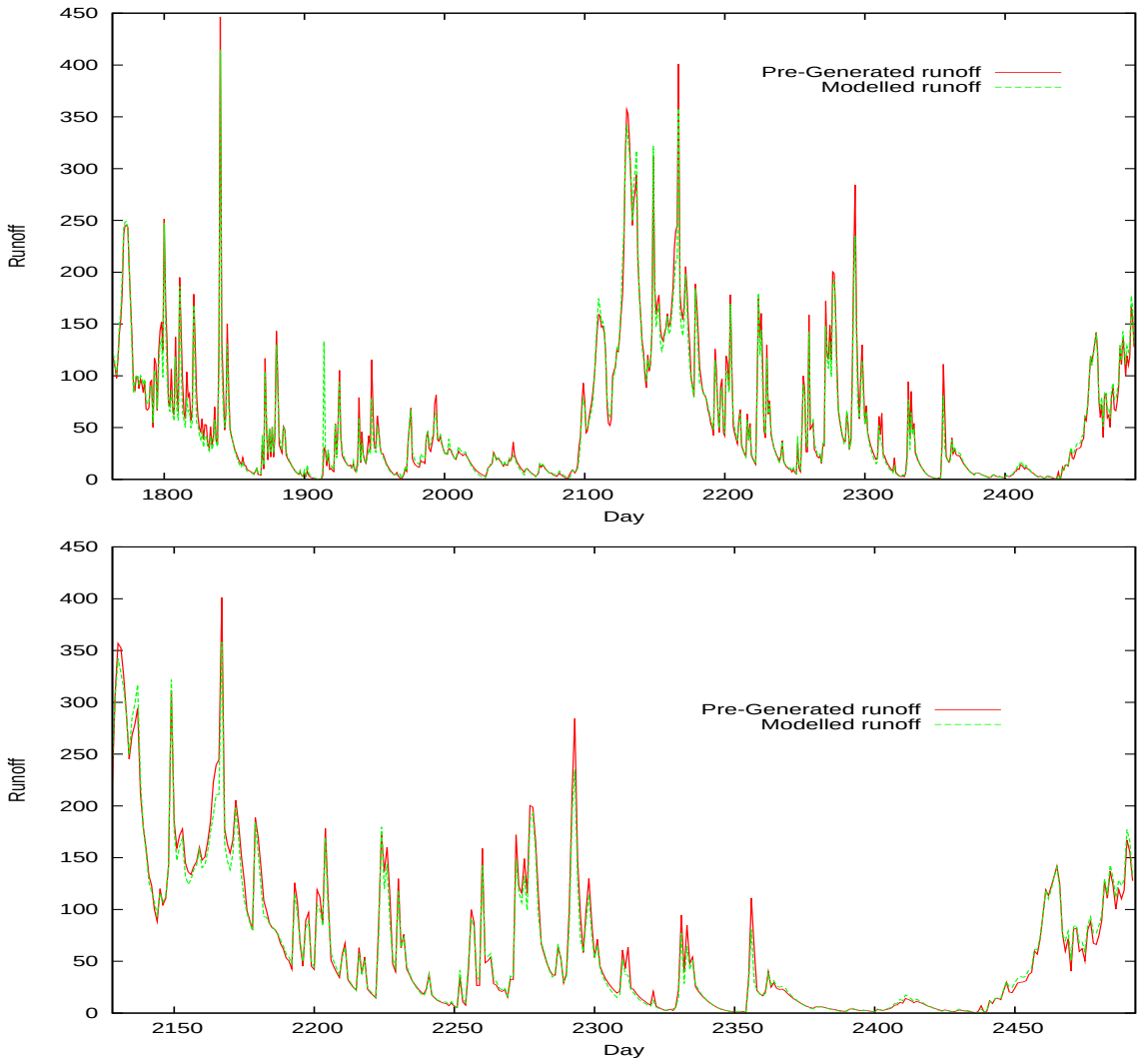


Figure 40: Shows the plot for the pre-generated result and the second best result from Genetic Algorithms. The top graph shows model output from the last two years, the bottom graph shows model output from the last year.

Figure 40 shows the plots for the observed pregeneratet water runoff and the modeled runoff. The pre-generated set was made so the configuration methods could have a set where they could get a very close to the optimum result, and that shows on the plots. When looking at the plots we see that they are almost identical. In general the modeled runoff seems to bee abit lower than the pre-generated runoff, but but the modeled result almost exactly follows the shape of the pre-generatedvvrunoff. This is because the pre-generated runoff has no measurment errors or unusual weather conditions, and is itself made by running the model contra to the historical observed runoff which is made by actually measuring the runoff.

3.4.2 GA test results: Versus observed historical values

The results from the historical data test:

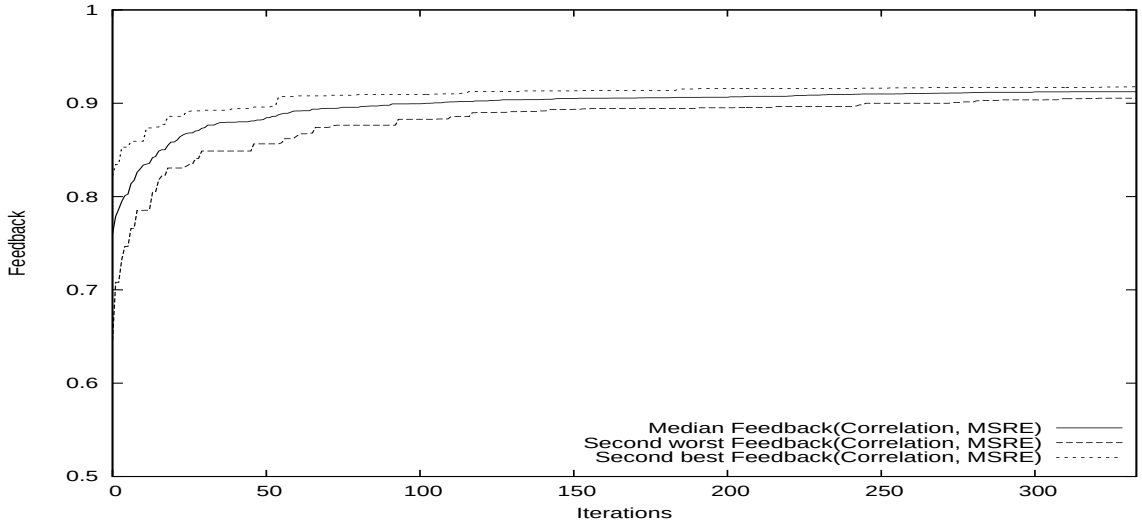


Figure 41: The graph shows the second highest and lowest value for each iteration and the median value for each iteration.

The Algorithm rises quickly until it levels out around a 100 iterations(2000 ohbv iterations). In those 100 iterations the median feedback goes from 0.7515 to 0.8966. Thats an improvement of 0.1451, an average of 0.001451 for each iteration. From a 100 iterations to the end of 333 iterations the median feedback goes from 0.8966 to its highest value 0.9122. In 233 iterations it increases by 0.0156, thats an average of 0.000078 pr. iteration. The second highest plot reaches a feedback of 0.9175. The shape of the distribution quite satisfying becuase it rises quickly and showing a consistent behaviour. The size of the variance in feedback is 0.0119

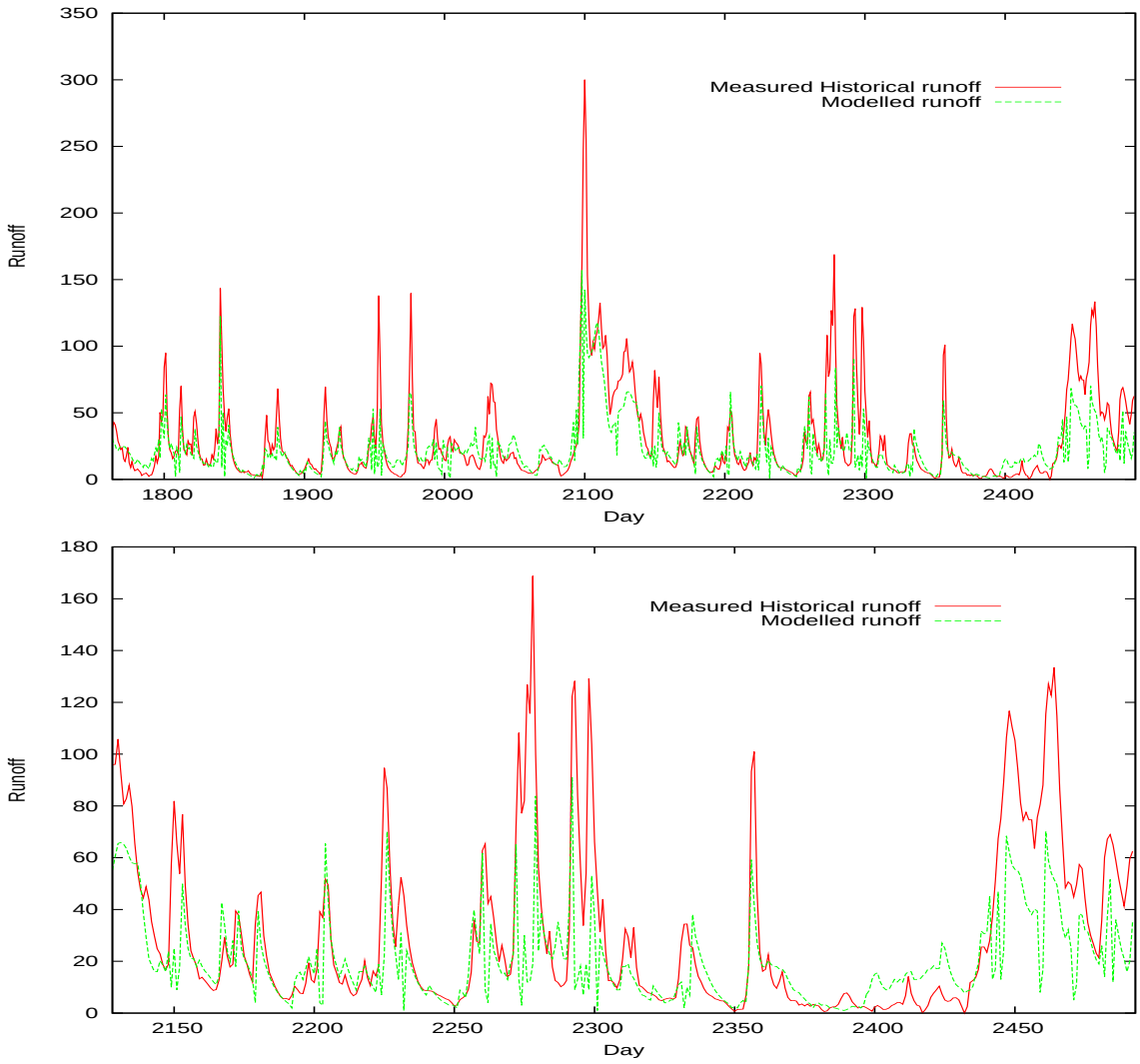


Figure 42: Shows the plot for the observed historical result and the second best result from Genetic Algorithms. The top graph shows modelled output from the last two years, teh bottom graph shows the modelled output from the last year.

Figure 42 shows the plots for the observed historical water runoff and the modeled runoff. The plots show that when the runoff is high, especially with narrow peaks, the observed runoff is higher than the modeled. But when the plots are not as extreme the observed runoff seems to generally be bit lower then the modeled runoff. We also note in figure 42 that around day 2460-2470 there seems to be som observed runoff that the model did not get, when looking at how closely the modeled runoff follows and imitates the observed runoff. Discrepancies like these could be due to some measurment error or extreme weather, or to the model not being able to perfectly reproduce the physical conditions of the water catchment, or the parameters being incorrectly calibrated.

3.5 Method Comparison

After presenting the figures for each method we now compare them. When comparing the methods we analyze the data we have gotten from each method and discuss how they compare to eachother.

3.5.1 Monte Carlo Scheme

In our tests the Monte Carlo Scheme (MCS) produced a reasonable feedback value, this means that the models output would within 10 000 iterations fit the observed values reasonably well. 29 The feedbacks of all the 30 runs, while not showing any obvious signs of actual convergence beyond the first 1000-2000 iterations, were still closely grouped, usually within the 0.01 - 0.02 range of each other when the model ouput was compared against observed values. (See figure 29) Whereas it was usually within the 0.01 range when model ouputs were compared against a pregenerated set. (Figure 27) Because the outputs remain at these close groupings from around the 2000 iterations we be reasonably certain that using an MCS to calibrate the model in this situation will usually give a resulting feedback value within the 0.856 - 0.883 range. The shape of both graphs (figures 29, 27) are what we would expect from an MCS. Initially (1000-2000 iterations) improvements in feedback are often made, however, as the highest feedback found increases, the probability of finding a higher feedback decreases, resulting in a flattening of the feedback curve. In both cases, we expected most of the improvements in the feedback values to happen within the earlier iterations, and our results show that it did.

While very little real convergence between feedback values happens, or could reasonably be expected from an MCS, there convergence that does happen happens within the first 1000-2000 iterations. Comparing modelled outputs against observed values vs comparing modelled outputs against pregenerated values shows little difference in how the MCS performs, aside from giving a generally lower feedback value. This implies that in our case the MCS did not diferentiate between a deterministic comparison situation and a non-deterministic comparison situation.

3.5.2 SCE-UA

In our case the SCE-UA gave a very high feedback value, both when comparing modelled output against pregenerated values, often in both cases with a correlation and MSRE of around 0.99, and when comparing modelled output against observed values.

Against both pregenerated values and observed values, the feedback showed a clear convergence between second best, second worst and median feedack value in the 0-4000 iterations range. Against observed values the feedback values end up very closely grouped, with a median value of 0.913, second best value of 0.916 and a second worst value of 0.907. A total variance of 0.009, a variance from median to second best of 0.003 and a variance from median to second worst of 0.006. This means that using the SCE-UA method to calibrate a parameter set in our situation will usually give an output feedback in the 0.907-0.916 range.

Against pregenerated values, the feedback values also end up very closely grouped. With a median value at 0.984 and a second best value at 0.988, a varianceof 0.004, and a second worst value at 0.979, a variance from the median value of 0.005 and a total variance once again of 0.009. In this case, though, the variance was more equally spread between median to second best and median to second worst.

We believe this discrepancy in variances is due to outliers affecting the results. In both cases the feedback graphs mostly converge within the first 4000 iterations. The graphs showed a distinct characteristic difference between comparing the modelled output against measured values and comparing it against pregenerated values. When comparing the modelled output against pregenerated values (figure 31) there is a relatively steep improvement curve during the first 4000 iterations, after which the curve essentially flattens and no easily discernible improvements are made. When comparing the modelled ouput against observed values on the other hand (figure 33), there is a relatively steep improvement curve within the first 1000 iterations, after which the curve becomes less steep but is still showing a steady and consistent improvement until it also flattens out after 9000 iterations.

We believe this is due to one situation being deterministic in nature (the pregenerated value) and the other being non-deterministic (historical measurements). By this we mean that the pregenerated values have a direct relationship with the model. Since the model is deterministic in nature, two equal parametersets using the same input data will produce the same output. We believe that the SCE-UA method can therefore more easily find the best parameter-set in this situation because a good output is in this case tied directly to specific parameter values rather than a combination of parameter values trying to match a model output to historical measurements that may contain errors or the model is not able to perfectly reproduce.

3.5.3 CALA

When compared to the other three methods CALA comes out as the worst. The feedback values are of the lowest and increases slowly, and it has a large variance in feedback values. We ran 36 CALAs as a team, and we believe this affected the results we got. We believe that because of the size of the CALA team the CALA's speed was affected but not the accuracy. As we can see from the plot in figure 37 that is is slowly improving.
We believe, based on the theory behind the CALA method, that if run for enough iterations it would get a feedback as good as or close to the rest, and a reduction in the variance of feedback values. However, in our case this shows that our implementation of the CALA method is less efficient than all the other methods, producing the lowest feedback values and the largest variance between feedback values.

We did not expect our CALA implementation to produce a feedback variance of this size. At iteration 5000 in figure 37 the variance between the second best and second worst is 0.1285. This is by far the biggest feedback variance of all the methods. This makes the CALA unpredictable as one would have to expect a large discrephancy every time the method is run (Ranging between -> 0.80 <- and 0.86). Compared to the other methods we find this to be unacceptable. When comparing the pregenerated feedback and the observed historical feedback we see that it follows the same pattern as the other methods. The pregenerated has a better feedback, a better distribution spread, and a better plot. But when compared to the other methods pregenerated results it is still the worst. We believe that our results show that the CALA method is clearly the worst of the four methods.

3.5.4 Genetic Algorithms

The feedback values for the Genetic Algorithms (GA) are amongst the highest with a second best of 0.9175, a second worst of 0.9097 and a median value of 0.9122. The distribution spread is at 0.0113, a spread of 0.0059 from median to the second best and a spread of 0.006 from median tol second lowest This means that using the GA method to calibrate a parameter set in our situation will usually give an output feed back in the 0.9097-0.9175 range. The caractheristics of the graf is also very good, it shows hat at the beginning it learns fast and then mellows out. When comparing the historical feedback to the pregenerated feedback we see that the pregenerated is better in almost all aspect except distribution spread. The difference in distribution spread is not large enough to cause concern. We belive this is a natural difference and if more runs where done the difference would disapear.

Against pregenerated values, the feedback values also end up very closely grouped. With a median value at 0.9648 and a second best value at 0.9827, a varianceof 0.0079, and a second worst value at 0.9671, a variance from the median value of 0.0077 and a total variance of 0.0159. In this case, though, the medain value was closer to the upper spread. When comparing the historical plot in figure 37 and pregenerated plot in figure 35 we see that the historical plot has a more classical shape. rising fast at the beginning an then mellow out. But the pregenerated plot starts alot higher than the historical, and this could explain why the pregenerated plot lacks that shape.

4 Conclusion and further work

4.1 Conclusion

From our results we make the following conclusions. They are relative to our situation, in which we attempt to use each method to find a good fit between modelled output and measured historical values or pregenerated values while running the OHBV model no more than 10 000 times. Each method was run 30 times to give us an acceptable sample size.

Which method is best at calibrating the model in our situation? In order to answer this question, we had to do several things. First, we had to gain some knowledge of the theory and technology involved, this included some knowledge of how the model we used worked and how the parameters were used by the model, chapter 2.1 details this. We had to research the calibration methods to such an extent that we could modify each algorithm to work as efficiently as possible for our situation of a maximum of 10 000 OHBV model iterations. We detail the theory behind the algorithms and how they were implemented in chapters 2.3, 2.4, 2.5 and 2.6. In addition to this, we had to find a way to provide a viable feedback to the methods that would allow them to improve their parameter sets. This we detail in chapter 2.2.
Once we had implemented and configured the automatic calibration methods, we tested them against a pregenerated output from the OHBV model and against observed historical values. This produced some results that we detail and analyze in chapter 3. From analyzing these results we believe we can make a viable comparison and answer this research question.
The results we got from the tests shows showed that the SCE-UA and Genetic Algorithms (GA) had very similar results. They resulted in the highest value of feedback with the smallest amount of variance in feedback values. While there were some small differences between their final model outputs, they were so small that we can make no decisive conclusion between these two methods based on the result we got from our sample size of 30 runs.

The Monte Carlo Scheme (MCS) gave an efficient and well spaced variance in feedback values, but was not able to achieve the high value and low variance that the GA and SCE-UA methods produced. The biggest difference between the MCS and the SCE-UA and GA methods was the value of the feedbacks.

In comparison, we conclude that the worst method for our situation was the CALA method. It produced the lowest feedback values of all the methods we compared, lower feedback values than the MCS, while producing the largest variance in feedback values. We believe that our results for the CALA method shows that it is not suitable for use in our situation. The theory behind CALA suggests that if the CALA was allowed to run the OHBV model for a larger number of times, it would eventually find better feedback values and decrease the spread in the variance of feedback values, however, if the other methods can find a high feedback value with low variance within 10 000-20 000 OHBV model iterations, this would make the CALA method far too inefficient to be comparable.

How does each method calibrate the parameter set when the modelled output is compared to a pregenerated output from the same model, compared to when the modelled output is compared to historical measurements? Out of our four methods, SCE-UA and GA seem to be impacted by the different situations of comparing the model output to pregenerated values and observed values. In figure 31, and as mentioned in results and discussion, we can see from our results that the SCE-UA method finds a very high value within 4000 iterations and then essentially flattens, producing very little improvements afterwards. As a comparison, in figure 40, the GA method appears to produce a relatively gentle but consistent slope of improvement in the feedback value. When the modelled output is compared to historical observed values the situation is reversed in figures 33 and 42 the SCE-UA method shows a gentle but consistend slope of improvement until about iteration 9000, after which it flattens, while the GA method appears to produce most of its improvements within the first 4000-4500 iterations of the OHBV model (between iteration 140-150), and then nearly flattening for the rest of the runs. (At iteration 150 the GA method produces a median feedback value of 0.905, up from 0.76 and only increasing to 0.912 at its final iteration.)

These differences are subtle, but we believe our results show that they may be the results of actual differences in how the methods calibrate the parameter sets rather than an artifact of the sample size because the differences are consistent throughout the entire calibration process, and because the MCS showed little difference between its results. (Figures 29 and 27). These same differences do appear in the SCE-UA and GA method data end points, but here they are too small for us to be able to confidently make any conclusions about them.

We feel that the CALA method has too big a variance in feedback values in both situations for us to make any conclusions regarding it.

4.1.1 Further Work

From our conclusion and experiences in working with this project, we would suggest the following future work:
Compare GA, SCE-UA, Monte Carlo scheme with a larger sample size and different pregenerated values By increasing the sample size, whether the GA or SCE-UA methods are differently affected by comparing modelled outputs to observed or pregenerated values or whether the difference we discuss in our results are an artifact of our sample size or the pregenerated values themselves.

Configure the CALA to run a tier type configuration We believe that the reason that CALA did so poorly is because of the different ”weigth” by each parameter (weight meaning how large of an impact the parameter has on the outcome). By analyzing and grouping the parameters by ”weight” and let the CALA configure them one tier at a time, starting with the heaviest ones. This method may not speed up the CALA, but it might clear up the results and get a higher feedback, and will give an indication of how to use CALA similar enviroments.

5 Bibliography

References

[1] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Number 0195099710. Oxford University Press, 1996.

[2] William Cook. Traveling salesman problem. http://www.tsp.gatech.edu/.

[3] William Cook. Traveling salesman problem explenation. http://www.tsp.gatech.edu/problem/index.html.

[4] Charles Darwin. On the origin of species. http://darwin-online.org.uk/EditorialIntroductions/Freeman_OntheOriginofSpecies.html.

[5] Charles Darwin. On the origin of species, survival of the fittest. http://darwin-online.org.uk/content/frameset?viewtype=side&itemID=CUL-DAR121.-&pageseq=238.

[6] J.G. Eckhardt, K.; Arnold. Automatic calibration of a distributed catchment model. *Journal of hydrology*, 2001.

[7] Abdel-Rahman Hedar. Shubert function. http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page1882.htm.

[8] Neuro Dimension INC. Mutation. http://www.nd.com/products/genetic/mutation.htm.

[9] Ingjerd; Langsholt Elin Lawrence, Deborah; Haddeland. Calibration of hbv hydrological models using pest parameter estimation. Technical Report 978-82-410-0680-7, Norwegian Water Resources and Energy Directorate, 2009.

[10] Wen-Jing. Liu, Yang; Ye. Time consuming numerical model calibration using genetic algorithm (ga), 1-nearest neighbor (1nn) classifier and principal component analysis (pca). Technical report, Department of Engineering, Exeter University, 2005.

[11] H Madsen. Automatic calibration of a conceptual rainfall-runoff. *Journal of hydrology*, 2000.

[12] John H. Holland Melanie Mitchell, Stephanie Forrest. The royal road for genetic algorithms: Fitness landscapes and ga performance. Technical report, University of Michigan, University of New Mexico„ 1992.

[13] Melanie Mitchell. *An introduction to genetic algorithms*. Number 0262631857. MIT Press, 1998.

[14] Nelder and Mead. A simplex method for function optimization. ..., 1965.

[15] Nils Roar Sælthun Ånund Killingtveit. *Hydrology*. Number 82-7598-026-7. Vol 7 of Hydropower Development; Norwegian Institute of Technology Division of Hydraulic Engineering, 1995.

[16] Hartmut Pohlheim. Evolutionary algorithms selection. http://www.geatbx.com/docu/algindex-02.html#P503_25798.

[17] Powel. Distributer of the software utilising the sce-au algorithm. http://www.powel.no/.

[18] Duan; Soroosh Sorooshian; Vijai K. Gupta Quingyun. Optimal use of the sce-ua global optimization method for calibrating watershed models. *Department of Hydrology and Water resources*, 1994.

[19] M.A.L. Thathachar; P.S. Sastry. *Networks of Learning Automata, Technique for online Stochastic Optimization*. Number 1-4020-7691-6. Kluwer Academic Publishers, 2004.

[20] SMIH. Welcome to the homepage of the original hbv-model. http://www.smhi.se/foretag/m/hbv_demo/html/welcome.html.

6 Appendix

Here we have all the extra information that we felt didn't fit in the main report

Figures

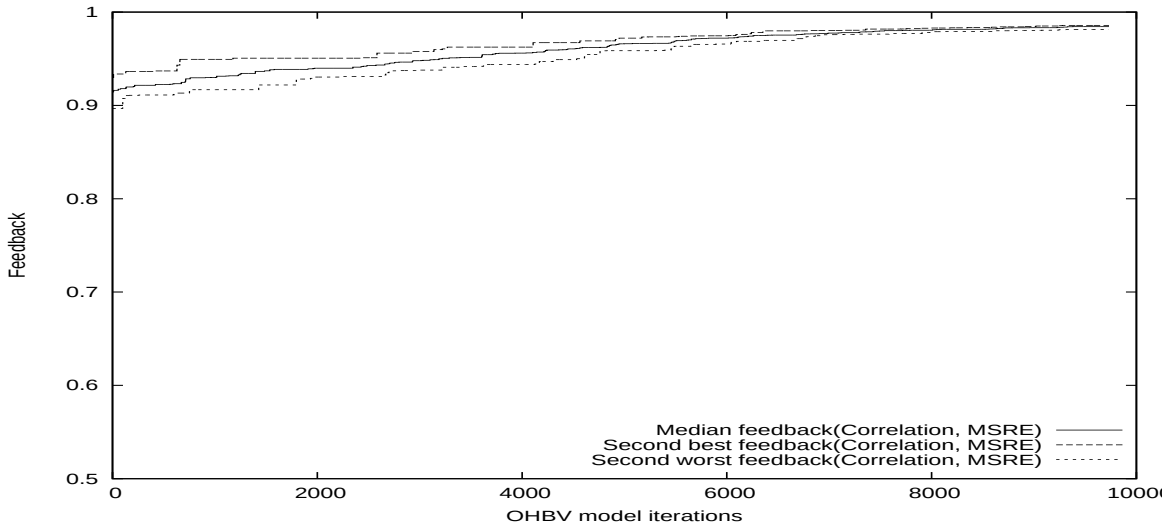


Figure 43: The graph shows the evaluation of running the SCE-UA auto calibration agains a pregenerated model output with a reflection constant of 0.5 and 8 complexes of size 73.

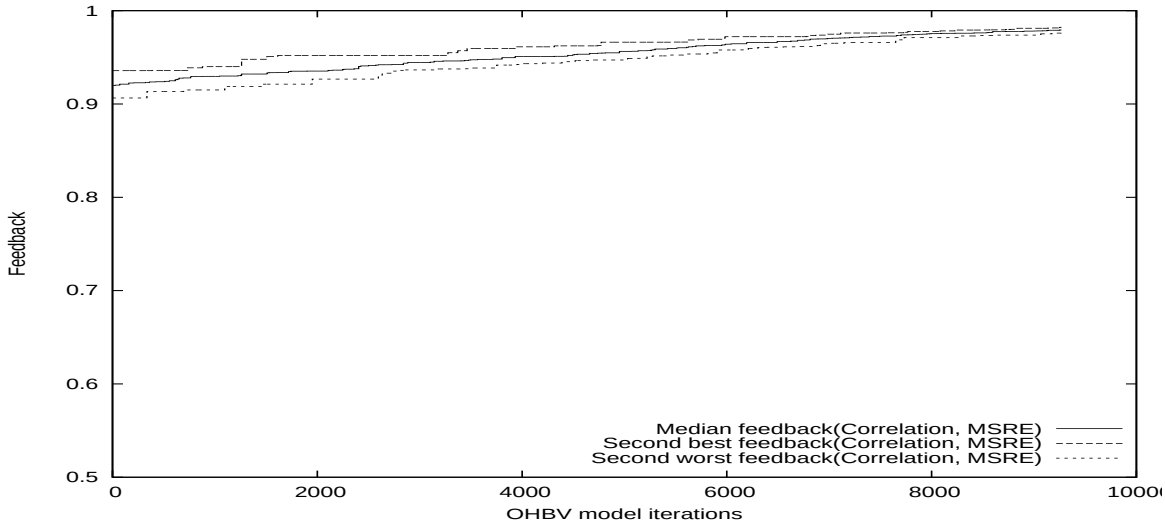


Figure 44: The graph shows the evaluation of running the SCE-UA auto calibration agains a pregenerated model output with a reflection constant of 0.5 and 10 complexes of size 10.

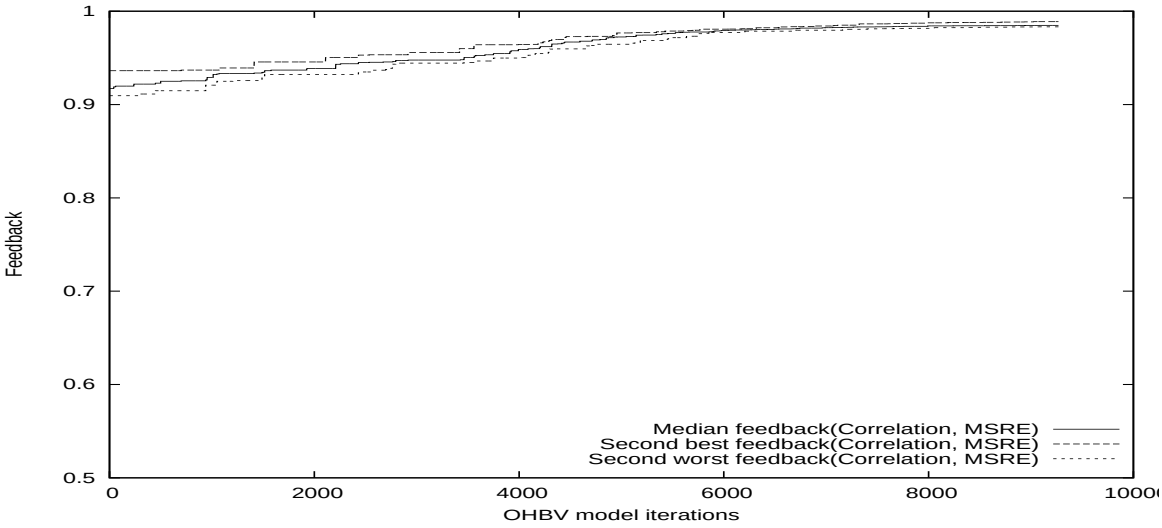


Figure 45: The graph shows the evaluation of running the SCE-UA auto calibration agains a pregenerated model output with a reflection constant of 0.5 and 10 complexes of size 10, here reducing the number of complexes based on the criterion value of the best complex.

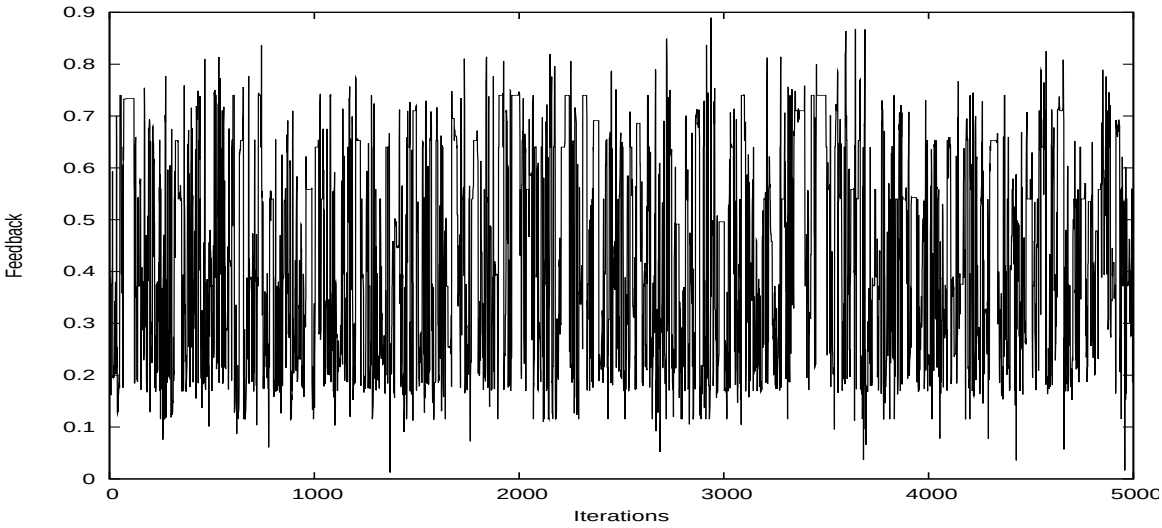


Figure 46: The plot shows the feedback we got when using all the parameters

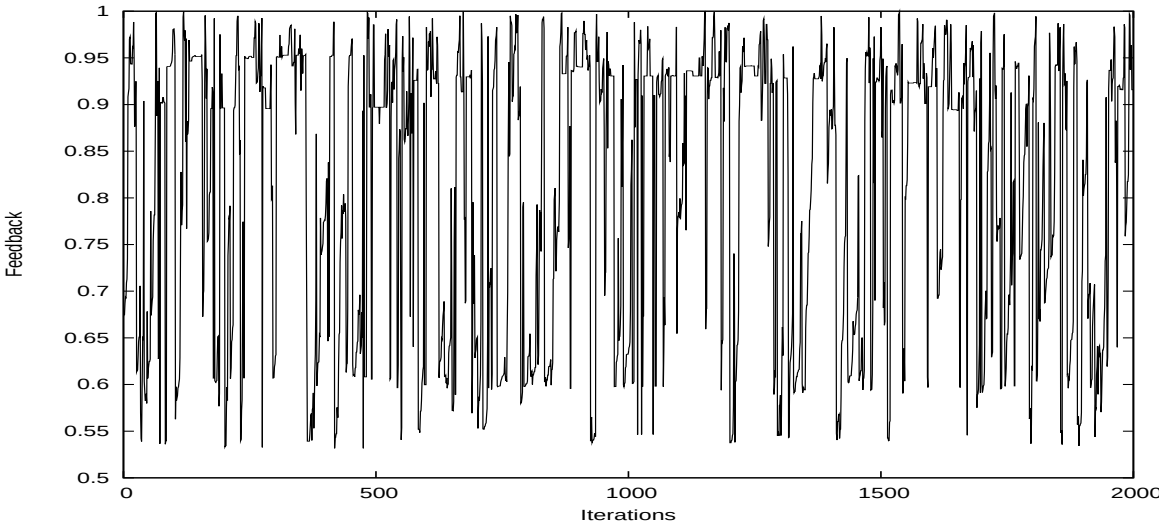


Figure 47: The plot shows the feedback when only using Bias

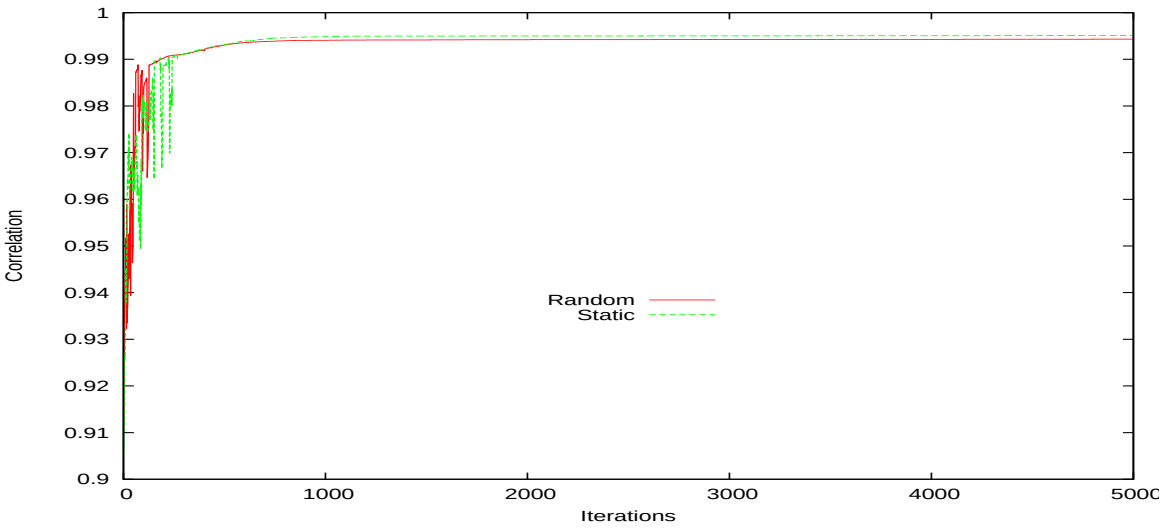


Figure 48: CALA Configuration test, configuration is $\sigma - 05\ \lambda$-03

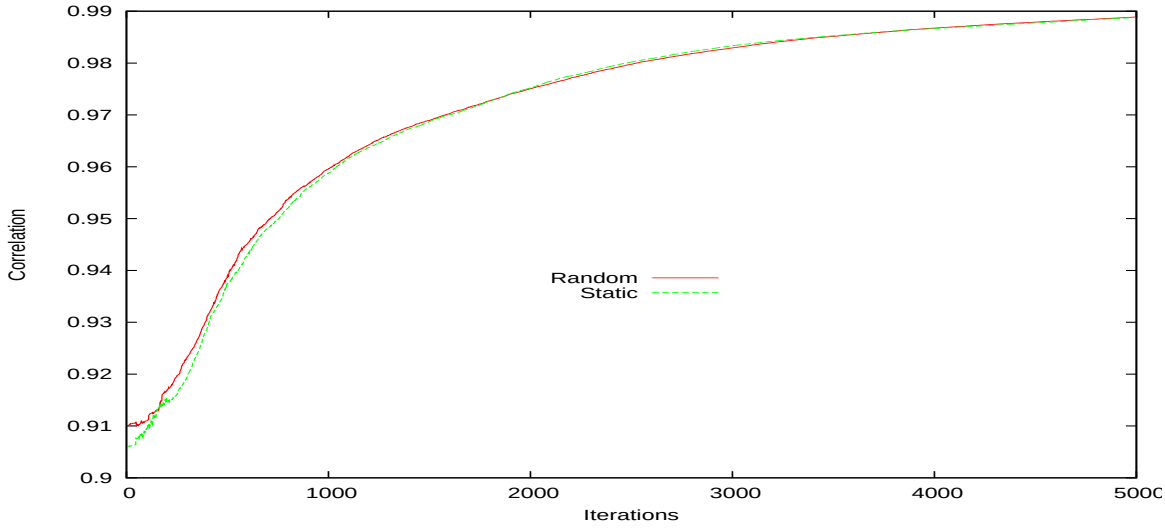


Figure 49: CALA Configuration test, configuration is $\sigma - 05$ λ-Dynamic

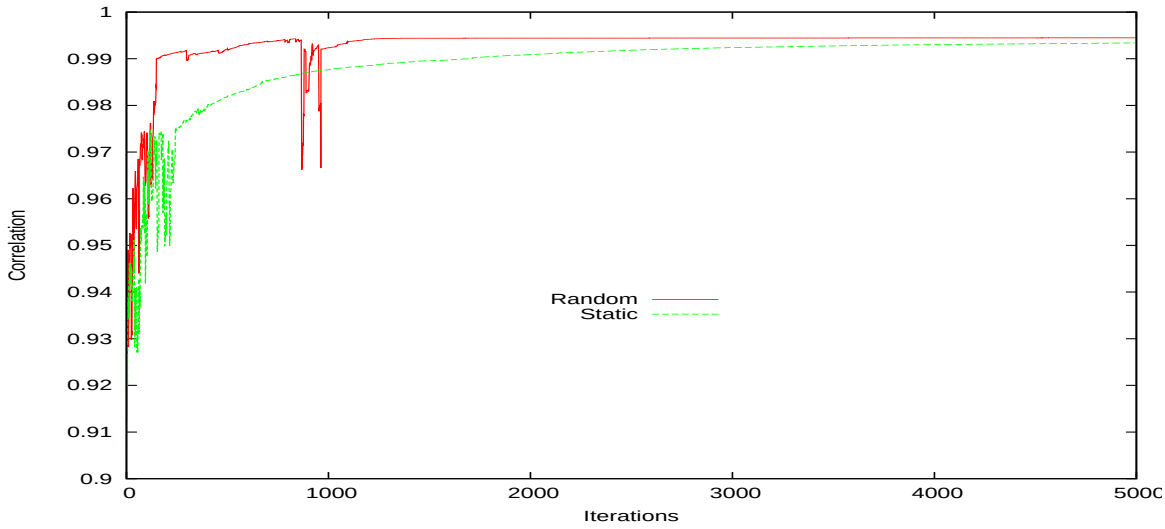


Figure 50: CALA Configuration test, configuration is σ-Dynamic λ-03

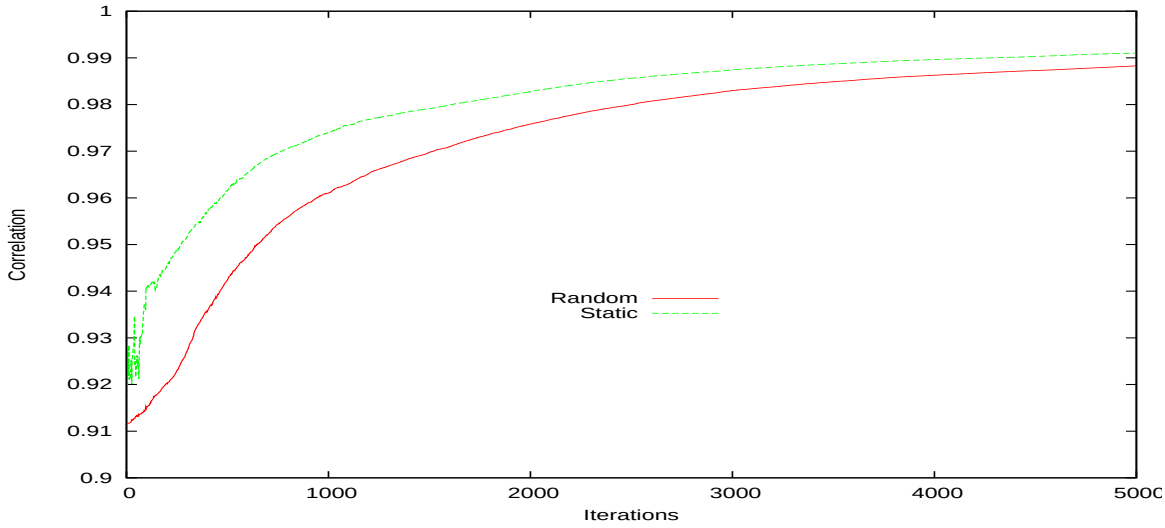


Figure 51: CALA Configuration test, configuration is σ-Dynamic λ-Dynamic

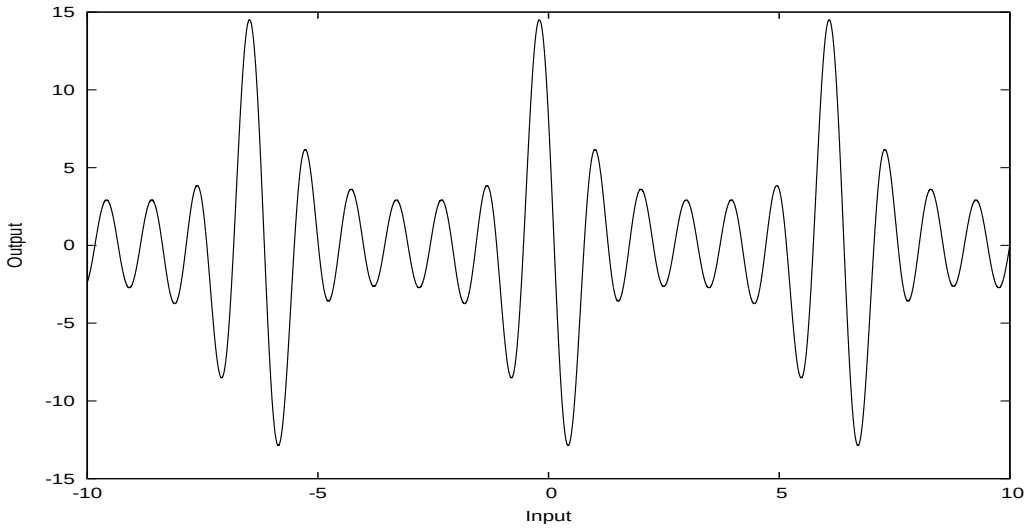


Figure 52: Illustration of the Shubert function between the values -10 and 10