university of groningen

faculty of science and engineering

Learning the Kalman Filter Parameters: an Expectation-Maximization Approach

Master Research Project Applied Mathematics

Student: L.H. Kriouar

First supervisor: Prof. Dr. M.A. Grzegorczyk

Second assessor: Dr. W.P. Krijnen

Applied Mathematics

2023, July 14

Abstract

The Kalman filter has been proven to be an optimal hidden-state estimator of a (linear) state-space model with Gaussian noise terms. In this thesis, we consider the Kalman filter in mathematical depth and derive the forming equations. We derive backward corrections on the produced estimates, also known as Kalman smoothing. Furthermore, we review estimations of the parameters of the underlying state-space model through the expectation-maximization algorithm and consider asymptotic theory of the resulting maximum likelihood estimates. We apply the methodology to simulations and an example from literature, before moving on to forecasting of stock prices, which we model in a state-space sense. The main result of this thesis is the implementation of a more flexible EM algorithm for the Kalman filter, compared to existing libraries, in Python. We also observed that although the chaotic nature of stock prices, the Kalman filter provides accurate estimates of the underlying hidden state.

Contents

Preface

Working on this Master research project throughout the whole academic year has been a wonderful experience. A priori, I did not imagine how interested and passionate I would have been about Kalman filtering as I am while writing this preface. Together with applying the EM algorithm to the Kalman filter setting, I am glad to present to you what has become my Master thesis in Applied Mathematics.

The upcoming work is of course not solely accomplished by myself. First of all, I would like to humbly thank my supervisor, Prof. Dr. M.A. Grzegorczyk for not only the excellent guidance and support during the project, but also for introducing the topic of Kalman filtering. Secondly, as a second assessor, Dr. W.P. Krijnen has also been involved above average and I would also like to thank him for his care and guidance throughout the year.

I also want to thank my friends and family for all the support and interest in this project. As a mathematician, you do not expect your friends nor your family to have a gasp of what you are exactly doing, but the people around me have always been interested in what I was doing, which kept me motivated throughout the academic year. A special thanks to my peer and best friend, J.H.C. van Ekelenburg, who was always available for some critical discussions about the subject and process. Especially from a systems and control point of view, he was really helpful in gaining insights in the mathematics behind e.g. the Kalman filter. Additionally, I want to thank my friend, J.H.G. Jeung, for taking some time to verify the linguistics of the thesis.

As the author of this thesis, I hope you, the reader will enjoy reading this thesis as much as I have enjoyed writing it. The Kalman filter's background is a very interesting field to explore and I hope this work will serve as confirmation.

In the memory of my late grandfather, who passed away in 2019. I would have loved for you to see me graduate.

My regards,

Lotvi Kriouar

1 Introduction

Time series analysis has been a part of statistics for decades. Methods for statistical inference as well as forecasting in a machine learning framework are rewarded for their performance. From simple examples as autoregressive methods (e.g. AR, ARIMA or ARMAX models) to advanced methods as the field of neural networks (e.g. long short-term memory), there are lots of modelling techniques which can be applied to time series data today.

One of these methods is the notion of *state-space models* (or *systems*). By denoting the observed time series by z_t, where t is the time index, we can introduce the following formulation of a state-space model:

$$\begin{aligned} X_t &= FX_{t-1} + w_t, \\ Z_t &= HX_t + v_t. \end{aligned} \tag{1}$$

Here X_t is called the *state* of the system. It is assumed that the observed time series z_t is influenced by the so-called *hidden state* X_t. By 'hidden', we mean that x_t cannot be observed and is regarded as an internal process. The system (1) becomes stochastic by incorporating the noise terms w_t and v_t. By the structure of (1), one could think of this model as a continuous counterpart of a hidden Markov model, where the dependency relations are described analogously (Figure 1).

The *Kalman filter* has been proven to serve as an optimal estimator of the hidden state x_t [9]. Starting from an initial condition $\hat{x}_0$, the Kalman filter propagates state estimations through time with minimal variance. Optimality of the Kalman filter requires Gaussian white noise terms and several independence assumptions. The filter has a predict-update nature. That is, after predicting the next hidden state x_{t+1}, it incorporates the current measurement z_{t+1} to update this prediction. Furthermore, the *Kalman smoother* provides backward recursions to correct the hidden state estimates produced by the Kalman filter, which enables us to reduce the filter's variance even more. After its introduction, the Kalman filter has been covered in great detail in [1, 3, 5, 6, 11, 16].

However, one downfall of the Kalman- filter and smoother is that they both require knowledge of the system's parameters (e.g. F and H in (1)), while there are occasions where these parameters are, or at least partially, unknown. Fortunately, parameter estimation methods for state-space models have been introduced in e.g. [5, 16]. In this thesis, we explore and apply the *expectation-maximization (EM) algorithm* to the Kalman filter setting. That is, we propose an EM based estimation procedure for (a subset of) the parameters in (1), which results in the so-called *EMKF algorithm*. Furthermore, by studying the asymptotic theory of the resulting maximum likelihood estimators, we are able to derive their confidence intervals.

In this thesis, we are interested in finding out about the performance a learned Kalman filter through the EMKF algorithm on real-world data. Therefore, we apply the Kalman filter to stock price data as a forecasting method. Before we do so, we perform simulations on the implentend EMKF algorithm as a proof of concept.

The main result of this thesis is a rather flexible EMKF algorithm implementation in Python. Compared to the existing `em` method from the `pykalman` library, the EMKF algorithm implemented by the author has the following additional options:

- While the `em` method only performs a predefined number of iterations, the EMKF algorithm written by the author actually optimizes the parameter estimates in terms of convergence to stationary point of the log-likelihood of the measurements (the observed data in the Kalman filter setting). A priori, one does generally not know how many iterations it would take to achieve maximum likelihood estimates. Therefore, it would be beneficial to keep track of the likelihood function and terminate the algorithm whenever this function has converged to a stationary point. To avoid potential overfitting and to limit computation times, the user of the function sets a maximum number of iterations to perform.
- Additional to the log-likelihood increase, the user can choose to define convergence of the EMKF algorithm in terms of the difference in parameter estimates as well. That is, if the parameter estimates deviate too low from the previous estimates, then convergence is achieved. In this way, one does not only have convergence of the log-likelihood of the measurements, but stable parameter estimates are guaranteed as well.

- The user can manually set the thresholds regarding the log-likelihood increase and the difference in parameter estimates. In this way, the user of can define their own degree of convergence of the EMKF algorithm.

We also provide a framework on how to numerically approximate the Hessian matrix of the log-likelihood of the measurements w.r.t. to the unknown parameters. In this way, one could also find out whether the stationary point of the log-likelihood of the measurements corresponds to a (local) maximum or a saddle point of this likelihood function.

The construction of the EMKF algorithm has already been accomplished in 1982 [15]. However, up to the best knowledge of the author, the EMKF algorithm has not been applied to forecasting stock prices yet. Therefore, the obtained results are hard to put into perspective. Forecasting using the Kalman filter in general is done in e.g. [13].

The remainder of this thesis is structured as follows. In Section 2, we cover the theoretical framework relevant for this thesis. The core subjects in this section are the Kalman filter, the Kalman smoother and the EM algorithm. In Section 3, we explain how the theoretical framework can be translated into a method to approach real-world data. Next, in Section 4, we perform the simulations and stock price forecasting. In Section 5, we conclude our findings and finally, in Section 6, we review the methods and results in this thesis.

2 Theory

The underlying methodology of Kalman filtering relies on fields such as probability theory, linear algebra and (matrix) calculus. In this section, we consider the Kalman filter in mathematical detail. Besides deriving the equations defining the filter, we also verify these equations from a conditional density viewpoint. Next, we consider the Kalman smoother, which serves as a correction of the estimates produced by the Kalman filter, given the entire data. Also, we explain how one could infer about the unknown parameters of the Kalman filter by the expectation-maximization (EM) algorithm. We derive the EM algorithm for the Kalman filter/smoother to introduce the EMKF algorithm and finish this section with considering the asymptotic theory of the resulting maximum likelihood estimates.

The remainder of this section is structured as follows: in Section 2.1, we cover mathematical preliminaries which are useful in understanding what follows. Section 2.2 is devoted to the Kalman filter, while the Kalman smoother is explained in Section 2.3. Next, in Section 2.4, we consider the expectation-maximization algorithm and we combine everything in Section 2.5 to derive an expectation-maximization algorithm for estimation of the Kalman filter parameters. Finally, we review asymptotic theory of these estimators in Section 2.6.

2.1 Mathematical Preliminaries

We initiate this theory section with an overview of relevant mathematical preliminaries. It is assumed that the reader has basic knowledge of linear algebra, probability theory, statistics and calculus. In Appendix A, we extend the basic calculus knowledge by covering several properties from matrix calculus, which are relevant in Section 2.5.

2.1.1 Hidden Markov Models

A *Hidden Markov model (HMM)* is a statistical model that is useful when we model an observable process Z that is believed to be influenced by a hidden process X [18]. We often refer to X as the *hidden state*. The hidden state X is assumed to possess the *Markov property*, which is defined as follows.

Definition 1 (Markov Property)**.** Let $X = \{X_t\}_{t=0}^{T}$ be a stochastic process. X is said to possess the Markov property if the distribution of X_t $(t \in \{0, \ldots, T\})$ only depends on X_{t-1}. That is, conditioned on X_{t-1}, X_t is independent of all previous states. Formally, we have

$$\mathbb{P}(X_t = x_t | X_{t-1} = x_{t-1}, \ldots, X_0 = x_0) = \mathbb{P}(X_t = x_t | X_{t-1} = x_{t-1})$$

The Markov property refers to a memoryless stochastic process. If we assume that the hidden Markov model has an initial hidden state X_0, we start observing data at $t = 1$ and we let $Z = \{Z_t\}_{t=1}^{T}$ and $X = \{X_t\}_{t=0}^{T}$, the model (and (1)) could be visualized as the Bayesian network in Figure 1.

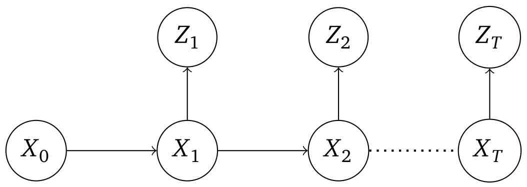


Figure 1: The hidden Markov model considered in this thesis

Figure 1 also shows the conditional independence structures within the model: we assume that the current observation Z_t is independent of all previous observations, conditioned on the current hidden state X_t. Hidden Markov models are particularly a popular choice when both the observations and the hidden states are discrete random variables. However, some methodology exists for the continuous case, where the HMMs are called *continuous hidden Markov models (CHMMs)*. For the continuous case, statistical inference is not as far developed as in the discrete case. However, the Kalman filter is an example of inference for a CHMM. For more text on hidden Markov models and certain applications, the reader is referred to [18].

2.1.2 Statistical Properties

This section is devoted to stating some statistical properties which are used throughout the upcoming derivations. Lemma 1 returns in Section 2.3 while deriving the Kalman smoother equations.

Lemma 1. Let X and Y be n- and m-dimensional random vectors respectively such that X and Y are jointly distributed by

$$\begin{pmatrix} X \\ Y \end{pmatrix} \sim \mathcal{N}_{n+m}\left(\begin{pmatrix} \mu_X \\ \mu_Y \end{pmatrix}, \begin{pmatrix} \Sigma_X & \Sigma_{XY} \\ \Sigma_{XY}^\top & \Sigma_Y \end{pmatrix}\right).$$

Then, for the marginal distribution of X and the conditional distribution of $X|(Y=y)$, we have

$$\begin{aligned} X &\sim \mathcal{N}_n(\mu_X, \Sigma_X), \\ X|(Y=y) &\sim \mathcal{N}_n\left(\mu_X + \Sigma_{XY}\Sigma_Y^{-1}(y-\mu_Y), \Sigma_X - \Sigma_{XY}\Sigma_Y^{-1}\Sigma_{XY}^\top\right). \end{aligned}$$

Proof. Let $\boldsymbol{\mu} = \begin{pmatrix} \mu_X \\ \mu_Y \end{pmatrix}$ and $\boldsymbol{\Sigma} = \begin{pmatrix} \Sigma_X & \Sigma_{XY} \\ \Sigma_{XY}^\top & \Sigma_Y \end{pmatrix}$. We first derive the marginal distribution of X, before deriving the conditional distribution $X|(Y=y)$.

To obtain the marginal distribution of X, consider the linear transformation

$$X = A\begin{pmatrix} X \\ Y \end{pmatrix},$$

where $A = \begin{pmatrix} I_n & 0_{m\times m} \end{pmatrix}$. Here, $0_{m\times m}$ is defined in Table 1. Since the Gaussian distribution is invariant under linear transformations, X is Gaussian distributed as well. Its mean and variance are computed by

$$\begin{aligned} \mathbb{E}(X) &= \mathbb{E}\left(A\begin{pmatrix} X \\ Y \end{pmatrix}\right) = A\boldsymbol{\mu} = \mu_x, \\ \mathrm{Var}(X) &= \mathrm{Var}\left(A\begin{pmatrix} X \\ Y \end{pmatrix}\right) = A\boldsymbol{\Sigma}A^\top = \begin{pmatrix} \Sigma_x & \Sigma_{XY} \end{pmatrix}A^\top = \Sigma_x. \end{aligned}$$

Hence, we have $X \sim \mathcal{N}_n(\mu_X, \Sigma_X)$. By taking $A = \begin{pmatrix} 0_{n\times n} & I_m \end{pmatrix}$, one could prove $Y \sim \mathcal{N}_m(\mu_Y, \Sigma_Y)$.

For the conditional distribution of $X|(Y=y)$, consider the following LDU decomposition of Σ, obtained by performing block Gauss-Jordan elimination.

$$\boldsymbol{\Sigma} = \begin{pmatrix} I_n & \Sigma_{XY}\Sigma_Y^{-1} \\ 0_{m\times n} & I_m \end{pmatrix}\begin{pmatrix} \Sigma_x - \Sigma_{XY}\Sigma_Y^{-1}\Sigma_{XY}^\top & 0_{n\times m} \\ 0_{m\times n} & \Sigma_y \end{pmatrix}\begin{pmatrix} I_n & 0_{n\times m} \\ \Sigma_y^{-1}\Sigma_{XY}^\top & I_m \end{pmatrix}.$$

Note the appearance of the Schur complement of Σ_y in the second (block) matrix. Note that the inverse of $\boldsymbol{\Sigma}$ is given by

$$\boldsymbol{\Sigma}^{-1} = \begin{pmatrix} I_n & 0_{n\times m} \\ -\Sigma_y^{-1}\Sigma_{XY}^\top & I_m \end{pmatrix}\begin{pmatrix} \Sigma_x - \Sigma_{XY}\Sigma_Y^{-1}\Sigma_{XY}^\top & 0_{n\times m} \\ 0_{m\times n} & \Sigma_y \end{pmatrix}\begin{pmatrix} I_n & -\Sigma_{XY}\Sigma_Y^{-1} \\ 0_{m\times n} & I_m \end{pmatrix}. \tag{2}$$

Denote the densities of $(X,Y)^\top$, X and $X|(Y=y)$ by $f_{X,Y}(x,y)$, $f_X(x)$ and $f_{X|Y}(x|y)$ respectively. Then, the definition of conditional probability reads $f_{X|Y}(x|y) = \dfrac{f_{X,Y}(x,y)}{f_y(y)}$. We compute

$$\begin{aligned} f_{X|Y}(x|y) &= \frac{f_{X,Y}(x,y)}{f_y(y)} \\ &= \frac{(2\pi)^{-1/2(n+m)}|\boldsymbol{\Sigma}|^{-1/2}\exp\left(-\frac{1}{2}\left(\begin{pmatrix} x \\ y \end{pmatrix} - \boldsymbol{\mu}\right)^\top \boldsymbol{\Sigma}^{-1}\left(\begin{pmatrix} x \\ y \end{pmatrix} - \boldsymbol{\mu}\right)\right)}{(2\pi)^{-1/2m}|\Sigma_Y|^{-1/2}\exp\left(-\frac{1}{2}(y-\mu_Y)^\top\Sigma_y^{-1}(y-\mu_Y)\right)} \\ &= (2\pi)^{-1/2n}\left(|\boldsymbol{\Sigma}|/|\Sigma_Y|\right)^{-1/2}\exp\left(-\frac{1}{2}\left(\left(\begin{pmatrix} x \\ y \end{pmatrix} - \boldsymbol{\mu}\right)^\top \boldsymbol{\Sigma}^{-1}\left(\begin{pmatrix} x \\ y \end{pmatrix} - \boldsymbol{\mu}\right) - (y-\mu_Y)^\top\Sigma_Y^{-1}(y-\mu_Y)\right)\right). \end{aligned}$$

For the product of the determinants, we have

$$\begin{aligned} |\boldsymbol{\Sigma}|/|\Sigma_Y| &= \frac{|\boldsymbol{\Sigma}|}{\begin{vmatrix} I_n & 0_{n\times m} \\ 0_{m\times n} & \Sigma_Y \end{vmatrix}} \qquad\qquad |\Sigma_Y| = \begin{vmatrix} I_n & 0_{n\times m} \\ 0_{m\times n} & \Sigma_Y \end{vmatrix} \\ &= \left|\boldsymbol{\Sigma}\begin{pmatrix} I_n & 0_{n\times m} \\ 0_{m\times n} & \Sigma_Y^{-1} \end{pmatrix}\right| \\ &= \begin{vmatrix} \Sigma_X & \Sigma_{XY}\Sigma_Y^{-1} \\ \Sigma_{XY}^\top & I_m \end{vmatrix} \\ &= \left|\Sigma_X - \Sigma_{XY}\Sigma_Y^{-1}\Sigma_{XY}^\top\right|. \end{aligned}$$

Using (2), one can show that

$$\left(\begin{pmatrix} x \\ y \end{pmatrix} - \boldsymbol{\mu}\right)^\top \boldsymbol{\Sigma}^{-1}\left(\begin{pmatrix} x \\ y \end{pmatrix} - \boldsymbol{\mu}\right) - (y-\mu_Y)^\top \Sigma_Y^{-1}(y-\mu_Y) =$$
$$\left(x - \left(\mu_x + \Sigma_{XY}\Sigma_Y^{-1}(y-\mu_Y)\right)\right)^\top \left(\Sigma_X - \Sigma_{XY}\Sigma_Y^{-1}\Sigma_{XY}^\top\right)^{-1}\left(x - \left(\mu_x + \Sigma_{XY}\Sigma_Y^{-1}(y-\mu_Y)\right)\right).$$

Hence, we have

$$\begin{aligned} f_{X|Y}(x|y) = (2\pi)^{-1/2n}&\left|\Sigma_X - \Sigma_{XY}\Sigma_Y^{-1}\Sigma_{XY}^\top\right|^{-1/2}\cdot \\ &\exp\left(-\frac{1}{2}\left(x - \left(\mu_x + \Sigma_{XY}\Sigma_Y^{-1}(y-\mu_Y)\right)\right)^\top \left(\Sigma_X - \Sigma_{XY}\Sigma_Y^{-1}\Sigma_{XY}^\top\right)^{-1}\left(x - \left(\mu_x + \Sigma_{XY}\Sigma_Y^{-1}(y-\mu_Y)\right)\right)\right), \end{aligned}$$

which characterizes the $\mathcal{N}_n\left(\mu_X + \Sigma_{XY}\Sigma_Y^{-1}(y-\mu_Y), \Sigma_X - \Sigma_{XY}\Sigma_Y^{-1}\Sigma_{XY}^\top\right)$ distribution. Hence, we conclude that

$$X|(Y=y) \sim \mathcal{N}_n\left(\mu_X + \Sigma_{XY}\Sigma_Y^{-1}(y-\mu_Y), \Sigma_X - \Sigma_{XY}\Sigma_Y^{-1}\Sigma_{XY}^\top\right).$$

□

Lemma 2 is used to compute the expected log-likelihood of the complete data for the expectation-maximization algorithm in Section 2.5.

Lemma 2. Let X and Y be n-dimensional random vectors and let $A \in \mathbb{R}^{n\times n}$. Then,

$$\mathbb{E}\left(X^\top AY\right) = \mathbb{E}(X)^\top A\mathbb{E}(Y)) + \operatorname{Tr}(A\operatorname{Cov}(Y,X)).$$

Proof. We compute

$$\begin{aligned} \mathbb{E}\left(X^\top AY\right) &= \mathbb{E}\left(\operatorname{Tr}\left(X^\top AY\right)\right) && X^\top AY \text{ is a scalar-valued random variable} \\ &= \mathbb{E}\left(\operatorname{Tr}\left(AYX^\top\right)\right) && \text{Cyclic property of the trace operator} \\ &= \operatorname{Tr}\left(A\mathbb{E}\left(YX^\top\right)\right) && \text{Trace- and expectation operator are linear} \\ &= \operatorname{Tr}\left(A\left(\operatorname{Cov}(Y,X) + \mathbb{E}(Y)\mathbb{E}(X)^\top\right)\right) \\ &= \operatorname{Tr}(A\operatorname{Cov}(Y,X)) + \operatorname{Tr}\left(A\mathbb{E}(Y)\mathbb{E}(X)^\top\right) \\ &= \operatorname{Tr}(A\operatorname{Cov}(Y,X)) + \operatorname{Tr}\left(\mathbb{E}(X)^\top A\mathbb{E}(Y)\right) \\ &= \mathbb{E}(X)^\top A\mathbb{E}(Y) + \operatorname{Tr}(A\operatorname{Cov}(Y,X)). \end{aligned}$$

□

2.1.3 Linear Time-Invariant Systems

State-space models are in fact a wide class of model formulations. In this thesis, we focus a particular class of state-space models: *(discrete) linear time-invariant (LTI) systems*. We provide a short introduction to the reader. A LTI system is formulated as follows.

$$\Sigma : \begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{3}$$

Here, $x : \mathcal{T} \to \mathcal{X}$ is called the **state** of the system, $u : \mathcal{T} \to \mathcal{U}$ is called the **input**, while $y : \mathcal{T} \to \mathcal{Y}$ is called the **output**. For convenience, we rather describe u, x and y as vectors rather than functions of t (i.e. $x(t) \in \mathcal{X}$). Usually, and in this thesis, $\mathcal{T} = \mathbb{R}_+$, $\mathcal{X} \subseteq \mathbb{R}^n$, $\mathcal{U} \subseteq \mathbb{R}^m$ and $\mathcal{Y} \subseteq \mathbb{R}^p$ where $n, m, p \in \mathbb{N}$.

If we write $x(t_0) = x_0$ and assume that the initial condition x_0 is given, we can solve (3) for the state x using the variation of constants formula [17]. The solution is given by

$$x(t) = e^{At}x_0 + \int_0^t e^{A(t-\tau)}Bu(\tau)\mathrm{d}\tau, \tag{4}$$

where the integral of the matrix is taken entry-wise. To evaluate this integral, recall the definition of the matrix exponential [17], which reads

$$e^{At} = \sum_{k=0}^{\infty} \frac{t^k A^k}{k!},$$

which arises from the multivariate variant of the power series definition of e^t.

An important property of a system as Σ, which comes back in Section 2.6, is the notions of *observability*. Observability is related to being able to reconstructing the state of a LTI system, by observing the output. Loosely speaking, a LTI system is called observable if there is no possible combination of two realizations of the state with a common input value u which produce the same output value y. The *Kalman rank test for observability* can be employed to verify observability of a system.

Lemma 3 (Kalman rank test for observability)**.** Consider the LTI system given by Σ. The system Σ is observable if and only if we have

$$\operatorname{rank}\begin{pmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{pmatrix} = n.$$

Hence, observability is satisfied whenever the so-called observability matrix of Σ has full column rank (and vice versa). A similar result is available whenever the system is discrete as in (1) [17]. Here, we replace C by H and A by F.

2.1.4 Notation

Besides the notation conventions that are used within this thesis, the upcoming sections also introduce lots of variables which might cause some confusion. Hence, we finish this section on preliminaries with two tables, which should provide a clear overview of the notation in this thesis.

0_n	n-dimensional vector of zeros
$0_{n\times m}$	n-by-m matrix of zeros

Table 1: Notation conventions within this thesis

x_t/X_t	The hidden state
$\hat{x}_t^-$	A priori hidden state estimate (Kalman filter)
$\hat{x}_t$	A posteriori hidden state estimate (Kalman filter)
$\tilde{x}_t$	Correction of hidden state estimate (Kalman smoother)
e_t^-	Error of $\hat{x}_t^-$
e_t	Error of $\hat{x}_t$
P_t^-	Covariance matrix of e_t^- (Kalman filter)
P_t	Covariance matrix of e_t (Kalman filter)
$\tilde{P}_t$	Correction of covariance matrix of e_t (Kalman smoother)
z_t/Z_t	The measurements
$\tilde{V}_{t,t-1}$	Lag-1 autocovariance of X_t (Kalman smoother)

Table 2: Most important variables within this thesis

Also, for readability of the derivations, conditional probabilities are abbreviated throughout this thesis. This means that the random variable $X|(Y = y)$ is written as $X|Y$, unless indicated otherwise.

Furthermore, from Section 2.2.3 onwards, the hidden states and measurements are written as either random variables (X_t and Z_t), while their realizations are written as x_t and z_t respectively. For simplicity, we write x_t and z_t in the first two sections of Section 2.2. This is done because we aim to provide a rather intuitive introduction to the Kalman filter, which is more difficult when formal notation is used.

2.2 The Kalman Filter

The Kalman filter serves as one of the core methods in this thesis. Also known as *linear quadratic estimation*), the predict-update algorithm predicts the next hidden state and uses the current measurement (history) to update this prediction. Under several assumptions, the Kalman filter is an optimal filter in the sense of being a *minimal variance unbiased estimator* (MVUE). We gently introduce the topic first (Section 2.2.1), before moving on to the actual mathematical derivations in Section 2.2.2. Next, we look at the Kalman filter from a more probabilistic view in Section 2.2.3.

2.2.1 Introduction

Consider the LTI system Σ defined by (3). Note that the functions x, u and y are continuous functions of time t, which makes the whole system Σ continuous. However, because of the shape of the Kalman filter, we aim to work with state-space models which are a discretized version of Σ. To this end, consider an equidistant grid of time points $\{t\}_{t=0}^{T}$ and let Δt be the distance between these time points. If we assume that A, B, C and D in Σ (c.f. (3)) are constant matrices and that u is constant during each time step (i.e. $u(t+\tau) = u(t)\ \forall \tau \in (0, \Delta t)$), we are able to compute

$$\begin{aligned}
x(t+1) &= x(t+\Delta t) \\
&= e^{A(t+\Delta t)}x_0 + \int_0^{t+\Delta t} e^{A(t+\Delta t-\tau)}Bu(\tau)\mathrm{d}\tau && \text{Plugging in (4)} \\
&= e^{A\Delta t}e^{At}x_0 + \int_0^{t} e^{A(t+\Delta t-\tau)}Bu(\tau)\mathrm{d}\tau + \int_t^{t+\Delta t} e^{A(t+\Delta t-\tau)}Bu(\tau)\mathrm{d}\tau \\
&= e^{A\Delta t}\left(e^{At}x_0 + \int_0^{t} e^{A(t-\tau)}Bu(\tau)\mathrm{d}\tau\right) + \int_t^{t+\Delta t} e^{A(t+\Delta t-\tau)}\mathrm{d}\tau Bu(t) && u \text{ constant along time steps} \\
&= e^{A\Delta t}x(t) - \int_{\Delta t}^{0} e^{As}\mathrm{d}sBu(t) && s = t+\Delta t-\tau \\
&= e^{A\Delta t}x(t) + \int_0^{\Delta t} e^{As}\mathrm{d}sBu(t).
\end{aligned}$$

Let $F = e^{A\Delta t}$ and $G = \int_0^{\Delta t} e^{As}\mathrm{d}s$. Then, the discretization of (3) is given by

$$x_{t+1} = Fx_t + Gu_t,\ x_t = x(t), \tag{5}$$

which is the state-space model formulation which we work with throughout this section. Here, F is called the *state transition matrix*. This formulation has several practical applications in fields as autonomous vehicles, airplane tracking as well as the financial world. However, in many cases, a deterministic equation such as (5) fails to completely describe the behaviour of the state x_t. This follows from the fact that it is likely to encounter imperfections during the process which (5) fails to incorporate. Think of bumps in the road a autonomous vehicles drives on, an airplane going through a gust or a stock price crash due to events happening in the world. Assuming that these things happen randomly, we could capture these imperfections in terms of adding a component of Gaussian white noise to (5). Recall that white noise is a sequence of independent random variables with zero mean and finite variance. Usually, we obtain the system given by

$$x_t = Fx_{t-1} + Gu_{t-1} + w_{t-1},\ w_{t-1} \sim \mathcal{N}_n(0_n, Q), \tag{6}$$

where $Q \in \mathbb{R}^{n\times n}$[1]. The term w_t is called the *process noise*. The system propagates the state x_t through time with initial condition x_0. The initial state x_0 could either be fixed or Gaussian distributed i.e. $x_0 \sim \mathcal{N}_n(\xi, \Lambda)$.

Since (6) represents x_1 as a linear combination of x_0 (a variable considered to be either fixed or a Gaussian random variable) and the Gaussian noise variable w_0, x_1 is Gaussian distributed. Hence, for $t \in \{2, \dots, T\}$, x_t is a linear combination of two Gaussian distributed random variables. This implies that under the presence of Gaussian noise in (6), the state is Gaussian distributed. This is actually one of the assumptions for optimality of the Kalman filter [9].

In certain applications, the state of (6) is not (fully) observable i.e. we cannot measure (some of) the components of x_t (this refers to the notion of (continuous) hidden Markov models in Section 2.1). We refer to these components (or the entire state vector x_t) as *hidden states*. For the sake of simplicity, from now on, x_t is referred to as the hidden state. Assume that at each time step, we obtain measurements $z_t \in \mathbb{R}^p$ ($p \in \mathbb{N}$) which gives us information about x_t. They are related by the following equation, which we refer to as the *measurement equation*:

$$z_t = Hx_t + v_t,\ v_t \sim \mathcal{N}_p(0_p, R), \tag{7}$$

where $H \in \mathbb{R}^{p\times n}$ and $R \in \mathbb{R}^{p\times p}$[2]. H is called the *measurement matrix*. The term v_t is called the *measurement noise*, which is included in (7) because we assume measurements to be noisy. Also, note that, as similar to the hidden state, the measurements follow a Gaussian distribution.

[1]There are occasions where Q is allowed to be time-dependent i.e. $w_t \sim \mathcal{N}_n(0, Q_t)$.
[2]Just as for the process noise covariance matrix Q, R is allowed to be time-dependent.

As an example for the relation between measurements and the hidden state, consider a GPS tracker of an airplane which is not necessarily fully accurate (e.g. due to the large velocity of the airplane).

Example. Suppose that we are modelling the trajectory and the velocity of an airplane by (6), which are the components of our hidden state x_t. Assume that F and G are known. u_t could e.g. denote the torque. If we obtain a measurement of the location of the airplane at any time step, we have that z_t is a measurement of the first component of our state. Hence, we have $H = \begin{pmatrix} 1 & 0 \end{pmatrix}$.

This example states that there are occasions where we measure components of the hidden state directly. For other cases, measurements could be linear combinations of the hidden state. These relations are defined by the shape of H.

The goal of Kalman filtering is to use the measurements z_t $(t \in \{1, \dots, T\})$[3] to estimate the hidden state x_t. Typically in a statistical/machine learning framework, estimation is subject to a certain loss function. This loss function quantifies the 'goodness' of a given estimate: a smaller loss indicates a better estimate. Consider the *squared error* as the loss function i.e. $L : \mathbb{R}^n \to \mathbb{R}$, which is defined by

$$L(y) = (x - y)^\top (x - y) = ||x - y||^2,$$

for any estimate of x, denoted by $y \in \mathbb{R}^n$. The usual approach in statistics/machine learning is to minimize the expected loss, or *risk*, which is given by

$$R(y) = \mathbb{E}(L(y)) = \mathbb{E}\left((x - y)^\top (x - y)\right). \tag{8}$$

If we assume that x follows a Gaussian distribution with mean y, then its covariance matrix is given by:

$$\Sigma_x = \mathbb{E}\left((x - y)(x - y)^\top\right).$$

Note that if we take the trace of Σ_x, we obtain

$$\operatorname{Tr} \Sigma_x = \mathbb{E}(L(y)) = R(y),$$

since the trace of an outer product of two vectors equals their inner product. Note that this equality only holds if y is the mean of the Gaussian distributed random variable x. In Section 2.2.3, we show that the a posteriori estimates produced by the Kalman filter are the means of the distribution of the hidden state[4].

Now, we fix t and consider x_t. Denoting our estimate of x_t by $\hat{x}_t$, (8) becomes

$$R(\hat{x}_t) = \mathbb{E}\left((x_t - \hat{x}_t)^\top (x_t - \hat{x}_t)\right),$$

which is exactly the (total[5]) variance of our estimate. Hence, this means that the aim of Kalman filtering is to produce an estimate of the hidden state x_t with minimal variance (if we consider the squared error loss function).

Everything that we have discussed so far is actually described formally by the *Wiener problem* [9], which is stated as follows.

Problem 1 (Wiener problem)**.** Consider the dynamic model

$$\begin{aligned} x_t &= F x_{t-1} + G u_{t-1} + w_{t-1}, \\ z_t &= H x_t + v_t \end{aligned} , t \in \{1, \dots, T\} \tag{9}$$

where $F \in \mathbb{R}^{n\times n}$, $G \in \mathbb{R}^{n\times m}$, $H \in \mathbb{R}^{p\times n}$, $x_t \in \mathbb{R}^n$, $u_t \in \mathbb{R}^m$, $z_t \in \mathbb{R}^p$, $w_{t-1} \sim \mathcal{N}_n(0_n, Q)$ and $v_t \sim \mathcal{N}_p(0_p, R)$ where $Q \in \mathbb{R}^{n\times n}$ and $R \in \mathbb{R}^{p\times p}$. Given the measurements $\{z_t\}_{t=1}^T$, find an estimate $\hat{x}_t$ of x_t for all t with initial condition $\hat{x}_0$ which minimizes the risk.

[3]Usually, we start obtaining measurements about the hidden state x_t from $t = 1$ onwards with x_0 as an initial condition.
[4]In fact, they are the means of a conditional distribution of the hidden state, but we omit this here for the sake of simplicity.
[5]'total' in the sense that we sum over the variances of individual components of the estimate vector.

Since we are talking about minimizing the risk, we refer to the estimate described above as 'optimal'. The Kalman filter is the optimal solution to the Wiener problem under certain assumptions. Hence, under these assumptions, which we consider below, the Kalman filter does not only produce an estimate of the hidden state, any other estimate of the hidden state has a larger variance.

The first assumption of the Kalman filter is that the hidden state x_t (and the measurements, something we consider in depth in Section 2.2.3) follow a normal (Gaussian) distribution. Although we already verified this by the presence of Gaussian noise terms, it is worth mentioning again. Secondly, x_t is assumed to be independent of the future process noise w_τ and the measurement noise v_κ for any times $t, \tau, \kappa \in \{1, \dots, T\}$ with $\tau > t$. A similar assumption holds for the measurements z_t, but additional with $\kappa \neq t$. Under these assumptions and the dynamics given by (9), the Kalman filter produces estimates of x_t with minimal variance.

As mentioned in the Introduction, the Kalman filter consists of an prediction step and an update step. In the prediction step, we predict the value of the state at the next time step. In the update step, we adjust our prediction obtained in the prediction step. We denote the prediction made in the prediction step by an *a priori estimate* $\hat{x}_t^-$. The adjusted prediction (resulting from the update step) is denoted by the *a posteriori estimate* $\hat{x}_t$. An schematic overview of the procedure of Kalman filtering is shown in Figure 2.

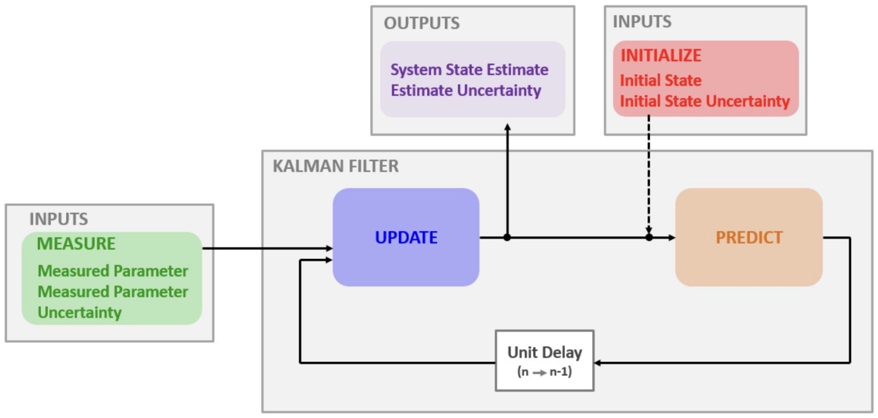


Figure 2: A schematic overview of the Kalman filter. Here, the time t is denoted by n. The figure is retrieved from kalmanfilter.net

If we aim to minimize the (total) variance of our hidden state estimates (read 'risk'), we need to keep track of the error between the prediction of the state and its true value. To this end, we define the *a priori error* $e_t^- = \hat{x}_t^- - x_t$. The *a posteriori error* is given by $e_t = \hat{x}_t - x_t$. Note that here, x_t is not known.

Note that minimizing the variance of our hidden state estimates makes the most sense if we are able to produce unbiased estimates (i.e. $\mathbb{E}(\hat{x}_t) = \mathbb{E}(x_t)$) of these hidden states. If we e.g. would have produced an (a posteriori) estimate with a bias, minimal variance would not yield an estimate in optimal sense (we could apply a translation to our estimate, but let us avoid that). However, unbiasedness of both (a priori and a posteriori) hidden state estimates is ensured by the construction of the Kalman filter if the initial condition $\hat{x}_0$ is unbiased. After defining the Kalman filter explicitly, we state and prove this result in Proposition 1.

For now, we assume that both hidden state estimates are unbiased. From this, it directly follows that $\mathbb{E}(e_t^-) = \mathbb{E}(e_t) = 0$. This implies that the covariance matrices of the error estimates are given by

$$P_t^- = \mathbb{E}\left(e_t^- e_t^{-\top}\right), \quad P_t = \mathbb{E}\left(e_t e_t^\top\right)$$

Another consequence from the unbiasedness of the Kalman filter estimates that arises here is that P_t^- and P_t are not only the covariance matrices of the a priori and a posteriori errors, but also of the hidden states. This is in fact a significant result, which we use throughout the remainder of this section.

Now that we have motivated and partially introduced the Kalman filter, we move on to its construction.

2.2.2 The Filtering Equations

We now derive the five equations which form the Kalman filter. To this end, suppose that we are at time $t \in \{1, \dots, T\}$. This means that we know what our a priori and a posteriori hidden state estimates, $\hat{x}_t^-$ and $\hat{x}_t$ are up to, but excluding, t. Furthermore, we have obtained measurements $z_1, \dots, z_{t-1}$. With this knowledge, the first Kalman filter equation is similar to (5) and is called the *state extrapolation equation*.

Definition 2 (State Extrapolation Equation). Let F, G, x and u be as in Problem 1. Suppose that $\hat{x}_t$ and u_t are known. The **state extrapolation equation** is then given by

$$\hat{x}_{t+1}^- = F\hat{x}_t + Gu_t.$$

The state extrapolation equation is used to predict the state at the next time step, assuming the system is fully deterministic (i.e. the process noise w_t is ignored).

Apart from keeping track of the state extrapolations, we aim to keep track of the uncertainty of these extrapolations as well. To do so, also assume that up to time t, we know what both estimates for the error covariance matrices, P_t^- and P_t are. We compute

$$\begin{aligned}
P_{t+1}^- &= \mathbb{E}\left(e_{t+1}^- e_{t+1}^{-\top}\right) \\
&= \mathbb{E}\left((\hat{x}_{t+1}^- - x_{t+1})(\hat{x}_{t+1}^- - x_{t+1})^\top\right) \\
&= \mathbb{E}\left((F\hat{x}_t + Gu_t - (Fx_t + Gu_t + w_t))(F\hat{x}_t + Gu_t - (Fx_t + Gu_t + w_t))^\top\right) && \text{Plugging in (9) and Definition 2} \\
&= \mathbb{E}\left((F(\hat{x}_t - x_t) - w_t)(F(\hat{x}_t - x_t) - w_t)^\top\right) \\
&= \mathbb{E}\left((F(\hat{x}_t - x_t) - w_t)((\hat{x}_t - x_t)^\top F^\top - w_t^\top)\right) \\
&= \mathbb{E}\left(F(\hat{x}_t - x_t)(\hat{x}_t - x_t)^\top F^\top - F(\hat{x}_t - x_t)w_t^\top - w_t(x_t - \hat{x}_t^-)^\top F^\top + w_t w_t^\top\right) \\
&= F\mathbb{E}((x_t - \hat{x}_t^-)(x_t - \hat{x}_t^-)^\top)F^\top - F\mathbb{E}((x_t - \hat{x}_t^-)w_t^\top) - \mathbb{E}(w_t(x_t - \hat{x}_t^-)^\top)F^\top + \mathbb{E}(w_t w_t^\top) \\
&= FP_tF^\top - F\mathbb{E}(x_t - \hat{x}_t^-)\mathbb{E}(w_t^\top) - \mathbb{E}(w_t)\mathbb{E}((x_t - \hat{x}_t^-)^\top)F^\top + Q && e_t^- \text{ and } w_t \text{ are independent} \\
&= FP_tF^\top - 0_{n\times n} - 0_{n\times n} + Q && \mathbb{E}(w_t) = 0 \\
&= FP_tF^\top + Q.
\end{aligned}$$

This prediction equation is known as the *covariance extrapolation equation* and is the second equation of the Kalman filter.

Definition 3 (Covariance Extrapolation Equation). Let F and Q be as in Section 2.2.1. Suppose that P_t is known. Then, the **covariance extrapolation equation** is given by

$$P_{t+1}^- = FP_tF^\top + Q.$$

Note that since ABA^T is symmetric if B is symmetric (where A and B are matrices of suitable dimensions) and Q is symmetric, meaning that the covariance extrapolation equation ensures that P_t^- remains symmetric over time. Also, since P_t is implicitly assumed to be positive definite and Q is positive definite, P_t^- remains positive definite throughout the filtering process. Now that we know how to predict the (uncertainty of the) state at time $t+1$, we move on to updating our predictions (recall the predict-update nature of the Kalman filter from Section 2.2.1).

Suppose that we have predicted our next state $\hat{x}_{t+1}^-$, together with its covariance matrix P_{t+1}^-. We move forward in time i.e. $t \to t+1$. Note that this also means $\hat{x}_{t+1}^- \to \hat{x}_t^-$ and $P_{t+1}^- \to P_t^-$. Right after this forward move, we collect our measurement z_t about the true state x_t, which are related by (9).

As discussed in Section 2.2.1, the goal of Kalman filtering is to minimize the state estimator's variance. The total variance of the a priori state estimate x_t^- equals the trace of P_t^- (since the off-diagonal elements denote covariances and are not of interest in the minimization problem). Hence, whenever this trace is small, we aim to put faith in our predictions. However, it could be the case that the total variance of our measurement noise R (c.f. (7)) is smaller than $\operatorname{Tr} P_t^-$. This implies that our measurements tend to be more accurate than our a priori hidden state estimates. Hence, whenever updating our prediction $\hat{x}_t^-$, we tend to move towards our measurement value if $\operatorname{Tr} R$ is smaller than $\operatorname{Tr} P_t^-$ and vice versa. We define the *innovation* by $z_t - H\hat{x}_t^-$, which is the difference between the measurement value and the (deterministic) prediction of our measurement value, based on $\hat{x}_t^-$. If our a priori estimates are precise, the innovation should be small and vice versa. Let the updated estimate of the hidden state be a linear combination of the a priori estimate and the innovation. By assigning a weight to the innovation, we are able to quantify how much we want to incorporate the innovation into the update equation. The weight assigned to the innovation is called the *Kalman gain* and denoted by $K_t \in \mathbb{R}^{n\times p}$[6]. The Kalman gain is actually computed in the third Kalman filter equation, but for now, assume it is known. This enables us to describe the fourth KF equation, the *state update equation*, in the following manner.

Definition 4 (State Update Equation)**.** Suppose that $\hat{x}_t^-$, H and z_t are known. Then, the **state update equation** is given by

$$\hat{x}_t = \hat{x}_t^- + K_t(z_t - H\hat{x}_t^-),$$

where K_t is the Kalman gain.

The state update equation plays an important role in deriving the fifth (and last) KF equation: the *covariance update equation*. We first rewrite the state update equation (using the measurement equation) by

$$\begin{aligned}\hat{x}_t &= \hat{x}_t^- + K_t(z_t - H\hat{x}_t^-)\\ &= \hat{x}_t^- + K_t(Hx_t + v_t - H\hat{x}_t^-),\end{aligned}$$

which we use to rewrite our a posteriori error expression by

$$\begin{aligned}e_t &= \hat{x}_t - x_t\\ &= \hat{x}_t^- + K_t(Hx_t + v_t - H\hat{x}_t^-) - x_t\\ &= (I_n - K_tH)\hat{x}_t^- + (K_tH - I_n)x_t + K_tv_t\\ &= (I_n - K_tH)(\hat{x}_t^- - x_t) + K_tv_t\\ &= (I_n - K_tH)e_t^- + K_tv_t\end{aligned}$$

Using this, we compute

$$\begin{aligned}P_t &= \mathbb{E}(e_te_t^\top)\\ &= \mathbb{E}\left(\left((I_n - K_tH)e_t^- + K_tv_t\right)\left(I_n - K_tH)e_t^- + K_tv_t\right)^\top\right)\\ &= \mathbb{E}\left(\left((I_n - K_tH)e_t^- + K_tv_t\right)\left(e_t^{-\top}(I_n - K_tH)^\top + v_t^TK_t^T\right)\right)\\ &= \mathbb{E}\left((I_n - K_tH)e_t^-e_t^{-\top}(I_n - K_tH)^\top + (I_n - K_tH)e_t^-v_t^\top K_t^\top + K_tv_te_t^{-\top}(I_n - K_tH)^\top + K_tv_tv_t^\top K_t^\top\right)\\ &= (I_n - K_tH)\mathbb{E}\left(e_t^-e_t^{-\top}\right)(I_n - K_tH)^\top + (I_n - K_tH)\mathbb{E}\left(e_t^-v_t^\top\right)K_t^\top + K_t\mathbb{E}\left(v_te_t^{-\top}\right)(I_n - K_tH)^\top + K_t\mathbb{E}\left(v_tv_t^\top\right)K_t^\top\\ &= (I_n - K_tH)P_t^-(I_n - K_tH)^\top + (I_n - K_tH)\mathbb{E}\left(e_t^-\right)\mathbb{E}\left(v_t^\top\right)K_t^\top + K_t\mathbb{E}(v_t)\mathbb{E}\left(e_t^{-\top}\right)(I_n - K_tH)^\top + K_tRK_t^\top && \mathbb{E}(v_tv_t^T) = R\\ &= (I_n - K_tH)P_t^-(I_n - K_tH)^\top + 0_{n\times n} + 0_{n\times n} + K_tRK_t^\top && \mathbb{E}(v_t) = 0\\ &= (I_n - K_tH)P_t^-(I_n - K_tH)^\top + K_tRK_t^\top,\end{aligned}$$

which is exactly the covariance update equation.

Definition 5 (Covariance Update Equation)**.** Suppose that P_t^-, H and R are known. Then, the **covariance update equation** is given by

$$P_t = (I_n - K_tH)P_t^-(I_n - K_tH)^\top + K_tRK_t^\top,$$

where K_t is the Kalman gain.

[6]We use the subscript t, because the Kalman gain could differ in time.

By similar reasoning regarding that P_t^- remains symmetric and positive definite over time (right after Definition 3), the same holds for P_t.

Remark. There exists a simpler formulation of the covariance update equation which could be derived by plugging in the upcoming definition of the Kalman gain:

$$P_t = (I_n - K_t H)^{-1} P_t^-.$$

However, symmetry and positive definiteness are not ensured by this equation. Also, this update equation tends to be numerically unstable [11].

Let us return to the Kalman gain K_t. Recall that it measures our faith in our state predictions (and hence, also our measurements). If we e.g. consider the state update equation, note that for $K_t = 0$, $\hat{x}_t = \hat{x}_t^-$. This means that $K_t = 0_{n\times p}$ corresponds to us fully trusting our prediction.

Recall that the total variance of the error of the state estimator $\hat{x}_t$ equals $\operatorname{Tr} P_t$, which is the quantity we want to minimize. By controlling the Kalman gain, we can actually achieve this optimality. In other words, K_t should minimize the hidden state estimator's variance. We now derive the corresponding formula for K_t, using the covariance update equation. We first rewrite this equation by

$$\begin{aligned}
P_t &= (I_n - K_t H) P_t^- (I_n - K_t H)^\top + K_t R K_t^\top \\
&= (I_n - K_t H) P_t^- (I_n - H^\top K_t^\top) + K_t R K_t^\top \\
&= (P_t^- - K_t H P_t^-)(I_n - H^\top K_t^\top) + K_t R K_t^\top \\
&= P_t^- - P_t^- H^\top K_t^\top - K_t H P_t^- + K_t H P_t^- H^\top K_t^\top + K_t R K_t^\top \\
&= P_t^- - P_t^- H^\top K_t^\top - K_t H P_t^- + K_t (H P_t^- H^\top + R) K_t^\top,
\end{aligned}$$

which we use, together with the linearity of the trace operator, to compute the trace of P_n by

$$\begin{aligned}
\operatorname{Tr} P_t &= \operatorname{Tr}\left(P_t^- - P_t^- H^\top K_t^\top - K_t H P_t^- + K_t (H P_t^- H^\top + R) K_t^\top\right) \\
&= \operatorname{Tr} P_t^- - \operatorname{Tr}(P_t^- H^\top K_t^\top) - \operatorname{Tr}(K_t H P_t^-) + \operatorname{Tr}\left(K_t (H P_t^- H^\top + R) K_t^\top\right) \\
&= \operatorname{Tr} P_t^- - 2\operatorname{Tr}(K_t H P_t^-) + \operatorname{Tr}\left(K_t (H P_t^- H^\top + R) K_t^\top\right), \qquad (10)
\end{aligned}$$

where the last line follows from the fact that $\operatorname{Tr} A = \operatorname{Tr} A^\top$ for any square matrix A. Now, if we want to minimize (10) w.r.t K_t, two issues arise:

– How do we define the (partial) derivative of a scalar function w.r.t. a matrix?

– How do we differentiate the trace of a matrix?

Both issues are solved in Appendix A (Definition A.1 and Lemma A.1). Let us optimize P_t w.r.t. the Kalman gain K_t. First, note that $H P_t^- H^\top + R$ is symmetric, since $(H P_t^- H^\top)^\top = H (P_t^-)^\top H^\top = H P_t^- H^\top$, since P_t^- is symmetric (c.f. the reasoning right after Definition 3). Also, R is symmetric by the definition of a covariance matrix, and the sum of two matrices is symmetric as well. With this in mind, we compute

$$\begin{aligned}
\frac{\partial}{\partial K_t} \operatorname{Tr} P_t &= \frac{\partial}{\partial K_t}\left(\operatorname{Tr} P_t^- - 2\operatorname{Tr}(K_t H P_t^-) + \operatorname{Tr}\left(K_t (H P_t^- H^\top + R) K_t^\top\right)\right) \\
&= 0 - 2(H P_t^-)^\top + 2 K_t (H P_t^- H^\top + R) \\
&= -2\left(P_t^- H^\top - K_t (H P_t^- H^\top + R)\right).
\end{aligned}$$

We set $\frac{\partial}{\partial K_t} \operatorname{Tr} P_t = 0$ and solve

$$\begin{aligned}
-2\left(P_t^- H^\top - K_t (H P_t^- H^\top + R)\right) = 0 &\iff \\
P_t^- H^\top - K_t (H P_t^- H^\top + R) = 0 &\iff \\
K_t (H P_t^- H^\top + R) = P_t^- H^\top &\iff \\
K_t = P_t^- H^\top \left(H P_t^- H^\top + R\right)^{-1}&,
\end{aligned}$$

provided this inverse exists. This equation is exactly the third KF equation: the Kalman gain.

Definition 6 (Kalman Gain)**.** Assume that P_t^-, H and R are known. Then, the **Kalman gain** is given by

$$K_t = P_t^- H^\top \left(HP_t^- H^\top + R\right)^{-1}.$$

Regarding the existence of the inverse of $HP_t^- H^\top + R$, note that by similar reasoning as for P_t^- and P_t, $HP_t^- H^\top + R$ is positive definite as well. Hence, it is invertible, meaning that the existence of the inverse of this matrix is guaranteed.

Let us consider the Kalman gain in more detail. Recall that the Kalman gain minimizes the total variance of our a posteriori hidden state estimate in the covariance update equation. This is in line with our discussion about minimizing the risk in Section 2.2.1. Intuitively speaking, the Kalman gain is computed as

$$K_t = \frac{\text{uncertainty in a priori estimate}}{\text{uncertainty in a priori estimate} + \text{uncertainty in measurement}}.$$

That is, the Kalman gain measures our faith in our a priori state estimate, compared to our faith in our measurements. Whenever the first is large (i.e. low uncertainty), the Kalman gain should result in an a posteriori state estimate which is close to our priori state estimate. Vice versa, whenever we put more faith in our measurements, the Kalman gain gives a rather large contribution to the innovation in the state update equation. Observe that this behaviour is contained in the definition of the Kalman gain. If the uncertainty in our a priori state estimate $\hat{x}_t^-$ is small, then the entries of P_t^- are small. In the (entry-wise) limiting case, we have

$$\begin{aligned}
\lim_{P_t^- \to 0} K_t &= 0, \\
\lim_{R \to 0} K_t &= \lim_{R \to 0} P_t^- H^\top \left(HP_t^- H^\top + R\right)^{-1} \\
&= P_t^- H^\top \left(HP_t^- H^\top\right)^{-1} \\
&= H_L^{-1} P_t^- H^\top \left(HP_t^- H^\top\right)^{-1} \\
&= H_L^{-1},
\end{aligned}$$

where H_L^{-1} is the left-inverse of H (i.e. $H_L^{-1} H = I_p$). When we plug these limiting cases into the state update equation, we obtain

$$\hat{x}_t = \hat{x}_t^- + 0(z_t - H\hat{x}_t^-) = \hat{x}_t^-,$$

for $P_t^- \to 0$. For $R \to 0$, we have

$$\begin{aligned}
\hat{x}_t &= \hat{x}_t^- + H_L^{-1}(z_t - H\hat{x}_t^-) \\
&= \hat{x}_t^- + H_L^{-1} z_t - \hat{x}_t^- \\
&= H_L^{-1} z_t \\
&= H_L^{-1}(Hx_t + v_t) \\
&= H_L^{-1}(Hx_t + 0) && v_t \sim \mathcal{N}_p(0, R) \text{ with } R \to 0 \\
&= x_t
\end{aligned}$$

Hence, if the measurements are noise-free, the state update equation gives a estimate equal to the true hidden state realization. To conclude, the Kalman gain updates the state estimate and its covariance matrix based on the ratio between uncertainty about our a priori estimate and the measurement uncertainty.

Now that we have derived the equations which form the Kalman filter, let us consider a brief summary. Given a set of measurements $\{z_t\}_{t=1}^T$ and an initial condition for the (a posteriori) state estimate $\hat{x}_0$, the Kalman filter propagates the state estimate, together with its error covariance matrix, for $t \in \{1, \ldots, T\}$ by the following five equations.

State extrapolation equation	$\hat{x}_t^- = F\hat{x}_{t-1} + Gu_{t-1}$
Covariance extrapolation equation	$P_t^- = FP_{t-1}F^\top + Q$
Kalman gain	$K_t = P_t^- H^\top \left(HP_t^- H^\top + R\right)^{-1}$
State update equation	$\hat{x}_t = \hat{x}_t^- + K_t(z_t - H\hat{x}_t^-)$
Covariance update equation	$P_t = (I_n - K_t H)P_t^-(I_n - K_t H)^\top + K_t R K_t^\top$

Table 3: The five equations which form the Kalman filter

A more detailed schematic overview of the Kalman filter than in Figure 2 is given in Figure 3.

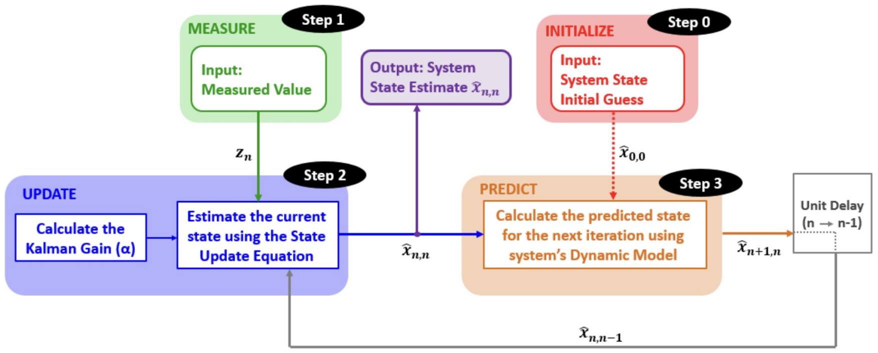

MEASURE
Step 1
Input: Measured Value
Output: System State Estimate $\hat{x}_{n,n}$
INITIALIZE
Step 0
Input: System State Initial Guess
z_n
$\hat{x}_{0,0}$
Step 2
UPDATE
Calculate the Kalman Gain (α)
Estimate the current state using the State Update Equation
$\hat{x}_{n,n}$
PREDICT
Step 3
Calculate the predicted state for the next iteration using system's Dynamic Model
$\hat{x}_{n+1,n}$
Unit Delay (n → n-1)
$\hat{x}_{n,n-1}$

Figure 3: A detailed overview of the Kalman filter. Here, n denotes t, so $\hat{x}_{n,n-1}$ denotes $\hat{x}_t^-$ and $\hat{x}_{n,n}$ denotes $\hat{x}_t$. The figure is retrieved from kalmanfilter.net

Now that we have characterized the Kalman filter, let us review the unbiasedness of the state estimators, something we already have discussed in Section 2.2.1.

Proposition 1. The a priori- and a posteriori state estimates $\hat{x}_t^-$ and $\hat{x}_t$ are unbiased if $\hat{x}_0$ is unbiased.

Proof. Suppose that $\hat{x}_0$ is unbiased. $\hat{x}_t^-$ is an unbiased estimator of x_t if $\mathbb{E}(\hat{x}_t^-) = \mathbb{E}(x_t)$. Similarly, $\hat{x}_t$ is unbiased if $\mathbb{E}(\hat{x}_t) = \mathbb{E}(x_t)$. We show that this holds for all $t \in \{1, \ldots, T\}$ by mathematical induction.

By the state extrapolation equation, we compute $\hat{x}_1^- = F\hat{x}_0 + Gu_0$, so we have

$$\begin{aligned}\mathbb{E}(\hat{x}_1^-) &= \mathbb{E}(F\hat{x}_0 + Gu_0)\\ &= F\mathbb{E}(\hat{x}_0) + Gu_0\\ &= F\mathbb{E}(x_0) + Gu_0,\end{aligned}$$

since $\hat{x}_0$ is an unbiased estimator for x_0. From (9), we have $x_1 = Fx_0 + Gu_0 + w_0$. Hence,

$$\begin{aligned}\mathbb{E}(x_1) &= \mathbb{E}(Fx_0 + Gu_0 + w_0)\\ &= F\mathbb{E}(x_0) + Gu_0 + \mathbb{E}(w_0)\\ &= F\mathbb{E}(x_0) + Gu_0 && \mathbb{E}(w_0) = 0\\ &= \mathbb{E}(\hat{x}_1^-).\end{aligned}$$

Hence, $\hat{x}_1^-$ is an unbiased estimator for x_1. By the state update equation, we have $\hat{x}_1 = \hat{x}_1^- + K_1(z_1 - H\hat{x}_1^-)$. We compute

$$\begin{aligned}
\mathbb{E}(\hat{x}_1) &= \mathbb{E}(\hat{x}_1^- + K_1(z_1 - H\hat{x}_1^-)) \\
&= \mathbb{E}(\hat{x}_1^-) + K_1\mathbb{E}(z_1 - H\hat{x}_1^-) \\
&= \mathbb{E}(x_1) + K_1\mathbb{E}(Hx_1 + v_1 - H\hat{x}_1^-) && \hat{x}_1^- \text{ is unbiased} \\
&= \mathbb{E}(x_1) + K_1(H(\mathbb{E}(x_1) - \mathbb{E}(\hat{x}_1^-)) + \mathbb{E}(v_1))) \\
&= \mathbb{E}(x_1) + K_1(H(\mathbb{E}(x_1) - \mathbb{E}(x_1)) + 0) && \mathbb{E}(v_1) = 0 \\
&= \mathbb{E}(x_1) + K_1 0 \\
&= \mathbb{E}(x_1).
\end{aligned}$$

Hence, for $t = 1$, $\hat{x}_t^-$ and $\hat{x}_t$ are both unbiased estimators for x_1. Now, suppose that for any $\tau \in \{2, \dots, T-1\}$, $\hat{x}_\tau^-$ and $\hat{x}_\tau$ are unbiased. Similarly as for $t = 1$, the computations show that $\hat{x}_{\tau+1}^-$ and $\hat{x}_{\tau+1}$ are unbiased estimators for $x_{\tau+1}$. This concludes the induction step.

Hence, by mathematical induction, we conclude that $\hat{x}_t^-$ and $\hat{x}_t$ are unbiased estimators for x_t $\forall t \in \{1, \dots, T\}$. $\square$

2.2.3 Probabilistic Origins

In Section 2.2.2, we have derived the five equations that form the Kalman filter. We now dive deeper into the probabilistic aspect. To this end, we again consider the Wiener problem. We alter our notation by describing the hidden states and measurements as the random variables X_t and Z_t respectively[7], while their realizations are denoted by x_t and z_t respectively. We assume that, conditioned on the current measurement history, X_t is normally distributed[8]. However we already mentioned it in Section 2.2.1, but it plays a key role in the upcoming derivations, we again point out that X_t, the future process noise w_t and the measurement noise v_t are assumed to be independent. The same holds for the measurement random variables Z_t w.r.t. future w_t and not current v_t. Furthermore, we denote the probability density function (pdf) of a random variable Y by $f_Y(y)$.

Suppose that we are at time $t-1$ and we have just collected the measurement z_{t-1}. From a Bayesian point of view, we are interested in the pdf of X_{t-1}, conditioned on the current measurement history and how we can propagate this to time t, after observing z_t. Before we continue, we define $\mathbf{Z}_t = (Z_1, \dots, Z_t)^\top \in \mathbb{R}^{tp}$, which is a tp-dimensional stacked random vector, and denote its realization by $\mathbf{z}_t = (z_1, \dots, z_t) \in \mathbb{R}^{tp}$. This enables us to write e.g. the conditional density of $X_t|(\mathbf{Z}_t = \mathbf{z}_t)$. Recall that conditionals are abbreviated throughout this thesis, meaning that we write the preceding as $X_t|\mathbf{Z}_t$.

We assume that $X_{t-1}|\mathbf{Z}_{t-1} \sim \mathcal{N}_n(\hat{x}_{t-1}, P_{t-1})$. We are able to verify this assumption by mathematical induction if we assume $X_0 \sim \mathcal{N}_n(\hat{x}_0, P_0)$ (note that at time $t = 0$, we do not have collected any measurement, so there is no measurement history to condition on). What follows actually counts as the induction step, meaning that such an inductive proof could be completed by verifying by e.g. propagating from time 0 to time 1 (after collecting our first measurement).

We want to propagate from time $t-1$ to time t after observing z_t. We decompose this into two steps: propagating from time $t-1$ to time t before observing z_t, which we denote by t^-, and propagating from time t^- to time t, right after collecting the measurement z_t. We start with the first propagation, meaning that we aim to learn about the pdf of $X_t|\mathbf{Z}_{t-1}$. Recall the dynamics of the hidden state, given by (9):

$$X_t = FX_{t-1} + Gu_{t-1} + w_{t-1}.$$

Since X_t is a linear combination of random variables X_{t-1} and w_{t-1} (Gu_t is considered to be fixed and is therefore not relevant in the derivation), $X_t|\mathbf{Z}_{t-1}$ is Gaussian if $(X_{t-1}, w_{t-1})|\mathbf{Z}_{t-1}$ is jointly Gaussian distributed. By the definition of conditional probability, we have

$$f_{X_{t-1}, w_{t-1}|\mathbf{Z}_{t-1}}(x_{t-1}, w_{t-1}|\mathbf{z}_{t-1}) = \frac{f_{X_{t-1}, w_{t-1}, \mathbf{Z}_{t-1}}(x_{t-1}, w_{t-1}, \mathbf{z}_{t-1})}{f_{\mathbf{Z}_{t-1}}(\mathbf{z}_{t-1})}.$$

[7]Note that X_t and Z_t correspond to a given time point, following the properties of a Markov chain.

[8]Note that we already argued this less formal in Section 2.2.1.

Since w_{t-1} is assumed to be independent of X_{t-1} and $\mathbf{Z}_{t-1}$, we have $f_{X_{t-1},w_{t-1},\mathbf{Z}_{t-1}}(x_{t-1},w_{t-1},\mathbf{z}_{t-1}) = f_{X_{t-1},\mathbf{Z}_{t-1}}(x_{t-1},\mathbf{z}_{t-1})f_{w_{t-1}}(w_{t-1})$, which gives

$$\begin{aligned} f_{X_{t-1},w_{t-1}|\mathbf{Z}_{t-1}}(x_{t-1},w_{t-1}|\mathbf{z}_{t-1}) &= \frac{f_{X_{t-1},\mathbf{Z}_{t-1}}(x_{t-1},\mathbf{z}_{t-1})f_{w_{t-1}}(w_{t-1})}{f_{\mathbf{Z}_{t-1}}(\mathbf{z}_{t-1})} \\ &= f_{X_{t-1}|\mathbf{Z}_{t-1}}(x_{t-1}|\mathbf{z}_{t-1})f_{w_{t-1}}(w_{t-1}), \end{aligned}$$

which is Gaussian, since $X_{t-1}|\mathbf{Z}_{t-1}$ is assumed to be Gaussian and w_{t-1} is defined as Gaussian and the product of two Gaussian random variables is Gaussian. Hence, $X_t|\mathbf{Z}_{t-1}$ is Gaussian distributed random variable. Its mean is given by

$$\begin{aligned} \mathbb{E}(X_t|\mathbf{Z}_{t-1}) &= \mathbb{E}(FX_{t-1}+Gu_{t-1}+w_{t-1}|\mathbf{Z}_{t-1}) \\ &= F\mathbb{E}(X_{t-1}|Z_{t-1})+Gu_{t-1}+\mathbb{E}(w_{t-1}|\mathbf{Z}_{t-1}) \\ &= F\hat{x}_{t-1}+Gu_{t-1}+\mathbb{E}(w_{t-1}) && w_t \text{ is independent of } \mathbf{Z}_t \\ &= F\hat{x}_{t-1}+Gu_{t-1}+0=\hat{x}_t^-. \end{aligned}$$

Note the appearance of the state extrapolation equation in the last line. Similarly as in the derivation of the covariance extrapolation equation (Section 2.2.2), the covariance matrix of $X_t|\mathbf{Z_{t-1}}$ is given by

$$\mathbb{E}\left((X_t-\hat{x}_t^-)(X_t-\hat{x}_t^-)^\top|\mathbf{Z}_{t-1}\right) = P_t^-.$$

Hence, we have $X_t|\mathbf{Z}_{t-1}\sim\mathcal{N}_n(\hat{x}_t^-,P_t^-)$. We take the conditional mean of $X_t|\mathbf{Z}_{t-1}$, x_t^-, as our estimate of X_t at time t^-. This is in line with the choice of our a priori state estimate in Section 2.2.2.

Now, we move on to the second propagation, from t^- to t. This implies that we have collected our measurement z_t, so instead of conditioning on $\mathbf{Z}_{t-1}$, we now condition on $\mathbf{Z}_t$. We have

$$\begin{aligned} f_{X_t|\mathbf{Z}_t}(x_t|\mathbf{z}_t) &= \frac{f_{X_t,\mathbf{Z}_t}(x_t,\mathbf{z}_t)}{f_{\mathbf{Z}_t}(\mathbf{z}_t)} \\ &= \frac{f_{X_t,\mathbf{Z}_{t-1},Z_t}(x_t,\mathbf{z}_{t-1},z_t)}{f_{\mathbf{Z}_{t-1},Z_t}(\mathbf{z}_{t-1},z_t)} && \mathbf{Z}_t=(\mathbf{Z}_{t-1},Z_t)^\top \\ &= \frac{f_{Z_t|X_t,\mathbf{Z}_{t-1}}(z_t|x_t,\mathbf{z}_{t-1})f_{X_t,\mathbf{Z}_{t-1}}(x_t,\mathbf{z}_{t-1})}{f_{Z_t|\mathbf{Z}_{t-1}}(z_t|\mathbf{z}_{t-1})f_{\mathbf{Z}_{t-1}}(\mathbf{z}_{t-1})} \\ &= \frac{f_{Z_t|X_t,\mathbf{Z}_{t-1}}(z_t|x_t,\mathbf{z}_{t-1})f_{X_t|\mathbf{Z}_{t-1}}(x_t|\mathbf{z}_{t-1})f_{\mathbf{Z}_{t-1}}(\mathbf{z}_{t-1})}{f_{Z_t|\mathbf{Z}_{t-1}}(z_t|\mathbf{z}_{t-1})f_{\mathbf{Z}_{t-1}}(\mathbf{z}_{t-1})} \\ &= \frac{f_{Z_t|X_t,\mathbf{Z}_{t-1}}(z_t|x_t,\mathbf{z}_{t-1})f_{X_t|\mathbf{Z}_{t-1}}(x_t|\mathbf{z}_{t-1})}{f_{Z_t|\mathbf{Z}_{t-1}}(z_t|\mathbf{z}_{t-1})}. \end{aligned} \tag{11}$$

Hence, $X_t|\mathbf{Z}_t$ is Gaussian if each density in (11) is Gaussian.

First, we consider $Z_t|X_t,\mathbf{Z}_{t-1}$. Recall the measurement equation given by (9). Since we condition on $X_t=x_t$, the term Hx_t becomes fixed. Also, v_t is independent of X_t and $\mathbf{Z}_{t-1}$, meaning that Z_t is a sum of a fixed vector and a Gaussian-distributed random variable, which implies that $Z_t|X_t,\mathbf{Z}_{t-1}$ is Gaussian itself. Its expectation is computed by

$$\begin{aligned} \mathbb{E}(Z_t|X_t,\mathbf{Z}_{t-1}) &= \mathbb{E}(HX_t+v_t|X_t=x_t,\mathbf{Z}_{t-1}) \\ &= \mathbb{E}(Hx_t+v_t|\mathbf{Z}_{t-1}) \\ &= H\mathbb{E}(x_t|\mathbf{Z}_{t-1})+\mathbb{E}(v_t|\mathbf{Z}_{t-1}) \\ &= Hx_t+\mathbb{E}(v_t) && x_t \text{ is a constant and } v_t \text{ is independent of } \mathbf{Z}_{t-1} \\ &= Hx_t+0=Hx_t. \end{aligned}$$

Its covariance matrix is given by

$$\begin{aligned}\mathbb{E}\left((Z_t - Hx_t)((Z_t - Hx_t)^\top | \mathbf{Z}_{t-1}\right) &= \mathbb{E}(v_t v_t^\top | \mathbf{Z}_{t-1}) \\ &= \mathbb{E}(v_t v_t^\top) = R. && v_t \text{ is independent of } \mathbf{Z}_{t-1}\end{aligned}$$

Hence, $Z_t|(X_t, \mathbf{Z}_{t-1}) \sim \mathcal{N}_p(Hx_t, R)$. The second density in the numerator of (11) is that of $X_t|\mathbf{Z}_{t-1}$, which is already shown to be Gaussian.

Now, consider $Z_t|\mathbf{Z}_{t-1}$. Recall that Z_t is a linear combination of X_t and v_t. Hence, $Z_t|\mathbf{Z}_{t-1}$ is Gaussian if $(X_t, v_t)|\mathbf{Z}_{t-1}$ is jointly Gaussian. By, again, the definition of conditional probability, we have

$$\begin{aligned}f_{X_t, v_t|\mathbf{Z}_{t-1}} &= f_{v_t|X_t,\mathbf{Z}_{t-1}}(v_t|x_t, \mathbf{z}_{t-1}) f_{X_t|\mathbf{Z}_{t-1}}(x_t|\mathbf{z}_{t-1}) \\ &= f_{v_t}(v_t) f_{X_t|\mathbf{Z}_{t-1}}(x_t|\mathbf{z}_{t-1})\,. && v_t \text{ is independent of both } X_t \text{ and } \mathbf{Z}_{t-1}\end{aligned}$$

Since both densities on the RHS are Gaussian, $(X_t, v_t)|\mathbf{Z}_{t-1}$ is also Gaussian. Its expectation is given by

$$\begin{aligned}\mathbb{E}(Z_t|\mathbf{Z}_{t-1}) &= \mathbb{E}(HX_t + v_t|\mathbf{Z}_{t-1}) \\ &= H\mathbb{E}(X_t|\mathbf{Z}_{t-1}) + \mathbb{E}(v_t|\mathbf{Z}_{t-1}) \\ &= H\hat{x}_t^- + \mathbb{E}(v_t) && v_t \text{ is independent of } \mathbf{Z}_{t-1} \\ &= H\hat{x}_t^- + 0 = H\hat{x}_t^-.\end{aligned}$$

Its covariance matrix is given by

$$\begin{aligned}\mathbb{E}\left((Z_t - H\hat{x}_t^-)(Z_t - H\hat{x}_t^-)^\top|\mathbf{Z}_{t-1}\right) &= \mathbb{E}\left((HX_t + v_t - H\hat{x}_t^-)(HX_t + v_t - H\hat{x}_t^-)^\top|\mathbf{Z}_{t-1}\right) \\ &= \mathbb{E}\left((H(X_t - \hat{x}_t^-) + v_t)(H(X_t - \hat{x}_t^-) + v_t)^\top|\mathbf{Z}_{t-1}\right) \\ &= \mathbb{E}\left((H(X_t - \hat{x}_t^-) + v_t)((X_t - \hat{x}_t^-)^\top H^\top + v_t^\top)|\mathbf{Z}_{t-1}\right) \\ &= \mathbb{E}\left(H(X_t - \hat{x}_t^-)(X_t - \hat{x}_t^-)^\top H^\top) + H(X_t - \hat{x}_t^-)v_t^\top + v_t(X_t - \hat{x}_t^-)^\top H^\top + v_t v_t^\top)|\mathbf{Z}_{t-1}\right) \\ &= H\mathbb{E}(X_t - \hat{x}_t^-)(X_t - \hat{x}_t^-)^\top|\mathbf{Z}_{t-1})H^\top + H\mathbb{E}((X_t - \hat{x}_t^-)v_t^\top|\mathbf{Z}_{t-1}) + \mathbb{E}(v_t(X_t - \hat{x}_t^-)^\top|\mathbf{Z}_{t-1})H^\top + \\ &\quad \mathbb{E}(v_t v_t^\top)|\mathbf{Z}_{t-1}) \\ &= HP_t^- H^\top + H\mathbb{E}((X_t - \hat{x}_t^-)|\mathbf{Z}_{t-1})\mathbb{E}(v_t^\top|\mathbf{Z}_{t-1}) + \mathbb{E}(v_t|\mathbf{Z}_{t-1})\mathbb{E}(X_t - \hat{x}_t^-)^\top|\mathbf{Z}_{t-1})H^\top + R \\ &= HP_t^- H^\top + 0 + 0 + R = HP_t^- H^\top + R.\end{aligned}$$

To summarize, we have

$$\begin{aligned}Z_t|(X_t, \mathbf{Z}_{t-1}) &\sim \mathcal{N}_p(Hx_t, R), \\ X_t|\mathbf{Z}_{t-1} &\sim \mathcal{N}_n(\hat{x}_t^-, P_t^-), \\ Z_t|\mathbf{Z}_{t-1} &\sim \mathcal{N}_p(H\hat{x}_t^-, HP_t^- H^T + R).\end{aligned}$$

Substituting these densities into (11) gives

$$f_{X_t|\mathbf{Z}_t}(x_t|\mathbf{z}_t) = \frac{|HP_t^- H^T + R|^{\frac{1}{2}}}{(2\pi)^{\frac{n}{2}}|P_t^-|^{\frac{1}{2}}|R|^{\frac{1}{2}}} \exp\Bigg(-\frac{1}{2}\Big((z_t - Hx_t)^\top R^{-1}(z_t - Hx_t) + (x_t - \hat{x}_t^-)^\top (P_t^-)^{-1}(x_t - \hat{x}_t^-) - \\ (z_t - H\hat{x}_t^-)^\top (HP_t^- H^\top + R)^{-1}(z_t - H\hat{x}_t^-)\Big)\Bigg) \tag{12}$$

It is not directly obvious that (12) is the density of a Gaussian distribution, since the three determinant terms before the exponent would have to correspond to a single determinant term involving the actual covariance matrix of $X_t|\mathbf{Z}_t$. Similarly, the exponent should correspond to the standard Gaussian form, involving the mean and covariance matrix of $X_t|\mathbf{Z}_t$. However, [11] proves that this actually is the case, meaning that $X_t|\mathbf{Z}_t$ is Gaussian distributed. Furthermore, its mean and covariance matrix are given by $\hat{x}_t$ and P_t respectively, which are defined exactly through the state update equation and the covariance update equation, making use of the definition of the Kalman gain. Again, by taking the expectation of this Gaussian distribution, we obtain $\hat{x}_t$ as our estimate of X_t at time t, which is in line with the a posteriori hidden state estimate in Section 2.2.2.

Hence, to conclude this section, we have shown that under the assumptions of Gaussian states and measurements, together with independence between the states X_t and noise variables w_t and v_t (the same holds for the measurements Z_t), both a priori and a posteriori state estimates $\hat{x}_t^-$ and $\hat{x}_t$ (respectively) are optimal estimators for the state in the sense of maximal probability. From this, we can derive 95% confidence intervals for the hidden state rather straightforward:

$$\hat{x}_{ti} - 1.96\sqrt{P_{tii}} \leqslant X_{ti} \leqslant \hat{x}_{ti} + 1.96\sqrt{P_{tii}},\ i \in \{1, \dots, n\}, \tag{13}$$

where 1.96 corresponds to the 95% quantiles of the normal distribution. Note that this confidence interval is conditioned on the current measurement history $\mathbf{z}_t$. If one conditions on the previous measurement history (i.e. on $\mathbf{z}_{t-1}$) for the current hidden state X_t, one would have

$$\hat{x}_{ti}^- - 1.96\sqrt{P_{tii}^-} \leqslant X_{ti} \leqslant \hat{x}_{ti}^- + 1.96\sqrt{P_{tii}^-},\ i \in \{1, \dots, n\}.$$

As mentioned before, under the assumptions of Gaussian distributed hidden states and measurements and independence between the hidden states/measurements and the noise terms, the Kalman filter is the optimal solution to the Wiener problem. The filtering process uses the measurement history at time t ($\mathbf{z}_t$) to give a MVUE of the hidden state at time t, denoted by $\hat{x}_t$. However, could we improve our estimates by incorporating the entire measurement history $\mathbf{z}_T$? This is a natural question that arises whenever we apply a so-called forward algorithm as the Kalman filter. To answer this question, we move on to the next topic: Kalman smoothing.

2.3 Kalman Smoother

In the last section, we have introduced and derived the equations of the Kalman filter. We also have dived into the probabilistic aspects and observed how the Kalman filter propagates the normally distributed (hidden) state through time. As discussed at the end of Section 2.2, at time $t \in \{1, \dots, T\}$, we obtain the prior- and posterior conditional distributions of the state, denoted by $X_t|\mathbf{Z}_{t-1}$ and $X_t|\mathbf{Z}_t$ respectively. Since we only condition on the data (measurements) that we have up to time t/the present (for the updated estimates/posterior distribution), the Kalman filter is referred to as a *forward algorithm* (which explains why the Kalman filter is referred to as a filtering problem) [2].

2.3.1 The Smoothing Equations

Suppose that we have performed Kalman filtering up to time $T \in \mathbb{N}$, which is the time step at which we obtain our final measurement. This implies that we have obtained a posteriori state- and its covariance matrix estimations, denoted by $\hat{x}_1, \dots, \hat{x}_T$ and $P_1, \dots, P_T$ respectively. However, only our final estimates, at time T, are conditioned on the entire measurement history $\mathbf{z}_T$. A natural question that arises if we could, again, update all estimates, but now based on the entire measurement history. This is exactly the idea behind *Kalman smoothing* and an explicit procedure is given by the *Rauch-Tung-Striebel (RTS) smoother*[9]. Here, 'smoothing' stands for estimation of a random process in the past [2]. The Kalman smoother is a *backward algorithm* in the sense that it serves as a correction on the estimates made by a forward algorithm, based on knowledge of the full data. In this section, we derive the three equations which form the Kalman smoother.

Just as in Section 2.2.3, we are going to derive the Kalman smoother equations through a conditional density-viewpoint. To this end, we are after the conditional density of $X_t|\mathbf{Z}_T$ ($t \in \{1, \dots, T\}$). Note we already know the density of $X_T|\mathbf{Z}_T$ from the Kalman filter, since the estimate of X_T is already conditioned on the entire measurement history. The smoothed hidden state estimates and their covariance matrices are denoted by $\tilde{x}_t$ and $\tilde{P}_t$ respectively.

Suppose that after the procedure of Kalman filtering, we have obtained our hidden state estimates and the covariance matrices $\hat{x}_1^-, \dots, \hat{x}_T^-$, $P_1^-, \dots, P_T^-$, $\hat{x}_1, \dots, \hat{x}_T$ and $P_1, \dots, P_T$. Note that $\tilde{x}_T = \hat{x}_T$ and $\tilde{P}_T = P_T$ as discussed

[9]In fact, the RTS smoother is one smoothing procedure for an equidistant time grid.

above. Then, for $t \in \{T-1, \ldots, 1\}$ (note the reverse order, because we propagate backwards in time), we aim to find $\tilde{x}_t$ and $\tilde{P}_t$.

Suppose that we have arrived at time $t \in \{1, \ldots, T-1\}$ in the smoothing procedure. Hence, we know $\tilde{x}_{t+1}$ and $\tilde{P}_{t+1}$, so $X_{t+1}|\mathbf{Z}_T \sim \mathcal{N}_n\left(\tilde{x}_{t+1}, \tilde{P}_{t+1}\right)$.

Consider the joint conditional density $(X_t, X_{t+1})|\mathbf{Z}_t$, which is given by

$$(X_t, X_{t+1})|\mathbf{Z}_t \sim \mathcal{N}_{2n}\left(\begin{pmatrix} \hat{x}_t \\ \hat{x}_{t+1}^- \end{pmatrix}, \begin{pmatrix} P_t & P_t F^\top \\ F P_t & P_{t+1}^- \end{pmatrix}\right),$$

where the lower left- and upper right blocks of the covariance matrix follow from $\mathrm{Cov}(X_t, X_{t+1}|\mathbf{Z}_t) = \mathrm{Cov}(X_t, FX_t + Gu_t + w_t|\mathbf{Z}_t) = \mathrm{Cov}(X_t, X_t|\mathbf{Z}_t)F^\top + G\mathrm{Cov}(X_t, u_t|\mathbf{Z}_t) + \mathrm{Cov}(X_t, w_t|\mathbf{Z}_t) = P_t^\top + 0 + 0 = P_t F^\top$ since u_t is constant and w_t and X_t are independent and the property $\mathrm{Cov}(X_{t+1}, X_t|\mathbf{Z}_t) = \mathrm{Cov}(X_t, X_{t+1}|\mathbf{Z}_t)^\top$.

Since the hidden state is assumed to have the Markov property (Definition 1), $X_t|X_{t+1}$ is independent of all future measurements [14]. Hence, we have

$$f_{X_t|X_{t+1}, \mathbf{Z}_t} = f_{X_t|X_{t+1}, \mathbf{Z}_T}.$$

From this, together with Lemma 1, we find the following conditional distribution of X_t:

$$X_t|(X_{t+1} = x_{t+1}, \mathbf{Z}_T) \sim \mathcal{N}_n\left(\hat{x}_t + S_t(x_{t+1} - \hat{x}_{t+1}^-), P_t - S_t P_{t+1}^- S_t^\top\right), \tag{14}$$

where $S_t = P_t F^\top (P_{t+1}^-)^{-1}$ is called the *smoothing gain*.

By the definition of conditional probability, we have

$$f_{X_t, X_{t+1}|\mathbf{Z}_T} = f_{X_t|X_{t+1}, \mathbf{Z}_T} f_{X_{t+1}|\mathbf{Z}_T}.$$

The first density in the RHS is computed in the previous step and the second one is assumed to be already known. Hence, by Lemma 1, we obtain

$$X_t, X_{t+1}|\mathbf{Z}_T \sim \mathcal{N}_{2n}\left(\begin{pmatrix} \hat{x}_t + S_t(\tilde{x}_{t+1} - \hat{x}_{t+1}^-) \\ \tilde{x}_{t+1} \end{pmatrix}, \begin{pmatrix} S_t \tilde{P}_{t+1} S_t^\top + P_t - S_t P_{t+1}^- S_t^T & S_t \tilde{P}_{t+1} \\ \tilde{P}_{t+1} S_t^\top & \tilde{P}_{t+1} \end{pmatrix}\right),$$

meaning, from Lemma 1, that the marginal distribution $X_t|\mathbf{Z}_T$ is given by

$$X_t|\mathbf{Z}_T \sim \mathcal{N}_n\left(\hat{x}_t + S_t(\tilde{x}_{t+1} - \hat{x}_{t+1}^-), P_t + S_t(\tilde{P}_{t+1} - P_{t+1}^-)S_t^\top\right).$$

From this, we obtain the following equations for the Kalman smoother:

Smoothing gain	$S_t = P_t F^\top (P_{t+1}^-)^{-1}$
State correction	$\tilde{x}_t = \hat{x}_t + S_t(\tilde{x}_{t+1} - \hat{x}_{t+1}^-)$
Covariance correction	$\tilde{P}_t = P_t + S_t(\tilde{P}_{t+1} - P_{t+1}^-)S_t^\top$

Table 4: The three equations which form the Kalman smoother

Similarly as for the Kalman filter, we can derive the following 95% confidence interval with the smoothed estimate i.e. conditioned on $\mathbf{z}_T$:

$$\tilde{x}_{ti} - 1.96\sqrt{\tilde{P}_{tii}} \leqslant X_{ti} \leqslant \tilde{x}_{ti} + 1.96\sqrt{\tilde{P}_{tii}}, \; i \in \{1, \ldots, n\}. \tag{15}$$

2.3.2 The Smoothed Autocovariance

The Kalman smoother gives us backward recursions to obtain the smoothed estimates $\tilde{x}_t$ and its conditional covariance $\tilde{P}_t$. During the derivation of the EMKF algorithm in Section 2.5, we also need the (smoothed) 1-lag autocovariance $\tilde{V}_{t,t-1} = \text{Cov}(X_t, X_{t-1}|\mathbf{Z}_T)$. The derivation of the following recursions for $\tilde{V}_{t,t-1}$ can be found in [16]. The result is formulated by the following backward recursion for $\tilde{V}_{t,t-1}$. For $t \in \{T-1, \dots, 1\}$, we have

$$\begin{aligned} \tilde{V}_{T,T-1} &= (I_n - K_T H) F P_{T-1}, \\ \tilde{V}_{t,t-1} &= P_t S_{t-1}^{\top} + S_t(\tilde{V}_{t+1,t} - F P_t) S_{t-1}^{\top} \end{aligned} \tag{16}$$

Combining Kalman filtering with the smoothing procedure as described here usually gives us accurate estimates of the hidden states x_t. However, the output of the Kalman filter/smoother heavily depends on the parameter set $\{F, G, H, Q, R\}$. Although these matrices are usually not difficult to derive in physical processes, in other fields (e.g. finance), these derivations are not trivial at all. Usually, whenever we know the realizations of the hidden states, we could easily estimate (e.g. by maximum likelihood) the parameters and vice versa. However, when knowledge of both is lacking, the situation becomes sort of a 'chicken and egg problem'. A typical approach to maximum likelihood estimation problems involving missing data (e.g. the hidden state realizations) is the expectation-maximization algorithm, which we now consider.

2.4 Expectation-Maximization

As discussed in Section 2.3, the Kalman filter/smoother itself is not always enough to infer about the hidden state. If the dynamics of the underlying system are not explicitly known, we do not know e.g. F and H explicitly. Furthermore, even if they are, the uncertainty of both the process and the measurements, expressed in R and Q, are also generally hard to know beforehand. One possible solution, which is of interest in this thesis, is to use the concept of expectation-maximization to not only infer about the hidden states, but about the dynamics simultaneously. In this section, we dive into the theory about the expectation-maximization algorithm and discuss its convergence properties.

The *expectation-maximization algorithm* (EM algorithm) is one of the most widely used algorithms in statistics/machine learning whenever the data is considered to be incomplete [4]. One popular example is the one about a two-component mixture model [7]. Here, and in many other cases, maximum likelihood (ML) estimation is hard to perform based on the observed data itself e.g. because its log-likelihood is hard to optimize analytically. Incomplete data may mean that the data is literally incomplete (e.g. there are some missing observations) or that it is assumed that the observed data is influenced by some process that is unobservable. It is assumed that whenever this *latent data* would be known (observable), maximum likelihood estimation is more straightforward to execute. In the Kalman filter setting, the observed data are the measurements $\{z_t\}_{t=1}^{T}$, while the latent data are the (realizations of the) hidden states $\{x_t\}_{t=0}^{T}$.

[16] proposes the Newton-Rapshon method next to the EM algorithm for maximum likelihood estimation in the Kalman filter setting. Newton-Rapshon is popular because its convergence rate is rather large, especially compared to the EM algorithm. However, the computational complexity of Newton-Rapshon increases rapidly with the number of parameters to-be estimated, because it requires computation and inversion of the Hessian of the log-likelihood of the observed data for each iteration. On the other hand, the computations carried out by the EM algorithm are rather simple, compared to inversion of a large matrix. Therefore, in the Kalman filter setting, the EM algorithm is often preferred, but it depends on the dimensions of the hidden state and measurements.

In line with earlier notation, we denote the observed data by $\mathbf{Z}$, while the latent data is denoted by $\mathbf{X}$. Their realizations are denoted by $\mathbf{z}$ and $\mathbf{x}$ respectively. The complete data is the collection of the observed- and latent data, given by $\mathbf{T} = \mathbf{Z} \cup \mathbf{X}$. Furthermore, we assume that the density of the complete data depends on a collection of parameters, denoted by $\boldsymbol{\theta}$[10]. As before, we denote the density of random variable X by f_X. Hence, the density

[10]The set of parameters defining the distributions of $\mathbf{Z}$ and $\mathbf{X}$ are also contained in θ.

of the complete data is given by $f_{\mathbf{T}}(\mathbf{t};\theta)$. By the definition of conditional probability, we have

$$\begin{aligned} f_{\mathbf{X}|\mathbf{Z};\theta} &= \frac{f_{\mathbf{Z},\mathbf{X};\theta}}{f_{\mathbf{Z};\theta}} = \frac{f_{\mathbf{T};\theta}}{f_{\mathbf{Z};\theta}} \Longleftrightarrow \\ f_{\mathbf{Z};\theta} &= \frac{f_{\mathbf{T};\theta}}{f_{\mathbf{X}|\mathbf{Z};\theta}} \Longleftrightarrow \\ \mathcal{L}(\theta|\mathbf{z}) &= \frac{\mathcal{L}(\theta|\mathbf{t})}{\mathcal{L}(\theta|(\mathbf{x}|\mathbf{z}))}, \end{aligned} \tag{17}$$

where $\mathcal{L}(\cdot)$ denotes the likelihood function. By taking the natural logarithm of (17), we get an equation in terms of log-likelihoods given by

$$\ell(\theta;\mathbf{Z}) = \ell_0(\theta;\mathbf{T}) - \ell_1(\theta;\mathbf{X}|\mathbf{Z}).,\tag{18}$$

where the subscripts on the RHS are meant to distinguish different log-likelihoods. Note that the LHS of (18) is independent of the missing data $\mathbf{X}$, while the RHS does depend on the missing data. Taking the conditional expectation of (18) w.r.t. the conditional density of $\mathbf{T}|\mathbf{Z}$ with parameter set θ' yields

$$\ell(\theta;\mathbf{Z}) = \mathbb{E}(\ell_0(\theta;\mathbf{T})|\mathbf{Z},\theta') - \mathbb{E}(\ell_1(\theta;\mathbf{X}|\mathbf{Z})|\mathbf{Z},\theta').\tag{19}$$

Let us consider (19) in some more detail. Note that taking the expectation w.r.t. $\mathbf{T}|\mathbf{Z}$ is equivalent to taking the expectation w.r.t. $\mathbf{X}|\mathbf{Z}$. Hence, since the LHS is independent of the missing data, it is treated as a constant in the considered expectation. Hence, taking the expectation w.r.t $\mathbf{X}|\mathbf{Z}$ yields the same likelihood again. Note that the first term in the RHS of (19) is defined as

$$\mathbb{E}(\ell_0(\theta;\mathbf{T})|\mathbf{Z},\theta') = \int_{\mathbf{x}\in\mathcal{X}} \log f_{\mathbf{T}}(\mathbf{z},\mathbf{x};\theta) f_{\mathbf{X}|\mathbf{Z}}(\mathbf{x}|\mathbf{z};\theta')\mathrm{d}\mathbf{x}.\tag{20}$$

Since $\mathbf{Z}$ is observed and hence, known, θ' is assumed to be known and $\mathbf{x}$ is marginalized out, the expected log-likelihood (20) becomes a function of θ.

Note that by the assumptions of EM, maximizing $\ell_0(\theta;\mathbf{T})$ should be more straightforward than maximizing $\ell(\theta;\mathbf{Z})$. However, we need the actual realizations of $\mathbf{X}$ to be able to maximize ℓ_0. Since the data is assumed to be missing, we need an alternative. This is exactly where $\mathbb{E}(\ell_0(\theta;\mathbf{T})|\mathbf{Z},\theta')$, or the *expected log-likelihood of the complete data*, becomes relevant. If we denote the value for θ which maximizes the expected log-likelihood by θ^*, then it can be shown that $\ell_0(\theta^*;\mathbf{Z}) \geqslant \ell_0(\theta';\mathbf{Z})$. This is justified by *Jensen's inequality* [7].

Lemma 4 (Jensen's inequality)**.** Let Y be a random variable on the (possibly infinite) interval (a,b), $a,b \in \overline{\mathbb{R}}$ and let the function g be differentiable and convex on (a,b). If $\mathbb{E}(Y)$ and $\mathbb{E}(g(Y))$ both exist, then

$$\mathbb{E}(g(Y)) \geqslant g(\mathbb{E}(Y)).$$

Proof. Since g is convex, for any $y_0 \in [a,b]$, we have

$$g(y) \geqslant g(y_0) + (y - y_0)g'(y_0)\ \forall y \in (a,b).$$

As $Y \in (a,b)$, $\mathbb{E}(Y) \in (a,b)$ as well. Hence, by taking $y = Y$ and $y_0 = \mathbb{E}(Y)$, we obtain

$$g(Y) \geqslant g(\mathbb{E}(Y)) + (Y - \mathbb{E}(Y))g'(\mathbb{E}(Y)).$$

By taking the expectation (w.r.t. Y) of the above, we obtain

$$\mathbb{E}(g(Y)) \geqslant g(\mathbb{E}(Y)) + (\mathbb{E}(Y) - \mathbb{E}(Y))g'(\mathbb{E}(Y)) = g(\mathbb{E}(Y)).$$

□

An important consequence of Jensen's inequality is Theorem 1.

Theorem 1. Let Y be a random variable with density f_Y defined by a set of parameters θ. Moreover, let φ be the true parameter. Then, if we take the expectation of $\log(f_Y(y))$ w.r.t. the density of Y defined by φ, we have

$$\mathbb{E}(\log(f_Y(y;\varphi))) \geqslant \mathbb{E}(\log(f_Y(y;\theta))).$$

Proof. We compute

$$\mathbb{E}(\log(f_Y(y;\varphi))) - \mathbb{E}(\log(f_Y(y;\theta))) = \mathbb{E}\left(-\log\left(\frac{f_Y(y;\theta)}{f_Y(y;\varphi)}\right)\right).$$

Since $-\log y$ is a convex function, we can apply Jensen's inequality with $g(y) = -\log y$ and $\tilde{Y} = \frac{f_Y(y;\theta)}{f_Y(y;\varphi)}$:

$$\begin{aligned}
\mathbb{E}\left(-\log\left(\frac{f_Y(y;\theta)}{f_Y(y;\varphi)}\right)\right) &\geqslant -\log\left(\mathbb{E}\left(\frac{f_Y(y;\theta)}{f_Y(y;\varphi)}\right)\right)\\
&= -\log\left(\int_{y\in S_Y} \frac{f_Y(y;\theta)}{f_Y(y;\varphi)} f_Y(y;\varphi)\mathrm{d}y\right)\\
&= -\log\left(\int_{y\in S_Y} f_Y(y;\theta)\mathrm{d}y\right)\\
&= -\log(1) = 0, \qquad f_Y \text{ is a pdf and the integral is over the sample space of } Y.
\end{aligned}$$

where S_y denotes the sample space of Y. Hence, we obtain

$$\mathbb{E}(\log(f_Y(y;\varphi))) \geqslant \mathbb{E}(\log(f_Y(y;\theta))).$$

□

Let θ^* maximize $\mathbb{E}(\ell_0(\theta;\mathbf{T})|\mathbf{Z},\theta')$ i.e.

$$\theta^* = \underset{\theta\in\Theta}{\operatorname{argmax}}\{\mathbb{E}(\ell_0(\theta;\mathbf{T})|\mathbf{Z},\theta')\},$$

where Θ is the parameter space of θ. This space depends on the set of parameters one wishes to estimate. Also, from (18) we have

$$\begin{aligned}
\ell(\theta';\mathbf{Z}) &= \mathbb{E}(\ell_0(\theta';\mathbf{T})|\mathbf{Z},\theta') - \mathbb{E}(\ell_1(\theta';\mathbf{X}|\mathbf{Z})|\mathbf{Z},\theta'),\\
\ell(\theta^*;\mathbf{Z}) &= \mathbb{E}(\ell_0(\theta^*;\mathbf{T})|\mathbf{Z},\theta') - \mathbb{E}(\ell_1(\theta^*;\mathbf{X}|\mathbf{Z})|\mathbf{Z},\theta').
\end{aligned}$$

Theorem 1 implies

$$\mathbb{E}(\ell_1(\theta';\mathbf{X}|\mathbf{Z})|\mathbf{Z},\theta') - \mathbb{E}(\ell_1(\theta^*;\mathbf{X}|\mathbf{Z})|\mathbf{Z},\theta') \geqslant 0,$$

since θ' is considered to be the true parameter here. This means that

$$\ell(\theta^*;\mathbf{Z}) - \ell(\theta';\mathbf{Z}) \geqslant \mathbb{E}(\ell_0(\theta^*;\mathbf{T})|\mathbf{Z},\theta') - \mathbb{E}(\ell_0(\theta';\mathbf{T})|\mathbf{Z},\theta'). \tag{21}$$

Note that the RHS of (21) must be nonnegative, since θ^* maximizes $\mathbb{E}(\ell_0(\theta;\mathbf{T})|\mathbf{Z},\theta')$. Hence, we obtain

$$\ell(\theta^*;\mathbf{Z}) \geqslant \ell(\theta';\mathbf{Z}).$$

The above result is actually crucial for the EM algorithm to converge: it tells us that if we alternate between computing the expected log-likelihood of the complete data (20) and maximizing this expectation, the log-likelihood of the observed data does not decrease at any time. It also implicitly tells us why we can maximize ℓ_0 instead of ℓ.

We now state the EM algorithm formally. We first initialize the parameter setting $\hat{\theta}^{(0)}$. Then, at iteration $i \in \mathbb{N}$, we perform two steps:

1. *Expectation step (E-step)*: Compute

$$Q_{\text{EM}}\left(\theta, \hat{\theta}^{(i)}\right) = \mathbb{E}(\ell_0\left(\theta; \mathbf{T})|\mathbf{Z}, \hat{\theta}^{(i)}\right).$$

2. *Maximization step (M-step)*: Compute

$$\hat{\theta}^{(i+1)} = \operatorname*{argmax}_{\theta \in \Theta} \left\{Q_{\text{EM}}\left(\theta, \hat{\theta}^{(i)}\right)\right\}.$$

Note the subscript 'EM', which is used to distinghuish the function Q_{EM} from the process noise covariance matrix Q. We alternate between the E-step and the M-step until some (pre-defined) convergence criterion is met. Whenever that occurs, we terminate the algorithm and we obtain the output $\hat{\theta} = \hat{\theta}^{(i^*)}$, where i^* is the iteration number when convergence is achieved.

The EM algorithm is visualized in Figure 4.

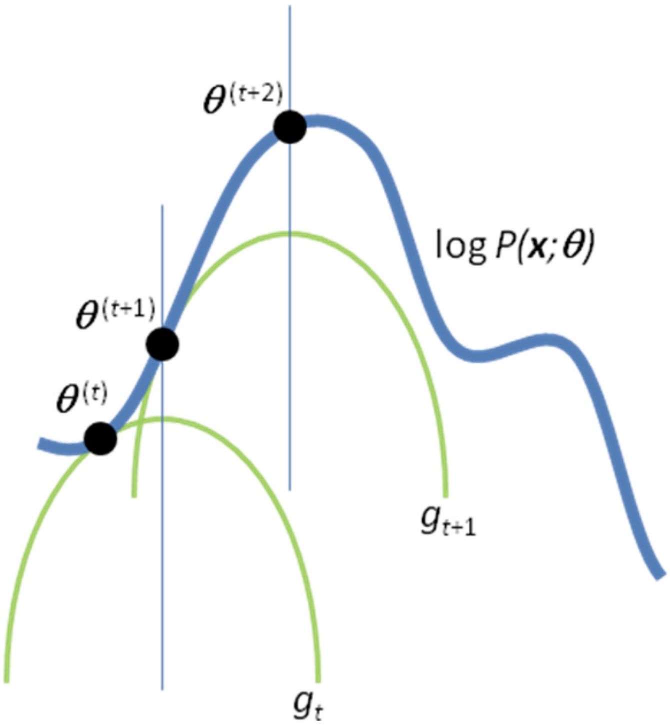


Figure 4: A visualization of several iterations of the EM algorithm. The E-step corresponds to finding the function Q_{EM} (here, denoted by g) which serves as a lower bound for the objective $\ell(\theta; Z)$. In the M-step, we maximize this lower bound in θ to update our estimate for θ. The figure is retrieved from people.duke.edu.

Typical stopping criteria for the EM algorithm are:

1. The increase in log-likelihood of the observed data is smaller than a pre-defined threshold i.e. $\ell(\hat{\theta}^{(i+1)}; \mathbf{Z}) - \ell(\hat{\theta}^{(i)}; \mathbf{Z}) < \epsilon, \ \epsilon > 0$.

2. The new MLE $\hat{\theta}^{(i+1)}$ is not far enough from the current MLE $\hat{\theta}^{(i)}$ i.e. $||\hat{\theta}^{(i+1)} - \hat{\theta}^{(i)}||_p < \epsilon$. Usually, $p \in \{1, 2\}$. In this thesis, we take $p = 1$.

We conclude this section by stating that the EM algorithm yields an increase the log-likelihood of the observed data, $\ell(\theta; \mathbf{Z})$ after each iteration. However, convergence to the global maximum of $\ell(\theta; \mathbf{Z})$ is not guaranteed. The EM algorithm can yield parameter estimates which in fact corresponds to a local maximum of the likelihood function, or a saddle point. Generally, the EM algorithm guarantees convergence to a stationary point of the log-likelihood of the observed data. Therefore, the algorithm is often executed multiple times with different initializations for the parameters. The parameters which come from the run which give the largest log-likelihood of the observed data could then be regarded as maximum likelihood estimates for the true parameters. In Theorem 2, we consider a method to check whether the output of the EM algorithm to the Kalman filter correspond to a maximum of the likelihood function.

2.5 Learning the Kalman Filter Parameters with an EM Approach: the EMKF algorithm

This section is devoted to tackle the problem of ignorance about the parameters of a Kalman filter and the realizations of the hidden states. We ignore the input term in the dynamics of the Wiener problem in the derivations, meaning that $u_t = 0 \; \forall t \in \{0, \dots, T\}$. The parameter set is given by $\theta = \{F, Q, H, R\}$, assuming that the initial condition x_0 is fixed. If not, we model X_0, as before, as a Gaussian distribution: $X_0 \sim \mathcal{N}_n(\xi, \Lambda)$. $\hat{x}_0$ is then taken as the mean of this distribution i.e. $\hat{x}_0 = \xi$. In this case, the parameter set becomes $\theta = \{F, Q, H, R, \xi, \Lambda\}$. The resulting algorithm is the main algorithm of this thesis and is referred to as Expectation-Maximization for the Kalman filter (EMKF) algorithm.

Recall the dynamics of the system given by (9). Since X_t and Z_t are considered to be Gaussian distributed random variables, we can write

$$X_t | (X_{t-1} = x_{t-1}) \sim \mathcal{N}_n(F x_{t-1}, Q),$$
$$Z_t | (X_t = x_t) \sim \mathcal{N}_p(H x_t, R).$$

Therefore, by using the definition of the multivariate normal distribution, we have

$$f_{X_0}(x_0) = \frac{1}{(2\pi)^{n/2} |\Lambda|^{1/2}} \exp\left(-\frac{1}{2} (x_0 - \xi)^\top \Lambda^{-1} (x_0 - \xi) \right),$$
$$f_{X_t|X_{t-1}}(x_t | x_{t-1}) = \frac{1}{(2\pi)^{n/2} |Q|^{1/2}} \exp\left(-\frac{1}{2} (x_t - F x_{t-1})^\top Q^{-1} (x_t - F x_{t-1}) \right),$$
$$f_{Z_t|X_t}(z_t | x_t) = \frac{1}{(2\pi)^{p/2} |R|^{1/2}} \exp\left(-\frac{1}{2} (z_t - H x_t)^\top R^{-1} (z_t - H x_t) \right).$$

Define $\mathbf{X} = (X_0, \dots, X_T)$ and $\mathbf{Z} = (Z_1, \dots, Z_T) = \mathbf{Z}_T$, where $\mathbf{Z}_T$ is defined in Section 2.2.3. Their realizations, $(\mathbf{z}, \mathbf{x})$ are referred to as the complete data. Since $\mathbf{z}$ are the measurements for the Kalman filter (i.e. what is observed), $\mathbf{z}$ forms the observed data. Hence, intuitively, $\mathbf{x}$ is the latent data in our setting, which agrees with the hidden state methodology.

2.5.1 The Expected Log-Likelihood of the Complete Data

Consider the joint density of $(\mathbf{Z}, \mathbf{X})$. By the definition of conditional probablity, we have

$$f_{\mathbf{Z},\mathbf{X}}(\mathbf{z}, \mathbf{x}) = f_{\mathbf{Z}|\mathbf{X}}(\mathbf{z}|\mathbf{x}) f_{\mathbf{X}}(\mathbf{x}).$$

By the Markov property of the system, we have

$$f_{\mathbf{Z}|\mathbf{X}}(\mathbf{z}|\mathbf{x}) = \prod_{t=1}^{T} f_{Z_t|X_t}(z_t | x_t),$$
$$f_{\mathbf{X}}(\mathbf{x}) = f_{X_0}(x_0) \prod_{t=1}^{T} f_{X_t|X_{t-1}}(x_t | x_{t-1}).$$

Hence, the joint density of $(\mathbf{Z}, \mathbf{X})$, as the likelihood of the complete data, is given by

$$f_{\mathbf{Z},\mathbf{X}}(\mathbf{z}, \mathbf{x}) = \mathcal{L}(\theta | \mathbf{z}, \mathbf{x}) = \prod_{t=1}^{T} f_{Z_t|X_t}(z_t | x_t) f_{X_0}(x_0) \prod_{t=1}^{T} f_{X_t|X_{t-1}}(x_t | x_{t-1}).$$

From $\mathcal{L}(\theta|\mathbf{z},\mathbf{x})$, we compute the log-likelihood $\ell_0(\theta|\mathbf{z},\mathbf{x})$, where the subscript is used to be consistent with the notation in Section 2.4. We compute

$$\begin{aligned}
\ell_0(\theta|\mathbf{z},\mathbf{x}) &= \log \mathcal{L}(\theta|\mathbf{z},\mathbf{x}) \\
&= \sum_{t=1}^{T} \log f_{Z_t|X_t}(z_t|x_t) + \log f_{X_0}(x_0) + \sum_{t=1}^{T} \log f_{X_t|X_{t-1}}(x_t|x_{t-1}) \\
&= \sum_{t=1}^{T} \log\left(\frac{1}{(2\pi)^{p/2}|R|^{1/2}} \exp\left(-\frac{1}{2}(z_t - Hx_t)^\top R^{-1}(z_t - Hx_t)\right)\right) + \log\bigg(\frac{1}{(2\pi)^{n/2}|\Lambda|^{1/2}} \cdot \\
&\quad \exp\left(-\frac{1}{2}(x_0 - \xi)^\top \Lambda^{-1}(x_0 - \xi)\right)\bigg) + \sum_{t=1}^{T} \log\bigg(\frac{1}{(2\pi)^{n/2}|Q|^{1/2}} \cdot \\
&\quad \exp\left(-\frac{1}{2}(x_t - Fx_{t-1})^\top Q^{-1}(x_t - Fx_{t-1})\right)\bigg) \\
&= \sum_{t=1}^{T}\left(\log\left(\frac{1}{(2\pi)^{p/2}|R|^{1/2}}\right) - \frac{1}{2}(z_t - Hx_t)^\top R^{-1}(z_t - Hx_t)\right) + \log\left(\frac{1}{(2\pi)^{n/2}|\Lambda|^{1/2}}\right) - \\
&\quad \frac{1}{2}(x_0 - \xi)^\top \Lambda^{-1}(x_0 - \xi) + \sum_{t=1}^{T}\left(\log\left(\frac{1}{(2\pi)^{n/2}|Q|^{1/2}}\right) - \frac{1}{2}(x_t - Fx_{t-1})^\top Q^{-1}(x_t - Fx_{t-1})\right) \\
&= -\frac{Tp}{2}\log(2\pi) - \frac{T}{2}\log|R| - \frac{1}{2}\sum_{t=1}^{T}(z_t - Hx_t)^\top R^{-1}(z_t - Hx_t) - \frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\Lambda| - \\
&\quad \frac{1}{2}(x_0 - \xi)^\top \Lambda^{-1}(x_0 - \xi) - \frac{Tn}{2}\log(2\pi) - \frac{T}{2}\log|Q| - \frac{1}{2}\sum_{t=1}^{T}(x_t - Fx_{t-1})^\top Q^{-1}(x_t - Fx_{t-1}) \\
&= -\frac{T(n+p)+n}{2}\log(2\pi) - \frac{T}{2}\log|R| - \frac{1}{2}\log|\Lambda| - \frac{T}{2}\log|Q| - \\
&\quad \frac{1}{2}\sum_{t=1}^{T}(z_t - Hx_t)^\top R^{-1}(z_t - Hx_t) - \frac{1}{2}(x_0 - \xi)^\top \Lambda^{-1}(x_0 - \xi) - \\
&\quad \frac{1}{2}\sum_{t=1}^{T}(x_t - Fx_{t-1})^\top Q^{-1}(x_t - Fx_{t-1}) \\
&= -\frac{T(n+p)+n}{2}\log(2\pi) - \frac{T}{2}\log|R| - \frac{1}{2}\log|\Lambda| - \frac{T}{2}\log|Q| - \frac{1}{2}\sum_{t=1}^{T}\big(z_t^\top R^{-1} z_t - z^\top R^{-1} H x_t - \\
&\quad x_t^\top H^\top R^{-1} z_t + x_t^\top H^\top R^{-1} H x_t\big) - \frac{1}{2}\big(x_0^\top \Lambda^{-1} x_0 - x_0^\top \Lambda^{-1}\xi - \xi^\top \Lambda^{-1} x_0 + \xi^\top \Lambda^{-1}\xi\big) - \\
&\quad \frac{1}{2}\sum_{t=1}^{T}\big(x_t^\top Q^{-1} x_t - x_t^\top Q^{-1} F x_{t-1} - x_{t-1}^\top F^\top Q^{-1} x_t - x_{t-1}^\top F^\top Q^{-1} F x_{t-1}\big). && Q^{-1} \text{ is symmetric} \\
&= -\frac{T(n+p)+n}{2}\log(2\pi) - \frac{T}{2}\log|R| - \frac{1}{2}\log|\Lambda| - \frac{T}{2}\log|Q| - \frac{1}{2}\sum_{t=1}^{T}\big(z_t^\top R^{-1} z_t - 2z^\top R^{-1} H x_t + \\
&\quad x_t^\top H^\top R^{-1} H x_t\big) - \frac{1}{2}\big(x_0^\top \Lambda^{-1} x_0 - 2x_0^\top \Lambda^{-1}\xi + \xi^\top \Lambda^{-1}\xi\big) - \frac{1}{2}\sum_{t=1}^{T}\big(x_t^\top Q^{-1} x_t - 2x_t^\top Q^{-1} F x_{t-1} + x_{t-1}^\top F^\top Q^{-1} F x_{t-1}\big). \\
&&& \text{Scalars are symmetric}
\end{aligned}$$

Hence, the log-likelihood of the parameter set θ, given the complete data, is given by

$$\ell_0(\theta|\mathbf{z},\mathbf{x}) = -\frac{T(n+p)+n}{2}\log(2\pi) - \frac{T}{2}\log|R| - \frac{1}{2}\log|\Lambda| - \frac{T}{2}\log|Q| - \frac{1}{2}\sum_{t=1}^{T}\left(z_t^\top R^{-1}z_t - 2z^\top R^{-1}Hx_t + x_t^\top H^\top R^{-1}Hx_t\right) - \frac{1}{2}\left(x_0^\top\Lambda^{-1}x_0 - 2x_0^\top\Lambda^{-1}\xi + \xi^\top\Lambda^{-1}\xi\right) - \frac{1}{2}\sum_{t=1}^{T}\left(x_t^\top Q^{-1}x_t - 2x_t^\top Q^{-1}Fx_{t-1} + x_{t-1}^\top F^\top Q^{-1}Fx_{t-1}\right)$$

Recall from Section 2.4 that in the E-step of the EM algorithm, we aim to find $Q_{\text{EM}}\left(\theta,\hat{\theta}^{(i)}\right)$, which is defined by (20) i.e. the expectation of the log-likelihood of the complete data, conditioned on the observed data and current parameter estimates $\hat{\theta}^{(i)} = \left\{\hat{F}^{(i)},\hat{Q}^{(i)},\hat{H}^{(i)},\hat{R}^{(i)},\hat{\xi}^{(i)},\hat{\Lambda}^{(i)}\right\}$. Hence, we aim to compute

$$Q_{\text{EM}}\left(\theta,\hat{\theta}^{(i)}\right) = \mathbb{E}\left(\ell_0(\theta|\mathbf{Z},\mathbf{X})|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right).$$

Let $\tilde{x}_t^{(i)}$ and $\tilde{P}_t^{(i)}$ be the output of the Kalman smoother, given parameter set $\hat{\theta}^{(i)}$. Note that since we are taking an expectation of the log-likelihood, the realizations $\mathbf{z}$ and $\mathbf{x}$ are replaced by the corresponding random variables $\mathbf{Z}$ and $\mathbf{X}$. By conditioning on $\mathbf{Z}=\mathbf{z}$, the linearity of the expectation operator and its invariance under the transpose operator, we can write

$$\begin{aligned}Q_{\text{EM}}\left(\theta,\hat{\theta}^{(i)}\right) =& \mathbb{E}\left(\ell_0(\theta|\mathbf{Z},\mathbf{X})|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right)\\ =& -\frac{T(n+p)+n}{2}\log(2\pi) - \frac{T}{2}\log|R| - \frac{1}{2}\log|\Lambda| - \frac{T}{2}\log|Q| - \frac{1}{2}\sum_{t=1}^{T}\left(z_t^\top R^{-1}z_t -\right.\\ & 2z_t^\top R^{-1}H\mathbb{E}\left(X_t|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(j)}\right) + \mathbb{E}\left(X_t^\top H^\top R^{-1}HX_t|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(j)}\right)\Big) - \frac{1}{2}\Big(\mathbb{E}\left(X_0^\top\Lambda^{-1}X_0|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(j)}\right) -\\ & 2\mathbb{E}\left(X_0|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(j)}\right)^\top\Lambda^{-1}\xi + \xi^\top\Lambda^{-1}\xi\Big) - \frac{1}{2}\sum_{t=1}^{T}\Big(\mathbb{E}\left(X_t^\top Q^{-1}X_t|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(j)}\right) - 2\mathbb{E}\left(X_t^\top Q^{-1}FX_{t-1}|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(j)}\right) +\\ & \mathbb{E}\left(X_{t-1}^\top F^\top Q^{-1}FX_{t-1}|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(j)}\right)\Big).\end{aligned}$$

Note that from the results of Section 2.3, we have $\mathbb{E}\left(X_t|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right) = \tilde{x}_t^{(i)}$ for $t\in\{1,\ldots,T\}$. For $t=0$, we have $\mathbb{E}\left(X_0|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right) = \xi^{(i)}$. Similarly, we have $\text{Var}\left(X_t|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right) = \tilde{P}_t^{(i)}$ for $t\in\{1,\ldots,T\}$ and $\text{Var}\left(X_0|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right) = \Lambda^{(i)}$. Also, from Lemma 2, we have

- $\mathbb{E}\left(X_t^\top AX_t|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right) = \mathbb{E}\left(X_t|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right)^\top A\mathbb{E}\left(X_t|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right) + \text{Tr}\left(A\text{Var}\left(X_t|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right)\right)$.
- $\mathbb{E}\left(X_t^\top AX_{t-1}|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right) = \mathbb{E}\left(X_t|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right)^\top A\mathbb{E}\left(X_{t-1}|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right) + \text{Tr}\left(A\text{Cov}\left(X_{t-1},X_t|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right)\right)$.

Note that from (16), we have $\text{Cov}\left(X_t,X_{t-1}|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right) = \tilde{V}_{t,t-1}^{(i)}$ and hence, $\text{Cov}\left(X_{t-1},X_t|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right) = \left(\tilde{V}_{t,t-1}^{(i)}\right)^\top$.

For $t\in\{1,\ldots,T\}$, we obtain

- $\mathbb{E}\left(X_t^\top AX_t|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right) = \left(\tilde{x}_t^{(i)}\right)^\top A\tilde{x}_t^{(i)} + \text{Tr}\left(A\tilde{P}_t^{(i)}\right)$.
- $\mathbb{E}\left(X_t^\top AX_{t-1}|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right) = \left(\tilde{x}_t^{(i)}\right)^\top A\tilde{x}_{t-1}^{(i)} + \text{Tr}\left(A\left(\tilde{V}_{t,t-1}^{(i)}\right)^\top\right)$.

Similarly, for $t=0$, we have $\mathbb{E}\left(X_0^\top AX_0|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right) = \left(\xi^{(i)}\right)^\top A\xi^{(i)} + \text{Tr}\left(A\Lambda^{(i)}\right)$. However, note that $\mathbb{E}\left(X_0|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right) = \tilde{x}_0^{(i)}$ and $\text{Var}\left(X_0|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right) = \tilde{P}_0^{(i)}$, meaning that we obtain

$$\mathbb{E}\left(X_0^\top AX_0|\mathbf{Z}=\mathbf{z},\hat{\theta}^{(i)}\right) = \left(\tilde{x}_0^{(i)}\right)^\top A\tilde{x}_0^{(i)} + \text{Tr}\left(A\tilde{P}_0^{(i)}\right).$$

Hence, the function $Q_{\text{EM}}\big(\theta,\hat{\theta}^{(i)}\big)$ is given by

$$Q_{\text{EM}}\big(\theta,\hat{\theta}^{(i)}\big) = -\frac{T(n+p)+n}{2}\log(2\pi) - \frac{T}{2}\log|R| - \frac{1}{2}\log|\Lambda| - \frac{T}{2}\log|Q| - \frac{1}{2}\sum_{t=1}^{T}\Big(z_t^\top R^{-1}z_t - 2z_t^\top R^{-1}H\tilde{x}_t^{(i)} + \left(\tilde{x}_t^{(i)}\right)^\top H^\top R^{-1}H\tilde{x}_t^{(i)} + \operatorname{Tr}\left(H^\top R^{-1}H\tilde{P}_t^{(i)}\right)\Big) - \frac{1}{2}\Big(\left(\tilde{x}_0^{(i)}\right)^\top \Lambda^{-1}\tilde{x}_0^{(i)} + \operatorname{Tr}\left(\Lambda^{-1}\tilde{P}_0^{(i)}\right) - 2\left(\tilde{x}_0^{(i)}\right)^\top \Lambda^{-1}\xi + \xi^\top\Lambda^{-1}\xi\Big) - \frac{1}{2}\sum_{t=1}^{T}\Big(\left(\tilde{x}_t^{(i)}\right)^\top Q^{-1}\tilde{x}_t^{(i)} + \operatorname{Tr}\left(Q^{-1}\tilde{P}_t^{(i)}\right) - 2\Big(\left(\tilde{x}_t^{(i)}\right)^\top Q^{-1}F\tilde{x}_{t-1}^{(i)} + \operatorname{Tr}\Big(Q^{-1}F\left(\tilde{V}_{t,t-1}^{(i)}\right)^\top\Big)\Big) + \left(\tilde{x}_{t-1}^{(i)}\right)^\top F^\top Q^{-1}F\tilde{x}_{t-1}^{(i)} + \operatorname{Tr}\left(F^\top Q^{-1}F\tilde{V}_{t,t-1}^{(i)}\right)\Big)\Big). \tag{22}$$

Hence, for the EM algorithm for the Kalman filter/smoother, the E-step is to compute the smoothed hidden state- and its covariance matrix estimates $\tilde{x}_t^{(i)}, \tilde{P}_t^{(i)}$ and the lag-1 autocovariance estimates $\tilde{V}_{t,t-1}^{(i)}$ ($t \in \{1,\ldots,T\}$), given the current parameter set $\hat{\theta}^{(i)}$, which are used to compute Q_{EM} as in (22).

2.5.2 The Log-Likelihood of the Measurements

Recall that the EM algorithm guarantees convergence to a (local) maximum of the log-likelihood of the observed data. Hence, the log-likelihood of the measurements (observed data) is used to determine whether convergence of the EM algorithm has been achieved (stopping criterion number 1 at the end of Section 2.4). This means that after each iteration of the EMKF algorithm, we have to evaluate the log-likelihood of the measurements in order to determine whether the increase in log-likelihood small enough. We now derive this log-likelihood.

The likelihood of the observed data is denoted by $L(\theta|\mathbf{z}) = f_{\mathbf{Z}}(\mathbf{z})$. We factorize this likelihood as

$$\begin{aligned} f_{\mathbf{Z}}(\mathbf{z}) &= f_{Z_1,\ldots,Z_T}(z_1,\ldots,z_T) \\ &= f_{Z_1}(z_1)\prod_{t=2}^{T} f_{Z_t|\mathbf{Z}_{t-1}}(z_t|\mathbf{z}_{t-1}). \end{aligned} \tag{23}$$

Note that in Section 2.2.3, we already established that $Z_t|\mathbf{Z}_{t-1} \sim \mathcal{N}_p(H\hat{x}_t^-, HP_t^-H^\top + R)$. This also holds for Z_1, since $\mathbf{Z}_0 = \emptyset$ (and hence, the conditional probability becomes a marginal probability). Substituting this result into (23) gives us, similar as before, the log-likelihood of the observed data $\mathbf{z}$:

$$\ell(\theta|\mathbf{z}) = -\frac{Tp}{2}\log(2\pi) - \frac{T}{2}\log\left|HP_t^-H^\top + R\right| - \frac{1}{2}\sum_{t=1}^{T}\left(z_t - H\hat{x}_t^-\right)^\top\left(HP_t^-H^\top + R\right)^{-1}\left(z_t - H\hat{x}_t^-\right). \tag{24}$$

Hence, by evaluating the log-likelihood of the measurements for current parameter estimates $\theta^{(i)}$ during each iteration, we can moderate how the log-likelihood increases throughout the EMKF algorithm. Furthermore, by computing the increase in log-likelihood after each iteration (i.e. $\ell\left(\theta^{(i)}|\mathbf{z}\right) - \ell\left(\hat{\theta}^{(i-1)}|\mathbf{z}\right)$), we can determine whether the algorithm has converged to a (local) maxima of the log-likelihood of the measurements.

2.5.3 The Parameter Update Equations

The M-step is about updating $\theta = \{F, Q, H, R, \xi, \Lambda\}$ such that Q_{EM} is maximized. Hence, we define

$$\theta^{(i+1)} = \underset{\theta\in\Theta}{\operatorname{argmax}}\, Q_{\text{EM}}\big(\theta,\hat{\theta}^{(i)}\big),$$

where Θ is, as defined in Section 2.4, the parameter space for θ[11]. This task obviously means that we have to

[11] In the Kalman filter/smoother setting, this space will be the Cartesian product of all vector/matrix spaces of the KF parameters. Due to the condition of symmetry and positive definiteness for the covariance matrices, we refrain from stating this space explicitly due to its complexity

solve the equations

$$\frac{\partial Q_{EM}}{\partial F}\big(\theta,\hat{\theta}^{(i)}\big)=0 \text{ for } F,\ \frac{\partial Q_{EM}}{\partial Q}\big(\theta,\hat{\theta}^{(i)}\big)=0 \text{ for } Q,\ \frac{\partial Q_{EM}}{\partial H}\big(\theta,\hat{\theta}^{(i)}\big)=0 \text{ for } H,$$
$$\frac{\partial Q_{EM}}{\partial R}\big(\theta,\hat{\theta}^{(i)}\big)=0 \text{ for } R,\ \frac{\partial Q_{EM}}{\partial \xi}\big(\theta,\hat{\theta}^{(i)}\big)=0 \text{ for } \xi \text{ and } \frac{\partial Q_{EM}}{\partial \Lambda}\big(\theta,\hat{\theta}^{(i)}\big)=0 \text{ for } \Lambda.$$

Let us start with the first equation. Using Lemma A.1, we compute

$$\begin{aligned}\frac{\partial Q_{EM}}{\partial F}\big(\theta,\hat{\theta}^{(i)}\big)=&\frac{\partial}{\partial F}\Bigg(-\frac{1}{2}\sum_{t=1}^{T}\Bigg(-2\Big(\left(\tilde{x}_t^{(i)}\right)^\top Q^{-1}F\tilde{x}_{t-1}^{(i)}+\mathrm{Tr}\Big(Q^{-1}F\left(\tilde{V}_{t,t-1}^{(i)}\right)^\top\Big)\Big)+\left(\tilde{x}_{t-1}^{(i)}\right)^\top F^\top Q^{-1}F\tilde{x}_{t-1}^{(i)}+\mathrm{Tr}\left(F^\top Q^{-1}F\tilde{V}_{t,t-1}^{(i)}\right)\Bigg)\Bigg)\\ =&-\frac{1}{2}\sum_{t=1}^{T}\Bigg(-2\left(Q^{-1}\tilde{x}_t^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top+Q^{-1}\tilde{V}_{t,t-1}^{(i)}\right)+2Q^{-1}F\tilde{x}_{t-1}^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top+2Q^{-1}F\tilde{P}_{t-1}^{(i)}\Bigg)\\ =&\sum_{t=1}^{T}\Bigg(\left(\tilde{x}_t^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top+\tilde{V}_{t,t-1}^{(i)}\right)-F\left(\tilde{x}_{t-1}^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top+\tilde{P}_{t-1}^{(i)}\right)\Bigg). \qquad Q^{-1} \text{ is invertible}\end{aligned}$$

Setting $\frac{\partial Q_{EM}}{\partial F}\big(\theta,\hat{\theta}^{(i)}\big)=0$ and solving for F gives

$$\begin{aligned}\sum_{t=1}^{T}\Bigg(\left(\tilde{x}_t^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top+\tilde{V}_{t,t-1}^{(i)}\right)-F\left(\tilde{x}_{t-1}^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top-\tilde{P}_{t-1}^{(i)}\right)\Bigg)=0 &\Leftrightarrow\\ F\sum_{t=1}^{T}\left(\tilde{x}_{t-1}^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top+\tilde{P}_{t-1}^{(i)}\right)=\sum_{t=1}^{T}\left(\tilde{x}_t^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top+\tilde{V}_{t,t-1}^{(i)}\right) &\Leftrightarrow\\ F\sum_{t=1}^{T}\left(\tilde{x}_{t-1}^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top+\tilde{P}_{t-1}^{(i)}\right)=\sum_{t=1}^{T}\left(\tilde{x}_t^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top+\tilde{V}_{t,t-1}^{(i)}\right) &\Leftrightarrow\\ F=\sum_{t=1}^{T}\left(\tilde{x}_t^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top+\tilde{V}_{t,t-1}^{(i)}\right)\left(\sum_{t=1}^{T}\left(\tilde{x}_{t-1}^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top+\tilde{P}_{t-1}^{(i)}\right)\right)^{-1}&,\end{aligned}$$

where the inverse on the last line always exists, since $\tilde{x}_{t-1}^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top$ is positive semi-definite and $\tilde{P}_{t-1}^{(i)}$ is positive definite, hence their sum is positive definite, which makes it invertible. The first observation follows from the fact that for any $y\in\mathbb{R}^n\setminus\{0\}$, we have

$$y^\top\tilde{x}_{t-1}^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top y=\left(\left(\tilde{x}_{t-1}^{(i)}\right)^\top y\right)^\top\left(\tilde{x}_{t-1}^{(i)}\right)^\top y=\left|\left|\left(\tilde{x}_{t-1}^{(i)}\right)^\top y\right|\right|^2\geqslant 0,$$

where $||\cdot||$ denotes a norm. Note that it also symmetric, which justifies the transposition on the third line. Also, any outer product has rank 1, $\tilde{x}_{t-1}^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top$ is singular for $n>1$, meaning that the inequality above is not strict.

Hence, the update equation for $\hat{F}$ is given by

$$\hat{F}^{(i+1)}=\sum_{t=1}^{T}\left(\tilde{x}_t^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top+\tilde{V}_{t,t-1}^{(i)}\right)\left(\sum_{t=1}^{T}\left(\tilde{x}_{t-1}^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top+\tilde{P}_{t-1}^{(i)}\right)\right)^{-1}$$

Let

$$A_1^{(i)}=\sum_{t=1}^{T}\left(\tilde{x}_t^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top+\tilde{V}_{t,t-1}^{(i)}\right) \text{ and } A_2^{(i)}=\sum_{t=1}^{T}\left(\tilde{x}_{t-1}^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top+\tilde{P}_{t-1}^{(i)}\right).$$

Then, we can rewrite the update equation for $\hat{F}$ as

$$\hat{F}^{(i+1)}=A_1^{(i)}\left(A_2^{(i)}\right)^{-1}.$$

We now move on to the update equation for $\hat{Q}$. Since the expected log-likelihood in (22) mostly involves the factor Q^{-1}, we chose to go with the typical approach (in ML estimation for covariance matrices) to differentiate the objective function w.r.t. Q^{-1} rather than Q. Using Lemma A.1, we compute

$$\begin{aligned}
\frac{\partial Q_{EM}}{\partial Q^{-1}}\left(\theta,\hat{\theta}^{(i)}\right) =& \frac{\partial}{\partial Q^{-1}}\Bigg(-\frac{T}{2}\log|Q| - \frac{1}{2}\sum_{t=1}^{T}\Bigg(\left(\tilde{x}_t^{(i)}\right)^\top Q^{-1}\tilde{x}_t^{(i)} + \operatorname{Tr}\left(Q^{-1}\tilde{P}_t^{(i)}\right) - 2\Bigg(\left(\tilde{x}_t^{(i)}\right)^\top Q^{-1}F\tilde{x}_{t-1}^{(i)} + \\
& \operatorname{Tr}\left(Q^{-1}F\left(\tilde{V}_{t,t-1}^{(i)}\right)^\top\right)\Bigg) + \left(\tilde{x}_{t-1}^{(i)}\right)^\top F^\top Q^{-1}F\tilde{x}_{t-1}^{(i)} + \operatorname{Tr}\left(F^\top Q^{-1}F\tilde{P}_{t-1}^{(i)}\right)\Bigg)\Bigg) \\
=& \frac{T}{2}Q^\top - \frac{1}{2}\sum_{t=1}^{T}\Bigg(\tilde{x}_t^{(i)}\left(\tilde{x}_t^{(i)}\right)^\top + \tilde{P}_t^{(i)} - 2\Bigg(\tilde{x}_t^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top F^\top + \tilde{V}_{t,t-1}^{(i)}F^\top\Bigg) + F\tilde{x}_{t-1}^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top F^\top + F\tilde{P}_{t-1}^{(i)}F^\top\Bigg) \\
=& \frac{T}{2}Q^\top - \frac{1}{2}\sum_{t=1}^{T}\Bigg(\tilde{x}_t^{(i)}\left(\tilde{x}_t^{(i)}\right)^\top + \tilde{P}_t^{(i)} - 2\Bigg(\tilde{x}_t^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top + \tilde{V}_{t,t-1}^{(i)}\Bigg)F^\top + F\Bigg(\tilde{x}_{t-1}^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top + \tilde{P}_{t-1}^{(i)}\Bigg)F^\top\Bigg) \\
=& \frac{T}{2}Q^\top - \frac{1}{2}\Bigg(\sum_{t=1}^{T}\Bigg(\tilde{x}_t^{(i)}\left(\tilde{x}_t^{(i)}\right)^\top + \tilde{P}_t^{(i)}\Bigg) - 2A_1^{(i)}F^\top + FA_2^{(i)}F^\top\Bigg).
\end{aligned}$$

Setting $\frac{\partial Q_{EM}}{\partial Q^{-1}}\left(\theta,\hat{\theta}^{(i)}\right) = 0$ and solving for Q gives

$$\begin{aligned}
\frac{T}{2}Q^\top - \frac{1}{2}\Bigg(\sum_{t=1}^{T}\Bigg(\tilde{x}_t^{(i)}\left(\tilde{x}_t^{(i)}\right)^\top + \tilde{P}_t^{(i)}\Bigg) - 2A_1^{(i)}F^\top + FA_2^{(i)}F^\top\Bigg) = 0 \Leftrightarrow \\
Q^\top = \frac{1}{T}\Bigg(\sum_{t=1}^{T}\Bigg(\tilde{x}_t^{(i)}\left(\tilde{x}_t^{(i)}\right)^\top + \tilde{P}_t^{(i)}\Bigg) - 2A_1^{(i)}F^\top + FA_2^{(i)}F^\top\Bigg) \Leftrightarrow \\
Q = \frac{1}{T}\Bigg(\sum_{t=1}^{T}\Bigg(\tilde{x}_t^{(i)}\left(\tilde{x}_t^{(i)}\right)^\top + \tilde{P}_t^{(i)}\Bigg) - 2F\left(A_1^{(i)}\right)^\top + FA_2^{(i)}F^\top\Bigg). \qquad & A_2^{(i)} \text{ is symmetric}
\end{aligned}$$

By using our current estimate for F $\left(\hat{F}^{(i+1)}\right)$, we obtain the following update equation for $\hat{Q}$:

$$\hat{Q}^{(i+1)} = \frac{1}{T}\Bigg(\sum_{t=1}^{T}\Bigg(\tilde{x}_t^{(i)}\left(\tilde{x}_t^{(i)}\right)^\top + \tilde{P}_t^{(i)}\Bigg) - 2\hat{F}^{(i+1)}\left(A_1^{(i)}\right)^\top + \hat{F}^{(i+1)}A_2^{(i)}\left(\hat{F}^{(i+1)}\right)^\top\Bigg).$$

By plugging in the definition of $\hat{F}^{(i+1)}$ and defining

$$A_3^{(i)} = \sum_{t=1}^{T}\Bigg(\tilde{x}_t^{(i)}\left(\tilde{x}_t^{(i)}\right)^\top + \tilde{P}_t^{(i)}\Bigg),$$

we obtain

$$\begin{aligned}
\hat{Q}^{(i+1)} =& \frac{1}{T}\Bigg(A_3^{(i)} - 2A_1^{(i)}\left(A_2^{(i)}\right)^{-1}\left(A_1^{(i)}\right)^\top + A_1^{(i)}\left(A_2^{(i)}\right)^{-1}A_2^{(i)}\left(A_1^{(i)}\left(A_2^{(i)}\right)^{-1}\right)^\top\Bigg) \\
=& \frac{1}{T}\Bigg(A_3^{(i)} - 2A_1^{(i)}\left(A_2^{(i)}\right)^{-1}\left(A_1^{(i)}\right)^\top + A_1^{(i)}\left(A_2^{(i)}\right)^{-1}A_2^{(i)}\left(A_2^{(i)}\right)^{-1}\left(A_1^{(i)}\right)^\top\Bigg) \\
=& \frac{1}{T}\Bigg(A_3^{(i)} - A_1^{(i)}\left(A_2^{(i)}\right)^{-1}\left(A_1^{(i)}\right)^\top\Bigg) \qquad (25) \\
=& \frac{1}{T}\Bigg(A_3^{(i)} - \hat{F}^{(i+1)}\left(A_1^{(i)}\right)^\top\Bigg).
\end{aligned}$$

Hence, the update equation for $\hat{Q}$ is given by

$$\hat{Q}^{(i+1)} = \frac{1}{T}\Bigg(A_3^{(i)} - \hat{F}^{(i+1)}\left(A_1^{(i)}\right)^\top\Bigg).$$

Note that this estimate is symmetric, which can be observed from (25) ($A_3^{(i)}$ is symmetric). It is less straightforward to show that $\hat{Q}^{i+1}$ is positive definite, which lies beyond the scope of this thesis.

Next, we derive the update equation for $\hat{H}$. In a similar approach as for the partial derivative w.r.t. F, we compute

$$\frac{\partial Q_{EM}}{\partial H}\left(\theta, \hat{\theta}^{(i)}\right) = \sum_{t=1}^{T}\left(z_t\left(\tilde{x}_t^{(i)}\right)^\top\right) - HA_3^{(i)}.$$

Setting $\frac{\partial Q_{EM}}{\partial H}\left(\theta, \hat{\theta}^{(i)}\right) = 0$ and solving for H gives, similarly for $\hat{F}$

$$H = A_4^{(i)}\left(A_3^{(i)}\right)^{-1},$$

where $A_4^{(i)}$ is given by

$$A_4^{(i)} = \sum_{t=1}^{T} z_t\left(\tilde{x}_t^{(i)}\right)^\top.$$

Note that the inverse of $A_3^{(i)}$ always exists by a similar reasoning as for $A_2^{(i)}$. Hence, the update equation for $\hat{H}$ is given by

$$\hat{H}^{(i+1)} = A_4^{(i)}\left(A_3^{(i)}\right)^{-1}.$$

Next up is the update equation for $\hat{R}$. As similar to the derivation of $\hat{Q}^{(i+1)}$, we compute

$$\frac{\partial Q_{EM}}{\partial R^{-1}}\left(\theta, \hat{\theta}^{(i)}\right) = \frac{T}{2}R^\top - \frac{1}{2}\sum_{t=1}^{T} z_t z_t^\top - 2A_4^{(i)}H^\top + HA_3^{(i)}H^\top.$$

Setting $\frac{\partial Q_{EM}}{\partial R^{-1}}\left(\theta, \hat{\theta}^{(i)}\right) = 0$ and solving for R gives, similarly for $\hat{Q}$,

$$\begin{aligned} R &= \frac{1}{T}\left(A_5 - A_4^{(i)}\left(A_3^{(i)}\right)^{-1}\left(A_4^{(i)}\right)^\top\right) \\ &= \frac{1}{T}\left(A_5 - \hat{H}^{(i+1)}\left(A_4^{(i)}\right)^\top\right), \end{aligned}$$

where A_5 is given by

$$A_5 = \sum_{t=1}^{T} z_t z_t^\top.$$

Note that A_5 is only dependent on the measurements. Hence, the update equation for $\hat{R}$ is given by

$$\hat{R}^{(i+1)} = \frac{1}{T}\left(A_5 - \hat{H}^{(i+1)}\left(A_4^{(i)}\right)^\top\right).$$

As similar to the structure of the update equation for $\hat{Q}$, we can observe that $\hat{R}$ remains symmetric throughout the EMKF algorithm. Regarding its positive definiteness, we have a similar argument as for $\hat{Q}$.

Until now, we have derived the update equations of the Kalman filter parameters F, Q, H and R. We now move on to the update equations for the initial conditions (recall that $X_0 \sim (\xi, \Lambda)$). We start with the update equation for $\hat{\xi}$. Again, using Lemma A.1, we compute

$$\begin{aligned}
\frac{\partial Q_{EM}}{\partial \xi}\left(\theta, \hat{\theta}^{(i)}\right) &= \frac{\partial}{\partial \xi}\left(-\frac{1}{2}\left(-\left(\tilde{x}_0^{(i)}\right)^\top \Lambda^{-1}\xi - \xi^\top \Lambda^{-1}\tilde{x}_0^{(i)} + \xi^\top \Lambda^{-1}\xi\right)\right) \\
&= -\frac{1}{2}\left(-\left(\tilde{x}_0^{(i)}\right)^\top \Lambda^{-1} - \left(\Lambda^{-1}\tilde{x}_0^{(i)}\right)^\top + 2\xi^\top \Lambda^{-1}\right) \\
&= -\frac{1}{2}\left(-\left(\tilde{x}_0^{(i)}\right)^\top \Lambda^{-1} - \left(\tilde{x}_0^{(i)}\right)^\top \Lambda^{-1} + 2\xi^\top \Lambda^{-1}\right) \\
&= \left(\tilde{x}_0^{(i)}\right)^\top \Lambda^{-1} - \xi^\top \Lambda^{-1} \\
&= \left(\tilde{x}_0^{(i)}\right)^\top - \xi^\top. && \Lambda^{-1} \text{ is invertible}
\end{aligned}$$

Setting $\frac{\partial Q_{EM}}{\partial \xi}\left(\theta, \hat{\theta}^{(i)}\right) = 0$ and solving for ξ obviously gives $\xi = \tilde{x}_0^{(i)}$. Hence, we obtain

$$\hat{\xi}^{(i+1)} = \tilde{x}_0^{(i)}.$$

Last, but not least, we derive the update equation for $\hat{\Lambda}$. Using Lemma A.1, we compute

$$\begin{aligned}
\frac{\partial Q_{EM}}{\partial \Lambda^{-1}}\left(\theta, \hat{\theta}^{(i)}\right) &= \frac{\partial}{\partial \Lambda^{-1}}\left(-\frac{1}{2}\log|\Lambda| - \frac{1}{2}\left(\left(\tilde{x}_0^{(i)}\right)^\top \Lambda^{-1}\tilde{x}_0^{(i)} + \operatorname{Tr}\left(\Lambda^{-1}\tilde{P}_0^{(i)}\right) - \left(\tilde{x}_0^{(i)}\right)^\top \Lambda^{-1}\xi - \xi^\top \Lambda^{-1}\tilde{x}_0^{(i)} + \xi^\top \Lambda^{-1}\xi\right)\right) \\
&= \frac{1}{2}\Lambda^T - \frac{1}{2}\left(\tilde{x}_0^{(i)}\left(\tilde{x}_0^{(i)}\right)^\top + \tilde{P}_0^{(i)} - \tilde{x}_0^{(i)}\xi^\top - \xi\left(\tilde{x}_0^{(i)}\right)^\top + \xi\xi^\top\right) \\
&= \frac{1}{2}\Lambda^T - \frac{1}{2}\left(\left(\tilde{x}_0^{(i)} - \xi\right)\left(\tilde{x}_0^{(i)} - \xi\right)^\top + \tilde{P}_0^{(i)}\right)
\end{aligned}$$

Setting $\frac{\partial Q_{EM}}{\partial \Lambda^{-1}}\left(\theta, \hat{\theta}^{(i)}\right) = 0$ and solving for Λ gives

$$\begin{aligned}
\frac{1}{2}\Lambda^T - \frac{1}{2}\left(\left(\tilde{x}_0^{(i)} - \xi\right)\left(\tilde{x}_0^{(i)} - \xi\right)^\top + \tilde{P}_0^{(i)}\right) = 0 \Longleftrightarrow \\
\Lambda^\top = \left(\tilde{x}_0^{(i)} - \xi\right)\left(\tilde{x}_0^{(i)} - \xi\right)^\top + \tilde{P}_0^{(i)}.
\end{aligned}$$

By the symmetry of the RHS of the last line and using the current update for $\hat{\xi}$, we obtain

$$\begin{aligned}
\hat{\Lambda}^{(i+1)} &= \left(\tilde{x}_0^{(i)} - \hat{\xi}^{(i+1)}\right)\left(\tilde{x}_0^{(i)} - \hat{\xi}^{(i+1)}\right)^\top + \tilde{P}_0^{(i)} \\
&= (\tilde{x}_0^{(i)} - \tilde{x}_0^{(i)})(\tilde{x}_0^{(i)} - \tilde{x}_0^{(i)})^\top + \tilde{P}_0^{(i)} \\
&= 0 + \tilde{P}_0^{(i)} = \tilde{P}_0^{(i)}.
\end{aligned}$$

Hence, the update equation for $\hat{\Lambda}$ is given by

$$\hat{\Lambda}^{(i+1)} = \tilde{P}_0^{(i)}.$$

It turns out that the update equations for $\hat{\xi}$ and $\hat{\Lambda}$ are given by the smoothed estimates of x_0 and its error covariance matrix P_0 respectively.

This ends the derivation of the EM algorithm for the Kalman filter: the EMKF algorithm. As a stopping criterion, one could use the stopping criteria which are mentioned in Section 2.4. Note that since the Kalman filter parameters are matrices, the latter stopping criterion should be adapted to the multivariate sense. In this thesis, we just consider the entry-wise convergence of the parameters. The thresholds should be taken in such a way that convergence is achieved, which could be a subjective matter. The thresholds which are used in this thesis are mentioned in Section 3.5. Whenever EMKF terminates, the obtained parameter estimates corresponds to a stationary point of the log-likelihood of the measurements. In Theorem 2, we show how one could classify the stationary point.

2.5.4 Summary

Let us summarize the algorithm. Starting with initial estimates $\hat{\theta}^{(0)} = \left\{\hat{F}^{(0)}, \hat{Q}^{(0)}, \hat{H}^{(0)}, \hat{R}^{(0)}, \hat{\xi}^{(0)}, \hat{\Lambda}^{(0)}\right\}$, we apply the Kalman- filter and smoother to obtain smoothed estimates of the hidden state $\left\{\tilde{x}_t^{(0)}\right\}_{t=0}^{T}$, state/error covariance matrices $\left\{\tilde{P}_t^{(0)}\right\}_{t=0}^{T}$ and smoothed lag-1 autocovariances $\left\{\tilde{V}_{t,t-1}^{(0)}\right\}_{t=1}^{T}$. These estimates are used to update the parameter estimates according to computing

$$A_1^{(i)} = \sum_{t=1}^{T}\left(\tilde{x}_t^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top + \tilde{V}_{t,t-1}^{(i)}\right),\ A_2^{(i)} = \sum_{t=1}^{T}\left(\tilde{x}_{t-1}^{(i)}\left(\tilde{x}_{t-1}^{(i)}\right)^\top + \tilde{P}_{t-1}^{(i)}\right),\ A_3^{(i)} = \sum_{t=1}^{T}\left(\tilde{x}_t^{(i)}\left(\tilde{x}_t^{(i)}\right)^\top + \tilde{P}_t^{(i)}\right)$$

$$A_4^{(i)} = \sum_{t=1}^{T} z_t\left(\tilde{x}_t^{(i)}\right)^\top, \text{ and } A_5 = \sum_{t=1}^{T} z_t z_t^\top,$$

which gives the following update equations:

$$\begin{aligned}
\hat{F}^{(i+1)} &= A_1^{(i)}\left(A_2^{(i)}\right)^{-1},\\
\hat{Q}^{(i+1)} &= \frac{1}{T}\left(A_3^{(i)} - A_1^{(i)}\left(A_2^{(i)}\right)^{-1}\left(A_1^{(i)}\right)^\top\right) = \frac{1}{T}\left(A_3^{(i)} - \hat{F}^{i+1)}\left(A_1^{(i)}\right)^\top\right),\\
\hat{H}^{(i+1)} &= A_4^{(i)}\left(A_3^{(i)}\right)^{-1},\\
\hat{R}^{(i+1)} &= \frac{1}{T}\left(A_5 - A_4^{(i)}\left(A_3^{(i)}\right)^{-1}\left(A_4^{(i)}\right)^\top\right) = \frac{1}{T}\left(A_5 - \hat{H}^{(i+1)}\left(A_4^{(i)}\right)^\top\right),\\
\hat{\xi}^{(i+1)} &= \tilde{x}_0^{(i)},\\
\hat{\Lambda}^{(i+1)} &= \tilde{P}_0^{(i)}.
\end{aligned} \tag{26}$$

The algorithm is formulated in Algorithm 1.

Algorithm 1: Expectation-maximization for the Kalman filter (EMKF)

```
Given: A time series of observations $\{z_t\}_{t=1}^T$;
A number of hidden states $n \in \mathbb{N}$;
Parameter set $\theta = \{F, Q, H, R, \xi, \Lambda\}$;
A log-likelihood increase tolerance level $\varepsilon_\ell$;
A parameter difference tolerance level $\varepsilon_\theta$
Initialize: Initial parameter setting $\hat{\theta}^{(0)} = \left\{\hat{F}^{(0)}, \hat{Q}^{(0)}, \hat{H}^{(0)}, \hat{R}^{(0)}, \hat{\xi}^{(0)}, \hat{\Lambda}^{(0)}\right\}$;
$i \leftarrow 0$;
while $\ell^{(i+1)} - \ell^{(i)} > \varepsilon_\ell$ and $\left|\hat{\theta}^{(i+1)} - \hat{\theta}^{(i)}\right| > \varepsilon_\theta$[a] do
    E-step
    Compute $\hat{x}_0^{(i)} = \xi^{(i)}$ and $P_0^{(i)} = \Lambda^{(i)}$;
    Compute $\left\{\left(\hat{x}_t^-\right)^{(i)}\right\}_{t=1}^T$, $\left\{\hat{x}_t^{(i)}\right\}_{t=1}^T$, $\left\{\left(P_t^-\right)^{(i)}\right\}_{t=1}^T$, $\left\{P_t^{(i)}\right\}_{t=1}^T$ by the Kalman filter with initial conditions
     $\hat{x}_0^{(i)}$ and $P_0^{(i)}$ and parameter setting $\hat{\theta}^{(i)}$ ;
    Compute $\left\{\tilde{x}_t^{(i)}\right\}_{t=1}^T$, $\left\{\tilde{P}_t^{(i)}\right\}_{t=1}^T$ and $\left\{\tilde{V}_{t,t-1}^{(i)}\right\}_{t=1}^T$ by the Kalman smoother with parameter setting $\hat{\theta}^{(i)}$;
    Compute $\ell^{(i)} = \ell\left(\hat{\theta}^{(i)} \middle| \mathbf{z}\right)$ by (24) using $\left\{\tilde{x}_t^{(i)}\right\}_{t=1}^T$;

    M-step
    Compute $\hat{\theta}^{(i+1)}$ according to (26);
    Compute $\left|\hat{\theta}^{(i+1)} - \hat{\theta}^{(i)}\right|$ for all entries in $\theta$.
end
```

[a]The difference is taken entry-wise for all parameters

2.5.5 Partial Optimization

Note that in some occasions, someone only wishes to optimize a subset of θ, because some parameters are already known. An occasion which could be suitable is the one where someone only wants to estimate the uncertainties e.g. Q and R (and maybe P_0). The EMKF algorithm could be suitably adapted in the E-step by running the Kalman filter/smoother with the fixed (known) parameters and the current estimates of the unknown parameters. However, one should avoid using the fixed parameters in the M-step (e.g. the known value for F to update $\hat{Q}$), since this could lead to negative-definite covariance matrices. The right way to update the unknown parameters is to use $A_j^{(i)}$ ($j \in \{1,2,3,4\}$) and A_5.

Although the EM(KF) algorithm yields us estimates of the log-likelihood of the measurements, these are only point estimates. We also want to know how precise our estimates are and therefore aim to learn more about the uncertainty in these estimates. Fortunately, asymptotic theory about the sampling distributions of the estimates coming from the EMKF algorithm exists, which we cover in the next section.

2.6 Asymptotic Theory and Confidence Intervals

The asymptotic properties of maximum likelihood estimates obtained from e.g. the EM algorithm (e.g. consistency and normality) of the Kalman filter have been studied rather generally in [3]. An essential condition is the stability of the estimated Kalman filter [16]. According to [1], stability of the Kalman filter is related to stability of the state equation, which is defined as follows.

Definition 7 (Stability)**.** Consider the state equation (6) without the input term. This state equation is called **stable** (or **causal**) if the spectrum of the state transition matrix F lies within the unit circle in the complex plain. That is,

$$\sigma(F) \subseteq \{z \in \mathbb{C} \mid |z| < 1\}.$$

Recall that the spectrum of a square matrix A is the set of all eigenvalues of A. Hence, when all eigenvalues of F are smaller than 1 in absolute value, the Kalman filter is said to be stable.

Stability of the filter is important because it yields innovation terms $z_t - H\hat{x}_t^-$ which stabilize as t increases, meaning that the covariance matrix of these innovations converges to a steady-state and does not depend on t anymore. This follows from the fact that the error covariance matrix P_t converges to a steady-state error covariance matrix P and the same argument holds for the Kalman gain K_t, which converts to steady-state gain K [16]. The steady-state error covariance matrix P solves the algebraic Ricatti equation

$$P = F\left(P - PH^\top(HPH^\top + R)^{-1}HP\right)F^\top + Q.$$

Furthermore, K satisfies

$$PH^\top(HPH^\top + R)^{-1},$$

which looks familiar to the definition of the Kalman gain K_t (Definition 6). [6] argues that an observability (see Section 2.1) of the system also yields the steady-state properties if the system is not stable, provided that $P_1^- \succ P$ (i.e. $P_1^- - P$ is positive definite), but not necessarily exponentially fast (c.f. Remark 3.3.3 from [6]).

Together with some additional assumptions, asymptotic normality of the parameter estimates of the state-space model in consideration is justified in e.g. [6, 16, 3]. To this end, define

$$\psi = \begin{pmatrix} \mathrm{vec}(F) \\ \mathrm{vec}(Q) \\ \mathrm{vec}(H) \\ \mathrm{vec}(R) \\ \xi \\ \mathrm{vec}(\Lambda) \end{pmatrix} \in \mathbb{R}^{n^2+(p+1)n+p^2},$$

where vec$(\cdot)$ is the vectorization operator. Note that ψ is just a vector containing the elements of θ. Then, we can state the following result.

Theorem 2. Let $\hat{\psi}$ be a maximum likelihood estimator of ψ. Assume that the following assumptions hold:

1. The system is stable.
2. ψ lies within the interior of the parameter space.
3. All partial derivatives of the log-likelihood of the measurements up to order three exist and are continuous in the neighbourhood of ψ.
4. ψ is identifiable.

Then, as $T \to \infty$, we have

$$\sqrt{T}(\hat{\psi} - \psi) \xrightarrow{d} \mathcal{N}_k(0_k, \mathcal{I}(\psi)^{-1},$$

where $k \in \mathbb{N}$ is the dimension of ψ and $\mathcal{I}(\psi)$ denotes the Fisher information matrix of ψ.

For the proof of Theorem 2, the reader is referred to [3]. A couple of remarks:

- Note that we denote the dimension of ψ by k, because we allow for cases where only a subset of the parameters have to be estimated (see the end of Section 2.5). Hence, the value of k could vary between situations. However, $k \leqslant n^2 + (p+1)n + p^2$. In the case of partial optimization, the dimension of ψ reduces, but the asymptotic distribution will be of the same dimension as ψ.
- The first assumption in Theorem 2 is called the *regularity condition*, which could be weakened to e.g. observability as discussed before. It should be noted that this regularity condition can be adopted to several types of state-space models [6].

Theorem 2 essentially means that if we manage to find maximum likelihood estimates for the parameters to be estimated, normality of the sampling distribution is justified if the sample size T is sufficiently large. Then, one would have

$$\hat{\psi} \sim \mathcal{N}_k\left(\psi, \frac{1}{T}\mathcal{I}(\psi)^{-1}\right).$$

The above result can be used to construct $(1-\alpha)\%$ confidence intervals for the parameter estimates. One would typically take an entry on the diagonal of $\mathcal{I}(\psi)^{-1}$ to obtain a confidence interval for the corresponding entry of the estimated matrix (or ξ, if it is estimated). For example, if the state-transition matrix F is estimated, then the first n^2 entries on the diagonal of $\mathcal{I}(\psi)^{-1}$ are the variances of the sampling distribution of the entries of $\hat{F}$ (column-wise, due to the definition of the vectorization operator). Using the general definition of the Fisher information matrix, in the setting of Kalman filtering, the *(expected) Fisher information matrix* is defined as follows.

$$\mathcal{I}(\psi) = -\mathbb{E}\left(\frac{\partial^2}{\partial\psi\partial\psi^\top}\ell(\psi|\mathbf{z})\middle|\psi\right). \tag{27}$$

As in the general case, the Fisher information matrix defined above is the negative expectation of the Hessian of the log-likelihood of the measurements (24) w.r.t. the unknown parameters.

2.6.1 Estimation of the Fisher Information Matrix

In order to obtain $(1-\alpha)\%$ confidence intervals for the parameter estimates, one would have to explicitly know the Fisher information matrix $\mathcal{I}(\psi)$. Recall the definition of the Fisher information matrix in (27). Hence, if we aim to compute the Fisher information for given parameters estimates $\hat{\psi}$, one would have to compute the Hessian of the log-likelihood of the measurements.

However, if we reconsider (19), [5] shows that the gradient of $\mathbb{E}(\ell_1(\psi; \mathbf{X}|\mathbf{Z})|\mathbf{Z}, \psi')$[12] (the latter term on the RHS of (19)) evaluated at $\psi = \hat{\psi}$ vanishes[13], which means that

$$\left.\frac{\partial \ell(\psi; \mathbf{z})}{\partial \psi}\right|_{\psi=\hat{\psi}} = \left.\frac{\partial Q_{\text{EM}}(\psi, \hat{\psi})}{\partial \psi}\right|_{\psi=\hat{\psi}}, \tag{28}$$

This result means that evaluated at a maximum likelihood estimate of ψ, the gradient of the log-likelihood of the observed data equals the gradient of the Q_{EM} function computed in the E-step of the EMKF algorithm. A similar argument holds for the Hessian in (27). Since we already have obtained the partial derivatives of all parameters in Section 2.5.3, we can compute the RHS in (28) once the EMKF algorithm has converged to maximum likelihood estimates. The evaluated gradient of Q_{EM} is then used to compute the Hessian in (27).

Although the Hessian should be analytically derivable, we numerically approximate it by a Monte Carlo simulation, following the methodology from [12].

Let $S_Q(\varphi) = \left.\frac{\partial Q_{\text{EM}}(\psi, \hat{\psi})}{\partial \psi}\right|_{\psi=\varphi}$. Note the difference between S_Q and the smoothing gain S_t (14). Let $\Delta = (\Delta_1, \ldots, \Delta_k) \in \mathbb{R}^k$ be a perturbation vector. According to [12], Δ should satisfy the following criteria $\forall i \in \{1, \ldots, k\}$:

- Δ_i has zero expectation and is symmetrically distributed.
- Δ_i is uniformly bounded and $\mathbb{E}(1/\Delta_i) < \infty$.
- Δ_i is independent from and identically distributed as $\Delta_j \forall j \in \{1, \ldots, k | j \neq i\}$.

A typical choice for Δ which satisfies the above criteria is a IID sample from the Bernoulli distribution with probability 0.5 of attaining $-c$ and c respectively, with $c \in \mathbb{R}$ a sufficiently small scalar. In this thesis, we take $c = 0.0005$. Next, by defining $\delta S_Q(\varphi) = S_Q(\varphi + \Delta) - S_Q(\varphi - \Delta)$, an approximation of the Hessian of the log-likelihood of the measurements at φ is given by

$$\hat{H}_s(\varphi) = \frac{1}{2}\left(\frac{\delta S_Q(\varphi)}{2}\begin{pmatrix}1/\Delta_1 & \cdots & 1/\Delta_k\end{pmatrix} + \left(\frac{\delta S_Q(\varphi)}{2}\begin{pmatrix}1/\Delta_1 & \cdots & 1/\Delta_k\end{pmatrix}\right)^{\top}\right). \tag{29}$$

Note that the second term ensures symmetry of the Hessian estimate. Supposing that we obtain parameter estimates from the EMKF algorithm, denoted by $\hat{\psi}$, one could approximate the Hessian by the following Monte Carlo simulation:

1. Set the large number of simulations $N \in \mathbb{N}$ and the scalar $c \in \mathbb{R}$, which should be rather small.
2. Generate a perturbation vector Δ by sampling Δ_i from the Bernoulli distribution mentioned above for $i \in \{1, \ldots, k\}$.
3. Apply (29) with $\hat{\psi}$ and Δ_k.
4. Repeat steps 2-3 by sampling a new perturbation vector Δ independently from the previous one $N-1$ times.
5. Compute the average of all computed matrices to obtain an approximation of the Hessian of the log-likelihood of the measurements.

Whenever we have an estimate of the Hessian, we can immediately check whether the estimate $\hat{\psi}$ is a (local) maximum of the log-likelihood of the measurements. Recall the second partial derivative test, which states that at a critical point ψ^* of the log-likelihood is a (local) maximum if the Hessian evaluated at ψ^* is negative definite. This means that the eigenvalues of this matrix are all negative. This makes sense, because the Fisher information matrix is then positive definite, according to (27), which is a requirement of a covariance matrix (in (2)). Hence,

[12]Note that ψ is just a vectorized form of θ appearing in (19).

[13]This results assumes that differentiation and integration are interchangeable.

estimation of the Fisher information is only valid if the estimate $\hat{\psi}$ corresponds to a (local) maximum of the log-likelihood function.

Recall that if the Hessian at ψ^* also has positive eigenvalues, then, according to the second order partial derivative test, then the corresponding critical point is a saddle point of the log-likelihood. This means that although the EM algorithm converged to estimates with zero gradient, it does not correspond to a local maximum of the log-likelihood[14].

Suppose that the obtained approximation of the Hessian matrix, denoted by $\hat{H}(\psi)$ is indeed negative definite. Then, the *observed Fisher information matrix* is approximated by

$$\mathcal{I}(\psi) \approx -\hat{H}_s\left(\hat{\psi}\right).$$

[14]The second order partial derivative test could also give the conclusion that the critical point is a local minimum, but, theoretically, the EM algorithm prevents convergence to a local minimum. Hence, we do not consider this case here.

3 Methods

As mentioned in the Introduction, the aim of this thesis is to apply the EMKF approach to a (multidimensional) time series of stock prices (measurements) to forecast their future values. In this section, we argue how the theoretical framework in Section 2 is translated into suitable prediction methods. In Section 3.1, we describe the data on which this application is based, before describing our approach in Section 3.4.

3.1 Data Description

We take the stock prices of the following companies from January 1st, 2016 up to August 1st, 2019:

1. Apple
2. Netflix
3. Tesla
4. Google

The data is retrieved from Yahoo Finance, using the `yfinance` API in Python. For each company, we take the adjusted closing price each day as its daily observation, giving us 901 observations. A plot of the resulting time series is given by Figure 5.

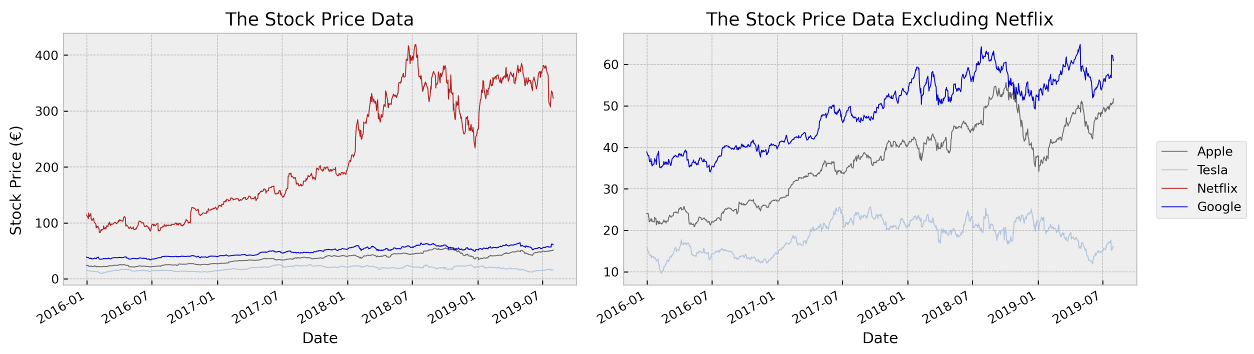


Figure 5: The stock price data. Netflix' stocks are excluded in the right figure due to the difference in scale.

3.2 Simulated Data

As a proof of concept of the EMKF algorithm, we first apply the methodology described in Section 3.4 to a simulated process. We consider two cases. The first one is a state-space model with approximately trivial parameters (Section 4.1.1), while the second one consists of randomized parameters, which is a more complex state-space model (Section 4.1.2). The methodology of these simulations is described in their respective sections in Section 4.

3.3 Example

After the simulations are performed in Section 4.1, we consider an example from the literature, aiming to reproduce the results. We consider Example 6.8 from [16].

3.4 Approach

We describe how we use the theoretical framework developed in Section 2 to the data to learn about the state-space model which describes the trajectory of the stock prices.

We are aiming to forecast the stock prices described in Section 3.1 through trend estimation. The standardized stock prices themselves serve as measurements. We standardize the stock prices company-specific, meaning that for each company, we use its specific mean and standard deviation. We are going to investigate how many trends (from one common trend for all four companies to one trend per company) are needed to provide accurate predictions. Since we consider four companies, we have $p = 4$. [5] proposes the decomposition of a trend in terms of a *level* μ_t and a *slope* β_t ($t \in \{1,\dots,T\}$), which yields the following model formulation:

$$\begin{aligned} z_t &= H_\mu \mu_t + v_t,\ v_t \sim \mathcal{N}_4(0,R) \\ \mu_t &= \mu_{t-1} + \beta_{t-1} + \epsilon_{t-1},\ \epsilon \sim \mathcal{N}_n(0_n, Q_\epsilon), \\ \beta_t &= \beta_{t-1} + \eta_{t-1},\ \eta \sim \mathcal{N}_n(0_n, Q_\eta) \end{aligned} \tag{30}$$

where $z_t \in \mathbb{R}^4$. Note the additive decomposition of the time series of measurements into trend and noise. Such a model is called a *local linear trend (LLT) model* [5]. Since we consider one, two, three and four trends, we have $n \in \{2,4,6,8\}$. We assume that the v_t is independent from the other noise terms appearing in (30). We do allow ϵ_t and η_t to be correlated. Both independence assumptions are made to be in line with the theory behind the Kalman filter (Section 2.2). By defining $x_t = \begin{pmatrix}\mu_t \\ \beta_t\end{pmatrix} \in \mathbb{R}^n$, we can derive the following state-space formulation of (30):

$$\begin{aligned} x_t &= F x_{t-1} + w_{t-1} \\ z_t &= H x_t + v_t \end{aligned}$$

where

$$F = \begin{pmatrix} I_{n/2} & I_{n/2} \\ 0_{(n/2)\times(n/2)} & I_{n/2} \end{pmatrix} \text{ and } H = \begin{pmatrix} H_\mu & 0_{(n/2)\times(n/2)} \end{pmatrix} \in \mathbb{R}^{p\times n}. \tag{31}$$

Furthermore, $w_t = \begin{pmatrix}\epsilon_t \\ \eta_t\end{pmatrix}$, so $w_t \sim \mathcal{N}_n(0_n, Q)$ where $Q \in \mathbb{R}^{n\times n}$.

For $n \neq 8$, H will have to be estimated, where initial estimates are stated below. This follows from the fact that in typical trend modelling, H maps the level of the trend (μ_t) to the stock price directly (where the difference between the stock price and the mapping is represented by the noise term v_t), see Equation 31. However, typically, each company's stock price is modelled its own separate trend (e.g. when $n = 8$, c.f. [5]). For $n \in \{2,4,6\}$, we aim to estimate the linear combinations of the $n/2$ trends to estimate each company's stock price.

The possible presence of seasonal- and cycle components in the stock data is ignored throughout the analysis for simplicity. One could also argue that those components are incorporated into the model implicitly through the noise term v_t.

Since we aim to develop a forecasting procedure for the stock prices, we partition the data into a training- and testing phase. The training data is defined for the stock prices z_t with $t \in \{1,\dots,T_{\text{train}}\}$, while the measurements z_t with $t \in \{T_{\text{train}}+1,\dots,T_{\text{test}}\}$ serves as test data. Note that $T_{\text{test}} = 901$, which is the total number of observations. The training set is used to estimate the unknown parameters and the test set is used to evaluate the model performance. Due to the well-known chaotic behaviour of stock prices, we take $T_{\text{train}} = 850$, which creates a larger proportion between train- and test data of 80%-20% (or 70%-30%). This allows us to make relatively short-term predictions, while we are provided with a larger sample of measurements to estimate the parameters. Since $T_{\text{train}} = 850$, we have the latter 51 observations as test data.

Training phase

The parameters are estimated by the EMKF algorithm. Note that F is known $\forall n \in \{2,4,6,8\}$, while H is only known whenever $n = 8$. Hence, we apply the methodology of Section 2.5 to the unknown parameters, which are denoted by the set $\theta = \{Q,H,R,\xi,\Lambda\}$ for $n \in \{2,4,6\}$. When $n = 8$, we have $\theta = \{Q,R,\xi,\Lambda\}$. Initial parameter estimates are derived as follows:

Q) Since we do not know the trend a priori, but we know that trends tend to behave rather stable, we take $\hat{Q}^{(0)} = \frac{1}{10} I_n$. That is, the trend should not be influenced greatly by the noise term w_t, meaning that the entries of Q should be rather small.

H As discussed, for $n \in \{2,4,6\}$, H has to be estimated. Due to the defined strucutre of the EMKF algorithm, we cannot estimate H as shaped in (31): H has to be estimated in its entire dimension. We discuss this below. We do initialize H as defined. That is, the right block of H in (31) is initialized as zeros. H_μ is initialized as a p-by-$n/2$ matrix with entries equal to $2/n$. In this way, the initial setting takes linear combinations of all levels (either one, two or three) with equal weights to estimate the stock prices (measurements).

R) Note that H maps the level of the trend to the stock price itself. Since the measurements are assumed to be noisy compared to the trend, we propose that R should represent the variance in the stock prices. To be explicit, we take $\hat{R}^{(0)}$ to be a diagonal matrix with the sample variance of each company's stock prices in the training phase on the diagonal in its respective row/column. Note that since the data is standardized, we automatically have $R_0 = I_4$.

ξ) Since the trend models averages the stock price trajectory, we take the initial level as the sample average of the first 100 measurements. For the one trend-case, we the only level μ_0 to be the average of all first 100 measurements. For the two- and three trend-cases, we set all initial levels equal to the average of all first 100 measurements. Finally, for the four trend-case, we take the first 100 measurements of the company corresponding to each trend, since we model trends company-specific here. The slopes β_0 are computed analogously and are defined as the slopes of the line segment between the smallest and the largest measurement, as a function of time.

Λ) It is generally difficult to assign an initial covariance matrix for the error [5]. One typically takes a diffuse prior (i.e. infinite variance, c.f. [5]), but this lies beyond the scope of this thesis. Instead, in line with the literature, we propose a rather large initial error covariance matrix i.e. $\Lambda = 100 I_n$.

Regarding the estimation of H, we can still map the state X_t to the estimated trend for each company by the mapping $H_\mu \mu_t$. To approximate the stock prices themselves, we use the mapping Hx_t, where x_t is the realization of X_t. Note that this is mathematically not completely in line with the model specification in (31), but it enables us to estimate the trend as linear combinations of the levels μ_t, which is part of our original goal.

To ensure that convergence to maximum likelihood estimates is achieved, EMKF is ran 100 times (note that this does not correspond to one run with 100 iterations as a maximum) with randomly perturbed initial parameters which are described above. For each set of estimates, we check whether convergence is achieved (i.e. whenever EMKF terminates before the maximum number of iterations is reached). From all of these estimates, we take the estimates corresponding to the largest log-likelihood value as our parameter estimates. Next, we identify the type of stationary point by the theory described in Section 2.6. Preferably, we observe that the set of estimates with the largest observed log-likelihood value of the measurements converged to a critical point and that it is in fact a local maximum of this log-likelihood function.

From Section 2.6, we have that for maximum likelihood estimates, the evaluated Hessian is an approximation of the Fisher information matrix, meaning that we can extract the covariance matrix of the asymptotic distribution of the estimator $\hat{\psi}$. Hence, we can derive confidence intervals for our maximum likelihood estimates. Note that this is only possible for parameter estimates which correspond to a (local) maximum of the log-likelihood of the measurements.

Whenever we have obtained parameter estimates $\hat{\theta}$, we run the Kalman- filter and smoother with those estimates in order to obtain (smoothed) hidden state estimates $\tilde{x}_t$ $(t \in \{0, 1, \ldots, T_{\text{train}}\}$. These estimates serve as the estimates for the trend and can be used to approximate the stock prices by the mapping $z_t - \hat{H}\tilde{x}_t$. We also verify steady-state behaviour by looking at the innovations $\hat{H}\hat{x}_t^-$, where the a priori trend estimates $\hat{x}_t^-$ result from the Kalman filter. We can also check whether the resulting model is observable by the Kalman rank test (Lemma 3), using $\hat{H}$.

Testing phase (Forecasting)

To forecast the stock prices, we introduce the *online Kalman filter*. The online Kalman filter propagates trend estimations throughout the testing phase, from the initial conditions $\tilde{x}_{T_{\text{train}}}$ and $\tilde{P}_{T_{\text{train}}}$, which are the smoothed estimates at the final observation in the training data, obtained from the training phase, together with the estimated parameters from the training phase. Note that for the initial condition of the online Kalman filter, one could also use the filtered estimates $\hat{x}_{T_{\text{train}}}$ and $P_{T_{\text{train}}}$, since they are equal to the smoothed estimates at $t = T_{\text{train}}$. Essentially, the online Kalman filter is just the Kalman filter applied to the test data with the described initial conditions.

Forecasts are extracted from $\hat{x}_t^-$, $t \in \{T_{\text{train}} + 1, \dots, T_{\text{test}}\}$, where $T_{\text{test}} = 901$. Since a priori trend estimates do not incorporate the current stock price, this method is actually valid as a forecasting procedure. By incorporating the actual stock price in the update step of the Kalman filter, the forecast remain on a similar trajectory as the stock prices themselves.

The trend forecast is computed similarly as the estimated trend during the training phase: $\hat{H}_\mu \hat{\mu}_t^-$, where $\hat{\mu}_t^-$ are the first $n/2$ entries of $\hat{x}_t^-$.

Accuracy measure

To measure the accuracy of the trend/stock price estimations/forecasts, we make use of the *mean absolute percentage error (MAPE)*. Denoting $\hat{y} \in \mathbb{R}^\tau$ as an estimate/forecast of the actual value $y \in \mathbb{R}^\tau$ for some $\tau \in \mathbb{N}$, the MAPE is given by

$$\text{MAPE}(\hat{y}) = \frac{100\%}{\tau} \sum_{i=1}^{\tau} \left| \frac{\hat{y}_i - y_i}{y_i} \right|$$

The MAPE is a relative accuracy measure: it compute the average percentage deviation from the actual values. We choose for a relative accuracy measure because the stock price data differs in scale among companies. In this way, we can compare the performance of the learned Kalman filter and the forecasts between companies more precise, compared to an absolute measure (e.g. the mean squared error).

3.5 Software Implementation

All methods (e.g. the Kalman filter/smoother and the EMKF algorithm) have been implemented in Python, where the source code is written by the author himself. The functions can be found in Appendix B. While developing the functions, results have been compared to those obtained from the `pykalman` library (pykalman.github.io), which shows identical results regarding the Kalman- filter and smoother.

The EMKF algorithm written by the author gives different results than the `em` method from the `pykalman` library, which follows from the fact that the `EMKF` function written by the author provides more flexibility regarding the convergence criteria. From now on, we refer to the written EMKF algorithm as the `EMKF` function. The additional flexibility is summarized as follows:

- While the `em` method only perform a predefined number of iterations, `EMKF` actually optimizes the parameter estimates in terms of convergence to stationary point of the log-likelihood of the measurements (the observed data in the Kalman filter setting). Additionally, the user sets a maximum number of iterations to perform to avoid potential overfitting and rather long computation times.
- Additionally to the log-likelihood increase user can choose to define convergence of the EMKF algorithm in terms of the difference in parameter estimates as well in `EMKF`. That is, if the parameter estimates deviate too less from the previous estimates, then convergence is achieved. In this way, one does not only has convergence of the log-likelihood of the measurements, but also stable parameter estimates.
- The user can manually set the thresholds regarding the log-likelihood increase and the difference in parameter estimates in `EMKF`. In this way, the user can choose his/her own degree of flexibility of the EMKF algorithm. Hence, if the user does not need the stable parameter estimate criterion, he/she could set the `tol_params` argument rather large (e.g. 100, but 1 should already suffice).
- `EMKF` provides warnings whenever one of the updated covariance matrices Q, R and/or Λ becomes non-positive definite, before forcing it to be positive definite (by adding the largest negative eigenvalue to the diagonal). This could indicate numerical underflow.

The standard values for some `EMKF` arguments are given in Table 5. The user can use their own preferences if preferred. However, by not providing other values, `EMKF` uses the values given below.

`EMKF` Argument	Description	Standard value
`max_it`	Maximum number of iterations	1000
`tol_likelihood`	Increase in log-likelihood threshold	0.01
`tol_params`	Difference in parameter estimates threshold	0.005
`em_vars`	Parameters to estimate	All parameters

Table 5: Standard arguments for the written EMKF function

Just as for the existing `em` method, one can specify the initial parameter estimates and set the parameters which need to be estimated (Section 2.5.5). Together with the above additions, `EMKF`is the most flexible EMKF algorithm which is applicable in Python. Namely, up to the best knowledge of the author, no other EMKF algorithm currently exists, except for the `em` method from the `pykalman` library. There exists variants in R, namely the `EM` function from the `astsa` library [16] and the `MARSS` function from the `MARSS` library. While the written EMKF algorithm competes with the `EM` function in terms of estimating H (which is not possible in the `EM` function) and parameter estimates' convergence (the `EM` function only uses convergence in terms of relative log-likelihood increase), the `EM` function gives the possiblity to include input terms u_t in the estimation procedure. A similar argument holds for the `em` method from the `pykalman` library, which uses the term 'offset' instead of 'input'. Regarding `MARSS`, an extensive comparison has not been made, but the flexiblity is rather similar.

We provide the computational complexity of `EMKF` for the first simulation in Section 4.1.1 (to be more precise, in Table 8) in terms of the time it takes to perform 50 iterations of EMKF for different hidden state- and measurement dimensions $n = p$ and sample sizes T.

4 Applications

As mentioned before, the Kalman- filter and smoother have a broad range of applications. Starting from e.g. random walks, we could move towards autonomous vehicles and visit several relevant applications before arriving at the core application within this thesis: stock prices. Before we do so, we perform a simulation study and reproduce an existing example from the literature.

4.1 Simulations

In this section, we consider simulated data to explore practical implementations of the Kalman filter/smoother and the EM algorithm (for the Kalman filter). We consider two separate cases: one where the state-space model in (9) consists fixed, 'simple' parameters, before moving on to a more 'complicated' one, shaped by randomly generated parameters.

4.1.1 Simulation 1

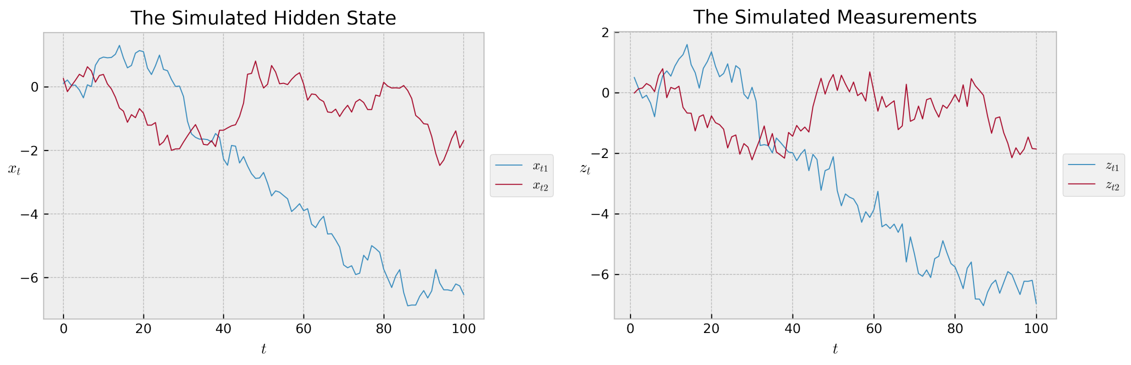
We start with the first simulation. Here, $n = p = 2$ and the state-space model is parameterized by

$$F = H = I_2, \quad Q = R = \Lambda = \tfrac{1}{10} I_2 \text{ and } \quad \xi = 0_2. \tag{32}$$

This means that the state-space model we are considering is formulated as

$$\begin{aligned} X_t &= X_{t-1} + w_{t-1},\ w_{t-1} \sim \mathcal{N}_2\left(0_2, \tfrac{1}{10} I_2\right),\ X_0 \sim \mathcal{N}_2\left(0_2, \tfrac{1}{10} I_2\right) \\ Z_t &= X_t + v_t,\ v_t \sim \mathcal{N}_2\left(0_2, \tfrac{1}{10} I_2\right) \end{aligned}$$

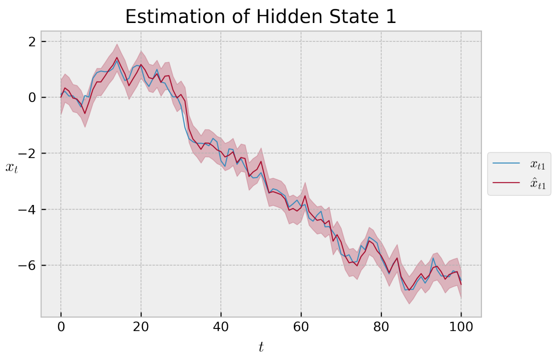
for $t \in \{1, \ldots, 100\}$. Note that this state-space model essentially simulates a random walk for both the hidden state X_t and the measurements Z_t. Sampling the initial hidden state from its distribution and propagating through the system described above in time (while sampling the noise terms from their respective distributions) with $T = 100$ generates the state- and measurement trajectory shown in Figure 6.



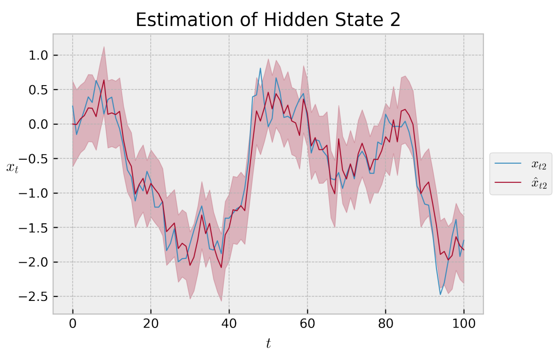
(a) (b)

Figure 6: The simulated true hidden state (Figure 6a) and the simulated measurements (Figure 6b) for the first simulation.

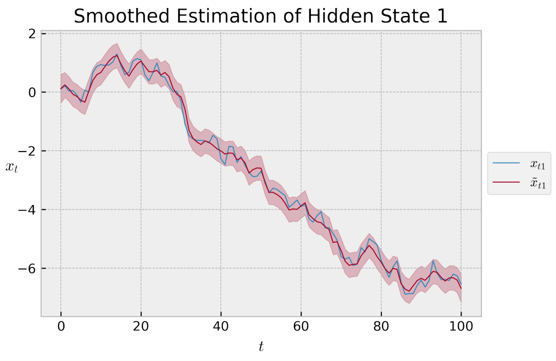
We acknowledge the hidden state trajectory in Figure 6a as the true hidden state. That is, while we cannot observe this process, we assume that if we could, this trajectory would be its realization. Note that the Kalman filter/smoother estimates this trajectory. Let us apply the Kalman filter/smoother to the generated measurements with $\hat{x}_0 = \xi$ and $P_0 = \Lambda$. The filtered- and smoothed estimates of the true hidden state are shown together with the actual values in Figure 7. We also include 95% confidence intervals shown by the confidence bands for the estimates, see (13) and (15).



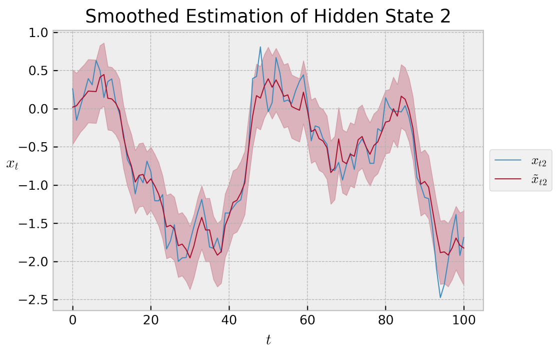
(a)



(b)



(c)



(d)

Figure 7: Estimation of the hidden state in Figure 6a. The confidence bands of the estimation are given by the shaded regions.

Overall, the estimates produced by the Kalman- filter ($\hat{x}_t$) and smoother ($\tilde{x}_t$) seem to be accurate. The true hidden state is contained in the confidence band for X_t for almost all times. Note that from Figures 7a–7b, the estimates produced by the Kalman smoother appear less oscilating than the filter estimates. Also, the confidence bands from the smoothed estimates is are smaller than from the estimate produced by the filter. The mean squared errors of the state estimates are given by

Estimate	MSE_1	MSE_2
Filter	0.058	0.067
Smoother	0.037	0.043

Table 6: Mean squared errors of the state estimates with the true parameters for the first simulation

Observe that the MSEs for the smoothed estimates is smaller than the filtered estimates. This is in line with what we have discussed in Section 2.3: the Kalman smoother improves the filtered estimates by conditioning the estimate on all measurement history.

Recall from Theorem 2 that the Kalman gain K_t and the state/error covariance matrix P_t converge to a steady value under certain assumptions. For $t = 100$, the Kalman gain and error covariance matrix are, rounded to three decimals given by

$$K_{100} = \begin{pmatrix} 0.618 & 0 \\ 0 & 0.618 \end{pmatrix}, \; P_{100} = \begin{pmatrix} 0.062 & 0 \\ 0 & 0.062 \end{pmatrix}.$$

The Kalman gain assigns for both components of the state estimates a weight of approximately 0.618 to the innovation term $z_t - H\hat{x}_t^-$. Also, the variances of the (smoothed) estimates produced by the Kalman- filter and smoother are rather small. Verifications show that these smoothed variances are already achieved at $t = 7$.

Let us apply the EM algorithm with the current parameters as initial estimates. That is, starting from the true setting, we are wondering if there is a better model in terms of the log-likelihood of the measurements (observed data). This 'better' model should then be a better explanation for the observed data. Applying the EMKF algorithm with initializations given by (32) with the default convergence criteria yields the following parameter estimates:

$$\hat{F} = \begin{pmatrix} 1.003 & 0.075 \\ 0.008 & 0.968 \end{pmatrix}, \quad \hat{Q} = \begin{pmatrix} 0.089 & -0.022 \\ -0.022 & 0.072 \end{pmatrix}, \quad \hat{H} = \begin{pmatrix} 1.003 & -0.025 \\ 0.002 & 0.969 \end{pmatrix}$$
$$\hat{R} = \begin{pmatrix} 0.071 & -0.020 \\ -0.020 & 0.070 \end{pmatrix}, \quad \hat{\xi} = \begin{pmatrix} 0.301 \\ 0.046 \end{pmatrix}, \quad \hat{\Lambda} = \begin{pmatrix} 0.009 & -0.003 \\ -0.003 & 0.008 \end{pmatrix}$$

The model parameters seem to have converged to similar values as the true parameters. EMKF has converged in 14 iterations. Note that some correlations between state- and measurements components have been introduced since Q and R have nonzero off-diagonal entries. The initial parameters seem to have converged to values which differ more from their true values. Note that the entries from $\hat{\Lambda}$ have become rather small, compared to their initial values.

The increase in log-likelihood of the measurements is shown in Figure 8.

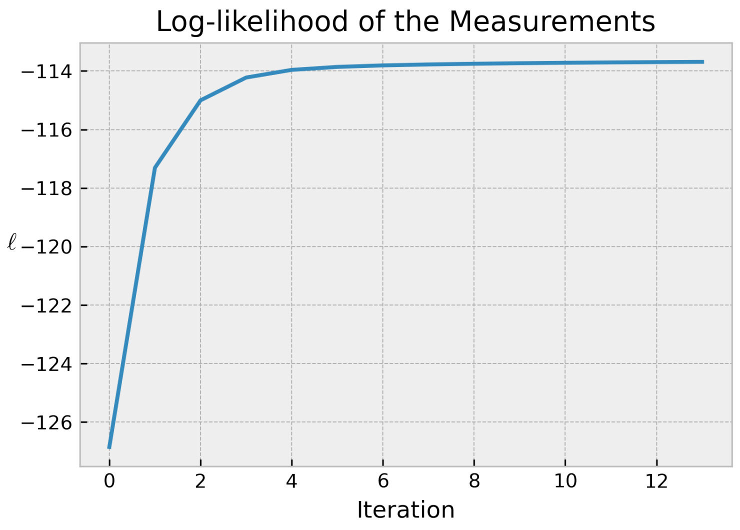


Figure 8: Log-likelihood of the EM algorithm for the first simulation

Figure 8 shows a steady increase in the log-likelihood of the observed data. When we apply the Kalman- filter and smoother to the measurements with the optimized parameters, it results in the MSEs in Table 7.

Estimate	MSE_1	MSE_2
Filter	0.056	0.066
Smoother	0.038	0.042

Table 7: Mean squared errors of the state estimates for the first simulation, resulting from the optimized parameters.

Comparing Tables 6 and 7 tells us that the MSEs before and after applying the EMKF algorithm are similar. Except for the smoothed estimate of hidden state component 1, the MSEs became smaller after optimizing the parameters. However, on average between components, the original parameters yields a slightly smaller MSE than the optimized ones for the smoothed estimates (0.03973 versus 0.03980 respectively). This indicates that the EMKF algorithm does not outperform the true parameter setting. However, note that we could only conclude this since we have fixed a true parameter setting, which generated the true hidden states. Also, note that the EMKF estimates are based on sample data and that they do not necessarily represent the population of measurements.

To conclude, EMKF shows a steady increase in the log-likelihood of the observed data (measurements) and the maximum likelihood estimates resulting from applying EMKF with the true parameters as our initializations produced hidden state estimates that are similarly accurate as the ones with the true parameters, in the MSE sense.

To give the reader an idea of the computational complexity of the EMKF algorithm, Table 8 contains the running time of the EMKF algorithm as before, but for different sample sizes T and state- and measurement dimension n and p, where $n = p$. The `timeit` function from the `timeit` library in Python executes 50 iterations of the EMKF algorithm 100 times for given T, n and p and computes the average running time. The program is executed on an Apple MacBook Pro 2020 with an Apple M1 CPU and 8 GB RAM.

	T		
n, p	100	1000	10000
2	1.715s	17.39s	335.17s
3	1.737s	17.92s	251.08s
4	1.771s	18.74s	309.09s
6	1.829s	20.75s	711.69s

Table 8: Computational complexity of 50 iterations of the EMKF algorithm in seconds for several instances.

As one can observe from Table 8, the computational complexity heavily depends on the sample size T, compared to the hidden state- and measurement dimension $n = p$. Also, the relevance of the hidden state dimension increases as the sample size increases. It appears that the EMKF algorithm is generally slower for $n = p = 2$ than larger hidden state- and measurement dimension for $T = 10000$. However, this deviation may follow from the fact that the computer has been used during this run, while the runs for $n \in \{3, 4\}$ have been done overnight.

4.1.2 Simulation 2

The aim of this simulation is to generate realizations of the (true) hidden state $x_t \in \mathbb{R}^n$ and measurements $z_t \in \mathbb{R}^p$ from a system with known parameters. Then, we apply the EM algorithm to the measurements with small perturbations on the the true parameters as initialization, aiming to (re-)estimate them.

In this study, we select $n = 3,\ p = 2$ and generate random parameters up to their constrains. The reader is referred to Appendix B to see how the parameters are generated. For the covariance matrices, we generate random SPD matrices, utilizing the `make_spd_matrix` function from the `scikit-learn` package. The parameters are, rounded to three decimals, given by

$$F = \begin{pmatrix} 0.901 & 0.045 & -0.036 \\ 0.045 & 0.881 & -0.008 \\ 0.033 & -0.009 & 0.905 \end{pmatrix},\ Q = \begin{pmatrix} 8.025\cdot 10^{-4} & -5.330\cdot 10^{-4} & -4.853\cdot 10^{-4} \\ -5.330\cdot 10^{-4} & 1.912\cdot 10^{-3} & 1.443\cdot 10^{-3} \\ -4.853\cdot 10^{-4} & 1.443\cdot 10^{-3} & 1.929\cdot 10^{-3} \end{pmatrix},\ H = \begin{pmatrix} -0.945 & 0.507 & 0.076 \\ -0.341 & 0.577 & -0.394 \end{pmatrix}$$

$$R = \begin{pmatrix} 1.913\cdot 10^{-3} & 6.096\cdot 10^{-4} \\ 6.096\cdot 10^{-4} & 3.264\cdot 10^{-4} \end{pmatrix},\ \xi = \begin{pmatrix} -0.019 \\ -0.146 \\ -0.039 \end{pmatrix} \text{ and } \Lambda = \begin{pmatrix} 0.008 & -0.005 & -0.005 \\ -0.005 & 0.019 & 0.014 \\ -0.005 & 0.014 & 0.019 \end{pmatrix}.$$

Sampling the initial hidden state $x_0 \sim \mathcal{N}_3(\xi, \Lambda)$ and propagating through the system described by the above parameters in time (the noise terms are sampled from their respective distributions) with $T = 1000$ generates the state- and measurement trajectory shown in Figure 9.

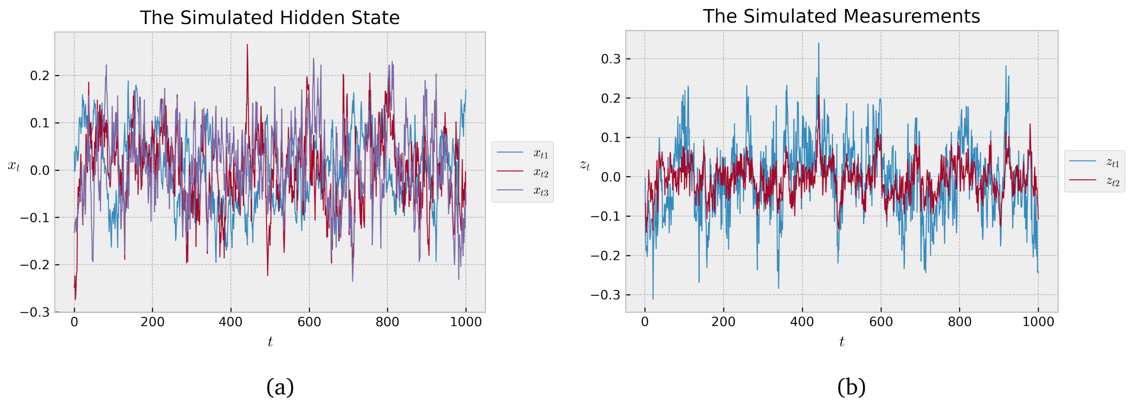


Figure 9: The simulated true hidden state (Figure 9a) and the simulated data (Figure 9b).

Observe from Figure 9 that both the true state and measurements are noisy. Let us apply the Kalman- and smoother to the data and plot the smoothed estimates together with the true value, component wise. This results in Figure 10.

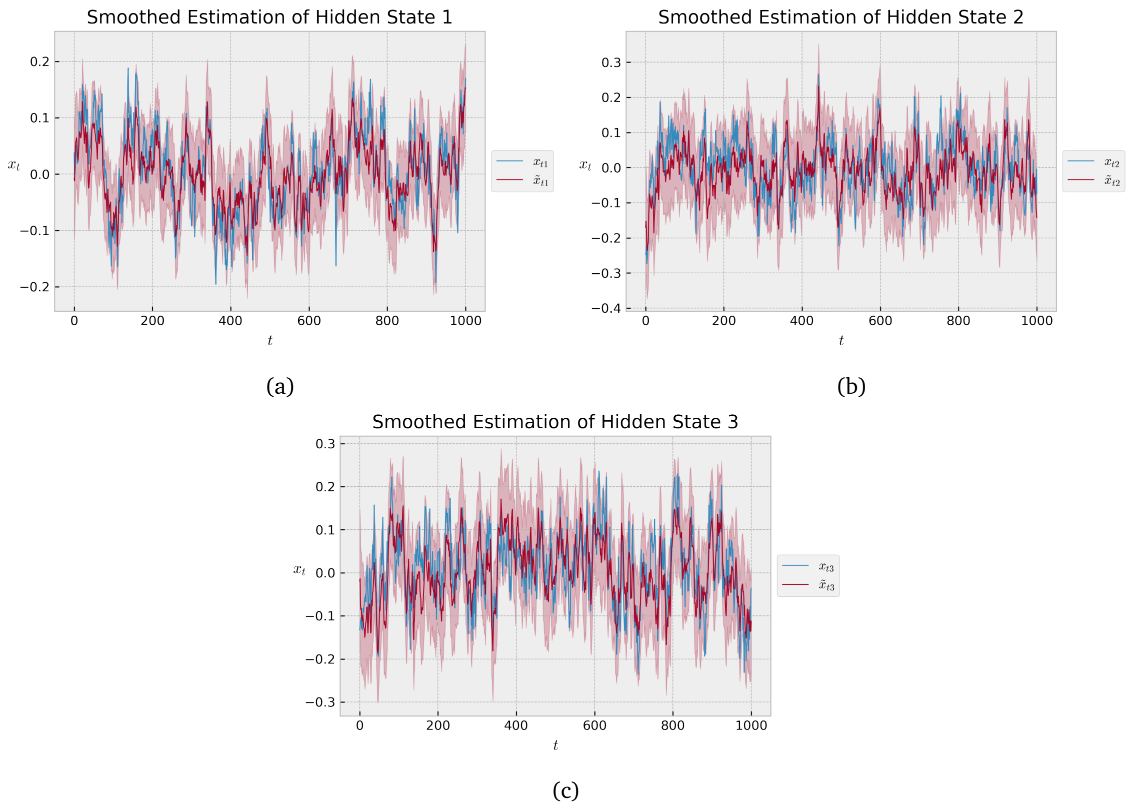


Figure 10: The smoothed estimates with 95% confidence bands for the true hidden state, based on the true parameter values.

Note that overall, the estimates of x_t ($t \in \{0, \dots, 1000\}$) seem to be precise, although it shows some (small) deviations. However, the true hidden state is almost always contained in the confidence intervals, generated by the distribution of $X_t|\mathbf{Z}_T$ (the smoothed estimates).

The mean squared error between the smoothed estimate and the true sequence of the hidden state are component-wise given by

MSE_1	MSE_2	MSE_3
0.0013	0.0034	0.0032

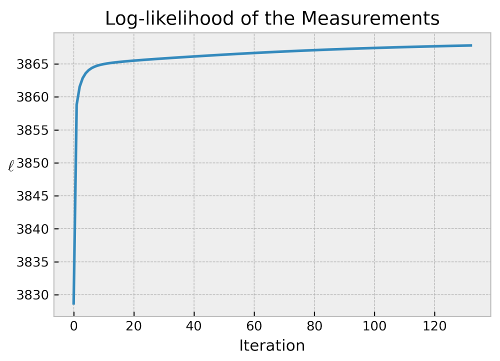
Table 9: Mean squared errors for the smoothed estimates of the true hidden states, given the true parameters.

We are now interested what kind of impact estimations of the parameters by the EMKF algorithm have on the component-wise mean squared errors given above. To this end, we slightly perturb the true parameters and use those perturbations as initial parameter set for the EM algorithm. The algorithm converged according to the standard convergence criteria in Section 3.5 after 133 iterations. The output is given by

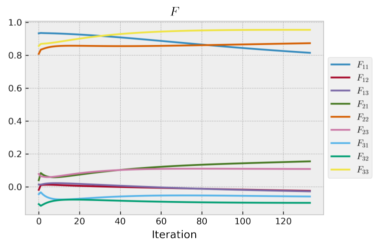
$$\hat{F} = \begin{pmatrix} 0.815 & -0.024 & -0.028 \\ 0.155 & 0.873 & 0.108 \\ -0.059 & -0.097 & 0.954 \end{pmatrix}, \hat{Q} = \begin{pmatrix} 0.001 & -3 \cdot 10^{-4} & -4 \cdot 10^{-4} \\ -3 \cdot 10^{-4} & 0.001 & 0.001 \\ -4 \cdot 10^{-4} & 0.001 & 0.002 \end{pmatrix},$$

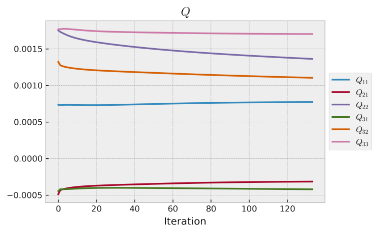
$$\hat{H} = \begin{pmatrix} -1.058 & 0.487 & 0.010 \\ -0.318 & 0.503 & -0.307 \end{pmatrix}, \hat{R} = \begin{pmatrix} 0.002 & 0.001 \\ 0.001 & 4 \cdot 10^{-4} \end{pmatrix}, \hat{\xi} = \begin{pmatrix} -0.029 \\ -0.207 \\ -0.059 \end{pmatrix} \text{ and } \hat{\Lambda} = \begin{pmatrix} 3 \cdot 10^{-5} & 1 \cdot 10^{-5} & 3 \cdot 10^{-5} \\ 1 \cdot 10^{-5} & 5 \cdot 10^{-4} & 7 \cdot 10^{-4} \\ 3 \cdot 10^{-5} & 7 \cdot 10^{-4} & 1 \cdot 10^{-4} \end{pmatrix}.$$

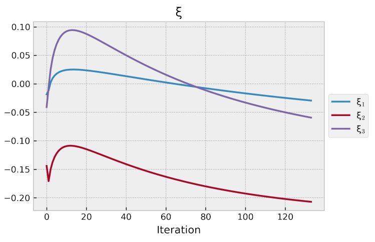
Figure 11 illustrates how the estimates converge to this output:



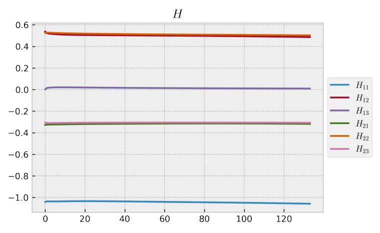
(a)

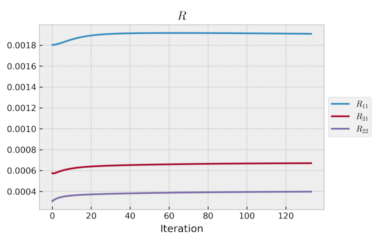


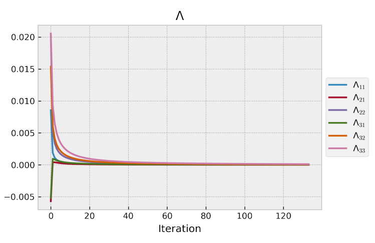




(b) (c) (d)







(e) (f) (g)

Figure 11: Likelihood convergence of the EMKF algorithm for the simulated data (Figure 11a) and entry-wise convergence of the parameters (Figures 11b–11g).

Figure 11 shows a steady increase in the log-likelihood of the measurements and convergence of the parameters. However, for ξ (recall that $\xi = \mathbb{E}(X_0)$), the values tend to diverge first, before converging to a steady value. The differences between the maximum likelihood estimates and the true values, together with their Frobenius norms (for the vector ξ, we take the 2-norm) are given by

$$\Delta F = \hat{F} - F = \begin{pmatrix} -0.087 & -0.070 & 0.008 \\ 0.110 & -0.008 & 0.116 \\ -0.092 & -0.088 & 0.049 \end{pmatrix}, \qquad ||\Delta F||_F = 0.238,$$

$$\Delta Q = \hat{Q} - Q = \begin{pmatrix} -3\cdot 10^{-05} & 2\cdot 10^{-04} & 6\cdot 10^{-05} \\ 2\cdot 10^{-04} & -5\cdot 10^{-04} & -3\cdot 10^{-04} \\ 6\cdot 10^{-05} & -3\cdot 10^{-04} & -2\cdot 10^{-04} \end{pmatrix}, \ ||\Delta Q||_F = 0.001,$$

$$\Delta H = \hat{H} - H = \begin{pmatrix} -0.113 & -0.020 & -0.067 \\ 0.023 & -0.074 & 0.087 \end{pmatrix}, \qquad ||\Delta H||_F = 0.176,$$

$$\Delta R = \hat{R} - R = \begin{pmatrix} -3\cdot 10^{-06} & 6\cdot 10^{-05} \\ 6\cdot 10^{-05} & 7\cdot 10^{-05} \end{pmatrix}, \qquad ||\Delta R||_F = 1\cdot 10^{-4},$$

$$\Delta \xi = \hat{\xi} - \xi = \begin{pmatrix} -0.011 \\ -0.061 \\ -0.021 \end{pmatrix}, \qquad ||\Delta \xi||_2 = 0.065,$$

$$\Delta \Lambda = \hat{\Lambda} - \Lambda = \begin{pmatrix} -0.008 & 0.005 & 0.005 \\ 0.005 & -0.019 & -0.014 \\ 0.005 & -0.014 & -0.019 \end{pmatrix}, \qquad ||\Delta \Lambda||_F = 0.036.$$

Especially Q and R are estimated quite accurately. In F and H, the entry-wise differences seem to be somewhat larger. However, the Frobenius norms (and the 2-norm for $\Delta\xi$) of all parameter differences indicate accurate re-estimation of the parameters.

4.2 Example

As a second example of the EMKF algorithm, we consider Example 6.8 from [16], which is a one-dimensional setting (one-dimensional- state and measurements). The source code of this example from the authors themselves can be found at github.com. The dynamics are given by the following state-space model, which describes an AR(1) process:

$$\begin{aligned} X_t &= 0.8X_{t-1} + w_t,\ w_t \sim \mathcal{N}(0,1),\ X_0 \sim \mathcal{N}(0, 0.122) \\ Z_t &= X_t + v_t,\ v_t \sim \mathcal{N}(0,1) \end{aligned},$$

with parameter set $\theta = \{f, q^2, r^2, \xi, \lambda^2\}$ (we consider lower case parameters since the setting is one-dimensional, which makes the covariance matrices variances, hence the squares). Note that $h = 1$ is fixed in this example. The measurements are generated according to this state-space model (as similar to the simulations in Section 4.1) and are copied from the authors' source code. The initializations are, rounded to three decimals, given by

$$\hat{f}^{(0)} = 0.837,\ \hat{q}^{2(0)} = 0.634,\ \hat{r}^{2(0)} = 1.235,\ \hat{\xi}^{(0} = 0 \text{ and } \hat{\lambda}^{2(0)} = 2.8.$$

Running the function `EM` from the `astsa` library in R (EMKF written by the authors of [16]) gives convergence in 18 iterations. Let us compare the output of `EM` to the output of `EMKF` in Table 10.

Parameter	EM Estimate	EMKF Estimate
f	0.796	0.795
q^2	0.914	0.922
r^2	0.989	0.980
ξ	1.256	1.262
λ^2	0.128	0.122

Table 10: Comparison of EMKF estimates from the literature and the source code fromt this thesis

The parameter estimates from Table 10 look rather similar. Note that ξ is badly estimated. We also have encountered this in Section 4.1.2, which warns us that estimation of ξ is something to be aware about: it does not necessarily converge to the true value. Fortunately, this does actually hold for the other parameters in this case.

It is also worth mentioning that the EMKF algorithm has not yet converged according to the standard criteria in 18 iterations. It takes 25 iterations for EMKF to converge, which follows from the fact that EM only keeps track of the log-likelihood of the measurements (and hence, not of the convergence of the parameters).

4.3 Forecasting Stock Prices

We now move on to the real-world application of this thesis, in terms of stock price predictions. We apply the methodology from Section 3.4 to the data in Section 3.1. First, we explore the data in a descriptive sense in Section 4.3.1, before entering the modelling and estimation phase in Section 4.3.2. Finally, we show the forecast in Section 4.3.3.

4.3.1 Exploratory Analysis

Let us first explore the data that we have obtained, which is described in Section 3.1 and visualized in Figure 5. The standardized data is shown in Figure 12.

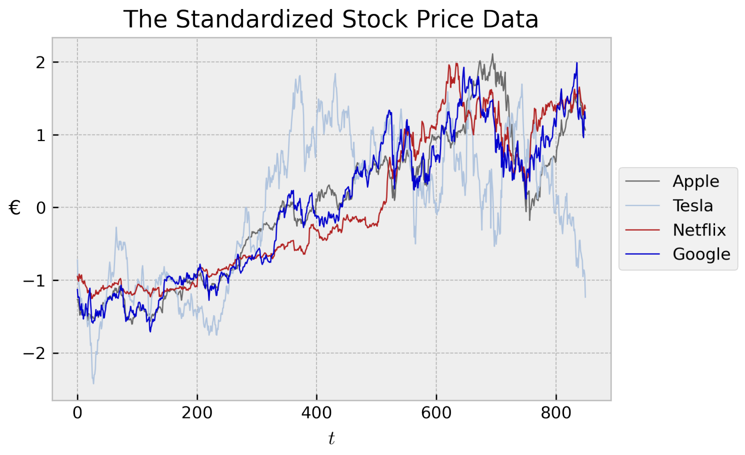


Figure 12: The standardized data which is used throughout the analysis.

From Figures 5 and 17, all four stock prices show an increasing trend, which would mean that the time series are non-stationary. To confirm this conjecture, we apply the *Augmented Dickey-Fuller (ADF) test* to each individual series with the usual significance level $\alpha = 0.05$. The ADF test assumes that the data is non-stationary under the null hypothesis. The alternative hypothesis depends on the situation, but here, it states stationarity. We could also have tested for trend-stationarity, meaning that the detrented[15] series is stationary, but since we do not know

[15]A detrended series is a time series with the trend component is removed.

anything about possible seasonality or cyclic behaviour, we aim the ADF test to be rather general. The results of the ADF test are provided in Table 11.

Company	Statistic	Used lag	p-value
Apple	-2.237	8	0.469
Tesla	-1.963	0	0.621
Netflix	-2.534	13	0.311
Google	-2.524	11	0.316

Table 11: ADF test results.

The used lag in the ADF test determines how many auto-regressive terms are incorporated during the testing procedure, which is actually an important factor for the results. Fortunately, `adfuller` from the `statsmodels` library automatically computes the lag with minimal AIC of the resulting autoregressive model.

Since all reported p-values are larger than the significance level 0.05, we conclude that the stock price series are indeed non-stationary. Non-stationarity should not arise any problems during the process of Kalman filtering, since measurements are not required to be stationary. Also, non-stationarity does imply that the trend of the measurements is non-constant, which justifies the modelling approach where we assume a possibly different level and slope of the trend at each trading day.

Next, we are interested what insights we could obtain from the detrended series. A popular method for detrending a time series is to take the *first differences*, which results in Figure 13.

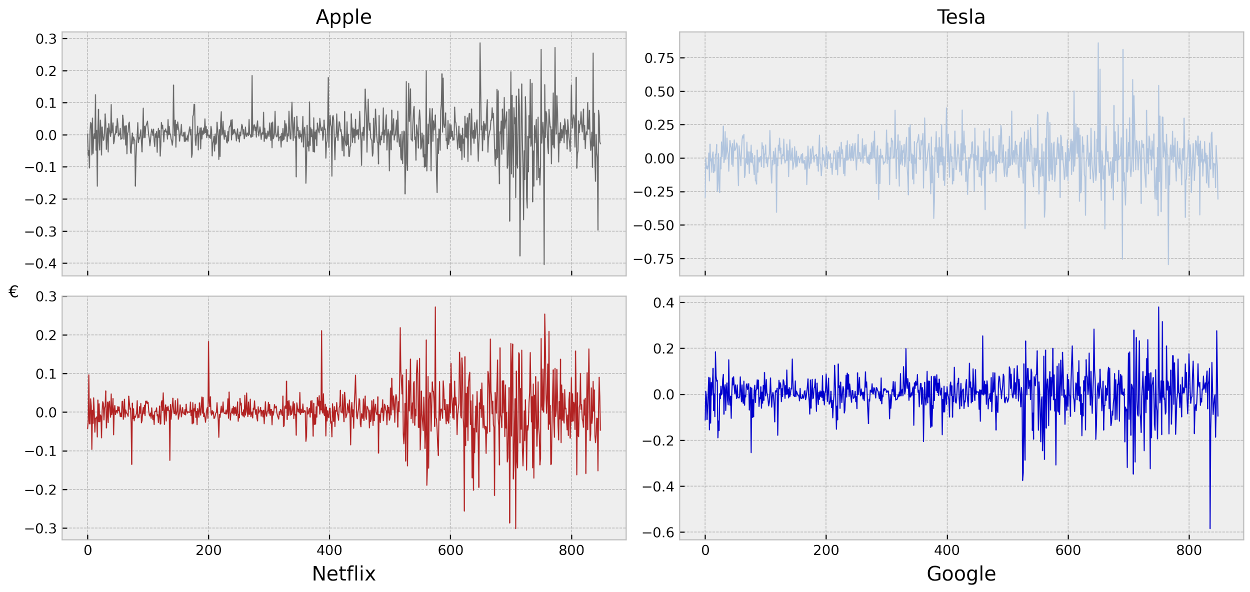


Figure 13: First differences of each company's time series

The series of first differences tend to be more stationary than the original series: the first differences seem to be centered around a constant, zero mean. Although it shows white noise behaviour, we also observe that the magnitude of the series increases over time (generally from $t = 500$), indicating that the variance of the process increases over time, meaning that the stock prices start to oscillate more throughout the training period. Hence, stationarity would be a strong conclusion to make. However, up to magnitude, the four time series of stock prices show similar behaviour in terms of first differences.

Recall that we have formulated our model as a additive time series decomposition in a trend and random component. Hence, we ignore the possible presence of seasonality and/or cyclic behaviour. We are interested if this modelling assumption is valid by checking whether seasonality is present. Seasonality behaviour means repeated behaviour of time series within a constant time interval. For example, if we consider a time series which consists of the amount of rain for each day for several years, we expect seasonal behaviour to occur (e.g. less rainfall during the summer). For stock prices, this seasonality is generally hard to assign to a fixed interval (e.g. each day or week), so we consider differencing with different lags: weekly, monthly and annually. When seasonality is present, we would expect the resulting differenced series to be centered around zero.

We fist consider the weekly differences. On average, there are five working days a week, meaning that the stock market is open, and hence, generates data, five days a week. Hence, we take the differences of the time series with lag 5. Note that this procedure is not completely precise due to holidays, but it should give a global overview of the weekly behaviour of the stock prices, which is given in Figure 14.

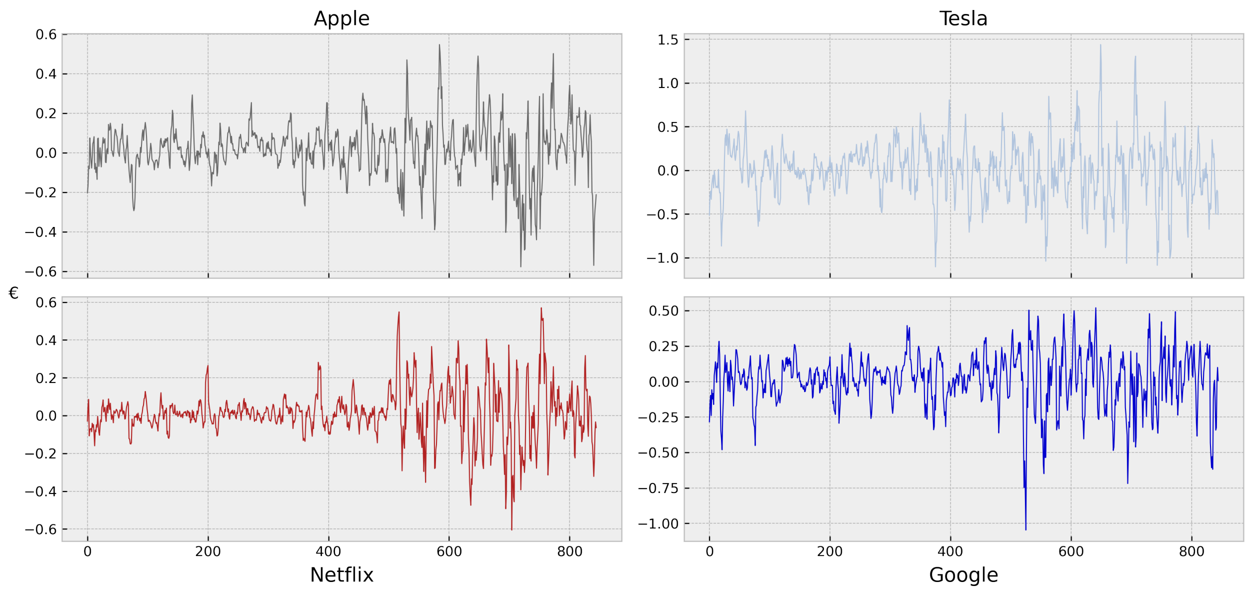


Figure 14: Weekly differences of each company's time series

If we compare this series with the first differences obtained in Figure 13, we still observe the zero mean behaviour for the weekly differences, but the variance seems to be larger. The increase in variance is less obvious as well. This would suggest the presence of a seasonal component, especially for the first 500 observations. For Tesla, the presence of a seasonal component seems less obvious.

Next, we consider the monthly differences. We take a lag of 20 trading days, which is the average training days within a month. Note that due to the fact that the number of days differ between months and, as similar to the weekly differences, holidays, this procedure is not complete precise as well. The results are shown in Figure 15.

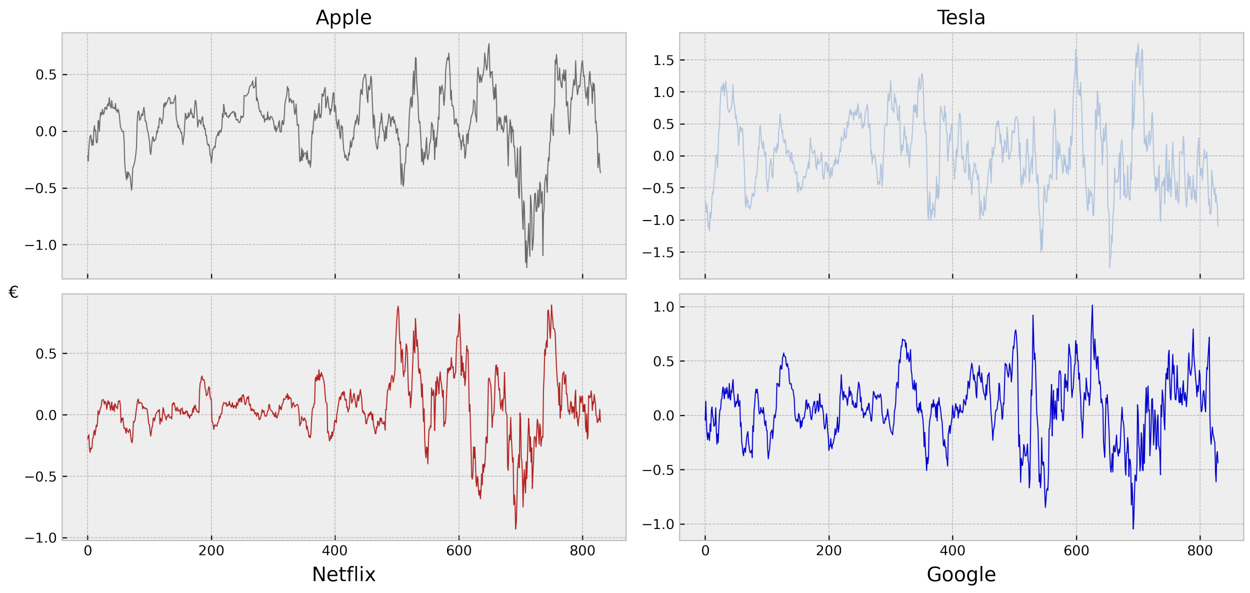


Figure 15: Monthly differences of each company's time series

Here, we still see the zero-mean behaviour as described before, but less compared to the weekly differences. This suggests that there is no monthly seasonal component, but not totally absent. However, this is up for discussion.

Finally, we consider the annually differences. There are 260 working days a year, so we take differences with a lag of 260. Since, additionally, there are leap years, the same argument regarding the preciseness holds, but note that 2016 is the only leap year contained in the dataset. The annually differences are given by Figure 16.

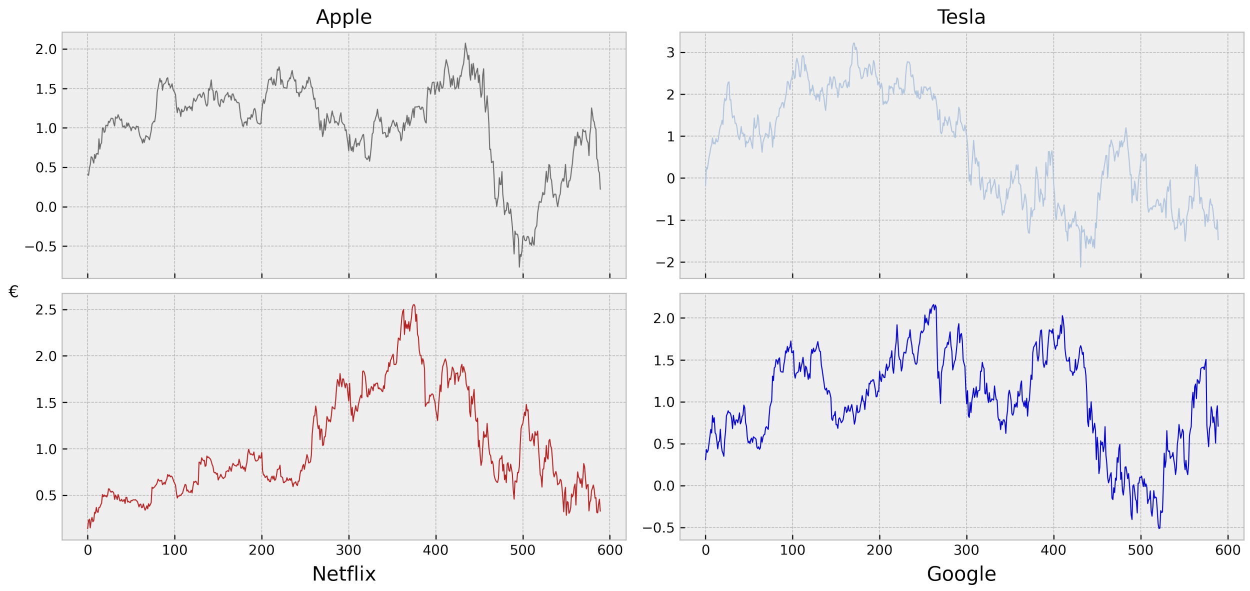


Figure 16: Annual differences of each company's time series

We now observe no centered behaviour around zero anymore. This suggest that we have no annual seasonal component in the time series of stock prices.

Overall, it seems that only on a weekly basis, we have conjectures about a seasonal component. However, it does not seem that a seasonal component must be included in the LLT model, meaning that we proceed with what we have.

4.3.2 Modelling, Estimation & Results

Now that we have obtained some more insight in the stock price data, we start the modelling and estimation phase. Recall that we model the stock prices by the LLT model where we investigate how many trends up to four we would need to optimize model performance. Hence, we consider four cases and show the obtained results. Recall that F is given in all cases by (31). Also, recall that are our program does not allow us to estimate $H_\mu \in \mathbb{R}^{(n/2)\times p}$ separately, so we have to estimate $H \in \mathbb{R}^{p\times n}$.

Single Trend

We start with the case of a single trend (i.e. $n = 2$). Hence $F = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ according to (31). We apply the methodology described in Section 3.4 and obtain the following estimates:

$$\hat{Q} = \begin{pmatrix} 9.665 \cdot 10^{-4} & -9.988 \cdot 10^{-7} \\ -9.988 \cdot 10^{-7} & 1.714 \cdot 10^{-6} \end{pmatrix}, \; \hat{H} = \begin{pmatrix} 1.741 & 20.362 \\ 1.350 & 65.739 \\ 1.535 & -16.196 \\ 1.675 & 13.444 \end{pmatrix}, \tag{33}$$

$$\hat{R} = \begin{pmatrix} 0.009 & -0.043 & -0.013 & -0.009 \\ -0.043 & 0.221 & 0.064 & 0.022 \\ -0.013 & 0.064 & 0.021 & 0.019 \\ -0.009 & 0.022 & 0.019 & 0.064 \end{pmatrix}, \; \hat{\xi} = \begin{pmatrix} -0.690 \\ -0.000 \end{pmatrix} \text{ and } \hat{\Lambda} = \begin{pmatrix} 2.071 \cdot 10^{-7} & -8.351 \cdot 10^{-10} \\ -8.351 \cdot 10^{-10} & 5.403 \cdot 10^{-10} \end{pmatrix}.$$

Note that the covariance matrix of the process noise Q is estimated to rather small. This makes sense, because we expect the trend to move rather deterministic, meaning that the noise term w_t should not be influential. Also, the initial condition for the covariance matrix Λ is estimated even smaller, which is similar to the example in Section 4.2.

The observability matrix of the resulting state-space model with $\hat{H}$ is verified to have rank 2, which is full-column rank. Therefore, the system is observable. However, the Hessian approximation yields a Hessian which is not negative definite when evaluated at the parameter estimates. Its eigenvalues are indicated by $\sigma\left(\hat{H}_s\right) \subset (-761.00, 501.21)$, meaning that these estimates corresponds to a saddle point of the log-likelihood of the measurements. Hence, we have not obtained maximum likelihood estimates. This follows from the fact that the Hessian has both positive and negative eigenvalues. Furthermore, Hessian evaluations for all other estimates obtained from different initializations for the EMKF algorithm did not gave us a negative definite Hessian. Hence, we did not find a local maximum at all, meaning that we cannot consider the asymptotic theory and confidence intervals for the estimates (in terms of the Fisher information) cannot be obtained.

Let us proceed with the estimates that we have. Although they are not maximum likelihood estimates, they do give the largest log-likelihood value found. Applying the Kalman- filter and smoother with the parameter estimates in (33) gives the trend- and stock price approximation in Figure 17. Here, we have rescaled all data back to to the original scale.

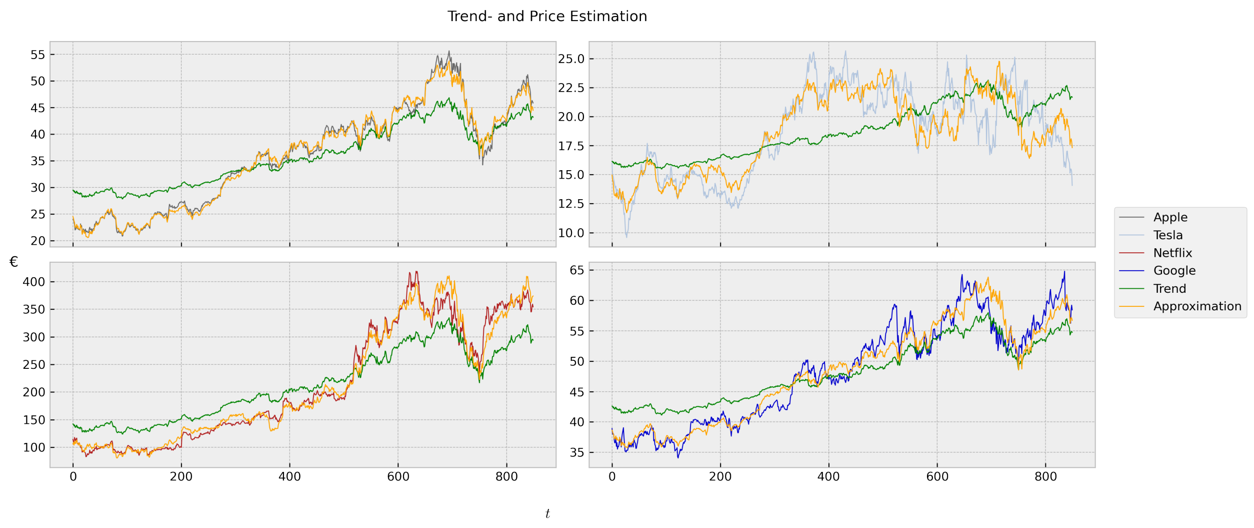


Figure 17: Trend- and price approximation for one trend. 95% confidence intervals for the trend are incorporated in the figure, but are too small to be visible.

It seems that the trend describes the average behaviour of all four companies. To quantify the difference between the predicted price (from the trend) to the actual stock price, we provide the mean absolute percentage errors for the training data in Table 12.

Company	MAPE
Apple	1.827
Tesla	7.639
Netflix	5.439
Google	3.216

Table 12: Mean absolute percentage training errors for the one-trend case.

It seems that the approximations for Tesla are the least accurate.

We now consider the steady-state behaviour of the estimated model. To this end, we compute the innovation sequence and check whether the a posteriori covariance matrix P and the Kalman gain K have stabilized. We obtain the following innovation sequence.

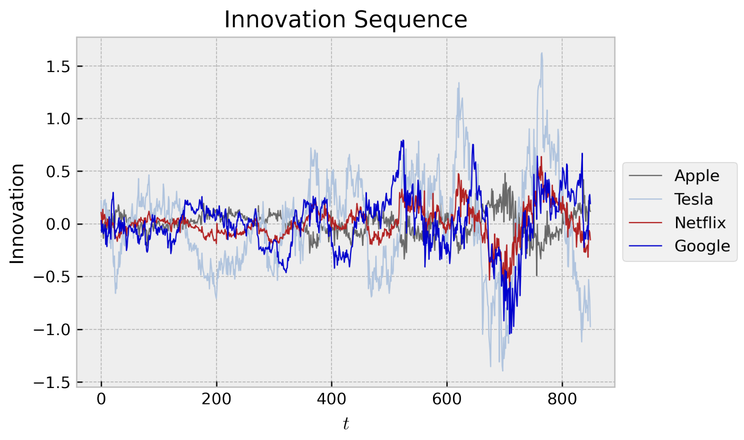


Figure 18: The innovation sequence for the one-trend case.

We also obtain

$$P_t \to \begin{pmatrix} 1.600 \cdot 10^{-5} & 1.109 \cdot 10^{-8} \\ 1.109 \cdot 10^{-8} & 1.386 \cdot 10^{-8} \end{pmatrix},\ K_t \to \begin{pmatrix} 3.407 \cdot 10^{-1} & -2.031 \cdot 10^{-2} & 3.146 \cdot 10^{-1} & -3.888 \cdot 10^{-2} \\ 7.803 \cdot 10^{-3} & 6.813 \cdot 10^{-3} & -1.993 \cdot 10^{-2} & 4.646 \cdot 10^{-3} \end{pmatrix} \text{ as } t \to T_{\text{train}}.$$

Although the a posteriori error covariance matrix P_t and the Kalman gain K_t converge to steady values, Figure 18 does not show the steady-state behaviour as described in Theorem 2. However, they remain bounded throughout the training phase.

Two trends

We now move on to the case where there are two trends (i.e. $n = 4$). Here, we have $F = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, according to (31). With the initializations as in Section 3.4, the estimates of the other parameters are given by

$$\hat{Q} = \begin{pmatrix} 3.726 \cdot 10^{-2} & -3.338 \cdot 10^{-2} & -1.795 \cdot 10^{-4} & -3.163 \cdot 10^{-4} \\ -3.338 \cdot 10^{-2} & 4.186 \cdot 10^{-2} & -1.968 \cdot 10^{-5} & 1.480 \cdot 10^{-4} \\ -1.795 \cdot 10^{-4} & -1.968 \cdot 10^{-5} & 6.175 \cdot 10^{-5} & -6.872 \cdot 10^{-5} \\ -3.163 \cdot 10^{-4} & 1.480 \cdot 10^{-4} & -6.872 \cdot 10^{-5} & 1.269 \cdot 10^{-4} \end{pmatrix}, \tag{34}$$

$$\hat{H} = \begin{pmatrix} 0.736 & 0.542 & 8.435 & 5.542 \\ 0.571 & 0.372 & -8.143 & 5.991 \\ 0.372 & 0.511 & 3.293 & 3.389 \\ 0.647 & 0.549 & 16.673 & 13.976 \end{pmatrix},$$

$$\hat{R} = \begin{pmatrix} 1.583 \cdot 10^{-4} & -9.091 \cdot 10^{-5} & 9.511 \cdot 10^{-5} & 2.057 \cdot 10^{-4} \\ -9.091 \cdot 10^{-5} & 9.560 \cdot 10^{-4} & 7.191 \cdot 10^{-5} & 9.239 \cdot 10^{-7} \\ 9.511 \cdot 10^{-5} & 7.191 \cdot 10^{-5} & 8.735 \cdot 10^{-5} & 1.385 \cdot 10^{-4} \\ 2.057 \cdot 10^{-4} & 9.239 \cdot 10^{-7} & 1.385 \cdot 10^{-4} & 2.911 \cdot 10^{-4} \end{pmatrix},$$

$$\hat{\xi} = \begin{pmatrix} -0.709 \\ -1.412 \\ -0.014 \\ 0.025 \end{pmatrix} \text{ and } \hat{\Lambda} = \begin{pmatrix} 1.203 \cdot 10^{-5} & -1.102 \cdot 10^{-5} & -9.183 \cdot 10^{-8} & -6.962 \cdot 10^{-8} \\ -1.102 \cdot 10^{-5} & 1.371 \cdot 10^{-5} & 1.894 \cdot 10^{-9} & 5.775 \cdot 10^{-8} \\ -9.183 \cdot 10^{-8} & 1.894 \cdot 10^{-9} & 3.807 \cdot 10^{-8} & -4.040 \cdot 10^{-8} \\ -6.962 \cdot 10^{-8} & 5.775 \cdot 10^{-8} & -4.040 \cdot 10^{-8} & 6.580 \cdot 10^{-8} \end{pmatrix}.$$

We observe the same as for the one trend-case, but additionally that the entries for R are smaller now. This suggests that two trends explain the measurements better than one, since the covariance matrix of the measurement noise is now smaller in magnitude (entry-wise).

We again observe an observability matrix of full rank (rank 4), using $\hat{H}$. Therefore, again, the system is observable. However, again, the estimated parameters do not correspond to maximum likelihood estimates, since the approximated Hessian is not negative definite. Its eigenvalues are now indicated by $\sigma\left(\hat{H}_s\right) \subset (-3019.98, 551.71)$. Since $\hat{H}_s$ has both positive and negative eigenvalues, the estimated parameters correspond, again, to a saddle point of the likelihood function of the measurements. Hessian computations for the other converged parameter estimates also did not yield a local maximum of the likelihood function. Hence, since all estimates correspond to a saddle point, we cannot consider asymptotic theory nor derive confidence intervals for the estimates in (34), as for the one trend-case.

We proceed with the estimates that we have. Again, applying the Kalman- filter and smoother with the parameter estimates (34) gives the trend- and stock price approximations in Figure 19. Just as for the one trend-case, we have rescaled everything back to its original scale. Also, as explained in Section 3.4, to compute the trend for each stock price, we pre-multiply μ_t with $\hat{H}_\mu$, the first two columns of $\hat{H}$ for all $t \in \{1, \ldots, T_{\text{train}}\}$.

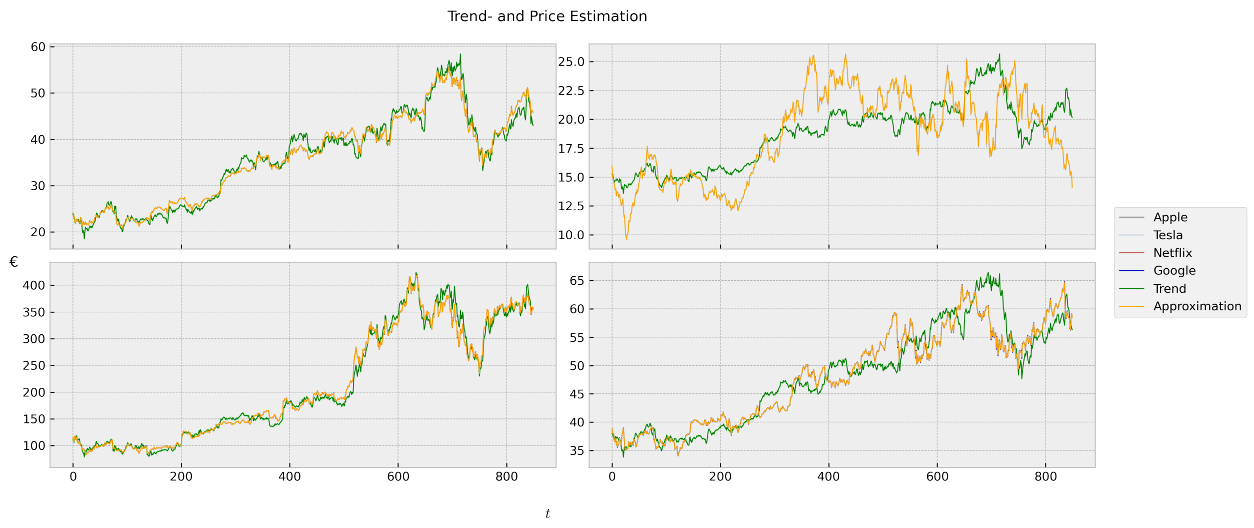


Figure 19: Trend- and price approximation for two trends. 95% confidence intervals for the trend are incorporated in the figure, but are too small to be visible.

The trend line does not show as averaged behaviour as for the one trend-case. This might follow from the fact that two trends provides more flexibility in estimation, meaning that Kalman filter forces the trend to be closer to the measurements. This follows from the fact that EMKF has estimated a measurement noise covariance matrix, R, of small magnitude. The mean absolute percentage errors for the training data are given in Table 13.

Company	MAPE
Apple	0.081
Tesla	0.147
Netflix	0.089
Google	0.065

Table 13: Mean absolute percentage training errors for two trends.

Note that the MAPEs are smaller than for the one-trend case. Also, the approximating for Tesla's stock prices is again the least accurate in terms of the MAPE.

We now move on to the steady-state behaviour of the estimated model. To this end, we compute the innovation sequence and check whether the a posteriori covariance matrix P and the Kalman gain K have stabilized. We obtain the following innovation sequence.

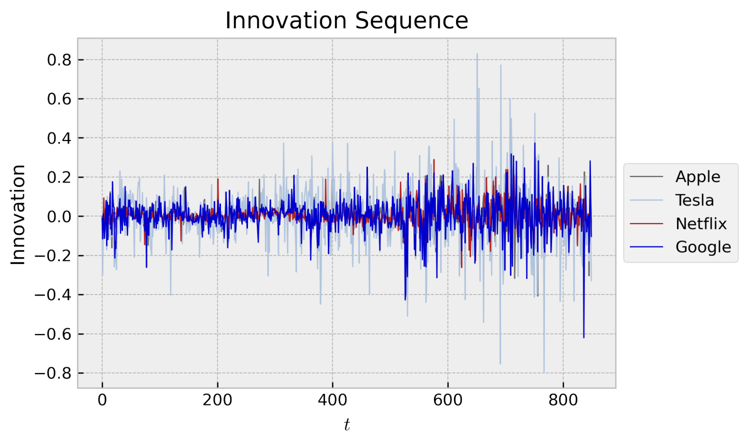


Figure 20: The innovation sequence for the two trend-case.

We also have

$$\left.\begin{aligned} P_t &\to \begin{pmatrix} 9.512\cdot 10^{-5} & -1.229\cdot 10^{-4} & 1.272\cdot 10^{-6} & 4.753\cdot 10^{-7} \\ -1.229\cdot 10^{-4} & 3.545\cdot 10^{-4} & 1.132\cdot 10^{-6} & 3.058\cdot 10^{-6} \\ 1.272\cdot 10^{-6} & 1.132\cdot 10^{-6} & 4.068\cdot 10^{-6} & -4.322\cdot 10^{-6} \\ 4.753\cdot 10^{-7} & 3.058\cdot 10^{-6} & -4.322\cdot 10^{-6} & 5.096\cdot 10^{-6} \end{pmatrix} \\ K_t &\to \begin{pmatrix} 3.022 & 0.497 & -2.764 & -0.746 \\ -1.655 & -0.396 & 4.162 & -0.203 \\ 0.038 & -0.056 & -0.015 & 0.011 \\ -0.123 & 0.060 & -0.018 & 0.099 \end{pmatrix} \end{aligned}\right\} \text{as } t \to T_{\text{train}}.$$

For the two trend-case, the innovations are closer to zero than for the one-trend case in their magnitudes, but steady-state behaviour is still not shown. This means that $H\hat{x}_t^-$ is generally closer to the measurement than for the one trend case, which agrees with Figure 19.

Three trends

We now consider the case where we have three trends, which gives us $n = 6$. We now have $F = \begin{pmatrix} I_3 & I_3 \\ 0_{3\times 3} & I_3 \end{pmatrix}$. As before, the other parameters are estimated by the EMKF algorithm. The estimates with the largest log-likelihood value are given by

$$\hat{Q} = \begin{pmatrix} 3.385\cdot 10^{-2} & 4.876\cdot 10^{-3} & -2.523\cdot 10^{-2} & -8.138\cdot 10^{-4} & -4.311\cdot 10^{-4} & 9.581\cdot 10^{-4} \\ 4.876\cdot 10^{-3} & 2.760\cdot 10^{-2} & -3.501\cdot 10^{-2} & -2.230\cdot 10^{-4} & -6.470\cdot 10^{-4} & 7.627\cdot 10^{-4} \\ -2.523\cdot 10^{-2} & -3.501\cdot 10^{-2} & 6.540\cdot 10^{-2} & 6.825\cdot 10^{-4} & 1.022\cdot 10^{-3} & -1.470\cdot 10^{-3} \\ -8.138\cdot 10^{-4} & -2.230\cdot 10^{-4} & 6.825\cdot 10^{-4} & 7.659\cdot 10^{-5} & 2.629\cdot 10^{-5} & -7.648\cdot 10^{-5} \\ -4.311\cdot 10^{-4} & -6.470\cdot 10^{-4} & 1.022\cdot 10^{-3} & 2.629\cdot 10^{-5} & 2.166\cdot 10^{-5} & -3.834\cdot 10^{-5} \\ 9.581\cdot 10^{-4} & 7.627\cdot 10^{-4} & -1.470\cdot 10^{-3} & -7.648\cdot 10^{-5} & -3.834\cdot 10^{-5} & 8.811\cdot 10^{-5} \end{pmatrix}, \quad (35)$$

$$\hat{H} = \begin{pmatrix} 0.516 & 0.577 & 0.425 & 1.929 & 0.875 & -1.956 \\ 0.601 & 0.151 & 0.692 & -3.266 & -0.515 & 5.040 \\ 0.193 & 0.651 & 0.530 & -0.819 & -0.136 & 0.413 \\ 0.394 & 0.528 & 0.572 & 2.315 & 1.756 & -2.689 \end{pmatrix},$$

$$\hat{R} = \begin{pmatrix} 1.437\cdot 10^{-4} & -1.556\cdot 10^{-4} & 6.819\cdot 10^{-5} & 1.574\cdot 10^{-4} \\ -1.556\cdot 10^{-4} & 9.520\cdot 10^{-4} & -2.793\cdot 10^{-6} & -8.972\cdot 10^{-5} \\ 6.819\cdot 10^{-5} & -2.793\cdot 10^{-6} & 8.352\cdot 10^{-5} & 6.564\cdot 10^{-5} \\ 1.574\cdot 10^{-4} & -8.972\cdot 10^{-5} & 6.564\cdot 10^{-5} & 1.987\cdot 10^{-4} \end{pmatrix},$$

$$\hat{\xi} = \begin{pmatrix} -0.966 \\ -1.099 \\ -0.158 \\ -0.030 \\ 0.008 \\ 0.014 \end{pmatrix} \text{ and } \hat{\Lambda} = \begin{pmatrix} 2.064\cdot 10^{-5} & 2.250\cdot 10^{-6} & -1.473\cdot 10^{-5} & -7.037\cdot 10^{-7} & -2.836\cdot 10^{-7} & 7.451\cdot 10^{-7} \\ 2.250\cdot 10^{-6} & 1.585\cdot 10^{-5} & -1.976\cdot 10^{-5} & -9.818\cdot 10^{-8} & -3.704\cdot 10^{-7} & 4.204\cdot 10^{-7} \\ -1.473\cdot 10^{-5} & -1.976\cdot 10^{-5} & 3.739\cdot 10^{-5} & 5.235\cdot 10^{-7} & 6.092\cdot 10^{-7} & -9.638\cdot 10^{-7} \\ -7.037\cdot 10^{-7} & -9.818\cdot 10^{-8} & 5.235\cdot 10^{-7} & 8.331\cdot 10^{-8} & 2.087\cdot 10^{-8} & -7.562\cdot 10^{-8} \\ -2.836\cdot 10^{-7} & -3.704\cdot 10^{-7} & 6.092\cdot 10^{-7} & 2.087\cdot 10^{-8} & 1.900\cdot 10^{-8} & -3.189\cdot 10^{-8} \\ 7.451\cdot 10^{-7} & 4.204\cdot 10^{-7} & -9.638\cdot 10^{-7} & -7.562\cdot 10^{-8} & -3.189\cdot 10^{-8} & 8.308\cdot 10^{-8} \end{pmatrix}.$$

We observe estimates of similar magnitude as for the two-trend case. Again, the measurement noise's covariance matrix is rather small in magnitude, compared to the one-trend case.

Once again, the state-space system is observable by the Kalman rank test for observability (rank 6, which is full column rank), using $\hat{H}$. Also, we again failed to obtain maximum likelihood estimates, since the approximated Hessian of the log-likelihood of the measurements has both positive and negative eigenvalues, evaluated at the above parameter estimates, meaning that the estimates correspond to a saddle point. Its spectrum is indicated by $\sigma(\hat{H}_s) \subset (-14165.50, 711.44)$. Similar to preceding cases, Hessian computations for other all other parameter estimates resulting from the EMKF algorithm did not indicated that we have reached a local maximum of likelihood of the measurements. Hence, we cannot consider the asymptotic distribution of the parameters nor the confidence intervals for the parameter estimates.

As before, we proceed with the estimates that we have. Applying the Kalman- filter and smoother with the parameter estimates (35) gives us the results visualized in Figure 21. Again, we have rescaled everything back to the original scale. Trend approximation is done in the same way as for the two-trend case.

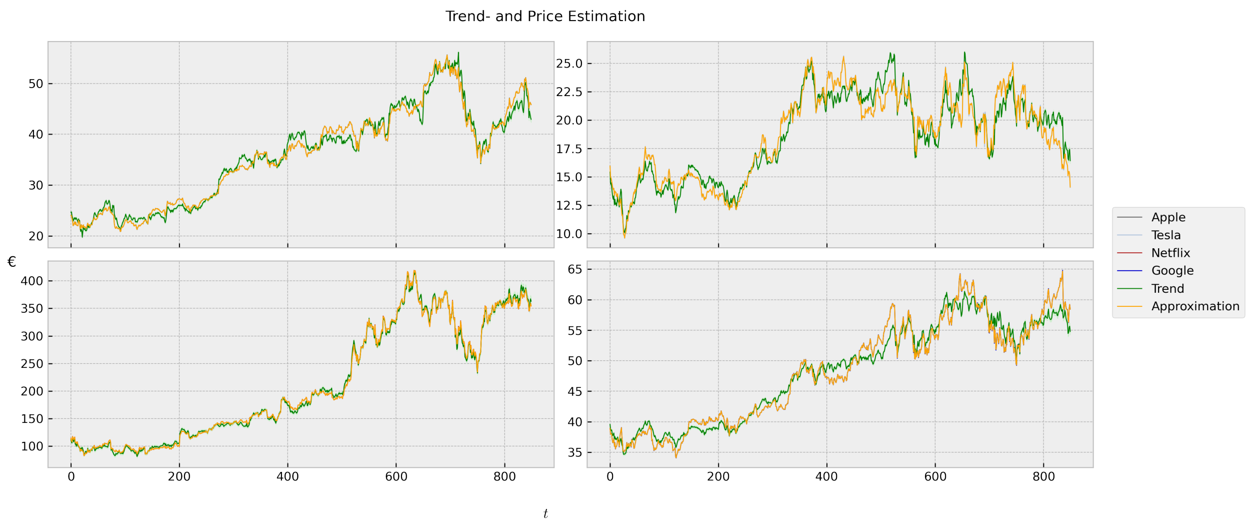


Figure 21: Trend- and price approximation for three trends. 95% confidence intervals for the trend are incorporated in the figure, but are too small to be visible.

The trend line shows similar behaviour as for the two trend-case, but even closer to the measurements. Let us see if this is justified by the accuracy of the mean absolute percentage errors, which are given in Table 14.

Company	MAPE
Apple	0.076
Tesla	0.158
Netflix	0.076
Google	0.052

Table 14: Mean absolute percentage training errors for the three trend-case

Note that the MAPEs in Table 14 are smaller than for the two trend-case in Table 13, except for Tesla. Unfortunately, this is not explainable from the visualizations, as they visually coincide with the measurements.

We now consider the steady-state behaviour of the estimated model. To this end, we compute the innovation sequence and check whether the a posteriori covariance matrix P_t and the Kalman gain K_t have stabilized. We obtain the following innovation sequence.

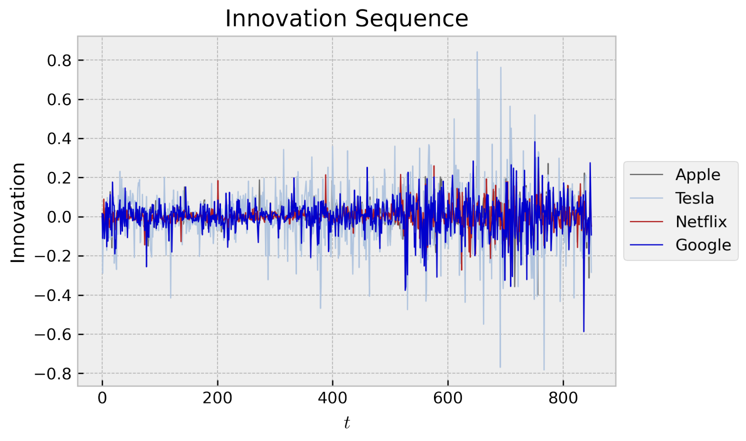


Figure 22: The innovation sequence for the three trend-case.

Figures 20 and 22 are almost indistinguishable from each other, meaning that the two trend- and three trend-case show very similar innovation sequences. Comparing to the one trend-case, we again observe a smaller upper bound for the innovations. The steady-state a posteriori error covariance matrix and Kalman gain are given by

$$\left.\begin{aligned} P_t &\to \begin{pmatrix} 1.154\cdot10^{-3} & 3.398\cdot10^{-4} & -1.425\cdot10^{-3} & -5.314\cdot10^{-4} & 5.148\cdot10^{-4} & -1.882\cdot10^{-4} \\ 3.398\cdot10^{-4} & 1.634\cdot10^{-3} & -2.181\cdot10^{-3} & -3.183\cdot10^{-4} & 4.271\cdot10^{-4} & -1.037\cdot10^{-4} \\ -1.425\cdot10^{-3} & -2.181\cdot10^{-3} & 4.057\cdot10^{-3} & 8.665\cdot10^{-4} & -9.848\cdot10^{-4} & 2.869\cdot10^{-4} \\ -5.314\cdot10^{-4} & -3.183\cdot10^{-4} & 8.665\cdot10^{-4} & 2.782\cdot10^{-4} & -2.733\cdot10^{-4} & 9.794\cdot10^{-5} \\ 5.148\cdot10^{-4} & 4.271\cdot10^{-4} & -9.848\cdot10^{-4} & -2.733\cdot10^{-4} & 3.088\cdot10^{-4} & -8.615\cdot10^{-5} \\ -1.882\cdot10^{-4} & -1.037\cdot10^{-4} & 2.869\cdot10^{-4} & 9.794\cdot10^{-5} & -8.615\cdot10^{-5} & 4.780\cdot10^{-5} \end{pmatrix}, \\ K_t &\to \begin{pmatrix} 2.916 & 0.618 & -1.409 & -1.571 \\ 1.696 & -0.713 & 1.793 & -2.050 \\ -3.224 & 0.579 & -0.044 & 3.395 \\ 0.001 & -0.033 & -0.100 & 0.112 \\ -0.052 & -0.012 & 0.014 & 0.063 \\ 0.054 & 0.033 & 0.057 & -0.137 \end{pmatrix} \end{aligned}\right\} \text{as } t \to T_{\text{train}}.$$

It seems that the variances of the steady-state a posteriori error covariance matrix is larger for the three-trend case is larger than for the two-trend case. This means that the prediction uncertainty of the Kalman filter is larger for the three trend case, which is reflected in the Kalman gain.

Four trends

Finally, we consider the four trend-case. Here, $n = 8$. Since we now model each stock price by its own trend, we have

$$F = \begin{pmatrix} I_4 & I_4 \\ 0_{4\times4} & I_4 \end{pmatrix} \text{ and } H = \begin{pmatrix} I_4 & 0_{4\times4} \end{pmatrix}.$$

The remainder of the parameters are estimated by the EMKF algorithm, that is $\theta = \{Q, R, \xi, \Lambda\}$. As before, the estimates corresponding to the largest log-likelihood are given by

$$\begin{aligned} \hat{Q} &= \begin{pmatrix} 3.619\cdot10^{-3} & 2.815\cdot10^{-3} & 1.495\cdot10^{-3} & 2.880\cdot10^{-3} & -2.451\cdot10^{-5} & 6.397\cdot10^{-6} & -1.478\cdot10^{-5} & -2.394\cdot10^{-5} \\ 2.815\cdot10^{-3} & 1.916\cdot10^{-2} & 2.682\cdot10^{-3} & 3.687\cdot10^{-3} & -4.313\cdot10^{-7} & -1.921\cdot10^{-4} & 9.498\cdot10^{-6} & 8.315\cdot10^{-6} \\ 1.495\cdot10^{-3} & 2.682\cdot10^{-3} & 3.140\cdot10^{-3} & 2.550\cdot10^{-3} & 2.170\cdot10^{-6} & -3.817\cdot10^{-5} & -8.487\cdot10^{-6} & -8.264\cdot10^{-6} \\ 2.880\cdot10^{-3} & 3.687\cdot10^{-3} & 2.550\cdot10^{-3} & 6.870\cdot10^{-3} & 1.184\cdot10^{-5} & -6.541\cdot10^{-5} & 2.092\cdot10^{-5} & -4.259\cdot10^{-5} \\ -2.451\cdot10^{-5} & -4.313\cdot10^{-7} & 2.170\cdot10^{-6} & 1.184\cdot10^{-5} & 4.256\cdot10^{-7} & -6.158\cdot10^{-7} & 3.434\cdot10^{-7} & 6.190\cdot10^{-8} \\ 6.397\cdot10^{-6} & -1.921\cdot10^{-4} & -3.817\cdot10^{-5} & -6.541\cdot10^{-5} & -6.158\cdot10^{-7} & 3.124\cdot10^{-6} & -5.943\cdot10^{-7} & -6.919\cdot10^{-8} \\ -1.478\cdot10^{-5} & 9.498\cdot10^{-6} & -8.487\cdot10^{-6} & 2.092\cdot10^{-5} & 3.434\cdot10^{-7} & -5.943\cdot10^{-7} & 4.194\cdot10^{-7} & -5.387\cdot10^{-8} \\ -2.394\cdot10^{-5} & 8.315\cdot10^{-6} & -8.264\cdot10^{-6} & -4.259\cdot10^{-5} & 6.190\cdot10^{-8} & -6.919\cdot10^{-8} & -5.387\cdot10^{-8} & 4.175\cdot10^{-7} \end{pmatrix}, \\ \hat{R} &= \begin{pmatrix} 1.428\cdot10^{-4} & -7.549\cdot10^{-5} & 9.766\cdot10^{-5} & 1.492\cdot10^{-4} \\ -7.549\cdot10^{-5} & 1.007\cdot10^{-3} & 5.104\cdot10^{-6} & 1.136\cdot10^{-5} \\ 9.766\cdot10^{-5} & 5.104\cdot10^{-6} & 1.043\cdot10^{-4} & 1.039\cdot10^{-4} \\ 1.492\cdot10^{-4} & 1.136\cdot10^{-5} & 1.039\cdot10^{-4} & 1.961\cdot10^{-4} \end{pmatrix}, \\ \hat{\xi} &= \begin{pmatrix} -1.268 & -0.746 & -0.952 & -1.134 & 0.001 & -0.005 & 0.001 & 0.000 \end{pmatrix}^\top, \\ \hat{\Lambda} &= \begin{pmatrix} 5.597\cdot10^{-6} & 4.020\cdot10^{-6} & 2.363\cdot10^{-6} & 4.430\cdot10^{-6} & -5.594\cdot10^{-8} & 2.661\cdot10^{-9} & -3.731\cdot10^{-8} & -5.435\cdot10^{-8} \\ 4.020\cdot10^{-6} & 3.010\cdot10^{-5} & 3.963\cdot10^{-6} & 5.505\cdot10^{-6} & -1.819\cdot10^{-8} & -3.868\cdot10^{-7} & 1.081\cdot10^{-8} & 1.989\cdot10^{-9} \\ 2.363\cdot10^{-6} & 3.963\cdot10^{-6} & 4.793\cdot10^{-6} & 3.887\cdot10^{-6} & -1.092\cdot10^{-8} & -6.096\cdot10^{-8} & -3.304\cdot10^{-8} & -2.954\cdot10^{-8} \\ 4.430\cdot10^{-6} & 5.505\cdot10^{-6} & 3.887\cdot10^{-6} & 1.045\cdot10^{-5} & 6.253\cdot10^{-10} & -1.019\cdot10^{-7} & 1.325\cdot10^{-8} & -9.085\cdot10^{-8} \\ -5.594\cdot10^{-8} & -1.819\cdot10^{-8} & -1.092\cdot10^{-8} & 6.253\cdot10^{-10} & 1.227\cdot10^{-8} & 7.748\cdot10^{-9} & 8.871\cdot10^{-9} & 1.061\cdot10^{-8} \\ 2.661\cdot10^{-9} & -3.868\cdot10^{-7} & -6.096\cdot10^{-8} & -1.019\cdot10^{-7} & 7.748\cdot10^{-9} & 4.373\cdot10^{-8} & 8.097\cdot10^{-10} & 5.678\cdot10^{-9} \\ -3.731\cdot10^{-8} & 1.081\cdot10^{-8} & -3.304\cdot10^{-8} & 1.325\cdot10^{-8} & 8.871\cdot10^{-9} & 8.097\cdot10^{-10} & 1.216\cdot10^{-8} & 9.610\cdot10^{-9} \\ -5.435\cdot10^{-8} & 1.989\cdot10^{-9} & -2.954\cdot10^{-8} & -9.085\cdot10^{-8} & 1.061\cdot10^{-8} & 5.678\cdot10^{-9} & 9.610\cdot10^{-9} & 1.279\cdot10^{-8} \end{pmatrix}. \end{aligned} \tag{36}$$

Regarding the parameter estimates, we observe the same as before. Also for the four trend-case, the state-space system is observable by the Kalman rank test for observability, using $\hat{H}$. We obtain a rank of of the observability matrix of 8, which is full column rank. However, EMKF again provided estimates which correspond to a saddle point of the log-likelihood of the measurements. Namely, the approximated Hessian of the log-likelihood of the measurements has both positive and negative eigenvalues, evaluated at the above parameter estimates. Its spectrum is indicated by $\sigma(\hat{H}_s) \subset (-600.26, 606.18)$. Similar to preceding cases, Hessian computations for other all other parameter estimates resulting from the EMKF algorithm did not indicated that we have reached a local maximum of log-likelihood of the measurements. Hence, since we do not have maximum likelihood estimates, we cannot consider the asymptotic distribution of the parameters nor the confidence intervals for the parameter estimates.

We gain proceed with the estimates that we have. Applying the Kalman- filter and smoother with the parameter estimates (36) gives us the results visualized by Figure 23. Again, we have rescaled everything back to the original scale. Note that we have company-specific trends for the four trend-case.

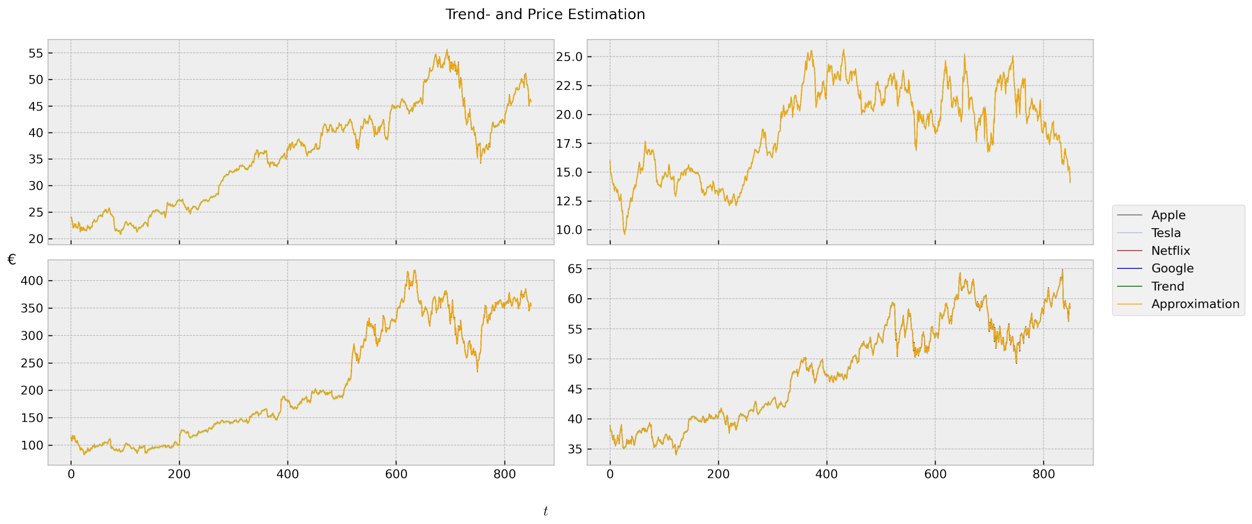


Figure 23: Trend- and price approximation for four trends. 95% confidence intervals for the trend are incorporated in the figure, but are too small to be visible.

Figure 23 shows us that for the four-trend case, the trend, the stock price and its approximation by the trend practically coincide for all four companies. The MAPEs for the training data are given by Table 15.

Company	MAPE
Apple	0.068
Tesla	0.152
Netflix	0.097
Google	0.044

Table 15: Mean absolute percentage training errors for the four trend-case.

Note that for the four trend-case, the MAPEs for the training data has slightly dropped, compared to the three trend-case. However, for Tesla, the MAPE for the two-trend case is still smaller.

The innovation sequence for the four trend-case is shown in Figure 24.

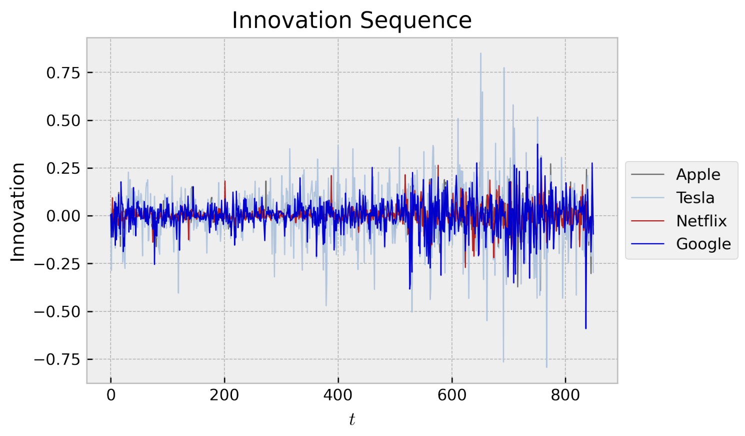


Figure 24: The innovation sequence for the four trend-case.

We also have

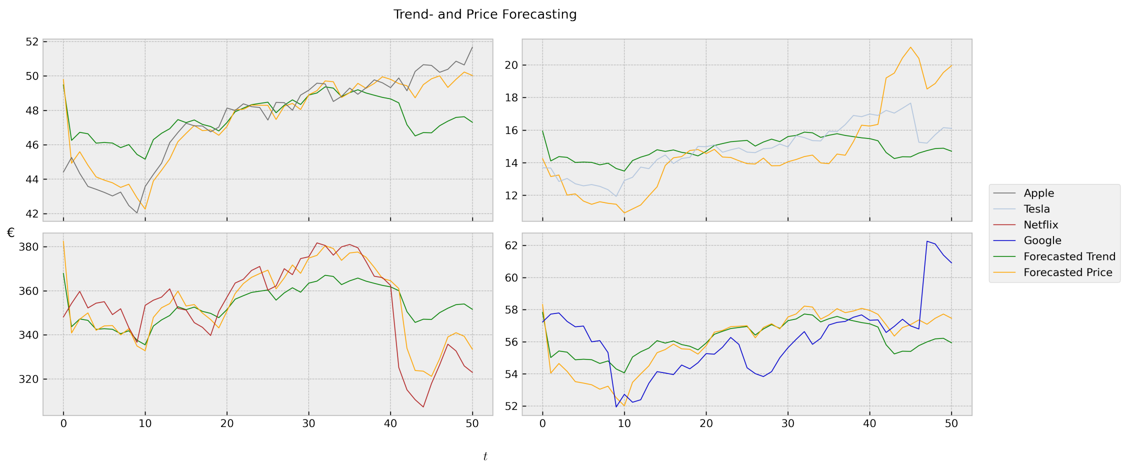
$$\begin{aligned}
P_t &\longrightarrow \begin{pmatrix}
1.351\cdot10^{-4} & -6.269\cdot10^{-5} & 9.232\cdot10^{-5} & 1.418\cdot10^{-4} & 1.692\cdot10^{-6} & -2.477\cdot10^{-6} & 1.081\cdot10^{-6} & 7.649\cdot10^{-8} \\
-6.269\cdot10^{-5} & 9.476\cdot10^{-4} & 1.144\cdot10^{-5} & 1.969\cdot10^{-5} & -1.876\cdot10^{-6} & 9.845\cdot10^{-6} & -6.440\cdot10^{-7} & 1.354\cdot10^{-6} \\
9.232\cdot10^{-5} & 1.144\cdot10^{-5} & 9.996\cdot10^{-5} & 9.872\cdot10^{-5} & 1.313\cdot10^{-6} & -1.685\cdot10^{-6} & 1.144\cdot10^{-6} & -1.550\cdot10^{-7} \\
1.418\cdot10^{-4} & 1.969\cdot10^{-5} & 9.872\cdot10^{-5} & 1.884\cdot10^{-4} & 1.733\cdot10^{-6} & -2.094\cdot10^{-6} & 1.087\cdot10^{-6} & 5.390\cdot10^{-7} \\
1.692\cdot10^{-6} & -1.876\cdot10^{-6} & 1.313\cdot10^{-6} & 1.733\cdot10^{-6} & 5.312\cdot10^{-5} & 4.657\cdot10^{-6} & 2.779\cdot10^{-5} & 2.924\cdot10^{-5} \\
-2.477\cdot10^{-6} & 9.845\cdot10^{-6} & -1.685\cdot10^{-6} & -2.094\cdot10^{-6} & 4.657\cdot10^{-6} & 3.588\cdot10^{-4} & 3.745\cdot10^{-6} & 1.691\cdot10^{-5} \\
1.081\cdot10^{-6} & -6.440\cdot10^{-7} & 1.144\cdot10^{-6} & 1.087\cdot10^{-6} & 2.779\cdot10^{-5} & 3.745\cdot10^{-6} & 4.421\cdot10^{-5} & 5.541\cdot10^{-6} \\
7.649\cdot10^{-8} & 1.354\cdot10^{-6} & -1.550\cdot10^{-7} & 5.390\cdot10^{-7} & 2.924\cdot10^{-5} & 1.691\cdot10^{-5} & 5.541\cdot10^{-6} & 8.294\cdot10^{-5}
\end{pmatrix} \\
K_t &\longrightarrow \begin{pmatrix}
0.969 & 0.011 & -0.017 & -0.006 \\
0.050 & 0.945 & 0.019 & -0.003 \\
-0.017 & 0.005 & 0.973 & 0.001 \\
-0.025 & 0.007 & -0.011 & 0.985 \\
0.004 & -0.002 & 0.006 & 0.003 \\
0.011 & 0.011 & -0.016 & -0.011 \\
-0.001 & -0.001 & 0.012 & 0.000 \\
-0.004 & 0.001 & -0.007 & 0.010
\end{pmatrix}
\end{aligned} \qquad \text{as } t \longrightarrow T_{\text{train}}.$$

Figure 24 shows that the bounds for the innovations resulting from the Kalman filter became even smaller for the four-trend case. However, steady-state behaviour is not observed here, just as for the other cases. Regarding the steady-state Kalman gain: the entries are smaller in magnitude than for the two- and three trend-cases. This is reflected in the steady-state a posteriori error covariance matrix: the Kalman filter's prediction uncertainty is rather small, indicated by the trace.

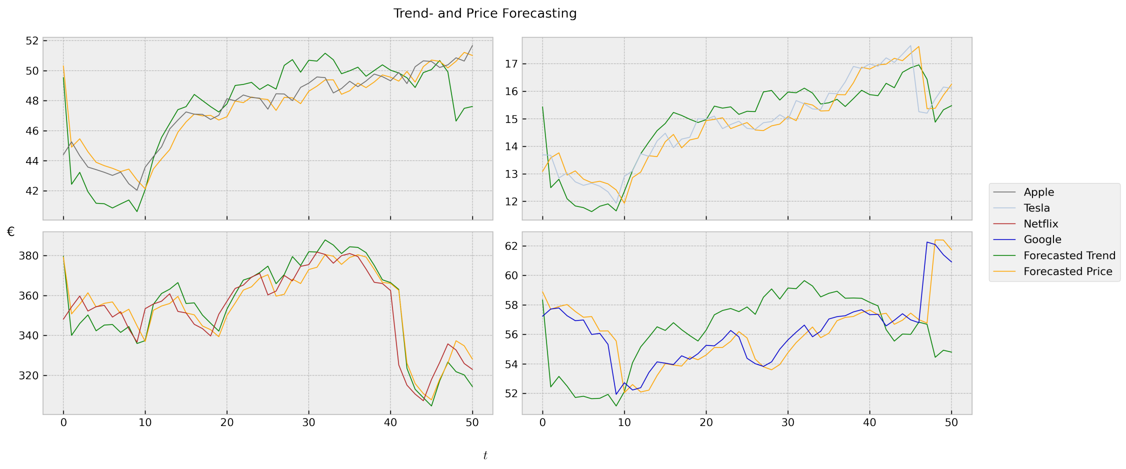
Overall, the EMKF-learned Kalman- filter and smoother provides hidden state estimates which could be mapped to the stock price measurements quite accurately in terms of the MAPE. We now move on to the testing phase, where we apply the forecasting procedure..

4.3.3 Forecasting

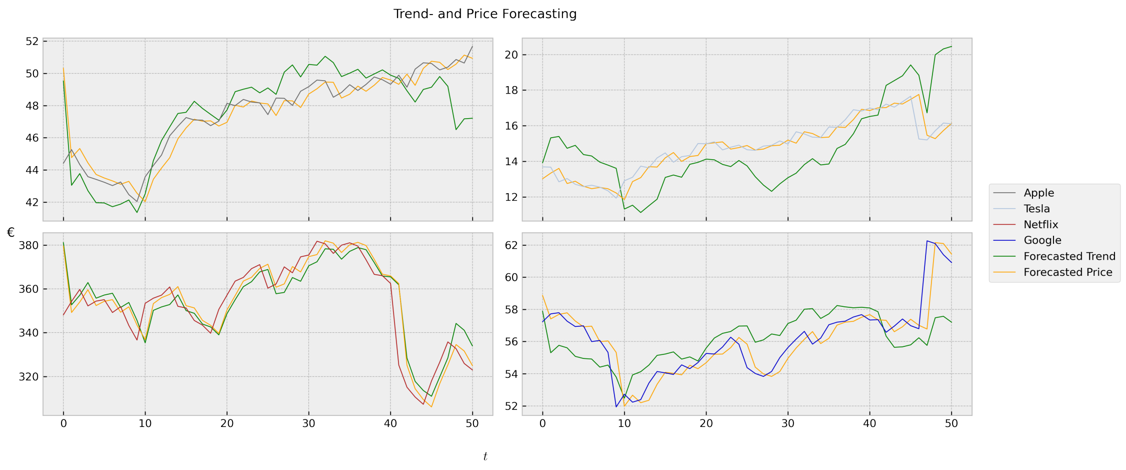
Now that we have been able to estimate the unknown parameters for each case, we can apply forecasting to the stock prices. We follow the online Kalman filter method in Section 3.4. In Figure 25, one could observe the estimated trend, the estimated stock price and the actual value for each company.



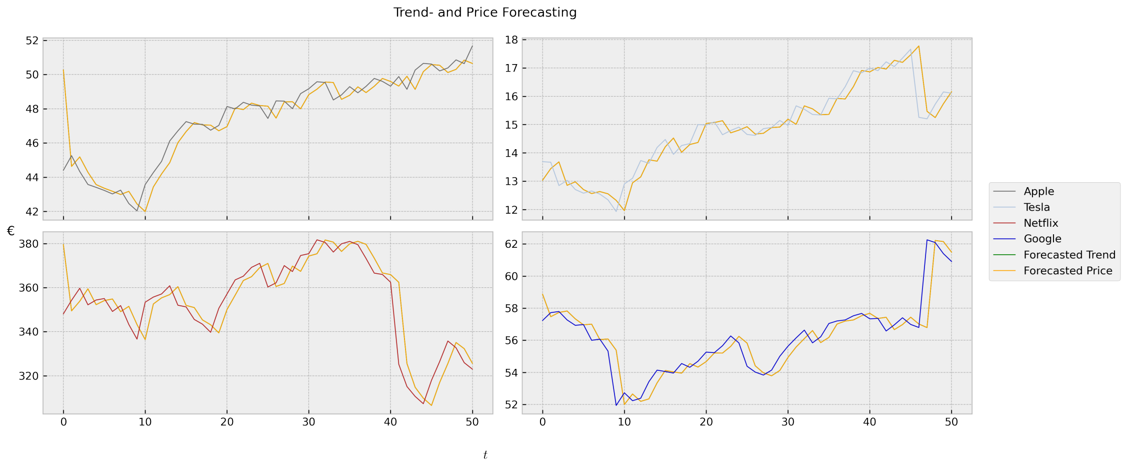
(a) Stock price forecast for one trend.



(b) Stock price forecast for two trends.



(c) Stock price forecast for three trends.



(d) Stock price forecast for four trends.

Figure 25: The forecast for all stock prices for each number of trends.

From Figure 25, we observe that for the one trend-case in Figure 25a, the forecast deviates more from the actual price than whenever we use more trends. For the two trend- and three trend-cases, the forecast seems to similar and more accurate. Due to the way we have defined H for the four-trend case, the trend coincides with the stock price forecast. Also, the four-trend forecast seems similar to the two trend- and three trend-cases. In Table 16, we quantify the accuracy of the forecast in terms of the MAPE. For reference purposes, we also included the training errors from Tables 12–15.

Company	MAPE							
	One trend		Two trends		Three trends		Four trends	
	Train	Test	Train	Test	Train	Test	Train	Test
Apple	1.827	1.482	0.081	1.387	0.076	1.339	0.068	1.290
Tesla	7.639	9.038	0.147	2.181	0.158	2.167	0.152	2.269
Netflix	5.439	2.190	0.089	1.803	0.076	1.779	0.097	1.859
Google	3.216	3.018	0.065	1.220	0.052	1.098	0.044	1.113

Table 16: Train- and test MAPEs trend wise

The forecasting errors seems to be the smallest for the three-trend case, except for Apple. For Apple, the test error is the smallest for the four-trend case. However, an additional trend increases the hidden state dimension n by two. Since the usage of less trends is less computationally expensive, we would prefer the usage of three trends over four. However, from Table 8, we have concluded that the hidden state dimension does not heavily improves the computational effort, compared to the sample size T. Therefore, for interpretability purposes, one could prefer the usage of four trends over three, since each stock price would have its own trend in that case. The same holds for the case of one common trend, but note the larger forecasting errors for the usage of one trend. This is justified by Figure 25 as we have already observed.

To conclude this forecasting section, in terms of forecasting errors and with computational effort in mind, the three trend-case gives the most accurate predictions in terms of the MAPE. However, if one would prefer to model each stock price by its own trend, the four trend-case would have been advised.

5 Conclusion

In this thesis, we have considered modelling time series in the state-space sense in a detailed manner. We have derived the five equations forming the Kalman filter and showed why the Kalman filter is an optimal hidden state estimator under certain assumptions: by minimizing the total a posteriori error variance, the predict-update nature of the Kalman filter provides optimal hidden state estimates. We have also derived smoothing equations which corrects the Kalman filter estimates: the Kalman smoother.

Furthermore, by employing the expectation-maximization (EM) algorithm, we could apply its methodology to the state-space setting to introduce the EMKF algorithm: the EM algorithm applied to the Kalman filter. Here, the E-step consists of applying the Kalman- filter and smoother with current parameter estimates to obtain the expected log-likelihood function of the complete data, conditioned on the observed data. In the M-step, we maximize this function w.r.t. the parameters we wish to estimate. By the nature of the EM algorithm, the log-likelihood of the measurements monotonically increases throughout the iteration procedure. Finally, we have considered the asymptotic distributions of maximum likelihood estimates obtained from the EMKF algorithm. It turns out that for a sufficiently large sample size, the parameter estimates are consistent and attain normality with explicit knowledge of the mean and covariance matrix. The Fisher information matrix can be estimated by numerically approximation of the Hessian of the log-likelihood of the measurements.

The main result of this thesis is the construction of the EMKF algorithm in Python by the author itself, which provides more flexibility than the existing one, which is the `em` method in the `pykalman` library: the written EMKF algorithm actually iterates until the log-likelihood of the measurements has converged to a stationary point and the difference in parameter estimates stabalizes (or until the maximum number of iterations is reached). In this way, we are actually able to obtain maximum likelihood estimates. One can manually set the the thresholds for the convergence criteria. Also, one can initialize the algorithm itself and select the parameters one wishes to be optimized, which was already possible in the existing EMKF implemntation.

The Kalman- filter and smoother have also been implemented by the author. Simulations show that the written functions operate as expected, as for the written `EMKF` function. For the Kalman- filter and smoother functions, the results are identical to the existing ones in the `pykalman` library. Furthermore, a comparison with an example from literature show the exact same results [16].

We have applied the methodology to real-world stock data in order to forecast their values. To this end, we have modelled the stock prices by a local linear trend model, which we brought into state-space form. By considering one up to four trends, we have estimated the model parameters which are unknown by the EMKF algorithm. Although the forecasting errors are rather small for the usage of more than two trends, the parameter estimates did not correspond to maximum likelihood estimates: all runs of the EMKF algorithm yielded parameter estimates which corresponded to saddle points of the log-likelihood of the measurements. Since the Fisher information matrix cannot be estimated for parameter estimates corresponding to saddle points of the log-likelihood, their asymptotic distribution could not be identified, meaning that confidence intervals for these estimates are not available.

6 Discussion

The last section of this thesis is devoted to a critical discussion of our approach and findings and suggest further research on the topic of Kalman filtering and the EMKF algorithm.

Modelling continuous random variables by a Gaussian distribution comes with a rather suitable mathematical framework. The fact that e.g. the Kalman filter recursions are rather straightforward to derive, follows from the result that the Gaussian distribution is invariant under linear transformation (that is, a linear combination of Gaussians is Gaussian). However, stock prices are not necessarily Gaussian distributed, while we have assumed them being so. More accurate results could have been obtained if the actual distributed of the stock prices would have been analyzed (it could still be Gaussian). One popular choice is to model stock prices by heavy-tail distributions, which are related to the notion of so-called α-stable distributions (the Gaussian distribution is in fact an α-stable distribution).

Also, while the Gaussian assumption of the system may be valid, the linearity of the dynamics (in time) for e.g. stock prices may not. Stock prices are often described as rather chaotic and this may better be modelled in terms of nonlinear dynamics. Fortunately, the *extended Kalman filter* already exists, which is the generalization of the Kalman filter for nonlinear dynamics. For highly nonlinear problems where the performance of the extended Kalman filter drops, one could use the *unscented Kalman filter*. This follows from the fact that the extended Kalman filter is essentially applied to linearizations of the dynamics. The unscented Kalman filter tackles problems this linearizations could cause.

Furthermore, it would have been beneficial to this thesis to focus more on the stability of the Kalman filter with the steady-state behaviour in particular. Stability analysis could describe why steady-state for the innovations has not been reached in the stock price application, while it follows partially from the theoretical framework that steady-state innovations should have been achieved (but necessarily in exponential time). The state-space formulation could e.g. have been adjusted to a stable or stabilized system. In the essence, stability of the Kalman filter is of greater importance as reflected in this thesis.

Next, while we simultaneously estimated ξ and Λ (recall that we assume $X_0 \sim \mathcal{N}_n(\xi, \Lambda)$, it is said by E. Holmes that [6] and [16] discuss that this is not valid and could lead to poor initial conditions estimates. This is justified by the results within this thesis. However, the mentioned discussions are not found in the corresponding literature to consider this in more detail. Two allowable cases that [6] mentions, according to Holmes, are

1. We consider $x_0 = \xi$ to be fixed (and omit Λ from our models).
2. We consider ξ and Λ to be known.

[16] argues that one can ξ and Λ could be estimated, but not simultaneously. This would mean that we have to apply the EMKF algorithm to all unknown parameters, except for ξ or Λ, and apply EMKF again with the obtained estimates to estimate either ξ or Λ solely. This would mean that in the first iteration, the ignored parameter is assumed to be known. For further research, one could verify which estimation procedures are valid and develop a general EMKF framework for estimation of the initial conditions for the Kalman filter.

Regarding the EMKF algorithm, it must be noted that all parameter estimates in Section 4.3 from the EMKF algorithm corresponded to saddle points of the log-likelihood function of the measurements. Although it makes sense for a large-dimensional function (note that in the worst case, we are estimating $3n^2 + (p+1)n + p^2$ parameter entries) to be non-concave, it is worrying that no local maxima have been found for different values of n. A possible solution to this problem would be to develop an EM(KF) algorithm which classifies each stationary point it converges to, identify in which direction(s) of a saddle point the log-likelihood increases and keep iterating in one that direction(s). In this way, saddle points could be overcome. One might recognize such an approach by (stochastic) gradient descent with momentum. Then, convergence to a local maximum of the log-likelihood can be guaranteed, meaning that we obtain maximum likelihood estimates.

Furthermore, although the EM algorithm in general provides a rather simple estimation procedure, its rate of convergence is small compared to e.g. Newton-Rapshon. Although the state- and measurement dimension are a great factor in the computational complexity of Newton-Rapshon (Newton-Raphson requires Hessian computations/approximations during each iteration), Newton-Rapshon may find stationary points faster. However, Newton-Rapshon suffers from the same problems as the EM algorithm regarding convergence to saddle points. For gradient methods, such as gradient descent, methodology exists to overcome saddle points, such as the momentum procedure mentioned above. Hence, suggestions for further research are designing an EM algorithm (or EMKF algorithm) which could overcome saddle points or develop gradient based methods for parameter estimation for the state-space setting.

Other suggestions for future research are:

- If one vectorizes the entire parameter set θ into ψ, analytical derivations of the Hessian of the log-likelihood of the measurements should be feasible. Due to time constrains, this could not be achieved in this thesis, but it would be beneficial over numerical approximations to be able to analytically compute the Hessian in terms of both computational complexity and accuracy.
- Since the online Kalman filter yields explicit parameters for the normality of the forecast values, prediction intervals should be easy to obtain. This could be employed in future forecasting applications.

References

[1] Brockwell, P. J. and Davis, R. A. (2009). Time series: theory and methods. 2nd Edition. Springer-Verlag. ISBN: 9781441903198.

[2] Brown, R. G. and Hwang, P. Y. (2012). Introduction to Random Signals and Applied Kalman Filtering with Matlab Exercises. 4th Edition. Wiley. ISBN: 9780470609699.

[3] Caines, P. E. (1988). Linear Stochastic Systems. Wiley Series in Probability and Statistics. Wiley. ISBN: 9780471081012.

[4] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1, 1–38. DOI: 10.1111/j.2517-6161.1977.tb01600.x.

[5] Durbin, J. and Koopman, S. J. (2012). Time series analysis by state space methods. 2nd Edition. OUP Oxford. ISBN: 9780199641178.

[6] Harvey, A. C. (1990). Forecasting, Structural Time Series Models and the Kalman Filter. Cambridge University Press. ISBN: 9780521321969.

[7] Hastie, T., Tibshirani, R., and Friedman, J. H. (2009). The elements of statistical learning: data mining, inference, and prediction. 2nd Edition. Springer. ISBN: 9780387848587.

[8] Kai-Tai, F. and Yao-Ting, Z. (1990). Generalized multivariate analysis. Science Press Beijing and Springer-Verlag, Berlin. ISBN: 9780387176512.

[9] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of Basic Engineering* 82.1, 35–45. DOI: 10.1115/1.3662552.

[10] Magnus, J. R. and Neudecker, H. (2019). Matrix differential calculus with applications in statistics and econometrics. 3rd Edition. Wiley. ISBN: 9781119541202.

[11] Maybeck, P. S. (1979). Stochastic models, estimation and control. Vol. 141. Academic Press. ISBN: 0124807011.

[12] Meng, L. (2016). Method for Computation of the Fisher Information Matrix in the Expectation-Maximization Algorithm.

[13] Morrison, G. W. and Pike, D. H. (1977). Kalman filtering applied to statistical forecasting. *Management Science* 23.7, 768–774. DOI: 10.1287/mnsc.23.7.768.

[14] Sarkka, S. (2013). Bayesian Filtering and Smoothing. Vol. 3. Cambridge University Press. ISBN: 9781107619289.

[15] Shumway, R. H. and Stoffer, D. S. (1982). An approach to time series smoothing and forecasting using the EM algorithm. *Journal of time series analysis* 3.4, 253–264. DOI: 10.1111/j.1467-9892.1982.tb00349.x.

[16] Shumway, R. H. and Stoffer, D. S. (2017). Time series analysis and its applications. 4th Edition. Springer. ISBN: 9783319524528.

[17] Stoorvogel, A. A., Trentelman, H. L., and Hautus, M. (2012). Control theory for linear systems. Springer. ISBN: 1852333162.

[18] Visser, I. and Speekenbrink, M. (2022). Mixture and hidden Markov models with R. 1st Edition. Springer. ISBN: 9783031014406.

A Matrix Calculus

Several derivations in this thesis (e.g. the Kalman gain and the update equations in the M-step of the EM algorithm for the Kalman filter/smoother) involve differentiation w.r.t. matrices. The concept of differentiating scalar-, vector- or matrix-valued functions w.r.t. matrices lies within the field of *matrix calculus*. This section is devoted to introduce the reader to the *scalar-by-matrix derivative*. Also, we provide the typical derivatives w.r.t matrices used in this thesis. We start with the definition of a scalar-by-matrix derivative [8].

Definition A.1 (Scalar-by-matrix derivative)**.** Let $f : \mathbb{R}^{n\times n} \to \mathbb{R}$ be a scalar function of the matrix $X = \begin{pmatrix} x_{11} & \cdots & x_{1q} \\ \vdots & \ddots & \vdots \\ x_{p1} & \cdots & x_{pq} \end{pmatrix} \in \mathbb{R}^{p\times q}$. Then, we define the (partial) derivative of f w.r.t. X by

$$\frac{\partial f}{\partial X}(X) = \begin{pmatrix} \frac{\partial f}{\partial x_{11}}(X) & \cdots & \frac{\partial f}{\partial x_{1q}}(X) \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_{p1}}(X) & \cdots & \frac{\partial f}{\partial x_{pq}}(X) \end{pmatrix}. \tag{A.1}$$

Typical examples of such a function f are $f(X) = \operatorname{Tr} X$ or $f(x) = \ln|X|$, where $|\cdot|$ denotes the determinant operator. An important remarks regarding scalar-by-matrix derivatives is that there exist several conventions in defining the scalar-by-matrix derivative. The Definition A.1 is the definition of a scalar-by-matrix derivative in the so-called *denominator layout*.

[10] considers matrix calculus in detail in the sense of differentials. From the obtained results, several identities for scalar-by-matrix derivatives can be derived which are needed to derive the update equations of the EMKF algorithm. They are given in Lemma A.1. We do not prove the whole lemma, the reader is referred to Chapter 9 of [10].

Lemma A.1. Let $A \in \mathbb{R}^{p\times q}$. Then, we have

1. $\frac{\partial}{\partial A} a^\top A b = \frac{\partial}{\partial A} b^\top A^\top a = a b^\top$ for any $a \in \mathbb{R}^p$ and $b \in \mathbb{R}^q$.
2. $\frac{\partial}{\partial A} a^\top A^\top B A a = 2BAaa^\top$ for any $a \in \mathbb{R}^p$ and symmetric $B \in \mathbb{R}^{p\times p}$.
3. $\frac{\partial}{\partial A} \operatorname{Tr}(AB) = \frac{\partial}{\partial A} \operatorname{Tr}(BA) = B^\top$ for any $B \in \mathbb{R}^{q\times p}$.
4. $\frac{\partial}{\partial A} \operatorname{Tr}(ABA^T) = AB^\top + AB$ for any $B \in \mathbb{R}^{q\times q}$. Furthermore, if B is symmetric, then $\frac{\partial}{\partial A} \operatorname{Tr}(ABA^\top) = 2AB$.
5. $\frac{\partial}{\partial A} \operatorname{Tr}(A^\top BAC) = 2BAC$ for any symmetric $B \in \mathbb{R}^{p\times p}$ and symmetric $C \in \mathbb{R}^{q\times q}$.
6. $\frac{\partial}{\partial A^{-1}} \log|A|^{-1} = A^\top$ if $p = q$ and A is invertible.

Proof. We prove statements 3 and 4.

Let $A = \begin{pmatrix} a_{11} & \cdots & a_{1q} \\ \vdots & \ddots & \vdots \\ a_{p1} & \cdots & a_{pq} \end{pmatrix}$. We first prove the statement 3, before moving on to statement 4.

3. Let $B = \begin{pmatrix} b_{11} & \cdots & b_{1p} \\ \vdots & \ddots & \vdots \\ b_{q1} & \cdots & b_{qp} \end{pmatrix}$. We compute

$$\begin{aligned} AB &= \begin{pmatrix} a_{11} & \cdots & a_{1q} \\ \vdots & \ddots & \vdots \\ a_{p1} & \cdots & a_{pq} \end{pmatrix} \begin{pmatrix} b_{11} & \cdots & b_{1p} \\ \vdots & \ddots & \vdots \\ b_{q1} & \cdots & b_{qp} \end{pmatrix} \\ &= \begin{pmatrix} \sum_{i=1}^q a_{1i} b_{i1} & \cdots & \sum_{i=1}^q a_{1i} b_{ip} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^q a_{pi} b_{i1} & \cdots & \sum_{i=1}^q a_{pi} b_{ip} \end{pmatrix} \Rightarrow \\ \operatorname{Tr}(AB) &= \sum_{i=1}^q a_{1i} b_{i1} + \ldots + \sum_{i=1}^q a_{pi} b_{ip} \\ &= \sum_{i=1}^q \sum_{j=1}^p a_{ji} b_{ij}. \end{aligned}$$

Then, by using (A.1), we obtain

$$\begin{aligned} \frac{\partial}{\partial A} \operatorname{Tr}(AB) &= \begin{pmatrix} \frac{\partial}{\partial a_{11}} \sum_{i=1}^q \sum_{j=1}^p a_{ji} b_{ij} & \cdots & \frac{\partial}{\partial a_{1q}} \sum_{i=1}^q \sum_{j=1}^p a_{ji} b_{ij} \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial a_{p1}} \sum_{i=1}^q \sum_{j=1}^p a_{ji} b_{ij} & \cdots & \frac{\partial}{\partial a_{pq}} \sum_{i=1}^q \sum_{j=1}^p a_{ji} b_{ij} \end{pmatrix} \\ &= \begin{pmatrix} b_{11} & \cdots & b_{q1} \\ \vdots & \ddots & \vdots \\ b_{1p} & \cdots & b_{qp} \end{pmatrix} = B^T. \end{aligned}$$

4. Now, let $B = \begin{pmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{q1} & \cdots & b_{qq} \end{pmatrix}$. We compute

$$\begin{aligned} ABA^T &= \begin{pmatrix} a_{11} & \cdots & a_{1q} \\ \vdots & \ddots & \vdots \\ a_{p1} & \cdots & a_{pq} \end{pmatrix} \begin{pmatrix} b_{11} & \cdots & b_{1p} \\ \vdots & \ddots & \vdots \\ b_{q1} & \cdots & b_{qp} \end{pmatrix} \begin{pmatrix} a_{11} & \cdots & a_{p1} \\ \vdots & \ddots & \vdots \\ a_{1q} & \cdots & a_{pq} \end{pmatrix} \\ &= \begin{pmatrix} \sum_{i=1}^q a_{1i} b_{i1} & \cdots & \sum_{i=1}^q a_{1i} b_{ip} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^q a_{pi} b_{i1} & \cdots & \sum_{i=1}^q a_{pi} b_{ip} \end{pmatrix} \begin{pmatrix} a_{11} & \cdots & a_{p1} \\ \vdots & \ddots & \vdots \\ a_{1q} & \cdots & a_{pq} \end{pmatrix} \\ &= \begin{pmatrix} \sum_{i=1}^q \sum_{j=1}^q a_{1i} b_{ij} a_{1j} & \cdots & \sum_{i=1}^q \sum_{j=1}^q a_{1i} b_{ij} a_{pj} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^q \sum_{j=1}^q a_{pi} b_{ij} a_{1j} & \cdots & \sum_{i=1}^q \sum_{j=1}^q a_{pi} b_{ij} a_{pj} \end{pmatrix} \Rightarrow \\ \operatorname{Tr}(ABA^T) &= \sum_{i=1}^q \sum_{j=1}^q a_{1i} b_{ij} a_{1j} + \ldots + \sum_{i=1}^q \sum_{j=1}^q a_{pi} b_{ij} a_{pj} \\ &= \sum_{i=1}^q \sum_{j=1}^q \sum_{k=1}^p a_{ki} b_{ij} a_{kj}. \end{aligned}$$

Then, by using (A.1) again, we obtain

$$
\begin{aligned}
\frac{\partial}{\partial A}\operatorname{Tr}(ABA^T) &= \begin{pmatrix} \frac{\partial}{\partial a_{11}} \sum_{i=1}^{q} \sum_{j=1}^{q} \sum_{k=1}^{p} a_{ki} b_{ij} a_{kj} & \cdots & \frac{\partial}{\partial a_{1q}} \sum_{i=1}^{q} \sum_{j=1}^{q} \sum_{k=1}^{p} a_{ki} b_{ij} a_{kj} \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial a_{p1}} \sum_{i=1}^{q} \sum_{j=1}^{q} \sum_{k=1}^{p} a_{ki} b_{ij} a_{kj} & \cdots & \frac{\partial}{\partial a_{pq}} \sum_{i=1}^{q} \sum_{j=1}^{q} \sum_{k=1}^{p} a_{ki} b_{ij} a_{kj} \end{pmatrix} \\
&= \begin{pmatrix} \sum_{j=1}^{q} a_{1j} b_{1j} + \sum_{i=1}^{q} a_{1i} b_{i1} & \cdots & \sum_{j=1}^{q} a_{1j} b_{qj} + \sum_{i=1}^{q} a_{1i} b_{iq} \\ \vdots & \ddots & \vdots \\ \sum_{j=1}^{q} a_{pj} b_{1j} + \sum_{i=1}^{q} a_{pi} b_{i1} & \cdots & \sum_{j=1}^{q} a_{pj} b_{qj} + \sum_{i=1}^{q} a_{pi} b_{iq} \end{pmatrix} \\
&= \begin{pmatrix} \sum_{j=1}^{q} a_{1j} b_{1j} & \cdots & \sum_{j=1}^{q} a_{1j} b_{qj} \\ \vdots & \ddots & \vdots \\ \sum_{j=1}^{q} a_{pj} b_{1j} & \cdots & \sum_{j=1}^{q} a_{pj} b_{qj} \end{pmatrix} + \begin{pmatrix} \sum_{i=1}^{q} a_{1i} b_{i1} & \cdots & \sum_{i=1}^{q} a_{1i} b_{iq} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^{q} a_{pi} b_{i1} & \cdots & \sum_{i=1}^{q} a_{pi} b_{iq} \end{pmatrix} \\
&= \begin{pmatrix} a_{11} & \cdots & a_{1q} \\ \vdots & \ddots & \vdots \\ a_{p1} & \cdots & a_{pq} \end{pmatrix} \begin{pmatrix} b_{11} & \cdots & b_{q1} \\ \vdots & \ddots & \vdots \\ b_{1q} & \cdots & b_{qq} \end{pmatrix} + \begin{pmatrix} a_{11} & \cdots & a_{1q} \\ \vdots & \ddots & \vdots \\ a_{p1} & \cdots & a_{pq} \end{pmatrix} \begin{pmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{q1} & \cdots & b_{qq} \end{pmatrix} \\
&= AB^\top + AB.
\end{aligned}
$$

Note that if B is symmetric, then we have

$$\frac{\partial}{\partial A}\operatorname{Tr}(ABA^T) = 2AB.$$

$\square$

B Script

In this section, we show the functions written by the author, including the novel EMKF function. The full script can be found at github.com. As of the publish date, the repository is not visible to the public yet. The repository may become available to the public in the future, where it consists of the functions mentioned below. All the functions are given by:

```python
#!/usr/bin/env python
# coding: utf-8

import numpy as np
import pandas as pd
from sklearn.datasets import make_spd_matrix
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error
import matplotlib.pyplot as plt

import yfinance as yf
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller

plt.style.use('bmh')

#Check whether a matrix is positive definite
def is_pos_def(x):
    return np.all(np.linalg.eigvals(x) > 0)

#Force a matrix to be positive definite if its not
def force_SPD(A, c = 1e-10):
    A = (A + A.T)/2
    if np.amin(np.linalg.eigvals(A)) < 0:
        A += (np.abs(np.amin(np.linalg.eigvals(A))) + c)*np.identity(A.shape[0])
    if np.amin(np.linalg.eigvals(A)) == 0:
        A += c*np.identity(A.shape[0])
    return(A)

def KalmanFilter(F, Q, H, R, z, x_0, P_0):
    T = len(z)
    #Check the dimension of the hidden state
    if isinstance(x_0, np.ndarray) == True:
        n = len(x_0)
    else:
        n = 1
    #Check the dimension of the measurements
    if isinstance(z[0], np.ndarray) == True:
         p = len(z[0])
    else:
        p = 1

    #Multidimensional case
    if n > 1 and p > 1:
        x_hat_minus = np.empty((1, n))
        x_hat = np.array([x_0])
        P_minus = np.empty((1, n, n))
        P = np.array([P_0])
        K = np.empty((1, n, p))

        for i in range(T):
            #State extrapolation equation
            x_hat_minus = np.append(x_hat_minus, [F @ x_hat[i]], axis = 0)

            #Covariance extrapolation equation
            P_minus = np.append(P_minus, [F @ P[i] @ F.T + Q], axis = 0)

            #Kalman gain
            K = np.append(K, [P_minus[i+1] @ H.T @ np.linalg.inv(H @ P_minus[i+1] @ H.T +
    R)], axis = 0)

```

```
            #State update equation
            x_hat = np.append(x_hat, [x_hat_minus[i+1] + K[i+1] @ (z[i] - H @ x_hat_minus[i+1])], axis = 0)

            #Covariance update equation
            P = np.append(P, [(np.identity(n) - K[i+1] @ H) @ P_minus[i+1] @ (np.identity(n) - K[i+1] @ H).T + K[i+1] @ R @ K[i+1].T], axis = 0)

    #One-dimensional measurements case
    elif n > 1 and p == 1:
        x_hat_minus = np.empty((1, n))
        x_hat = np.array([x_0])
        P_minus = np.empty((1, n, n))
        P = np.array([P_0])
        K = np.empty((1, n, p))

        for i in range(T):
            #State extrapolation equation
            x_hat_minus = np.append(x_hat_minus, [F @ x_hat[i]], axis = 0)

            #Covariance extrapolation equation
            P_minus = np.append(P_minus, [F @ P[i] @ F.T + Q], axis = 0)

            #Kalman gain
            K = np.append(K, [P_minus[i+1] @ H.T @ np.linalg.inv(H @ P_minus[i+1] @ H.T + R)], axis = 0)

            #State update equation
            x_hat = np.append(x_hat, [x_hat_minus[i+1] + K[i+1] @ (z[i] - H @ x_hat_minus[i+1])], axis = 0)

            #Covariance update equation
            P = np.append(P, [(np.identity(n) - K[i+1] @ H) @ P_minus[i+1] @ (np.identity(n) - K[i+1] @ H).T + R * K[i+1] @ K[i+1].T], axis = 0)

    #One-dimensional case
    else:
        x_hat_minus = np.empty(1)
        x_hat = np.array([x_0])
        P_minus = np.empty(1)
        P = np.array([P_0])
        K = np.empty(1)

        for i in range(T):
            #State extrapolation equation
            x_hat_minus = np.append(x_hat_minus, [F * x_hat[i]], axis = 0)

            #Covariance extrapolation equation
            P_minus = np.append(P_minus, [F ** 2 * P[i] + Q], axis = 0)

            #Kalman gain
            K = np.append(K, [P_minus[i+1] * H / (H**2 * P_minus[i+1] + R)], axis = 0)

            #State update equation
            x_hat = np.append(x_hat, [x_hat_minus[i+1] + K[i+1] * (z[i] - H * x_hat_minus[i+1])], axis = 0)

            #Covariance update equation
            P = np.append(P, [(1 - K[i+1] * H) ** 2 * P_minus[i+1] + K[i+1] ** 2 * R], axis = 0)

    x_hat_minus = np.delete(x_hat_minus, 0, axis = 0)
    P_minus = np.delete(P_minus, 0, axis = 0)
    K = np.delete(K, 0, axis = 0)

    return(x_hat_minus, P_minus, K, x_hat, P)

def KalmanSmoother(F, x_hat_minus, P_minus, x_hat, P):
    T = len(x_hat_minus)
    if isinstance(x_hat[0], np.ndarray) == True:
        n = len(x_hat[0])
    else:
        n = 1

    #Multidimensional case
    if n > 1:
        x_tilde = np.array([x_hat[T]])
        P_tilde = np.array([P[T]])
        S = np.empty((1, n, n))

        for i in reversed(range(T)):
            #Smoothing gain
            S = np.insert(S, 0, [P[i] @ F.T @ np.linalg.inv(P_minus[i])], axis = 0)

            #State correction
            x_tilde = np.insert(x_tilde, 0, [x_hat[i] + S[0] @ (x_tilde[0] - x_hat_minus[i
    ])], axis = 0)

            #Covariance correction
            P_tilde = np.insert(P_tilde, 0, [P[i] + S[0] @ (P_tilde[0] - P_minus[i]) @ S
    [0].T], axis = 0)

    #One-dimensional case
    else:
        x_tilde = np.array([x_hat[T]])
        P_tilde = np.array([P[T]])
        S = np.empty(1)

        for i in reversed(range(T)):
            #Smoothing gain
            S = np.insert(S, 0, [P[i] * F / P_minus[i]], axis = 0)

            #State correction
            x_tilde = np.insert(x_tilde, 0, [x_hat[i] + S[0] * (x_tilde[0] - x_hat_minus[i
    ])], axis = 0)

            #Covariance correction
            P_tilde = np.insert(P_tilde, 0, [P[i] + S[0]**2 * (P_tilde[0] - P_minus[i])],
    axis = 0)

    S = np.delete(S, len(S) - 1, axis = 0)

    return(S, x_tilde, P_tilde)

def Lag1AutoCov(K, S, F, H, P):
    T = len(P) - 1
    if isinstance(F, np.ndarray) == True:
        n = F.shape[0]
    else:
        n = 1

    #Multidimensional case
    if n > 1:
        V = np.array([(np.identity(n) - K[T-1] @ H) @ F @ P[T-1]])

        for i in reversed(range(1, T)):
            V = np.insert(V, 0, [P[i] @ S[i-1].T + S[i] @ (V[0] - F @ P[i]) @ S[i-1].T],
    axis = 0)

    #One-dimensional case
    else:
        V = np.array([(1 - K[T-1] * H) * F * P[T-1]])

        for i in reversed(range(1, T)):
            V = np.insert(V, 0, [P[i] * S[i-1].T + S[i] * (V[0] - F * P[i]) * S[i-1].T],
    axis = 0)

    return(V)

def ell(H, R, z, x, P):
    T = len(z)
    if isinstance(x[0], np.ndarray) == True:
        n = len(x[0])
    else:
        n = 1
    if isinstance(z[0], np.ndarray) == True:
         p = len(z[0])
    else:
        p = 1

    likelihood = -T*p/2*np.log(2*np.pi)

    #Multidimensional case
    if n > 1 and p > 1:
        for i in range(T):
            likelihood -= 0.5*(np.log(np.linalg.det(H @ P[i] @ H.T + R)) + (z[i] - H @ x[i
    ]).T @ np.linalg.inv(H @ P[i] @ H.T + R) @ (z[i] - H @ x[i]))

    #One-dimensional measurements case
    elif n > 1 and p == 1:
        for i in range(T):
            likelihood -= 0.5*(np.log(np.linalg.det(H @ P[i] @ H.T + R)) + (z[i] - H @ x[i
    ]) ** 2 / (H @ P[i] @ H.T + R))
        likelihood = likelihood[0][0]

    #One-dimensional case
    else:
        for i in range(T):
            likelihood -= 0.5*(np.log(H**2 * P[i] + R) + (z[i] - H * x[i])**2 / (H**2 * P[
    i] + R))

    return(likelihood)

def EMKF(F_0, Q_0, H_0, R_0, z, xi_0, L_0, max_it = 1000, tol_likelihood = 0.01,
    tol_params = 0.005, em_vars = ["F", "Q", "H", "R", "xi", "L"]):
    T = len(z)
    if isinstance(xi_0, np.ndarray) == True:
        n = len(xi_0)
    else:
        n = 1
    if isinstance(z[0], np.ndarray) == True:
         p = len(z[0])
    else:
        p = 1

    #Initialization
    F = np.array([F_0])
    Q = np.array([Q_0])
    H = np.array([H_0])
    R = np.array([R_0])
    xi = np.array([xi_0])
    L = np.array([L_0])

    likelihood = np.empty(1)

    #Multidimensional case
    if n > 1 and p > 1:
        A_5 = np.zeros((p, p))
        for j in range(T):
            A_5 += np.outer(z[j], z[j])

        for i in range(max_it):
            if i > 0 and i % 50 == 0:
                print(f"Iteration {i}")
            #E-step
            x_hat_minus, P_minus, K, x_hat, P = KalmanFilter(F[i], Q[i], H[i], R[i], z, xi
    [i], L[i])
            S, x_tilde, P_tilde = KalmanSmoother(F[i], x_hat_minus, P_minus, x_hat, P)
            V = Lag1AutoCov(K, S, F[i], H[i], P)

            likelihood = np.append(likelihood, [ell(H[i], R[i], z, x_hat_minus, P_minus)],
     axis = 0)

            #Convergence check for likelihood
            convergence_count = 0
            if i >= 1 and likelihood[i+1] - likelihood[i] < tol_likelihood:
                convergence_count += 1

            #M-step
            A_1 = np.zeros((n, n))
            A_2 = np.zeros((n, n))
            A_3 = np.zeros((n, n))
            A_4 = np.zeros((p, n))

            for j in range(T):
                A_1 += np.outer(x_tilde[j+1], x_tilde[j]) + V[j]
                A_2 += np.outer(x_tilde[j], x_tilde[j]) + P_tilde[j]
                A_3 += np.outer(x_tilde[j+1], x_tilde[j+1]) + P_tilde[j+1]
                A_4 += np.outer(z[j], x_tilde[j+1])

            if "F" in em_vars:
                #Update equation for F
                F = np.append(F, [A_1 @ np.linalg.inv(A_2)], 0)

                #Convergence check for F
                if i >= 1 and np.all(np.abs(F[i+1] - F[i]) < tol_params):
                    convergence_count += 1
            else:
                F = np.append(F, [F_0], 0)

            if "Q" in em_vars:
                #Update equation for Q
                if "F" in em_vars:
                    Q_i = (A_3 - F[i+1] @ A_1.T)/T
                else:
                    Q_i = (A_3 - A_1 @ np.linalg.inv(A_2) @ A_1.T)/T

                #Check whether the updated estimate for Q is positive definite
                if is_pos_def(Q_i) == False:
                    print(f"Q NON-SPD at iteration {i}")
                    Q_i = force_SPD(Q_i)

                Q = np.append(Q, [Q_i], 0)

                #Convergence check for Q
                if i >= 1 and np.all(np.abs(Q[i+1] - Q[i]) < tol_params):
                    convergence_count += 1
            else:
                Q = np.append(Q, [Q_0], 0)

            if "H" in em_vars:
                #Update equation for H
                H = np.append(H, [A_4 @ np.linalg.inv(A_3)], 0)

                #Convergence check for H
                if i >= 1 and np.all(np.abs(H[i+1] - H[i]) < tol_params):
                    convergence_count += 1
            else:
                H = np.append(H, [H_0], 0)

            if "R" in em_vars:
                #Update equation for R
                if "H" in em_vars:
                    R_i = (A_5 - H[i+1] @ A_4.T)/T
                else:
                    R_i = (A_5 - A_4 @ np.linalg.inv(A_3) @ A_4.T)/T
                #Check whether the updated estimate of R is positive definite
                if is_pos_def(R_i) == False:
                    print(f"R_{i} NON-SPD")
                R_i = force_SPD(R_i)

                R = np.append(R, [R_i], axis = 0)

                #Convergence check for R
                if i >= 1 and np.all(np.abs(R[i+1] - R[i]) < tol_params):
                    convergence_count += 1
            else:
                R = np.append(R, [R_0], 0)

            if "xi" in em_vars:
                #Update equation for xi
                xi = np.append(xi, [x_tilde[0]], axis = 0)

                #Convergence check for xi
                if i >= 1 and np.all(np.abs(xi[i+1] - xi[i]) < tol_params):
                    convergence_count += 1
            else:
                xi = np.append(xi, [xi_0], 0)

            if "L" in em_vars:
                #Update equation for Lambda
                L = np.append(L, [P_tilde[0]], axis = 0)

                #Convergence check for Lambda
                if i >= 1 and np.all(np.abs(L[i+1] - L[i]) < tol_params):
                    convergence_count += 1
            else:
                L = np.append(L, [L_0], axis = 0)

            if convergence_count == len(em_vars) + 1:
                break

        iterations = i + 1

    #One-dimensional measurements case
    elif n > 1 and p == 1:
        A_5 = 0
        for j in range(T):
            A_5 += z[j] ** 2

        for i in range(max_it):
            if i > 0 and i % 50 == 0:
                print(f"Iteration {i}")
            # E-step
            x_hat_minus, P_minus, K, x_hat, P = KalmanFilter(F[i], Q[i], H[i], R[i], z, xi[i], L[i])
            S, x_tilde, P_tilde = KalmanSmoother(F[i], x_hat_minus, P_minus, x_hat, P)
            V = Lag1AutoCov(K, S, F[i], H[i], P)

            likelihood = np.append(likelihood, [ell(H[i], R[i], z, x_hat_minus, P_minus)], axis = 0)

            convergence_count = 0
            if i >= 1 and likelihood[i+1] - likelihood[i] < tol_likelihood:
                convergence_count += 1

            # M-step
            A_1 = np.zeros((n, n))
            A_2 = np.zeros((n, n))
            A_3 = np.zeros((n, n))
```

```python
            A_4 = np.zeros((p, n))

            for j in range(T):
                A_1 += np.outer(x_tilde[j+1], x_tilde[j]) + V[j]
                A_2 += np.outer(x_tilde[j], x_tilde[j]) + P_tilde[j]
                A_3 += np.outer(x_tilde[j+1], x_tilde[j+1]) + P_tilde[j+1]
                A_4 += z[j] * x_tilde[j+1]

            if "F" in em_vars:
                F = np.append(F, [A_1 @ np.linalg.inv(A_2)], 0)

                if i >= 1 and np.all(np.abs(F[i+1] - F[i]) < tol_params):
                    convergence_count += 1
            else:
                F = np.append(F, [F_0], 0)

            if "Q" in em_vars:
                if "F" in em_vars:
                    Q_i = (A_3 - F[i+1] @ A_1.T)/T
                else:
                    Q_i = (A_3 - A_1 @ np.linalg.inv(A_2) @ A_1.T)/T
                if is_pos_def(Q_i) == False:
                    print(f"Q_{i} NON-SPD")
                Q_i = force_SPD(Q_i)

                Q = np.append(Q, [Q_i], 0)

                if i >= 1 and np.all(np.abs(Q[i+1] - Q[i]) < tol_params):
                    convergence_count += 1
            else:
                Q = np.append(Q, [Q_0], 0)

            if "H" in em_vars:
                H = np.append(H, [A_4 @ np.linalg.inv(A_3)], 0)

                if i >= 1 and np.all(np.abs(H[i+1] - H[i]) < tol_params):
                    convergence_count += 1
            else:
                H = np.append(H, [H_0], 0)

            if "R" in em_vars:
                if "H" in em_vars:
                    R_i = float((A_5 - H[i+1] @ A_4.T)/T)
                else:
                    R_i = float((A_5 - A_4 @ np.linalg.inv(A_3) @ A_4.T)/T)

                R = np.append(R, [R_i], axis = 0)
                if i >= 1 and np.abs(R[i+1] - R[i]) < tol_params:
                    convergence_count += 1
            else:
                R = np.append(R, [R_0], 0)

            if "xi" in em_vars:
                xi = np.append(xi, [x_tilde[0]], axis = 0)

                if i >= 1 and np.all(np.abs(xi[i+1] - xi[i]) < tol_params):
                    convergence_count += 1
            else:
                xi = np.append(xi, [xi_0], 0)

            if "L" in em_vars:
                L = np.append(L, [P_tilde[0]], axis = 0)

                if i >= 1 and np.all(np.abs(L[i+1] - L[i]) < tol_params):
                    convergence_count += 1
            else:
                L = np.append(L, [L_0], axis = 0)

            if convergence_count == len(em_vars) + 1:
                break

        iterations = i + 1

    #One-dimensional case
    else:
        A_5 = 0
        for j in range(T):
            A_5 += z[j] ** 2

        for i in range(max_it):
            if i > 0 and i % 50 == 0:
                print(f"Iteration {i}")
            #E-step
            x_hat_minus, P_minus, K, x_hat, P = KalmanFilter(F[i], Q[i], H[i], R[i], z, xi
    [i], L[i])
            S, x_tilde, P_tilde = KalmanSmoother(F[i], x_hat_minus, P_minus, x_hat, P)
            V = Lag1AutoCov(K, S, F[i], H[i], P)

            likelihood = np.append(likelihood, [ell(H[i], R[i], z, x_hat_minus, P_minus)],
     axis = 0)

            convergence_count = 0
            if i >= 1 and likelihood[i+1] - likelihood[i] < tol_likelihood:
                convergence_count += 1

           #M-step
            A_1 = 0
            A_2 = 0
            A_3 = 0
            A_4 = 0

            for j in range(T):
                A_1 += x_tilde[j+1] * x_tilde[j] + V[j]
                A_2 += x_tilde[j] ** 2 + P_tilde[j]
                A_3 += x_tilde[j+1] ** 2 + P_tilde[j+1]
                A_4 += z[j] * x_tilde[j+1]

            if "F" in em_vars:
                F = np.append(F, [A_1 / A_2], 0)

                if i >= 1 and np.abs(F[i+1] - F[i]) < tol_params:
                    convergence_count += 1
            else:
                F = np.append(F, [F_0], 0)

            if "Q" in em_vars:
                if "F" in em_vars:
                    Q_i = (A_3 - F[i+1] * A_1)/T
                else:
                    Q_i = (A_3 - A_1 ** 2 / A_2)/T

                Q = np.append(Q, [Q_i], 0)

                if i >= 1 and np.abs(Q[i+1] - Q[i]) < tol_params:
                    convergence_count += 1
            else:
                Q = np.append(Q, [Q_0], 0)

            if "H" in em_vars:
                H = np.append(H, [A_4 / A_3], 0)

                if i >= 1 and np.abs(H[i+1] - H[i]) < tol_params:
                    convergence_count += 1
            else:
                H = np.append(H, [H_0], 0)

            if "R" in em_vars:
                if "H" in em_vars:
                    R_i = (A_5 - H[i+1] * A_4)/T
                else:
                    R_i = (A_5 - A_4 ** 2 / A_3)/T

                R = np.append(R, [R_i], axis = 0)
                if i >= 1 and np.all(np.abs(R[i+1] - R[i]) < tol_params):
                    convergence_count += 1
            else:
                R = np.append(R, [R_0], 0)

            if "xi" in em_vars:
                xi = np.append(xi, [x_tilde[0]], axis = 0)

                if i >= 1 and np.abs(xi[i+1] - xi[i]) < tol_params:
                    convergence_count += 1
            else:
                xi = np.append(xi, [xi_0], 0)

            if "L" in em_vars:
                L = np.append(L, [P_tilde[0]], axis = 0)

                if i >= 1 and np.abs(L[i+1] - L[i]) < tol_params:
                    convergence_count += 1
            else:
                L = np.append(L, [L_0], axis = 0)

            if convergence_count == len(em_vars) + 1:
                break

        iterations = i + 1

    likelihood = np.delete(likelihood, 0, axis = 0)

    return(F, Q, H, R, xi, L, likelihood, iterations)

def grad(F, Q, H, R, xi, L, z, x, P, V, em_vars = ["F", "Q", "H", "R", "xi", "L"]):
    T = len(z)
    n = len(x[0])
    p = len(z[0])

    A_1 = np.zeros((n, n))
    A_2 = np.zeros((n, n))
    A_3 = np.zeros((n, n))
    A_4 = np.zeros((p, n))
    A_5 = np.zeros((p, p))

    for j in range(T):
        A_1 += np.outer(x[j+1], x[j]) + V[j]
        A_2 += np.outer(x[j], x[j]) + P[j]
        A_3 += np.outer(x[j+1], x[j+1]) + P[j+1]
        A_4 += np.outer(z[j], x[j+1])
        A_5 += np.outer(z[j], z[j])

    gradient = np.empty(1)

    if "F" in em_vars:
        gradient = np.append(gradient, np.ndarray.flatten(A_1 - F @ A_2, order = 'F'),
    axis = 0)

    if "Q" in em_vars:
        gradient = np.append(gradient, np.ndarray.flatten((T * Q.T - A_3 + 2 * A_1 @ F.T -
     F @ A_2 @ F.T)/2, order = 'F'), axis = 0)

    if "H" in em_vars:
        gradient = np.append(gradient, np.ndarray.flatten(A_4 - H @ A_3, order = 'F'),
    axis = 0)

    if "R" in em_vars:
        gradient = np.append(gradient, np.ndarray.flatten((T * R.T - A_5 + 2 * A_4 @ H.T - H @ A_3 @ H.T)/2, order = 'F'), axis = 0)

    if "xi" in em_vars:
        gradient = np.append(gradient, np.ndarray.flatten((x[0] - xi).T , order = 'F'), axis = 0)

    if "L" in em_vars:
        gradient = np.append(gradient, np.ndarray.flatten(np.outer(x[0] - xi, x[0] - xi) + P[0] , order = 'F'), axis = 0)

    gradient = np.delete(gradient, 0, axis = 0)
    return(gradient)

def HessianApprox(F, Q, H, R, xi, L, z, x, P, V, em_vars = ["F", "Q", "H", "R", "xi", "L"], shift = 0.5, scale = 1000, MC_size = 10000):
    T = len(z)
    n = len(x[0])
    p = len(z[0])

    gen = np.random.default_rng(seed=None)

    d = 0
    for var in em_vars:
        d += np.prod(locals()[var].shape)

    Hessian = np.empty((1, d, d))

    for i in range(MC_size):
        if i % 100 == 0:
            print(f"Simulation {i}")
        delta = np.empty(1)

        if "F" in em_vars:
            delta = np.append(delta, (gen.binomial(1, 0.5, n ** 2) - shift) / scale, axis = 0)
            F_per_plus = F + np.reshape(delta[len(delta) - n ** 2:], (n, n), order = 'F')
            F_per_minus = F - np.reshape(delta[len(delta) - n ** 2:], (n, n), order = 'F')
        else:
            F_per_plus = F
            F_per_minus = F

        if "Q" in em_vars:
            delta = np.append(delta, (gen.binomial(1, 0.5, n ** 2) - shift) / scale, axis = 0)
            Q_per_plus = Q + np.reshape(delta[len(delta) - n ** 2:], (n, n), order = 'F')
            Q_per_minus = Q - np.reshape(delta[len(delta) - n ** 2:], (n, n), order = 'F')
        else:
            Q_per_plus = Q
            Q_per_minus = Q

        if "H" in em_vars:
            delta = np.append(delta, (gen.binomial(1, 0.5, p * n) - shift) / scale, axis = 0)
            H_per_plus = H + np.reshape(delta[len(delta) - p * n:], (p, n), order = 'F')
            H_per_minus = H - np.reshape(delta[len(delta) - p * n:], (p, n), order = 'F')
        else:
            H_per_plus = H
            H_per_minus = H

        if "R" in em_vars:
            delta = np.append(delta, (gen.binomial(1, 0.5, p ** 2) - shift) / scale, axis = 0)
            R_per_plus = R + np.reshape(delta[len(delta) - p ** 2:], (p, p), order = 'F')
            R_per_minus = R - np.reshape(delta[len(delta) - p ** 2:], (p, p), order = 'F')
        else:
            R_per_plus = R
            R_per_minus = R

        if "xi" in em_vars:
            delta = np.append(delta, (gen.binomial(1, 0.5, n) - shift) / scale, axis = 0)
            xi_per_plus = xi + np.reshape(delta[len(delta) - n:], n, order = 'F')
            xi_per_minus = xi - np.reshape(delta[len(delta) - n:], n, order = 'F')
        else:
            xi_per_plus = xi
            xi_per_minus = xi

        if "L" in em_vars:
            delta = np.append(delta, (gen.binomial(1, 0.5, n ** 2) - shift) / scale, axis = 0)
            L_per_plus = L + np.reshape(delta[len(delta) - n ** 2:], (n, n), order = 'F')
            L_per_minus = L - np.reshape(delta[len(delta) - n ** 2:], (n, n), order = 'F')
        else:
            L_per_plus = L
            L_per_minus = L

        delta = np.delete(delta, 0, axis = 0)

        grad_per_plus = grad(F_per_plus, Q_per_plus, H_per_plus, R_per_plus, xi_per_plus, L_per_plus, z, x, P, V, em_vars)
        grad_per_minus = grad(F_per_minus, Q_per_minus, H_per_minus, R_per_minus, xi_per_minus, L_per_minus, z, x, P, V, em_vars)
        delta_inv = 1/delta

        grad_diff = grad_per_plus - grad_per_minus
        Hessian = np.append(Hessian, [0.5 * (np.outer(grad_diff/2, delta_inv) + np.outer(grad_diff/2, delta_inv).T)], axis = 0)

    Hessian = np.delete(Hessian, 0, axis = 0)

    return(np.mean(Hessian, axis=0))
```