Evaluation of Kalman filter for meteorological time series imputation

Simone Massaro

Master Thesis
Forest and Ecosystem Sciences

Department Name
Georg-August Universität Göttigen
February 2023

Contents

1 Introduction

1.1 Eddy Covariance and gaps in meteorological variables

Eddy Covariance (EC) is a state of the art technique for measuring green house gases and energy exchanges between ecosystems and the atmosphere [1]. The technique allows for non-destructive measurements at the ecosystem level with a high temporal resolution. Eddy Covariance data is used for ecological and physiological research of ecosystems, as well as for validation of ecosystem process models and remote sensing observations [14] ecological level. The core of Eddy Covariance site is the 3D anemometer and gas analysers, which allows estimating the fluxes of interests (e.g. CO_2, H_2O, CH_4). Beside the fluxes, an Eddy Covariance setups collects measurements of meteorological variables and ecosystem parameters. This additional data provides the context to use and interpret the fluxes measurements.

The acquisition of the meteorological variables can be interrupted by failures in the instruments or power outages, resulting in gaps in the time series [1]. The presence of gaps is a problem for several uses of the EC data.
An important application of EC is the validation of Land Surface Models [2, 9, 4, 12], which are process based model that estimate fluxes using meteorological conditions as input. The errors of Land Surface Models deriving from inaccuracies in the input are comparable to the errors arising from the imperfection in the models formulations [22]. This highlights the need of high quality continuous meteorological measurement that reflect that exact condition at the flux station are needed. Meteorological observations are used as a driver to impute gaps in the fluxes measurements [1], which requires complete meteorological time series. Finally, the presence of gaps affects the calculation of long term averages for meteorological variables.

The described use cases make it necessary to impute the gaps in the meteorological variables. The first approach to reduce the number of gaps in to have redundant instruments on the site, however is this is not always possible and statistical models are used for imputing the gaps [1]. There are three different approaches that can be used to reconstruct missing data: 1) use the *temporal autocorrelation* of the variables, the measurements before and after the gap provide information on the missing data; 2) use *correlation* between different variables, if not all variables are missing the correlation between variables can be used for imputing the missing variable; 3) use *other measurements*, meteorological variables not only measured in EC tower and the data from nearby meteorological stations can also be used for imputation.

1.2 Current Imputation methods

EC post-processing pipeline impute Gap meteorological time series. Arguably the most widely used post-processing pipeline is ONEFlux [15], which is adopted by FLUXENT the global EC network, ICOS the European as well as AmeriFlux, the American EC network. ONEFlux uses two different methods for imputing the meteorological data: Marginal Distribution Sampling (MDS) and ERA-Interim (ERA-I). The final gap-filled meteorological product uses either MDS or ERA-I, dending on the quality flag of MDS.

MDS Marginal Distribution Sampling [17] estimates the value of the missing variable by using the observations of the variable from other data points with similar meteorological conditions. The algorithm finds all the similar conditions by taking the observations from a time window around the gap and where other meteorological variables have similar values in the gap. All the observations of the variables of interest from similar conditions are then averaged to generate the filling value. In case there are not similar meteorological conditions in the starting time window, the size of the time window is progressively increased. If other meteorological variables are also missing, they are not used to find similar conditions.
The algorithm implemented in ONEFlux uses as drivers the incoming shortwave radiation (`SW_IN`), air temperature (`TA`) and Vapour pressure deficit (`VPD`). If `TA` or `VPD` is missing, `SW_IN` is used as the only driver. If all drivers are missing, the mean value at the same of the day is used for gap filling. The starting size of the time window is 7 days.
MDS has a quality flag with 3 possible values (i.e. 1,2,3) that depends on the size of the time window. in ONEFlux MDS is used only if the quality flag is 1, which means that there are similar conditions are found in a time window smaller than 14 days.

ERA-Interim ERA-Interim is a global meteoroligical dataset provided by the European Centre for Medium-range Weather Forecast (ECMWF). Weather forecast models are used to reanalyse part observations and produce a continuous and complete dataset for all the globe. The main drawback is the low spatial resolution and temporal resolution, that are respectively 0.7° (roughly 38km) and 3 hours. Moreover, only a subset of the meteoroligical variables are available in ERA-I. ONEFlux reduces the error of the ERA-I data by doing a bias correction with a linear regression and temporally downscaled to match the half-hourly frequency of FLUXNET [19].

Other methods ONEFLux is not the only EC post-processing pipeline which gap fills meteorological data. The imputation approaches are very similar, in other libraries like REddyProc [20] or OzFlux [10]). A minor difference is that OzFlux includes both ERA-Interim and the Australian Weather Service and for each gaps select the dataset with the smallest error for a window of 90 days around the gap.

Limitations of current methods There are three possible direction to improve the current imputation methods 1) make a better use of temporal autocorrelation of the variables 2) combine different imputation approaches in one prediction 3) provide detailed information on the quality of imputation.

Temporal autocorrelation MDS uses the temporal autocorrelation only in a limited way, as it takes the average of the missing variable across the whole time window and doesn't attribute more weight to the observations closes to the gap, which have the high correlation with the data in the gap. This is especially relevant for short and medium gaps, which are the majority of gaps in FLUXNET (Appendix figure **??**). The bias correction for ERA-Interim doesn't take into consideration the observations around the gap, either. Therefore, there is potential in improving the imputation performance by making a better use of the temporal autocorrelation.

Combination of imputation approaches ONEFlux employs both ERA-I and MDS, but the two methods are not used together to improve the predictions, but they are used independently. The criteria to select the method to use is hardcoded in the MDS quality control flags. The information on the missing data from temporal autocorrelation, correlation with other variables and other measurements can be combined to make one more accurate prediction.

Uncertainty A limitation of the current methods is the lack of a robust assessment of the uncertainty of the imputed values. MDS has a quality flag, but it derives from hardcoded values and has only 3 possible values, moreover in the final ONEFlux product the quality flag indicates only which gap filling method has been used. Ideally, each predicted data point has an associated uncertainty, which varies continuously and is interpretable. In this way, the level of confidence of the model in each prediction is available to the data user. The uncertainty can be used either to discard the data above a custom threshold, which can change depending on the application, or directly included in the downstream calculations.

1.3 Kalman Filter

Imputation of missing values is a topic that has been extensively researched. A wide range of methods have been developed ranging from replacing with the mean to deep neural networks. Specifically for meteorological time series there are many different methods. However, imputation in the EC contest has some specific characteristics: the absence of a spatial component (each EC site is modelled separately) and the relatively high number of variables. Moreover, the focus is to impute short and medium gaps (up to 1 week) as almost 99 % of gaps of meteorological variables in FLUXNET are shorter than a week (Appendix figure **??**). For this work, we focused into methods that combine all imputation approaches, include interpretable uncertainty and can be applied globally.

There are many families of models that can be suitable for the task. Probabilistic machine learning algorithms are particularly suited as they directly provide an interpretability uncertainty. Kalman Filter is a probabilistic model that models the evolution of a multivariate latent state over time using a Markov chain. The temporal autocorrelation and the variable correlation is directly considered by the model and is possible to also include ERA-I observations. Other modelling approaches were evaluated: Gaussian Process Factor Analysis and GP-VAE (Gaussian Processes

Variational Autoencoders). Gaussian Processes Factor Analysis [21] uses Gaussian Processes to model a latent variable over time, this is a powerful modelling approach that is fully probabilist and that can model both short and long term changes. The main limitation is the computational complexity, which in the naive formulation scales cubically with the number of observations. GP-VAE [8] combines Gaussian Processes and deep learning, to provide high quality imputation with uncertainties. Kalman Filter was selected, since it's the simplest model of the one considered that still fulfils all the requirements.

The first goal of this work is to develop an imputation method for meteorological time series in the context of EC that employs a Kalman Filter. The imputation performance of the new method is then evaluated by comparing it with the state-of-the-art methods (ERA-I and MDS) and assessing the behaviour in different scenarios. For simplicity, only data from one EC site, Hainich (Germany), will be used.

2 Methods

2.1 Kalman Filter Theory

Kalman Filter models over time a latent variable (x), that represent the state of the system. The state cannot be directly observed, but we can observe meteorological variables (y) that reflect the state of the system. It is possible to use Kalman Filters to impute missing values, as the value of the state can be updated over time even when there are missing observations. The values of the state are hence available for all time steps, which can be used to then predict the missing observed variables. Kalman Filter is a probabilist machine learning algorithms, so it keeps track of the entire distribution of the latent state ($p(x_t)$). The time is considered to be discrete, the state is modelled only at specific times (x_t).

In order to be to model the state over time, assumptions on the behaviour of the system are made. There are three key assumptions 1) the states are connected by a Markov chain, which means that the state at time t depends only on the state at time $t-1$ and not the states at previous times $p(x_t|x_{t-1}) = p(x_t|x_{t-1}, x_{t-2}, \ldots, x_0)$ 2) The value of the observed variable depends on the latent state 3) all the relationships are linear and all distributions are Gaussian. Additionally, the mean of the state at time t may depend also on an external control variable c_t at time t. This control variable doesn't depend on the state of the models, but provide information on how the mean should change. Equations 1 and 2 describe the assumptions on the behaviour of the system:

$$p(x_t|x_{t-1}) = \mathcal{N}(x_t; Ax_{t-1} + b + Bc, Q) \tag{1}$$

$$p(y_t|x_t) = \mathcal{N}(y_t; Hx_t + d, R) \tag{2}$$

The probability distributions of the state are computed using Bayesian inference. The computational cost of probabilistic inference can be drastically reduced in this context, since all the relations are linear and all distributions are Gaussian, which means that probabilistic inference can be performed using linear algebra operations.

Kalman Filter is a recursive algorithm (Figure 1), at time t the *predicted state* (x_t^-) is obtained from the previous state (x_{t-1}) and the *control variable* (c_t). Then the state is update using the *observation* (y_t) to obtain the *filtered state*, observations can be partially or totally missing. This is repeated recursively for all time steps. At this point, each time step, the state x_t depends only on the observations until time t. The *smoothed state* (x_t^s) is the final state that depends also on all the observations after time t. The smoothing phase works by starting from the last time step and recursively updating x_t^s using x_{t+1}^s, x_{t+1} and x_{t+1}^-. Finally, for all the time steps where there is a gap, the *predicted observations*, $\hat{y}_t^g$, are calculated from the state x_t.
The model always considers the probability distribution for the state ($p(x_t) = \mathcal{N}(x_t; m_t, P_t)$), for each state at each time step the mean (m_t) and the covariance are (P_t) are stored that are the parameters for a multivariate Gaussian distribution. Similarly, the model predictions are a distribution $p(\hat{y}_t)\mathcal{N}(\hat{y}_t; \mu_{y_t}, \Sigma_{y_t})$.

2.1.1 Time update

The first step in a Kalman Filter is computing the probability distribution of the predicted state (x_t^-), from the state at the previous time step (x_{t-1}) and the control variable c_t. The predicted state distribution is $p(x_{t-1}) = \mathcal{N}(m_{t-1}, P_{t-1})$. Using equation 1 and the properties of a linear map of Gaussian distributions the following equation can be derived:

$$p(x_t^-) = \mathcal{N}\left(x_t^-; m_t^-, P_t^-\right) \tag{3}$$
$$m_t^- = Am_{t-1} + Bc_t + d \tag{4}$$
$$P_t^- = AP_{t-1}A^T + Q \tag{5}$$

2.1.2 Measurement update

The predicted probability distribution is the updated to obtain the distribution of the filtered state, using the current observation (y_t). Equation 2 describes the distribution of y_t given x_t, but using Bayes theorem is possible to compute the distribution of x_t given an observation y_t.

$$p(x_t|y_t) = \mathcal{N}(x_t; m_t, P_t) \tag{6}$$
$$z_t = Hm_t^- + d \tag{7}$$
$$S_t = HP_t^-H^T + R \tag{8}$$
$$K_t = P_t^-H^TS_t^{-1} \tag{9}$$
$$m_t = m_t^- + K_t(y_t - z_t) \tag{10}$$
$$P_t = (I - K_tH)P_t^- \tag{11}$$

Missing observations The Kalman Filter is robust to missing data and can update the state even though there is missing data. If all the observations at time t are missing, the measurement update step is skipped and the filtered (x_t) is the same of the predicted state (x_t^-). If only some observations in y_t are missing, then a partial measurement step is performed. The vector containing the observations that are not missing at time t, y_t^{ng}, can be expressed as a linear transformation of y_t

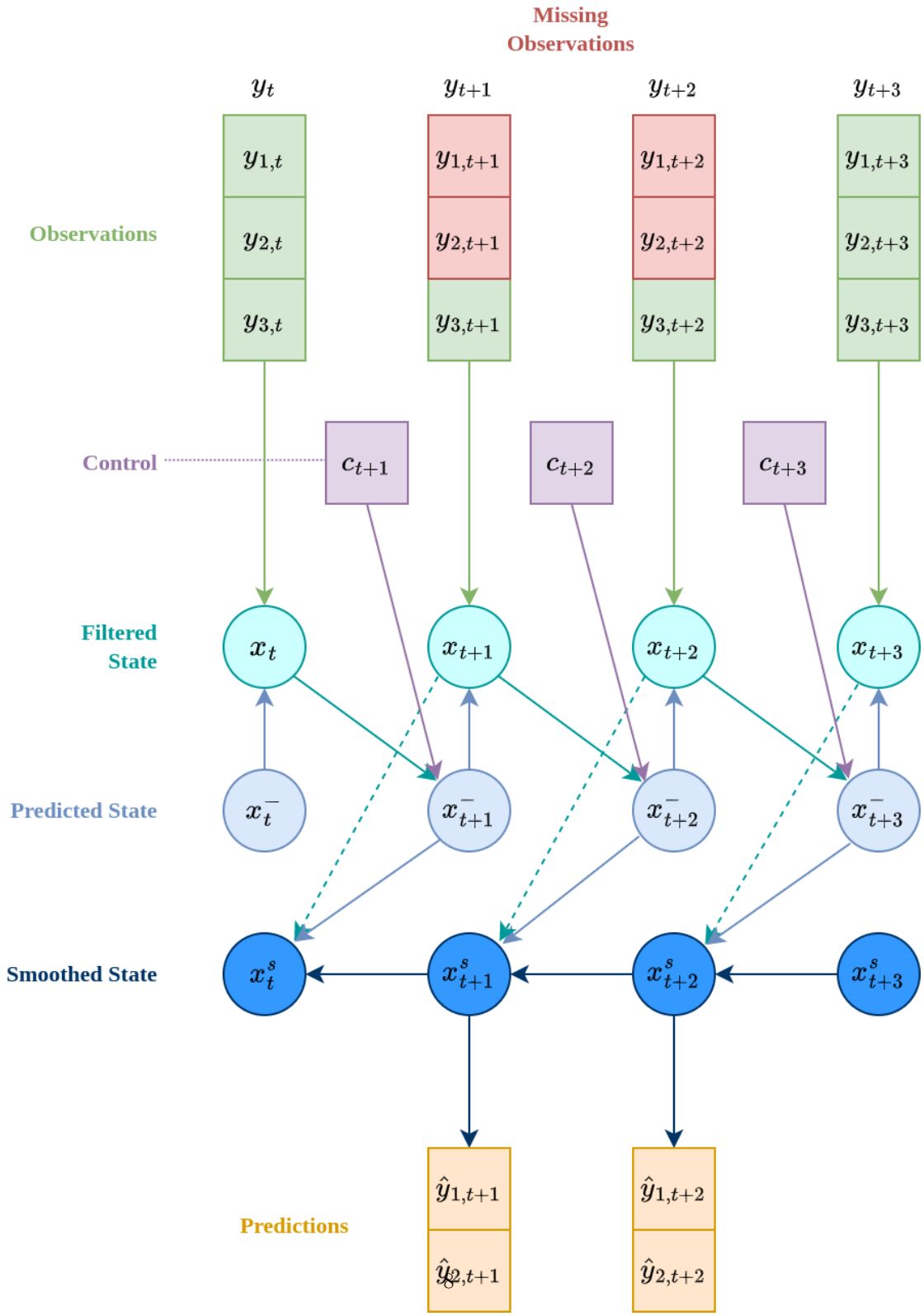

Figure 1: Schematic representation of a Kalman Filter

$$y_t^{ng} = My_t \tag{12}$$

where M is a mask matrix that is used to select the subset of y_t that is observed. $M \in \mathbb{R}^{n^{ng} \times n}$ and is made of rows which are made of all zeros but for an entry 1 at column corresponding to the of the index of the non-missing observation.
For example, if $y_t = [y_{0,t}, y_{1,t}, y_{2,t}]^T$ and $y_{0,t}$ is the missing observation then

$$M = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{13}$$

using the properties of linear projections of Gaussian distribution we can then derive the distribution $p(y_t^{ng} \mid y_t)$ and from it $p(y_t^{ng} \mid x_t)$

$$p(y_t^{ng}|y_t) = \mathcal{N}\left(y_t^{ng}; M\mu_{y_t}, M\Sigma_{y_t}M^T\right) \tag{14}$$
$$p(y_t^{ng}|x_t) = \mathcal{N}\left(y_t^{ng}; MHx_t + Mb, MRM^T\right) \tag{15}$$

Therefore, it is possible to perform the measurement update step when some observations are missing using a variation of equation 6, where H is replaced by MH, b by Mb and R by MRM^T.

2.1.3 Smoothing

In the smoothing step, the filtered state at time t is update using the state $t+1$ corrected by the measurement update. A set of equations for the smoothing pass of a Kalman Filter has been derived by Rauch-Tung-Striebel. They calculate the smoothed state x_t^s from the smoothed, filtered and predicted state at the successive time step. For the last time step, the smoothed state is set to be equal to the filtered state.

$$p(x_t^s \mid Y) = \mathcal{N}\left(x_t^s; m_t^s, P_t^s\right) \tag{16}$$
$$G_t = P_t A^T (P_{t+1}^-)^{-1} \tag{17}$$
$$m_t^s = m_t + G_t(m_{t+1}^s - m_{t+1}^-) \tag{18}$$
$$P_t^s = P_t + G_t(P_{t+1}^s - P_{t+1}^-)G_t^T \tag{19}$$

2.1.4 Predictions

From the state (x_t) it is possible to directly obtain the predictions of the model $\hat{y}_t^g$ by using equation 2 and a mask as described in equation 12

$$p(\hat{y}_t^g) = \mathcal{N}\left(\hat{y}_t^g; \mu_{y_t}, \Sigma_t\right) \tag{20}$$
$$\mu_{y_t} = MHx_t + Md \tag{21}$$
$$\Sigma_{y_t} = MRM + MHP_t^s H^T M^T \tag{22}$$

2.2 Kalman Filter Implementation

2.2.1 Requirements

Kalman Filter are a widely used algorithm and there are several python libraries that implement Kalman Filter (e.g. `statsmodels`, `pykalman`, `filterpy`). However, the application of Kalman Filters in this context we didn't manage to identify a library that meets all the requirement: support gaps and partial measurements updates, use log likelihood for maximization and use a numerically stable implementation of the smoother.
support control variable
Therefore a custom library for Kalman Filters was developed using the PyTorch library, which has the advantage of automatic differentiation, possibility to use GPUs and better integration with other Machine Learning methods.

2.2.2 Numerical stability

Background The direct implementation of the Kalman filter equations suffer by numerically stability issues [13, 6], hence several techniques have been developed to mitigate this.
Numerical instability arises from the fact that digital computers store numbers only with a limited number of decimal digits, which results in a loss of information, so that some operations may be incorrectly performed by a computer (e.g. summing a big number and a small number).
For Kalman Filter the components that are most affect by numerically instability are the covariance matrices. To analyse the stability of the operations on these matrices it is relevant to consider the condition number for inversion [13, 11], which describes if the matrix is going to be singular on the numerical representation in the computer and thus cannot be inverted. The condition number ($k(A)$) is the ratio between the biggest singular value ($\sigma(A)$). $\sigma^2(A) = \lambda(AA^T)$, with $\lambda(A)$ being the eigenvalue of A

$$k(A) = \frac{\sigma_{max}(A)}{\sigma_{min}(A)} \tag{23}$$

The condition number it's 1 for well-conditioned matrices and tends to infinite for ill-conditioned matrices. As a general rule a matrix cannot be inverted when the reciprocal of the condition number for inversion is close to the machine precision $1/k(A) < \varepsilon$ [13].

Mitigation strategies

Machine precision The simplest to improve the numerical stability is to use higher accuracy in the representation of numbers [6]. Practically, this means to use 64bit floats instead of 32bit floats, which is default in PyTorch.

Matrix decomposition Another way to improve the numerical stability is to reduce the condition number of the matrices, for the Kalman filter the key matrix is the state covariance (P). A positive definite matrix has a square root factor, $P^{1/2}$, such as that $P = P^{1/2}(P^{1/2})^T$. If the store $P^{1/2}$ instead of P the effective numerical resolution of the filter can be doubled [11] [6] [18]. This is due to the fact that the eigenvalues of $P^{1/2}$ are the square root of the eigenvalues of P ($\lambda(P) = \lambda^2(P^{1/2})$), thus the conditioning number of P is the square of the conditioning number of $P^{1/2}$, which results in practically doubled. Therefore, if in the filter implementation P is never explicitly computed, the numerical stability of the filter is significantly improved. There are several approached of filter

implementations that follow this approach ([16], [5], [3]) and are generally called "square-root filter". Different implementat which have different tradeoff and often are computationally more expensive that the conventional implementation

2.2.3 Implementation in PyTorch

There are different approaches for square root filtering. According to [13] the best approach is the UD Filter ([3]), since it has the smallest computational cost. However, the filter is based on the UD factorization and a custom matrix factorization [13] and both of those algorithms cannot be efficiently implemented in PyTorch. The PyTorch function `torch.linalg.ldl_factor` performs an UD factorization, but it's an experimental function and is not differentiable. Moreover, the custom matrix factorization would need to be implemented using scalar operations, which aren't efficient with PyTorch eager execution.
For this reason, a square root filter that propagates Cholesky factors of the covariance matrices is implemented. In this way all the required computations can be expressed in QR factorization, which is a numerically stable method and is a routine implemented in PyTorch.

2.2.4 Time update Square Root Filter

The equations for the time update step of the filter are from [13] eq. 6.60.
For notation simplicity, we define $P_t = P_t^{1/2} P_t^{T/2}$ where $P_t^{1/2}$ is lower triangular matrix, and $P_t^{T/2} = (P_t^{1/2})^T$. In the same way, $Q = Q^{1/2} Q^{T/2}$. In the Kalman filter literature $P_t^{1/2}$, is the "square root" of P_t, hence the name of this type of filter. The Cholesky decomposition is an algorithm to find a square root of a matrix, however the Cholesky decomposition calculates only one of possibly many square roots of the matrix. In fact, $P_t^{1/2}$ is not unique.
The goal is to derive from the equations of prediction step (eq. **??**) an algorithm to obtain $P_t^{1/2}$ given $P_{t-1}^{1/2}$, without explicitly computing P_t or P_{t-1}. In this way, the numerical stability of the model is effectively doubled.
If we define

$$W = \begin{bmatrix} AP_{t-1}^{1/2} & Q^{1/2} \end{bmatrix} \tag{24}$$

then using **??** we can prove that

$$WW^T = P_t \tag{25}$$

$$WW^T = \begin{bmatrix} AP_{t-1}^{1/2} & Q^{1/2} \end{bmatrix} \begin{bmatrix} P_{t-1}^{T/2} A^T \\ Q^{T/2} \end{bmatrix}$$
$$= AP_{t-1}^{1/2} P_{t-1}^{T/2} A^T + Q^{1/2} Q^{T/2} = AP_{t-1}A^T + Q = P_t \tag{26}$$

If we can factorize $W = LU$, where L is a lower triangular matrix and U is an orthogonal matrix, such as that $UU^T = I$. Then $WW^T = LU(LU)^T = LUU^T L^T = LL^T$ and by equation 34 $LL^T = P_t$. Hence, L is a square root of P_t, which is what we were looking for.
This procedure never explicitly compute P_t and requires only the factorization of a matrix, which is implemented efficiently and in a numerical stable way by PyTorch `torch.linalg.qr` function.

PyTorch implementation PyTorch doesn't support natively a LU decompositions. It implements the QR factorization: $W = QR$, where Q is an orthogonal matrix and R an upper triangular matrix. This can be easily converted into a LU factorization, since by factorizing W^T then $W^{=}QR = (QR^T) = R^T Q^T$ and R^T is a lower triangular matrix.

Summary The steps of the Square Root time update are:

1. let $W = \begin{bmatrix} AP_{t-1}^{1/2} & Q^{1/2} \end{bmatrix}$
2. do a QR factorization $W^T = TR$
3. define $P_t^{1/2} = R^T$

2.2.5 Measurement update Square Root Filter

A similar procedure can be followed for the measurement update step of the filter. The equations here described are from [6].
The starting point is equation **??** and for simplicity the time subscript are omitted in the following equations
Letting:

$$M = \begin{bmatrix} R^{1/2} & H(P^-)^{1/2} \\ 0 & (P^-)^{1/2} \end{bmatrix} \tag{27}$$

$$V = \begin{bmatrix} S^{1/2} & 0 \\ \bar{K} & P^{1/2} \end{bmatrix} \tag{28}$$

$$\bar{K} = KS^{1/2} \tag{29}$$

Then we can prove that

$$MM^T = UU^T \tag{30}$$

Therefore, if we decompose $M = LU$ then $LL^T = UU^T$ and the bottom left block of size $k \times k$ of L is a square root of P.
Here we prove equation 30

$$\begin{aligned} MM^T &= \begin{bmatrix} R^{1/2} & HP^- \\ 0 & (P^-)^{1/2} \end{bmatrix} \begin{bmatrix} R^{T/2} & 0 \\ (P^-)^{T/2}H^T & (P^-)^{T/2} \end{bmatrix} = \\ &= \begin{bmatrix} R^{1/2}R^{T/2} + H(P^-)^{1/2}(P^-)^{T/2}H^T & HP^-)^{1/2}P^-)^{T/2} \\ (P^-)^{T/2}P^-)^{1/2}H^T & (P^-)^{1/2}P^-)^{T/2} \end{bmatrix} = \\ &= \begin{bmatrix} S & HP^- \\ (P^-)^T H^T & P^- \end{bmatrix} \end{aligned} \tag{31}$$

$$
\begin{aligned}
VV^T &= \begin{bmatrix} S^{1/2} & 0 \\ \bar{K} & P^{1/2} \end{bmatrix} \begin{bmatrix} S^{T/2} & \bar{K}^T \\ 0 & P^{T/2} \end{bmatrix} = \begin{bmatrix} S^{1/2}S^{T/2} & S^{1/2}\bar{K}^T \\ \bar{K}S^{T/2} & \bar{K}\bar{K}^T + P^{1/2}P^{T/2} \end{bmatrix} \\
&= \begin{bmatrix} S & S^{1/2}S^{T/2}K^T \\ KS^{1/2}S^{T/2} & KS^{1/2}S^{T/2}K^T + P \end{bmatrix} \\
&= \begin{bmatrix} S & HP^- \\ P^-H^T & KHP + P \end{bmatrix}
\end{aligned} \tag{32}
$$

We can see that all blocks of VV^T are directly equal to MM^T, but the bottom left one, which is equal due to the measurement update for the covariance (equation 11).

Summary The steps of the Square Root measurement update are:

1. let $M = \begin{bmatrix} R^{1/2} & H(P^-)^{1/2} \\ 0 & (P^-)^{1/2} \end{bmatrix}$
2. do a QR factorization of $M^T = TV$
3. $P^{1/2}$ is the bottom left $k \times k$ block of V^T

2.2.6 Predictions Square Root Filter

The prediction equation for the square root filter are similar to the equations for the time update. We define:

$$
W = \begin{bmatrix} HP_t^{1/2} & R^{1/2} \end{bmatrix} \tag{33}
$$

then using equation **??** we can prove that

$$
WW^T = O_t \tag{34}
$$

$$
\begin{aligned}
WW^T = \begin{bmatrix} AP_t^{1/2} & R^{1/2} \end{bmatrix} \begin{bmatrix} P_t^{T/2}H^T \\ R^{T/2} \end{bmatrix} \\
= HP_t^{1/2}P_t^{T/2}H^T + R^{1/2}R^{T/2} = HP_tH^T + R = O_t
\end{aligned} \tag{35}
$$

Summary The steps of the Square Root predictions are:

1. let $W = \begin{bmatrix} HP_t^{1/2} & R^{1/2} \end{bmatrix}$
2. do a QR factorization of $W^T = TU$
3. define $\Sigma_{y_t}^{1/2} = U^T$

2.2.7 Smoother

The available literature for implementing Square Root Smoothing is scarce compared to Square root filter, so no solution has been identified to implement a square root smoother. Therefore, a standard smoother is employed. Nonetheless, steps were taken to improve the numerical stability of the smoother. The computation in the smoother that is most numerically unstable is the inversion of P_{t+1}^- in equation 17. The direct matrix inversion is avoided by using the `torch.cholesky_solve` function, which solves for X the linear system $P_{t+1}^- X = P_t A$, which is equivalent of computing $X = (P_t A^T (P_{t+1}^-)^-1)^T$. This use directly the Cholesky Factor $(P_{t+1}^-)^{1/2}$ to avoid the computation of P_{t+1}^-. A further step to improve the numerical stability is forcing the covariance matrix to be symmetric, by averaging to upper and lower part at after every time step $P_{t,sym}^s = (P_t^s + (P_t^s)^T)/2$, as suggested in [6]. This approach to numerical stability in the smoother is the same implemented in state of the art libraries, such as 'statsmodels'.

2.3 Model

2.3.1 Parameters

The Kalman Filter is implemented as PyTorch module, whose parameters are described in Table 1. There is no change over time of the parameters, and the state of the filter is initialized always at the same value from the parameters m_0 and P_0.

Constraint An important aspect for implementing a Kalman Filter in PyTorch is constraining the parameters that represents covariance (Q, R and $P0$) to be positive definite. To achieve this goal the optimizer works on a raw parameter, which is then transformed into a positive definite matrix. The transformation into a positive definite matrix is done by transforming the raw parameter into a lower triangular matrix with a positive diagonal. The diagonal is enforced to be positive by transforming the diagonal of the raw parameter with the softplus function (eq. 36), which is positive function. In addition to the diagonal a small offset 1×10^{-5} is added to avoid that the diagonal is close to zero, which would result in positive semi-definite matrix.

$$x = \log(1 + e^x) \tag{36}$$

The inverse of the positive definite transformation is implemented, so that parameters can be manually set.
This implementation of the positive definite constraint makes it is that is straightforward to obtain the Cholesky factor of the parameters, which are needed by the Square Root Filter, and at the same time the full parameter, which are needed by the smoother.

2.3.2 Parameters initialization

The model parameters could be initialized using random values, however random parameters results in an higher change of numerical stability issues and have non-deterministic training. Therefore, parameters are initialized with realist values.

State transition matrix A is initialized using a "local linear trend" model [7]. The idea of a local slope model is that half of the state $x_{l_{t-1}}$ represent the level of the current state and the second

Table 1: Parameters of the Kalman Filter Model. n is the number of dimension of the observations, k the number of dimensions of the state, n_{ctr} the number of dimensions of the control variable.

Parameter name	Notation	Shape	Initial value
State transition matrix	A	$k \times k$	$\begin{bmatrix} I & I \\ 0 & I \end{bmatrix}$
Observation matrix	H	$n \times k$	$\begin{bmatrix} \Lambda & 0 \end{bmatrix}$
State transition covariance	Q	$k \times k$	$0.1I$
Observation covariance	R	$n \times n$	$0.01I$
State transition offset	d	k	0
Observation offset	b	n	0
Control matrix	B	$k \times n_{ctr}$	$\begin{bmatrix} -I & I \\ 0 & 0 \end{bmatrix}$
Initial state mean	m_0	k	0
Initial state covariance	P_0	$k \times k$	$3I$

half the slope $x_{s_{t-1}}$ of a linear function that describes the rate of change of the state between time steps. The next state level is equal to the current level plus the slope and a random change, while the slope remains constant. In this way, the model can also keep track of the previous states.

Observation Matrix H is initialized by using the transpose of the factor loadings matrix of the principal component analysis. This is done to have a latent variable to represent the state between different variables, so that in the presence of a gap. The second part is 0 as in a local trend model the observations don't depend on the slope but only the level of the state

Control matrix B is initialized to the difference between the previous observation and the current observation. The number of dimensions of the control is potentially different from the state and the observations, therefore is difficult to find a proper correspondence between the state and the control. The solution found is to initialize the upper part of the matrix to the difference between the control, which roughly correspond to the variables and then pad the rest with zero to match the correct shape.

Covariance The state transition covariance Q and then observation covariance R are initialized as diagonal matrix with values of 0.1 and 0.01 respectively. This number has been chosen to represent an uncertainty in the state transition that is compatible with standardized variables and low uncertainty in the observations.

Offsets The observation and state transition offsets are initialized to zero.

Initial state The initial state is set to a zero mean and as covariance a matrix with 3 on the diagonal. The number 3 is an arbitrary number significantly bigger than the state transition covariance, which should represent the high level of uncertainty for the initial state.

2.3.3 Loss Function

The loss function is used to train the model is the negative log likelihood, computed for each data point. At each time step the model predicts a multivariate normal distribution $p(\hat{y}_t^g)$, which is used to compute the negative log likelihood given the actual observations y_t^g. The negative log likelihoods between different time steps in the same gaps are summed. Then negative log likelihood is averaged between batches.
The actual loss function of the model should be the log likelihood of the joint distribution $p(Y^g)$. However, the analytical form of the joint distribution cannot to easily derived from the Kalman Filter equations and the sum of the log likelihood of marginal distributions is a lower bound to the log likelihood of the joint distribution. If we define $q(x)$ the predicted joint distribution, $p(x)$ the real joint distribution and $q_i(x)$ the marginal distribution at the ith time step

$$q_i(x) = \int q(x_1, ..., x_k) dx_{\neg i} \tag{37}$$

Then, if the family of distribution of $q(x \mid \theta)$, where θ are the model parameters, is the same of $\prod_i q_i(x \mid \theta)$ then

$$\max_\theta \langle \log q(x \mid \theta) \rangle_{x \sim p(x)} \geq \max_\theta \langle \log \prod_i q_i(x \mid \theta) \rangle_{x \sim p(x)} \tag{38}$$

because $\prod_i q_i(x)$ is more restricted. This means that $q(x)$ can fit at least as good as $\prod_i q_i(x)$. For the Kalman Filter $q_i(x)$ is a Guassian distribution, so $\prod_i q_i(x)$ is also a Guassian distribution and equation 38 is true. The conclusion is that if we can fit the model by maximizing the sum of the log likelihoods at each time steps.

2.3.4 Metrics

The main metric used to assess the model performance is the Root Mean Square Error (RMSE).

$$RMSE = \sqrt{\frac{\sum_i^n (y_i^g - \hat{y}_i^g)^2}{n}} \tag{39}$$

The advantages of the RMSE is that can be used for non-probabilistic methods (e.g. MDS) and that it's value has the same physical dimension of the observed variable. The main drawback is that is cannot be used for comparison between variables, for that the standardized RMSE is used, which is the RMSE computed on the standardized variables.

2.3.5 Performance considerations

The iterative nature of the filter, where the current state depends on the previous state, makes it impossible to use PyTorch vectorization across different time steps. This can significantly limit the performance of the filter, especially when executed on GPUs. In order to mitigate this issue, all functions in the Kalman Filter library support batches, so at every time step different data is processed in parallel.

Table 2: Meteorological variables used to evaluate the Kalman Filter imputation. ERA-I column indicates whether the variable is available in the ERA-Interim dataset

Variable Name	Abbreviation	Unit	ERA-I
Air Temperature	**TA**	°C	✓
Incoming Shortwave Radiation	**SW_IN**	W/m^2	✓
Incoming Longwave Radiation	**LW_IN**	W/m^2	✓
Vapour Pressure Deficit	**VPD**	hPa	✓
Wind Speed	**WS**	m/s	✓
Air Pressure	**PA**	hPa	✓
Precipitation	**P**	mm	✓
Soil Temperature	**TS**	°C	✘
Soil Water Content	**SWC**	%	✘

2.4 Data

2.4.1 Dataset

The data used to evaluate the performance of Kalman Filters is from the Hainich (Germany) site, which is managed by the University of Göttingen. The source of the data is the FLUXNET 2015 Dataset, which includes measurements with a 30 mins frequency between 2000 and 2012 for Hainich. In total XXX observations are available. For simplicity the entire dataset was used for the model training, which includes also gap-filled observations. All the meteorological variables that are gap-filled in the FLUXNET 2015 dataset were selected for the analysis (Table 2).

ERA-Interim The control variables used in the Kalman Filter are the bias corrected and downscaled ERA-I observations included in the FLUXNET dataset. The variables of interest are also present in ERA-I, except for TS and SWC.

Gap distribution The entire FLUXNET 2015 dataset was used to compute the distribution of gap lengths across the all the sites for each variable. A gap was definite when the QC flag of the variable is different from 0 or the data itself is missing (Appendix figure **??**).

MDS Imputation The implementation of the MDS used in the results comparison is from REddyProc ([20]). This package has been used because it provides an R interface, that can be easily integrated with Python. Conversely, ONEFlux implements only a C interface, whose integration in Python is significantly more challenging. The MDS algorithm in REddyProc and ONEFlux are fully equivalent. In detail, the function `REddyProc::sEddyProc_sMDSGapFill` was used, with the defaults setting of using SW_IN, TA an VPD are driver with a tolerance of 2.5 °C, 50 W/m^s and 5 hPa respectively as described in [**reichstein˙separation˙2005**].

2.5 Data processing

2.5.1 Data preparation pipeline

The data needs to be prepared for training. The data preprocessing pipeline starts with an object that contains a) `i` the index of the block b) `shift` the shift c) `var_sel` the variables in the gap d) `gap_len` the gap length. For every item, the pipeline 1) splits the index complete data frame from Hainich into blocks of a given length and selects the ith element 2) add the shift to move the starting point of the data block and select the data from the data frame. For the control variable it also adds the observations adds the observations with a lag 1, so the model can use the difference in the control variable 3) create one continuous artificial gap in the middle of the block for the variables specified in `var_sel` and with a length of `gap_len` 4) convert from Pandas data frame to a PyTorch Tensor 5) Standardize each variable, using the mean (μ_Y) and standard deviation (σ_Y) of the from the whole dataset.

$$y_t^z = \frac{(y_t - \mu_Y)}{\sigma_Y} \tag{40}$$

After this, the tensors are collated into a batch and potentially moved to the GPU.
The list of items to be used in the pipeline are generated by taking all possible block index and the value of the shift is sample from a normal distribution with means 0 and standard deviation 50, the variable is select by sampling one variable from the dataset and the gap length is sampled from a normal distribution between 12 (6 hours) and 336 (1 week). For example, the Hainich FLUXNET dataset has XXX with a a block length of 446 there are XXX data blocks.

2.5.2 Prediction pipeline

The model predicts the mean and the covariance for each time steps for the standardized variables. This needs to be converted back to be scale of the variable to be used for imputation. This operation needs to scale the whole distributions and not only the mean of the prediction. The standardized prediction $\hat{y}_t^z$ is distributed $p(\hat{y}_t^z) = \mathcal{N}\left(\hat{y_t}^z; \mu_{\hat{y}_t}^z, \Sigma_{\hat{y}_t}^z\right)$, the prediction in the original scale $\hat{y}_t$ is distributed $\hat{y}_t \sim \mathcal{N}\left(\hat{y}_t; \mu_{\hat{y}_t}, \Sigma_{\hat{y}_t}\right)$ and $\Sigma_Y = \text{diag}(\sigma_Y)$.
Then from the inverse of equation 40

$$\hat{y}_t = \Sigma_Y \hat{y}_t^z + \mu_Y \tag{41}$$

Then using the properties of the linear projections of Gaussian distributions

$$p(\hat{y}_t) = \mathcal{N}\left(\hat{y}_t; \Sigma_Y \mu_{\hat{y}_t}^z + \mu_Y, \Sigma_Y \Sigma_{\hat{y}_t}^z \Sigma_Y^T\right) \tag{42}$$

2.6 Model Training

State is double of observed

Generic model The first model to be trained is a generic model, where each data block as a gap in one variable with the length of sampled from a uniform distribution between 12 (6 hours) and 336 (1 week). The missing variable is sampled from the list of all variables. For each block of data in the original data frame 10 different artificial gaps were created, resulting in a total of XXX data

blocks used for training an XXX for validation. The length of the block of data is 446, so that at least 50 observations are available to the model before and after the gap. The batch size is 20. The model was trained for 3 epochs with a learning rate of 1×10^{-3}.

Variable fine-tuning The generic model has been fine tune for each variable, resulting in 9 different models. The training settings are the same for the generic model, expect that the gaps is only in one variable and then number of repetitions for each is 5 (training XXX blocks validation XXX blocks). Each variable, was fine tuned with a different number of epochs depending on the variable: TA 4 epochs, SW_IN 4 epochs, LW_IN2 epochs, VPD 2 epochs, WS 2 epochs, PA 3 epochs, P 0 epochs, TS 4 epochs, SWC 4 epochs. The learning was manually stopped when the training loss started being constant or the validation loss started to increase.

Gap all variables A version of the model was trained with a gap in all variables. The length of the gap was 30 for numerical stability issues. For gaps in all variables with a length of more than 30, the predicted covariance becomes not positive definite, hence is impossible to compute the log likelihood. The model was trained from the beginning for 3 epochs with a learning rate of 1×10^{-3}

No Control The last version of the model is one where the use of the control variables was disabled. The generic model was fine-tuned for 3 epochs with a learning rate of 1×10^{-3}.

2.7 Code Details and Availability

The core PyTorch module has been developed and the data preparation using FastAI. The plots use Altair . The source code is available at https://github.com/mone27/meteo_imp and the documentation of the library at https://mone27.github.io/meteo_imp/libs
Details model training available at ...

3 Results

3.1 Variables Correlation

- 3 variables (WS, PA, P) have a very correlation with other variables
- TA is the variables that has the highest correlation with other variables. TS similar to TA
- `SWC` correlated only temperature
- Other variables (`SW_IN` `LW_IN`, `VPD`) have a correlation ranging between .4 and .6 with at least two other variables

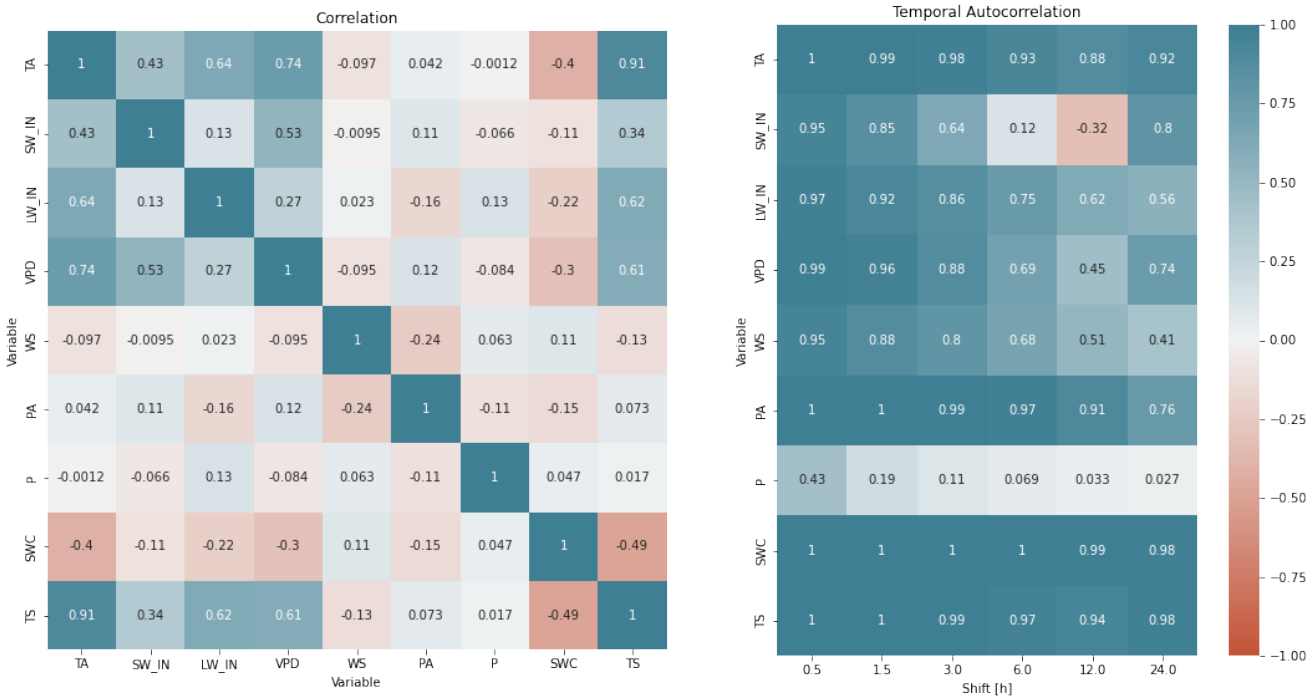

Figure 2: Correlation and Temporal autocorrelation of variables. Subfigure (a) shows the correlation between the meteorological variables. Abbreviations: Air Temperature `TA`, Incoming Shortwave Radiation `SW_IN`, Incoming Longwave Radiation `LW_IN`, Vapour Pressure Deficit `VPD`, Wind Speed `WS`, Air Pressure `PA`, Precipitation `P`, Soil Temperature `TS`, Soil Water Content `SWC`

3.2 Comparison state of art

-

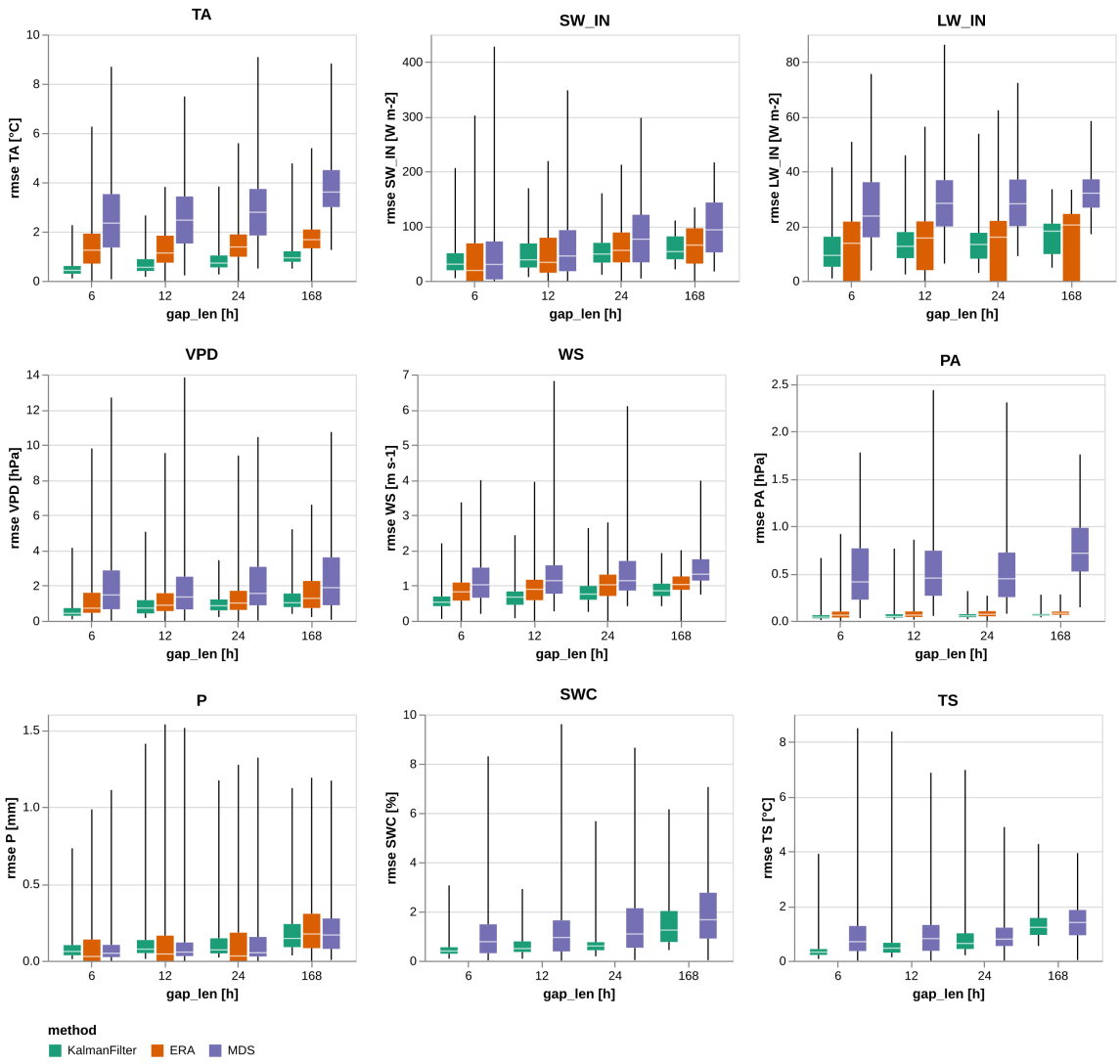

Figure 3: Box plot to compare Root Mean Square Error(RMSE) for each variable between the different methods: Kalman Filter and the state of the art methods ERA and MDS. Each box is delimitated by the first and third quartile, the white mark is the median and then vertical lines extend from the minimum to the maximum. For each variable and each gap length 400 artificial gaps has been made only in the variable of interest and the gap imputed using the different methods. The Kalman Filter model has been fine tuned to each variable.

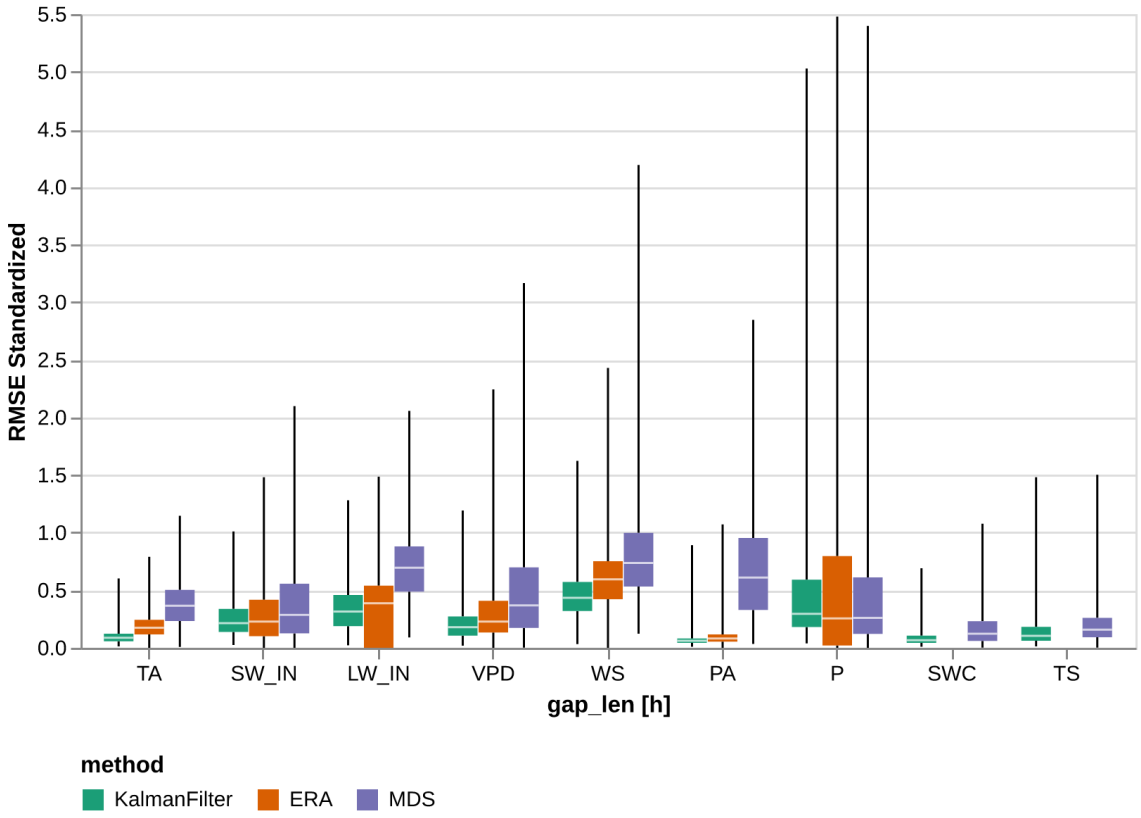

Figure 4: Box plot to compare Standardized Root Mean Square Error(RMSE) for each variable between the different methods: Kalman Filter and the state of the art methods ERA and MDS. The same data from figure 4 has been aggregated for all gap lengths

Table 3: RMSE Comparison imputation methods. The best method for each gap length is highligthed in bold

		KalmanFilter		ERA		MDS	
	RMSE	mean	std	mean	std	mean	std
Variable	Gap [h]						
TA [C]	6	**0.510**	0.335	1.443	1.030	2.624	1.677
	12	**0.685**	0.412	1.347	0.788	2.665	1.546
	24	**0.898**	0.549	1.460	0.730	3.024	1.717
	168	**1.078**	0.561	1.765	0.714	3.803	1.232
SW_IN [W/m^2]	6	42.053	36.503	**41.878**	57.449	56.796	79.190
	12	**49.334**	33.321	51.587	48.850	63.173	63.886
	24	**54.067**	26.651	61.879	38.392	86.693	63.036
	168	**59.130**	24.263	66.392	33.962	99.610	53.139
LW_IN [W/m^2]	6	**11.647**	8.194	14.299	13.000	27.309	15.357
	12	**14.098**	8.483	15.719	12.606	29.660	14.269
	24	**13.846**	7.289	13.970	11.779	29.612	13.066
	168	16.756	6.761	**15.921**	11.370	32.895	8.250
VPD [hPa]	6	**0.575**	0.496	1.356	1.546	2.200	2.280
	12	**0.897**	0.749	1.352	1.497	2.027	2.192
	24	**1.014**	0.632	1.359	1.172	2.168	1.915
	168	**1.285**	0.799	1.625	1.111	2.400	1.898
WS [m/s]	6	**0.585**	0.303	0.894	0.491	1.194	0.667
	12	**0.702**	0.343	0.956	0.595	1.279	0.808
	24	**0.857**	0.385	1.049	0.510	1.369	0.794
	168	**0.912**	0.278	1.063	0.322	1.454	0.508
PA [hPa]	6	**0.052**	0.068	0.077	0.095	0.520	0.387
	12	**0.055**	0.056	0.080	0.072	0.573	0.434
	24	**0.057**	0.029	0.078	0.043	0.550	0.406
	168	**0.065**	0.026	0.083	0.030	0.770	0.338
P [mm]	6	**0.100**	0.120	0.105	0.170	0.101	0.150
	12	0.125	0.161	0.131	0.229	**0.119**	0.193
	24	0.139	0.177	0.138	0.220	**0.138**	0.217
	168	**0.197**	0.171	0.223	0.197	0.215	0.188
SWC [%]	6	**0.481**	0.360	nan	nan	1.150	1.287
	12	**0.605**	0.395	nan	nan	1.380	1.629
	24	**0.764**	0.760	nan	nan	1.665	1.714
	168	**1.508**	0.982	nan	nan	2.023	1.493
TS [C]	6	**0.403**	0.457	nan	nan	0.983	1.113
	12	**0.613**	0.764	nan	nan	0.975	0.844
	24	**0.853**	0.767	nan	nan	0.977	0.817
	168	**1.358**	0.697	nan	nan	1.507	0.829

Table 4: StandardizedRMSE Comparison imputation methods. The best method for each gap length is highligthed in bold

Variable	RMSE Gap [h]	KalmanFilter mean	KalmanFilter std	ERA mean	ERA std	MDS mean	MDS std
TA [C]	6	**0.064**	0.042	0.182	0.130	0.331	0.212
	12	**0.086**	0.052	0.170	0.099	0.336	0.195
	24	**0.113**	0.069	0.184	0.092	0.382	0.217
	168	**0.136**	0.071	0.223	0.090	0.480	0.155
SW_IN [W/m^2]	6	0.206	0.179	**0.205**	0.282	0.278	0.388
	12	**0.242**	0.163	0.253	0.239	0.310	0.313
	24	**0.265**	0.131	0.303	0.188	0.425	0.309
	168	**0.290**	0.119	0.325	0.166	0.488	0.260
LW_IN [W/m^2]	6	**0.278**	0.195	0.341	0.310	0.651	0.366
	12	**0.336**	0.202	0.375	0.300	0.707	0.340
	24	**0.330**	0.174	0.333	0.281	0.706	0.311
	168	0.399	0.161	**0.379**	0.271	0.784	0.197
VPD [hPa]	6	**0.132**	0.113	0.310	0.354	0.504	0.522
	12	**0.205**	0.171	0.310	0.343	0.464	0.502
	24	**0.232**	0.145	0.311	0.268	0.496	0.438
	168	**0.294**	0.183	0.372	0.254	0.549	0.435
WS [m/s]	6	**0.360**	0.186	0.550	0.302	0.734	0.411
	12	**0.432**	0.211	0.588	0.366	0.787	0.497
	24	**0.527**	0.237	0.645	0.314	0.842	0.488
	168	**0.561**	0.171	0.654	0.198	0.895	0.312
PA [hPa]	6	**0.060**	0.080	0.090	0.111	0.608	0.453
	12	**0.065**	0.065	0.093	0.085	0.670	0.507
	24	**0.066**	0.034	0.091	0.051	0.643	0.474
	168	**0.077**	0.030	0.097	0.036	0.900	0.395
P [mm]	6	**0.359**	0.430	0.374	0.608	0.359	0.534
	12	0.447	0.574	0.466	0.817	**0.425**	0.689
	24	0.497	0.632	0.493	0.785	**0.492**	0.773
	168	**0.702**	0.610	0.795	0.701	0.767	0.671
SWC [%]	6	**0.054**	0.040	nan	nan	0.129	0.144
	12	**0.068**	0.044	nan	nan	0.155	0.183
	24	**0.086**	0.085	nan	nan	0.187	0.192
	168	**0.169**	0.110	nan	nan	0.227	0.168
TS [C]	6	**0.071**	0.081	nan	nan	0.174	0.197
	12	**0.108**	0.135	nan	nan	0.172	0.149
	24	**0.151**	0.136	nan	nan	0.173	0.144
	168	**0.240**	0.123	nan	nan	0.266	0.146

3.2.1 Example Time series

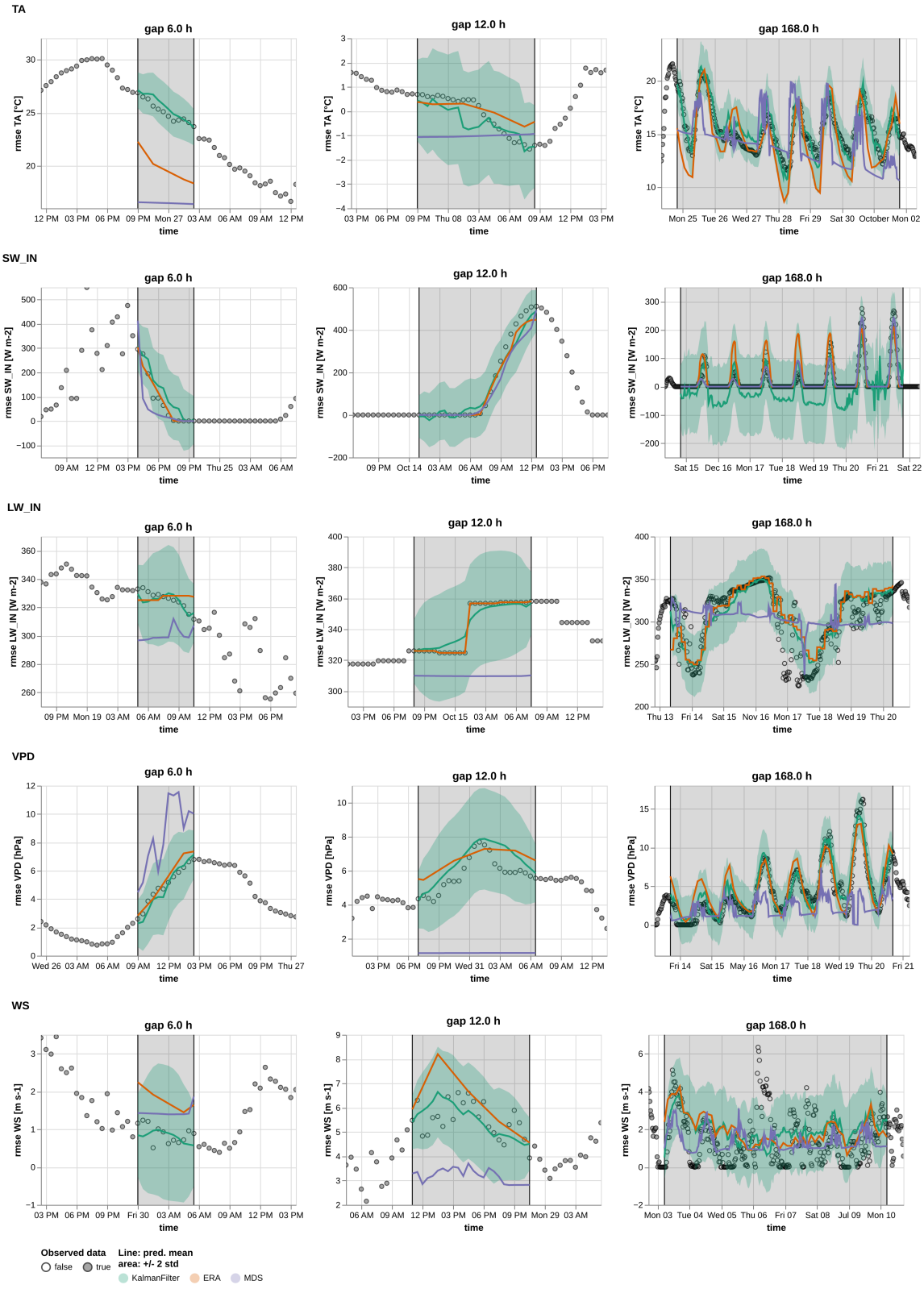

Figure 5: Timeseries 1

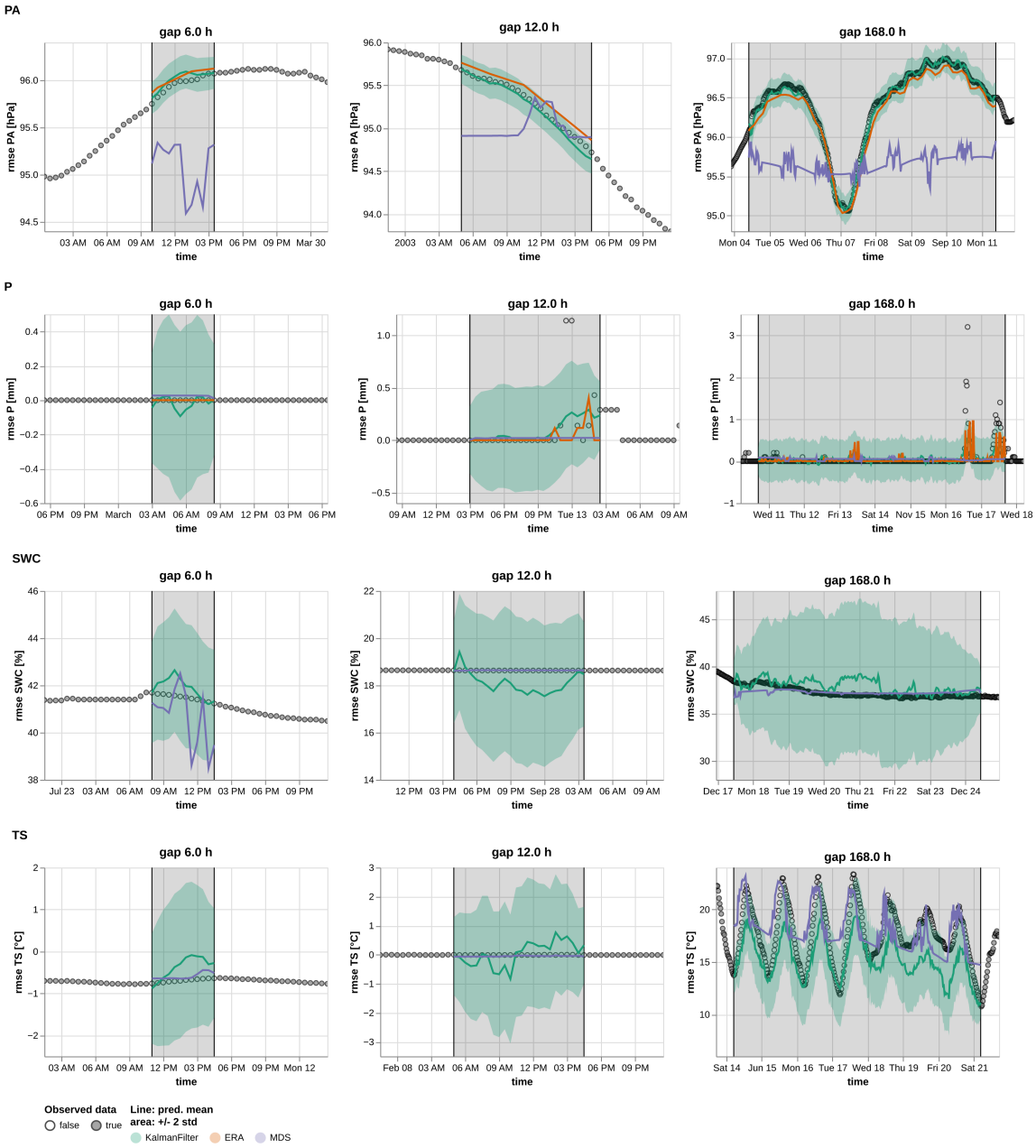

Figure 6: Timeseries 2

3.3 Analysis Kalman Filter

3.3.1 Gap Length

3.3.2 Gap other variable

3.3.3 Control variable

3.3.4 Generic model

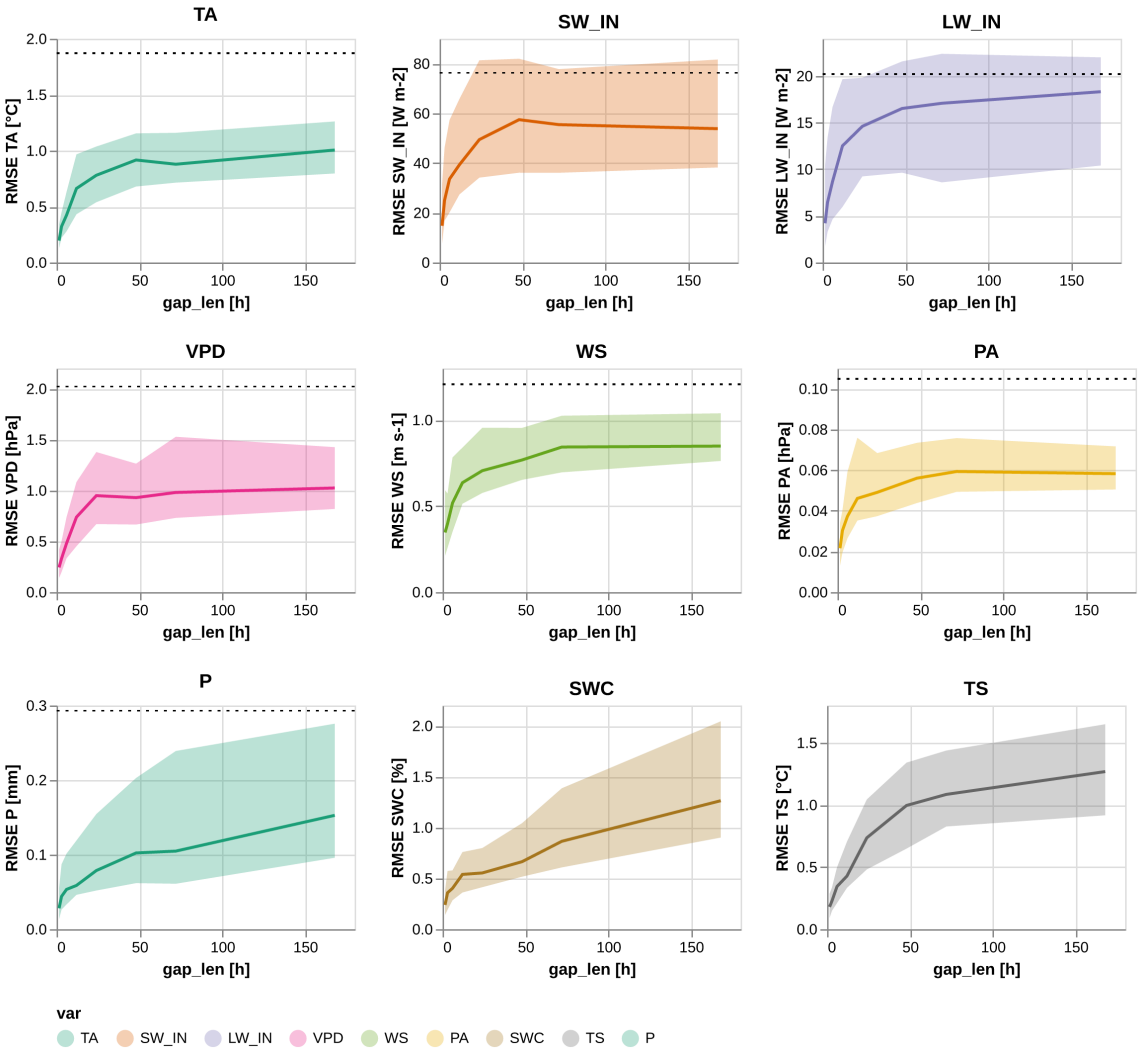

Figure 7: Gap len

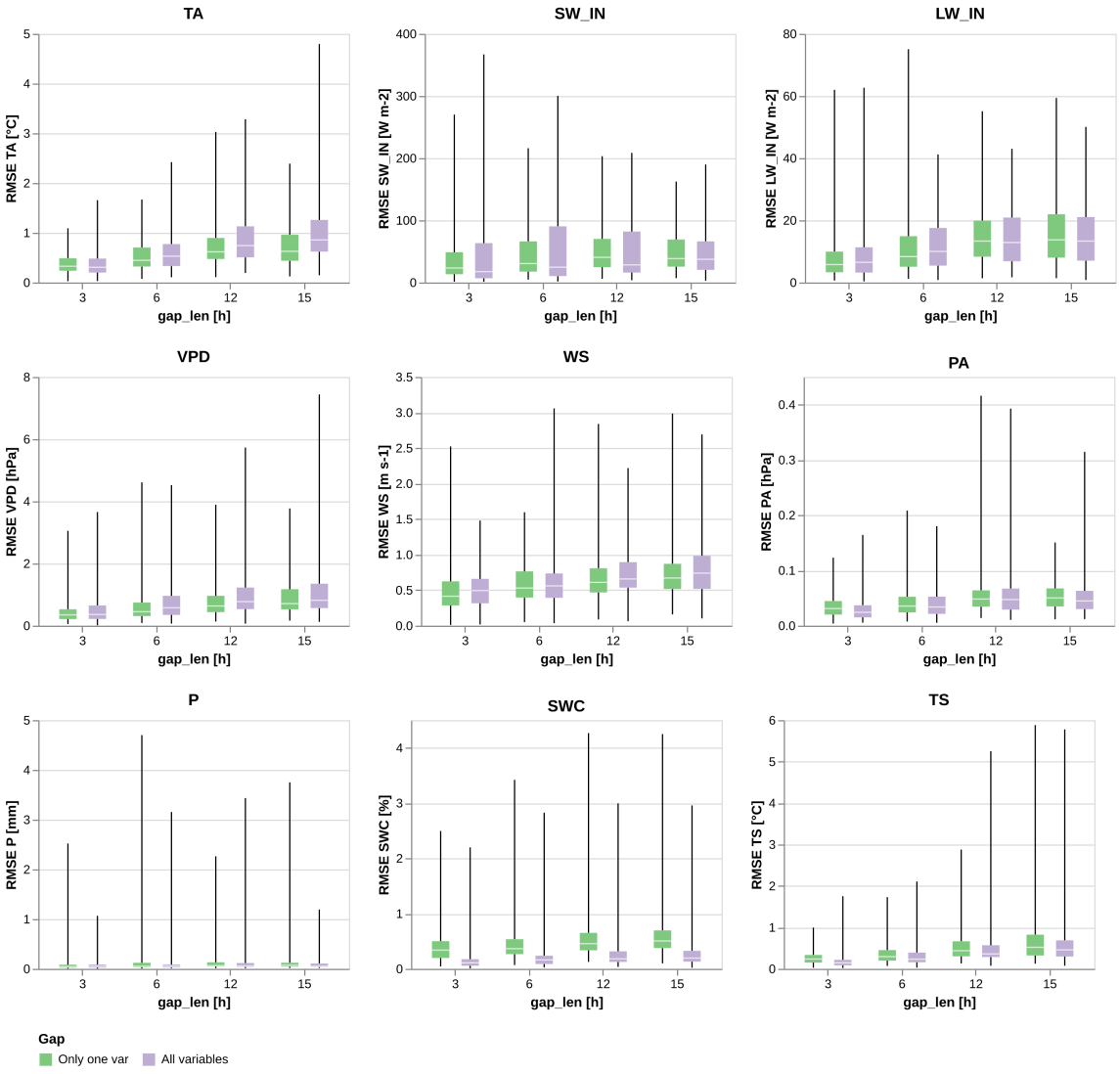

Figure 8: Gap single var

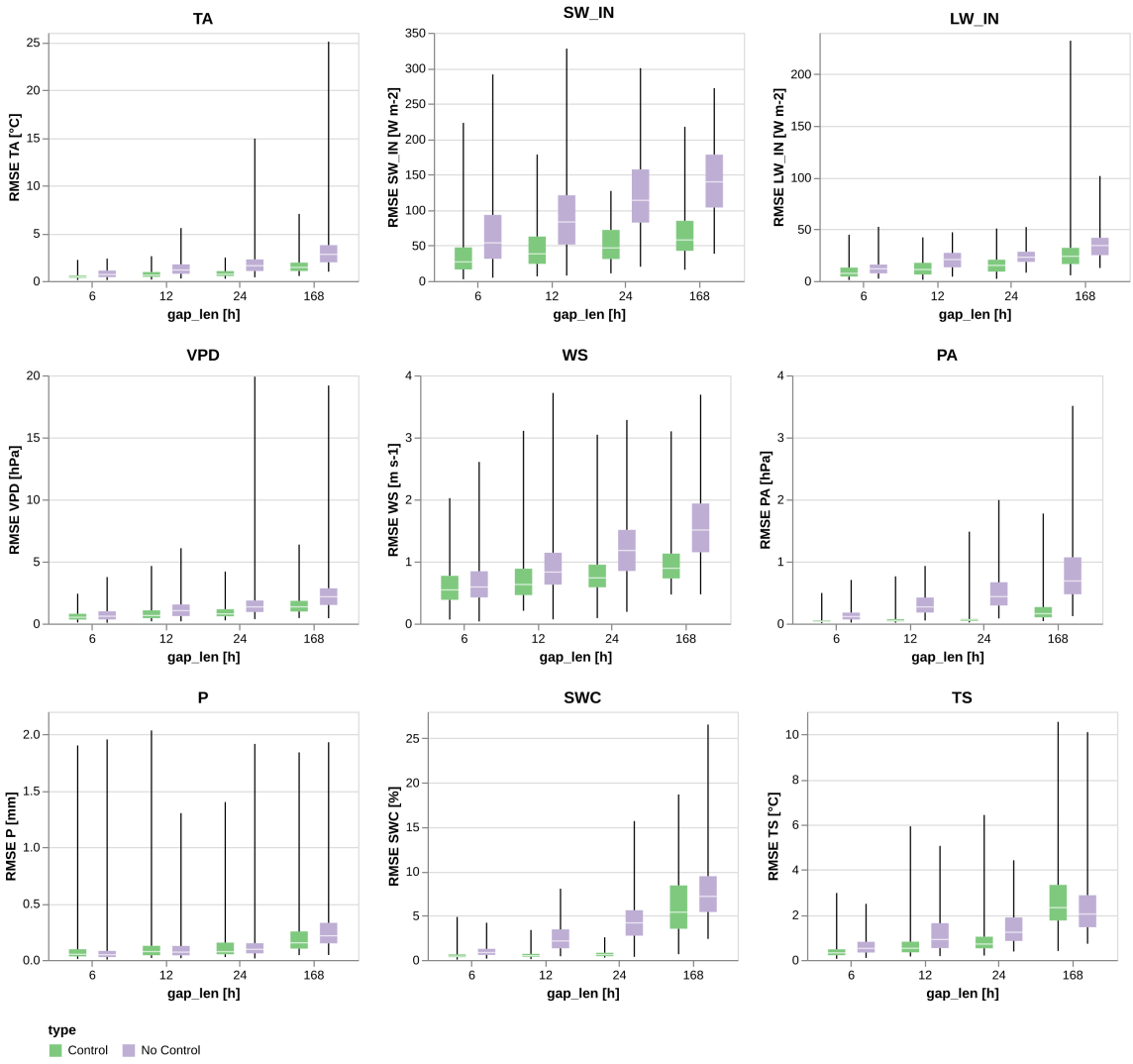

Figure 9: Control

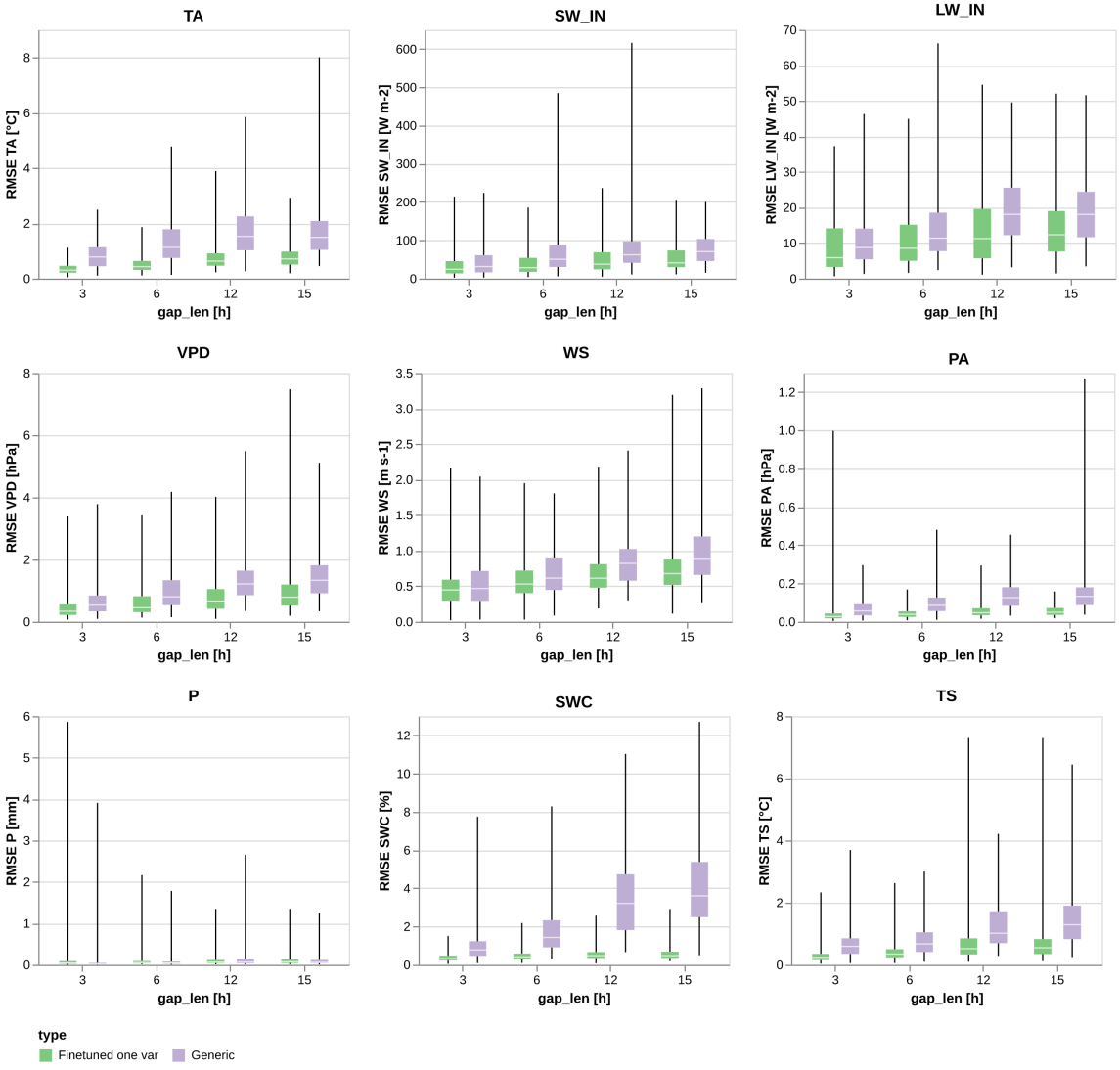

Figure 10: Generic

4 Discussion

4.1 Comparison other methods

- Kalman Filter works much better than MDS as it can use observation before/after
- overall low correlation between meteo variables -¿ MDS is limited
- more precise than ERA as uses local conditions
- problem with precipitation
- SW IN poor performance?

4.2 Kalman Filter Components

- as expected the longer gap length the worse the performance
- the standard deviation of the increases with gap len
- the presence of other variables in the gap doesn't help a lot for gap filling
- exception is TA and LW IN, which have higher correlation
- control ?

4.3 Impact of numerical stability

- max gap len is 15 hours model can be trained
- Kalman filter is a local model, so for long gaps can't really use it
- still need to work to improve numerical stability which should make this better (see appendix)

4.4 Future steps

4.4.1 data

- train model for different sites and see difference between sites
- use ERA-5 Land data

4.4.2 Gap filling quality assessment

- can train with more realistic gaps properties (length/other variables missing/time of day)
- better analysis of gaps in fluxnet

4.4.3 Model improvement

- numerical stability
- make a model to predict the parameters of the Kalman Filter depending on the conditions
- use Neural Network to process control variable
- training and understand variables that are weird

5 Conclusions

References

References

[1] Marc Aubinet, Timo Vesala, and Dario Papale, eds. *Eddy Covariance: A Practical Guide to Measurement and Data Analysis*. Dordrecht: Springer Netherlands, 2012. ISBN: 978-94-007-2350-4 978-94-007-2351-1. DOI: 10.1007/978-94-007-2351-1. URL: https://link.springer.com/10.1007/978-94-007-2351-1 (visited on 02/10/2023).

[2] M. Balzarolo et al. "Evaluating the Potential of Large-Scale Simulations to Predict Carbon Fluxes of Terrestrial Ecosystems over a European Eddy Covariance Network". In: *Biogeosciences* 11.10 (May 20, 2014), pp. 2661–2678. ISSN: 1726-4170. DOI: 10.5194/bg-11-2661-2014. URL: https://bg.copernicus.org/articles/11/2661/2014/ (visited on 02/10/2023).

[3] Gerald J. Bierman and Catherine L. Thornton. "Numerical Comparison of Kalman Filter Algorithms: Orbit Determination Case Study". In: *Automatica* 13.1 (Jan. 1, 1977), pp. 23–35. ISSN: 0005-1098. DOI: 10.1016/0005-1098(77)90006-1. URL: https://www.sciencedirect.com/science/article/pii/0005109877900061 (visited on 01/12/2023).

[4] Gordon B. Bonan et al. "Improving Canopy Processes in the Community Land Model Version 4 (CLM4) Using Global Flux Fields Empirically Inferred from FLUXNET Data". In: *Journal of Geophysical Research: Biogeosciences* 116.G2 (2011). ISSN: 2156-2202. DOI: 10.1029/2010JG001593. URL: https://onlinelibrary.wiley.com/doi/abs/10.1029/2010JG001593 (visited on 02/10/2023).

[5] NEAL A. CARLSON. "Fast Triangular Formulation of the Square Root Filter." In: *AIAA Journal* 11.9 (1973), pp. 1259–1265. ISSN: 0001-1452. DOI: 10.2514/3.6907. URL: https://doi.org/10.2514/3.6907 (visited on 02/15/2023).

[6] Dan Simon. *Optimal State Estimation Kalman, H and Nonlinear Approaches*. 2006. ISBN: 100-47 1-70858-5.

[7] James Durbin and Siem Jan Koopman. *Time Series Analysis by State Space Methods*. Mar. 2, 2012. URL: https://doi.org/10.1093/acprof:oso/9780199641178.001.0001.

[8] Vincent Fortuin et al. *GP-VAE: Deep Probabilistic Time Series Imputation*. Feb. 20, 2020. DOI: 10.48550/arXiv.1907.04155. arXiv: 1907.04155 [cs, stat]. URL: http://arxiv.org/abs/1907.04155 (visited on 07/28/2022).

[9] Andrew D. Friend et al. "FLUXNET and Modelling the Global Carbon Cycle". In: *Global Change Biology* 13.3 (2007), pp. 610–633. ISSN: 1365-2486. DOI: 10.1111/j.1365-2486.2006.01223.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2486.2006.01223.x (visited on 02/10/2023).

[10] Peter Isaac et al. "OzFlux Data: Network Integration from Collection to Curation". In: *Biogeosciences* 14.12 (June 19, 2017), pp. 2903–2928. ISSN: 1726-4170. DOI: 10.5194/bg-14-2903-2017. URL: https://bg.copernicus.org/articles/14/2903/2017/ (visited on 05/12/2022).

[11] P. Kaminski, A. Bryson, and S. Schmidt. "Discrete Square Root Filtering: A Survey of Current Techniques". In: *IEEE Transactions on Automatic Control* 16.6 (Dec. 1971), pp. 727–736. ISSN: 1558-2523. DOI: 10.1109/TAC.1971.1099816.

[12] K. Kramer et al. "Evaluation of Six Process-Based Forest Growth Models Using Eddy-Covariance Measurements of CO2 and H2O Fluxes at Six Forest Sites in Europe". In: *Global Change Biology* 8.3 (2002), pp. 213–230. ISSN: 1365-2486. DOI: 10.1046/j.1365-2486.2002.00471.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1046/j.1365-2486.2002.00471.x (visited on 01/18/2023).

[13] Mohinder S. Grewal and Angus P. Andrews. *Kalman Filtering: Theory and Practice Using MATLAB, Second Edition*. 2001. ISBN: 0-471-26638-8.

[14] Dario Papale. "Ideas and Perspectives: Enhancing the Impact of the FLUXNET Network of Eddy Covariance Sites". In: *Biogeosciences* 17.22 (Nov. 17, 2020), pp. 5587–5598. ISSN: 1726-4189. DOI: 10.5194/bg-17-5587-2020. URL: https://bg.copernicus.org/articles/17/5587/2020/ (visited on 01/18/2023).

[15] Gilberto Pastorello et al. "The FLUXNET2015 Dataset and the ONEFlux Processing Pipeline for Eddy Covariance Data". In: *Scientific Data* 7.1 (1 July 9, 2020), p. 225. ISSN: 2052-4463. DOI: 10.1038/s41597-020-0534-3. URL: https://www.nature.com/articles/s41597-020-0534-3 (visited on 05/12/2022).

[16] JAMES POTTER and ROBERT STERN. "STATISTICAL FILTERING OF SPACE NAVIGATION MEASUREMENTS". In: *Guidance and Control Conference*. American Institute of Aeronautics and Astronautics, 1963. DOI: 10.2514/6.1963-333. URL: https://arc.aiaa.org/doi/abs/10.2514/6.1963-333 (visited on 01/13/2023).

[17] Markus Reichstein et al. "On the Separation of Net Ecosystem Exchange into Assimilation and Ecosystem Respiration: Review and Improved Algorithm". In: *Global Change Biology* 11.9 (2005), pp. 1424–1439. ISSN: 1365-2486. DOI: 10.1111/j.1365-2486.2005.001002.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2486.2005.001002.x (visited on 05/12/2022).

[18] Mark G. Rutten. "Square-Root Unscented Filtering and Smoothing". In: *2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing*. 2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing. Apr. 2013, pp. 294–299. DOI: 10.1109/ISSNIP.2013.6529805.

[19] N. Vuichard and D. Papale. "Filling the Gaps in Meteorological Continuous Data Measured at FLUXNET Sites with ERA-Interim Reanalysis". In: *Earth System Science Data* 7.2 (July 13, 2015), pp. 157–171. ISSN: 1866-3508. DOI: 10.5194/essd-7-157-2015. URL: https://essd.copernicus.org/articles/7/157/2015/ (visited on 05/11/2022).

[20] Thomas Wutzler et al. "Basic and Extensible Post-Processing of Eddy Covariance Flux Data with REddyProc". In: *Biogeosciences* 15.16 (Aug. 23, 2018), pp. 5015–5030. ISSN: 1726-4170. DOI: 10.5194/bg-15-5015-2018. URL: https://bg.copernicus.org/articles/15/5015/2018/ (visited on 05/15/2022).

[21] Byron M Yu et al. "Gaussian-Process Factor Analysis for Low-Dimensional Single-Trial Analysis of Neural Population Activity". In: *Advances in Neural Information Processing Systems*. Vol. 21. Curran Associates, Inc., 2008. URL: https://proceedings.neurips.cc/paper/2008/hash/ad972f10e0800b49d76fed33a21f6698-Abstract.html (visited on 07/07/2022).

[22] Y. Zhao et al. "How Errors on Meteorological Variables Impact Simulated Ecosystem Fluxes: A Case Study for Six French Sites". In: *Biogeosciences* 9.7 (July 11, 2012), pp. 2537–2564. ISSN: 1726-4170. DOI: 10.5194/bg-9-2537-2012. URL: https://bg.copernicus.org/articles/9/2537/2012/ (visited on 02/13/2023).

A Additional Results

A.1 Gap length distribution in FLUXNET

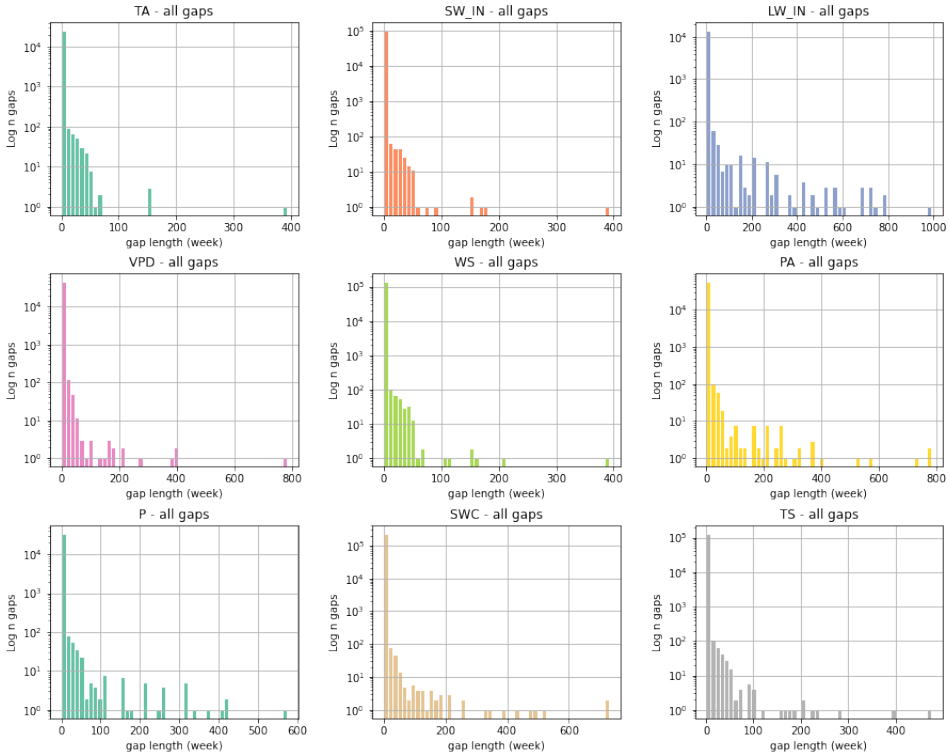

Figure 11: FLUXNET gap len

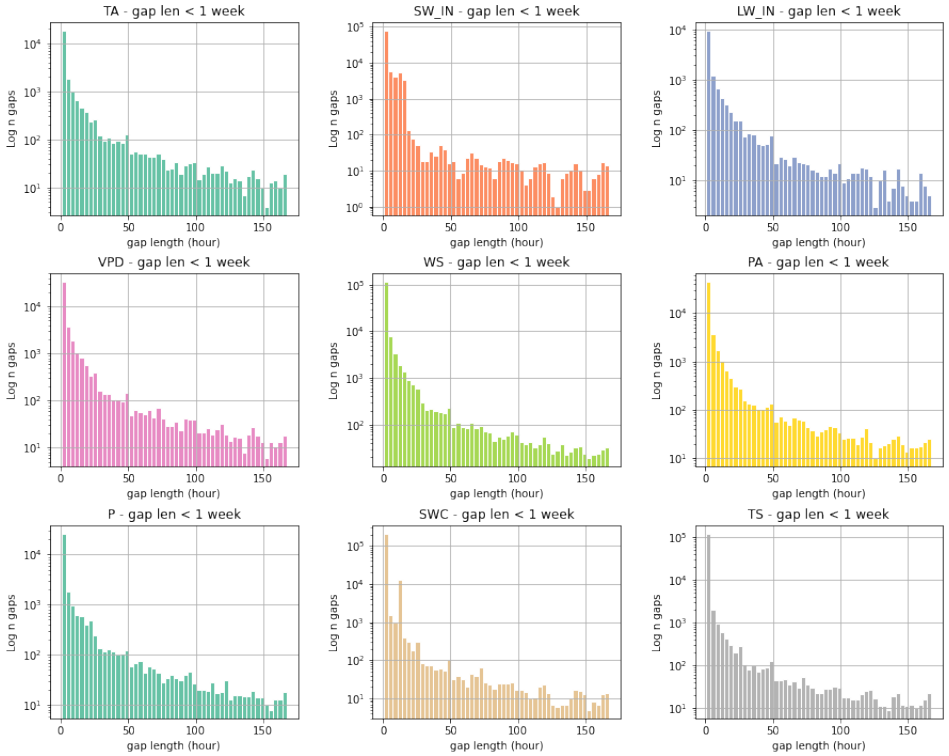

Figure 12: FLUXNET gap len less than a week

Table 5: RMSE Comparison Kalman filter with and without control variable. The best result for each for each gap length is highligthed in bold

	type	Control		No Control	
	RMSE	mean	std	mean	std
Variable	Gap [h]				
TA [C]	6	**0.510**	0.297	0.829	0.513
	12	**0.736**	0.448	1.356	0.854
	24	**0.842**	0.456	1.947	1.514
	168	**1.617**	0.903	3.331	2.666
SW_IN [W/m^2]	6	**40.120**	39.130	70.388	54.538
	12	**49.898**	33.761	93.560	56.716
	24	**54.603**	28.246	119.991	55.637
	168	**66.438**	32.454	140.487	49.705
LW_IN [W/m^2]	6	**9.785**	7.658	13.286	7.984
	12	**12.705**	8.150	20.720	8.854
	24	**15.656**	8.584	23.654	8.217
	168	**27.743**	21.584	35.018	13.047
VPD [hPa]	6	**0.618**	0.399	0.756	0.586
	12	**0.840**	0.638	1.238	0.930
	24	**1.021**	0.703	1.590	1.594
	168	**1.542**	0.911	2.523	2.126
WS [m/s]	6	**0.606**	0.327	0.654	0.343
	12	**0.748**	0.440	0.953	0.534
	24	**0.809**	0.370	1.253	0.567
	168	**0.982**	0.391	1.613	0.606
PA [hPa]	6	**0.043**	0.038	0.135	0.090
	12	**0.062**	0.064	0.319	0.195
	24	**0.074**	0.115	0.522	0.323
	168	**0.231**	0.236	0.846	0.531
P [mm]	6	0.106	0.178	**0.091**	0.173
	12	**0.128**	0.185	0.138	0.210
	24	**0.151**	0.222	0.169	0.244
	168	**0.213**	0.219	0.276	0.247
SWC [%]	6	**0.548**	0.493	1.032	0.706
	12	**0.597**	0.428	2.587	1.728
	24	**0.733**	0.448	4.603	2.759
	168	**6.252**	3.772	7.719	3.386
TS [C]	6	**0.427**	0.385	0.619	0.419
	12	**0.662**	0.587	1.192	0.872
	24	**0.860**	0.658	1.508	0.927
	168	2.771	1.647	**2.301**	1.238

Variable	Gap RMSE Gap [h]	All variables mean	All variables std	Only one var mean	Only one var std
TA [C]	3	**0.373**	0.256	0.379	0.208
	6	0.632	0.425	**0.532**	0.296
	12	0.907	0.566	**0.716**	0.379
	15	1.033	0.649	**0.774**	0.450
SW_IN [W/m^2]	3	46.737	60.343	**41.845**	46.964
	6	57.614	64.037	**48.103**	43.868
	12	**52.908**	51.094	52.938	40.110
	15	**48.308**	38.911	52.844	36.699
LW_IN [W/m^2]	3	8.982	8.190	**8.431**	8.249
	6	11.893	7.856	**10.861**	8.483
	12	**14.816**	9.212	14.946	9.448
	15	**15.096**	9.149	15.320	9.744
VPD [hPa]	3	0.534	0.539	**0.450**	0.376
	6	0.779	0.680	**0.602**	0.538
	12	0.962	0.730	**0.766**	0.513
	15	1.119	0.903	**0.901**	0.575
WS [m/s]	3	0.515	0.259	**0.473**	0.290
	6	0.627	0.411	**0.605**	0.302
	12	0.755	0.358	**0.699**	0.391
	15	0.811	0.419	**0.736**	0.338
PA [hPa]	3	**0.029**	0.020	0.035	0.021
	6	**0.041**	0.027	0.043	0.029
	12	0.055	0.043	**0.055**	0.040
	15	**0.053**	0.035	0.054	0.026
P [mm]	3	**0.083**	0.114	0.103	0.225
	6	**0.106**	0.270	0.175	0.417
	12	0.145	0.314	**0.128**	0.213
	15	**0.121**	0.189	0.159	0.342
SWC [%]	3	**0.182**	0.289	0.396	0.304
	6	**0.238**	0.335	0.489	0.440
	12	**0.305**	0.371	0.591	0.476
	15	**0.338**	0.425	0.636	0.493
TS [C]	3	**0.185**	0.176	0.270	0.172
	6	**0.311**	0.264	0.383	0.289
	12	0.566	0.710	**0.561**	0.415
	15	**0.668**	0.743	0.684	0.642

Variable	type RMSE Gap [h]	Finetuned one var mean	 std	Generic mean	 std
TA [C]	3	**0.361**	0.218	0.879	0.556
	6	**0.516**	0.301	1.319	0.817
	12	**0.778**	0.491	1.725	0.948
	15	**0.782**	0.386	1.649	0.929
SW_IN [W/m^2]	3	**40.067**	41.468	46.200	43.903
	6	**41.876**	36.044	64.059	50.976
	12	**51.764**	36.886	74.468	55.727
	15	**52.965**	32.700	77.988	40.209
LW_IN [W/m^2]	3	**9.404**	8.212	10.940	7.912
	6	**10.981**	8.009	14.277	9.797
	12	**13.791**	9.894	19.098	9.107
	15	**14.679**	9.597	19.144	8.941
VPD [hPa]	3	**0.455**	0.407	0.719	0.614
	6	**0.616**	0.466	1.006	0.671
	12	**0.815**	0.575	1.380	0.776
	15	**1.041**	0.918	1.483	0.817
WS [m/s]	3	**0.503**	0.333	0.551	0.350
	6	**0.605**	0.318	0.671	0.313
	12	**0.697**	0.333	0.893	0.429
	15	**0.787**	0.447	0.992	0.496
PA [hPa]	3	**0.042**	0.090	0.070	0.052
	6	**0.045**	0.026	0.102	0.067
	12	**0.057**	0.037	0.140	0.079
	15	**0.056**	0.027	0.156	0.117
P [mm]	3	0.136	0.489	**0.121**	0.432
	6	0.115	0.256	**0.097**	0.185
	12	**0.121**	0.173	0.149	0.254
	15	**0.124**	0.164	0.134	0.195
SWC [%]	3	**0.386**	0.243	1.039	0.926
	6	**0.493**	0.331	1.830	1.340
	12	**0.578**	0.377	3.600	2.192
	15	**0.607**	0.415	4.192	2.400
TS [C]	3	**0.307**	0.267	0.677	0.477
	6	**0.411**	0.285	0.834	0.596
	12	**0.707**	0.693	1.261	0.783
	15	**0.738**	0.755	1.571	1.033

A.2 Additional Timeseries

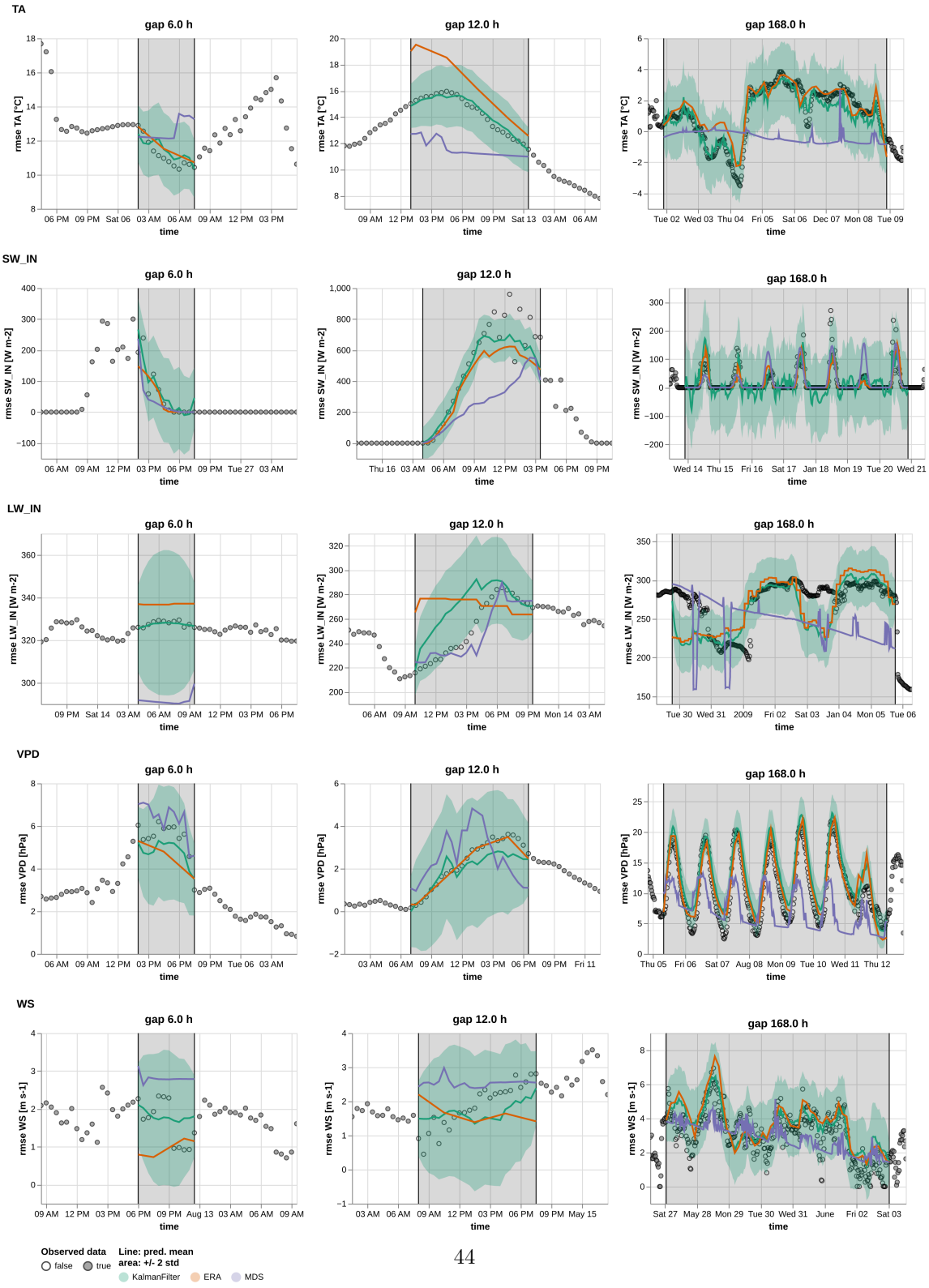

Figure 13: Timeseries 1

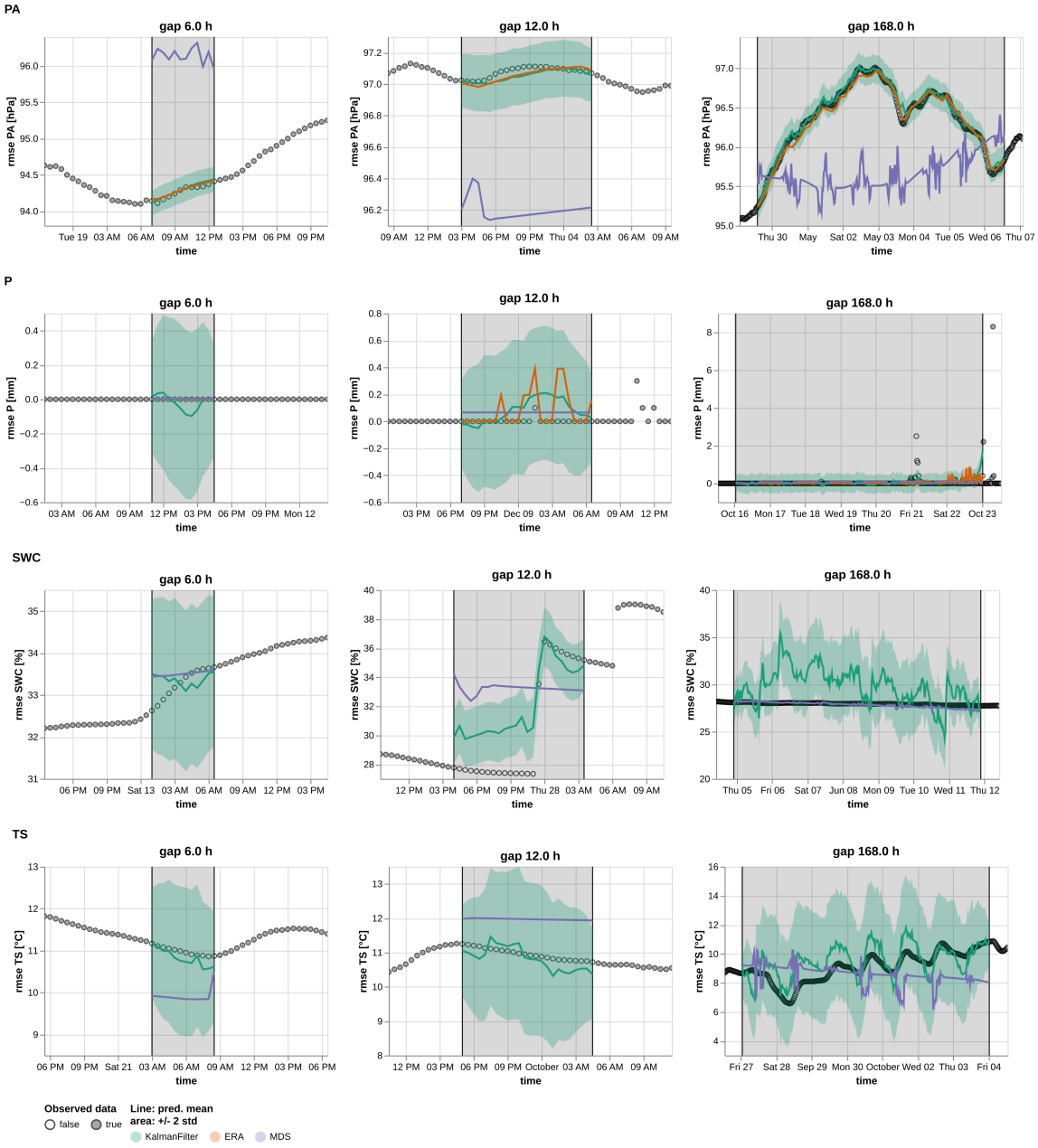

Figure 14: Timeseries 1

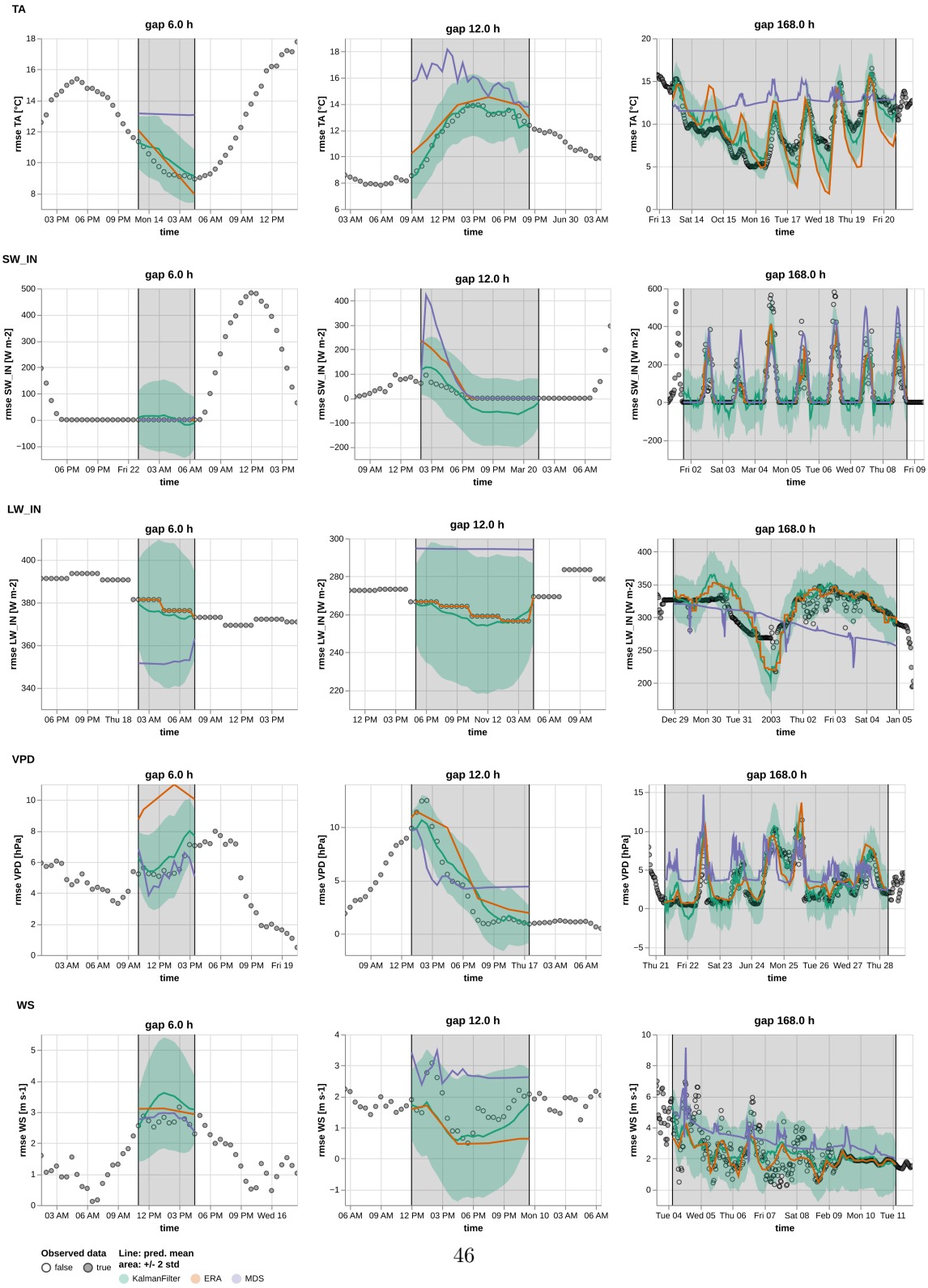

Figure 15: Timeseries 1

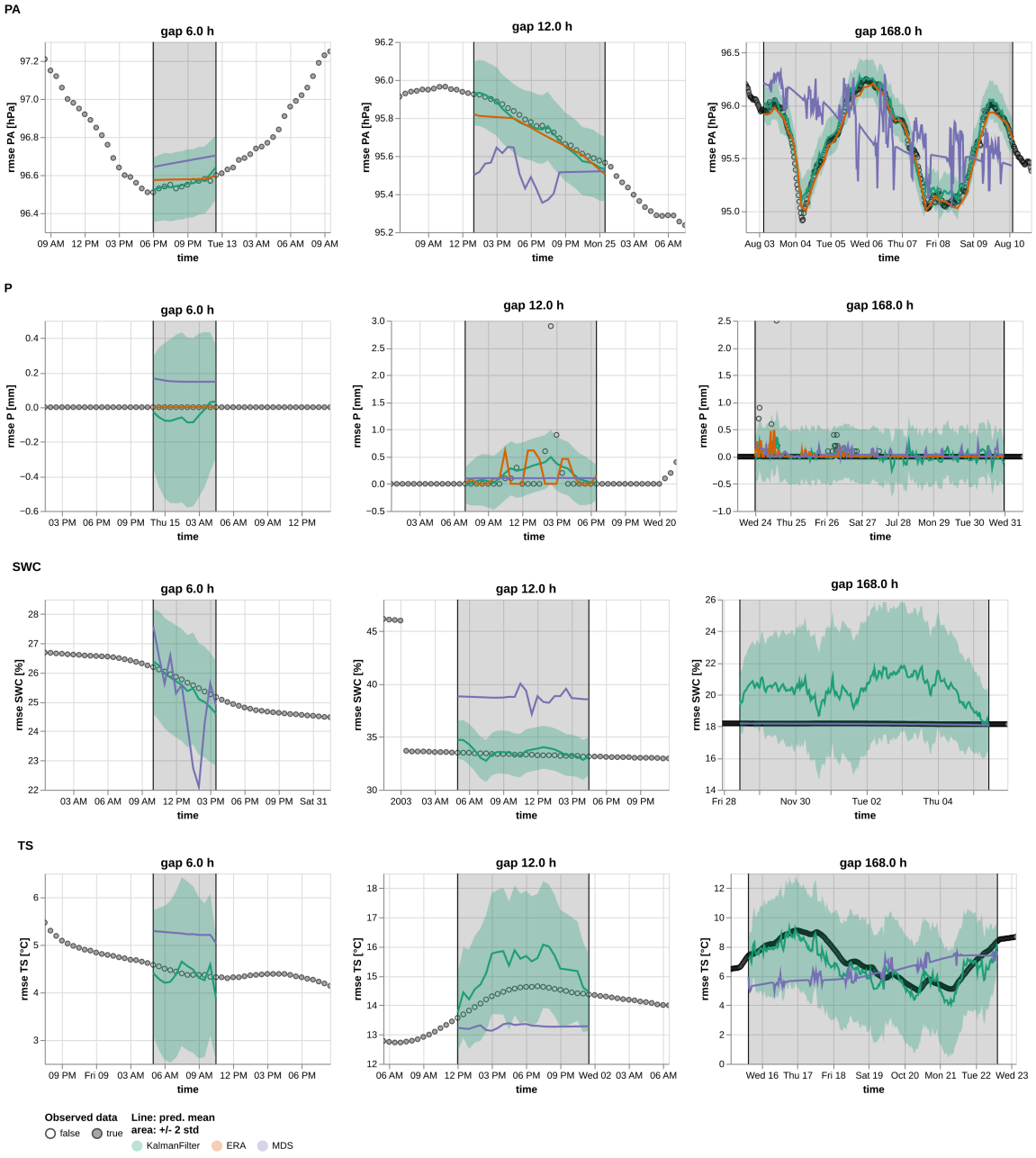

Figure 16: Timeseries 1

B Comparison Standard Square Root Kalman Filter

B.1 Numerical Stability

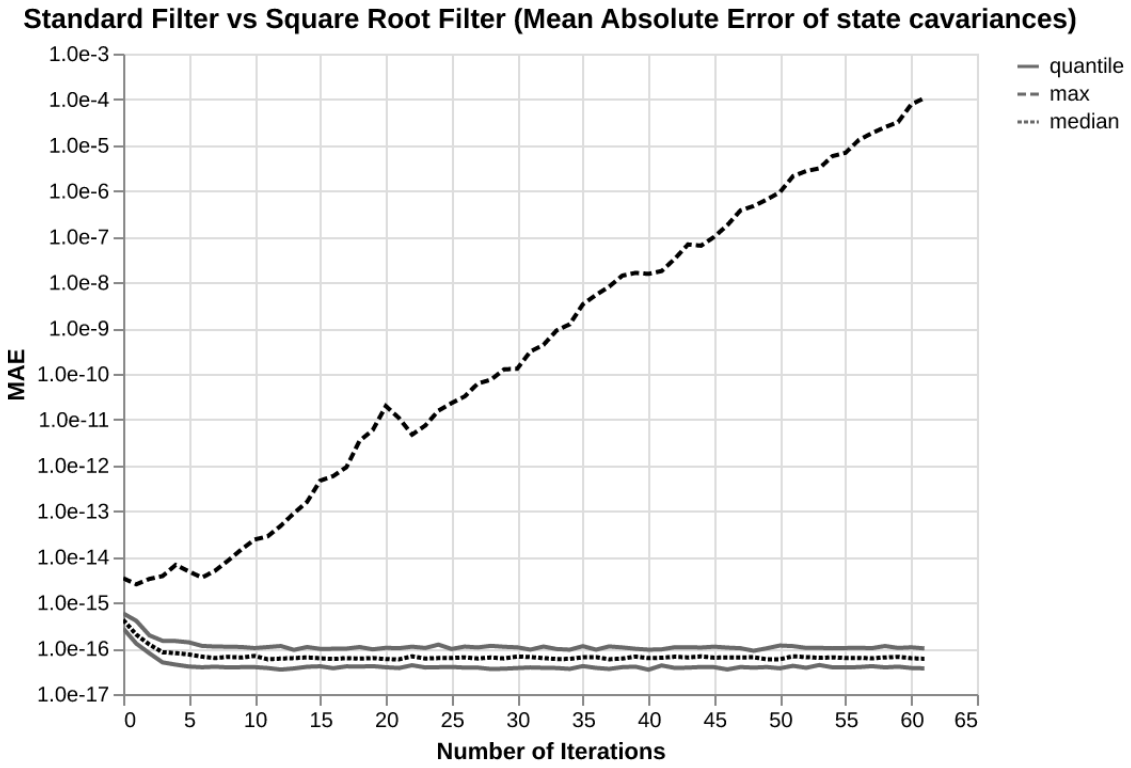

Figure 17: Comparison of standard Kalman Filter implementation and Square Root Filter. For 100 times the filter has been initialized with random parameters (drawn from a uniform distribution range 0-1) and then filtered 62 observations. At each filter iteration calculated the Mean Absolute Error (MAE) between the state covariance from the standard filter and the square root filter. The plot shows the median, 1 and 3 quartile and the maximum of the MAE across the 100 samples. You can see that usually the two filter implementations agree, but for a few parameters combinations the error is growing with the number of iterations of the filter suggesting a numerical stability issue. After 62 iterations the standard filter crashes because the covariance is not positive definite anymore. The initial bigger error can be explained by the slightly different initial state covariance.