Advanced Data Analysis and Machine Learning - Project Report

Mining Process (Level A) - Final Report

Shrestha, Prashant; Vahteristo, Ilmari; Vanegas, Sergio

December 19, 2023

1 Introduction

For our Advanced Data Analysis project, we were asked to develop a Dynamic soft sensor for Silica content in a flotation plant. The model must be able to predict the outlet Silica in the ore using dynamic models. We were tasked with investigating how the regression coefficients change with time and, based on this analysis, using lagged variables and reducing them to the minimum possible. The developed model had to be able to operate online and adapt to the changing dynamics of the process in real-time, which meant some constrictions applied to how recent the measured variables could be when performing an outlet Silica content estimation.

For the final report, we were asked to compile all previous submissions and apply the necessary corrections requested during week 7. The document is organised as follows: in Section 2, we get a first glance at the raw data, identifying the necessary corrections that are eventually implemented in Section 3 and visualised in Section 4; in Section 5 we go over the mathematical theory required to build the soft sensor; in Section 6 we go over the first practical considerations regarding the usage of the pre-processed samples; in Section 7, the pipeline of the algorithm is described and the development roles are assigned; the implementation of the aforementioned algorithm is described in Section 8, with the numerical results of running it over the given dataset being presented in Section 9; finally, further conclusions and possibilities of improvement based on the numerical results are presented in Section 10.

2 Dataset challenges

Our dataset contains time-series data from a mining plant's froth flotation phase, where hydrophobic materials are separated from hydrophilic materials. Since we lacked proficiency in ore refinement, we did not have much prior knowledge about the specifics of what each sensor measured.

The time series contains sensor readings, settings, and measurements about the input and output ore purity. The dataset was gathered between 10.03.2017 - 09.09.2017. Sensor and setting data were measured every 20 seconds, but input and output mineral concentrations (iron and silica) were only measured every hour since getting a concentration measurement was not automated.

The first two days of the dataset presented hours with less than 180 sensor measurements (once every 20 seconds). The dataset only had hourly timestamps, so it was impossible to figure out where the missing measurements were located. The removal of the first two days would only remove 3.6% of the data, and that is what we ended up doing.

The difference in sampling frequencies makes it hard to correlate sensor readings to concentrations. This problem can be dealt with by re-sampling the time series; i.e., interpolating rare measurements (up-sampling) or combining frequent measures (downsampling) to match each other. Another problem we faced with the dataset was that the mining plant had periods of abnormal functioning and/or incorrect measurements, which we identified both graphically and numerically (applying the discrete-difference operator to the time vector and locating abnormal values).

3 Data pre-treatment

The pre-processing of the mining dataset for predicting the outlet silica concentration involves addressing three primary challenges: Dealing with missing samples, dealing with dissimilar frequencies, and dealing with the static feature.

Before going over these, however, some localisation (format) issues had to be resolved; this is done through the `fish` script `ds_preprocessing.fish`. If unavailable, the commands below (written in standard Unix `Bash`) can be executed from the project's root directory to get the required file.

```
mkdir ./datasets
cp Mining_Dataset.csv ./datasets/mining_process_database_reformatted.csv
sed -E -i '2,$s/"([0-9]+)\,([0-9]+)"/"\1.\2"/g;s/,/;/g;s/"//g' \
    ./datasets/mining_process_database_reformatted.csv
```

All figures and analysis presented in this section can be obtained by running the `dataset_regularisation.mlx` Live Script.

3.1 Dealing with missing samples

There is an absence of samples between March 16 at 05:00 and March 29 at 12:00.The abnormalities in this section of the dataset can be fixed using two options: Imputation or removal. Since the abnormalities are situated at the beginning of the time series and constitute only 3.63% of the total dataset, the section of the dataset is removed for further analysis. Furthermore, following the removal of the section, we encountered a missing sample for the time April 10, at 00:00. To fix the missing sample, we imputed the data with the last sample hour. Figures 1 and 2 depict the absence of the sample in the dataset. Figure 3 shows the imputation of data for the missing sample for April 10, at 00:00.

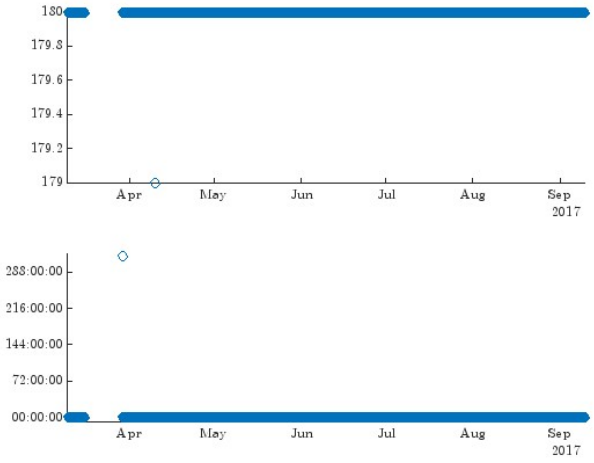

Figure 1: Missing Sample 1

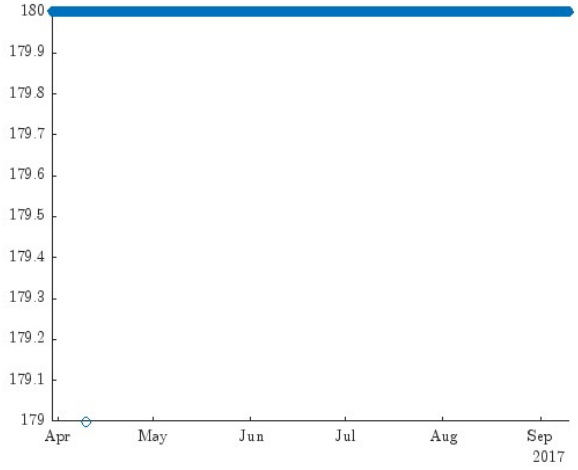

Figure 2: Missing Sample 2

3.2 Dealing with Dissimilar Frequencies

From the descriptions of the dataset, we know that the observation for some features changes every 20 seconds, while some features remain constant for an hour. The problem can be effectively addressed through resampling, which includes both down-sampling and up-sampling methods. For up-sampling, we assume that the initial sample is collected at the 0^{th} second of the hour, with subsequent samples captured regularly at 20-second intervals. We can rectify the up-sampling process by adjusting the timestamp in the dataset. However, for down-sampling, we are considering two different approaches. One of the methods involves the calculation of hourly medians for each sample, and the other

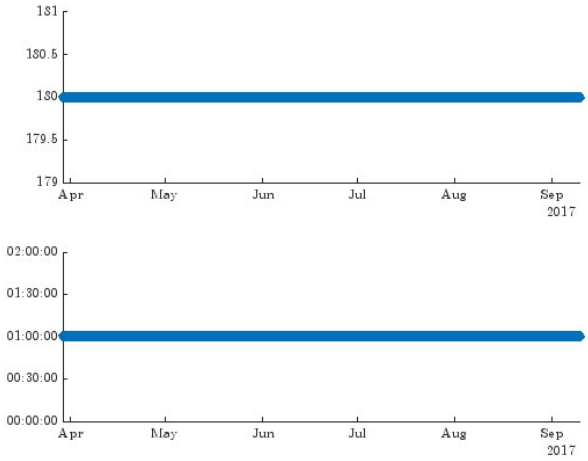

Figure 3: Imputation

method involves the aggregation of feature data into hourly vectors. Since new Silica outlet samples are collected only every hour, we decided on down-sampling our data using vector-wise feature extraction, which later ended up being the extraction of an hourly channel-wise median.

3.3 Dealing with static features

From observing Feature 1 and Feature 2 i.e., the percentages of iron feed and silica feed, we noticed that their values remain constant within the time frame from 13.05.2017 to 15.06.2017. In other words, there is no variation in the percentage of iron/silica feed, despite the percentage of iron/silica outlet concentrate oscillating as in the rest of the time series. Such a long period without any change cannot be regarded as a valid measure since it will severely impact the predictive capabilities of our model. We should immediately address this problem by eliminating this section of the dataset if any of those variables were used to calibrate a model. However, we cannot proceed as we did with the missing samples at the beginning of the dataset and simply discard all samples before the end of the steady period, since this will result in the loss of nearly half of the total remaining samples.

We decided to eliminate the data from this particular period (i.e., 13.05.2017 to 15.06.2017). Furthermore, we decided to split the dataset into initialization/test subsets using this anomaly as the breaking point. We will use the first half of the dataset for optimizing the model's hyperparameters, and the latter half for testing our dynamic model. Figure 4 can help to understand the situation and reason for the pre-treatment of the static features.

We also plotted the other time-series (Figures 5 to 7) and looked for similar temporal irregularities, but found none. Thus, the only period masked out from the dataset will be the one marked by the irregularities in the percentual Iron/Silica feed time series.

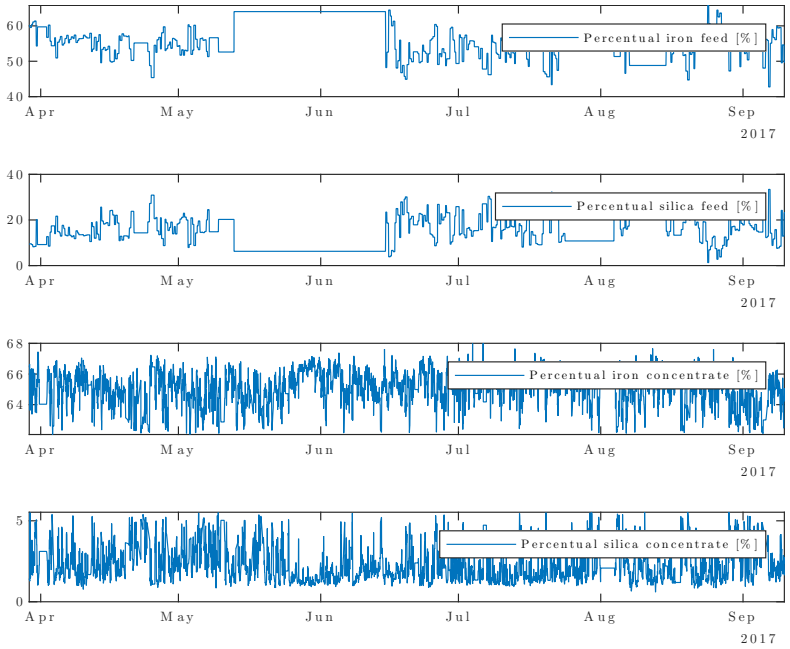

Figure 4: Percentual flow variables at inlet/outlet

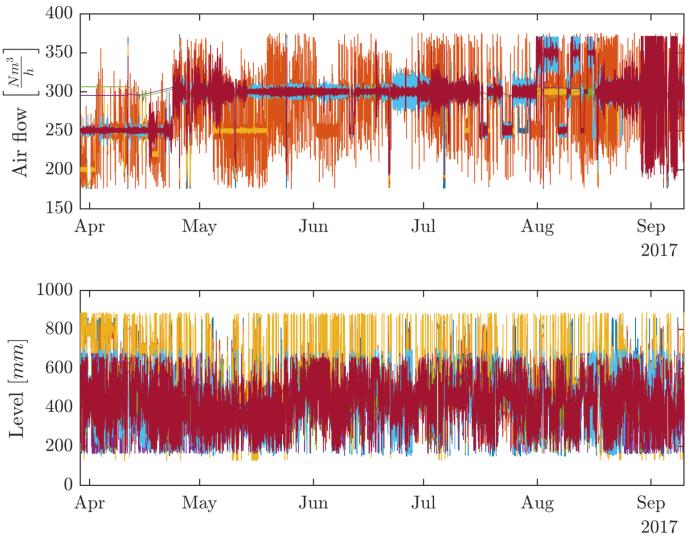

Figure 5: Temporal variables concerning the flotation columns

4 Data visualisation

With the aforementioned *mask* in mind, we proceeded with the variable distribution visualisation, for which we used both box plots (when scale grouping made physical sense)

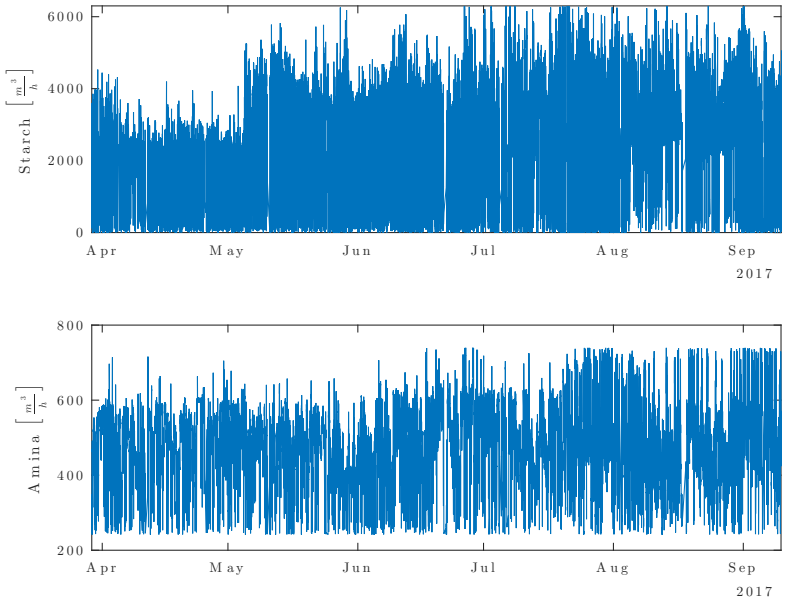

Figure 6: Flow temporal variables for Starch and Amina

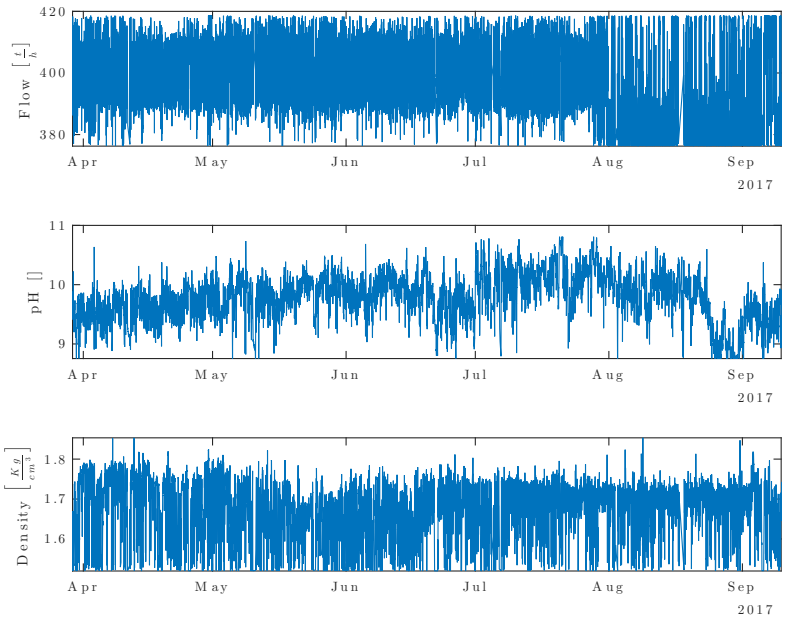

Figure 7: Ore pulp temporal variables

and histograms (when variables were either too different in scale or belonged to unique physical categories).

On the first box plot (Figure 8) we observed no anomalies; rather, the correlation between the proportion of both Iron and Silica at the inlet and their corresponding

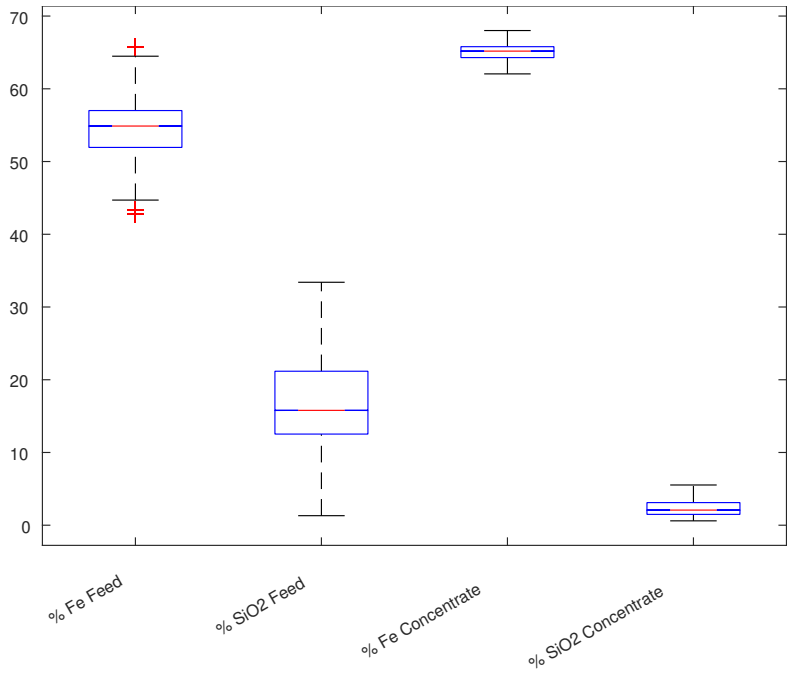

Figure 8: Percentual flow variables at inlet/outlet (box plot)

ratios at the outlet was coherent with what one would expect from a physical system. Furthermore, the lower dispersion of mineral concentrations at the outlet when compared to those at the inlet evidences a regularisation effect induced by the flotation plant on the flow.

On the Flotation Column box plots (Figures 9 and 10) we could observe several distinct behavioural patterns depending on the column number:

- From the column airflow:
  - All columns have almost the same median (50th percentile).
  - Columns 1, 6, and 7 present a similar sample distribution, with most samples located slightly above the median.
  - Columns 2 and 3 are the most dispersed, presenting several outliers near the 180 $\frac{Nm^3}{h}$ mark.
  - Columns 4 and 5 are the most concentrated, with the marked outliers falling inside the range all the other columns would interpret as normal samples.
- From the column level:
  - Columns 1 to 3 have a higher variance and mean but present no outliers.
  - Columns 4 to 7 have a lower variance and mean, but evidence several outliers (both upper and lower).

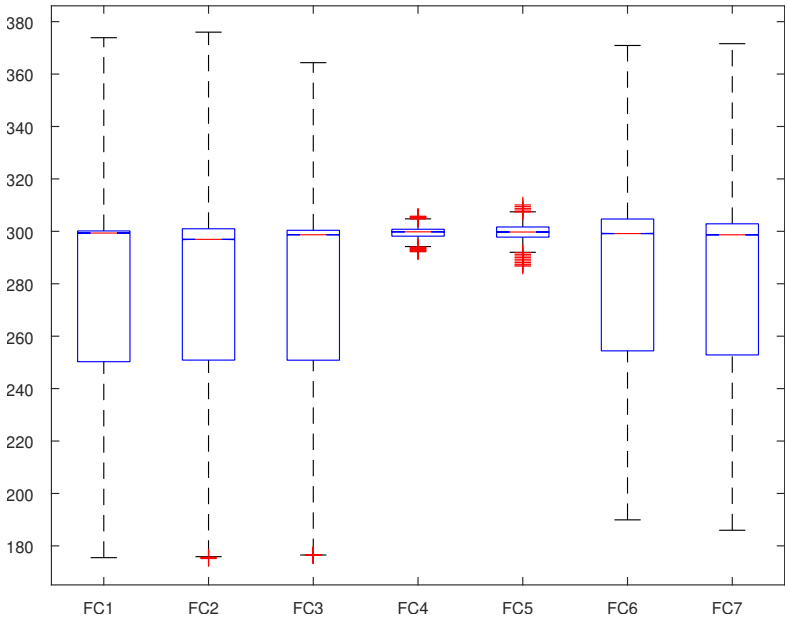

Figure 9: Flotation column airflow $\left[\frac{Nm^3}{h}\right]$ (box plot)

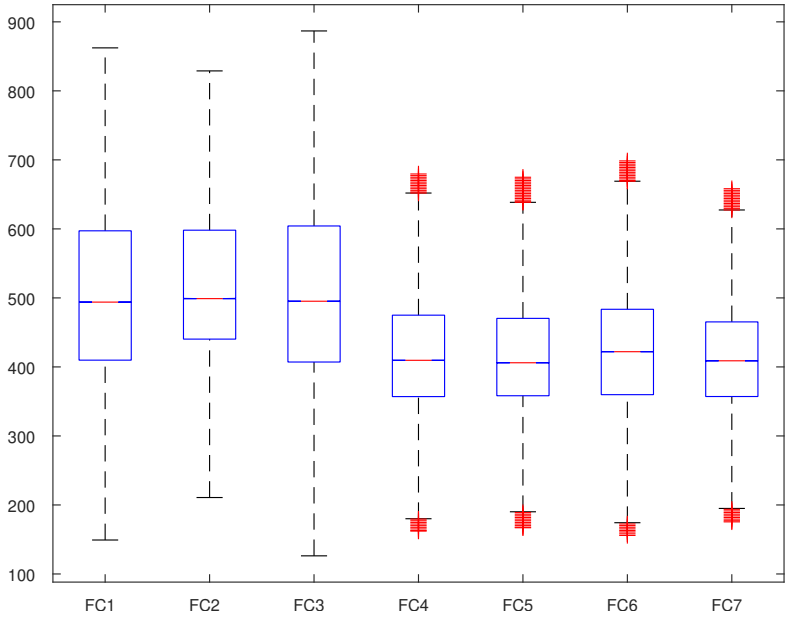

Figure 10: Flotation column level $[mm]$ (box plot)

The similarities between some of the flotation columns might suggest that a dimensional reduction can be performed by grouping these variables according to their *behavioural patterns* and formulating class descriptors, both increasing the robustness of the features and reducing the size of the estimation model.

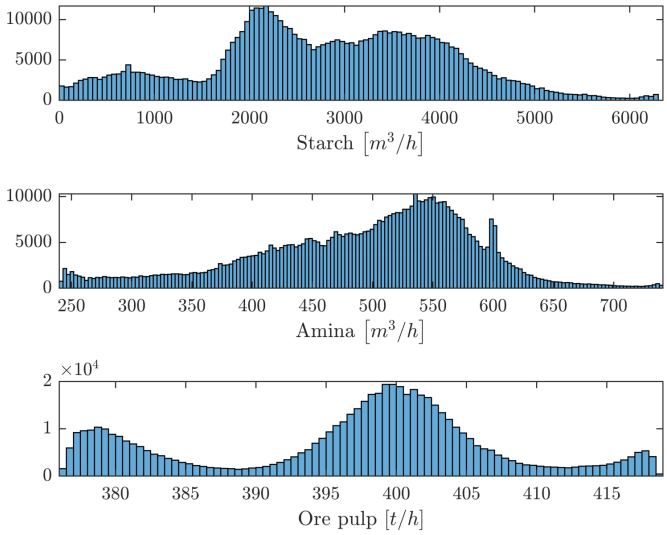

Figure 11: Flow variables (histograms)

Not a lot could be said about the flow histograms (Figure 11), other than noting that the *Amina* flow distribution is the closest to a *Normal* random distribution (with an unexpected concentration of samples around 600 $\frac{m^3}{h}$), while the *Ore pulp* one presents a distinct concentration of outliers near its lower and upper limits.

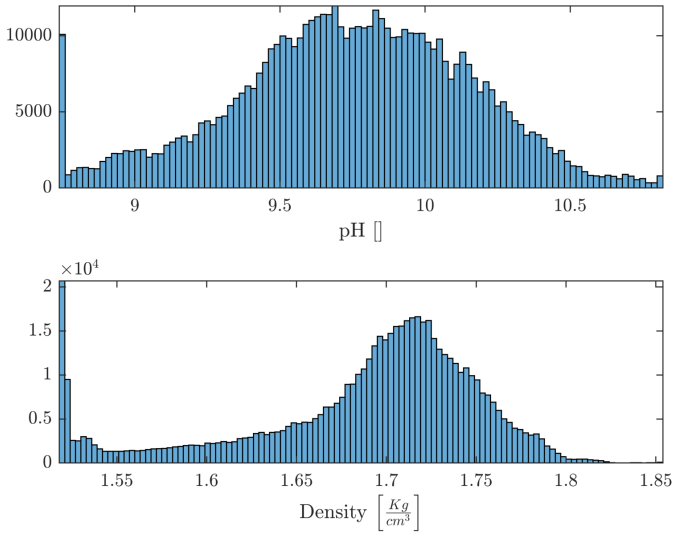

Figure 12: Ore-pulp-associated variables (histograms)

Finally, regarding the remaining Ore-pulp-associated variables (Figure 12), we see an

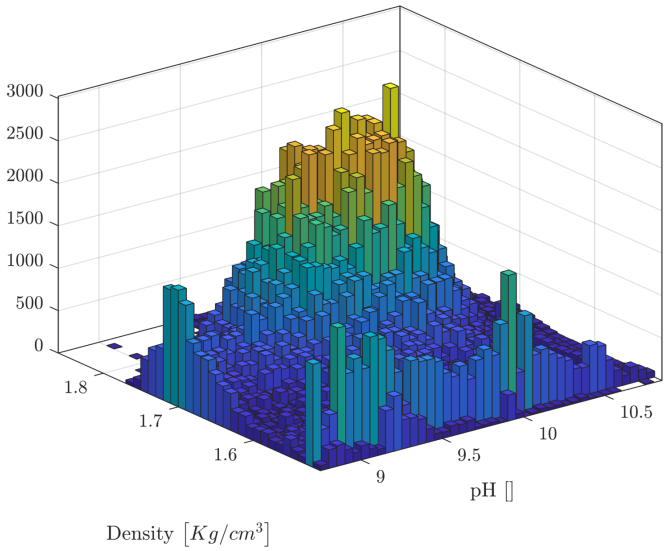

Figure 13: Ore-pulp-associated variables (bi-variate histogram)

almost perfect couple of normal distributions, with a marked increased count on their lower extrema. Not only that, but these phenomena are uncorrelated as evidenced by the bi-variate histogram in Figure 13. Going back to Figure 7, we can spot a short period near the end of August with a lower-than-normal average Ore pulp pH value, which might explain the histogram's shape; unfortunately, there is no immediately evident equivalent event in the Ore pulp density time series.

5 Mathematical Methods

5.1 Feature extraction and resampling

Our approach will rely on feature engineering to synchronise input variables sampled at different frequencies with the target variable.

Our feature extraction, which started as very complex, was reduced down to perhaps the simplest possible: Compressing one hour of fast sensor measurements to each signal mean over the hour. We have 19 sensors and each sensor sends a measure 180 times per hour. We call these 'fast variables'.

Since the laboratory measurements take one hour to finish, we use this feature extraction to synchronise the fast variables to the slow laboratory measurements.

It is also possible to not do feature extraction if, for example, one would like to conserve the information from fast variables fully. Then, to synchronise the fast variables to slow variables, one would flatten the 19×180 matrix containing the fast variables from the last hour.

The extracted information for each operation hour is stored as a vector by horizontally

concatenating the extracted features with the lab measurements from the samples sent for analysis at the beginning of the hour.

To estimate the current silica concentrate based on the last N hours, our model will take in N such vectors (once again concatenated horizontally).

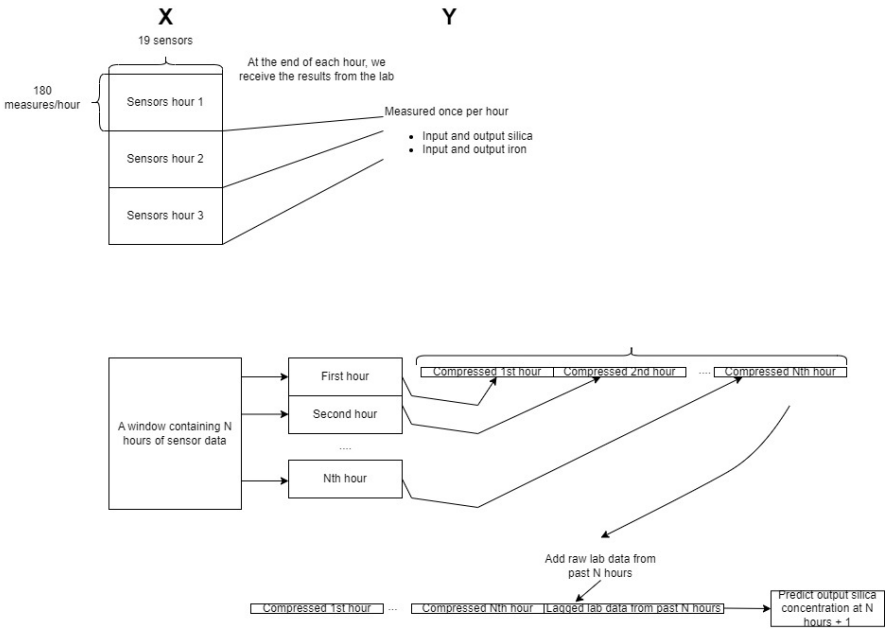

Figure 14: Slow variables are measured once every hour, and the fast variables are measured 180 times per hour. To estimate the current silica concentration (predict the next lab result) data from the N last hours is used.

5.2 Partial Least Squares (PLS) Regression

The project uses a Partial Least Squares (PLS) Regression model as its core mathematical framework for dynamic modelling. PLS models offer distinct advantages and help overcome the limitations of Principal Component Regression (PCR). PCR models use the principal components that explain X rather than Y. On the other hand, PLS regression models regress Y from X by finding the components in X also relevant to Y. PLS models achieve this by generating a set of latent variables that performs a decomposition of X and Y at the same time where the components explain maximum covariance between X and Y.

Partial Least Squares (PLS) decomposition is a fundamental technique used in data analysis. It decomposes the X data according to the following equation:

$$X = TP^T + E \tag{1}$$

In Equation (1), T denotes object scores, P represents variable loading, W denotes x-weights, and E accounts for error and noise.

Similarly, for the Y data, PLS decomposition is as follows:

$$Y = UQ^T + F \tag{2}$$

In Equation (2), U stands for object scores, Q represents variable loading, and F represents the regression error.

Following the decomposition of X and Y, the covariance between X and Y is maximised by rotating the PCA solutions; i.e., the covariance between object scores T and U. After the rotation P changes to W and W is applied in computing the PLS regression coefficient as outlined in Equation (3).

$$b = W(P^TW)^{-1}Q^T, \tag{3}$$

6 Model Calibration/Validation/Testing

Since the overall objective of the project is to develop a dynamically updated model for estimating the current Silica ratio at the outlet to avoid the necessity to wait for the samples to be processed at the lab, clearly delimited Training/Validation and Testing stages cannot be defined (in contrast to the case of static models).

As such, the following pipeline is proposed:

1. The first "isolated" segment of the dataset will be used as the initialisation dataset, allowing us to optimise hyper-parameters and train the first model.

2. Output estimates and measurements during the second period of the dataset are stored as they become available until enough samples have been accumulated; then, two different model update rules are proposed:

   a) Performance-based update: a minimum performance score threshold is set; if model performance is lacking over recent estimates, the model is updated.

   b) Time-based update: performance is not evaluated; instead, the model is assumed to be invalid after a fixed amount of time, and is updated regardless of its recent accuracy.

   Out of these, we use the Time-based update.

3. The model is re-fit to follow the current plant dynamics by calculating PLS over the latest data (hyperparameter).

7 Model Flowchart and Member Roles

In Figure 15 we can see a general view of the algorithm's functioning; likewise, in Figure 16 we can see the detailed flow diagram describing the (proposed) real-time operation of our dynamic model[1]. First, the following parameters are provided:

[1]Based on the peer review from week 5, we corrected the flowchart and elaborated a simplified version of our pipeline.

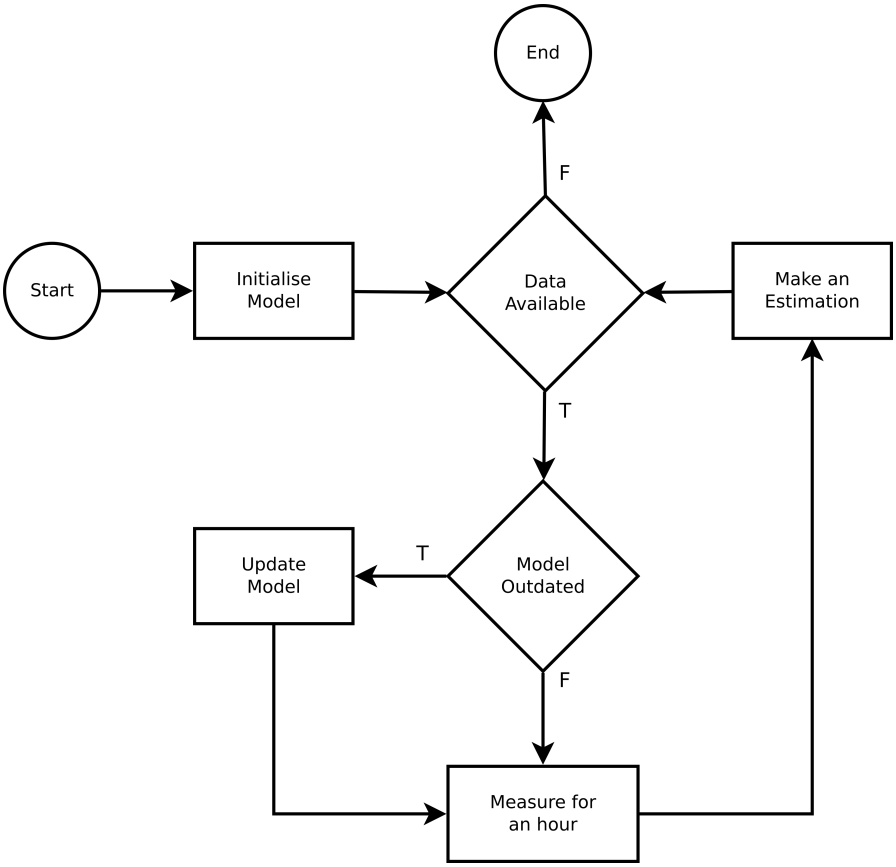

Figure 15: Simplified algorithm flowchart

- *perf*: Boolean flag controlling whether performance or time-based model updates are used.
- *Q2_thr*: Real parameter that sets the minimum acceptable Q^2 score for the model to still be considered *valid* (only used if *perf* is set to *True*).
- *N_trn*: Integer parameter that defines how many hours of the latest data is used to re-train the model.
- *T_recal*: Integer parameter that sets the time (in hours) between each time-based update of the model.

A starting dataset is used to train a first estimation model and to optimise hyper-parameters, such as the number of Principal Components, which variables to use, etc. Then, the following loop algorithm is used:

1. If there are enough new samples the model update condition (be it time or performance-constrained) is checked and, if deemed necessary, the model is re-calibrated

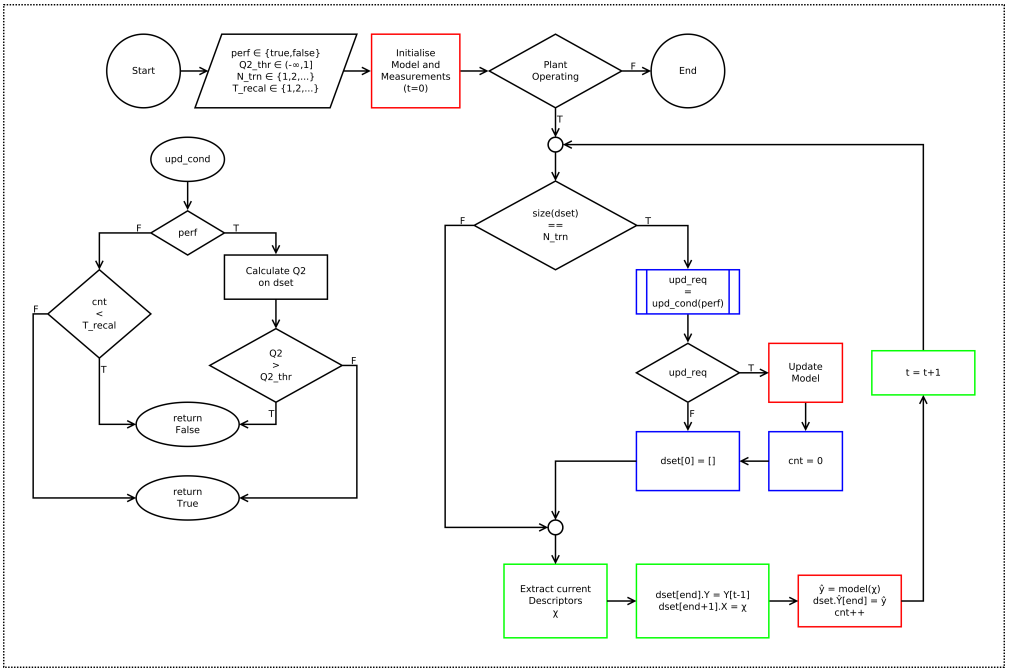

Figure 16: Model Flow Diagram (algorithm)

- The oldest sample is discarded regardless.

2. The latest measurements are stored (comprising the last hour), and the current descriptor variables χ are generated; these are:
   - Current and past (lagged) sensor measurements;
   - Synthetic variables generated by encoding the information of the faster sensors (and their lagged versions).
   - Moreover, the measured Silica ratio at the outlet is stored in case it is later needed for model re-calibration.
3. The regression is performed, with the latest estimate being returned and recorded. Within the following hour, the algorithm goes back to step 1 so the model can be ready once a new input is available.

In practice, the model is updated every time a new estimation has to be made, which is equivalent to setting *perf* to false and *T_recal* to 1.

The blocks in the diagram are colour-coded based on the group member's assigned tasks as follows:

- Prashant Shrestha (Red): regression tasks, including model initialisation, model re-calibration, and output formatting.

- Ilmari Vahteristo (Green): feature engineering tasks, heuristic sub-feature selection, and input formatting.
- Sergio Vanegas (Blue): dataset and control tasks, including model update logic, IO synchronised storage and database cleanup.

It is worth noting that the above diagram is formulated for a *virtual sensor* use-case, which estimates the current Silica ratio at the outlet. In the case of a true predictor (estimate future output values with present information), the only change that has to be made is how the *ground truth* measurements are aligned with the forecast values as they become available. For this reason, some sections of the report are written for an arbitrary prediction horizon instead of the 1-hour "prediction horizon" (in practice, estimation) used for the numerical analysis.

8 Model Implementation

The initialisation of the model helps identify the ideal number of components in the Partial Least Squares (PLS) model; thus, this section is split into Dataset/Model Optimisation and Model Construction.

8.1 Dataset and Model Optimisation

For experimentation, various PLS models were constructed and compared to one another, assessing their performance using metrics such as the R2 and Q2 scores, and performing variable selection based on the fitted coefficients. This comparative experimentation stage helped identify the most promising model hyper-parameters. The figures and analysis displayed in this section can be found in the `Variable_and_Optimal_Window.mlx` Live Script.

The raw (sanitised) dataset contains 23 time-indexed variables, which are the following:

- prc_iron_feed: Percentage of iron in the slurry being fed to the flotation cells (0-100%).
- prc_silica_feed: Percentage of silica in the slurry being fed to the flotation cells. (0-100%).
- starch_flow: Flow rate of starch (reactive) measured in m3/h.
- amina_flow: Flow rate of amine (reactive) measured in m3/h.
- ore_pulp_flow: Feed flow rate of pulp measured in t/h.
- ore_pulp_ph: pH of the pulp, scale from 0 to 14.
- ore_pulp_density: Density of the pulp measured in kg/cm3.
- fc(1-7)_air_flow: Air flow rate entering flotation cells 1 through 7.

- fc(1-7)_level: Height of the bubble layer at the top of flotation cells 1 through 7.
- prc_iron_concentrate: Percentage of iron in the concentrate at the end of the flotation process (%), obtained through subsequent laboratory analysis.
- prc_silica_concentrate: Percentage of silica in the concentrate at the end of the flotation process (%), obtained through subsequent laboratory analysis.

The number of features, however, depends on the desired number of heuristic encoders (median in our specific case) and lags, which is implemented as a horizontal concatenation of shifted columns. The target variable/feature is the outlet silica concentration measured at the start of the following hour (i.e., at time $t + 1$), which introduces an additional shift between the input and output columns. Both these operations are performed by the `init_dataset` function.

Not all variables are desired to be used by the final model, however. Instead, the `Var_name` function is used to optimise both the per-feature input memory, returning a list with the most significant ones for a PLS regression. A maximum lag of 10 hours is explored.

Figures 17 and 18 provide insight into the observed changes in R2 and Q2 scores as lagged variables were introduced. The analysis revealed that the Q2 and R2 scores remained relatively stable until the introduction of eight-hour lagged variables. However, the inclusion of nine-hour lagged variables decreases the R2 and Q2 significantly. The Q2 and R2 value increases again with the inclusion of 10-hour lagged variables. Furthermore, the figure depicts the model with the inclusion of 4-hour memory has the highest R2 and relatively good Q2. Therefore, 4-hour input memory is selected for creating the PLS model. It helps to ensure the predictive performance of the model.

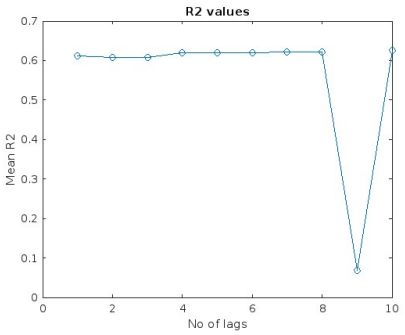

Figure 17: Mean of R2 vs Lag

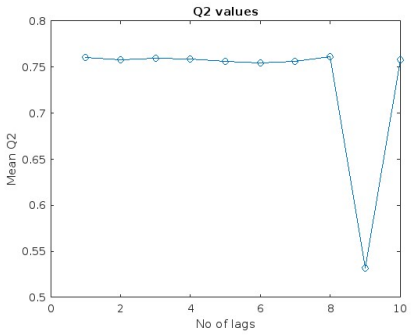

Figure 18: Mean of Q2 vs lags

8.2 Model Construction

The dynamic model construction is supported by three main functions, described in Sections 8.2.1 to 8.2.3.

8.2.1 Function 1: `Var_name`

The `Var_name` function is an important component of the dynamic model, responsible for variable selection. This function iteratively selects variables to optimise the model's performance based on Q2 scores while considering variable importance. It dynamically determines which variables are most informative for the modelling process. The function takes the input dataset containing original and lagged variables i.e., X_init and Y_init as parameters. Moreover, it takes the maximum number of components to be used in models i.e.,ncomps_max and the desired minimum number of features to be selected i.e., min_features as the parameters.

The `Var_name` function operates through the following step-by-step process to determine the optimal set of input variables for the dynamic model:

1. Initialisation:
   - `test_ratio` is set to 0.4, indicating that 40% of the data will be used for testing.
   - `alpha_perf` is set to 0.8, serving as an alpha threshold for performance improvement.
   - `div_idx` is calculated to split the data into training and testing sets.
   - `best_input_vars` is initialised with the variable names of the predictor variables in X.
   - `pls_options` is configured to enable parallel processing.
2. Data Pre-processing: The data is divided into *training* and *test* data. The *training* data is normalised and parameters i.e., mean and standard deviation are saved. The testing dataset is normalised using the same normalisation parameters.
3. Iterative Variable Removal: For each iteration, it performs PLS regression with a varying number of components. The Q2 score is calculated for each iteration, and the features that result in the highest Q2 score are selected.
4. Iterative Variable Selection: For each iteration, it identifies the feature that results in the smallest change in the Q2 score when removed from the model.
5. Final Output: The function returns `variable_names,` which is a list of the names of the selected features.

8.2.2 Function 2: `pls_model_initialization`

The `pls_model_initialization` function is responsible for obtaining the optimal Partial Least Squares (PLS) model. It iteratively constructs and evaluates PLS models with varying numbers of components to determine the best-fitting model based on the Q2 score.

The parameters used in the function are given below:

- `X_1`: The independent variable dataset containing original and selected variables.
- `Y_1`: The dependent variable dataset represents the target variable.
- `variable_names`: A vector specifying the variable names considered for modelling.
- `max_n_components`: The maximum number of components to consider in the PLS model.

The `pls_model_initialization` function operates through the following step-by-step process to determine the optimal PLS model for the given dataset:

1. Data Preparation: The function begins by selecting a subset of variables specified by `variable_names` from the independent variable dataset `X_1`.
2. Component Counts Initialisation: An array `component_counts` is created, ranging from 1 to `max_n_components`, to represent the number of components to consider in the PLS model.
3. Model Initialisation: The function initialises a structure `model` to store information about different PLS models, including coefficients, MSE, and statistics. A variable `max_q2` is initialised to negative infinity to track the highest Q2 score achieved during the iteration.
4. Data Split: A train-test split is performed (indicated by `train_indices` and `test_indices`), allocating approximately 70% of the data to the training set.
5. Iterative PLS Model Construction:
   - For each component count `j` in `component_counts`, the following steps are executed:
   - The training data (`X`) based on the selected variables is scaled.
   - The test data (`Xt`) is scaled using the mean (`mu`) and standard deviation (`sig`) obtained from the training data.
   - The training and test target variable (`Y` and `Yt`) is scaled.
   - PLS model with `j` components is built and the relevant model information is stored.
   - The R2 score and Q2 score for the model are calculated.
   - The model with the highest Q2 score is updated as the optimal model.
6. Final Output: The function returns the optimal PLS model, which includes the input and target datasets, the number of components used, the variable names, and the scaling parameters for the input data.

8.2.3 Function 3: `update_model`

The `update_model` function updates the Partial Least Squares (PLS) model with new data while keeping the optimal number of components and variable selection. It takes the old PLS model, new independent variable data (`X_new`), and new dependent variable data (`Y_new`) as inputs.

The parameters used in the function are given below:

- `old_model`: The old model that contains information about variable selection and scaling.
- `X_new`: The new independent variable dataset to be integrated into the model.
- `Y_new`: The new dependent variable dataset.

The `update_model` function operates on the following steps to integrate new data into the old model.

1. Variable names from the old model (`old_model.variable_names`) are used to select relevant variables from the new independent variable dataset `X_new`.
2. The optimal number of components used in the old model (`old_model.Components`) is extracted.
3. The `X_new` matrix is scaled using the mean and standard deviation obtained from the old model.
4. The new PLS model is constructed using the updated data and the optimal number of components.
5. R2 and Q2 scores are calculated for the updated model, indicating its performance.
6. New model structure (`updated_model`) is created to store the updated model's information i.e., coefficients, MSE, statistics, R2, and Q2 scores.
7. The function returns the updated PLS model.

8.3 Execution Loop

Online execution was simulated by streaming the dataset line-by-line so that the real-time capabilities of the model could be verified. The entire online execution can be found in the `soft_sensing_algorithm.mlx` Live Script.

To ensure causality, a minimum prediction horizon of 1 hour was enforced so that:

- For fast variables, all information gathered during the previous hour is used to generate the resampled feature vector.
- For slow variables, only information extracted from samples sent for analysis an hour prior could be utilised.

In addition to the buffer mentioned in the algorithm flowchart, an additional buffer for fast variable resampling had to be implemented as a way to imitate online operation conditions: for every 180 *fast* samples ($180 \times 20s = 1h$), a new observation is encoded heuristically and concatenated with the slow variables taken one hour earlier in the same way the initialisation dataset was generated.

A table containing past non-lagged input vectors is kept and lagged on-demand to avoid redundant sample storage. This table is updated every time a new input vector is created, removing the oldest sample and keeping the minimum amount of samples required to update the model. The selected re-calibration dataset length was 1 week (the rationale for this will be further discussed in Section 9), which resulted in a total of 172 input vectors stored ($N_{\text{trn}} + \text{Input Memory} = 168 + 4 = 172$ samples). For output records, however, all past estimations are stored for analysis in a timetable; new estimations (saturated between 0 and 100 since the regressed variable is a percentage) are added as they are made, and stored with a timestamp according to the prediction horizon, whereas the measured values are recorded in the current timestamp once available. At the end of the execution, all predictions without a ground truth are removed from the record.

During the loop's execution, the most recent estimations and measurements (with the window being determined by N_{trn}) are plotted for monitoring purposes, as shown by Figure 19.

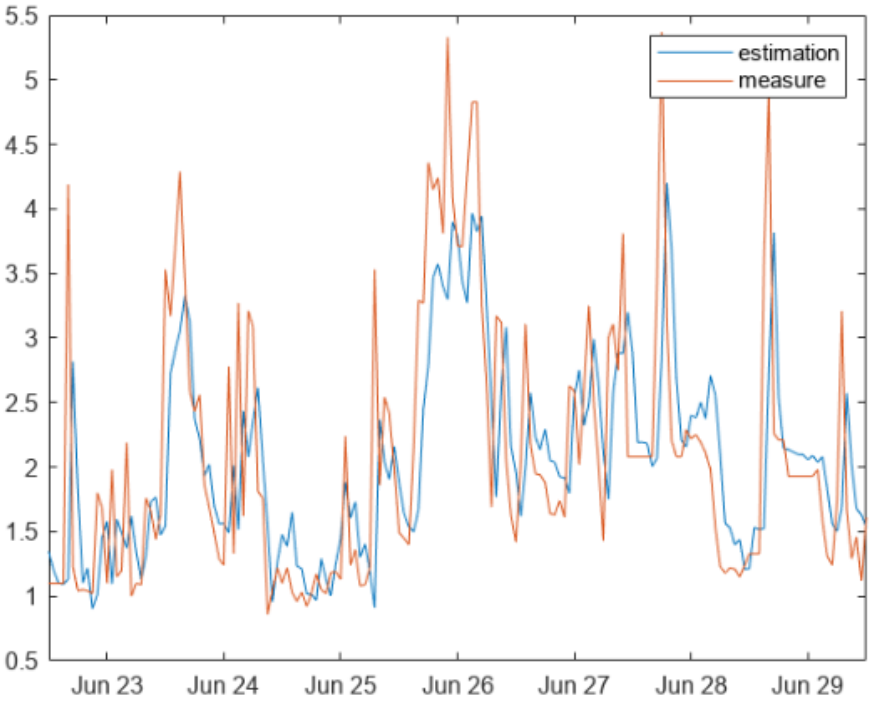

Figure 19: Online estimation/measurement plotting

9 Results and Discussion

The described implementation yields a global Q2 score of 0.5294, an RMS error of 0.7312%, and a maximum absolute error of 3.9981%[2]. The entire estimation is shown in Figure 20, while the evolution of the PLS coefficients is shown in Figure 21. As expected, past outlet Silica and Iron concentrates play a big role in the estimation, with the starch flow complementing the information on the process.

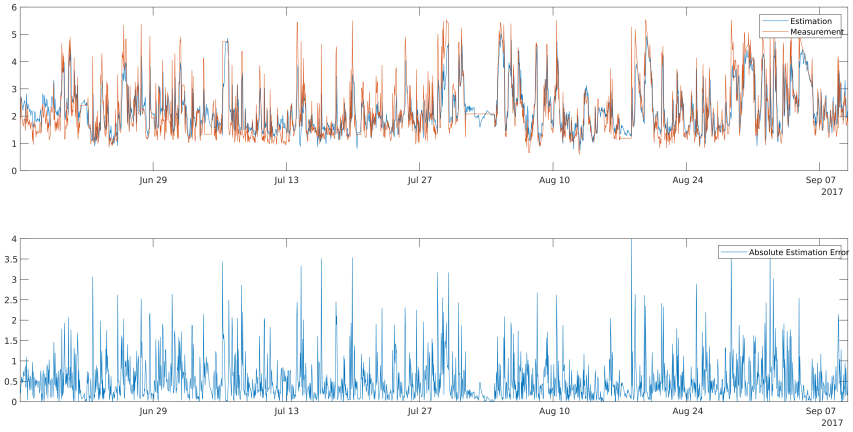

Figure 20: Simulation results

Figure 21: Coefficient evolution

Despite extensive data pre-processing, short-lived irregularities persist in the dataset (as

[2]The above percentages correspond to the Outlet Silica Ratio, and not to a relative measure with respect to the experimental values

shown in Figure 22; by oversizing the training dataset to a degree where the irregularities will never represent the majority of it, we can make the model robust against inaccurate data without adding any logic or control to the training procedure (as evidenced by the ability of the coefficients in Figure 23).

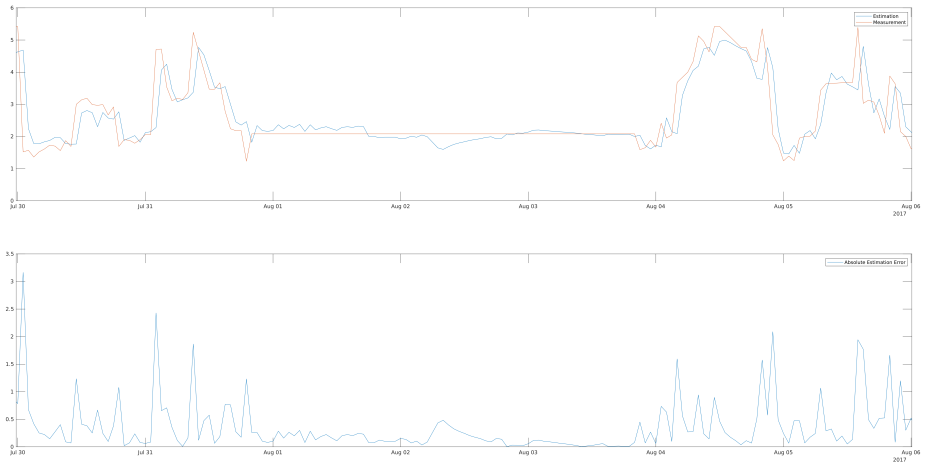

Figure 22: Simulation results - irregularity zoom-in

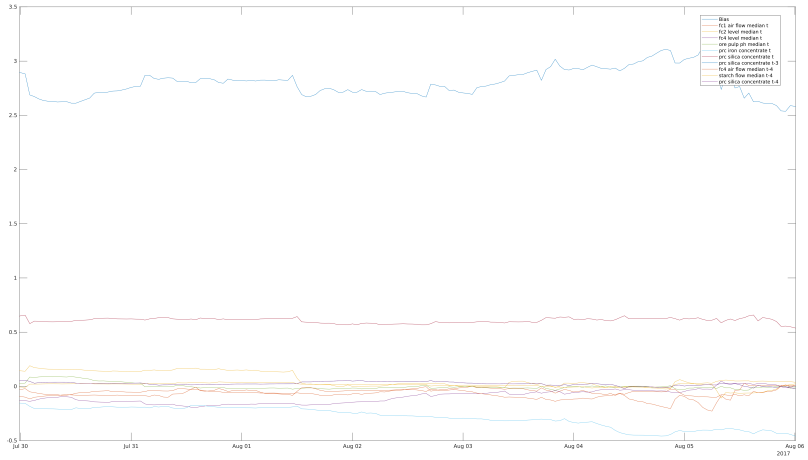

Figure 23: Coefficient evolution - irregularity zoom-in

10 Conclusions

- The proposed model initialisation pipeline is able to identify the optimal model hyper-parameters (input memory, input features, number of PCs), which renders the update operation feasible in real time since the algorithm only has to update the linear regression coefficients.
  - In this implementation, a monotonously decreasing variable selection scheme was used. Alternative approaches (such as Univariate Feature Selection) or more complex variations of the decremental approach (such as Recursive Combined Feature Elimination/Aggregation) could be exploited to reduce input redundancy; nevertheless, these might simultaneously reduce the resilience of the model against single-channel noise, turning this choice into a trade-off situation.
- Likewise, the proposed operation loop is able to track the variations in the dynamics of the plant while keeping the stored input information to a minimum and avoiding redundant encoding operations.
- The simplicity of the model's approach to robustness makes it highly adaptable while remaining fairly simple to implement from a hardware POV; nevertheless, if there was enough resource overhead available, a more complex approach involving sample discrimination and irregularity detection could be added to the re-calibration logic to increase its adaptability to new information without losing model stability.