POLITECNICO DI MILANO

Natural Resources Management

Matlab project - Lake Como

REPORT

Prof Castelletti Andrea Franceso
Prof. Giuliani Matteo
Prof. Sangiorgio Matteo

Carta Flavio
Penasa Carlo Andrea
Peverali Alessandro

Academic year 2022/2023

Contents

1 Matlab project - Part one

The first part of this report describes a forecast model of the cumulative 3-day inflow to Lake Como. The model is inspired by the piecewise linear model: the 11 years sample dataset is divided in two parts using k-means clustering and two ARX models are than calibrated, one for each dataset.

The idea is to capture the non-linearities of reality within the k-means clustering partition.
From a physical point of view reality drastically changes its behavior in rainy situations (called in this report "wet conditions") as opposed to drought conditions ("dry conditions"). For example, in rainy situations the ground is saturated with water and therefore it can no longer drain water efficiently, this has an impact on the amount of water that reaches lake Como.

Following below are all the main steps taken during the implementation of the model, each describing carefully all the modeling choices, followed by a short discussion on the implementation of the code.

Please reference the code "script part 1" if interested in any detail of the code.

1.1 Dataset management

Date	Input	Output
1	u(1)	CumInf(2+3+4)
2	u(2)	CumInf(3+4+5)
3	u(3)	CumInf(4+5+6)
4	u(4)	CumInf(5+6+7)
5	u(5)	CumInf(6+7+8)
6	u(6)	CumInf(7+8+9)
7	u(7)	CumInf(8+9+10)
⋮	⋮	⋮
⋮	⋮	⋮
⋮	⋮	⋮
⋮	⋮	⋮

(a)

Date	Input	Output
1	u(1)	
2	u(2)	
3	u(3)	
4	u(4)	CumInf(2+3+4)
5	u(5)	CumInf(3+4+5)
6	u(6)	CumInf(4+5+6)
7	u(7)	CumInf(5+6+7)
⋮	⋮	CumInf(6+7+8)
⋮	⋮	CumInf(7+8+9)
⋮	⋮	CumInf(8+9+10)
⋮	⋮	⋮

(b)

Date	Input	Output
1	u(1)	mean(Output(4:8))
2	u(2)	mean(Output(4:8))
3	u(3)	mean(Output(4:8))
4	u(4)	CumInf(2+3+4)
5	u(5)	CumInf(3+4+5)
6	u(6)	CumInf(4+5+6)
7	u(7)	CumInf(5+6+7)
⋮	⋮	CumInf(6+7+8)
⋮	⋮	CumInf(7+8+9)
⋮	⋮	CumInf(8+9+10)
⋮	⋮	⋮

(c)

Figure 1

The original dataset covers 11 years, from 1 Jen 2009 to 31 Dec 2019. It's composed of 25 inputs and one output which is the cumulative inflow of 3 days ahead (e.g. the output of 04/01 is given by the sum of the inflow inputs from the 05/01 to 07/01) (1a).

The structure of this dataset has been changed, shifting down the output column in order to have as output the cumulative inflow inputs of today and the two days before[1] (e.g., the output of 04/01 is now given by the sum of the inflow inputs from the 02/01 to 04/01) (1b).

It's possible to see that with the provided dataset the output of each row (the value to be forecasted) is a value that is not known in that time instant.
The purpose of the newly structured dataset is to have in every row information that is totally available. The main reason for this shifting is because the output values are crucial to perform K-means clustering (see cap 1.5), splitting the dataset in two sub-datasets (wet and dry). This new shape of the dataset prevents the risk of using not yet available data.
This has no impact on the forecasting of the cumulative 3 days ahead, because the predictor of the model has a lead time of 3 days, this means that on the 04/01 the model will predict the output (the

[1] This new output has been called improperly "cumulative inflow of 3 days before" for semplicity.

cum inflow of 3 days before) of the 07/01, that is exactly the cumulative inflow of 3 days ahead of the provided dataset.
Since this shift leaves behind 3 rows of empty outputs, it's been decided to fill them with the mean of the first 5 available output values (mean(Output(4:8))) (1c).

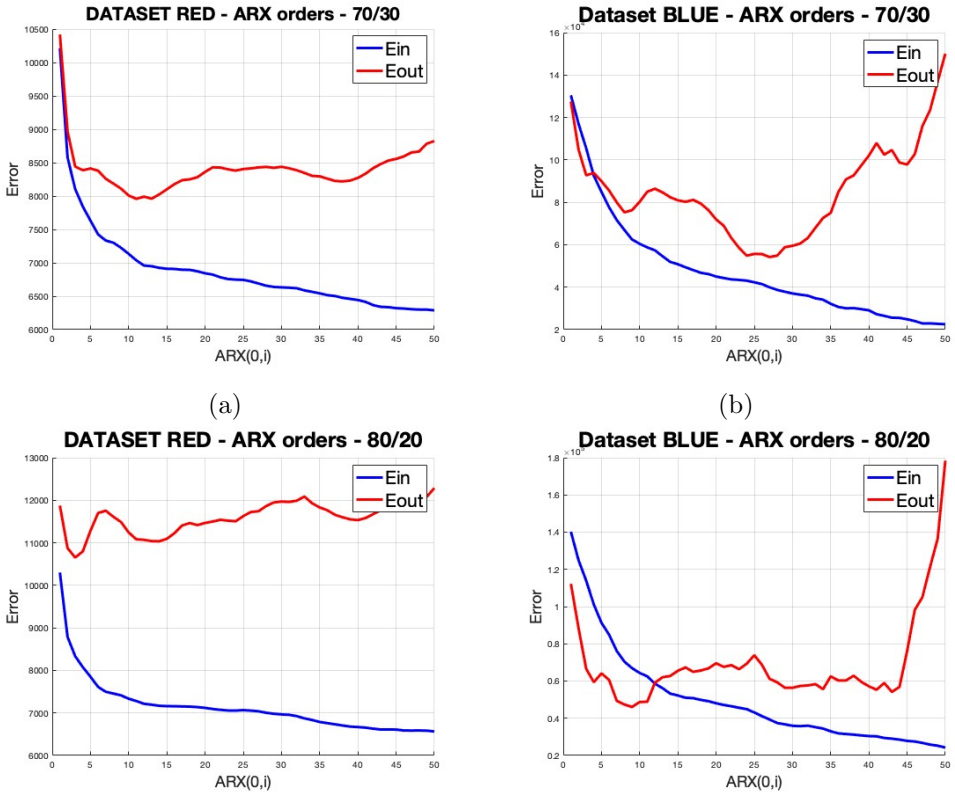
It's important to note that this substitution is used only to preserve the original size of the output array. These values are not used in the forecasting process by the ARX models, nor are they considered in the computation of the performances indicators.

1.2 Training-validation partition

The training-validation partition has been computed experimentally, where the first 70% of the dataset is used for training and the last 30% for validation. It is common for theoretical studies to have a smaller validation dataset: a general rule of thumb is to use 1/5 of the size of the dataset.

In this project a bigger validation dataset is used, as it seems to provide more robustness (smaller variance of Eout) and a more accurate out of sample error (smaller bias in respect to the actual value).

The continuous interruption of the temporal sequence of the data created by the K-means partitioning may increase disturbances inside the datasets, thus requiring a bigger validation dataset to better approximate the real Eout.
This intuition is supported by the fact that looking at the plots of Eout and Ein the ones obtained from an 70/30 separation of the dataset (a and b) shows that the two errors at the beginning (where there is not overfitting) are closer to each other than the case with 80/20 separation (c and d).



(a) (b)

(c) (d)

Figure 2

A further confirmation of this idea is given by the fact that when the performances of the ARX models are assessed with a training-validation-testing partition (respectively 0,9*0,7 / 0,9*0,3 / 0,10), the E_out (and therefore r^2) found in validation and testing are more similar to each other rather than when computed starting from a partition with smaller validation (0,9*0,8 / 0,9*0,2 / 0,1).

1.3 Data pre-processing

First of all a NAN analysis has been performed: all the NAN values have been replaced with the previous value of the column. The reason for it is that a substitution with the mean can disturb the calibration of the model (as it can be far away from the temporal trend of the current situation), while the input value from one day to the next one has a more contained effect.

```
%% NAN Analysis
dim_ptot = size(p_all);
nan_idx=isnan(p_all);
nan_col=sum(nan_idx);
nan_row=sum(nan_idx, 2);
tot_nan=sum(nan_row);

T = 365;
% Substitute Nan value with the previous value of the column
% m_ptot = mean(p_all,1,'omitnan');
for i = 1:dim_ptot(1) % rows
    for h = 1:dim_ptot(2) % colomns
        if nan_idx(i,h) == 1
            p_all(i,h) = p_all(i-1,h); %m_ptot(h);
        end
    end
end
```

Figure 3

As already seen in the laboratory, the inputs and the outputs are used with the moving average function (fixed windows t=21 days) to compute the their cyclostationary mean and variance in order to be standardized (inputs and output mean=0 and variance=1).

These deseasonalized variables are called “u” for the inputs and “x” for the output. In this way the conditions for a stationary stochastic process are guaranteed, and therefore ARX models can be used.

1.4 Input analysis

An analysis on the inputs has been performed to understand the available information, to see which among all the inputs are the most significant ones.
Recalling sensitivity analysis theory, the significance of inputs has been assessed with these criteria:

- Strong correlation between the input and the output
- Weak correlation between the input and other significant ones.

The first criterion makes sure that the significant input contains valuable information that can be exploited by the ARX model predictors, and the second one assures that there is not redundant information, that would only increase the computational load and potentially the noise/uncertainties.

This type of analysis led to the following default selection of inputs (figure 4).

The project script is written to allow the user the possibility to change the selection of the inputs that he considers significative just indicating them in the corresponding matrix.

```
%  ------------------------------------------------
%%   Train the linear model: K-means clustering
%  ------------------------------------------------

% ____________________________________________________
% OPERATOR SELECTION
% Selection of the input used for ARX
sel_input_red = [1 9 13 16 22];        % dry % PCA 1 2
sel_input_blue = [1 6 8 13 16 22];     % wet % PCA 1 2
k_input = [0]; % 0 refers to the output while 1-25 the corresponding inputs
```

Figure 4

1.5 K-means clustering

K-means clustering is implemented by the K-means function kmeans(X,k), that takes as input a matrix X of the values that have to be partitioned into K clusters. The X matrix is built using the variables selected by the user.

This selection shows the strategy adopted by the operator to capture the different conditions, that are related to the non-linearities of reality, as already described in the introduction.
The variable selected is the cumulative inflow, that appears to be the one that better captures the average conditions of these two very distinctive behaviors (wet/dry conditions).

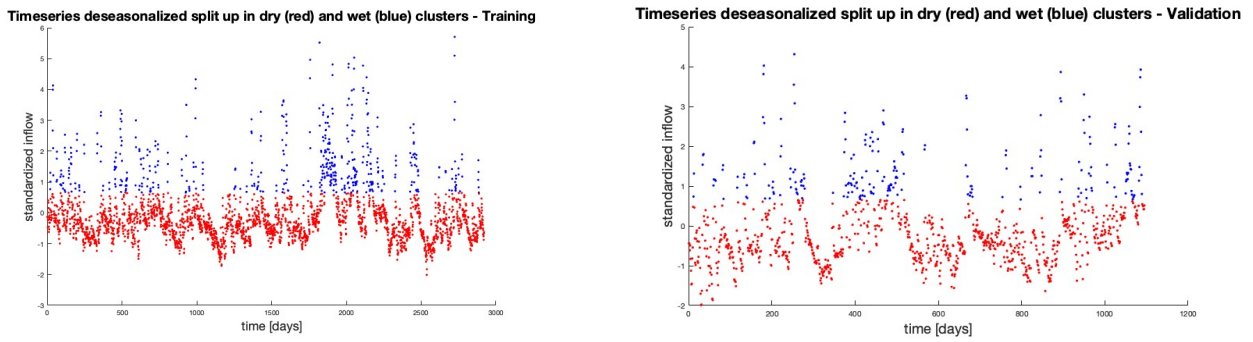
```
% Selection of the variables used for K-means. Training and Validation must be
% the same
k_sel_train = [];
k_sel_test = [];

for i = k_input
    if i == 0
        k_sel_train = [k_sel_train x];
        k_sel_test = [k_sel_test x_test];
    end
    if not(i==0)
        k_sel_train = [k_sel_train u(:,i)];
        k_sel_test = [k_sel_test u_test(:,i)];
    end
end
```

Figure 5

In the script, after the operator has indicated the variables in k_input, the corresponding values are saved in k_sel_train and k_sel_test matrixes.
During training the K-means functions returns the values of the two centroids, they are reordered in ascending order to have the first one related to the dry condition and the second one to the wet condition.
Then according to the value of the index variable idx (that tells if a particular day belongs to the wet or dry training dataset) the training dataset is partitioned in Cr (red) and Cb (blue) containing the input values and Dr and Db with the output variables.

In validation, where forecasting really happens, the dataset is partitioned by looking at the current day cumulative inflow value (that considers the values from that day to two days before) and checks to which centroid it's closer. If it's the first one, that day belongs to the dry dataset so its inputs (the chosen ones to be plugged into the ARX model) are saved into Crtest and the output to Drtest, while if it's the second one the inputs and outputs are stored into Cbtest and Dbtest.



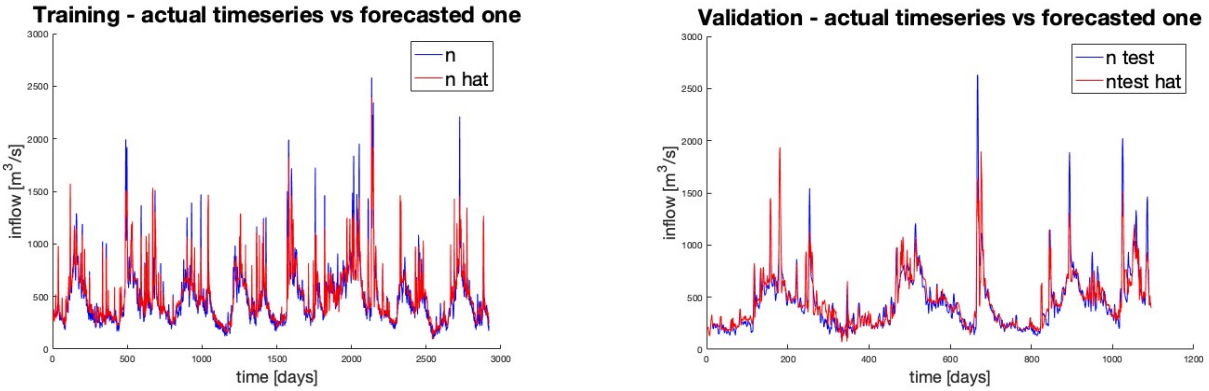
(a) (b)

Figure 6: Plot showing the separation into the clusters

1.6 Recursive ARX models

The ARX model used is a linear model, implemented as the linear combinations of the inputs considering only the exogenous part, ARX(0,...).
Once the 2 sub-datasets are defined, a code is written in order to find the best two ARX models, one for each sub-dataset.

With the optimal models found, the performances are assessed with the r^2 values (figure 2 a and b) and the comparison between the forecasted outputs time series and the actual one for both training and validation (figure 7 a and b):



(a) (b)

Figure 7

Going into further details (figure 8), only the red (dry) case will be taken into consideration. The same procedure has been done for the blue case too.
It's called k the lead time between the present time and the forecasted one (fixed to k=3 by the project requirements), and ARXmax the maximum order of ARX models tested.

After an initialization, the "for" cycle runs the value of q (current order of the ARX model[2]) from 1 to ARXmax. Here a function ARX is recalled and takes as inputs q, the lead time k, the input Cr and the output Dr matrixes of the red training sub-dataset.
The function ARX returns as outputs the matrix M of the exogenous part, the matrix beta of the

[2]to be precise by theory the order of this ARX model should be q+k for the exogenous part, but for semplicity it'll be called q, as only q is used as variable parameter. The ARX(0,1) to ARX(0,k) would be improper models for a k days ahead predictor, therefrore what's called here ARX(1) would be in fact an ARX(0,k+1).

parameters and the first q+k values from the output dataset (values that cannot be forecasted, only used to keep coherence between matrixes dimensions. This values are not considered into the error computation).

Once the parameters matrix is found (using the ARX function with the training set), it's used to forecast both the (red) time series for training and validation, in order to compare them with the actual corresponding values of the (red) output dataset. This comparison leads to the calculation of the r^2 values, of E_in and E_out.
It's important to stress the fact that this performance assessments do not consider any of the values Yin directly taken from the ouput dataset, but only the result of forecasting.

This whole process happens for each iteration (order) of the "for" cycle discussed before (this means that every iteration produces a new parameter matrix. The corresponding values of r^2, E_in and E_out are saved into their row of the arrays R2_i_red, Ein_i_red and Eout_i_red).

The arrays Ein_i_red and Eout_i_red are plotted in figure 2 a and b. The order corresponding to minumin value of Eout_i_red is the optimal one for the red ARX model.

```
%% Recursive ARX - RED
k=3; % This is the lead time between the actual istant time and the predicted
one.
ARXmax = 50;    % maximum ARX order tested

% Initialization
R2_i_red = zeros(ARXmax,2); % first column --> Ein ; second column --> Eout;
Ein_i_red = zeros(ARXmax,1);
Eout_i_red = zeros(ARXmax,1);
% q=10;

for q = 1:1:ARXmax % i is the actual ARX model

    % training
    [Mprova_red,Yin,betaprova_red] = ARX(q,k,Dr,Cr);
    Y_hatprova_red = [Yin;Mprova_red*betaprova_red];
    n_hatprova_red = (Y_hatprova_red.*sigma(indicer))+m(indicer);
    R2_i_red(q,1)=1-sum((n(indicer(length(Yin)+1:end))-
n_hatprova_red(length(Yin)+1:end)).^2)/sum((n(indicer(length(Yin)+1:end))-
m(indicer(length(Yin)+1:end))).^2);
    Ein_i_red(q) = sum((n(indicer(length(Yin)+1:end))-
n_hatprova_red(length(Yin)+1:end)).^2)/length(n(indicer(length(Yin)+1:end)));
    clear Yin

    % validation
     [Mtestprova_red,Yin,betatestprova_red] = ARX(q,k,Drtest,Crtest);
     Ytest_hatprova_red = [Yin;Mtestprova_red*betaprova_red];

ntest_hatprova_red=(Ytest_hatprova_red.*sigma_test(indicetestr))+m_test(indicete
str);
     R2_i_red(q,2)=1-sum((n_test(indicetestr(length(Yin)+1:end))-
ntest_hatprova_red(length(Yin)+1:end)).^2)/sum((n_test(indicetestr(length(Yin)+1
:end))-m_test(indicetestr(length(Yin)+1:end))).^2);
     Eout_i_red(q) = sum((n_test(indicetestr(length(Yin)+1:end))-
ntest_hatprova_red(length(Yin)+1:end)).^2)/length(n_test(indicetestr(length(Yin)
+1:end)));
     clear Yin
     disp('ARX '),disp(q);
end
```

(a)

```
%% CHOICE OF BEST ARX MODELS
% RED
for i = 1:length(Eout_i_red)
    if Eout_i_red(i)==min(Eout_i_red(k+1:end))
        disp('BEST RED ARX:'),disp(i)
        Best_red_arx = i;
    end
end
```

(b)

Figure 8: (a) It's possible to see the code referred to the red (dry) sub-dataset, the recursive 'for' loop in which each order of the ARX is tuned in training and tested in validation.
(b) Code used to identify the best order of the ARX

ARX function

Following there is the description of the ARX function. The code is written to make it as flexible as possible, it considers the dimension of the input matrix in order to understand how many inputs are selected by the user, as discussed in cap. 4. It's also possible to change the value of k to forecast with different lead times.
(the code presented during the contest phase considered also the autoregressive part, that now for simplicity has been removed as there is not really an advantage in performances).
The code is shown in figure 9.

U initialization

There is a first initialization phase where the U matrix (input matrix) is computed to be used afterwards in the iteration cycle. The reason is to fix before its dimension to avoid an iterative reallocation of the memory used for every iteration of the while cycle (see U computation). This dimension is given by a number of columns equal to the quantity of chosen inputs times the order q of the model, and a number of rows equal to the number of predictions, this number is computed considering, first of all that the fundamental ARX equation is the following:

$$\hat{y}(t+k) = \alpha u(t) + \beta u(t-1) + \cdots + \theta u(t-q)$$

where y, the output (the prediction) at the instant t+k, is equal to a linear combination of the inputs from time t up to time t-q
For each row an iteration of this equation is performed. Considering that at the first iteration t, the "oldest" input sample is u(t-q), and therefore t=q.
During the last iteration the most recent input sample is u(end-k), the one corresponding to the last forecasted value (where end=number of input samples).
With this reasoning, the number of rows is given by the number of input samples – k – q + 1.

Time instants						
t=q	u(q)	u(q-1)	⋯	u(1)	y(q+k)
t=q+1	u(q+1)	u(q)	⋯	u(2)	y(q+k+1)
⋮	⋮	⋮	⋱	⋮	⋮
t=end-k	u(end-k)	u(end-k-1)	⋯	u(end-k-q+1)	y(end)

U computation

In this stage a while cycle is used to fill up the U matrix. For every iteration of the "while" cycle (as many as the model's order) it fills the U matrix's "block of columns" corresponding to the current iteration (e.g. if there are 3 inputs and the model's order is 10, there will be 30 columns in the U matrix, and each iteration will fill up a 3 columns block).

Beta tuning

The last part of the code deals with the computation of the coefficent matrix of the ARX's inputs linear combination and the initial values of the output.

```
function [M,Yin,beta] = ARX(q,k,x,u)

% q is the order of the exogenous part
% k is the lead time
% x is the output dataset (training)
% u is the input dataset (training)

% M is the [U] matrix, Y_hat = M*beta
% Yin are the initial values of Y matrix
% beta is the parameter matrix (which has been tuned)

dim_u = size(u);

% U Initialization
lq = q;
j = 0;
U = zeros(dim_u(1)-k-q+1,dim_u(2));
U = repmat(U,1,q);

%  U Computation
while (lq > 0)
    U(:,1+dim_u(2)*j:dim_u(2)*(j+1)) = [u(lq:end-k-j,:)];
    lq = lq-1;
    j = j+1;
end
clear lq

% beta tuning
M = [U];
Y = [x(q+k:end)];
beta = M\Y;
Yin = [x(1:q+k-1)];
```

Figure 9

1.7 Reconstruction of the time series and results

In the end, a temporal reconstruction of the forecasted samples is necessary in order to obtain the final timeseries.

The indexes functions idx and idxtest are used to see if the value corresponding to a given day has to be taken from the dry (red) or wet (blue) dataset (figure 10) .

One value at a time the forecasted timeseries are built (plotted in figure 7 for training and validation respectively) and in figure 11 for the overall dataset.
In figure 11 there is the plot of the forecasted timeseries of the whole dataset

```
%% RED + BLUE reconstruction

% Training
jr = 1;
jb = 1;
for i = 1:1:length(idx)
    if idx(i)==1
        n_hat(i,1) = n_hat_red(jr);
        jr = jr+1;
        a(i) = 1; % to check if all values are 1
    end
    if idx(i)==2
        n_hat(i,1) = n_hat_blue(jb);
        jb = jb+1;
        a(i) = 1;
    end
end
clear i, clear jr, clear jb

for i = 1:1:length(a)
    if not(a(i)==1)
        disp('A problem in reconstructing training set has occured!')
    end
end
clear i, clear a
```

Figure 10: Script of the dry timeseries reconstruction, only for training

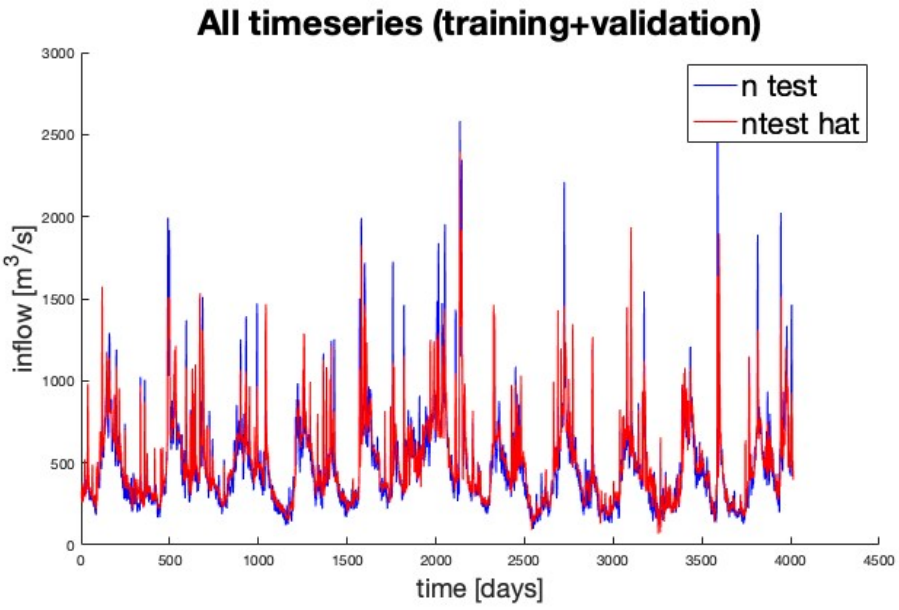


Figure 11

1.8 Conclusions

	Dry dataset	Wet dataset
Arx order	11	27
E_in	6989,9	3993,2
E_out	7957,7	5420,5

(a)

	Dry training	Dry validation	Wet training	Wet validation	Final training	Final validation
r^2	0,63	0,62	0,84	0,71	0,78	0,71

(b)

Figure 12

Looking at the results (figure 12), the described forecasting model appears to have satisfying performances. A comparison with other types of models implemented beforehand (such as Neural Networks or ARXs without K-means clustering) shows a r^2 value significantly higher (r2=0,71 against $r2 < 0,5$).

Other important remarks can be drawn by analyzing the correlograms between the residual (error) and the predictions, to evaluate the amount of information that the model has not been able to use in the forecasting process.

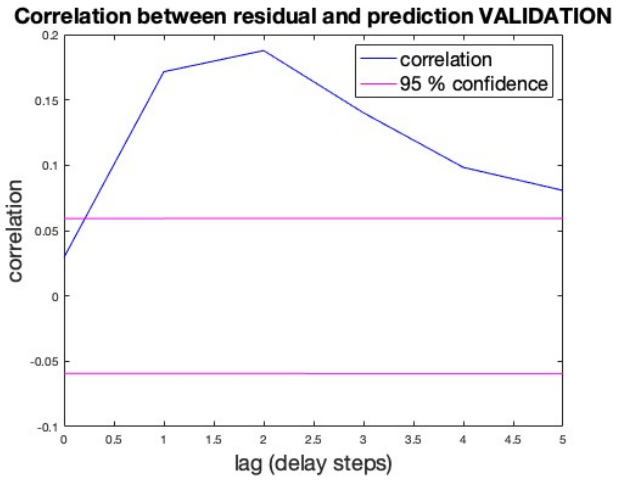


Figure 13

A perfectly flat correlogram would mean that the error is white noise, meaning that all the information has been extracted from the data. It's possible to see that there is some room for improvement, for example by using ARMA processes and by implementing cross-validation techniques.

Furthermore ARX models are based on a minimization of the Mean Square Errror, that implicitly favours parametrizations that perform better on high values of the output rather than low values. This could explain why the results from the wet dataset are better than the ones from the red dataset, so a model calibration based on the minimization of different cost functions for the red dataset could have a positive effect on the overall result.

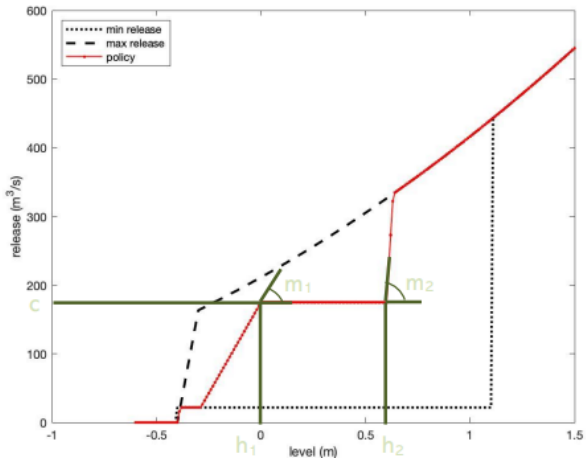
2 Matlab Project - Part two

The second part of the project was a study about water resources allocation in lake Como.
First of all the actual operating policy (AOP) has been addressed, understanding the reasoning behind it and simulating the system evolution under said policy, starting from a given input dataset.

Some parameters that describe the performances of the policy have been defined and calculated for the AOP, in order to have a starting point to improve from.
From there, an algorithm has been implemented to create a better policy, comparing in the end the various solutions obtained from the exploration of the Pareto front.

2.1 Actual operating policy (AOP)

The AOP and its defining parameters were part of the data provided at the beginning of the analysis (a and b):



(a)

Parameter	Description	Value	Units
h_1	Low reservoir level	0	[m]
h_2	High reservoir level	0.6	[m]
c	Constant release	175	[m³/s]
m_1	Slope for low level	530	[m²/s]
m_2	Slope for high level	4900	[m²/s]

(b)

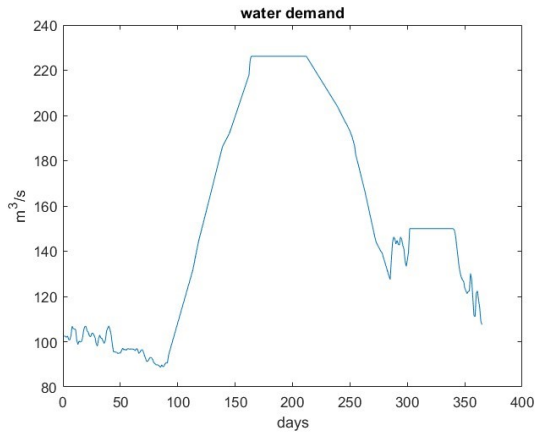
Figure 14

Figure 14a shows in red the function that binds the level of the lake with the corresponding cubic meters per second of water release. The policy is inherently limited by a maximum release flow (given by the maximum possible flow of the lake itself without constraints).

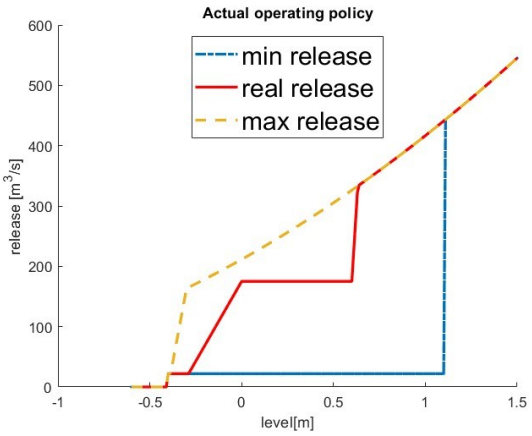
The State provides guidelines for minimum release to guarantee a sufficient amount of water for the downstream ecosystems to survive.
It also defines a lower and upper threshold for the level of the lake that force all the dams to be closed or opened (depending on whether the level is getting critically low or there's the risk of floods, respectively).

With the constraints as described, there is a large "central" portion of the possible levels for which the decision about the release flow is not forced by law, but can be set in order to fulfill at best the interests of the stakeholders involved with the lake's operations.

In this AOP this "free" region is regulated with a piece-wise linear function, consisting in 3 lines defined by the 5 parameters (figure 14b) corresponding to the position and slope of the segments (with the slope of the central one set to zero). An operating policy designed like that has the benefits of being easy to implement, effective in preventing floods and dry periods and having that constant central release flow that can be set equal to the water demand.



(a)



(b)

Figure 15

Although effective, this policy could be not the optimal one if the water demand through the year is not constant as is the case for lake Como, as seen in figure 15a.

To find better solutions, the slope of the central part of new operating policies will not be always null (constant release), but will become a sixth parameter that may change to improve the performance of the policy.

Before implementing this, it was requested to replicate the AOP (figure 15b) as a starting point to analyze its performances.

2.2 Simulations under AOP

During this simulation, the system of Lake Como is treated as a standard reservoir system.
The provided data are the following:

- past 10 years daily lake inflow (figure 16a)
- algorithm for the simulation of the lake's behaviour.
- starting lake level (day 1)
- area of the surface of the lake, considered cylindrical for the purposes of this project (therefore the volume of the lake is given by this fixed area times its level)
- maximum possible output corresponding to a given lake level.

The algorithm divides the day in the 24 hours and performs a cycle in which for every hour starting from the level it simulates the lake volume, the maximum possible release and the optimal release following the AOP. The new level is computed and the cycle continues.

At the end of this procedure the function outputs the daily lake levels and releases (16b and 16c).

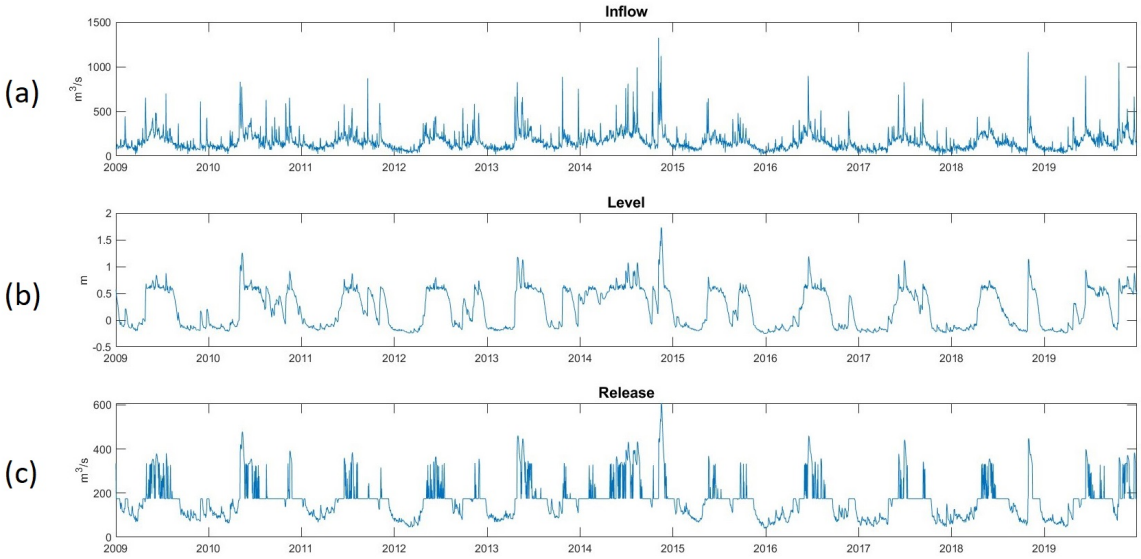


Figure 16

2.3 Performance indexes

In order to evaluate and compare different policies it's crucial to define some parameters that reflect the satisfaction of the stakeholders.

In the context of this project three types of stakeholders are considered:

- inhabitants of the areas near the lake, that will benefit from having as few floods as possible. The corresponding policy performance index "**flood**" is computed as the average yearly number of floods simulated with that policy.
  Although this index is easy to be computed and gives a clear indication of interests of the people surrounding the lake, it does not distinguish between the lake level going slightly above the flood threshold or completely flooding the area (in this way a policy that leads to very few disastrous floods will be preferred rather than one that leads to a multitude of insignificant ones).

  The index could be modified to address this problem, for example by defining it as the sum of the positive differences between the lake level and the flooding threshold, elevated by an appropriate coefficient.
  Albeit interesting, such modification has not been done in this project.

- farmers and hydropower plants, that use the release water for irrigation and power production. The corresponding policy performance index "**deficit**" is given by the average daily deficit (difference between water release and demand) over the simulated year.
  In the period from the 1st of April to the 10th of October the daily deficits are squared in order to highlight their dangerousness especially in the warmer period of the year.

- the environment itself, that would be damaged if the lake level ever goes below a certain threshold. The corresponding policy performance index "**low**" is average number of day when the level goes beneath such threshold over the simulated year.

The best policy will be the one that minimizes these three indexes.
At the end of the simulation done with AOP (see chapter 2.2), the corresponding performance indicators are the following:

flood	3,09
deficit	601,67
low	23,45

2.4 Optimizing the parameters

The problem of finding the best 6 parameters of the operating policy poses two crucial questions:

- which algorithm to use in oder to find the optimal policies that minimize the indexes (Pareto front)?
- how to decide which solution from the Pareto front is the best one?

2.4.1 NSGA-II algorithm

The possible combinations of policy parameters are infinite, so it would be impossible to simulate them all to find the ones that lead to the best performances.

The function that binds together the policy parameters and performance indexes is not easily identifiable, therefore an analytical solution would be very difficult to implement.

The chosen approach has been to perform a multi objective direct policy search using the NSGA-II evolutionary algorithm, with the objective to find the sets of policy parameters that minimize the indexes.

The inputs of the algorithm are: the number of different sets in every generation, the number of generations to simulate and most importantly the lower limit, upper limit and constraints of the parameters in order to produce realistic policies (e.g. a negative policy slope would not be considered realistic, as well as a perfectly vertical one).

The evolutionary algorithm starts from random sets of parameters, it simulates the performance indexes and evaluate the best ones.
New sets for the new generation are created by combining the parameters from the best ones of the previous generation and by randomly changing them (mutation).

The last generation is composed of the sets of policy parameters that better minimize the indexes. These solutions are form the Pareto front, all optimal.

2.4.2 Performance indexes comparison

Once the solutions of the Pareto front are obtained, they need to be sorted out in order to find the best one.
A valid strategy could to find the one closest to the origin (minimum), but since the indexes are very different between each others (both in units of measure and absolute values) and it's not known if one has priority over another, if done like this they would not be considered equally.
The one with the bigger absolute value (the "deficit" index) would dominate over the others in the minimization process (its minimization would be far more favoured than the minimization of the other two). To solve this problem a standardization is needed.

In order to standardize, after having performed the simulations each index is recalculated by subtracting from every value the minimum one and dividing by the delta between the maximum and minimum values simulated. After this process every index is between 0 and 1.

Since the three indexes are all to be minimized it was decided to compute the average of their absolute values. This would result in a single new index that will indicate directly the effectiveness of the policy (the smaller is the index, the better performing is the policy).

The average of the three new indexes between 0 and 1 is computed and the minimum one will correspond to the best policy (considering equally important the stakeholder's interests).

2.5 Final solutions

Once all the generations have been simulated, it's possible to plot all the corresponding performance indexes in a 3D space (figure 17, where on the three axis there are the 3 indexes "flood", "deficit" and "low".

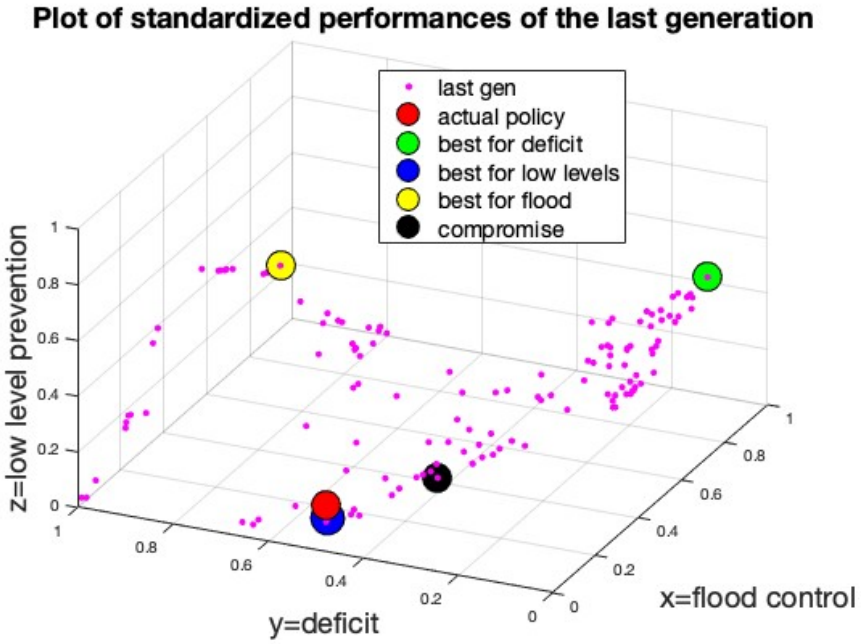


Figure 17

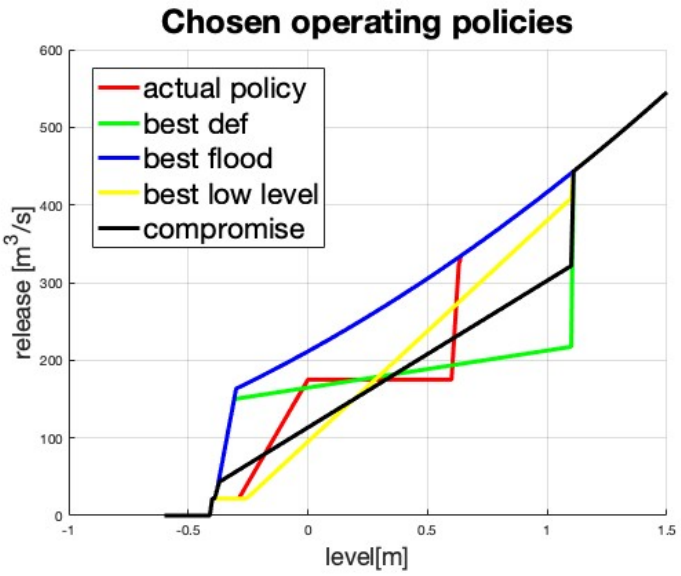
The ideal point would be the origin, corresponding to the perfect minimization of all three indexes. A perfect situation like this one may not be possible, so the best one will be that closer to the origin. The purple small dots correspond to the policies of the last generation.
The big red dot corresponds to the performances of the AOP.
The point closest to the origin (big black dot) will correspond to the policy with the smallest "unique performance index" (see subsection 2.4.2), which is the one that gives the best "compromise" option if the stakeholder's interests are considered equally.
There could be other points that better represent an ideal compromise in the real case, but without knowing the weights of the individual parameters they are not possible to be individuated.

Other three points can be highlighted (big blu, green and yellow dots), which are the ones corresponding to the maximum possible minimization of each of the performance indexes.

The collection of the blu small dots forms the Pareto front, that is the collection of the Pareto optimal solutions.

It's very likely that the found Pareto front is not the real one because of the finite number of generations simulated by the alghoritm.
It could be interesting to asses how close this Pareto front is to the real one by computing the distances from the points of one generation to the ones of the next one, to see in the end how much progress is left to be done.

In figure 18 it's possible to see the shape of the four new suggested policies, as well as the performance indexes of each of them.



(a)

	Def	Flood	Low
Actual	3,1	601,7	23,4
Best Deficit	1,1	625,5	212,5
Best Flood	14,7	305,1	137,5
Best Low	3,9	607,6	0
Compromise	9,1	507,2	26,3

(b)

Figure 18