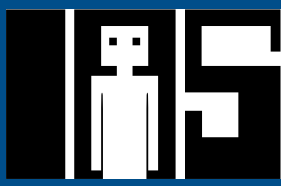
Variational Autoencoders for Koopman Dynamical Systems

Variational Autoencoder für Koopman-Dynamische Systeme
Bachelor thesis by Fabian Damken
Date of submission: November 20, 2020

1. Review: Jan Peters
2. Review: Joe Watson
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT



Variational Autoencoders for Koopman Dynamical Systems
Variational Autoencoder für Koopman-Dynamische Systeme

Bachelor thesis by Fabian Damken

1. Review: Jan Peters
2. Review: Joe Watson

Date of submission: November 20, 2020

Darmstadt

Abstract

In control theory, we try to control a system to reach a specific goal, for example for a pendulum to stand upright. For linear dynamical systems, highly evolved control methods like the linear quadratic regulator exist. In the case of nonlinear systems, however, we do not have such control methods. This is the reason why linearization of a nonlinear system is quite appealing as it offers a way for us to apply methods of linear control theory to nonlinear systems. Typical linearization approaches like small angle approximation have the disadvantage of only being valid locally. Koopman theory offers a way to linearize a system globally by mapping the nonlinear behavior to an infinite-dimensional embedding using nonlinear observation functions. In this embedding, the dynamics behave linearly with an infinite-dimensional operator called the *Koopman operator*, advancing the observations forward in time.

Prior work has studied the Koopman operator and how to approximate it using a finite-dimensional linear embedding and to find the Koopman eigenfunctions. These functions only scale when applying the Koopman operator to it, spanning an invariant subspace of the embedding. Most of the prior work employed a deterministic view on the Koopman operator which makes gauging the uncertainty of the system impossible.

In this thesis we present an algorithm based on the theory of Linear Gaussian Dynamical Systems by replacing the linear with nonlinear observations. This algorithm, the *Koopman inference* algorithm, is an Expectation-Maximization algorithm that alternates between estimating the linear latent state, and optimizing the latent state dynamics and the observation function. With this approach the algorithm will be able to approximate the dynamics of nonlinear systems using a finite-dimensional linear embedding.

We can show that our algorithm performs decently on common nonlinear environments like a pendulum, successfully finding an embedding where the dynamics behave linearly.

Acknowledgments

First of all I would like to thank my supervisor Joe Watson who not only introduced the whole Koopman theory to me and had the idea for this topic, but who always provided strong and invaluable feedback. I would also like to thank him for impressively quick responses to any question I had.

Furthermore I would like to thank Jan Peters, who first aroused my interest in robotics and machine learning.

I want to thank my fellow student Claas for always being there when I had any question and providing feedback from the beginning. I also want to express my appreciation for many helpful discussions in the past.

I want to thank Claas, Heiko, Stefanie and Tim for proofreading this thesis and providing feedback to the last minute. I also want to thank my parents for their support especially throughout my studies and the time of writing this thesis.

Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Fabian Damken, die vorliegende Bachelorarbeit gemäß §22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, November 20, 2020

Fabian Damken

Contents

Algorithms, Figures, Listings and Tables

List of Algorithms

List of Figures

List of Listings

List of Tables

Abbreviations, Operators and Symbols

List of Abbreviations

Notation	Description
AEVB	Auto-Encoding Variational Bayes
CPU	Central Processing Unit
DMD	Dynamic Mode Decomposition
DVK	Deep Variational Koopman
eDMD	extended Dynamic Mode Decomposition
e.g.	for example
EKF	Extended Kalman Filter
ELBO	Evidence Lower Bound
EM	Expectation-Maximization
GD	Gradient Descent
GPU	Graphics Processing Unit
HMM	Hidden Markov Model
i.e.	that is
i.i.d.	independently and identically distributed
iLQR	iterative Linear Quadratic Regulator
IVP	Initial Value Problem
KL	Kullback-Leibler
LGDS	Linear Gaussian Dynamical System
LQG	Linear Quadratic Gaussian
LQR	Linear Quadratic Regulator
LSTM	Long Short-Term Memory
MAP	Maximum A-Posteriori

MBRL	Model-Based Reinforcement Learning
MCMC	Markov Chain Monte Carlo
ML	Maximum Likelihood
MPC	Model-Predictive Control
MSE	Mean Squared Error
NRMSE	Normalized Root Mean Squared Error
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
ReLU	Rectified Linear Unit
RK4	4-th Order Runge-Kutta Method
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
RTS	Rauch-Tung-Striebel
UT	Unscented Transformation
VAE	Variational Auto-Encoder
w.r.t.	with respect to

List of Operators

Notation	**Description**	**Operator**
Cov	Covariance	$\mathrm{Cov}[\cdot]$
SRC	Evaluation of spherical-radical cubature rule of function $\mathbf{f}$ under mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$.	$SRC[\mathbf{f}; \boldsymbol{\mu}, \boldsymbol{\Sigma}]$
$\mathbb{E}$	Expectation value	$\mathbb{E}[\cdot]$
diag	Represents a n-dimensional diagonal matrix with diagonal entries α_i, $i = 1, 2, \cdots, n$.	$\mathrm{diag}(\alpha_1, \alpha_2, \cdots, \alpha_n)$
tr	Trace of a matrix	$\mathrm{tr}(\cdot)$
$\log_k$	Logarithm of base k. If no base is given, the natural logarithm.	$\log_k(\cdot)$

1. Introduction

A dynamical system often describes some kind of physical process, e.g. a pendulum swinging around a fixed point or two pendulums swinging together. However, these systems often exhibit complicated and in some cases even chaotic movement. In (optimal) control theory, we try to control these systems to accomplish a desired state, e.g. to stand upright. As we do not necessarily have the equations of motion for complex systems (e.g. with friction models), we have to learn the model in a process called *model learning*.

Motivation Learning nonlinear models is a tedious task. For optimal control, there are highly evolved methods like Linear Quadratic Regulator (LQR) to get an optimal trajectory. Unfortunately, such solution theories do not exist for nonlinear models, even if we had models that perfectly describe the reality. Most methods used for controlling dynamics systems, e.g. Model-Predictive Control (MPC) and iterative Linear Quadratic Regulator (iLQR), use approximations and locally linear dynamics models.

This motivates globally linear embeddings that we can project the nonlinear dynamics to. This enables doing control in the linear embedding rather than the complex nonlinear state space, yielding better overall results.

Approach Finding such a linear embedding to perform control in is theoretically backed by the Koopman theory [Koo31], stating that every dynamical system can be lifted into a linear embedding. Such methods have proven to be useful in the past [KKB20, HHV20, MWK19], however, most of the approaches do not tackle the problem from a probabilistic view or have very complex models.

Contribution Our contribution is an extension of the well-known model learning algorithm for Linear Gaussian Dynamical Systems (LGDSs) [GH96, Min99] by using nonlinear measurements. Backed by Koopman theory [Koo31] and results from prior work [LKB18], we assume the existence of an observation function that maps from a linear Koopman embedding to the nonlinear state space, allowing model predictions of the complex underlying dynamics using linear transformations.

We evaluate our model on well-known environments like the (damped) pendulum, cartpole and a double pendulum. Finally, we examine the model performance on an error-basis and explore the influence of hyperparameters on the performance. Also, we set our results in relation to results from related work and evaluate them.

Goals Our goal is to provide a compact and easy-to-train model for learning nonlinear dynamical systems in a linear embedding with decent forecasting abilities. This lays the groundwork for future modification in control and Bayesian treatment of the model.

Outline In the chapter “Preliminaries” we give an overview over the basic techniques we use throughout this thesis, namely dynamical systems, the Koopman operator and the expectation-maximization algorithm. Afterwards, we give an introduction to inference in dynamical systems in the chapter “Inference in Dynamical Systems”, laying the groundwork for the following chapters that include our contribution. In chapter “The Koopman Inference Algorithm”, we state the problem to solve formally and propose a method to solve the upcoming inference problem using an approximate nonlinear expectation-maximization algorithm utilizing cubature rules. We will also cover implementation details and assess the numerical stability of our algorithm. We will cover experiments and experimental results in the chapter “Experiments” and discuss the results in the chapter “Discussion”, giving also some propositions for future work. Finally, we summarize all results in the chapter “Conclusion”.

2. Related Work

In this chapter we put our work into the context of other papers, most of which use the Koopman operator or a similar approach for learning a linear embedding of nonlinear dynamical systems.

Starting with work from 2018 [LKB18], Lusch et al. utilize a classic autoencoder for finding a linear embedding. They use the Mean Squared Error (MSE) of the output of the decoder with the input to the encoder to force the embedding to be decodeable. In addition to this simple loss function, they add a matrix to approximate the Koopman operator between the two neural networks and put a loss function on the outcome compared with the next time step, forcing the embedding to advance linearly. A similar loss function is put on the embedding directly to enforce that not the autoencoder but the Koopman matrix forwards the states in time. They also add another loss function for regularization. All these loss functions are then put together with hand-tuned weights to a single loss function the two neural networks are trained on via gradient descent. Compared to our work, however, they do not impose a probabilistic view on the embedding or the decoded values, making uncertainty-aware control impossible. Another common approach of identifying the Koopman eigenfunctions is using Dynamic Mode Decomposition (DMD) or extended Dynamic Mode Decomposition (eDMD) [BBPK16, KKB20, WKR15]. These approaches seek for a parsimonious representation in the linear embedding rather than reaching for high forecasting and state estimation precision.

An approach closer to ours is from 2019 [MWK19] and also treats the Koopman embedding probabilistically by not modeling the Koopman observations directly but treating the linear embedding as a series of states to be inferred. Additionally, they assume control inputs to be present in the linear embedding in the form of an additive control with a control matrix. Their model uses six neural networks for encoding the dynamics:

- the *decoder network* to model the decoding of the embedding states back to the nonlinear manifold,
- the *temporal encoder network* using a bidirectional Long Short-Term Memory (LSTM) summarizing the time evolution,
- the *initial observation inference network* to infer the initial observation,
- the *observation encoder network* that is a Recurrent Neural Network (RNN) taking the observations and outputting an encoding of the time evolution,
- an *observation inference network* encoding a distribution over the observations given the output of the temporal and observation encoder networks and the *conditional prior network* which outputs the next observation distribution conditioned on the previous one.

Compared to our work, their model is far more complex with more neural networks and way more parameters, possibly leading to higher overfitting. On the other hand, it is possible using (uncertainty-aware) MPC on their model which is (currently) not possible using the Koopman Inference algorithm.

Non-Koopman-related work includes the recurrent Kalman filter [BPG+19] which introduces a new type of RNNs, the recurrent Kalman network. This network learns high-dimensional linear embeddings with factorized covariances, leading to simple Kalman update rules with scalar operations.

3. Preliminaries

In this chapter we will introduce the basics of this work both to clarify notation and to introduce the reader to potentially unknown concepts and methodologies. We will start by having a look at dynamical systems that form the basis of this thesis.

3.1. Dynamical Systems

A *dynamical system* [Bir27] is a (physical) system that evolves over time t and is completely defined by the values of m real variables

$$x_1, x_2, \cdots, x_m \quad \longleftrightarrow \quad \mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_m \end{bmatrix}^T$$

called the *state* and often written in vector form. Given the differentiability of these values, we can also study their rate of change (often referred to as the "velocity") and the rate of change of the velocity (often referred to as the "acceleration"):

$$\dot{\mathbf{x}} = \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} \qquad \ddot{\mathbf{x}} = \frac{\mathrm{d}^2\mathbf{x}}{\mathrm{d}t^2}$$

Describing these systems is possible using differential equations, both ordinary and partial ones. A general k-th order Ordinary Differential Equation (ODE) is given by an implicit equation

$$\mathbf{0} = \mathbf{F}\big(\mathbf{x}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(k-1)}, \mathbf{x}^{(k)}; t\big), \quad \mathbf{x}^{(l)} \coloneqq \frac{\mathrm{d}^l\mathbf{x}}{\mathrm{d}t^l} \tag{3.1}$$

which establishes a connection between the state itself and its time derivatives. We call a function $\mathbf{x}(t)$ a *solution* of an ODE if its derivatives fulfill the given ODE (3.1). In all of the following, we assume to have explicit, autonomous, first order ODEs. This is valid because we can transform every explicit higher order ODE into a system of first order ODEs. Also we can introduce another "clock state" which makes our ODE autonomous (i.e. not explicitly time-dependent).

The solution theory for linear ODEs is highly evolved and solutions exist for nearly every possible ODE. But for nonlinear ODEs, we do not have such solution theories. With the exception of some rare cases, nonlinear ODEs are not tractable. Hence, we often need approximations for the nonlinear case. Some well-known approaches for these approximations are e.g. *small angle approximation* for Sines and Cosines. In small angle approximations, we Taylor-expand $\sin/\cos$ at $\varphi_a = 0$ and cut all higher order terms:

$$\sin(\varphi) = \varphi - \underbrace{\frac{\varphi^3}{3!} + \frac{\varphi^5}{5!} + \cdots}_{\text{higher order terms}} \approx \varphi$$

$$\cos(\varphi) = 1 - \underbrace{\frac{\varphi^2}{2!} + \frac{\varphi^4}{4!} - \frac{\varphi^6}{6!} + \cdots}_{\text{higher order terms}} \approx 1$$

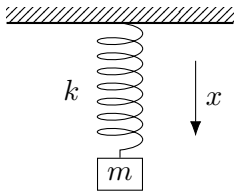


Figure 3.1.: Illustration of a simple harmonic oscillator with mass m, spring stiffness k and position x that is not affected by any external force like gravity. The mass is in equilibrium if $x = 0$.

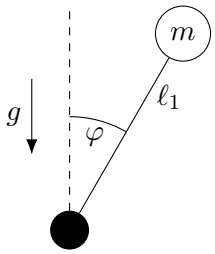


Figure 3.2.: Illustration of an inverse pendulum with mass m and displacement φ that is only affected by gravity and no other external force. The mass is in equilibrium for both $\varphi = 0$ and $\varphi = \pi$, where the former is an unstable equilibrium.

We now look at two examples of dynamical systems, the first of which is linear and second which is nonlinear.

Harmonic Oscillator The *simple harmonic oscillator* is a linear system that describes the dynamical system of a mass m that is attached to a spring that is following Hooke's Law with stiffness k (see Figure 3.1). This harmonic oscillator is described by the ODE

$$m\ddot{x} = -kx \quad \Longleftrightarrow \quad \ddot{x} = -\frac{k}{m}x \tag{3.2}$$

where x and $\ddot{x}$ are the position and acceleration of the mass, respectively. Note that if $x = 0$, the mass is in equilibrium and no force is acting on it.

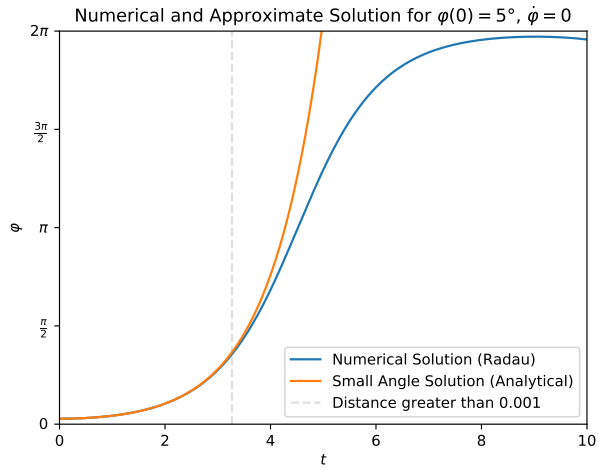
By using basic results in the solution theory of linear ODEs, we see that the general solution is given as

$$x(t) = A\cos\left(t\sqrt{k/m} + \varphi\right)$$

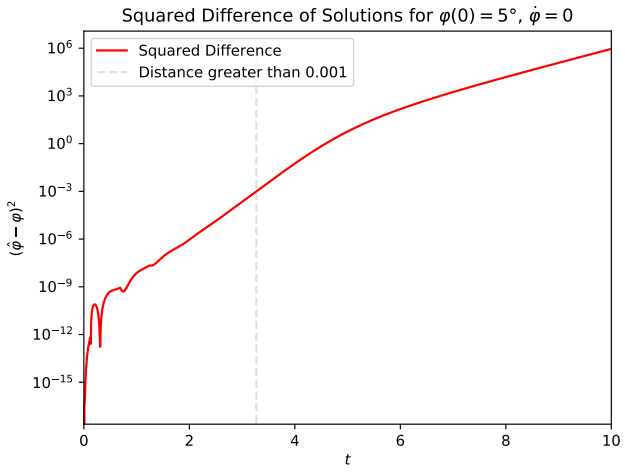
with the amplitude A and the phase φ (see Appendix C for the derivation of the solution). As neither gravity nor damping or other external forces are involved in the dynamical system, the motion continues forever with a non-changing amplitude.

Inverse Pendulum The *inverse pendulum* describes the dynamical system of a mass m that is attached to a rigid pole of length L which can freely swing around a suspension point (see Figure 3.2). The pendulum stands upright if $\varphi = 0$ and its equation of motion is described by the ODE

$$\ddot{\varphi} = \frac{g}{L}\sin(\varphi)$$



(a) Trajectories of two solution strategies to the inverse pendulum, where the blue is a numerical solution of the actual motion of equation (solved using the Radau IIA method [GH01, p. 72]) and the orange one is the analytically computed solution linearized ODE. The latter is linearized using small angle approximation. The dashed gray vertical line shows when the distance tolerance of $\varepsilon = 10^{-3}$ is exceeded.



(b) Differences between the small angle approximation and the numerical solution of the ODE. The dashed gray vertical line shows when the distance tolerance of $\varepsilon = 10^{-3}$ is exceeded.

Figure 3.3.: Comparison of a numerical solution to the ODE of the inverse pendulum given in (3.3) and the analytical solution of the linearized ODE given in (3.4). A tolerance value of $\varepsilon = 10^{-3}$ is used to show when both solutions diverge from each other.

where g, L, φ and $\ddot{\varphi}$ describe the gravity acceleration, pole length, displacement and acceleration of the mass, respectively. Note that if $\varphi = 0$, the mass is in an unstable equilibrium and no force is acting on it.

In comparison to the harmonic oscillator (section 3.1), this differential equation is nonlinear. And, even for the case with unit gravity acceleration $g = 1$ and unit pole length $L = 1$, where the ODE looks really simple

$$\ddot{\varphi} = \sin(\varphi) \tag{3.3}$$

it is not tractable analytically (i.e. there exists no solution in closed form).

Still, we can apply the small angle approximation introduced before (in this case, $\sin(\varphi) \approx \varphi$) which yields the simple ODE

$$\ddot{\varphi} \approx \varphi \tag{3.4}$$

solved by

$$\varphi(t) = \frac{1}{2}e^{-t}\left(\varphi_0 + e^{2t}\varphi_0 - \dot{\varphi}_0 + e^{2t}\dot{\varphi}_0\right)$$

where φ_0 and $\dot{\varphi}_0$ are the initial displacement and velocity, respectively.

However, this small angle approximation can only forecast small displacements $\varphi \ll \pi/2$. And, as the equilibrium at $\varphi = 0$ is unstable, the approximation becomes worse as time goes by because the pendulum falls down. This behavior is shown in Figure 3.3.

3.1.1. Discrete-Time Dynamical Systems

In comparison to continuous-time dynamical systems described by ODEs, discrete-time systems are described by a *dynamics function* $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$ advancing all states forward in time:

$$\mathbf{x}_{t+1} = \mathbf{F}(\mathbf{x}_t)$$

But we should note that, while seeming more restrictive, discrete-time dynamical systems are more general [BBPK16] than continuous-time systems as we can discretize every continuous-time system

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$$

as a discrete-time dynamical system

$$\mathbf{x}_{t+1} = \mathbf{F}(\mathbf{x}_t)$$

using the state dynamics function

$$\mathbf{F}\big(\mathbf{x}(t_0)\big) = \mathbf{x}(t_0 + \Delta_t) = \mathbf{x}(t_0) + \int_{t_0}^{t_0+\Delta_t} \mathbf{f}\big(\mathbf{x}(\tau)\big)\,\mathrm{d}\tau$$

where Δ_t is called the *discretization interval* and $\mathbf{x}_k = \mathbf{x}(k\Delta_t)$.

3.2. Koopman Theory of Dynamical Systems

Our first contribution is based on the need of a linearization technique that generalizes globally. We have seen this need in the previous chapter when looking at simple nonlinear systems and small angle approximations. This brings us directly to Koopman theory, original introduced by B. Koopman [Koo31] in the context of Hamiltonian systems and transformations in Hilbert spaces. Considering a nonlinear discrete-time dynamical system

$$\mathbf{x}_{t+1} = \mathbf{F}(\mathbf{x}_t), \quad \mathbf{F} : \mathbb{R}^k \to \mathbb{R}^k$$

and observations $\mathbf{h} : \mathbb{R}^p \to \mathbb{R}^p$ of this system, i.e. $\mathbf{y}_t \coloneqq \mathbf{h}(\mathbf{x}_t)$, the infinite-dimensional *Koopman operator* $\mathcal{K}$ advances all of the measurements forward in time. This relation can also be expressed as the composition

$$\mathcal{K}\,\mathbf{h} \coloneqq \mathbf{h} \circ \mathbf{F} \qquad \Longleftrightarrow \qquad \mathcal{K}\,\mathbf{h}(\mathbf{x}_t) = \mathbf{h}\big(\mathbf{F}(\mathbf{x}_t)\big) = \mathbf{h}(\mathbf{x}_{t+1})$$

and can be visualized as a transition diagram between the states (see Figure 3.4). This time evolution works for every possible measurement function $\mathbf{h}$ at any point of the system space $\mathbb{R}^k$ [BBPK16]. All possible measurement functions $\mathbf{h}$ span an infinite-dimensional Hilbert space $\mathcal{H}$. A finite set of measurements $\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_p$ that span an invariant subspace $\hat{\mathcal{H}} \subset \mathcal{H}$ can be considered as a basis of that subspace. That is, applying the Koopman operator to a linear combination of these functions keeps them in the subspace:

$$\begin{aligned}\mathbf{h} &= \alpha_1\mathbf{h}_1 + \alpha_2\mathbf{h}_2 + \cdots + \alpha_p\mathbf{g}_p \\ \mathcal{K}\,\mathbf{h} &= \beta_1\mathbf{h}_1 + \beta_2\mathbf{h}_2 + \cdots + \beta_p\mathbf{h}_p\end{aligned}$$

If that is possible, we can restrict the Koopman operator onto $\hat{\mathcal{H}}$. Functions that both span the subspace $\hat{\mathcal{H}}$ and do only scale when applying the Koopman operator, i.e. $\mathcal{K}\,\boldsymbol{\varphi} = \lambda\boldsymbol{\varphi}$ are called *eigenfunctions* of the Koopman operator. Finding these eigenfunctions is extremely desirable as it allows us to get a finite Koopman operator matrix $\mathbf{K}$ globally linearizing the dynamical system $\mathbf{F}$. However, for this thesis, we do not seek the eigenfunctions directly but an approximation of the *inverse* observation function to forecast the dynamical system.

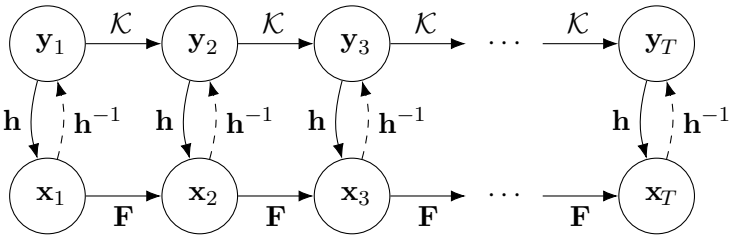


Figure 3.4.: Illustration of the transition diagram imposed by applying the Koopman operator $\mathcal{K}$ to system dynamics $\mathbf{F}$. The observation function $\mathbf{h}$ is forwarded in time linearly by the Koopman operator, while the states $\mathbf{x}$ are forwarded nonlinearly. The dashed lines and $\mathbf{h}^{-1}$ indicate that we want to recover the original state from the linear system.
Adopted from [BBPK16].

3.3. The Expectation-Maximization Algorithm

The Expectation-Maximization (EM) algorithm, first introduced by Ceppellini et al. [CSS55] and popularized by Dempster et al. [DLR77] can be used for tackling the following problem: Assuming some model with latent (hidden) states $\mathbf{x}$, observations $\mathbf{y}$ and model parameters $\boldsymbol{\theta}$, we want to maximize the likelihood $p(\mathbf{y} \mid \boldsymbol{\theta})$ w.r.t. the latent states $\mathbf{x}$ and the parameters $\boldsymbol{\theta}$. However, the marginal distribution

$$p(\mathbf{y} \mid \boldsymbol{\theta}) = \int p(\mathbf{x}, \mathbf{y} \mid \boldsymbol{\theta}) \, \mathrm{d}\mathbf{x}$$

is generally intractable. As usual on maximum likelihood approaches, it is useful not to maximize the likelihood directly, but to maximize the log-likelihood

$$\mathcal{L}(\boldsymbol{\theta}) \coloneqq \log p(\mathbf{y} \mid \boldsymbol{\theta}) = \log \int p(\mathbf{x}, \mathbf{y} \mid \boldsymbol{\theta}) \, \mathrm{d}\mathbf{x}$$

instead. This yields the same maximum as the logarithm is a monotonically increasing function. By introducing an auxiliary probability distribution $q(\mathbf{x} \mid \mathbf{y})$ over the latent variables, we can rewrite the marginal and find a lower bound on $\mathcal{L}$ by using Jensen's inequality [Jen06]:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}) &= \log \int p(\mathbf{x}, \mathbf{y} \mid \boldsymbol{\theta}) \, \mathrm{d}\mathbf{x} \\
&= \log \int q(\mathbf{x} \mid \mathbf{y}) \frac{p(\mathbf{x}, \mathbf{y} \mid \boldsymbol{\theta})}{q(\mathbf{x} \mid \mathbf{y})} \, \mathrm{d}\mathbf{x} \\
&\geq \int q(\mathbf{x} \mid \mathbf{y}) \log \frac{p(\mathbf{x}, \mathbf{y} \mid \boldsymbol{\theta})}{q(\mathbf{x} \mid \mathbf{y})} \, \mathrm{d}\mathbf{x} \\
&= \int q(\mathbf{x} \mid \mathbf{y}) \log p(\mathbf{x}, \mathbf{y} \mid \boldsymbol{\theta}) \, \mathrm{d}\mathbf{x} - \int q(\mathbf{x} \mid \mathbf{y}) \log q(\mathbf{x} \mid \mathbf{y}) \, \mathrm{d}\mathbf{x} =: \mathcal{L}_{\mathrm{EM}}[q, \boldsymbol{\theta}]
\end{aligned} \tag{3.5}$$

This draws a lower bound $\mathcal{L}_{\mathrm{EM}}[q, \boldsymbol{\theta}]$ on the log-likelihood $\mathcal{L}(\boldsymbol{\theta})$ we can maximize instead. Doing this simultaneously increases the log-likelihood. Note that this lower bound is in fact a functional of the distribution $q(\mathbf{x} \mid \mathbf{y})$.

The EM algorithm now iteratively maximizes the lower bound and thus indirectly maximizes the original objective, the likelihood $p(\mathbf{y} \mid \boldsymbol{\theta})$. The E and M step are as follows:

E-Step Infers the auxiliary distribution $q(\mathbf{x} \mid \mathbf{y})$ using the current estimations of the parameters $\boldsymbol{\theta}$ by maximizing the lower bound with respect to (w.r.t.) the auxiliary distribution.

M-Step Infers the parameters $\boldsymbol{\theta}$ using the current auxiliary distribution by maximizing the lower bound w.r.t. the parameters.

Expressed in equations with the index $\cdot^{(n)}$ to denote the values of the n-th iteration, we get the procedure which will be repeated until convergence:

$$\textbf{E-Step} \quad q^{(n+1)} = \arg\max_{q} \mathcal{L}_{\mathrm{EM}}\left[q, \boldsymbol{\theta}^{(n)}\right]$$

$$\textbf{M-Step} \quad \boldsymbol{\theta}^{(n+1)} = \arg\max_{\boldsymbol{\theta}} \mathcal{L}_{\mathrm{EM}}\left[q^{(n+1)}, \boldsymbol{\theta}\right]$$

Additionally, the E-step has the constraint that $q(\mathbf{x} \mid \mathbf{y})$ really is a probability distribution, so it must integrate to one:

$$\int q(\mathbf{x} \mid \mathbf{y}) \, \mathrm{d}\mathbf{x} = 1$$

We can incorporate this into the maximization, e.g. by using Lagrange multipliers. Using a bit of variational calculus, it can be shown [Bea03] that the maximization is gained by choosing the auxiliary distribution $q(\mathbf{x} \mid \mathbf{y})$ to be the same as the distribution $p(\mathbf{x} \mid \mathbf{y}, \boldsymbol{\theta})$. That is, we set $q^{(n+1)}(\mathbf{x} \mid \mathbf{y}) = p\big(\mathbf{x} \,\big|\, \mathbf{y}, \boldsymbol{\theta}^{(n)}\big)$ while holding the parameters $\boldsymbol{\theta}$ fixed. This maximization turns the inequality (3.5) into an equality.

For the M-step, we keep the auxiliary distribution fixed and maximize the lower bound w.r.t. the parameters $\boldsymbol{\theta}$. As this involves taking the derivative of $\mathcal{L}_{\mathrm{EM}}$ w.r.t. $\boldsymbol{\theta}$, we can safely omit the right hand side of the lower bound as it does not depend on $\boldsymbol{\theta}$. Hence, we get the M-step as:

$$\begin{aligned}
\boldsymbol{\theta}^{(n+1)} &= \arg\max_{\boldsymbol{\theta}} \mathcal{L}_{\mathrm{EM}}\left[q^{(n)}, \boldsymbol{\theta}\right] \\
&= \arg\max_{\boldsymbol{\theta}} \int q^{(n)}(\mathbf{x} \mid \mathbf{y}) \log p(\mathbf{x}, \mathbf{y} \mid \boldsymbol{\theta}) \, \mathrm{d}\mathbf{x} \\
&= \arg\max_{\boldsymbol{\theta}} \int p\big(\mathbf{x} \,\big|\, \mathbf{y}, \boldsymbol{\theta}^{(n)}\big) \log p(\mathbf{x}, \mathbf{y} \mid \boldsymbol{\theta}) \, \mathrm{d}\mathbf{x} \\
&= \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{p(\mathbf{x} \mid \mathbf{y}, \boldsymbol{\theta}^{(n)})}[\log p(\mathbf{x}, \mathbf{y} \mid \boldsymbol{\theta})]
\end{aligned}$$

The quantity to optimize, $Q(\boldsymbol{\theta}) \coloneqq \mathbb{E}[\log p(\mathbf{x}, \mathbf{y} \mid \boldsymbol{\theta})]$, is also called the *expected complete log-likelihood* as it involves both the observables and the latent variables.

As shown in [Bea03], the lower bound turns into an equality after the E-step. Hence, we are guaranteed to always rise the log-likelihood after each EM iteration if we do not already have the optimal auxiliary distribution and parameters. If this would be the case, both the E- and the M-step would not change anything, so our log-likelihood is monotonically increasing. Also, we might not want to calculate the whole distribution in the E-step, but we might want to restrict our computations to sufficient statistics that cover our whole distribution. For the rest of this thesis, we know our distribution $p(\mathbf{x} \mid \mathbf{y}, \boldsymbol{\theta})$ is Gaussian, so we can assume the auxiliary distribution to be Gaussian too, given that we set them equal. Hence, we only need to calculate the mean and correlation or covariance of each variable to cover the whole distribution and to be able to proceed.

3.4. Variational Autoencoders and the Evidence Lower Bound

Variational autoencoders have been first introduced in the context of the Auto-Encoding Variational Bayes (AEVB) algorithm by Kingma and Welling [KW14]. They tackle inference and learning in probabilistic models with latent variables, similar to the EM algorithm. To keep the notation analogous to the derivation of the EM algorithm, we deviate from the notation used in [KW14] in terms that we keep $\mathbf{x}$ as our latent variables and $\mathbf{y}$ as the observables.

To perform inference, we want to maximize the (log-)likelihood

$$\mathcal{L}(\boldsymbol{\theta}) \coloneqq \log p_{\boldsymbol{\theta}}(\mathbf{y}) = \log \int p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y}) \, \mathrm{d}\mathbf{x}$$

by maximizing the Evidence Lower Bound (ELBO) $\mathcal{L}_{\mathrm{ELBO}}$ which we can derive by using Jensen's inequality [Jen06]:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}) &= \log \int p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y}) \, \mathrm{d}\mathbf{x} \\
&= \log \int q(\mathbf{x} \,|\, \mathbf{y}, \boldsymbol{\phi}) \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y})}{q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{y})} \, \mathrm{d}\mathbf{x} \\
&\geq \int q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{y}) \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y})}{q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{y})} \, \mathrm{d}\mathbf{x} \\
&= \int q(\mathbf{x} \,|\, \mathbf{y}, \boldsymbol{\phi}) \log \frac{p_{\boldsymbol{\theta}}(\mathbf{y} \,|\, \mathbf{x}) p_{\boldsymbol{\theta}}(\mathbf{x})}{q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{y})} \, \mathrm{d}\mathbf{x} \\
&= \int q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{y}) \frac{p_{\boldsymbol{\theta}}(\mathbf{x})}{q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{y})} \, \mathrm{d}\mathbf{x} + \int q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{y}) p_{\boldsymbol{\theta}}(\mathbf{y} \,|\, \mathbf{x}) \, \mathrm{d}\mathbf{x} \\
&= -D_{\mathrm{KL}}\big(q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{y}) \,\|\, p_{\boldsymbol{\theta}}(\mathbf{x})\big) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{y})}[p_{\boldsymbol{\theta}}(\mathbf{y} \,|\, \mathbf{x})] =: \mathcal{L}_{\mathrm{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\psi})
\end{aligned} \tag{3.6}$$

Note that we have introduced an auxiliary parametric distribution $q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{y})$ which originally leads to the first integral to be an expectation and makes the application of Jensen's inequality possible. We now maximize the ELBO (3.6) w.r.t. the variational and generative parameters $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ respectively using the *reparametrization trick* [KW14].

Shifting to Variational Auto-Encoders (VAEs), we now represent the auxiliary distribution $q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{y})$ using a neural network with some prior $p_{\boldsymbol{\theta}}(\mathbf{x})$, e.g. a standard Gaussian $p_{\boldsymbol{\theta}}(\mathbf{x}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ to keep the latents "close to the center", enforced by the Kullback-Leibler (KL) divergences in the ELBO. For a Gaussian latent distribution, the neural network, called the *amortization network*, produces the mean and the diagonal covariance of q. A second neural network is used for decoding the latent dimension, representing the decoding distribution $p_{\boldsymbol{\theta}}(\mathbf{y} \,|\, \mathbf{x})$. If we assume a Gaussian decoding distribution with constant variance, the right side of the ELBO (3.6) just becomes the squared error between the decoding mean and the input $\mathbf{y}$. Such a network architecture is illustrated in Figure 3.5.

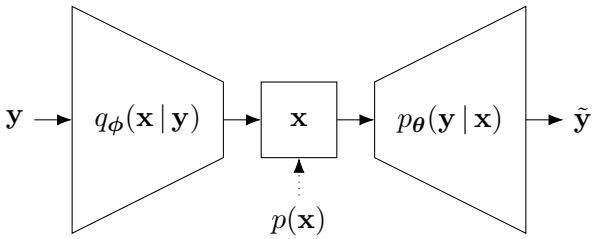


Figure 3.5.: Illustration of a Variational Auto-Encoder with the amortization network on the left and the decoder network on the right. The dotted line represents the influence of the prior to the latent representation and the solid lines represent where the data flow, where $\mathbf{y}$ is the input and $\tilde{\mathbf{y}}$ is the (reconstructed) output.

3.4.1. Connection between EM and VAEs

As we have seen, the log-likelihood $\mathcal{L}(\boldsymbol{\theta})$ gives rise to a lower bound $\mathcal{L}_{\mathrm{EM}}$ (3.5) and the ELBO $\mathcal{L}_{\mathrm{ELBO}}$ (3.6):

$$\mathcal{L}_{\mathrm{EM}}[q, \boldsymbol{\theta}] = \int q(\mathbf{x} \mid \mathbf{y}) \log p(\mathbf{x}, \mathbf{y} \mid \boldsymbol{\theta}) \, \mathrm{d}\mathbf{x} - \int q(\mathbf{x} \mid \mathbf{y}) \log q(\mathbf{x} \mid \mathbf{y}) \, \mathrm{d}\mathbf{x}$$
$$\mathcal{L}_{\mathrm{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\psi}) = -D_{\mathrm{KL}}\big(q_{\boldsymbol{\phi}}(\mathbf{x} \mid \mathbf{y}) \,\|\, p_{\boldsymbol{\theta}}(\mathbf{x})\big) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{x} \mid \mathbf{y})}[p_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x})]$$

We saw that the two concepts are similar to each other, but the maximization procedure differs in the following way:

- In the EM algorithm, we separately maximize the components of the lower bound, firstly finding the next auxiliary distribution q and then maximizing the lower bound w.r.t. the parameters.
- In VAEs, we use an amortization network to model the auxiliary distribution q in a parameterized way. Then we maximize the lower bound w.r.t. to both the variational and the generative parameters at once.

An obvious advantage of the EM algorithm is that we are guaranteed to always rise our lower bound and we will never get worse. Also the EM algorithm requires less computation and has less parameters, depending on the model choices.

4. Inference in Dynamical Systems

Our second contribution is a probabilistic perspective on “classical” Koopman theory as presented in various publications [BBPK16, HHV20, KKB20, LKB18, WKR15]. In this chapter we will build the groundwork for variational inference to combine them to the single Koopman Inference algorithm presented in chapter 5.

4.1. Hidden Markov Models and LGDS

Hidden Markov Models (HMMs) are simple Bayesian networks described by a non-observable (*hidden*) Markov chain. This non-observable discrete chain can be indirectly observed with observations that are emitted by every state. One of the key features of a Markov chain is that a state does only depend on the previous state, but not states before that. That is, knowing only the previous state is sufficient and no more information can be gathered by knowing every other state coming before. This is described by the state transition distribution

$$s_{k+1} \sim p(s_{k+1} \mid s_k)$$

being only dependent on the previous state s_k. Analogous, a observation $\mathbf{y}_k$ of a state s_k is only dependent on that specific state:

$$\mathbf{y}_k \sim p(\mathbf{y}_k \mid s_k)$$

These assumptions are called the *Markov property* and a system fulfilling this property is called *Markovian*. These conditional distributions can be written in a graphical model as shown in Figure 4.1. Note that, in general, no assumption has to be made on the “type” of state/observation (i.e. whether it is a scalar, a vector or something completely different). Also, no assumption is made on the specific transition distributions, HMMs can also be used to model deterministic transitions using a Dirac delta distribution.

A set of closely related dynamics models are LGDSs which are described by a noisy state transition

$$\mathbf{s}_{t+1} = \mathbf{A}\mathbf{s}_t + \mathbf{w}_t, \quad \mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$$

with covariance matrix $\mathbf{Q}$. Similarly, the observations $\mathbf{y}_t = \mathbf{C}\mathbf{s}_t + \mathbf{v}_t$ underlie a Gaussian noise $\mathbf{v}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ too. Together they form the probabilistic model

$$\begin{aligned} \mathbf{s}_{t+1} &= p(\mathbf{A}\mathbf{s}_t, \mathbf{Q}) \\ \mathbf{y}_t &= p(\mathbf{C}\mathbf{s}_t, \mathbf{R}) \end{aligned}$$

which fulfills the Markov property, hence the system is Markovian:

$$p(\mathbf{s}_{1:T}, \mathbf{y}_{1:T}) = p(\mathbf{s}_1) \prod_{t=2}^{T} p(\mathbf{s}_t \mid \mathbf{s}_{t-1}) \prod_{t=1}^{T} p(\mathbf{y}_t \mid \mathbf{s}_t)$$

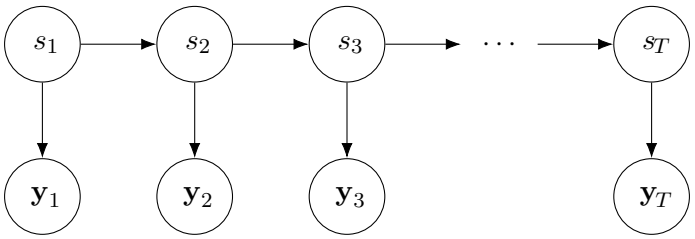


Figure 4.1.: Illustration of a Hidden Markov Model with states s_k and emissions/observations $\mathbf{y}_k$.

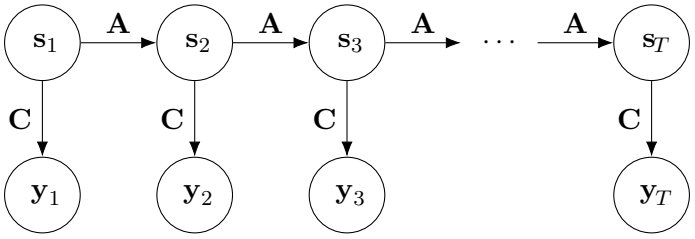


Figure 4.2.: Illustration of a Linear Gaussian Dynamical System with states $\mathbf{x}_t$ and observations $\mathbf{y}_t$. Solid arrows represent probabilistic dependency, where the matrix $\mathbf{A}$ is the dynamics matrix indicating that the mean transitions linearly, as do the observations with the observation matrix $\mathbf{C}$.

Note that we write $\mathbf{s}_{t_0:t_1}$ for the sequence $(\mathbf{s}_{t_0}, \mathbf{s}_{t_0+1}, \cdots, \mathbf{s}_{t_1})$ and analogous for the observation sequence $\mathbf{y}_{t_0:t_1}$. The probabilistic model of a LGDS is shown in Figure 4.2.

This system is similar to a HMM and in fact a LGDS really is an HMM, just with continuous states $\mathbf{s}_t \in \mathbb{R}^k$ and observations $\mathbf{y}_t \in \mathbb{R}^p$.

4.2. Inference: Filtering and Smoothing

When looking at an LGDS, one of the first problems that one might consider is how to get the latent states back? This procedure is called *inference*. In this section we will give an overview on how to perform inference on an LGDS, meaning on a linear system that is Markovian.

For time-series data, the inference procedure is widely know as *filtering* and *smoothing*, where the following distributions on the latent states $\mathbf{s}_t$ are calculated:

- Filtering: $p(\mathbf{s}_t \,|\, \mathbf{y}_{1:t})$
- Smoothing: $p(\mathbf{s}_t \,|\, \mathbf{y}_{1:T})$

That is, the filtered distribution only depends on the observations until that time, where in contrast the smoothed distribution has used all data to the last time step T to estimate the latent states. An intermediate step in the filtering algorithm is called *prediction* where the distribution $p(\mathbf{s}_t \,|\, \mathbf{y}_{1:t-1})$ is being calculated, using only the data to the previous time step. The differences between what data is used in the three steps is shown in Figure 4.3.

We will now take a look on the most used filter, the well-known *Kalman Filter* and the *Rauch-Tung-Striebel Smoother*, which is also sometimes called the *Kalman Smoother* as it requires a run of the Kalman filter first.

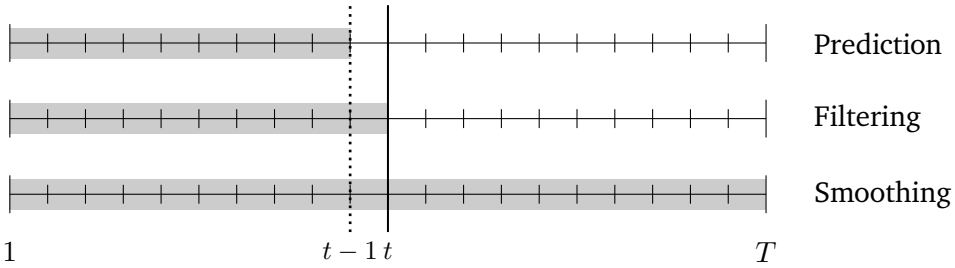


Figure 4.3.: This diagram shows the main differences between prediction, filtering and smoothing. The shaded region describes the observations that have been used to estimate the state at time step t, indicated by a solid line. The dotted line represents time step $t-1$, to which all observations are used in all three procedures.
Adopted from [Sol10].

4.2.1. Kalman Filter

The Kalman filter was originally introduced by R. Kalman in 1960 [Kal60] in the context of signal processing to predict random signals, separate them from noise and detect signals of known form in the presence of noise [Kal60]. For the normal Kalman filter we assume an underlying model of the form of an LGDS where the observations are detected and the state has to be estimated from the data. The process of state estimation is done in a two-step fashion:

1. Prediction: Estimate the distribution $p(\mathbf{s}_t \mid \mathbf{y}_{1:t-1})$
2. Correction: Estimate the distribution $p(\mathbf{s}_t \mid \mathbf{y}_{1:t})$

For a simple linear system with Gaussian noise, i.e. an LGDS, the equations for both the prediction and the correction can be given in closed form and are fairly straightforward. For the prediction step, the equations are given as

$$\hat{\mathbf{s}}_{t|t-1} = \mathbf{A}\hat{\mathbf{s}}_{t-1|t-1} \tag{4.1}$$

$$\hat{\mathbf{V}}_{t|t-1} = \mathbf{A}\hat{\mathbf{V}}_{t-1|t-1}\mathbf{A}^T + \mathbf{Q} \tag{4.2}$$

where $\hat{\mathbf{s}}_{t\,|\,t-1}$ and $\hat{\mathbf{V}}_{t\,|\,t-1}$ are the mean and covariance of the prediction distribution $p(\mathbf{s}_t \mid \mathbf{y}_{1:t-1})$, respectively. The correction step, which integrates new knowledge gained from the observation $\mathbf{y}_t$ into the estimate, is given as

$$\hat{\mathbf{s}}_{t|t} = \hat{\mathbf{s}}_{t|t-1} + \mathbf{K}_t\tilde{\mathbf{y}}_t \tag{4.3}$$

$$\hat{\mathbf{V}}_{t|t} = \hat{\mathbf{V}}_{t|t-1} - \mathbf{K}_t\mathbf{S}_t\mathbf{K}_t^T \tag{4.4}$$

with the auxiliary variables

$$\tilde{\mathbf{y}}_t = \mathbf{y}_t - \mathbf{C}\hat{\mathbf{s}}_{t|t-1} \tag{4.5}$$

$$\mathbf{S}_t = \mathbf{C}\hat{\mathbf{V}}_{t|t-1}\mathbf{C}^T + \mathbf{R} \tag{4.6}$$

$$\mathbf{K}_t = \hat{\mathbf{V}}_{t|t-1}\mathbf{C}^T\mathbf{S}_t^{-1} \tag{4.7}$$

known as the innovation $\tilde{\mathbf{y}}$, residual covariance $\mathbf{S}_t$ and Kalman gain $\mathbf{K}_t$. The resulting estimates $\hat{\mathbf{s}}_{t|t}$ and $\hat{\mathbf{V}}_{t|t}$ then form the mean and covariance of the filtered distribution $p(\mathbf{s}_t \mid \mathbf{y}_{1:t})$ respectively. These equations

form a recursive algorithm starting off with initial state estimates $\hat{\mathbf{s}}_{0|0} = \mathbf{m}_0$ and $\hat{\mathbf{V}}_{0|0} = \mathbf{V}_0$ that have to be determined in externally.

But these equations only work for linear systems. Lots of extensions of the regular Kalman filter have been proposed to extend the filter to nonlinear systems. One of them is the Extended Kalman Filter (EKF) which assumes locally linear dynamics and uses first-order Taylor expansions to linearize them. Throughout this thesis we will use the cubature Kalman filter, which we will introduce later in subsection 4.3.2. We have decided against using the EKF as it is slightly less accurate as the cubature Kalman filter and as we need the Cholesky decompositions of the covariance matrices, which we can estimate directly using a square-root cubature Kalman filter (see subsection 4.3.2).

4.2.2. Rauch-Tung-Striebel / Kalman Smoother

The Rauch-Tung-Striebel (RTS) was been introduced after the original Kalman filter by H. E. Rauch, F. Tung and C. T. Striebel [RTS65] for optimal smoothing of linear systems with Gaussian noise, i.e. LGDS. With the smoother, we want to estimate the smoothed distribution, also called the *posterior*

$$p(\mathbf{s}_t \mid \mathbf{y}_{1:T})$$

which uses all observations (see again Figure 4.3 for an illustration). By letting the Kalman filter "run" beforehand, we get the filtered distribution $p(\mathbf{s}_t \mid \mathbf{y}_{1:t})$. The smoothing equations are given as

$$\hat{\mathbf{s}}_{t|T} = \hat{\mathbf{s}}_{t|t} + \mathbf{J}_t(\hat{\mathbf{s}}_{t+1|T} - \hat{\mathbf{s}}_{t+1|t}) \tag{4.8}$$

$$\hat{\mathbf{V}}_{t|T} = \hat{\mathbf{V}}_{t|t} + \mathbf{J}_t(\hat{\mathbf{V}}_{t+1|T} - \hat{\mathbf{V}}_{t+1|t})\mathbf{J}_t^T \tag{4.9}$$

with the auxiliary variable $\mathbf{J}_t = \hat{\mathbf{V}}_{t|t}\mathbf{A}^T\hat{\mathbf{V}}_{t+1|t}^{-1}$. The resulting estimates $\hat{\mathbf{s}}_{t|T}$ and $\hat{\mathbf{V}}_{t|T}$ then form the mean and covariance of the smoothed/posterior distribution $p(\mathbf{s}_t \mid \mathbf{y}_{1:T})$. These equations form a recursive algorithm starting off with the final state and covariance estimates of the filter.

Hence, we need the Kalman filter before using the RTS smoother. Due to this hard wiring between the two algorithms, the RTS smoother is also often referred to as the Kalman smoother.

Note that these equations only depend on the state dynamics matrix $\mathbf{A}$, so a system that is linear in the state and nonlinear in the observations can still use the standard RTS smoother and no further modification is required.

4.3. Cubature Rules and Filtering

As we have already discussed, the plain Kalman filter not applicable to nonlinear systems without modifications. In this section we want to motivate the need for *cubature rules* and outline their derivation and how to apply them.

Often when deriving probabilistic algorithms like EM or the Kalman filter, we have to evaluate expectations $\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\big[\mathbf{f}(\mathbf{x})\big]$ and hence integrals of the form:

$$\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\big[\mathbf{f}(\mathbf{x})\big] = \int \mathbf{f}(\mathbf{x})p(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

In this thesis and lots of other literature, we have Gaussian distributions $p(\mathbf{x}) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$, leading to Gaussian integrals of the form

$$\mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})}\left[\mathbf{f}(\mathbf{x})\right] = \int \mathbf{f}(\mathbf{x}) \, \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) \, \mathrm{d}\mathbf{x}$$

which do not have closed form solutions for arbitrary, nonlinear, functions $\mathbf{f}(\mathbf{x})$. Two common approaches for approximating the expectations/integrals are Monte Carlo Estimation, leading to Markov Chain Monte Carlo (MCMC), particle filters, and numerical integration methods, known as cubature[1] rules.

To approximate Gaussian integrals, we need a set of m *sigma points* $\mathbf{x}_i$ to evaluate $\mathbf{f}(\mathbf{x})$ at and a set weights w_i to form a sum over the sigma points:

$$\int \mathbf{f}(\mathbf{x}) \, \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) \, \mathrm{d}\mathbf{x} \approx \sum_{i=1}^{m} w_i \mathbf{f}(\mathbf{x}_i)$$

In this thesis we utilize the spherical-radial cubature rule [Sol10] which is a third-degree approximation for Gaussian integrals. A third-degree approximation means that "this rule is exact for all monomials up to degree three" [Sol10, p. 18]. In this cubature rule, the approximation is a finite sum over $2n$ points, where n is the dimensionality of $\mathbf{x}$, i.e. $\mathbf{x} \in \mathbb{R}^n$. With $\boldsymbol{\Sigma}^{1/2}$ being the Cholesky decomposition of $\boldsymbol{\Sigma}$, so $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}^{1/2}\boldsymbol{\Sigma}^{1/2,T}$, the approximation is given as

$$\int \mathbf{f}(\mathbf{x}) \, \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) \, \mathrm{d}\mathbf{x} \approx \frac{1}{2n} \sum_{i=1}^{2n} \mathbf{f}(\boldsymbol{\Sigma}^{1/2}\boldsymbol{\xi}_i - \boldsymbol{\mu})$$

with the cubature points $\boldsymbol{\xi}_i = \sqrt{n}[\mathbf{1}]_i$. Here the points $[\mathbf{1}]_i$ are "from the intersections between the Cartesian axes and the n-dimensional unit hypersphere" [Sol10]. That is, the point $[\mathbf{1}]_i$ is the i-th row vector of the block matrix $\begin{bmatrix}\mathbf{I} & -\mathbf{I}\end{bmatrix}$, where $\mathbf{I}$ is the n-dimensional identity matrix. In the rest of this thesis we will write

$$SRC[\mathbf{f}; \boldsymbol{\mu}, \boldsymbol{\Sigma}] \coloneqq \sum_{i=1}^{m} w_i \mathbf{f}(\mathbf{x}_i)$$

for evaluations of the spherical-radial cubature rule as it is much shorter. Also, it highlights that the usage of the cubature rule and not the formula on how to compute it.

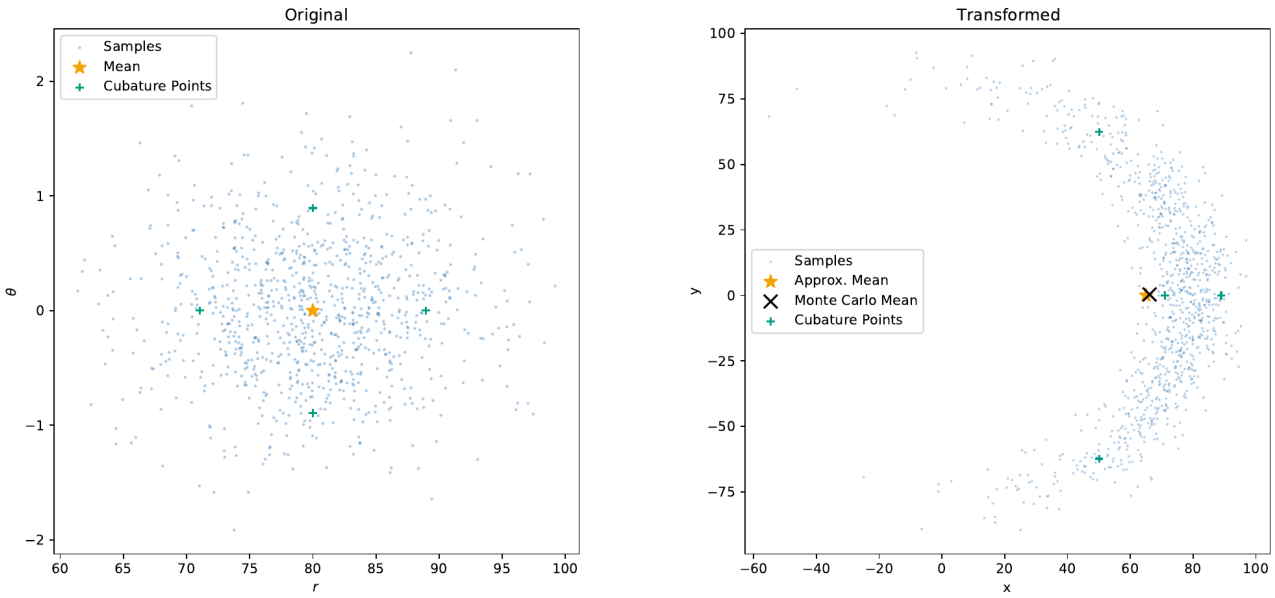
Figure 4.4 shows the spherical-radial cubature rule working on a transition from polar coordinates to Cartesian coordinates where the transformation is given by:

$$x = r\cos\theta \qquad\qquad y = r\sin\theta$$

The polar coordinate particles have been sampled from a multivariate Gaussian with diagonal covariance:

$$r \sim \mathcal{N}(80,\, 40) \qquad\qquad \theta \sim \mathcal{N}(0,\, 0.4)$$

[1] "Classical" numerical integration methods are called "quadrature" as they form a rectangle under the curve to approximate the area, while in higher dimensions this is more or less a "cube" or "hypercube", hence it is called "cubature".



(a) The original samples in polar coordinates with the real mean shown as an orange star in the middle.

(b) The approximate mean calculated via the spherical-radial cubature rule is shown as an orange star, positioned at the mean of the transformed cubature points. This plot also shows the Monte Carlo estimate as a black cross right above the cubature estimate, showing that the cubature estimate is really accurate with significantly less computational effort.

Figure 4.4.: These plots show how the spherical-radial cubature rule is used to estimate the mean of the Gaussian on the left after it has been transformed with a polar transformation to the right. To illustrate the shape of the Gaussian after it has been transformed, we added the blue dots as sample values of the Gaussian. These are 1000 sample points also used to calculate the Monte Carlo estimate of the mean. The sigma points are shown as green pluses.
Adopted from [Sol10].

4.3.1. The Unscented Transform

The *unscented* transform, originally introduced by S. J. Julier and J. K. Uhlmann [JUD95] is another method for approximating nonlinear Gaussian integrals with a parameterized transformation, the Unscented Transformation (UT). As shown by [Sol10], the spherical-radial cubature rule is just a special case of the more general UT. We will follow Solins derivation of connection here. For UT, $2n+1$ sigma points are formed as a matrix

$$\boldsymbol{\mathcal{X}} = \begin{bmatrix} \boldsymbol{\mu} & \boldsymbol{\mu}\mathbf{1}^T + \gamma\boldsymbol{\Sigma}^{1/2} & \boldsymbol{\mu}\mathbf{1}^T - \gamma\boldsymbol{\Sigma}^{1/2} \end{bmatrix} \tag{4.10}$$

where $\gamma \coloneqq \sqrt{n+\lambda}$ and $\lambda \coloneqq \alpha^2(n+\kappa) - n$ are scaling parameters defined using the parameters of the UT α and κ. The approximation for the transformed mean and covariance is then given as

$$\hat{\boldsymbol{\mu}} \coloneqq \mathbb{E}_{\mathbf{x}\sim\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})}\big[\mathbf{f}(\mathbf{x})\big] \approx \sum_{i=1}^{2n+1} w_{i-1}^{(m)} \mathbf{f}(\boldsymbol{\mathcal{X}}_i) \tag{4.11}$$

$$\hat{\boldsymbol{\Sigma}} \coloneqq \mathrm{Cov}_{\mathbf{x}\sim\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})}\big[\mathbf{f}(\mathbf{x})\big] \approx \sum_{i=1}^{2n+1} w_{i-1}^{(c)} \big(\mathbf{f}(\boldsymbol{\mathcal{X}}_i) - \hat{\boldsymbol{\mu}}\big)\big(\boldsymbol{\mathcal{X}}_i - \hat{\boldsymbol{\mu}}\big)^T \tag{4.12}$$

where $\boldsymbol{\mathcal{X}}_i$ is the i-th column of the sigma points matrix $\boldsymbol{\mathcal{X}}$. The weights $w_i^{(m)}$ for the mean and $w_i^{(c)}$ for the covariance are defined as

$$\begin{aligned} w_0^{(m)} &\coloneqq \frac{\lambda}{n+\lambda} & \qquad w_0^{(c)} &\coloneqq \frac{\lambda}{n+\lambda} + (1-\alpha^2+\beta) \\ w_i^{(m)} &\coloneqq \frac{1}{2(n+\lambda)} & \qquad w_i^{(c)} &\coloneqq \frac{1}{2(n+\lambda)} \end{aligned}$$

where β is another parameter of the UT. To get back the spherical radial cubature rule, where all weights $w_i = 1/(\,2n)$ are the same for each sigma point and the sigma point $\boldsymbol{\mathcal{X}}_0$ has a weight of 0, i.e. the mean is ignored, we have to set $\lambda = 0$. Hence, the spherical-radial cubature rule is a special case of the UT with the parameters set as $\alpha = \pm 1$, $\beta = \kappa = 0$. That means we can apply all theory developed for the UT to the cubature rule by setting the parameters accordingly.

4.3.2. Cubature Kalman Filtering and Smoothing

Following [DO11, Sol10], we can use the spherical-radial cubature rule to extend the Kalman filter and RTS smoother to nonlinear systems with Gaussian noise, i.e.

$$\begin{aligned} \mathbf{s}_{t+1} &\sim \mathcal{N}\big(\mathbf{f}(\mathbf{s}_t), \mathbf{Q}\big) \\ \mathbf{y}_t &\sim \mathcal{N}\big(\mathbf{g}(\mathbf{s}_t), \mathbf{R}\big) \end{aligned}$$

In algorithm 4.1 we show the cubature Kalman filter, where the code line/equation correspondence to the linear Kalman filter is shown in Table 4.1. Analogous, the cubature RTS smoother is shown in algorithm 4.2 and the correspondence table is shown in Table 4.2.

Algorithm 4.1: Spherical-Radial Cubature Kalman Filter

Data: Initial state $\mathbf{m}_0 \in \mathbb{R}^k$, covariance $\mathbf{V}_0$ and observations $\mathbf{y}_{1:T} \in \mathbb{R}^{p \times T}$.

1 $\hat{\mathbf{s}}_{0|0} \leftarrow \mathbf{m}_0$
2 $\hat{\mathbf{V}}_{0|0} \leftarrow \mathbf{V}_0$
3 **for** $t = 1, 2, \cdots, T$ **do**
4 $\quad \boldsymbol{\xi}_i \leftarrow \sqrt{n}[\mathbf{1}]_i$
$\quad$ // **Prediction:**
5 $\quad\quad \mathbf{x}_{i,t-1|t-1} \leftarrow \hat{\mathbf{V}}_{t-1|t-1}^{1/2} \boldsymbol{\xi}_i + \hat{\mathbf{s}}_{t-1|t-1}$
6 $\quad\quad\quad \hat{\mathbf{s}}_{t|t-1} \leftarrow \frac{1}{2n} \sum_{i=1}^{2n} \mathbf{f}(\mathbf{x}_{i,t-1|t-1})$
7 $\quad\quad\quad \hat{\mathbf{V}}_{t|t-1} \leftarrow \frac{1}{2n} \sum_{i=1}^{2n} \mathbf{f}(\mathbf{x}_{i,t-1|t-1}) \mathbf{f}^T(\mathbf{x}_{i,t-1|t-1}) - \hat{\mathbf{s}}_{t|t-1} \hat{\mathbf{s}}_{t|t-1}^T + \mathbf{Q}$
$\quad$ // **Correction:**
8 $\quad\quad\quad \mathbf{x}_{i,t|t-1} \leftarrow \hat{\mathbf{V}}_{t|t-1}^{1/2} + \hat{\mathbf{s}}_{t|t-1}$
9 $\quad\quad\quad\quad \hat{\mathbf{y}}_{t|t-1} \leftarrow \frac{1}{2n} \sum_{i=1}^{2n} \mathbf{g}(\mathbf{x}_{i,t|t-1})$
10 $\quad\quad\quad\quad\quad \mathbf{S}_t \leftarrow \frac{1}{2n} \sum_{i=1}^{2n} \mathbf{g}(\mathbf{x}_{i,t|t-1}) \mathbf{g}^T(\mathbf{x}_{i,t|t-1}) - \hat{\mathbf{y}}_{t|t-1} \hat{\mathbf{y}}_{t|t-1}^T + \mathbf{R}$
11 $\quad\quad\quad\quad\quad \mathbf{J}_t \leftarrow \frac{1}{2n} \sum_{i=1}^{2n} \mathbf{f}(\mathbf{x}_{i,t-1|t-1}) \mathbf{g}^T(\mathbf{x}_{i,t|t-1}) - \hat{\mathbf{s}}_{t|t-1} \hat{\mathbf{y}}_{t|t-1}^T$
12 $\quad\quad\quad\quad\quad \mathbf{K}_t \leftarrow \mathbf{J}_t \mathbf{S}_t^{-1}$
13 $\quad\quad\quad\quad\quad \hat{\mathbf{s}}_{t|t} \leftarrow \hat{\mathbf{s}}_{t|t-1} + \mathbf{K}_t(\mathbf{y}_t - \hat{\mathbf{y}}_{t|t-1})$
14 $\quad\quad\quad\quad\quad \hat{\mathbf{V}}_{t|t} \leftarrow \hat{\mathbf{V}}_{t|t-1} + \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^T$

Name	Code Line	Linear Equation
Predicted State	6	(4.1)
Predicted Covariance	7	(4.2)
Filtered State	13	(4.3)
Filtered Covariance	14	(4.4)
Innovation	9	(4.5)
Residual Covariance	10	(4.6)
Kalman Gain	12	(4.7)

Table 4.1.: This table shows how the code lines of the cubature Kalman filer relate to the corresponding linear Kalman filter equations.

Algorithm 4.2: Spherical-Radial Cubature Rauch-Tung-Striebel Smoother

Data: All values from the Kalman filter.

1 **for** $t = T-1, T-2, \cdots, 1$ **do**
2 $\quad \boldsymbol{\xi}_i \leftarrow \sqrt{n}[\mathbf{1}]_i$
3 $\quad \mathbf{x}_i \leftarrow \hat{\mathbf{V}}_{t|t}^{1/2} \boldsymbol{\xi}_i + \hat{\mathbf{s}}_{t|t}$
4 $\quad \mathbf{D}_t \leftarrow \frac{1}{2n} \sum_{i=1}^{2n} (\mathbf{x}_i - \hat{\mathbf{s}}_{t|t}) \big(\mathbf{f}(\mathbf{x}_i) - \hat{\mathbf{s}}_{t+1|t}\big)^T$
5 $\quad \mathbf{J}_t \leftarrow \mathbf{D}_t \hat{\mathbf{V}}_{t+1|t}^{-1}$
6 $\quad \hat{\mathbf{s}}_{t|T} \leftarrow \hat{\mathbf{s}}_{t|t} + \mathbf{J}_t(\hat{\mathbf{s}}_{t+1|T} - \hat{\mathbf{s}}_{t+1|t})$
7 $\quad \hat{\mathbf{V}}_{t|T} \leftarrow \hat{\mathbf{V}}_{t|t} + \mathbf{J}_t(\hat{\mathbf{V}}_{t+1|T} - \hat{\mathbf{V}}_{t+1|t}) \mathbf{C}_t^T$

Name	Code Line	Linear Equation
Smoothed State	6	(4.8)
Smoothed Covariance	7	(4.8)

Table 4.2.: This table shows how the code lines of the cubature RTS smoother relate to the corresponding linear RTS smoother equations.

4.3.3. Square-Root Filtering / Smoothing

While cubature rules are extremely handy for approximating Gaussian integrals, they require the Cholesky decomposition of the covariance $\mathbf{\Sigma}$. The Cholesky decomposition might lead to numerical instabilities if the matrix is ill-conditioned or even not positive definite, in which case no Cholesky decomposition exists. As we will see in chapter 5, we need the Cholesky decomposition of every smoothed covariance $\hat{\mathbf{V}}_{t|T}$, $t = 1, 2, \cdots, T$, leading to numerical instabilities if the cubature approximation in the filtering does not "create" valid covariance matrices, i.e. positive definite and symmetric matrices. One approach for solving this problem is estimating the Cholesky decomposition directly, without explicitly decomposing the matrix. Such filtering/smoothing is known as *square-root filtering/smoothing* [VW01, Rut13].

We will mostly follow the derivation in [Rut13] and only outline the key results shortly. With the sigma points $\boldsymbol{\mathcal{X}}$ from the UT equation (4.10) and the corresponding weights, we can define the transformed sigma points

$$\boldsymbol{\mathcal{X}}'_i = \mathbf{f}(\boldsymbol{\mathcal{X}}_i)$$

and utilize the UT mean $\hat{\boldsymbol{\mu}}$ (4.11) and covariance $\hat{\mathbf{\Sigma}}$ (4.12) estimates. By applying another transformation to the transformed sigma points,

$$\boldsymbol{\mathcal{Y}}_i = \sqrt{w_i^{(c)}}(\boldsymbol{\mathcal{X}}' - \hat{\boldsymbol{\mu}})$$

which are the "scaled residuals" of the sigma points, we can choose an orthogonal matrix $\mathbf{\Theta}$ such that

$$\boldsymbol{\mathcal{Y}}\mathbf{\Theta} = \begin{bmatrix}\mathbf{B} & \mathbf{O}_{k\times(k+1)}\end{bmatrix} \tag{4.13}$$

Squaring both sides of this equation yields the covariance estimate

$$\hat{\mathbf{\Sigma}} = \boldsymbol{\mathcal{Y}}\boldsymbol{\mathcal{Y}}^T = \boldsymbol{\mathcal{Y}}\mathbf{\Theta}\mathbf{\Theta}^T\boldsymbol{\mathcal{Y}} = \begin{bmatrix}\mathbf{B} & \mathbf{O}_{k\times(k+1)}\end{bmatrix}\begin{bmatrix}\mathbf{B}^T \\ \mathbf{O}_{(k+1)\times n}\end{bmatrix} = \mathbf{B}\mathbf{B}^T$$

so $\mathbf{B}$ is the Cholesky decomposition of $\hat{\mathbf{\Sigma}}$. But the equation (4.13), where we neither now the orthogonal matrix $\mathbf{\Theta}$ nor the right side of the equation, is exactly the type of matrix decomposition known as the *QR decomposition* that is numerically more stable than the Cholesky decomposition as it does not require any special structure like positive definiteness of the matrix. By applying this argumentation in an analogous way to every covariance computation in the cubature Kalman filter and RTS smoother, we find a complete square-root filtering and smoothing algorithm propagating the Cholesky decomposition of the covariance matrices $\hat{\mathbf{V}}_{t|t-1}$, $\hat{\mathbf{V}}_{t|t}$ and $\hat{\mathbf{V}}_{t|T}$.

The square-root cubature Kalman filter is summarized in algorithm 4.3 and the square-root cubature RTS smoother is summarized in algorithm 4.4.

The square-root filtering now allows us to pass the Cholesky decomposition of the covariances directly through the smoothing and filtering process, allowing higher numerical stability because we guarantee the covariances to be positive definite even with approximations.

Algorithm 4.3: Square-Root Cubature Kalman Filter

Data: Initial state $\mathbf{m}_0 \in \mathbb{R}^k$, covariance sqrt $\mathbf{V}_0^{1/2}$ and observations $\mathbf{y}_{1:T} \in \mathbb{R}^{p \times T}$.

$\hat{\mathbf{s}}_{0|0} \leftarrow \mathbf{m}_0$

$\hat{\mathbf{V}}_{0|0}^{1/2} \leftarrow \mathbf{V}_0^{1/2}$

for $t = 1, 2, \cdots, T$ **do**

$$\mathbf{\Lambda} \leftarrow \sqrt{n} \cdot \hat{\mathbf{V}}_{t-1|t-1}^{1/2}$$

$$\boldsymbol{\mathcal{X}}_{t-1|t-1} \leftarrow \left[\hat{\mathbf{s}}_{t-1|t-1}\mathbf{1}^T + \mathbf{\Lambda} \quad \hat{\mathbf{s}}_{t-1|t-1}\mathbf{1}^T - \mathbf{\Lambda}\right]$$

$$\boldsymbol{\mathcal{X}}^s_{t|t-1,i} \leftarrow \mathbf{f}(\boldsymbol{\mathcal{X}}_{t-1|t-1,i}) + (-1)^i \cdot \mathbf{Q}, \quad i = 1, 2, \cdots, 2k$$

$$\boldsymbol{\mathcal{X}}^y_{t|t-1,i} \leftarrow \mathbf{g}\big(\boldsymbol{\mathcal{X}}^s_{t|t-1,i}\big) + (-1)^i \cdot \mathbf{R}, \quad i = 1, 2, \cdots, 2k$$

$$\hat{\mathbf{s}}_{t|t-1} \leftarrow \tfrac{1}{2k}\sum_{i=1}^{2k} \boldsymbol{\mathcal{X}}^s_{t|t-1,i}$$

$$\hat{\mathbf{y}}_{t|t-1} \leftarrow \tfrac{1}{2k}\sum_{i=1}^{2k} \boldsymbol{\mathcal{X}}^y_{t|t-1,i}$$

$$\boldsymbol{\mathcal{Y}}^s_{t|t-1,i} \leftarrow \big(\boldsymbol{\mathcal{X}}^s_{t|t-1,i} - \hat{\mathbf{s}}_{t|t-1}\big)/\sqrt{2k}$$

$$\boldsymbol{\mathcal{Y}}^y_{t|t-1,i} \leftarrow \big(\boldsymbol{\mathcal{X}}^y_{t|t-1,i} - \hat{\mathbf{y}}_{t|t-1}\big)/\sqrt{2k}$$

// Find $\mathbf{S}_t^{1/2}$, $\mathbf{J}_t$ and $\hat{\mathbf{V}}_{t|t}^{1/2}$ using the QR decomposition:

$$\begin{bmatrix} \boldsymbol{\mathcal{Y}}^y_{t|t-1} \\ \boldsymbol{\mathcal{Y}}^s_{t|t-1} \end{bmatrix} \mathbf{\Theta} = \begin{bmatrix} \mathbf{S}_t^{1/2} & \mathbf{O} & \mathbf{O} \\ \mathbf{J}_t & \hat{\mathbf{V}}_{t|t}^{1/2} & \mathbf{O} \end{bmatrix}$$

$$\mathbf{K}_t \leftarrow \mathbf{J}_t \mathbf{S}_t^{-1/2}$$

$$\hat{\mathbf{s}}_{t|t} \leftarrow \hat{\mathbf{s}}_{t|t-1} + \mathbf{K}_t(\mathbf{y}_t - \hat{\mathbf{y}}_{t|t-1})$$

Algorithm 4.4: Square-Root Cubature Rauch-Tung-Striebel Smoother

Data: All values from the Kalman filter.

for $t = T-1, T-2, \cdots, 1$ **do**

$$\boldsymbol{\mathcal{Y}}^s_{t|t,i} \leftarrow (\boldsymbol{\mathcal{X}}_{t|t,i} - \hat{\mathbf{s}}_{t|t})$$

// Find $\mathbf{D}_t$ and $\mathbf{Z}_t$ using the QR decomposition:

$$\begin{bmatrix} \boldsymbol{\mathcal{Y}}^s_{t+1|t} \\ \boldsymbol{\mathcal{Y}}^s_{t|t} \end{bmatrix} \mathbf{\Theta} = \begin{bmatrix} \hat{\mathbf{V}}_{t+1|t}^{1/2} & \mathbf{O} & \mathbf{O} \\ \mathbf{D}_t & \mathbf{Z}_t & \mathbf{O} \end{bmatrix}$$

$$\mathbf{J}_t \leftarrow \mathbf{D}_t \hat{\mathbf{V}}_{t+1|t}^{-1/2}$$

$$\hat{\mathbf{s}}_{t|T} \leftarrow \hat{\mathbf{s}}_{t|t} + \mathbf{J}_t(\hat{\mathbf{s}}_{t+1|T} - \hat{\mathbf{s}}_{t+1|t})$$

// Find $\hat{\mathbf{V}}_{t|T}^{1/2}$ using the QR decomposition:

$$\begin{bmatrix} \mathbf{Z}_t & \mathbf{J}_t \hat{\mathbf{V}}_{t+1|T}^{1/2} \end{bmatrix} \mathbf{\Theta} = \begin{bmatrix} \hat{\mathbf{V}}_{t|T}^{1/2} & \mathbf{O} \end{bmatrix}$$

5. The Koopman Inference Algorithm

In this chapter we will introduce the theoretical background of our contribution, the Koopman inference algorithm, that we will implement as a proof of concept in chapter 6. We will start by introducing the thoughts that led to the idea of using variational autoencoder and EM for Koopman dynamical systems, then formulate and solve the arising (approximate) inference problem.

By looking at the graphical model of an LGDS and the state transition model for a Koopman dynamical system in Figure 5.1, we can see that there are lots of similarities. The first similarity is that both systems assume a latent state that transitions linearly, either with a state dynamics matrix $\mathbf{A}$ for the LGDS or with the Koopman operator[1] $\mathbf{K}$. The greatest difference is that measurements in classic LGDS are taken linearly with an observation matrix $\mathbf{C}$ and nonlinear in the Koopman system with the measurement function $\mathbf{h}(\cdot)$.

Our idea is to "flip" the Koopman dynamical system and replace the observation matrix $\mathbf{C}$ with a nonlinear observation function $\mathbf{g}(\cdot)$, that takes the linear states $\mathbf{s}_t$ and maps them into a nonlinear observation space $\mathbf{y}_t$. In other words, we seek to find the inverse function of $\mathbf{h}(\cdot)$ to map out of the linear embedding that Koopman theory guarantees us to exist. Our assumption that such an inverse function exists is backed by previous accomplishments in data-driven Koopman analysis [LKB18] which also seek and find such an inverse mapping (see chapter 2 for more information).

Additionally, we contribute a probabilistic view on the Koopman operator, being able to gauge our uncertainty about the embedding and the inverse mapping to the observation space.

At this point we want to clear up some inconsistencies in notation and naming that builds when working with two systems where "observation" means opposing concepts. From now on, we will work on the graphical system shown in Figure 5.2 characterized by the dynamics

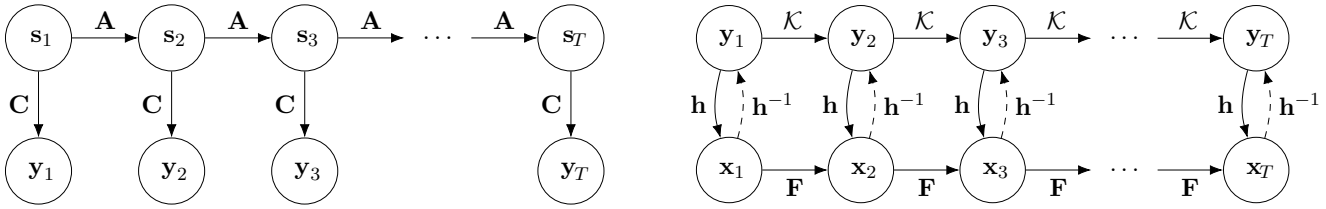
$$\begin{aligned} \mathbf{s}_{t+1} &= \mathbf{A}\mathbf{s}_t + \mathbf{w}_t, \quad \mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}) \\ \mathbf{y}_t &= \mathbf{g}(\mathbf{s}_t) + \mathbf{v}_t, \quad \mathbf{v}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}) \end{aligned}$$

that can equivalently be formulated as:

$$\begin{aligned} \mathbf{s}_{t+1} &\sim \mathcal{N}(\mathbf{A}\mathbf{s}_t, \mathbf{Q}) \\ \mathbf{y}_t &\sim \mathcal{N}\big(\mathbf{g}(\mathbf{s}_t), \mathbf{R}\big) \end{aligned}$$

We call $\mathbf{s}_t$ the *latent variables* or *latents* in the *latent space* with dimensionality k, $\mathbf{y}_t$ the *observation variables* or *observations* in the *observation space* with dimensionality p, $\mathbf{g} : \mathbb{R}^k \to \mathbb{R}^p$ the *observation function* mapping latents to observations, $\mathbf{A}$ the *(state) dynamics matrix*, $\mathbf{Q}$ the state covariance matrix and $\mathbf{R}$ the observation covariance matrix. Note that we also assume the Gaussian additive noise to be uncorrelated, i.e. we assume diagonal covariance matrices $\mathbf{Q}$ and $\mathbf{R}$.

[1] From now on, we suppose a finite-dimensional matrix approximation $\mathbf{K}$ of the Koopman operator $\mathcal{K}$.



(a) Graphical model of a linear Gaussian dynamical system with the latent state $\mathbf{s}_t$ and the (linear) observations $\mathbf{y}_t$. In contrast to the Koopman system, this system is not deterministic and the arrows represent probabilistic dependency rather than hard transitions.

(b) State transition model of a Koopman dynamical system with the Koopman operator $\mathcal{K}$ that can be approximated with a matrix $\mathbf{K}$. In contrast to the LGDS the state is represented via nonlinear transitions and the observations are the linear dynamics.
Adopted from [BBPK16].

Figure 5.1.: Side-by-side comparison of the a LGDS on the left and a Koopman dynamical system on the right. This side-by-side view highlights our idea of interpreting an Koopman system as a semi-linear dynamical system (i.e. an LGDS with nonlinear observations).

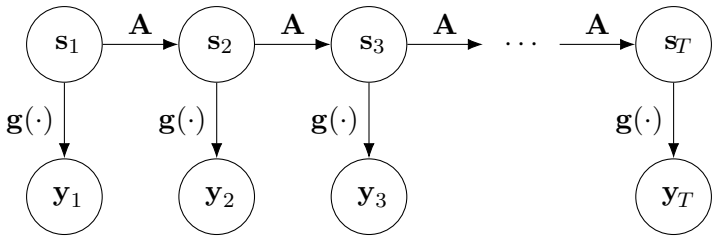


Figure 5.2.: The graphical model of the Koopman inference model. Given an observation sequence $\mathbf{y}_{1:T}$, we seek the latent dynamics and the corresponding nonlinear mapping $\mathbf{g}(\cdot)$ from the latent space to the observations.

5.1. Formulating and Solving the Inference Problem using an EM Algorithm

We seek to find the matrices $\mathbf{A}$, $\mathbf{Q}$ and $\mathbf{R}$ and the observation function $\mathbf{g}_{\boldsymbol{\theta}}(\cdot)$, parameterized by $\boldsymbol{\theta}$ and the initial state distribution $\mathbf{s}_1 \sim \mathcal{N}(\mathbf{m}_0, \mathbf{V}_0)$ that maximize the likelihood

$$p(\mathbf{s}_{1:T}, \mathbf{y}_{1:T}) = p(\mathbf{s}_1) \prod_{t=2}^{T} p(\mathbf{s}_t \mid \mathbf{s}_{t-1}) \prod_{t=1}^{T} p(\mathbf{y}_t \mid \mathbf{s}_t)$$

which factors over the conditional distributions due to the Markovian assumption. Subsequently, we can proceed by not maximizing the likelihood but the log-likelihood which is easier to maximize as the Gaussian lies in the exponential family, exhibiting a convex optimization problem[2]:

$$\log p(\mathbf{s}_{1:T}, \mathbf{y}_{1:T}) = \log p(\mathbf{s}_1) + \sum_{t=2}^{T} \log p(\mathbf{s}_t \mid \mathbf{s}_{t-1}) + \sum_{t=1}^{T} \log p(\mathbf{y}_t \mid \mathbf{s}_t)$$

Inserting the Gaussian distributions

$$p(\mathbf{s}_1) = \mathcal{N}(\mathbf{s}_1 \mid \mathbf{m}_0, \mathbf{V}_0) \qquad p(\mathbf{s}_t \mid \mathbf{s}_{t-1}) = \mathcal{N}(\mathbf{s}_t \mid \mathbf{A}\mathbf{s}_{t-1}, \mathbf{Q}) \qquad p(\mathbf{y}_t \mid \mathbf{s}_t) = \mathcal{N}\big(\mathbf{y}_t \mid \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{s}_t), \mathbf{R}\big)$$

yields the log-likelihood

$$\begin{aligned}
\log p(\mathbf{s}_{1:T}, \mathbf{y}_{1:T}) = &-\frac{T(k+p)}{2}\log(2\pi) - \frac{1}{2}\log|\mathbf{V}_0| - \frac{T-1}{2}\log|\mathbf{Q}| - \frac{T}{2}\log|\mathbf{R}| \\
&- \frac{1}{2}(\mathbf{s}_1 - \mathbf{m}_0)^T \mathbf{V}_0^{-1}(\mathbf{s}_1 - \mathbf{m}_0) \\
&- \frac{1}{2}\sum_{t=2}^{T}(\mathbf{s}_t - \mathbf{A}\mathbf{s}_{t-1})^T \mathbf{Q}^{-1}(\mathbf{s}_t - \mathbf{A}\mathbf{s}_{t-1}) \\
&- \frac{1}{2}\sum_{t=1}^{T}(\mathbf{y}_t - \mathbf{g}_t)^T \mathbf{R}^{-1}(\mathbf{y}_t - \mathbf{g}_t)
\end{aligned}$$

with the abbreviation $\mathbf{g}_t \coloneqq \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{s}_t)$.

But maximizing this quantity is impossible as we do not have the latent states $\mathbf{s}_{1:T}$. Hence, as usual in EM (see section 3.3), we instead maximize the expected log-likelihood and estimate the latent states based on out previous guess for the state dynamics. These two steps form our E- and M-step of the Koopman inference algorithm:

- E-Step: Estimate the latent states $\mathbf{s}_{1:T}$.
- M-Step: Maximize the expected log-likelihood w.r.t. the state dynamics and observation function parameters.

We will proceed with deriving the M-step and finish this section off by deriving the E-step. Be aware that we only briefly touch the derivation here and focus on the primary steps. See section A.1 for the complete derivation with explanations throughout every step.

[2] Strictly speaking, the optimization problem is only convex w.r.t. $\mathbf{A}$ and $\mathbf{Q}$. For $\mathbf{R}$ and $\boldsymbol{\theta}$, the problem is still non-convex, but numerically more stable as sums are easier to differentiate and to compute than products.

For the M-step, we first need the expected log-likelihood $Q \coloneqq \mathbb{E}_{\mathbf{s}_{1:T}}\left[p(\mathbf{s}_{1:T}, \mathbf{y}_{1:T})\right]$ as stated previously. This quantity depends on three expectations that are estimated in the E-step:

$$\hat{\mathbf{s}}_{t|t_0}^{(n)} \coloneqq \mathbb{E}\left[\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t_0}\right] \qquad \hat{\mathbf{s}}_{t|t_0} \coloneqq \frac{1}{N}\sum_{n=1}^{N} \hat{\mathbf{s}}_{t|t_0}^{(n)} \tag{5.1}$$

$$\mathbf{P}_{t|t_0}^{(n)} \coloneqq \mathbb{E}\left[\mathbf{s}_t^{(n)}\mathbf{s}_t^{(n),T} \,\middle|\, \mathbf{y}_{1:t_0}\right] \qquad \mathbf{P}_{t|t_0} \coloneqq \frac{1}{N}\sum_{n=1}^{N} \mathbf{P}_{t|t_0}^{(n)} \tag{5.2}$$

$$\mathbf{P}_{t,t-1|t_0}^{(n)} \coloneqq \mathbb{E}\left[\mathbf{s}_t^{(n)}\mathbf{s}_{t-1}^{(n),T} \,\middle|\, \mathbf{y}_{1:t_0}\right] \qquad \mathbf{P}_{t,t-1|t_0} \coloneqq \frac{1}{N}\sum_{n=1}^{N} \mathbf{P}_{t,t-1|t_0}^{(n)} \tag{5.3}$$

These expectations are called the expected state (5.1), the self-correlation (5.2) and the cross-correlation (5.3), respectively. We also introduced an upper index, $\cdot^{(n)}$, which indicates which observation sequence our current observation is from, because the algorithm can simultaneously learn on N observation sequences. For brevity, we may write $\hat{\mathbf{s}}_t$ instead of $\hat{\mathbf{s}}_{t|T}$ (analogous for all other values). Additionally, we have to define two quantities for the expectations of the nonlinear observation function $\mathbf{g}_{\boldsymbol{\theta}}$ that we cannot evaluate in closed form:

$$\hat{\mathbf{g}}_t^{(n)} \coloneqq \mathbb{E}\left[\mathbf{g}_t^{(n)} \,\middle|\, \mathbf{y}_{1:T}\right] \tag{5.4}$$

$$\mathbf{G}_t^{(n)} \coloneqq \mathbb{E}\left[\mathbf{g}_t^{(n)}\mathbf{g}_t^{(n),T} \,\middle|\, \mathbf{y}_{1:T}\right] \tag{5.5}$$

We can now evaluate the expected log-likelihood depending on the above quantities using the trace-trick[3], giving the following (enormous) quantity:

$$\begin{aligned}
Q &= \mathbb{E}_{\mathbf{s}_{1:T}}\left[p(\mathbf{s}_{1:T}, \mathbf{y}_{1:T})\right] \\
&= -\frac{NT(k+p)}{2}\ln(2\pi) - \frac{N}{2}\ln|\mathbf{V}_0| - \frac{N(T-1)}{2}\ln|\mathbf{Q}| - \frac{NT}{2}\ln|\mathbf{R}| \\
&\quad - \frac{N}{2}\operatorname{tr}\left(\mathbf{P}_1\mathbf{V}_0^{-1}\right) + \frac{N}{2}\operatorname{tr}\left(\hat{\mathbf{s}}_1\mathbf{m}_0^T\mathbf{V}_0^{-1}\right) + \frac{N}{2}\operatorname{tr}\left(\mathbf{m}_0\hat{\mathbf{s}}_1^T\mathbf{V}_0^{-1}\right) - \frac{N}{2}\operatorname{tr}\left(\mathbf{m}_0\mathbf{m}_0^T\mathbf{V}_0^{-1}\right) \\
&\quad - \frac{N}{2}\sum_{t=2}^{T}\operatorname{tr}\left(\mathbf{P}_t\mathbf{Q}^{-1}\right) - \operatorname{tr}\left(\mathbf{P}_{t,t-1}\mathbf{A}^T\mathbf{Q}^{-1}\right) - \operatorname{tr}\left(\mathbf{A}\mathbf{P}_{t,t-1}\mathbf{Q}^{-1}\right) - \operatorname{tr}\left(\mathbf{A}\mathbf{P}_{t-1}\mathbf{A}^T\mathbf{Q}^{-1}\right) \\
&\quad - \frac{1}{2}\sum_{n=1}^{N}\sum_{t=1}^{T}\operatorname{tr}\left(\mathbf{y}_t^{(n)}\mathbf{y}_t^{(n),T}\mathbf{R}^{-1}\right) - \operatorname{tr}\left(\mathbf{y}_t^{(n)}\hat{\mathbf{g}}_t^{(n),T}\mathbf{R}^{-1}\right) - \operatorname{tr}\left(\hat{\mathbf{g}}_t^{(n)}\mathbf{y}_t^{(n),T}\mathbf{R}^{-1}\right) + \operatorname{tr}\left(\mathbf{G}_t^{(n)}\mathbf{R}^{-1}\right)
\end{aligned}$$

[3] The trace of a product of matrices/vectors is invariant under even permutations of the matrices/vectors within the trace.

By maximizing this quantity w.r.t. the dynamics matrix $\mathbf{A}$, the covariance matrices $\mathbf{Q}$ and $\mathbf{R}$, the initial state mean $\mathbf{m}_0$ and the initial state covariance $\mathbf{V}_0$, i.e. setting the derivative w.r.t. these parameters to zero and rearranging the equations, we get the closed-form solutions

$$
\begin{aligned}
\mathbf{A}^{\text{new}} &= \left(\sum_{t=2}^{T} \mathbf{P}_{t,t-1}\right)\left(\sum_{t=2}^{T} \mathbf{P}_{t-1}\right)^{-1} \\
\mathbf{Q}^{\text{new}} &= \frac{1}{T-1}\left(\sum_{t=2}^{T} \mathbf{P}_t - \mathbf{A}^{\text{new}} \sum_{t=2}^{T} \mathbf{P}_{t,t-1}\right) \\
\mathbf{m}_0^{\text{new}} &= \hat{\mathbf{s}}_1 = \frac{1}{N}\sum_{n=1}^{N} \hat{\mathbf{s}}_1^{(n)} \\
\mathbf{V}_0^{\text{new}} &= \mathbf{P}_1 - \hat{\mathbf{s}}_1 \hat{\mathbf{s}}_1^T \\
\mathbf{R}^{\text{new}} &= \frac{1}{NT}\sum_{n=1}^{N}\sum_{t=1}^{T} \mathbf{y}_t^{(n)}\mathbf{y}_t^{(n),T} - \hat{\mathbf{g}}_t^{(n)}\mathbf{y}_t^{(n),T} - \mathbf{y}_t^{(n)}\hat{\mathbf{g}}_t^{(n),T} + \mathbf{G}_t^{(n),T}
\end{aligned}
$$

where we assume an optimal $\mathbf{g}_{\boldsymbol{\theta}}(\cdot)$ in the last equation (to compute $\mathbf{R}$). To maximize the expected log-likelihood w.r.t. the observation function $\mathbf{g}_{\boldsymbol{\theta}}$, we choose a learnable function approximator, e.g. a neural network and maximize Q using simple gradient descent or a more advanced variant like Adam [KB17]. We should note that we do not need to fully maximize Q in every M-step but that it is enough to just get better in every M-step for the convergence properties to still hold [DLR77].

We now look into the problem of evaluating $\hat{\mathbf{g}}_t^{(n)}$ and $\mathbf{G}_t^{(n)}$. Instead of using Monte Carlo methods that are both very computationally demanding (especially as the gradient has to flow through the evaluation), we are employing the spherical-radial cubature rule that we have introduced in section 4.3. That way, we can approximate the integrals both efficiently and deterministically (as compared to Monte Carlo methods):

$$
\hat{\mathbf{g}}_t^{(n)} = \int \mathbf{g}_t^{(n)} p\left(\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:T}\right) \mathrm{d}\mathbf{s}_{1:T}^{(n)} \approx SRC\left[\mathbf{g}_{\boldsymbol{\theta}};\, \hat{\mathbf{s}}_t^{(n)}, \mathbf{V}_t^{(n)}\right] \tag{5.6}
$$

$$
\mathbf{G}_t^{(n)} = \int \mathbf{g}_t^{(n)} \mathbf{g}_t^{(n),T} p\left(\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:T}\right) \mathrm{d}\mathbf{s}_{1:T}^{(n)} \approx SRC\left[\mathbf{g}_{\boldsymbol{\theta}}\mathbf{g}_{\boldsymbol{\theta}}^T;\, \hat{\mathbf{s}}_t^{(n)}, \mathbf{V}_t^{(n)}\right] \tag{5.7}
$$

This concludes the derivation of the M-step, where we do not have explicit formulas for the observation parameters $\boldsymbol{\theta}$ as we do learn these using a numerical optimizer like Adam [KB17].

Deriving the E-step is quite straightforward given the groundwork we developed in section 4.2 and subsection 4.3.2. As we heavily employ cubature rules in the M-step (in fact, in every Gradient Descent (GD) iteration in the M-step), we chose to use a square-root Kalman filter and a square-root RTS smoother (see subsection 4.3.3) to directly get the Cholesky decomposition of the used covariance matrices. We apply algorithm 4.3 after algorithm 4.4 with the state dynamics function $\mathbf{f} : \mathbb{R}^k \to \mathbb{R}^k : \mathbf{s} \mapsto \mathbf{A}\mathbf{s}$ and the observations function $\mathbf{g}_{\boldsymbol{\theta}}$. Additionally to the covariances, we have to compute the self- and cross-correlation. Following [Min99], they are given by

$$
\begin{aligned}
\mathbf{P}_t^{(n)} &= \hat{\mathbf{V}}_t^{(n)} - \hat{\mathbf{s}}_t^{(n)}\hat{\mathbf{s}}_t^{(n),T} \\
\mathbf{P}_{t,t-1}^{(n)} &= \hat{\mathbf{J}}_{t-1}\hat{\mathbf{V}}_t - \hat{\mathbf{s}}_t^{(n)}\hat{\mathbf{s}}_{t-1}^{(n),T}
\end{aligned}
$$

The whole Koopman inference algorithm is outlined in algorithm 5.1.

Algorithm 5.1: Koopman Inference

Data: N observation sequences $\mathbf{y}_{1:T}^{(n)} \in \mathbb{R}^{p \times T}$.

Initialize $\mathbf{A}$, $\mathbf{Q}$, $\mathbf{R}$, $\mathbf{m}_0$, $\mathbf{V}_0$ and $\boldsymbol{\theta}$ somehow (e.g. identity matrices).

while *not converged* **do**

// **E-Step:**

$\mathbf{s}_{1:T}^{(1:N)}$, $\hat{\mathbf{V}}_{1:T} \leftarrow$ RTS-Smoother

$\mathbf{P}_t^{(n)} \leftarrow \hat{\mathbf{V}}_t^{(n)} - \hat{\mathbf{s}}_t^{(n)} \hat{\mathbf{s}}_t^{(n),T}$ for $t = 1, 2, \cdots, T$ and $n = 1, 2, \cdots, N$

$\mathbf{P}_{t,t-1}^{(n)} \leftarrow \hat{\mathbf{J}}_{t-1} \hat{\mathbf{V}}_t - \hat{\mathbf{s}}_t^{(n)} \hat{\mathbf{s}}_{t-1}^{(n),T}$ for $t = 2, 3, \cdots, T$ and $n = 1, 2, \cdots, N$

// **M-Step:**

$\boldsymbol{\theta} \leftarrow \text{GD}\big(Q(\boldsymbol{\theta})\big)$

$$\mathbf{R} \leftarrow \frac{1}{NT} \sum_{n=1}^{N} \sum_{t=1}^{T} \mathbf{y}_t^{(n)} \mathbf{y}_t^{(n),T} - \hat{\mathbf{g}}_t^{(n)} \mathbf{y}_t^{(n),T} - \mathbf{y}_t^{(n)} \hat{\mathbf{g}}_t^{(n),T} + \mathbf{G}_t^{(n),T}$$

$$\mathbf{A} \leftarrow \left(\sum_{t=2}^{T} \mathbf{P}_{t,t-1} \right) \left(\sum_{t=2}^{T} \mathbf{P}_{t-1} \right)^{-1}$$

$$\mathbf{Q} \leftarrow \frac{1}{T-1} \left(\sum_{t=2}^{T} \mathbf{P}_t - \mathbf{A}^{\text{new}} \sum_{t=2}^{T} \mathbf{P}_{t,t-1} \right)$$

$$\mathbf{m}_0 \leftarrow \frac{1}{N} \sum_{n=1}^{N} \hat{\mathbf{s}}_1^{(n)}$$

$$\mathbf{V}_0 \leftarrow \mathbf{P}_1 - \hat{\mathbf{s}}_1 \hat{\mathbf{s}}_1^T$$

5.2. Implementation

We implemented the Koopman inference algorithm in Python using the popular array processing library NumPy [HMv+20] throughout the whole implementation, the deep learning library PyTorch [PGM+19] for the numerical optimization of $\boldsymbol{\theta}$ and Sacred [GKC+17] for experiment management. The whole implementation can be found on GitHub[4].

During the implementation we made sure on designing the algorithm as reusable as possible by separating all data generation and investigation code from the actual implementation. Separating the data generation also has the advantage that we can run benchmarks on the exact same data to compare model performance. We separated the optimization of $\boldsymbol{\theta}$ as much as possible (as outlined in algorithm 5.1) and made it possible for the user of the algorithm to easily change the optimization algorithm to use in conjunction with the parameters of the algorithm like the learning rate (defaulting to Adam [KB17] with a learning rate of 0.01). We also made all of the following configurable:

- Whether to perform whitening of the observation data or not. That is, if we perform whitening, we normalize and standardize the training observations beforehand using $\hat{\mathbf{y}}_{1:T,i} = (\mathbf{y}_{1:T,i} - \bar{\mathbf{y}}_i)/\sigma_{y_i}$ where $\bar{\mathbf{y}}_i$ is the mean and σ_{y_i} is the standard deviation of the i-th component of all measurements. Doing the whitening first sometimes improves the performance as we will see in chapter 6.
- The convergence precision, i.e. how much the log-likelihood has to change in order to continue the EM iterations. We will see that, due to numerical instabilities and approximation errors in the cubature rules, the algorithm rarely really converges before becoming numerically unstable.

  That is why we also made the maximum number of iterations configurable to support *early stopping*. Once the algorithm performed these number of EM iterations, it will abort and report the results.
- Similarly it is possible to configure the convergence precision and maximum number of iterations for the subsequent optimization of $\boldsymbol{\theta}$.

These are only the parameters that can be configured directly on the EM algorithm but the user is also able to specify how to initialize everything, i.e. the matrices $\mathbf{A}$, $\mathbf{Q}$, $\mathbf{R}$, $\mathbf{V}_0$, $\mathbf{m}_0$ and the parameters $\boldsymbol{\theta}$ of the observation function $\mathbf{g}_{\boldsymbol{\theta}}$. We used a neural network with $\tanh$ activation function and a single hidden layer for all experiments (see chapter 6 for more information about the experiment setup). We also made it possible to pass arbitrary PyTorch modules as the function approximator that can be learned via a GD-like algorithm (of course they must be differentiable and expose parameters).

For learning multiple observation sequences, as outlined in the derivation of the Koopman inference algorithm in chapter 5, it turns out to be useful to add noise to the training data which has the effect of regularizing the algorithm a bit.

5.2.1. Insights, Problems and Solutions

In this section we will focus on interesting insights and problems we found during the implementation and how to possibly solve them.

[4] See `https://github.com/fdamken/bachelors-thesis_code` on the tag `submission`.

Diagonal Covariance

As we assume the noise to be independent, we can safely omit the non-diagonal entries of the covariance $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{V}_0$. That way we further reduce the computational overhead that would otherwise be generated as we still (even with square-root filtering) have to compute the Cholesky decomposition of $\mathbf{Q}$ and $\mathbf{R}$. We can now just take the square-root of the diagonal entries and get the Cholesky decomposition around ten times faster[5].

Regularization and Learning Cholesky Decompositions

There are also a few points where our code is extensible to make it faster, improve numerical stability and so on (see also section 7.3). The main points are regularization and/or learning the Cholesky decomposition directly.

As we will see in chapter 6, having singular and non-positive definite matrices is a big problem that should be assessed. We already implemented the square-root smoothing that increased the numerical stability through the smoothing by not allowing non-positive definite matrices to be produced by the smoothing at all (as we directly learn the Cholesky decomposition). Additional regularization might make sense at the computation of the covariance matrices $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{V}_0$. Another approach would be to learn the Cholesky decomposition of the covariance matrices $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{V}_0$ directly by replacing the explicit formulas with a GD algorithm that directly learn the Cholesky or the square-root diagonal. As the optimization problem may still be convex or close to convex, finding the global optimum should be easily possible.

We also address this in section 7.3 (Future Work).

5.2.2. Hyperparameters

One of the largest hyperparameter with high influence is obviously the architectural choice for $\mathbf{g}_{\boldsymbol{\theta}}$. As it turns out, even simple models like a single hidden layer produce good results as we will see in chapter 6 and 7. A $\tanh$ activation function turns out to be really suitable as it produces smooth results compared to e.g. a step function or Rectified Linear Unit (ReLU). The other high-influence hyperparameter is the dimensionality k of the latent space, i.e. how many dimensions the Koopman operator has to build up a linear system that, taking the observations, behaves the same as the original nonlinear system. We will further investigate into the dependence on this hyperparameter in section 6.1.3.

On the same page is the maximum number of iterations in the numerical optimization of $\boldsymbol{\theta}$. High maximum iterations seem to lead to overfitting on the neural network, causing the whole algorithm to not generalize well. On the other hand, too few iterations do not raise the likelihood enough to get reasonable results.

[5] We ran a experiment on 1000 random 1000×1000 matrices on NumPy's [HMv+20] Cholesky decomposition and square-root implementation. Decomposing all matrices took approx. $5.1\,\mathrm{s}$, taking the square-root took approx. $0.5\,\mathrm{s}$.

6. Experiments

In this chapter we will present all environments we experimented with as well as *hyper-experiments*, i.e. experiments with the hyperparameters and their influence on the algorithm performance (in terms of the prediction error and similar metrics, not the computing time).

We split this chapter into two parts: Firstly, we will present the experiment setup we used and the different environments we tested on. Secondly, we present our results and put them in context of other results. All critical discussion and comparison with related work will take place in chapter 7.

6.1. Setup

In this section we will introduce the experiment setup and all the different environments we used for evaluation.

6.1.1. Environments

This section covers the different environments we used and changes we have applied to them. Note that we separated the data generation process from running the experiment. For each of the following experiments we include the experiment ID at the beginning that can be used to locate the experiment in the code.

Proof-of-Concept: Classic LGDS

- Experiment ID: `lgds`

As a proof-of-concept and to empirically verify our proof on the exactness of the cubature rule and hence the similar expected performance for plain linear systems, we benchmark our algorithm against a simple linear system. The linear system has a two-dimensional state $\mathbf{y} \coloneqq \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T$ with the dynamics

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -x_1 \end{aligned}$$

The initial state $\mathbf{s}_1$ is sampled from a Gaussian distribution with mean $\begin{bmatrix} 0.1 & 0.2 \end{bmatrix}^T$ and covariance $\operatorname{diag}\left(10^{-5}, 10^{-5}\right)$. The system is integrated using the implicit Runge-Kutta Radau IIA [GH01] method with an evaluation interval of $h = 0.1$ for $T = 240$ time steps where only the first $T_{\text{train}} = 120$ steps are used for training and the remaining 120 are used for validation.

As the system's eigenvalues $-i$, i are purely imaginary, the integrated system rings around the equilibrium $\begin{bmatrix} 0 & 0 \end{bmatrix}^T$ indefinitely.

(Damped) Pendulum

- Experiment ID: `pendulum` and `pendulum_damped`

The first nonlinear system we look at is the (inverted) pendulum in both an undamped and a damped setting. For this experiment, we use the angular state $\mathbf{y} = \begin{bmatrix}\theta & \dot{\theta}\end{bmatrix}$ where θ is the displacement angle (see Figure 3.2). The dynamics are specified by the ODE

$$\ddot{\theta} = \sin\theta - d\dot{\theta}$$

that is solved using the Radau IIA [GH01] Initial Value Problem (IVP) integrator (after transforming the ODE to a first-order ODE system). The initial velocity is set to 0 and the initial position is sampled from a Gaussian distribution with mean $\pi/36$ and variance $\pi/8$. This puts the pendulum in motion as it falls down from its initial position. For the undamped pendulum, we set $d = 0$. For both environments, we use an evaluation interval of $h = 0.1$ for $T = 1000$ time steps where only the first $T_{\text{train}} = 500$ steps are used for training and the remaining 500 are used for validation.

The damped pendulum is especially interesting as the system looses energy (i.e. the sum of the kinetic and potential energy decreases over time) and the embedding has to encode this behavior.

Gym Pendulum

- Experiment ID: `pendulum_gym`

This is a sine/cosine version of the pendulum introduced before, but without damping. We use the environment of OpenAI Gym [BCP+16] for generating the data used for training. The motion equations are still the same as for the non-Gym pendulum

$$\ddot{\theta} = \sin\theta$$

with $d = 0$ as the Gym pendulum does not include damping, but the state is defined as the sine and cosine of the angle:

$$\mathbf{y} \coloneqq \begin{bmatrix}\cos\theta \\ \sin\theta \\ \dot{\theta}\end{bmatrix}$$

The Gym environment uses the Euler method for integrating the ODE with a step size of $h = 0.05$. We generate $T = 100$ time steps of which $T_{\text{train}} = 50$ are used for training and the other 50 for validation.

Gym Cartpole

- Experiment ID: `cartpole_gym`

The second to last environment we run experiment on is the cartpole environment. In the cartpole environment, an inverted pendulum is build on a (typically controlled, but otherwise freely movable) cart. If the pendulum falls down, the torque on the joint is translated into a force moving the cart around. This creates nonlinear coupling and therefore, highly nonlinear dynamics. A sketch of the cartpole environment is given in Figure 6.1. As for the previous pendulum, we rely on the cartpole implementation of Gym. We slightly modified the

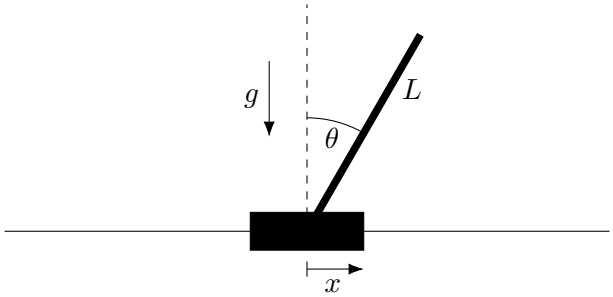


Figure 6.1.: Illustration of the cartpole environment. The cart has mass m_c, the pole m_p with length L. The cart can move freely on the x axis, while the pendulum can swing freely around the center of the cart, causing the cart to move.

environment to be uncontrolled, as the environment usually has discrete actions for pushing the cart left or right. The state of the environment is given as

$$\mathbf{y} \coloneqq \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}$$

where x is the cart position, $\dot{x}$ is the cart velocity, θ is the pole angle and $\dot{\theta}$ is the angular velocity. The implemented equations of movement, taken from [Flo05], are given as:

$$\ddot{\theta} = \frac{g \sin\theta + \cos\theta \left(\frac{-F - m_p \ell \dot{\theta}^2 \sin\theta}{m_c + m_p} \right)}{\ell \left(\frac{4}{3} - \frac{m_p \cos^2\theta}{m_c + m_p} \right)}$$

$$\ddot{x} = \frac{F + m_p \ell \left(\dot{\theta}^2 \sin\theta - \ddot{\theta} \cos\theta \right)}{m_c + m_p}$$

Here $g = 9.8\,\mathrm{m\,s^{-2}}$ is the gravitational acceleration, $m_p = 0.1\,\mathrm{kg}$ and $m_c = 1\,\mathrm{kg}$ are the masses of the pole and cart, respectively, $2\ell = 1\,\mathrm{m}$ is the pole length and $[F] = \mathrm{N}$ is the external (control) force acting on the cart that we set to $0\,\mathrm{N}$. The Gym environment uses an implicit Euler method for integrating the ODE with a step size of $h = 0.02$. We generate $T = 300$ time steps of which we use $T_{\text{train}} = 150$ for training and the remaining 150 steps for validation.

Gym Double Pendulum

- Experiment ID: `acrobot_gym`

The last environment we test on is the double pendulum, implemented in Gym as the *acrobot* (as for the cartpole, we removed all control inputs and modified the initial state to start on top rather than hanging straight down). The double pendulum consists of a pendulum on a fixed joint and a second pendulum attached to the end of the first pendulum. This creates highly nonlinear coupling and is the most common example of a

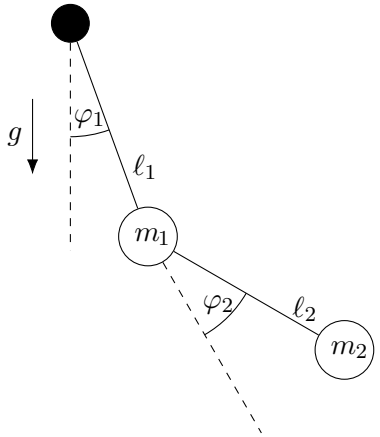


Figure 6.2.: Illustration of the double pendulum environment. The pendulums have lengths ℓ_1 and ℓ_2 with the masses m_1 and m_2 attached to the respective ends. The inner pendulum can swing freely around the center while the other pendulum can swing freely around the end of the inner pendulum. Hence, fixing one of the pendulums would transform the system back to a simple pendulum. If both pendulums can swing, the system is chaotic.

chaotic system [SGWY92]. We observe the state vector

$$\mathbf{y} \coloneqq \begin{bmatrix} \cos\varphi_1 \\ \sin\varphi_1 \\ \cos\varphi_2 \\ \sin\varphi_2 \\ \dot{\varphi}_1 \\ \dot{\varphi}_2 \end{bmatrix}$$

where φ_1 and φ_2 are the displacement of the first and second joint, respectively. See Figure 6.2 for a sketch of the double pendulum. The governing equations of motion are given as:

$$\ddot{\varphi}_1 = \frac{g(\sin\varphi_2\,\cos\varphi_\Delta - \mu\sin\varphi_1) - (\ell_2\dot{\varphi}_2^2 + \ell_1\dot{\varphi}_1^2\cos\varphi_\Delta)\sin\varphi_\Delta}{\ell_1(\mu - \cos^2\varphi_\Delta)}$$
$$\ddot{\varphi}_2 = \frac{g\mu(\sin\varphi_2\,\cos\varphi_\Delta - \mu\sin\varphi_1) + (\mu\ell_1\dot{\varphi}_1^2 + \ell_2\dot{\varphi}_2^2\cos\varphi_\Delta)\sin\varphi_\Delta}{\ell_2(\mu - \cos^2\varphi_\Delta)}$$

with $\varphi_\Delta \coloneqq \varphi_1 - \varphi$, $\mu \coloneqq 1 + m_1/m_2$ where $g = 9.8\,\mathrm{m\,s^{-2}}$ is the gravitational acceleration, $m_1 = 1\,\mathrm{kg}$ and $m_2 = 1\,\mathrm{kg}$ are the masses of the two links and $\ell_1 = 1\,\mathrm{m}$ and $\ell_2 = 1\,\mathrm{m}$ are the lengths of the two links. The Gym environment uses a 4-th Order Runge-Kutta Method (RK4) for integrating the ODE with an evaluation interval of $h = 0.2$. We modified the initial position to be drawn from a Gaussian distribution with mean π and standard deviation $\pi/8$ and the initial velocity to be drawn from a uniform distribution in the interval $[-0.1, 0.1]$. We generate $T = 75$ time steps of which we use $T_{\text{train}} = 56$ (75%) for training and the remaining 19 for validation.

6.1.2. Experiment Setup

We now introduce the experiment setup and initialization we used for each environment. For all experiments, we initialized the latent state dynamics matrix $\mathbf{A}$ with an identity matrix, the initial state $\mathbf{m}_0$ with a one-vector

and all covariance matrices $\mathbf{R}$, $\mathbf{Q}$ and $\mathbf{V}_0$ with a small diagonal covariance of 10^{-5}. We set the maximum number of g-optimization iterations to 1000 for all environments. As described in section 5.2, we used a neural network for the observation function with $\tanh$ activation functions (except for the output layer of course) and a single hidden layer with 50 neurons. We used 100 maximum iterations for the pendulum, 200 for the damped pendulum, 500 for the Gym pendulum, 500 for the Gym cartpole and 250 for the Gym double pendulum.

For the proof-of-concept environment (the linear system), we used a slightly different setup as the environment is a lot simpler. We used a zero-layer neural network with no activation function, i.e. a learnable matrix/linear transformation. We also fixed the latent dimensionality to be 2 for the linear system as we only have two states.

6.1.3. Hyper-Experiments

Besides the performance on specific environments, we were interested in how hyperparameters like the latent dimensionality affect the performance of the system. In this section we will discuss and present our experiment setup for the "hyper-experiments". All hyperparameters that are not part of the experiment where set as in in subsection 6.1.2.

If not stated differently, we always ran the experiment with five different seeds to average out initialization noise.

Influence of the Latent Dimensionality

As the latent dimensionality directly influences how well the Koopman matrix can approximate the infinite-dimensional operator, it is one of the most interesting hyperparameters we evaluate. Hence, we evaluate multiple different latent dimensionalities from $k = 1$ to $k = 50$.

6.2. Results

In this section we will look at the results of the experiments described above. We organized the section into the different environments and will take a look at both the hyper-experiments first and then the individual results for optimized hyperparameters.

For evaluating the performance of the model, we use different measures and metrics, both qualitative and quantitative. For a qualitative evaluation, we simply plot the produced data. These plots can get quite noisy as we will see, but they comprise all things we need for a qualitative assessment:

- The *ground truth*, generated as described above with the adequate numerical integration of the equations of motion.
- The *smoothed states* as they are produced from the E-step of the Koopman Inference algorithm, i.e. $\hat{\mathbf{s}}_{1:T}$.
- The *rollout*, starting both from the learned initial value as well as from the last smoothed state. While the former visualizes the ability of the model to work completely "on its own", the latter shows how well the linear dynamics generalize beyond the training data.

- And of course the confidence (i.e. the learned variance) around each trajectory.

For quantitative comparison, we focus on the Normalized Root Mean Squared Error (NRMSE) across all dimensions and time steps, computed as

$$NRMSE = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{y_{\max}-y_{\min}}\sqrt{\frac{1}{T}\sum_{t=1}^{T}(y_{t,i}-\tilde{y}_{t,i})^2}$$

where $y_{t,i}$ is the true data and $\tilde{y}_{t,i}$ is the corresponding data to evaluate at the t-th time step of the i-th dimension. We can compute four different error values: The error over the rollout from start to finish ($t = 1, 2, \cdots, T$), over the training data only ($t = 1, 2, \cdots, T_{\text{train}}$), over the prediction only ($t = T_{\text{train}}, T_{\text{train}} + 1, \cdots, T$) or over the smoothed trajectory (i.e. the results from the E-step, $\hat{\mathbf{s}}_{1:T}$). The latter primarily checks that our algorithm performs correct and that it is even capable of learning the dynamics and it should be near zero or at least below one for a decent parameter choice and model performance. On the other hand, the NRMSE over the complete rollout describes the generalization abilities. In conjunction with the rollout error over the training and prediction set, we can see which rollout primarily caused the complete error. We expected a low error on the training set for every environment.

Using the NRMSE instead of the plain Root Mean Squared Error (RMSE) allows us to compare model performance across the difference dimensions as we are scaling the error to a reasonable interval. For error values below 0.01, we say that the error is close to zero as it only deviates at most 1% from the true data. For < 0.1 we say it is a good fit, and for < 0.2 we say the approximation is decent. However, we have to take these numbers with care as they summarize a lot of data into a single scalar.

All subsections (of non-proof-of-concept environments) are separated into two main parts: Looking at the hyper-experiment of the latent dimensionality and inspect how well the model performs with different latent dimensionalities. Then we will pick two or three of those latent dimensionalities and look at the qualitative plots of the runs (exemplary evaluation).

Each exemplary evaluation also contains the ID of the run containing the data we used and the raw data of the runs is added to the GitHub repository along with the code.

6.2.1. Proof-of-Concept: LGDS

For the LGDS environment, we did not run multiple latent dimensionalities as we already know how many states we have. Figure 6.3 shows the rollout plot for $k = 3$ latent dimensions. As expected, both the rollout and the smoothed states perfectly match the true data, even in the prediction time span.

6.2.2. Pendulum

Influence of the Latent Dimensionality

For the pendulum experiment, we tested 50 latent dimensionalities from $k = 1$ to $k = 50$.

We start by having a first look at the NRMSE of the smoothed trajectory in Figure 6.4 to see how many latent dimensions we need at least to get a model that is even slightly capable of learning the pendulum dynamics. We see that the NRMSE shrinks to near zero (less than 0.01). Even for $k = 1$, the error is only approximately 0.18.

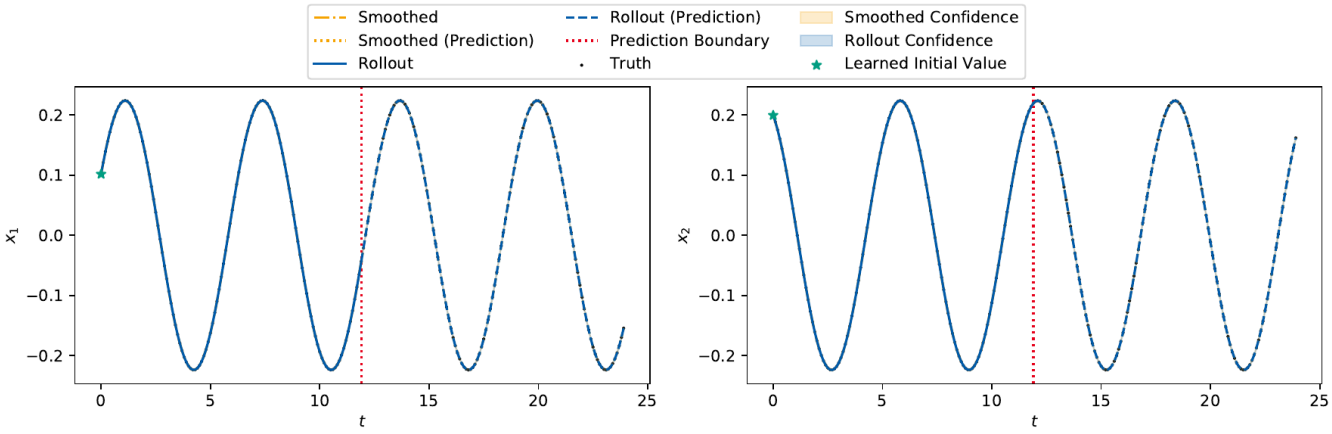


Figure 6.3.: The rollout plot in the observation space of the LGDS environment for $k = 3$. The left plot shows the first dimension, the right plot the second dimension. The black dots represent the true data of which the model used everything till the red prediction boundary to train on. The blue line is the rollout, starting from the learned initial value (marked with a green star). The orange dash-dotted line is the smoothed data. The dotted orange line then is the rollout starting from the last smoothed state, forming the "smoothed prediction". The orange lines are directly behind the blue ones, hence they are not visible. The shaded regions show the confidence, i.e. two times the standard deviation.

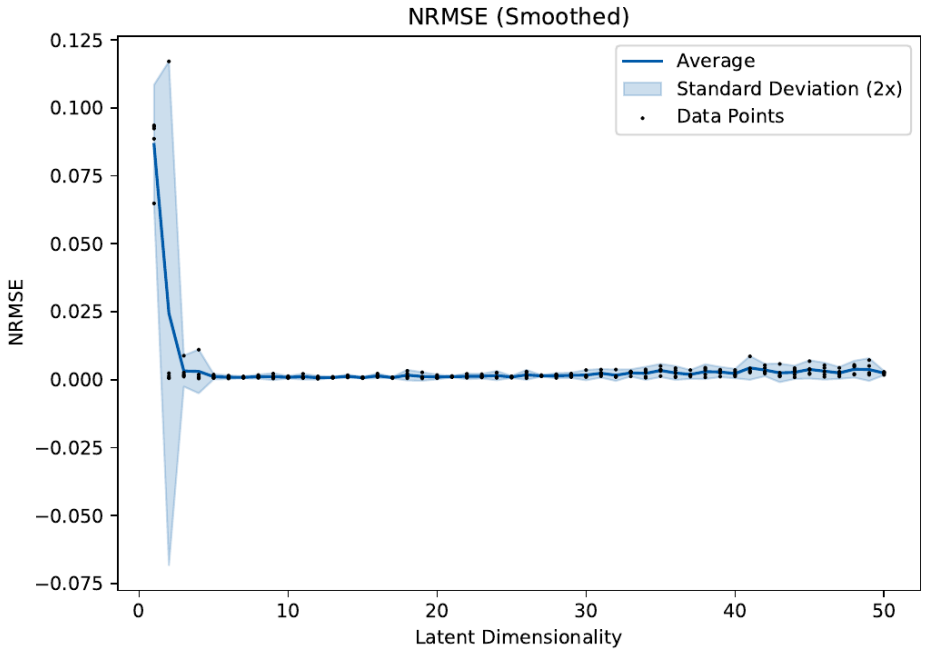


Figure 6.4.: The NRMSE of the smoothed trajectory on the training data of the pendulum environment.

Looking at the NRMSE of the complete rollout in Figure 6.5a, we see that the error is quite noisy due to initialization. Nevertheless, the error shrinks until $k \geq 10$ and then stabilizes around an error of approximately 0.15. Comparing the complete rollout error with the training and prediction NRMSE in Figure 6.5b and Figure 6.5c, respectively, we see that the training error falls until $k \geq 10$ and then stabilizes around 0.08. For the prediction error, we see a similar connection to $k \geq 10$, but the error does not shrink as much and always is around 0.20.

Exemplary Evaluation: 2-Dimensional Latent

We now look at an exemplary run for the latent dimensionality $k = 2$ (run `100`). Looking at the errors, the rollout trajectory should be off the real data, but the smoothed trajectory should nearly match the true data. Figure B.5 shows that the rollout and also the rollout of the smoothed trajectory are far off with low confidence, but the true data is in the confidence region. However, the smoothed trajectory follows the true data really good.

Exemplary Evaluation: 10-Dimensional Latent

We now look at an exemplary run for the latent dimensionality $k = 10$ (run `10`) as we postulated from the NRMSE data that this is the boundary, after which we expected diminishing returns. Figure 6.6 shows the rollout of the model, where the dynamics of the system are captured quite well with high confidence. Also the prediction is good, so that the differences between true data, smoothed prediction and rollout are small.

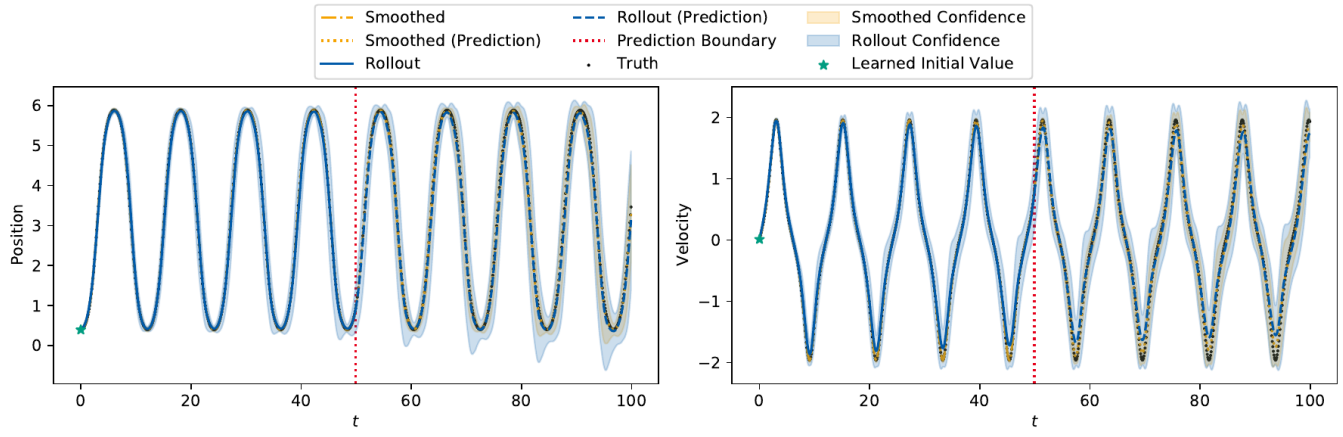
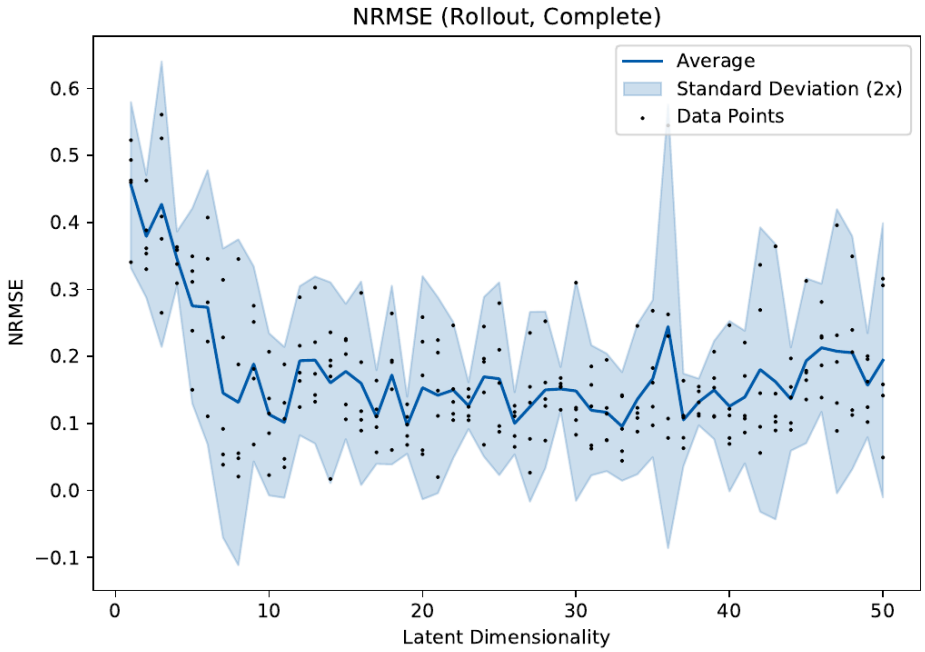
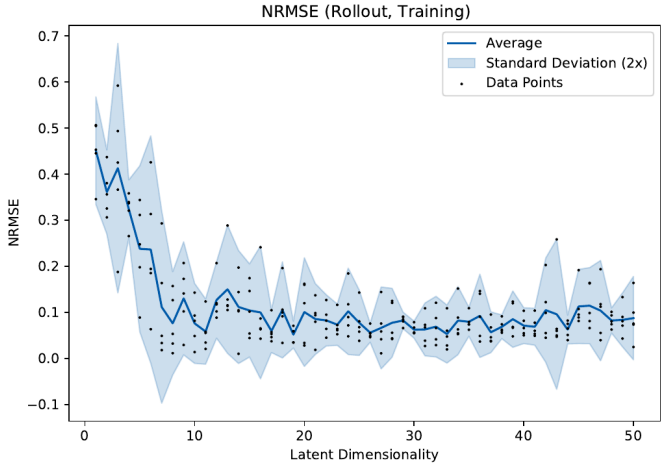


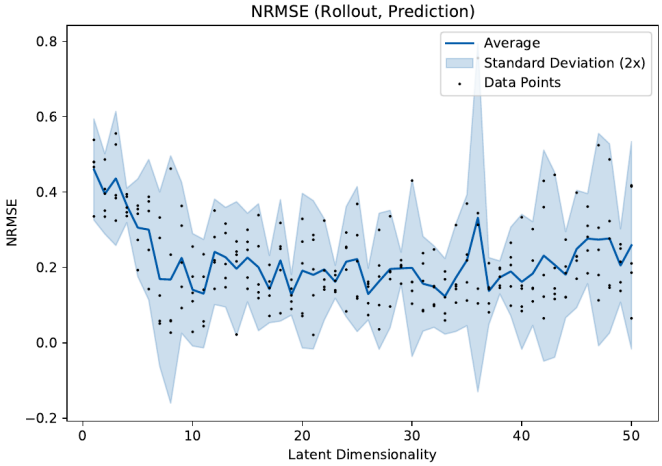
Figure 6.6.: The rollout plot in the observation space of the pendulum environment for $k = 10$. The left plot shows the displacement and the right plot the angular velocity. The black dots represent the true data of which the model used everything till the red prediction boundary to train on. The blue line is the rollout, starting from the learned initial value (marked with a green star). The orange dash-dotted line is the smoothed data. The dotted orange line then is the rollout starting from the last smoothed state, forming the "smoothed prediction". The shaded regions show the confidence, i.e. two times the standard deviation.



(a) Error of the complete rollout on the pendulum environment.



(b) Error of the rollout on the training data only on the pendulum environment.



(c) Error of the rollout on the prediction only on the pendulum environment.

Figure 6.5.: Plot of the errors of the pendulum environment for different latent dimensions. The black dots show the measured data, the blue line the average of the data points for a specific latent dimensionality. The blue shaded region shows two times the standard deviation.

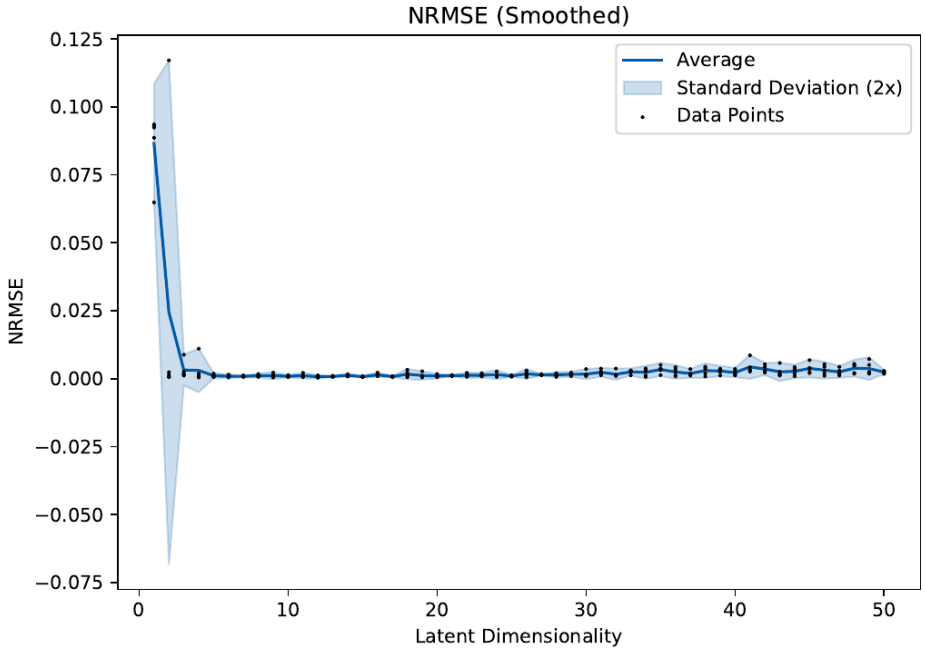


Figure 6.7.: The NRMSE of the smoothed trajectory on the training data of the damped pendulum environment.

Exemplary Evaluation: 14-Dimensional Latent

We now look at an exemplary run for the latent dimensionality $k = 14$ (run `210`) for which we got the smallest overall NRMSE. Figure B.6 shows the rollout of the model. In comparison to the 10-dimensional latent, nothing has changed, fulfilling the expectation of diminishing returns with high latent dimensionality.

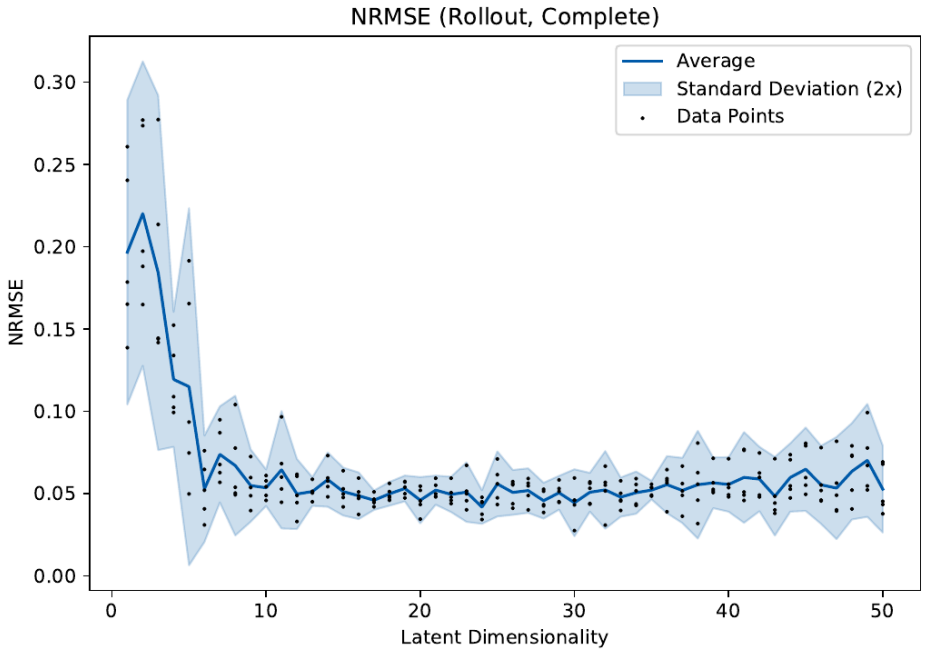
6.2.3. Damped Pendulum

Influence of the Latent Dimensionality

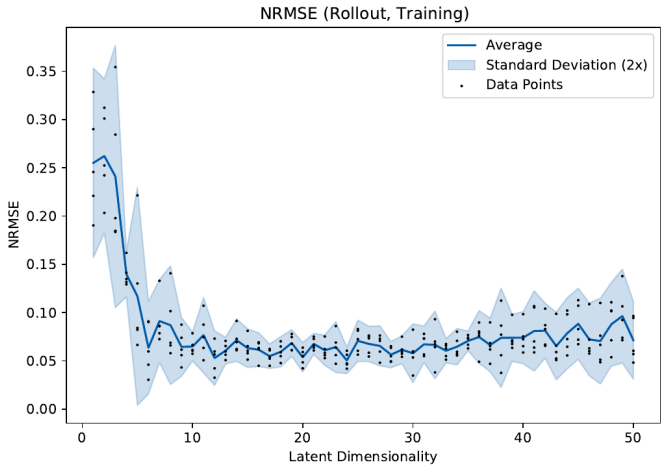
For the damped pendulum experiment, we tested 50 latent dimensionalities from $k = 1$ to $k = 50$.

We start by having a first look at the NRMSE of the smoothed trajectory in Figure 6.7 to see how many latent dimensions we need at least to get a model that is even slightly capable of learning the damped pendulum dynamics. We see that the NRMSE shrinks to near zero (less than 0.01) for latent dimensionalities of $k \geq 2$. Even for $k = 1$, the error is only approximately 0.09.

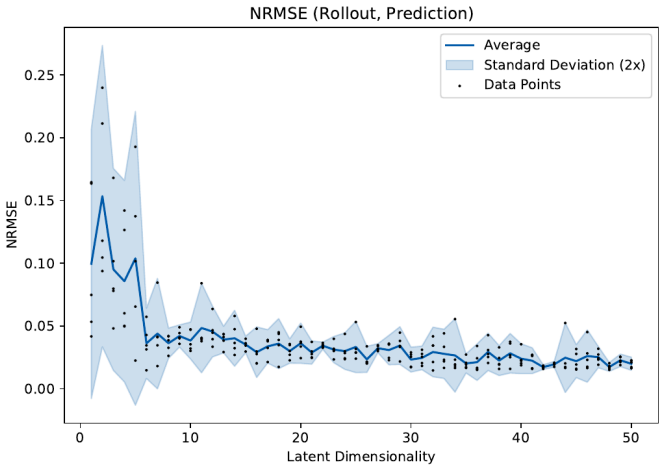
Looking at the NRMSE of the complete rollout in Figure 6.8a, we see that there is a point of diminishing returns in terms of the latent dimensionality for $k \geq 10$, where we get decent errors (around 0.05) for all latent dimensionalities. Looking at the parts the complete NRMSE is composed of, the training and prediction error in Figure 6.8b and Figure 6.8c, respectively, we see that the latent dimensionality affects both parts, the training and prediction error. However, the prediction error is generally smaller, which is mostly caused by the value of the state and hence the rollout being smaller.



(a) Error of the complete rollout on the damped pendulum environment.



(b) Error of the rollout on the training data only on the damped pendulum environment.



(c) Error of the rollout on the prediction only on the damped pendulum environment.

Figure 6.8.: Plot of the errors of the damped pendulum environment for different latent dimensions. The black dots show the measured data, the blue line the average of the data points at a specific latent dimensionality. The blue shaded region shows two times the standard deviation.

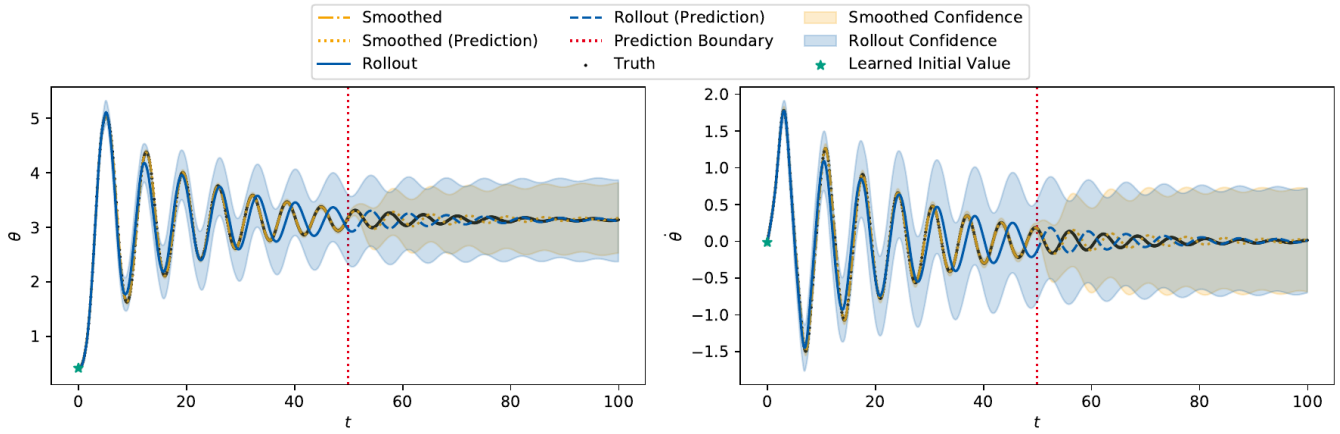


Figure 6.9.: The rollout plot in the observation space of the damped pendulum environment for $k = 10$. The left plot shows the displacement and the right plot the angular velocity. The black dots represent the true data of which the model used everything till the red prediction boundary to train on. The blue line is the rollout, starting from the learned initial value (marked with a green star). The orange dash-dotted line is the smoothed data. The dotted orange line then is the rollout starting from the last smoothed state, forming the "smoothed prediction". The shaded regions show the confidence, i.e. two times the standard deviation.

Exemplary Evaluation: 2-Dimensional Latent

We now look at an exemplary run for the latent dimensionality $k = 2$ (run `2`). According to the errors, the rollout error should be quite high, but the smoothed trajectory should follow the true data really good. Figure B.7 shows that most trajectories are far off the true trajectory with low confidence, while the smoothed trajectory almost perfectly matches the true data.

Exemplary Evaluation: 10-Dimensional Latent

We now look at an exemplary run for the latent dimensionality $k = 10$ (run `110`). We chose this latent dimensionality as it is the boundary of diminishing returns in terms of the NRMSE. Figure 6.9 shows the rollout of the model. In comparison to the four-dimensional latent, the behavior of the system is captured pretty good, with a decent confidence. Also, the prediction is at nearly the same amplitude as the real data. But the rollout gets phase-shifted in comparison to the true data further in the rollout horizon.

Exemplary Evaluation: 24-Dimensional Latent

We now look at an exemplary run for the latent dimensionality $k = 30$ (run `130`) for which we got the smallest overall NRMSE. Figure B.8 shows the rollout of the model. In comparison to the ten-dimensional latent, the model is less confident but the rollout is nearly the same as the rollout of the ten-dimensional latent.

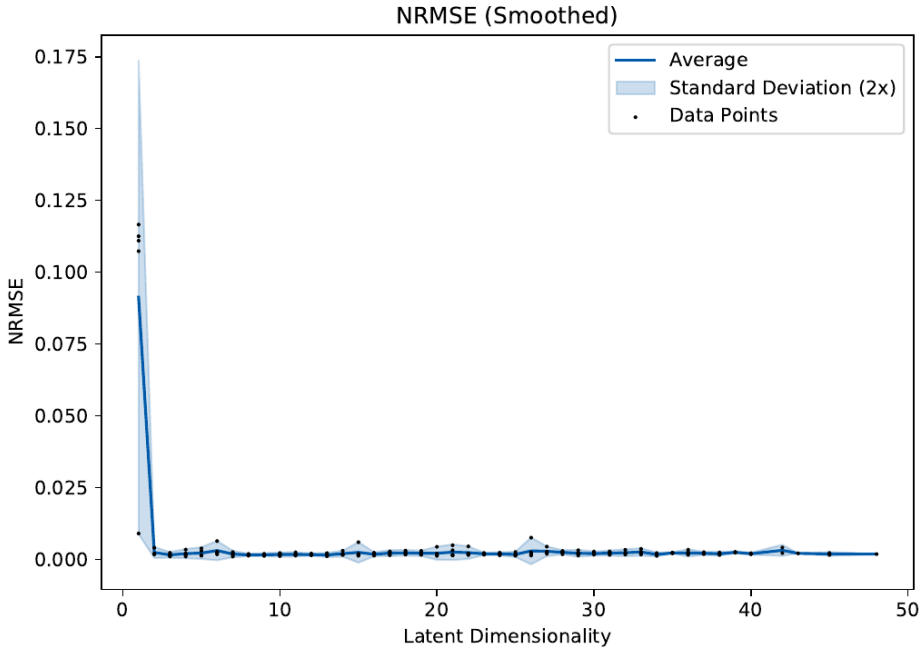


Figure 6.10.: The NRMSE of the smoothed trajectory on the training data of the Gym pendulum environment.

6.2.4. Gym Pendulum

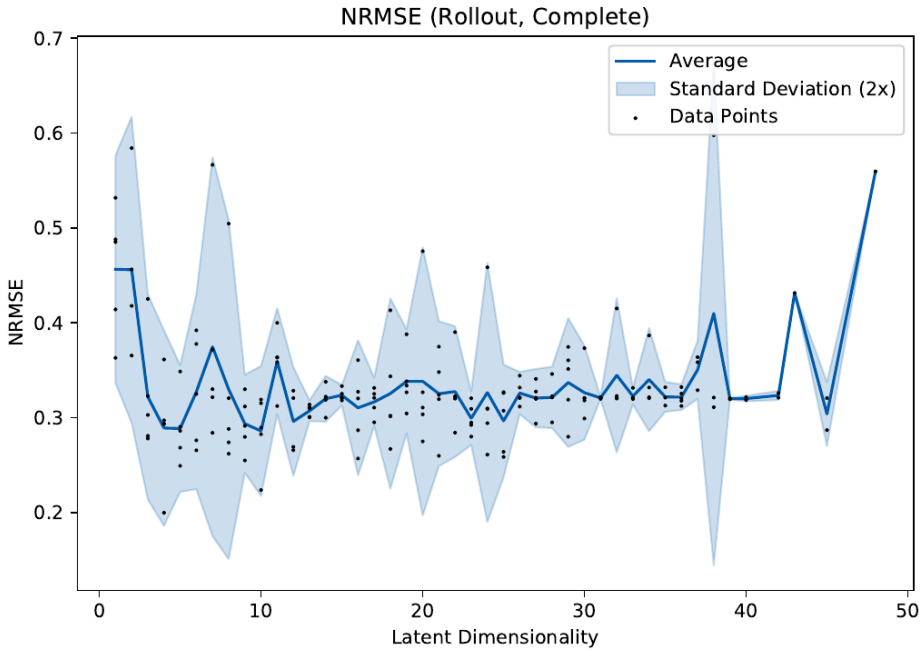
Influence of the Latent Dimensionality

For the Gym pendulum experiment, we tested 50 latent dimensionalities from $k = 1$ to $k = 50$. The difference of the Gym pendulum in comparison to the pendulum from before is that we do not measure the angle directly but use the sine/cosine of the angle, directly encoding the symmetry of a swing-around into the features. This environment is extremely numerically brittle, thus we have less data points for higher latent dimensionalities (to the right of the plots).

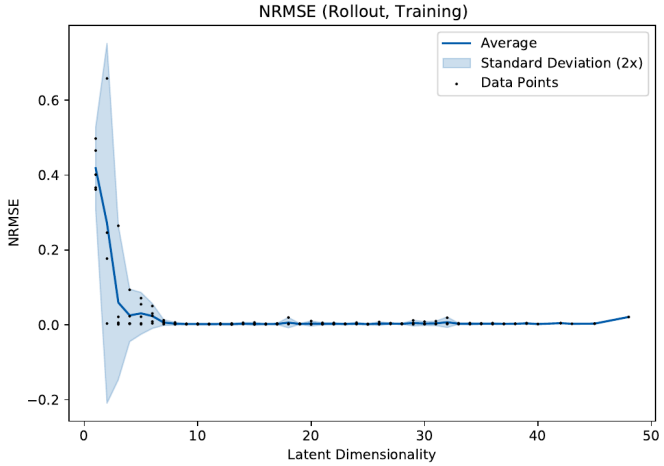
We start by having a first look at the NRMSE of the smoothed trajectory in Figure 6.10 to see how many latent dimensions we need at least to get a model that is even slightly capable of learning dynamics. We see that the NRMSE shrinks to near zero (less than 0.01) for latent dimensionalities of $k \geq 2$. Even for $k = 1$, the error is only approximately 0.09.

Looking at the NRMSE of the complete rollout in Figure 6.11a, we see a slight improvement at the beginning, which is, looking at the training and prediction error in Figure 6.11b and Figure 6.11c, respectively, fully caused by the training error. Looking at the prediction error, we see that the error does not change much with different latent dimensionalities[1]. In comparison, the training error shrinks to near zero (less than 0.01) for latent dimensionalities $k \geq 7$. Hence, we expect the model not to predict well behind the prediction boundary.

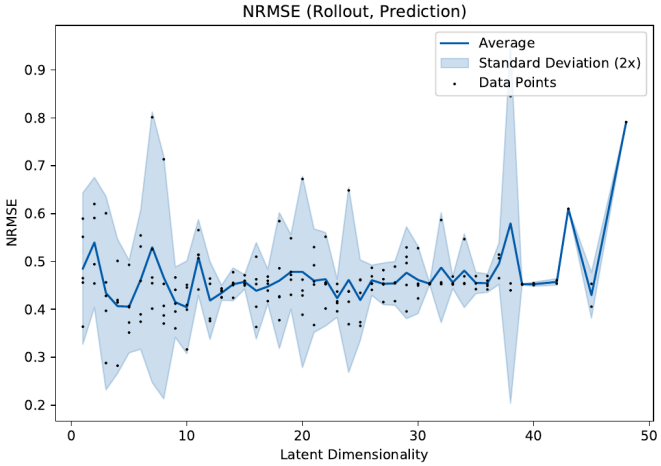
[1] The spike on the right is possibly due to bad initialization and is only chosen as the mean as all other seeds crashed due to negative definite matrices.



(a) Error of the complete rollout on the Gym pendulum environment.



(b) Error of the rollout on the training data only on the Gym pendulum environment.



(c) Error of the rollout on the prediction only on the Gym pendulum environment.

Figure 6.11.: Plot of the errors of the Gym pendulum environment for different latent dimensions. The black dots show the measured data, the blue line the average of the data points at a specific latent dimensionality. The blue shaded region shows two times the standard deviation.

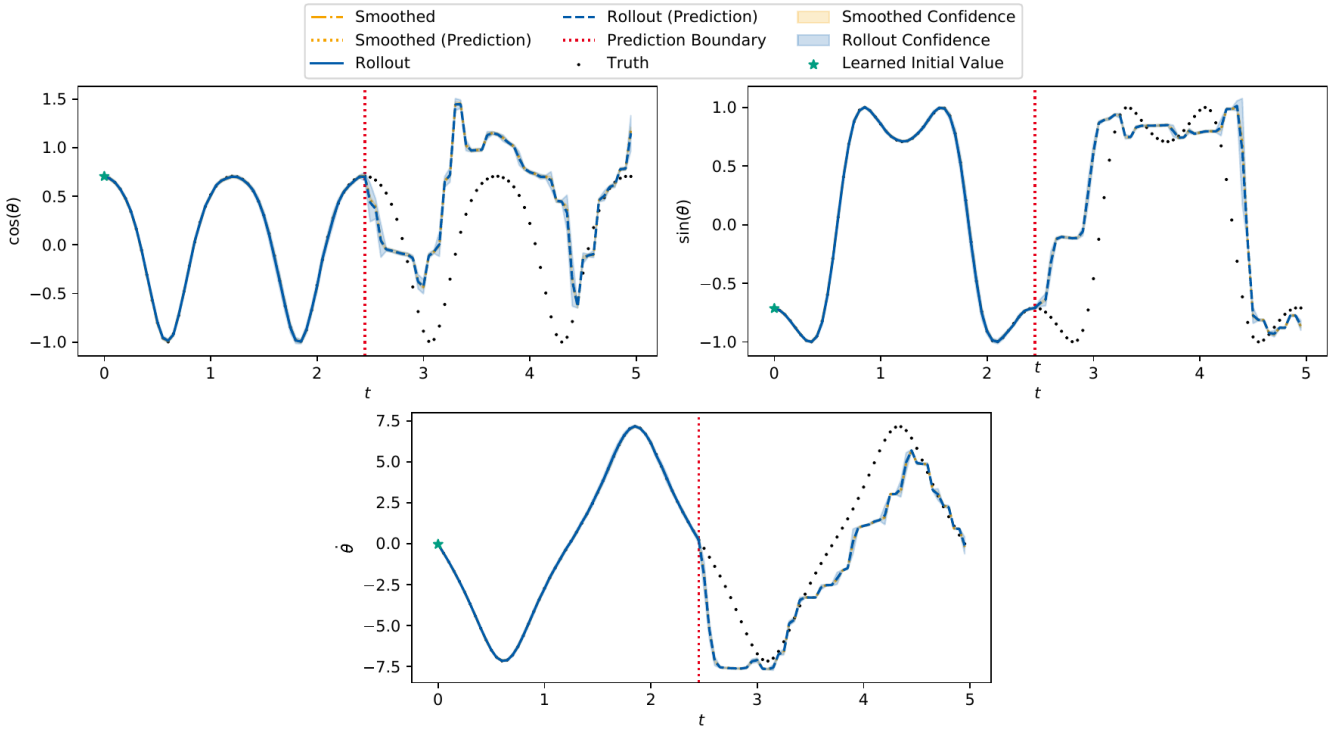


Figure 6.12.: The rollout plot in the observation space of the Gym pendulum environment for $k = 4$. The left plot shows the displacement and the right plot the angular velocity. The black dots represent the true data of which the model used everything till the red prediction boundary to train on. The blue line is the rollout, starting from the learned initial value (marked with a green star). The orange dash-dotted line is the smoothed data. The dotted orange line then is the rollout starting from the last smoothed state, forming the "smoothed prediction". The shaded regions show the confidence, i.e. two times the standard deviation.

Exemplary Evaluation: 2-Dimensional Latent

We now look at an exemplary run for the latent dimensionality $k = 2$ (run `2`). According to the errors, the rollout trajectory should be off the true data, but the smoothed trajectory should be mostly on the true data. Figure B.9 shows exactly this behavior: all trajectories are far sway from the true trajectory with the exception of the smoothed trajectory that almost matches the true data.

Exemplary Evaluation: 4-Dimensional Latent

We now look at an exemplary run for the latent dimensionality $k = 4$ (run `104`). This run is especially interesting for comparison with [MWK19] which we will do in chapter 7. Figure 6.12 shows the rollout of the model with four latent dimensions. We see that the smoothed trajectory equals the true data as expected and so does the training rollout. However, on the prediction the rollout only roughly follows the true data, not capturing the dynamics of the system well.

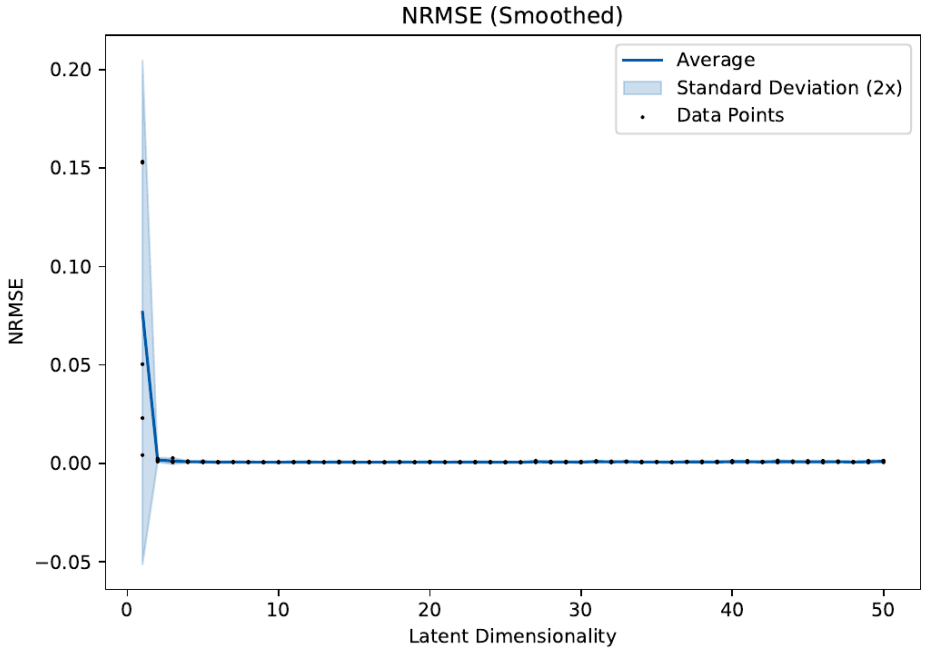


Figure 6.13.: The NRMSE of the smoothed trajectory on the training data of the cartpole environment.

Exemplary Evaluation: 7-Dimensional Latent

We now look at an exemplary run for the latent dimensionality $k = 7$ (run 157) as we postulated from the NRMSE data that this is the boundary of diminishing returns. Figure B.10 shows the rollout of the model. We see that the training rollout and the smoothed trajectory fits the true data well, as expected, but the prediction does not capture the dynamics of the system.

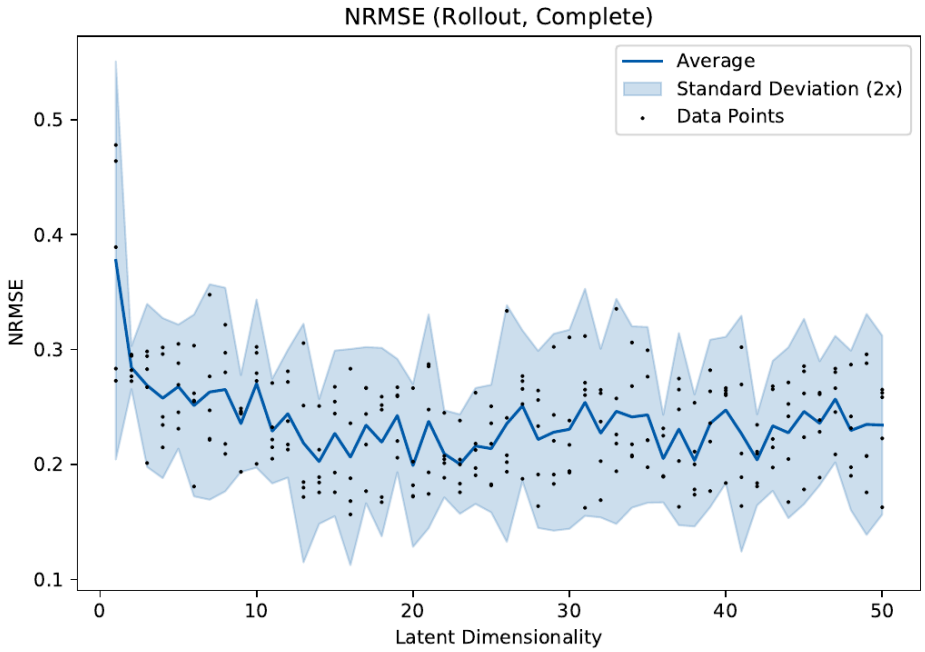
6.2.5. Gym Cartpole

Influence of the Latent Dimensionality

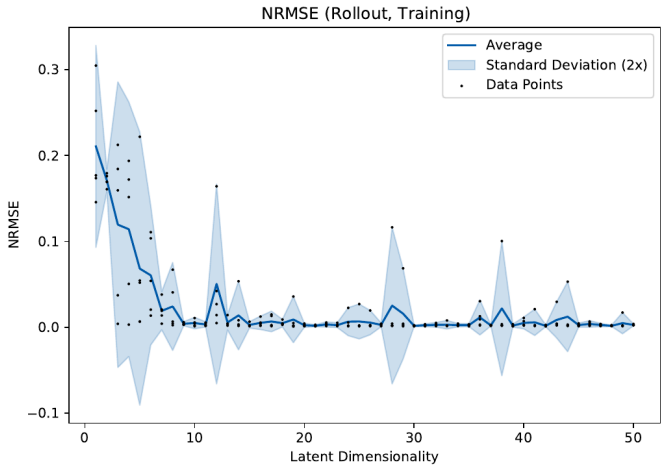
For the cartpole experiment, we tested 50 latent dimensionalities from $k = 1$ to $k = 50$.

We start by having a first look at the NRMSE of the smoothed trajectory in Figure 6.13 to see how many latent dimensions we need at least to get a model that is even slightly capable of learning the cartpole dynamics. We see that the NRMSE shrinks to near zero (approximately 0.001) for latent dimensionalities of $k \geq 2$. Hence, we need at least 2 latent dimensions to learn the dynamics.

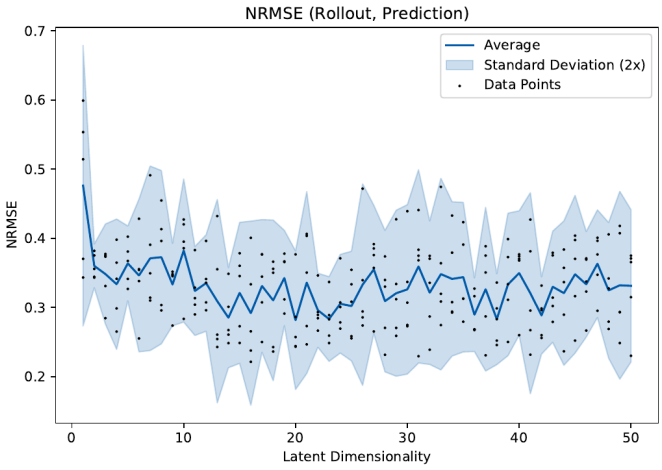
Looking at the NRMSE of the complete rollout in Figure 6.14a, we see that the error does not shrink much any more for latent dimensionalities of $k \geq 10$, where the error stabilizes around 0.23. Looking at the training and prediction plots in Figure 6.14b and Figure 6.14c, respectively, we see that the training error falls by a large margin until $k \geq 10$ latent dimensions where the error stabilized near zero, while the prediction error does not fall much after the first few latent dimensionalities. Hence, we can predict that our model learns the training rollout well, but does not generalize beyond the prediction boundary.



(a) Error of the complete rollout from on the cartpole environment.



(b) Error of the rollout on the training data only on the cartpole environment.



(c) Error of the rollout on the prediction only on the cartpole environment.

Figure 6.14.: Plot of the errors of the cartpole environment for different latent dimensions. The black dots show the measured data, the blue line the average of the data points at a specific latent dimensionality. The blue shaded region shows two times the standard deviation. The data points on the right side of the plot get increasingly sparse as the algorithm gets increasingly numerically brittle for higher latent dimensionalities.

Exemplary Evaluation: 2-Dimensional Latent

We now look at an exemplary run for the latent dimensionality $k = 2$ (run `2`). As we have already seen, the rollout should not perform too god, but the smoothed trajectory should be really accurate. Looking at Figure B.11, we see that nearly all trajectories are off the true trajectory. Only the smoothed trajectory fits the data perfectly (until the prediction boundary). We can also see that the rollout starts to go near the true data, learning roughly how the true data looks.

Exemplary Evaluation: 10-Dimensional Latent

We now look at an exemplary run for the latent dimensionality $k = 10$ (run `10`). According to the NRMSE of the training data, this should perform decently on the training data and, according to the NRMSE, bad on the prediction. Figure 6.15 shows that the rollout and the smoothed trajectory actually follow the true data, while the rollout does not match the true data after the prediction boundary, but roughly forecasts the qualitative behavior. Interestingly, the confidence in the rollout of the pendulum displacement and velocity is a lot higher than the confidence in the rollout of the cart position and velocity.

Exemplary Evaluation: 16-Dimensional Latent

We now look at an exemplary run for the latent dimensionality $k = 16$ (run `166`), the run which had the lowest NRMSE over the complete rollout data. Still, according to the NRMSE of the prediction, we do not expect a good prediction. Figure B.14 confirms this, as the rollout on the training data follows the true data closely while the prediction is off the true trajectory. Compared to the 14-dimensional latent, we do not see much improvement in the prediction.

6.2.6. Gym Double Pendulum

Influence of the Latent Dimensionality

For the double pendulum experiment, we tested 50 latent dimensionalities from $k = 1$ to $k = 50$.

We start by having a first look at the NRMSE of the smoothed trajectory in Figure 6.16 to see how many latent dimensions we need at least to get a model that is even slightly capable of learning the double pendulum dynamics. We see that the NRMSE shrinks below 0.05 for latent dimensionalities of $k \geq 3$, so we need at least 3 latent dimensions to get a decent model.

Looking at the NRMSE of the complete rollout in Figure 6.17a, we see that the error shrinks until $k \geq 6$ latent dimensions and than stabilizes to an error around 0.27. Looking at the training and prediction error in Figure 6.17b and Figure 6.17c, we see that mostly only the training error shrinks while the prediction errors seems to be random noise. For the training error, however, we see that the error shrinks until $k \geq 18$ and then stabilized around 0.17. Hence, we can assume that our model does not generalize well (we will assess this in chapter 7). On the other hand, our rollout on the training data should be a good fit.

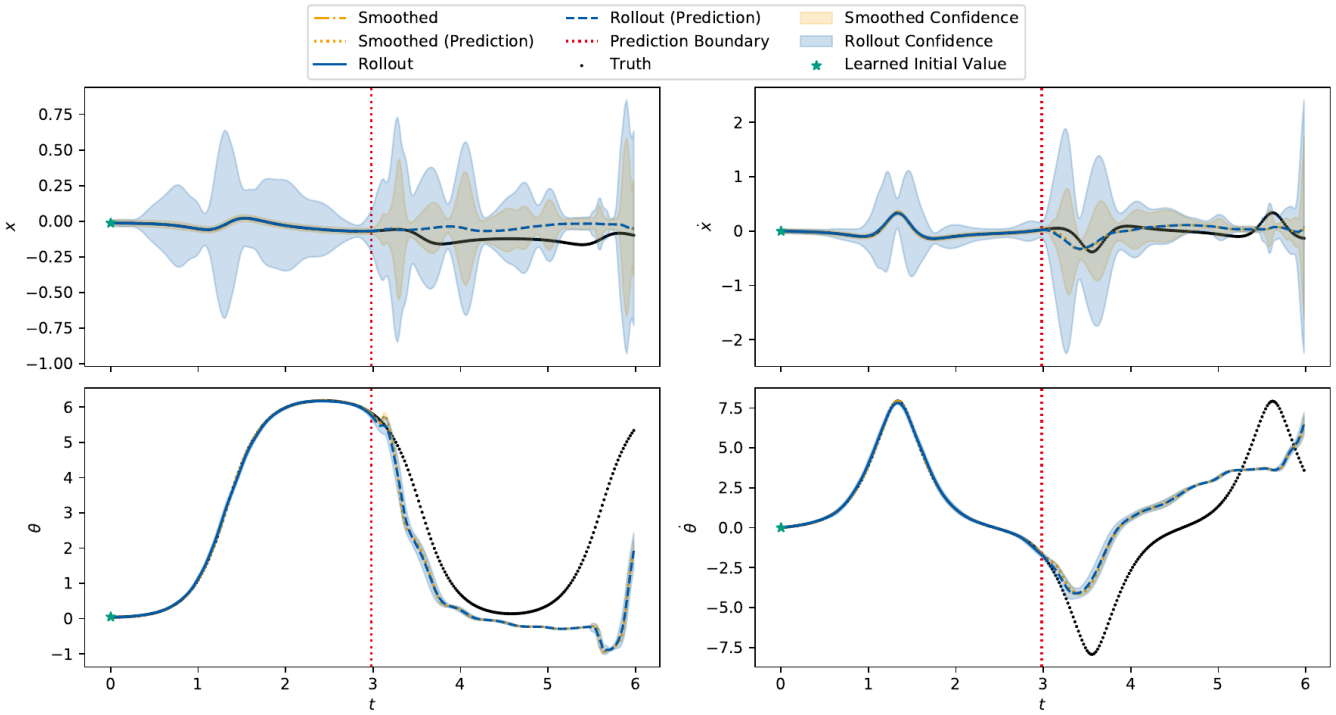


Figure 6.15.: The rollout plot in the observation space of the cartpole environment for $k = 10$. The top plot is the cart position (left) and velocity (right), the row the pole displacement (left) and angular velocity (right). The black dots represent the true data of which the model used everything until the red prediction boundary to train on. The blue line is the rollout, starting from the learned initial value (marked with a green star). The orange dash-dotted line is the smoothed data. The dotted orange line then is the rollout starting from the last smoothed state, forming the "smoothed prediction". The shaded regions show the confidence, i.e. two times the standard deviation. As the variances in the top plots are very high, see Figure B.13 in section B.3 for the plot without the confidence.

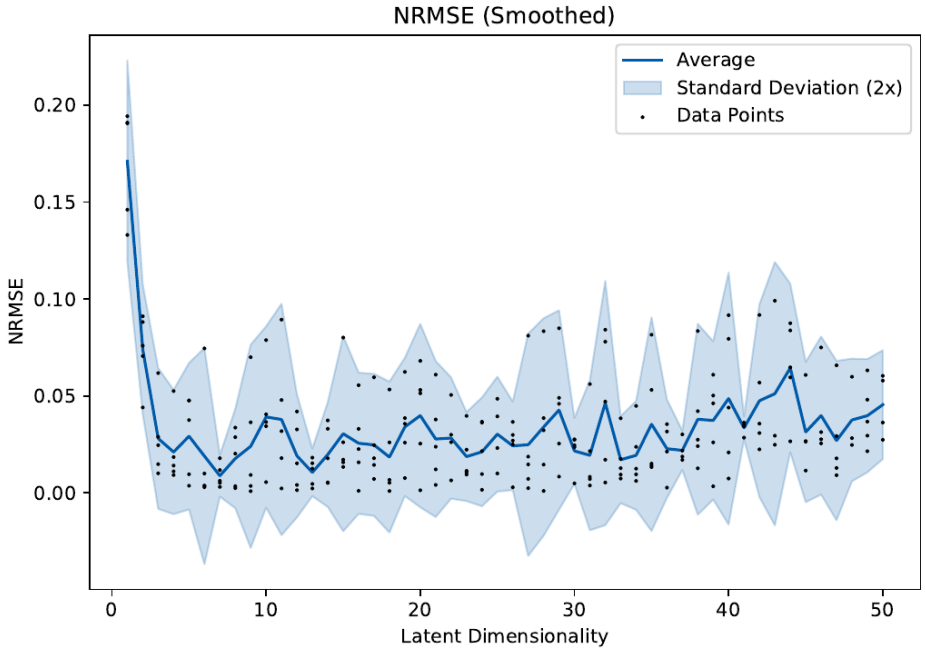


Figure 6.16.: The NRMSE of the smoothed trajectory on the training data of the double pendulum environment.

Exemplary Evaluation: 3-Dimensional Latent

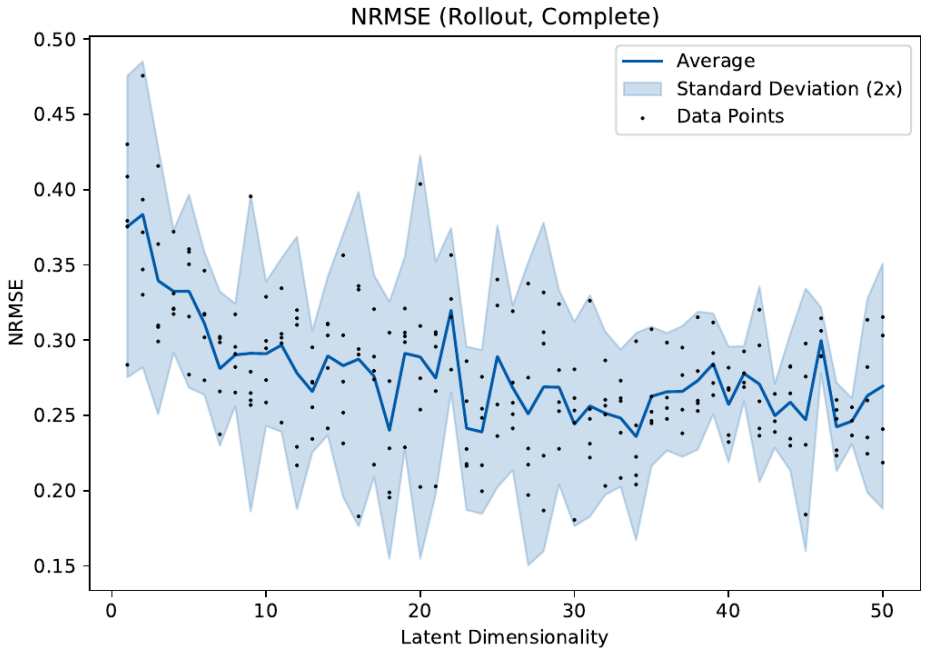
We now look at an exemplary run for the latent dimensionality $k = 3$ (run `153`). As we have already seen, the rollout should not perform well, but the smoothed trajectory should be really accurate. Looking at Figure B.16, we see that nearly all trajectories do not match the true trajectory. Only the smoothed trajectory follows the data decently (until the prediction boundary, of course).

Exemplary Evaluation: 18-Dimensional Latent

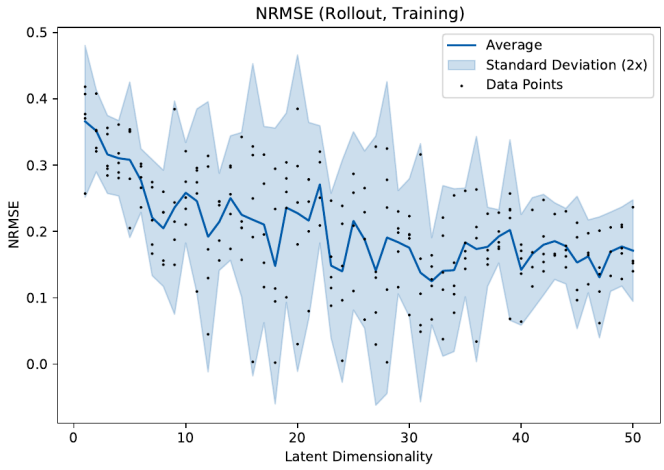
We now look at an exemplary run for the latent dimensionality $k = 18$ (run `168`). We chose this latent dimensionality as it is at the boundary before the training error does not get much batter in terms of the NRMSE. Figure 6.18 shows the rollout of the model. We see that the rollout on the training trajectory nearly perfectly follows the true data, while the prediction is quite off the true trajectory, not even capturing some of the motion. This is especially bad as the model is quite confident about the state.

Exemplary Evaluation: 30-Dimensional Latent

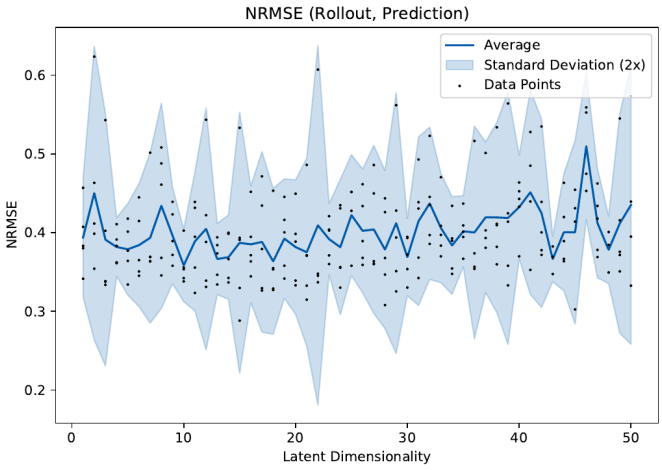
We now look at an exemplary run for the latent dimensionality $k = 30$ (run `80`) for which we got the smallest overall NRMSE. Figure B.17 shows the rollout of the model. In comparison to the 18-dimensional latent, the rollout is not nearly as good. However, the low confidence in the prediction produces is better than for the 18-dimensional latent as more parts of the true trajectory lie within the region of confidence.



(a) Error of the complete rollout from on the double pendulum environment.



(b) Error of the rollout on the training data only on the double pendulum environment.



(c) Error of the rollout on the prediction only on the double pendulum environment.

Figure 6.17.: Plot of the errors of the double pendulum environment for different latent dimensions. The black dots show the measured data, the blue line the average of the data points at a specific latent dimensionality. The blue shaded region shows two times the standard deviation. The data points on the right side of the plot get increasingly sparse as the algorithm gets increasingly numerically brittle for higher latent dimensionalities.

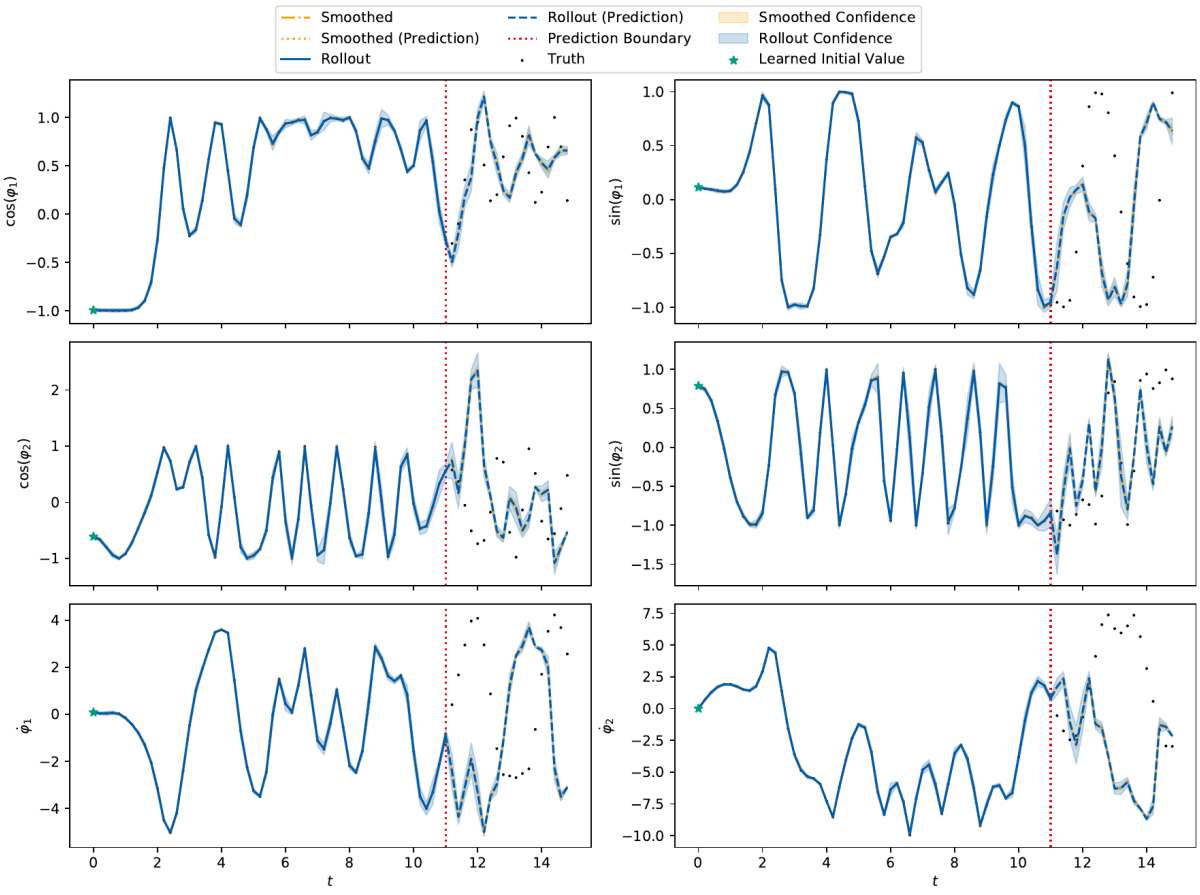


Figure 6.18.: The rollout plot in the observation space of the double pendulum environment for $k = 18$. The top row shows the cosine/sine of the displacement of the inner pendulum, the middle row shows the cosine/sine of the displacement of the outer pendulum and the bottom row shows the angular velocity of the inner and outer pendulum. The black dots represent the true data of which the model used everything till the red prediction boundary to train on. The blue line is the rollout, starting from the learned initial value (marked with a green star). The orange dash-dotted line is the smoothed data. The dotted orange line then is the rollout starting from the last smoothed state, forming the "smoothed prediction". The shaded regions show the confidence, i.e. two times the standard deviation.

7. Discussion

In this chapter we will analyze the results from chapter 6 and set them into context with results from related work. We will discuss the performance on the different environments first and then compare our results with the results of some related work.

7.1. Model Performance

7.1.1. Proof-of-Concept: LGDS

As we have seen in Figure 6.3, the LGDS proof-of-concept environment works pretty well and the prediction perfectly matches the true data. The model has an NRMSE of approximately 0.0001 for every metric. This shows that we are still able to learn linear dynamical systems. We expected this result as we have proven that our algorithm is equivalent to the LGDS inference algorithm for linear systems in section A.2. Hence, our algorithm does not reduce the "power" of the linear algorithm on linear systems and only extends its usability to nonlinear systems.

7.1.2. Pendulum

The pendulum is the first nonlinear environment we assess.

As we have seen in the experiment for different latent dimensionalities, we need at least ten latent dimensions to model the pendulum with a decent prediction error, where we expected some downsides on the prediction and expected a really good rollout in the training area. As we have seen in the results for a ten-dimensional latent in section 6.2.2, we see a good rollout in the training area and also a good prediction for the pendulum position, but a slightly smaller amplitude in the velocity plot. This corresponds to an energy loss of the pendulum that should not happen as it is undamped. Figure 7.1 highlights this by showing the total, kinetic and potential energy of the system, where the rollout energy changes with time which should not happen. However, the potential energy almost matches the true potential energy, supporting our inspection that the angular displacement is well fit. The kinetic energy generated by the velocity is, on the other hand, a bit off. We can further inspect this behavior by looking at the rollout in the latent space in Figure 7.2. We see that the energy loss is primarily happening in the tenth dimension. Looking at the real parts of the eigenvalues of the latent state dynamics matrix,

$$\sigma = \{1.00, 1.00, 1.00, 0.96, 0.65, 0.67, 0.86, 0.80, 0.73, 0.75\}$$

we see that not all are close to one, i.e. they vanish after some time. This directly corresponds to the energy loss in the system (the eigenvalues of a system capture the time evolution). We also see that the tenth dimension

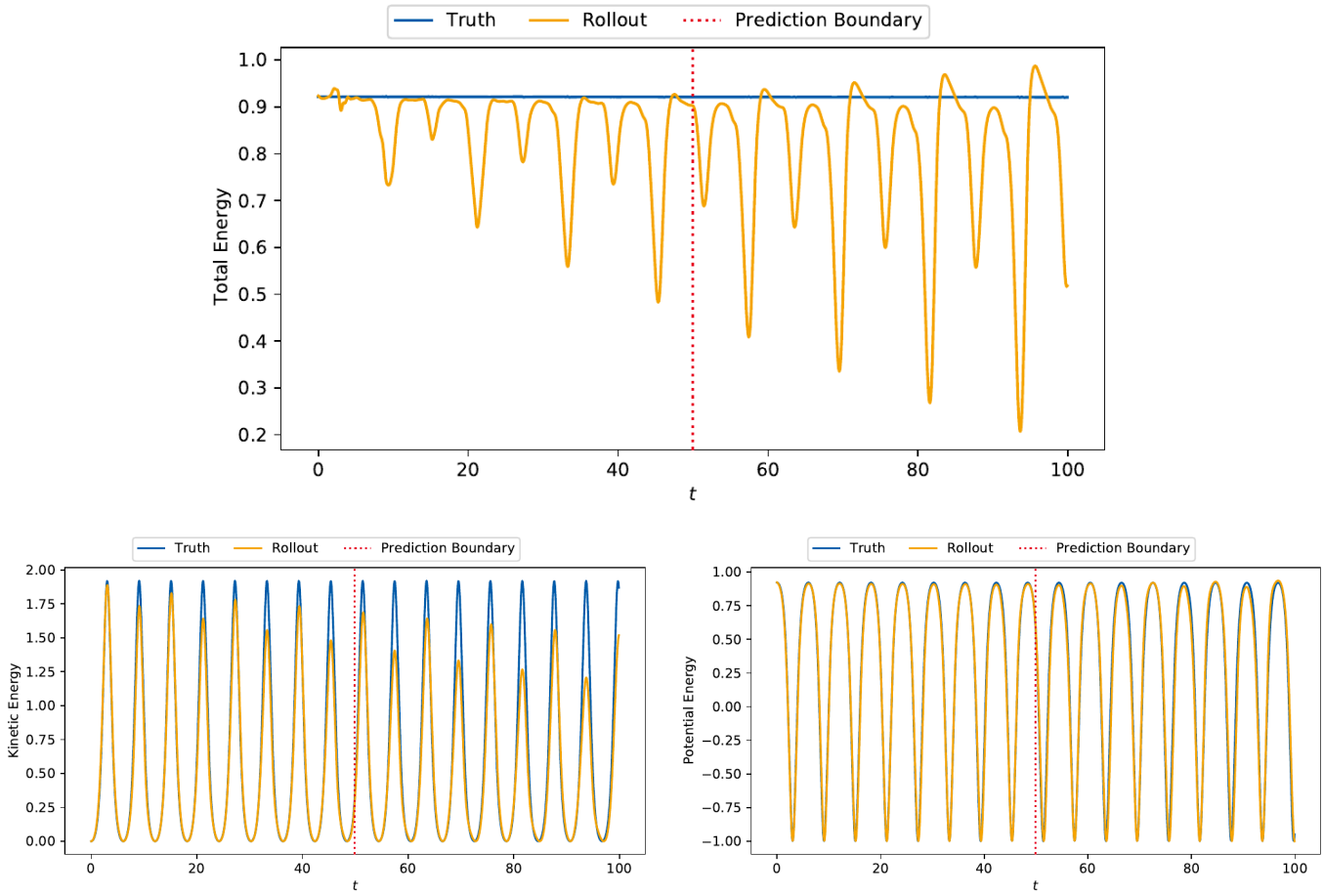


Figure 7.1.: Total energy of the undamped pendulum composed of the kinetic energy on the bottom left and the potential energy on the bottom right. The blue line represents the true energy, calculated from the training and validation data. The orange line is the energy calculated from the rollout. As usual, the red line is the prediction boundary up until all training data was used; afterwards, the model predicted the rest of the data.

has a really high variance in its states, indicating that the model already has enough latent dimensions to explain the dynamics. Looking at the rollout of a nine-dimensional latent (run 58) in Figure 7.3, however, we see that the energy loss is still there. It does not make sense to inspect higher-dimensional latent spaces as we have seen in the latent dimensionalities experiment that the error does not shrink much. Additionally, the 14-dimensional latent that we inspected in section 6.2.2 also looses energy. We do not expect this to get better with higher latent dimensionalities. Also, higher dimensionalities impose numerical instability, a phenomenon we will address in subsection 7.1.9.

For this environment, we get really small confidences with the true trajectory still being in the region of confidence. We also see that the variance is higher on the turns of the pendulum which makes sense as that is the most critical part of the movement.

While these first results cast a shadow on our model performance, the energy loss builds hope for the damped pendulum to work out well.

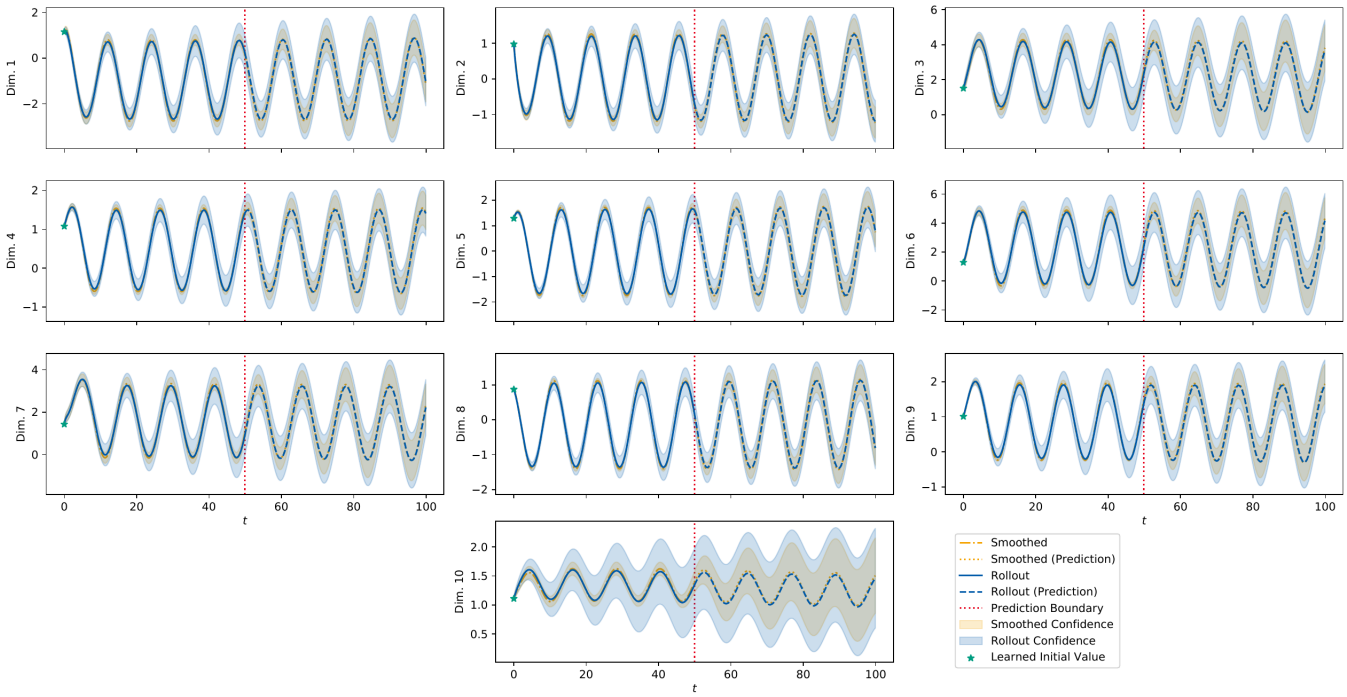


Figure 7.2.: Rollout of the latent dimensions of the pendulum environment for $k = 10$ latents.

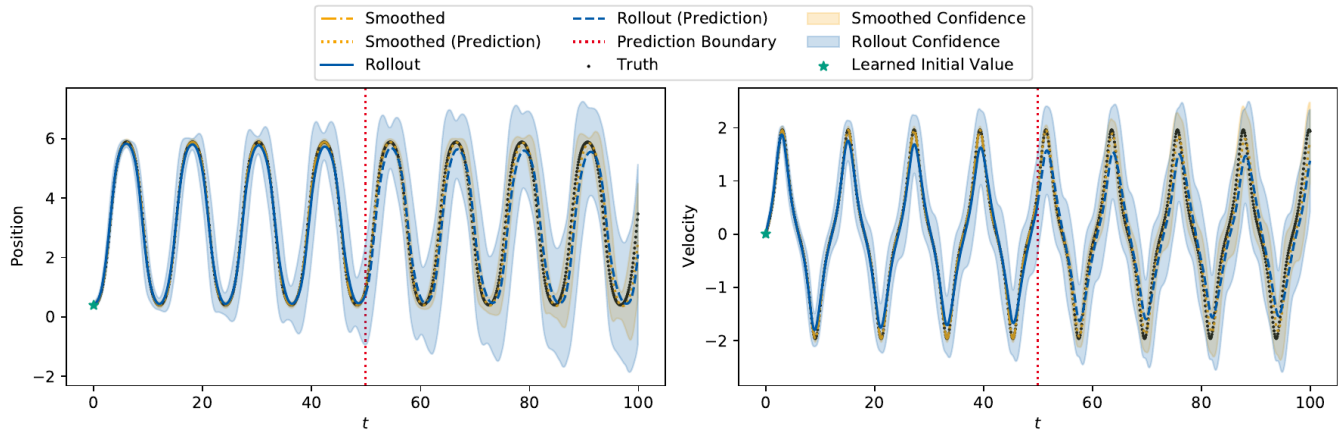


Figure 7.3.: The rollout plot in the observation space of the pendulum environment for $k = 9$. The left plot shows the displacement and the right plot the angular velocity. The black dots represent the true data of which the model used everything till the red prediction boundary to train on. The blue line is the rollout, starting from the learned initial value (marked with a green star). The orange dash-dotted line is the smoothed data. The dotted orange line then is the rollout starting from the last smoothed state, forming the "smoothed prediction". The shaded regions show the confidence, i.e. two times the standard deviation.

7.1.3. Damped Pendulum

As for the undamped pendulum, we have seen in the experiment with different latent dimensionalities that we need at least ten latent dimensions. For ten dimensions, we get a decent NRMSE on both training and prediction. We have seen this behavior in section 6.2.3 and in the rollout plot in Figure 6.9. In contrast to the undamped pendulum, the amplitudes of the oscillations are now on the correct heights, so we definitely learn the energy loss of the system. We can also see this behavior in the plot of the total, kinetic and potential energy in Figure 7.4. We see in the total and the kinetic and potential energy that the damped pendulum model is really close to the real energy levels, in all energy types. This supports the first interpretation of the rollout that we really learn the energy loss. On the other hand, the rollout is phase-shifted to the real trajectory, sometimes even predicting the pendulum to be on the opposite side of a swing. Looking at the rollout in the latent space in Figure 7.5, we see a similar behavior in the latent space, so our learned observation function seems to be correct, but the latent dynamics are a bit off.

Improving this might be possible by fixing the observation function and optimizing just the state dynamics matrix on its own. This reduces the parameters the algorithm has to learn by a lot, possibly yielding a more accurate rollout. But overall we are satisfied with this result as we learn the energy loss and get a confidence that is not too high such that the real position is still in the region of variance. As for the undamped pendulum, we get higher variances in regions where the pendulum turns over.

As the rollout is generated from a linear system in the background, it makes sense that energy-loosing systems can are easier to model: Taming a linear system to keep its energy for a long time is a lot harder than "letting it converge to zero". A stable non-zero system corresponds to eigenvalues that are approximately one (as $\lim_{k\to\infty} 1^k = 1$ and $\lim_{k\to\infty} x^k = 0$ for $|x| < 1$, but $\lim_{k\to\infty} x^k \to \infty$ for $x > 0$). Small disturbances to a system with eigenvalues close can cause the system to diverge quickly.

7.1.4. Gym Pendulum

As we have seen in the experiment with different latent dimensionalities in section 6.2.4, we need at least seven latent dimensions to get a decent NRMSE on the training rollout. However, we noticed that we get better results in one run for four latent dimensions (see section 6.2.4) which we looked at for a comparison with [MWK19]. This shows that our algorithm is extremely sensitive to the initialization as the neural network is initialized randomly. Looking at the rollout of the seven-dimensional latent run in section 6.2.4, we see that the latent "stops to change" after the prediction boundary in comparison to the four-dimensional latent where the rollout still moves (and roughly captures the dynamics). By taking a look at the rollout in the latent space in Figure 7.6 and Figure 7.7 for the four- and seven-dimensional run, we see completely different time behaviors.

As we have already outlined in subsection 7.1.3, it is hard for a linear system to be stable when not converging to zero. We see such a behavior in the latent rollout: While the latents of the four-dimensional run all rise exponentially after the prediction horizon, causing movement in the observation space, approximately half of the latents in the seven-dimensional exponentially increase while the other half decreases exponentially. This seems to lead to vanishing within the neural network, explaining the non-movement in the observation space. This behavior might be improved by regularizing the latent dynamics to a system with eigenvalues really close to one, leading to more stable dynamics. Another idea would be to make the latent dynamics time-dependent so compensate for the exploding states by adding an extra latent state just representing the time step. However, this would impose other difficulties and would make the system non-autonomous.

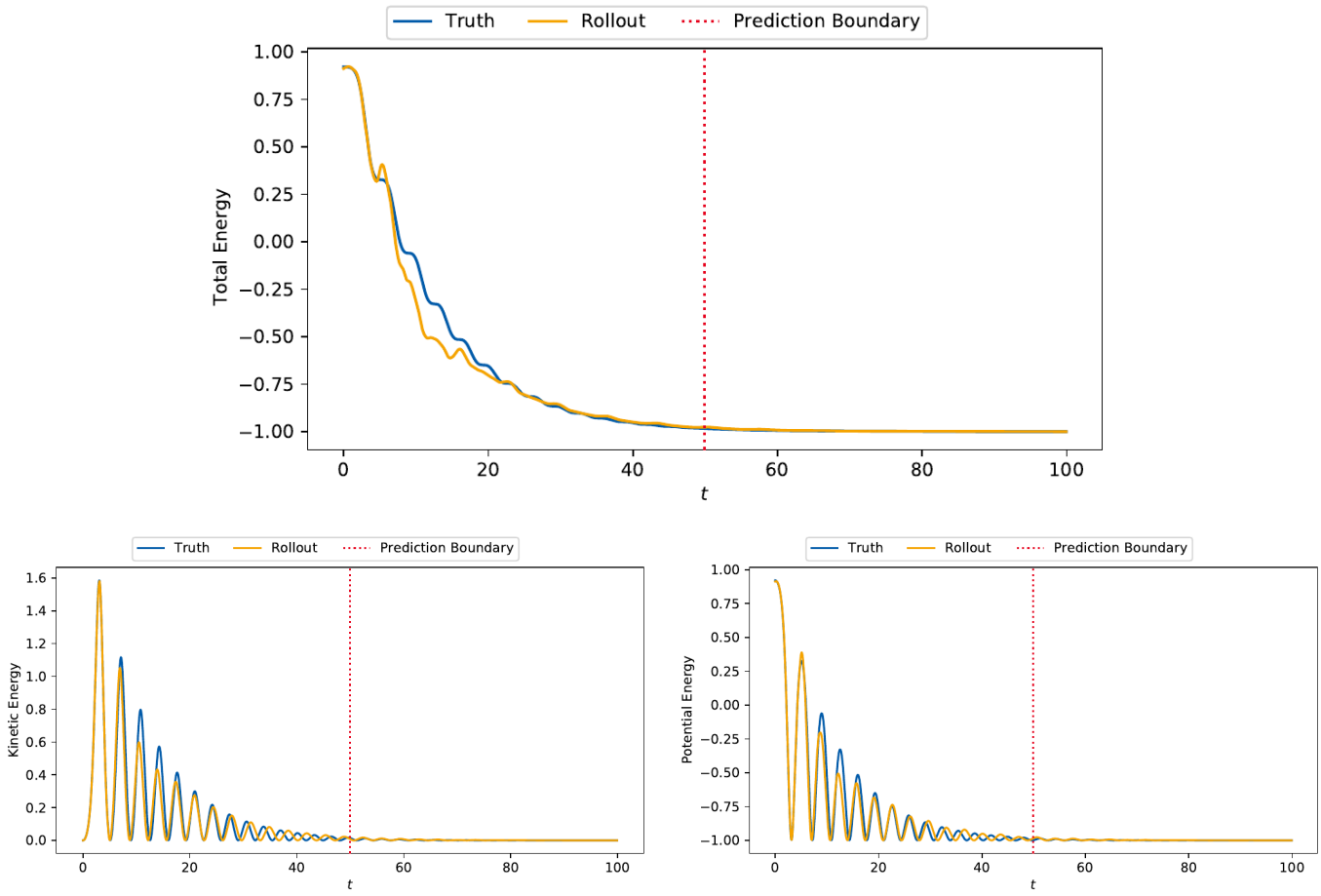


Figure 7.4.: Total energy of the damped pendulum composed of the kinetic energy on the bottom left and the potential energy on the bottom right. The blue line represents the true energy, calculated from the training and validation data. The orange line is the energy calculated from the rollout. As usual, the red line is the prediction boundary up until all training data was used; afterwards, the model predicted the rest of the data.

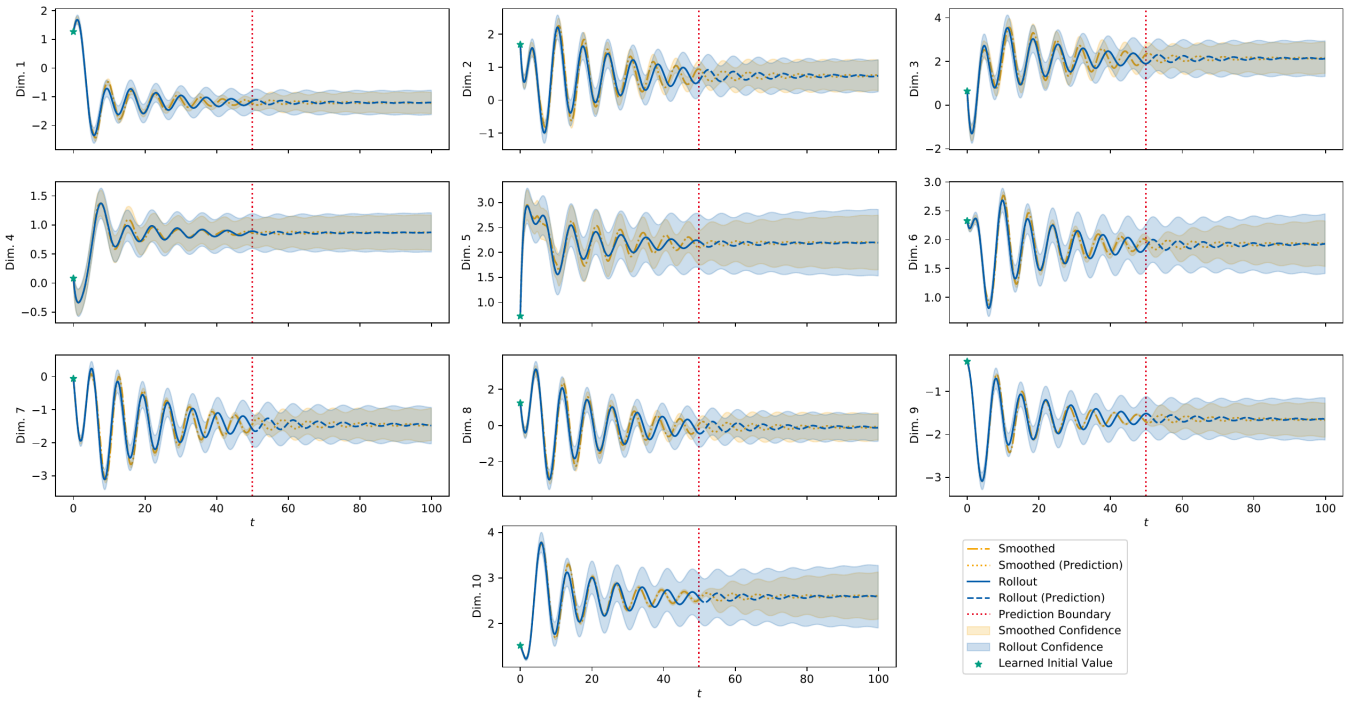


Figure 7.5.: Rollout of the latent dimensions of the damped pendulum environment for $k = 10$ latents.

Another interesting result of the Gym pendulum experiment is that it performs a lot worse than the simple angular pendulum in subsection 7.1.2. This might be caused by the sine/cosine terms as these add more nonlinearity to the system (with small angle approximations it is possible to model the pendulum for small displacements, this is not possible using the sine/cosine of the angle). But as taking the sine/cosine of the angle is actually a nonlinear feature transformation typically used to implicitly encode the symmetries in a polar coordinate environment like the pendulum (swinging the pendulum around one time does not increase the angle by 2π, instead the pendulum works like a congruence class generalized to $\mathbb{R}$), we could use the inverse feature transformation to recover the angle from the sine/cosine data and get a similar performance as for the angular pendulum.

7.1.5. Gym Cartpole

As we have seen in the experiments with different latent dimensionalities in section 6.2.5, we need at least ten latent dimensions to get a decent NRMSE on the (training) rollout. We have also seen that more latent dimensions (e.g. 16 dimensions in section 6.2.5) do not yield better results than ten latent dimensions (see section 6.2.5). However, in both cases we get a near-to-perfect rollout before the prediction boundary but fail at the prediction. For the ten dimensional latent, the model captures the dynamics of the pole displacement/velocity and also the cart velocity after the prediction boundary roughly, but for the cart position the trajectory is quite off. Another interesting result is that the model is a lot more confident in the pole displacement and velocity than the cart position. This makes sense as the movement of the car is more complex as it accelerates nonlinearly only through the movement of the pendulum. This does not impose a linear movement on the cart, but rather it is centered around zero with a light shifting occurring when the pendulum

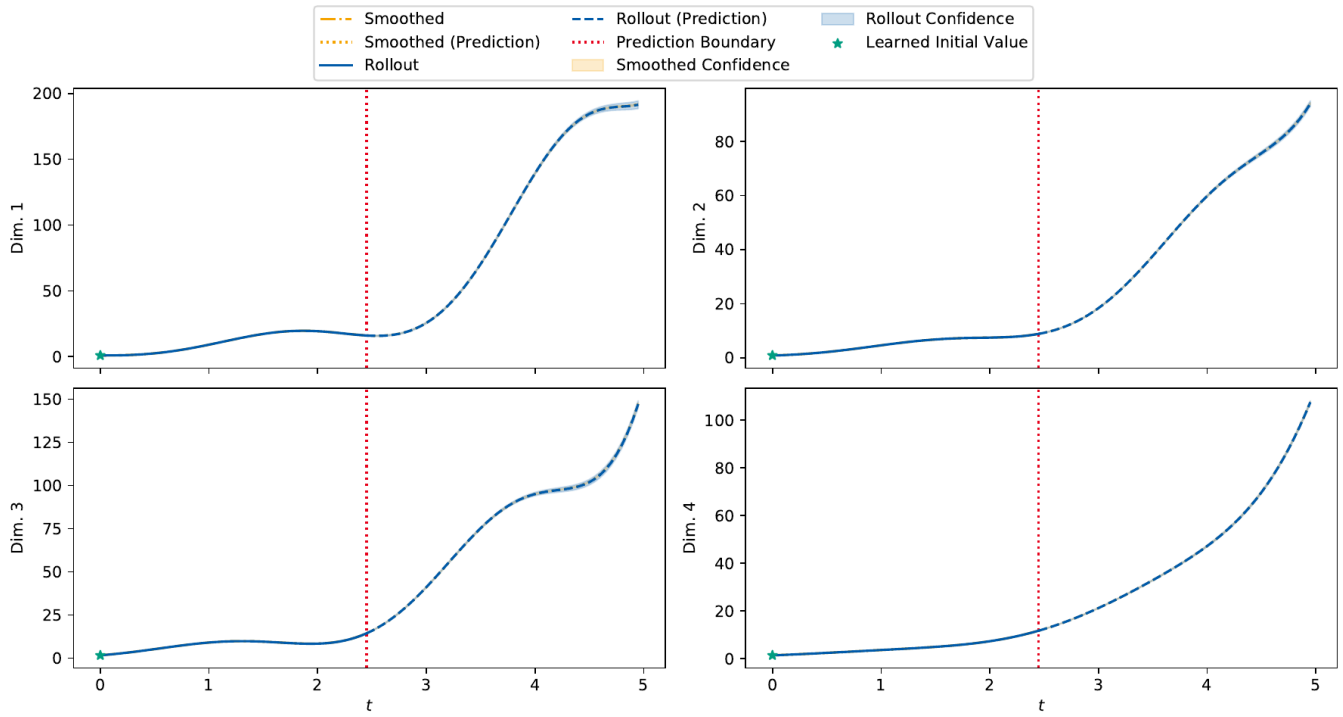


Figure 7.6.: Rollout of the latent dimensions of the Gym pendulum environment for $k = 6$ latents.

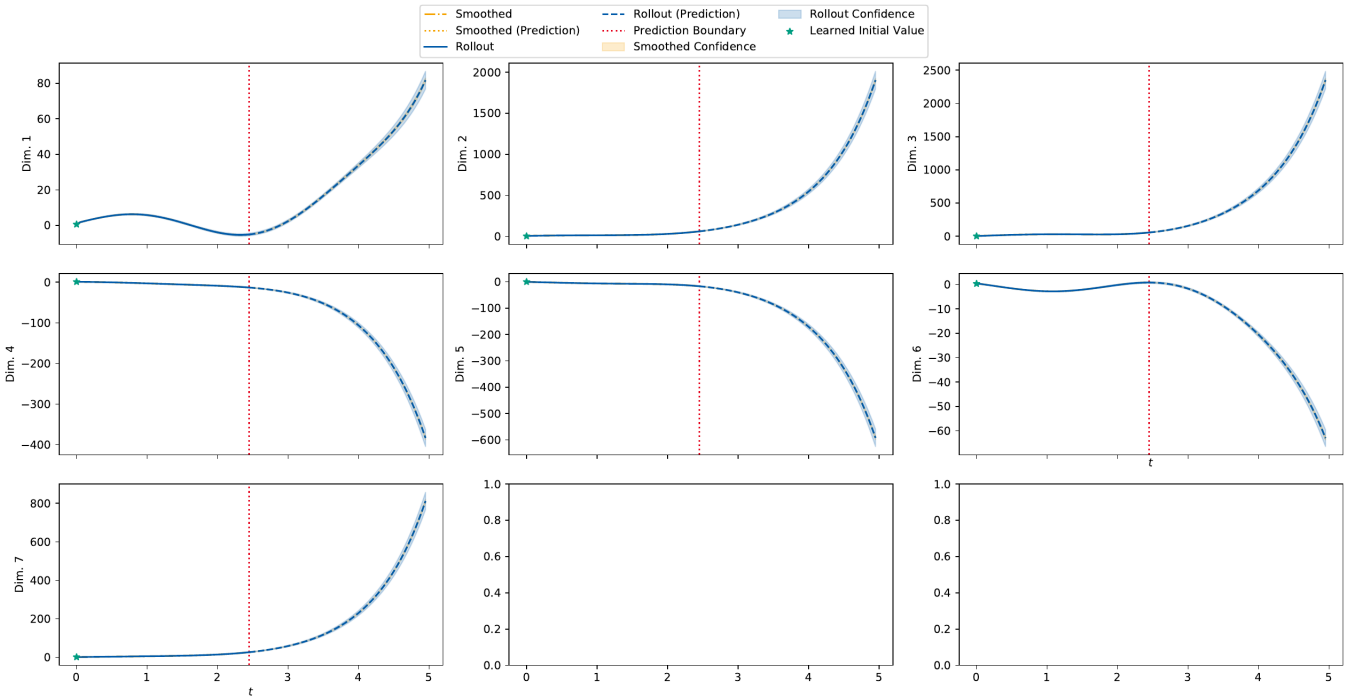


Figure 7.7.: Rollout of the latent dimensions of the Gym pendulum environment for $k = 7$ latents.

falls down or swings up[1]. Also, as we have seen in subsection 7.1.2 and subsection 7.1.3, we are able to learn the movement of an undamped and a damped pendulum, where the motion of the pole on a cart can be seen as a slightly damped pendulum as the system is transferring energy from the pendulum to the cart.

7.1.6. Gym Double Pendulum

As we have seen in the experiment with different latent dimensionalities, we need at least 18 latent dimensions to model the double pendulum adequately in the training rollout. As expected from the NRMSE, we get a decent fit on the rollout for 18 latent dimensions (see Figure 6.18), but the prediction is far off the true trajectory. The prediction does not even capture the dynamics roughly and also it is pretty certain on its wrong trajectory. This matches our expectations as the double pendulum is a chaotic environment with nonlinear coupling that is really hard to learn.

Improving the performance on the double pendulum might be possible by using a shorter integration interval h to provide the algorithm more information between the states such that it can better extrapolate from the data. Another idea, similar to the one outlined in subsection 7.1.4, would be to apply an inverse feature transform on the sine/cosine of the angles to reduce the dimensionality of the observation space and let "the model discover the symmetries". We will discuss this again in Future Work. But overall we are satisfied that we managed to learn at least the rollout on the training data even in a chaotic system.

7.1.7. Running on the CPU or GPU Makes a Difference

We noticed that running our code on the Central Processing Unit (CPU) or on the Graphics Processing Unit (GPU) makes a huge difference in the quality of the results, where running on the GPU is better. The environment we noticed the greatest difference is the double pendulum, of which we added two plots in section B.1. Figure B.1 shows the double pendulum experiment result running on the CPU and Figure B.2 shows the double pendulum experiment result running on the GPU. We expect both plots to be the same, but in fact the plot of the experiment running on the CPU looks better. These runs are the runs `1564` and `1565`, respectively. In both cases we set the seed for both NumPy and PyTorch to the same value. As the data was generated beforehand, the Gym seed does not matter in this case. We expect this to be an issue with a gradient flowing too far through the computation graph or an issue with floating point operations working differently on the CPU/GPU.

7.1.8. Learning Multiple Sequences at Once

We derived the Koopman inference algorithm in a way such that it should be possible to learn on multiple observation sequences at once (see section 5.1). However, we noticed that the results of learning from multiple sequences at once yields worse results than learning one observation sequence. We have multiple explanations for this issue, the most probable being an error in the implementation we did not find. Another explanation is that in the single-sequence case the neural network of the observation function overfits to the training data and does not have enough "memory" to also overfit to another observation sequence and therefore breaks. This is a serious problem to address and we might be able to overcome this problem by adding noise to the input to improve generalization.

[1] See GitHub for a GIF of the cartpole movement.

Figure B.3 and Figure B.4 show two rollouts on the damped pendulum environment, the former only learning on a single observation sequence and the latter learning on multiple sequences at once. We see that while the result of a single sequence captures the dynamics decently (see subsection 7.1.3 for an in-depth discussion), the model that learned on two sequences does not capture the dynamics correctly.

7.1.9. Discussion on Numerical Stability

The primary thing we noticed in terms of numerical stability is the definiteness of the covariance matrices $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{V}_0$. While this can be fixed by learning the Cholesky decomposition directly, better regularization or putting priors on the matrices (see section 7.3 for more information on these topics), we also faced singular matrices. This primarily happened for high latent dimensionalities, a phenomenon that can also be seen in the latent dimensionalities comparison plots in section 6.2. This is caused by the algorithm "not knowing what to do" with a latent dimension, i.e. when it already has "enough" dimensions to explain the data. We could use this behavior to automatically detect which latents are important and which are not, called *automatic relevance determination* (see subsection 7.3.3) for more proposals on this topic.

With implementing the square-root filtering/smoothing in the E-step we gained a lot of numerical stability (also also speed) by not having to compute the Cholesky decomposition of every smoothed covariance.

With all that combined, we are satisfied with the numerical stability of the Koopman inference algorithm, especially given that one of the great instabilities, singular matrices, lead to future work on automatic relevance determination.

7.2. Comparison with Related Work

In this section we will discuss the performance of the Koopman inference algorithm in the context of the results from related work, the Deep Variational Koopman (DVK) model proposed in [MWK19]. We used the code Morton et al. originally published along with their paper and modified it to work with the latest TensorFlow [AAB+16] and applied the changes mentioned in [MWK19] (i.e. made the actions of the acrobot and cartpole environment continuous). Additionally, we removed the actuation abilities of the environments along with the control inputs to mimic the behavior of our work which currently does not work with control inputs. Our modified version can be found on GitHub[2].

We ran four different environments:

- The acrobot environment that is the same as our double pendulum environment presented in section 6.1.1.
- Two cartpole environments, one equal to our cartpole environment presented in section 6.1.1 and one with sine and cosine of the pole angle displacement as it was done initially in the DVK reference implementation.
- The pendulum environment that that is the same as our Gym pendulum environment presented in section 6.1.1 with sine/cosine features.

[2] See `https://github.com/fdamken/variational-koopman` on the branch `without-control`.

If possible, we used the same amount of time steps during training for both our and the other model. However, due to numerical instabilities in their implementation, we had to deviate in terms of the number of time steps for some environments[3]:

- We could only use 32 steps on the cartpole as opposed to our model which used 150.
- We used 64 steps on the double pendulum as opposed to our model which used only 56.

Also, our algorithm currently only learns on one observation sequence, i.e. if we learn on a sequence that covers T_{train} time steps, we only use T_{train} data points, while the DVK model uses 3968 equally sized sequences. This sums up to $253{,}952$ data points for the acrobot environment, $126{,}976$ for the cartpole environment and $198{,}400$ for the pendulum environment.

We assess the performance of both models by looking at the rollout plots and evaluating the performance qualitatively, and by comparing the Normalized Root Mean Squared Error. We compute the NRMSE along the whole trajectory, although we would like to point out that the "rollout" of DVK is not a complete rollout as the observations on the training data are reconstructions of the learning data. That is, the rollout starts from the prediction boundary, possibly yielding better results before the boundary as if the data would be computed from the beginning. But this most probably does not affect our assessment in meaningful ways.

We will now discuss the qualitative performance on each environment separately and afterwards assess the quantitative results for all environments at once.

7.2.1. Pendulum

The rollout on the pendulum environment of the DVK model is shown in Figure 7.8. We compare these results with both our results for the Gym pendulum (with sine/cosine) features and our results on the angular pendulum. As discussed in subsection 7.1.4, we can assume an inverse feature transformation back to the polar coordinate space.

We see that the reconstruction before the prediction boundary works quite well, however, the sine feature of the angle has a dent around $t \approx 25$. In comparison to our rollout in Figure 6.12 (on the Gym pendulum), our rollout in the training error before the prediction boundary matches the data better. However, the prediction of the DVK model captures the behavior of the system better than our prediction. Looking at the angular pendulum in Figure 6.6 (on ten latent dimensions), our model captures both the dynamics before and after the prediction boundary better than the DVK model.

[3] The exact parameters we ran their code with can be found in the GitHub repository.

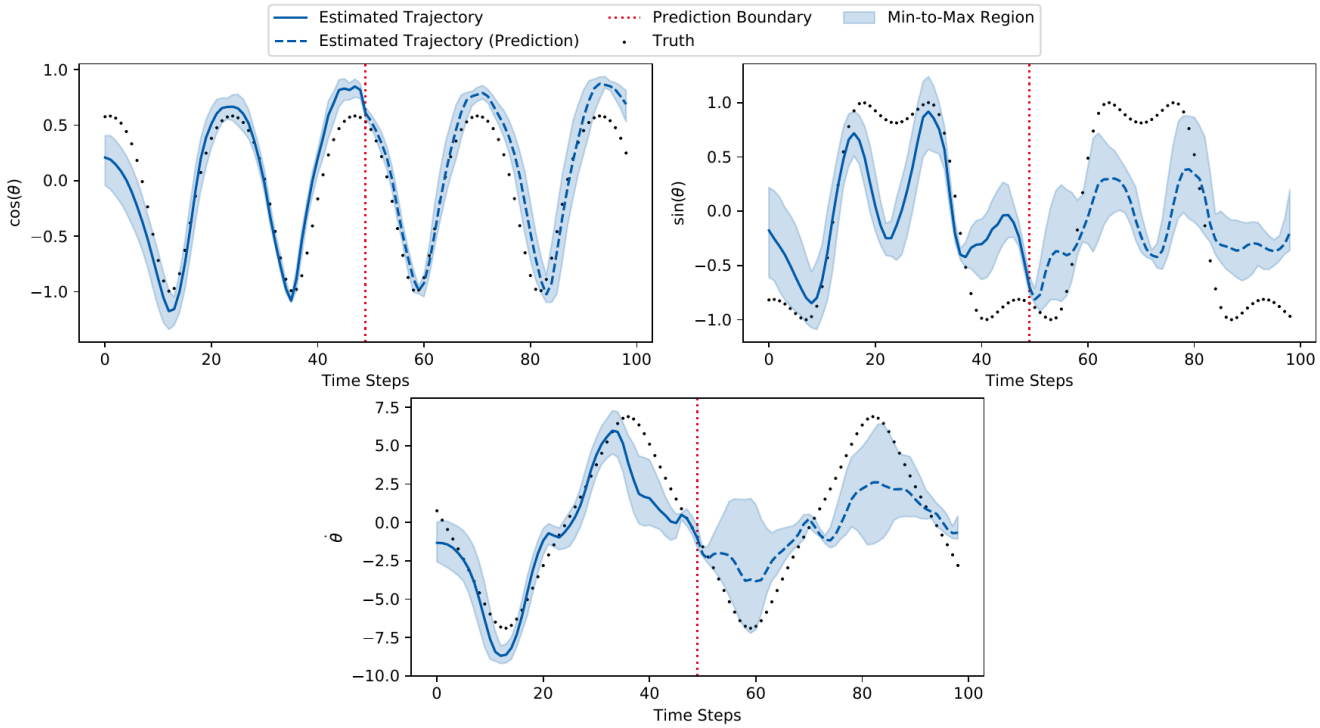


Figure 7.8.: Rollout of the pendulum environment of the DVK model, generated by us. The top row shows the sine/cosine of the displacement of the pendulum, the bottom plot shows the angular velocity. The black dots show the ground truth of the trajectory, the solid blue line the reconstructed states from the latents, the dashed blue line the prediction of the model and the shaded region shows the min-to-max distance. This is generated by sampling multiple sequences from the model and taking the maximal and minimal values at each time step. This can be roughly interpreted as the model confidence. The blue line is the mean of the sampled predictions. As usual, the red line is the prediction boundary, until which the blue line represents the reconstruction; afterwards, the model predicted the rest of the data.

7.2.2. Cartpole

For the cartpole environment, we now have two versions of the environment for the DVK model: One that uses the angle of the pole displacement directly with the rollout in Figure 7.9 and one that uses sine/cosine features of the angle in Figure 7.10.

We see that both the reconstruction and the prediction for both cases diverge greatly from the true data. Interestingly, in case of the sine/cosine features, the prediction diverges more from the true data compared to the angular features, even though the former was a modification originally done in the DVK model. But we have to point out that we were not able to fully reproduce the original results, so the original version might have yielded better results. Compared to our result in Figure 6.15 (on ten latent dimensions), our model captures the dynamics more accurately in the training section, but worse after the boundary (i.e. for prediction), especially in the cart position and velocity. Overall, we can say from this qualitative comparisons, that our model performs better on the cartpole[4].

7.2.3. Double Pendulum

The final environment we benchmarked on the DVK model is the double pendulum, for which both we and the DVK model used cosine/sine feature transformations. The rollout of the DVK model is shown in Figure 7.11.

We see that the reconstruction and prediction of the angular velocity is quite accurate as well as the reconstruction and prediction of the sine of both angles. However, the cosine only matches some of the data points. Compared to our rollout in Figure 6.18 (on 18 latent dimensions), the reconstruction is worse but the prediction is a lot more accurate than ours on every dimension.

7.2.4. Quantitative Comparison and Summary

We now look at a quantitative comparison of both models, the DVK model and the Koopman inference algorithm (ours). We use the NRMSE on the complete rollout, the rollout on the training data (or, in case of DVK, the reconstruction) and the prediction. We summarized the different errors in Table 7.1, where our algorithm is almost consistently better at environments that use the angle directly instead of the cosine/sine. Also, we are almost always better in terms of the training rollout, while we perform worse in the prediction. However, these numbers should be taken with care as the NRMSE is not scale-independent (even though it is better than the RMSE due to the normalization), but as the errors reflect our results from the qualitative comparison, we are confident that our algorithm performs better.

Overall the performance of out algorithm matched our expectations compared to the DVK Koopman model for multiple reasons: Firstly, we gauge the uncertainty directly instead of sampling multiple trajectories and treating the min-max distance as uncertainty. Secondly, our algorithm is much more sample-efficient (see the numbers at the beginning of this section). Thirdly, as already mentioned in related work, our method has a lot simpler model in terms of number of parameters and size of neural networks. Nevertheless, the DVK has multiple of advantages over our method: It generalizes to more environments than our approach, is numerically more stable in some senses (e.g. it does not have covariance matrices that can become negative definite or singular in some cases) and it already has the ability to learn the inputs of an actuated system, as shown in [MWK19].

[4] Note that we used more time steps for training our model, but overall less data points. Also we were not able to train the DVK model on more time steps as it became numerically unstable.

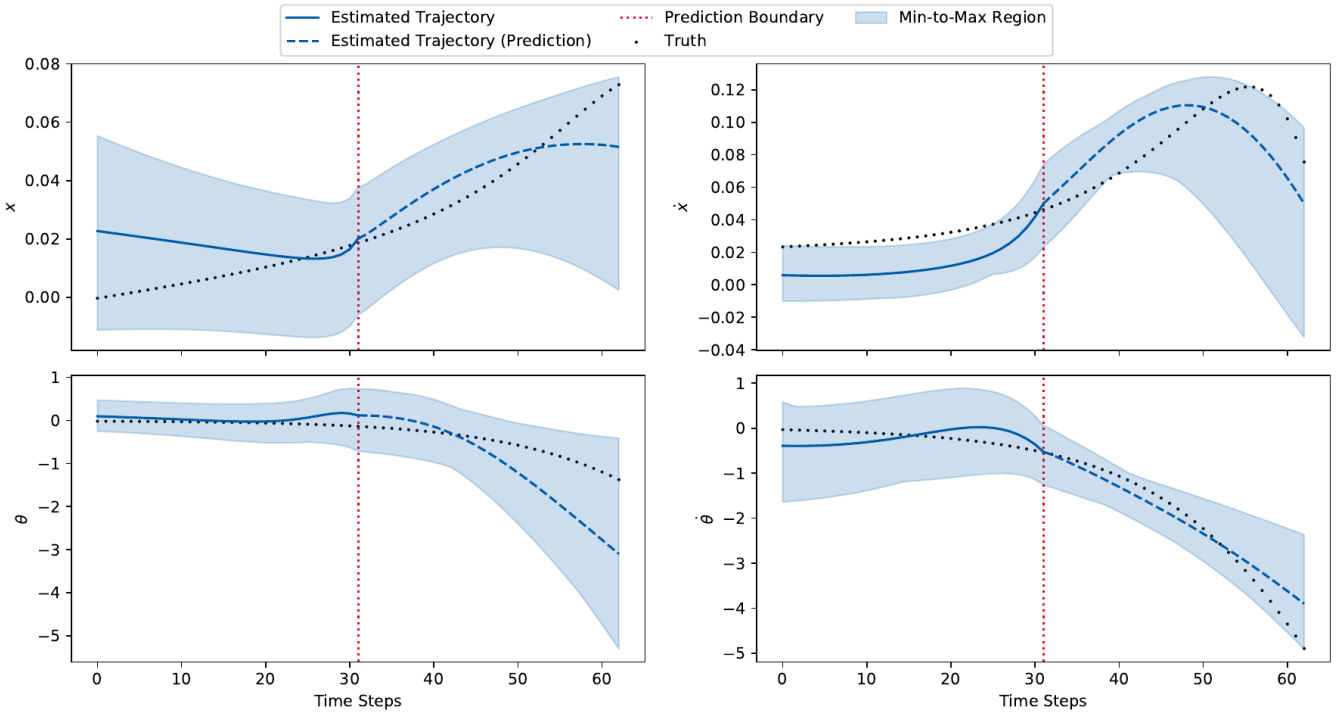


Figure 7.9.: Rollout of the cartpole environment of the DVK model with angular features, generated by us. The top row shows the position/velocity of the cart, the bottom row shows the angle and angular velocity of the pole one the cart. The black dots show the ground truth of the trajectory, the solid blue line the reconstructed states from the latents, the dashed blue line the prediction of the model and the shaded region shows the min-to-max distance. This is generated by sampling multiple sequences from the model and taking the maximal and minimal values at each time step. It can be roughly interpreted as the model confidence. The blue line is the mean of the sampled predictions. As usual, the red line is the prediction boundary, until which the blue line represents the reconstruction; afterwards, the model predicted the rest of the data.

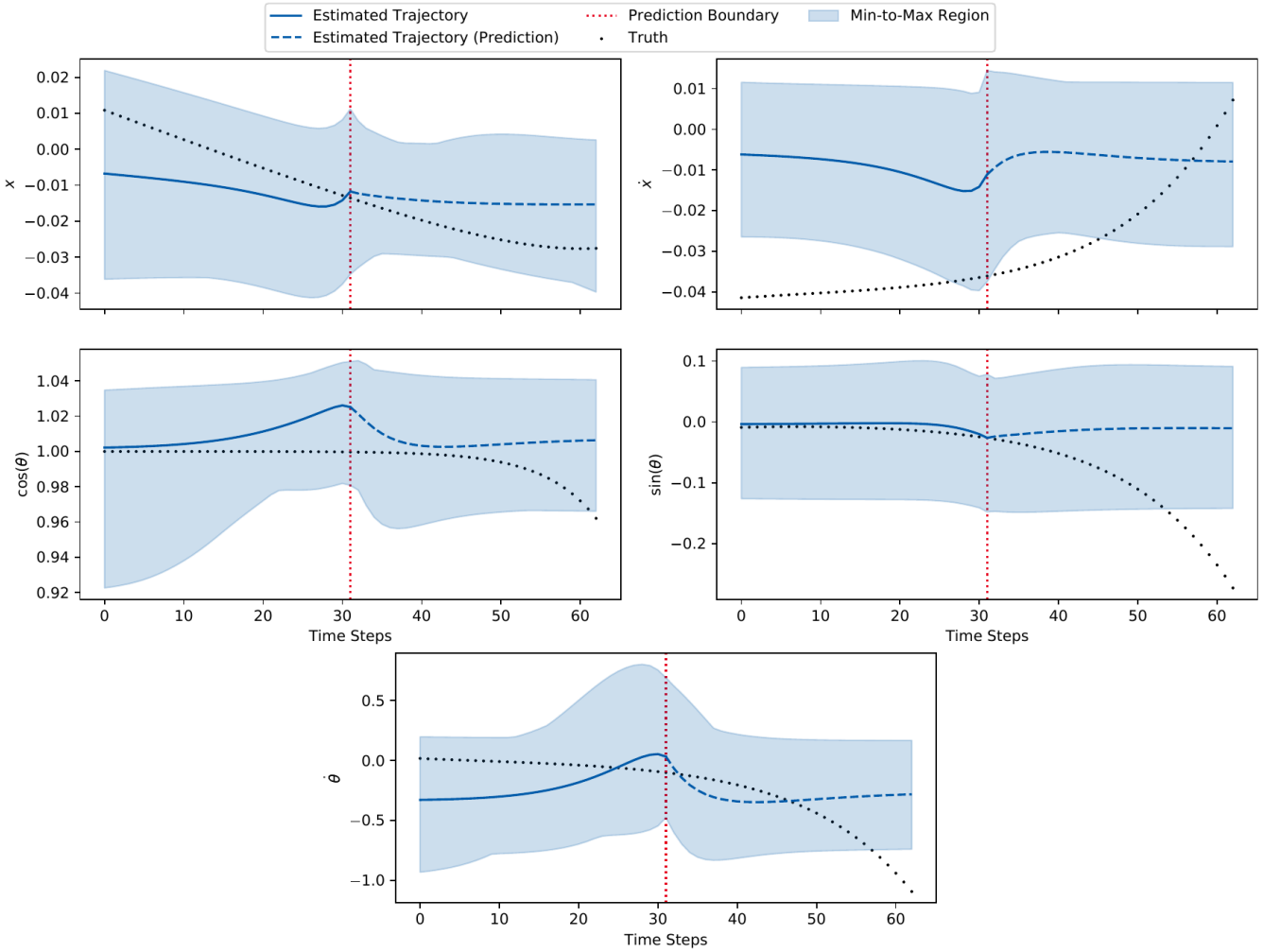


Figure 7.10.: Rollout of the cartpole environment of the DVK model with angular features, generated by us. The top row shows the position/velocity of the cart, the middle row shows the cosine/sine of displacement and the bottom row shows the angular velocity of the pole on the cart. The black dots show the ground truth of the trajectory, the solid blue line the reconstructed states from the latents, the dashed blue line the prediction of the model and the shaded region shows the min-to-max distance. This is generated by sampling multiple sequences from the model and taking the maximal and minimal values at each time step. It can be roughly interpreted as the model confidence. The blue line is the mean of the sampled predictions. As usual, the red line is the prediction boundary, until which the blue line represents the reconstruction; afterwards, the model predicted the rest of the data.

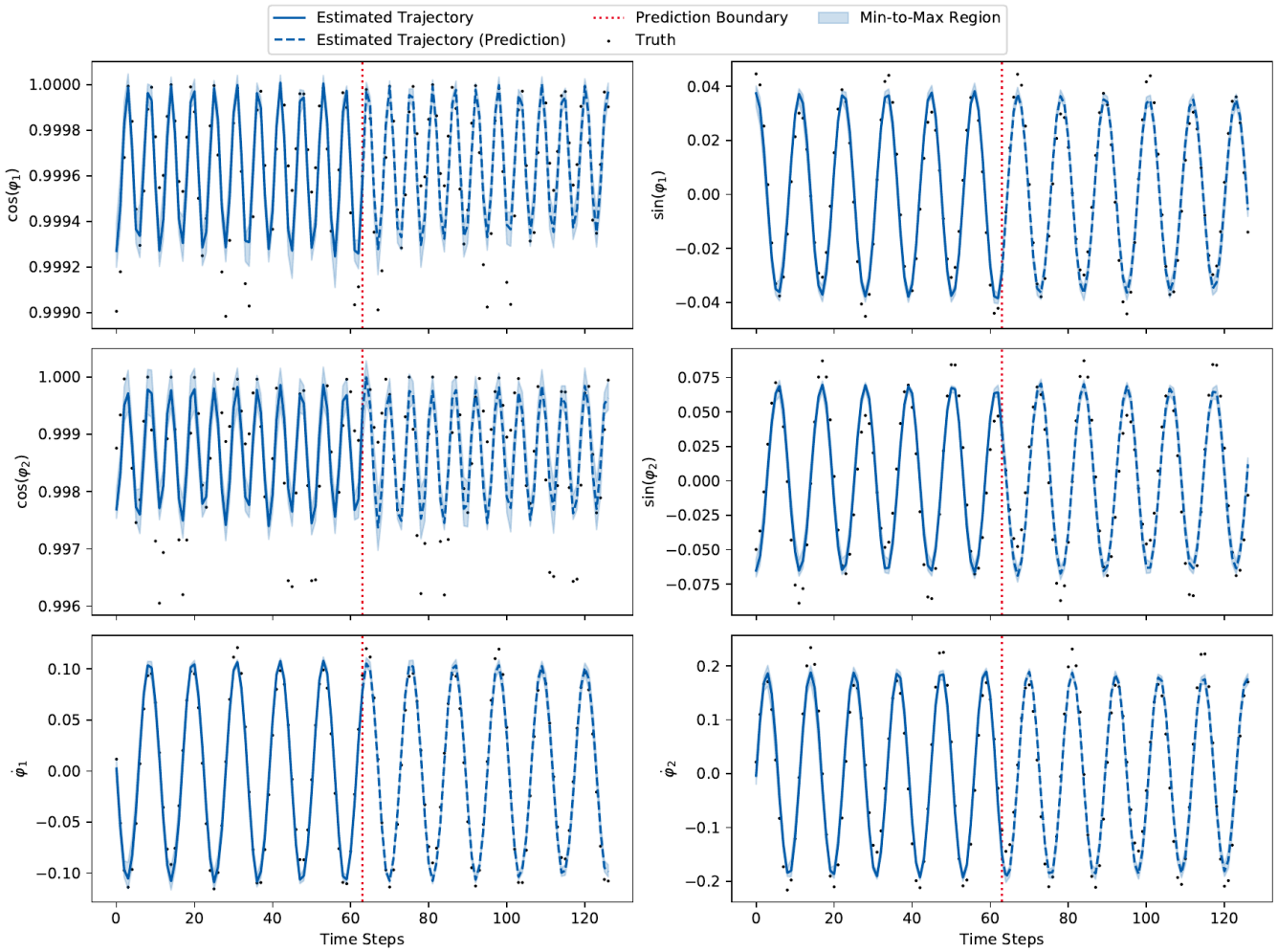


Figure 7.11.: Rollout of the double pendulum environment of the DVK model, generated by us. The top row shows the cosine/sine of the displacement of the inner pendulum, the middle row shows the cosine/sine of the displacement of the outer pendulum and the bottom row shows the angular velocities of the inner and outer pendulum, from left to right. The black dots show the ground truth of the trajectory, the solid blue line the reconstructed states from the latents, the dashed blue line the prediction of the model and the shaded region shows the min-to-max distance. This is generated by sampling multiple sequences from the model and taking the maximal and minimal values at each time step. It can be roughly interpreted as the model confidence. The blue line is the mean of the sampled predictions. As usual, the red line is the prediction boundary, until which the blue line represents the reconstruction; afterwards, the model predicted the rest of the data.

Environment	Error on...	Deep Variational Koopman	Koopman Inference
Sine/Cosine Pendulum	Complete	0.198	**0.023**
vs.	Training	0.168	**0.014**
Angular Pendulum	Prediction	0.222	**0.029**
Sine/Cosine Pendulum	Complete	**0.198**	0.200
vs.	Training	0.168	**0.003**
Sine/Cosine Pendulum	Prediction	**0.222**	0.282
Angular Cartpole	Complete	0.265	**0.200**
vs.	Training	0.466	**0.009**
Angular Cartpole	Prediction	0.420	**0.377**
Sine/Cosine Cartpole	Complete	0.470	**0.200**
vs.	Training	17.222	**0.009**
Angular Cartpole	Prediction	0.817	**0.377**
	Complete	**0.102**	0.228
Double Pendulum	Training	0.103	**0.002**
	Prediction	**0.104**	0.562

Table 7.1.: Comparison of the errors of all environments we benchmarked DVK and Koopman inference on. The environment in the first column describes which environment the data of the DVK is collected from (before "vs.") and from which environment the data of the Koopman inference algorithm is collected from (after "vs."). The complete error is over the rollout/reconstruction over the whole sequence, the training error is over the rollout/reconstruction on the training data and prediction is over rollout on the validation data. The numbers represent the NRMSE. The respective lowest error is marked in bold font.

7.3. Future Work

We will now discuss and propose future work and ideas that can be tried based on the results of this thesis.

7.3.1. Control

In this thesis we presented a method for learning a dynamical system with no control inputs. The most obvious extension would be to add control inputs to the latent space in an additive fashion $\mathbf{s}_{t+1} = \mathbf{A}\mathbf{s}_t + \mathbf{B}\mathbf{u}_t$ with a learnable control matrix $\mathbf{B}$. With this approach it might be possible to actually perform MPC and uncertainty-aware control like in [MWK19], but with a simpler model. Our first approach to this would be to test this method on a simple environment like the pendulum and first learn the control rollout and afterwards perform a stabilization task and try the swing-up of the pendulum.

7.3.2. Bayesian Treatment

Another extension would be to employ a Bayesian view on the parameters and treat e.g. the state dynamics matrix, covariance matrices and so on as random variables. This would allow to gauge the uncertainty on the dynamics itself and would allow to restrict the matrices to small ones by placing a prior on them. The approach would be to remove the Maximum Likelihood (ML) estimator and marginalize over the parameters to get a predictive distribution. Another semi-Bayesian approach would be to use point-estimates using a Maximum A-Posteriori (MAP) estimator rather than a predictive distribution.

7.3.3. Automatic Relevance Determination (ARD)

In combination with going full Bayesian on the model goes the implementation of automatic relevance determination of latent dimensions as proposed in [BG00] for linear systems. Choosing a zero-mean prior might lead to a determination of non-relevant latent states, driving the state dynamics matrix to zero. Implementing this could solve the problems described in subsection 7.1.9 that the algorithm does already have “enough” latent dimensions and drives the variance to a high level on some dimensions.

7.3.4. Better Regularization and Learning the Cholesky Decomposition

As already outlined in section 5.2.1, it might be beneficial for numerical stability to directly learn the Cholesky decomposition of the covariance matrices by using numerical optimization instead of closed-form optimization. Another approach could be to impose better regularization on the matrices if they become negative (semi-) definite. However, while giving this a short first try during our implementation, we found out that just adding some values to the diagonal does not lead to better model performance but rather drives the likelihood to negative infinity.

7.3.5. Speed Improvements by Full PyTorch or Enhanced QR Decomposition

As the square-root smoothing heavily uses QR decompositions which are faster on the CPU but the M-step uses backpropagation which is faster on the GPU, we had to copy the data over in every EM iteration. It might be beneficial to implement a more advanced QR decomposition (e.g. [ABDK11]) and run the whole algorithm in the GPU.

7.3.6. (Inverse) Feature Transformations

As we have seen in 7.1.2 and subsection 7.1.4, using the angle directly instead of the sine/cosine improves prediction capabilities a lot. It might be beneficial to try out the proposed inverse feature transformation to get the angular displacement back from the sine and cosine and to apply this technique to other environments as well (e.g. the double pendulum).

8. Conclusion

Linearization is very important in control to handle nonlinear systems that exhibit complicated dynamics. We have looked at Koopman theory, a technique to find globally linear embedding for a nonlinear system. Unfortunately, the Koopman operator, the core of Koopman theory, is generally infinite-dimensional. Prior work [BBPK16, KKB20, LKB18] has shown that we can find a finite-dimensional representation of the operator. These representations seek Koopman eigenfunctions that span an invariant subspace of the embedding, and do not transform under the influence of the Koopman operator other than being scaled. Nevertheless, most of the existing approaches work on deterministic models, seeking the Koopman eigenfunctions. While methods have been proposed which take a probabilistic perspective on the Koopman operator, they use complicated model setups with multiple neural networks and do not directly gauge the uncertainty of model predictions.

In this thesis we present a combination of existing theory for Linear Gaussian Dynamical Systems with cubature rules to approximate arising nonlinear Gaussian integrals in a deterministic way to build up an Expectation-Maximization algorithm. We called the resulting algorithm the *Koopman inference* algorithm. It alternates between estimating the states in the E-step and optimizing the linear latent state dynamics and a nonlinear observation function that maps the linear latent states back to the nonlinear dynamics in the M-step. This way, we are able to predict the nonlinear dynamics and learned the Koopman observation functions without hand-tuned loss functions like in [LKB18].

We evaluated the model performance on multiple different environments: A simple pendulum, a damped pendulum, the cartpole environment and a double pendulum. We also proved, and showed empirically, that our algorithm extends the common algorithm for learning LGDS by showing that the used integral approximations are exact for linear functions and by validating the result on a regular LGDS. We discovered out that systems that lose energy (e.g. the damped pendulum) are generally easier to learn than systems that conserve energy (e.g. the undamped pendulum), as the latent dynamics have to be stable but should not approach zero to keep moving. Additionally, we discovered that the pendulum with sine/cosine features is a lot harder to learn, presumably because of the additional nonlinearity added by the feature transformation. Also, the algorithm was able to provide a rough approximation for the dynamics of the double pendulum environment, even though this is a chaotic system and it was not able to predict further movements beyond the training data.

Finally, we compared our Koopman inference algorithm against the Deep Variational Koopman model proposed by [MWK19]. We saw that our algorithm performs similar, however, the DVK model is better at predicting the dynamics beyond the training data, while our model is primarily able to predict until the end of the training data. We should note that the DVK performs reconstruction from the latents that were learned during the training while our model performs the rollout from the beginning.

Overall we are satisfied with the performance of the Koopman inference algorithm. It allows further improvements to learning the influence of control inputs on actuated systems or employing a Bayesian view on the latent dimensions and further gauge the uncertainty on the state dynamics.

Bibliography

[AAB+16] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv:1603.04467 [cs]*, March 2016.

[ABDK11] Michael Anderson, Grey Ballard, James Demmel, and Kurt Keutzer. Communication-Avoiding QR Decomposition for GPUs. In *2011 IEEE International Parallel Distributed Processing Symposium*, pages 48–58, May 2011.

[BBPK16] Steven L. Brunton, Bingni W. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Koopman Invariant Subspaces and Finite Linear Representations of Nonlinear Dynamical Systems for Control. *PLOS ONE*, 11(2):e0150171, February 2016.

[BCP+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv:1606.01540 [cs]*, June 2016.

[Bea03] Matthew J. Beal. *Variational Algorithms for Approximate Bayesian Inference*. Doctoral, UCL (University College London), 2003.

[BG00] Matthew J. Beal and Zoubin Ghahramani. The Variational Kalman Smoother. Technical report, Gatsby Computational Neuroscience Unit, May 2000.

[Bir27] George David Birkhoff. *Dynamical Systems*. American Mathematical Soc., December 1927.

[BPG+19] Philipp Becker, Harit Pandya, Gregor Gebhardt, Cheng Zhao, James Taylor, and Gerhard Neumann. Recurrent Kalman Networks: Factorized Inference in High-Dimensional Deep Feature Spaces. *arXiv:1905.07357 [cs, stat]*, May 2019.

[CSS55] By R. Ceppellini, M. Siniscalco, and C. a. B. Smith. The Estimation of Gene Frequencies in a Random-Mating Population. *Annals of Human Genetics*, 20(2):97–115, 1955.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data Via the EM Algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[DO11] Marc Peter Deisenroth and Henrik Ohlsson. A Probabilistic Perspective on Gaussian Filtering and Smoothing. *arXiv:1006.2165 [cs, math, stat]*, June 2011.

[Flo05] Răzvan V. Florian. *Correct Equations for the Dynamics of the Cart-Pole*. 2005.

[GH96] Zoubin Ghahramani and Geoffrey E. Hinton. Parameter estimation for linear dynamical systems. Technical Report CRG-TR-96-2, University of Toronto, Februrary 22, 1996.

[GH01] Nicola Guglielmi and Ernst Hairer. Implementing Radau IIA Methods for Stiff Delay Differential Equations. *Computing*, 67(1):1–12, July 2001.

[GKC+17] Klaus Greff, Aaron Klein, Martin Chovanec, Frank Hutter, and Jürgen Schmidhuber. The Sacred Infrastructure for Computational Research. *Proceedings of the 16th Python in Science Conference*, pages 49–56, 2017.

[HHV20] Yiqiang Han, Wenjian Hao, and Umesh Vaidya. Deep Learning of Koopman Representation for Control. *arXiv:2010.07546 [cs, eess]*, October 2020.

[HMv+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[Jen06] J. L. W. V. Jensen. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, 30:175–193, 1906.

[JUD95] S. J. Julier, J. K. Uhlmann, and H. F. Durrant-Whyte. A new approach for filtering nonlinear systems. In *Proceedings of 1995 American Control Conference - ACC'95*, volume 3, pages 1628–1632 vol.3, June 1995.

[Kal60] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, March 1960.

[KB17] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017.

[KKB20] Eurika Kaiser, J. Nathan Kutz, and Steven L. Brunton. Data-driven discovery of Koopman eigenfunctions for control. *arXiv:1707.01146 [math]*, May 2020.

[Koo31] B. O. Koopman. Hamiltonian Systems and Transformation in Hilbert Space. *Proceedings of the National Academy of Sciences of the United States of America*, 17(5):315–318, May 1931.

[KW14] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, May 2014.

[LKB18] Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1):4950, December 2018.

[Min99] Thomas P. Minka. From Hidden Markov Models to Linear Dynamical Systems. Technical Report TR-531, July 1999.

[MWK19] Jeremy Morton, Freddie D. Witherden, and Mykel J. Kochenderfer. Deep Variational Koopman Models: Inferring Koopman Observations for Uncertainty-Aware Dynamics Modeling and Control. *arXiv:1902.09742 [cs, stat]*, June 2019.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8026–8037. Curran Associates, Inc., 2019.

[RTS65] H. E. Rauch, F. Tung, and C. T. Striebel. Maximum likelihood estimates of linear dynamic systems. *AIAA Journal*, 3(8):1445–1450, 1965.

[Rut13] Mark G. Rutten. Square-root unscented filtering and smoothing. In *2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pages 294–299, April 2013.

[SGWY92] Troy Shinbrot, Celso Grebogi, Jack Wisdom, and James A. Yorke. Chaos in a double pendulum. *American Journal of Physics*, 60(6):491–499, June 1992.

[Sol10] Arno Solin. *Cubature Integration Methods in Non-Linear Kalman Filtering and Smoothing*. Bachelor's Thesis, 2010.

[VW01] R. Van der Merwe and E.A. Wan. The square-root unscented Kalman filter for state and parameter-estimation. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, volume 6, pages 3461–3464, May 2001.

[WKR15] Matthew O. Williams, Ioannis G. Kevrekidis, and Clarence W. Rowley. A Data–Driven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346, December 2015.

A. Notes on the Koopman Inference Algorithm

A.1. Full Derivation

In this appendix we cover the full derivation of the Koopman inference algorithm, starting with the formulation of the expected log-likelihood, going over the M-step and finally to the E-step with the regular, non-square-root, filter/smoother equations.

Our model is given as

$$\begin{aligned} p(\mathbf{s}_t \,|\, \mathbf{s}_{t-1}) &\sim \mathcal{N}(\mathbf{A}\mathbf{s}_{t-1}, \mathbf{Q}) \\ p(\mathbf{y}_t \,|\, \mathbf{s}_t) &\sim \mathcal{N}\big(\mathbf{g}(\mathbf{s}_t), \mathbf{R}\big) \end{aligned}$$

with diagonal noise covariances $\mathbf{Q}$ and $\mathbf{R}$.

M-Step For a single observation sequence, the log-likelihood $\log p\Big(\mathbf{s}_{1:T}^{(n)}, \mathbf{y}_{1:T}^{(n)}\Big)$ is given as,

$$\begin{aligned} \log p\Big(\mathbf{s}_{1:T}^{(n)}, \mathbf{y}_{1:T}^{(n)}\Big) &= \log \left(p\Big(\mathbf{s}_1^{(n)}\Big) \prod_{t=2}^{T} p\Big(\mathbf{s}_t^{(n)} \,|\, \mathbf{s}_{t-1}^{(n)}\Big) \prod_{t=1}^{T} p\Big(\mathbf{y}_t^{(n)} \,|\, \mathbf{s}_t^{(n)}\Big) \right) \\ &= \log p\Big(\mathbf{s}_1^{(n)}\Big) + \sum_{t=2}^{T} \log p\Big(\mathbf{s}_t^{(n)} \,|\, \mathbf{s}_{t-1}^{(n)}\Big) + \sum_{t=1}^{T} \log p\Big(\mathbf{y}_t^{(n)} \,|\, \mathbf{s}_t^{(n)}\Big) \\ &= -\frac{1}{2}\log|\mathbf{V}_0| - \frac{k}{2}\log(2\pi) - \frac{1}{2}\Big(\mathbf{s}_1^{(n)} - \mathbf{m}_0\Big)^T \mathbf{V}_0^{-1}\Big(\mathbf{s}_1^{(n)} - \mathbf{m}_0\Big) \\ &\qquad + \sum_{t=2}^{T} -\frac{1}{2}\log|\mathbf{Q}| - \frac{k}{2}\log(2\pi) - \frac{1}{2}\Big(\mathbf{s}_t^{(n)} - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\Big)^T \mathbf{Q}^{-1}\Big(\mathbf{s}_t^{(n)} - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\Big) \\ &\qquad + \sum_{t=1}^{T} -\frac{1}{2}\log|\mathbf{R}| - \frac{p}{2}\log(2\pi) - \frac{1}{2}\Big(\mathbf{y}_t^{(n)} - \mathbf{g}_t^{(n)}\Big)^T \mathbf{R}^{-1}\Big(\mathbf{y}_t^{(n)} - \mathbf{g}_t^{(n)}\Big) \\ &= -\frac{T(k+p)}{2}\log(2\pi) - \frac{1}{2}\log|\mathbf{V}_0| - \frac{T-1}{2}\log|\mathbf{Q}| - \frac{T}{2}\log|\mathbf{R}| \\ &\qquad - \frac{1}{2}\Big(\mathbf{s}_1^{(n)} - \mathbf{m}_0\Big)^T \mathbf{V}_0^{-1}\Big(\mathbf{s}_1^{(n)} - \mathbf{m}_0\Big) \\ &\qquad - \frac{1}{2}\sum_{t=2}^{T}\Big(\mathbf{s}_t^{(n)} - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\Big)^T \mathbf{Q}^{-1}\Big(\mathbf{s}_t^{(n)} - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\Big) \\ &\qquad - \frac{1}{2}\sum_{t=1}^{T}\Big(\mathbf{y}_t^{(n)} - \mathbf{g}_t^{(n)}\Big)^T \mathbf{R}^{-1}\Big(\mathbf{y}_t^{(n)} - \mathbf{g}_t^{(n)}\Big) \end{aligned}$$

as the system behaves Markovian and $\mathbf{m}_0$ and $\mathbf{V}_0$ are the initial state mean and covariance. We assume the observation sequences to be independently and identically distributed (i.i.d.), we can formulate the joint complete log-likelihood $\log p(\mathbf{s}_{1:T}, \mathbf{y}_{1:T})$ as the sum of all subsequent log-likelihoods:

$$\begin{aligned}
\log p(\mathbf{s}_{1:T}, \mathbf{y}_{1:T}) &= \log \prod_{n=1}^{N} p\Big(\mathbf{s}_{1:T}^{(n)}, \mathbf{y}_{1:T}^{(n)}\Big) = \sum_{n=1}^{N} \log p\Big(\mathbf{s}_{1:T}^{(n)}, \mathbf{y}_{1:T}^{(n)}\Big) \\
&= -\frac{NT(k+p)}{2}\log(2\pi) - \frac{N}{2}\log|\mathbf{V}_0| - \frac{N(T-1)}{2}\log|\mathbf{Q}| - \frac{NT}{2}\log|\mathbf{R}| \\
&\quad - \frac{1}{2}\sum_{n=1}^{N}\Big(\mathbf{s}_1^{(n)} - \mathbf{m}_0\Big)^T \mathbf{V}_0^{-1}\Big(\mathbf{s}_1^{(n)} - \mathbf{m}_0\Big) \\
&\quad - \frac{1}{2}\sum_{n=1}^{N}\sum_{t=2}^{T}\Big(\mathbf{s}_t^{(n)} - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\Big)^T \mathbf{Q}^{-1}\Big(\mathbf{s}_t^{(n)} - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\Big) \\
&\quad - \frac{1}{2}\sum_{n=1}^{N}\sum_{t=1}^{T}\Big(\mathbf{y}_t^{(n)} - \mathbf{g}_t^{(n)}\Big)^T \mathbf{R}^{-1}\Big(\mathbf{y}_t^{(n)} - \mathbf{g}_t^{(n)}\Big)
\end{aligned}$$

To derive the M-step formulas, we need to maximize the *expected* log-likelihood. Therefore, we will now derive the expected log-likelihood:

$$Q = \mathbb{E}\big[p(\mathbf{s}_{1:T}, \mathbf{y}_{1:T}) \,|\, \mathbf{y}_{1:T}\big]$$

This quantity depends on five expectations $\hat{\mathbf{s}}_{t|t_0}^{(n)}$, $\mathbf{P}_{t|t_0}^{(n)}$, $\mathbf{P}_{t,t-1|t_0}^{(n)}$, $\hat{\mathbf{g}}_t^{(n)}$ and $\mathbf{G}_t^{(n)}$ as defined in equations (5.1) through (5.5) in section 5.1. For brevity, let

$$\begin{aligned}
q_1 &\coloneqq -\frac{NT(k+p)}{2}\log(2\pi) - \frac{N}{2}\log|\mathbf{V}_0| - \frac{N(T-1)}{2}\log|\mathbf{Q}| - \frac{NT}{2}\log|\mathbf{R}| \\
q_2 &\coloneqq -\frac{1}{2}\sum_{n=1}^{N}\Big(\mathbf{s}_1^{(n)} - \mathbf{m}_0\Big)^T \mathbf{V}_0^{-1}\Big(\mathbf{s}_1^{(n)} - \mathbf{m}_0\Big) \\
q_3 &\coloneqq -\frac{1}{2}\sum_{n=1}^{N}\sum_{t=2}^{T}\Big(\mathbf{s}_t^{(n)} - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\Big)^T \mathbf{Q}^{-1}\Big(\mathbf{s}_t^{(n)} - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\Big) \\
q_4 &\coloneqq -\frac{1}{2}\sum_{n=1}^{N}\sum_{t=1}^{T}\Big(\mathbf{y}_t^{(n)} - \mathbf{g}_t^{(n)}\Big)^T \mathbf{R}^{-1}\Big(\mathbf{y}_t^{(n)} - \mathbf{g}_t^{(n)}\Big)
\end{aligned}$$

such that $\log p(\mathbf{s}_{1:T}, \mathbf{y}_{1:T}) = q_1 + q_2 + q_3 + q_4$ and, with Q_1, Q_2, Q_3 and Q_4 being the corresponding expectations, $Q = Q_1 + Q_2 + Q_3 + Q_4$.

We will now evaluate the respective expectations.

- Compute Q_1:

$$Q_1 = \mathbb{E}[q_1 \,|\, \mathbf{y}_{1:T}] = -\frac{NT(k+p)}{2}\ln(2\pi) - \frac{N}{2}\ln|\mathbf{V}_0| - \frac{N(T-1)}{2}\ln|\mathbf{Q}| - \frac{NT}{2}\ln|\mathbf{R}|$$

- Compute Q_2:

$$
\begin{aligned}
Q_2 &= \mathbb{E}[q_2 \,|\, \mathbf{y}_{1:T}] \\
&= \mathbb{E}\left[-\frac{1}{2} \sum_{n=1}^{N} \left(\mathbf{s}_1^{(n)} - \mathbf{m}_0\right)^T \mathbf{V}_0^{-1} \left(\mathbf{s}_1^{(n)} - \mathbf{m}_0\right) \,\middle|\, \mathbf{y}_{1:T} \right] \\
&= -\frac{1}{2} \sum_{n=1}^{N} \mathbb{E}\left[\left(\mathbf{s}_1^{(n)} - \mathbf{m}_0\right)^T \mathbf{V}_0^{-1} \left(\mathbf{s}_1^{(n)} - \mathbf{m}_0\right) \,\middle|\, \mathbf{y}_{1:T} \right] \\
&= -\frac{1}{2} \sum_{n=1}^{N} \mathbb{E}\left[\operatorname{tr}\left(\left(\mathbf{s}_1^{(n)} - \mathbf{m}_0\right)^T \mathbf{V}_0^{-1} \left(\mathbf{s}_1^{(n)} - \mathbf{m}_0\right) \right) \,\middle|\, \mathbf{y}_{1:T} \right] \\
&= -\frac{1}{2} \sum_{n=1}^{N} \mathbb{E}\left[\operatorname{tr}\left(\left(\mathbf{s}_1^{(n)} - \mathbf{m}_0\right) \left(\mathbf{s}_1^{(n)} - \mathbf{m}_0\right)^T \mathbf{V}_0^{-1} \right) \,\middle|\, \mathbf{y}_{1:T} \right] \\
&= -\frac{1}{2} \sum_{n=1}^{N} \mathbb{E}\left[\operatorname{tr}\left(\left(\mathbf{s}_1^{(n)} - \mathbf{m}_0\right) \left(\mathbf{s}_1^{(n),T} - \mathbf{m}_0^T\right) \mathbf{V}_0^{-1} \right) \,\middle|\, \mathbf{y}_{1:T} \right] \\
&= -\frac{1}{2} \sum_{n=1}^{N} \mathbb{E}\left[\operatorname{tr}\left(\left(\mathbf{s}_1^{(n)} \mathbf{s}_1^{(n),T} - \mathbf{s}_1^{(n)} \mathbf{m}_0^T - \mathbf{m} \mathbf{s}_1^{(n),T} + \mathbf{m}_0 \mathbf{m}_0^T \right) \mathbf{V}_0^{-1} \right) \,\middle|\, \mathbf{y}_{1:T} \right] \\
&= -\frac{1}{2} \sum_{n=1}^{N} \operatorname{tr}\left(\mathbf{P}_1^{(n)} \mathbf{V}_0^{-1} - \hat{\mathbf{s}}_1^{(n)} \mathbf{m}_0^T \mathbf{V}_0^{-1} - \mathbf{m} \hat{\mathbf{s}}_1^{(n),T} \mathbf{V}_0^{-1} + \mathbf{m}_0 \mathbf{m}_0^T \mathbf{V}_0^{-1} \right) \\
&= -\frac{1}{2} \sum_{n=1}^{N} \operatorname{tr}\left(\mathbf{P}_1^{(n)} \mathbf{V}_0^{-1} \right) - \operatorname{tr}\left(\hat{\mathbf{s}}_1^{(n)} \mathbf{m}_0^T \mathbf{V}_0^{-1} \right) - \operatorname{tr}\left(\mathbf{m}_0 \hat{\mathbf{s}}_1^{(n),T} \mathbf{V}_0^{-1} \right) + \operatorname{tr}\left(\mathbf{m}_0 \mathbf{m}_0^T \mathbf{V}_0^{-1} \right) \\
&= -\frac{N}{2} \operatorname{tr}\left(\mathbf{P}_1 \mathbf{V}_0^{-1} \right) + \frac{N}{2} \operatorname{tr}\left(\hat{\mathbf{s}}_1 \mathbf{m}_0^T \mathbf{V}_0^{-1} \right) + \frac{N}{2} \operatorname{tr}\left(\mathbf{m}_0 \hat{\mathbf{s}}_1^T \mathbf{V}_0^{-1} \right) - \frac{N}{2} \operatorname{tr}\left(\mathbf{m}_0 \mathbf{m}_0^T \mathbf{V}_0^{-1} \right)
\end{aligned}
$$

- Compute Q_3:

$$
\begin{aligned}
Q_3 &= \mathbb{E}[q_3 \,|\, \mathbf{y}_{1:T}] \\
&= \mathbb{E}\left[-\frac{1}{2} \sum_{n=1}^{N} \sum_{t=2}^{T} \left(\mathbf{s}_t^{(n)} - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\right)^T \mathbf{Q}^{-1} \left(\mathbf{s}_t^{(n)} - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\right) \,\middle|\, \mathbf{y}_{1:T} \right] \\
&= -\frac{1}{2} \sum_{n=1}^{N} \sum_{t=2}^{T} \mathbb{E}\left[\left(\mathbf{s}_t^{(n)} - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\right)^T \mathbf{Q}^{-1} \left(\mathbf{s}_t^{(n)} - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\right) \,\middle|\, \mathbf{y}_{1:T} \right] \\
&= -\frac{1}{2} \sum_{n=1}^{N} \sum_{t=2}^{T} \mathbb{E}\left[\mathrm{tr}\left(\left(\mathbf{s}_t^{(n)} - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\right)^T \mathbf{Q}^{-1} \left(\mathbf{s}_t^{(n)} - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\right) \right) \,\middle|\, \mathbf{y}_{1:T} \right] \\
&= -\frac{1}{2} \sum_{n=1}^{N} \sum_{t=2}^{T} \mathbb{E}\left[\mathrm{tr}\left(\left(\mathbf{s}_t^{(n)} - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\right) \left(\mathbf{s}_t^{(n)} - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\right)^T \mathbf{Q}^{-1} \right) \,\middle|\, \mathbf{y}_{1:T} \right] \\
&= -\frac{1}{2} \sum_{n=1}^{N} \sum_{t=2}^{T} \mathbb{E}\left[\mathrm{tr}\left(\left(\mathbf{s}_t^{(n)} - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\right) \left(\mathbf{s}_t^{(n),T} - \mathbf{s}_{t-1}^{(n),T}\mathbf{A}^T\right) \mathbf{Q}^{-1} \right) \,\middle|\, \mathbf{y}_{1:T} \right] \\
&= -\frac{1}{2} \sum_{n=1}^{N} \sum_{t=2}^{T} \mathbb{E}\left[\mathrm{tr}\left(\left(\mathbf{s}_t^{(n)}\mathbf{s}_t^{(n),T} - \mathbf{s}_t^{(n)}\mathbf{s}_{t-1}^{(n),T}\mathbf{A}^T - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\mathbf{s}_t^{(n),T} - \mathbf{A}\mathbf{s}_{t-1}^{(n)}\mathbf{s}_{t-1}^{(n),T}\mathbf{A}^T \right) \mathbf{Q}^{-1} \right) \,\middle|\, \mathbf{y}_{1:T} \right] \\
&= -\frac{1}{2} \sum_{n=1}^{N} \sum_{t=2}^{T} \mathrm{tr}\left(\mathbf{P}_t^{(n)}\mathbf{Q}^{-1} - \mathbf{P}_{t,t-1}^{(n)}\mathbf{A}^T\mathbf{Q}^{-1} - \mathbf{A} \underbrace{\mathbf{P}_{t-1,t}^{(n)}}_{=\mathbf{P}_{t,t-1}^{(n)}} \mathbf{Q}^{-1} - \mathbf{A}\mathbf{P}_{t-1}^{(n)}\mathbf{A}^T\mathbf{Q}^{-1} \right) \\
&= -\frac{1}{2} \sum_{n=1}^{N} \sum_{t=2}^{T} \mathrm{tr}\left(\mathbf{P}_t^{(n)}\mathbf{Q}^{-1} \right) - \mathrm{tr}\left(\mathbf{P}_{t,t-1}^{(n)}\mathbf{A}^T\mathbf{Q}^{-1} \right) - \mathrm{tr}\left(\mathbf{A}\mathbf{P}_{t,t-1}^{(n)}\mathbf{Q}^{-1} \right) - \mathrm{tr}\left(\mathbf{A}\mathbf{P}_{t-1}^{(n)}\mathbf{A}^T\mathbf{Q}^{-1} \right) \\
&= -\frac{N}{2} \sum_{t=2}^{T} \mathrm{tr}\left(\mathbf{P}_t\mathbf{Q}^{-1} \right) - \mathrm{tr}\left(\mathbf{P}_{t,t-1}\mathbf{A}^T\mathbf{Q}^{-1} \right) - \mathrm{tr}\left(\mathbf{A}\mathbf{P}_{t,t-1}\mathbf{Q}^{-1} \right) - \mathrm{tr}\left(\mathbf{A}\mathbf{P}_{t-1}\mathbf{A}^T\mathbf{Q}^{-1} \right)
\end{aligned}
$$

- Compute Q_4:

$$
\begin{aligned}
Q_4 &= \mathbb{E}[q_4 \,|\, \mathbf{y}_{1:T}] \\
&= \mathbb{E}\left[-\frac{1}{2} \sum_{n=1}^{N} \sum_{t=1}^{T} \left(\mathbf{y}_t^{(n)} - \mathbf{g}_t^{(n)} \right)^T \mathbf{R}^{-1} \left(\mathbf{y}_t^{(n)} - \mathbf{g}_t^{(n)} \right) \,\middle|\, \mathbf{y}_{1:T} \right] \\
&= -\frac{1}{2} \sum_{n=1}^{N} \sum_{t=1}^{T} \mathbb{E}\left[\left(\mathbf{y}_t^{(n)} - \mathbf{g}_t^{(n)} \right)^T \mathbf{R}^{-1} \left(\mathbf{y}_t^{(n)} - \mathbf{g}_t^{(n)} \right) \,\middle|\, \mathbf{y}_{1:T} \right] \\
&= -\frac{1}{2} \sum_{n=1}^{N} \sum_{t=1}^{T} \mathbb{E}\left[\operatorname{tr}\left(\left(\mathbf{y}_t^{(n)} - \mathbf{g}_t^{(n)} \right)^T \mathbf{R}^{-1} \left(\mathbf{y}_t^{(n)} - \mathbf{g}_t^{(n)} \right) \right) \,\middle|\, \mathbf{y}_{1:T} \right] \\
&= -\frac{1}{2} \sum_{n=1}^{N} \sum_{t=1}^{T} \mathbb{E}\left[\operatorname{tr}\left(\left(\mathbf{y}_t^{(n)} - \mathbf{g}_t^{(n)} \right) \left(\mathbf{y}_t^{(n)} - \mathbf{g}_t^{(n)} \right)^T \mathbf{R}^{-1} \right) \,\middle|\, \mathbf{y}_{1:T} \right] \\
&= -\frac{1}{2} \sum_{n=1}^{N} \sum_{t=1}^{T} \mathbb{E}\left[\operatorname{tr}\left(\left(\mathbf{y}_t^{(n)} - \mathbf{g}_t^{(n)} \right) \left(\mathbf{y}_t^{(n),T} - \mathbf{g}_t^{(n),T} \right) \mathbf{R}^{-1} \right) \,\middle|\, \mathbf{y}_{1:T} \right] \\
&= -\frac{1}{2} \sum_{n=1}^{N} \sum_{t=1}^{T} \mathbb{E}\left[\operatorname{tr}\left(\left(\mathbf{y}_t^{(n)} \mathbf{y}_t^{(n),T} - \mathbf{y}_t^{(n)} \mathbf{g}_t^{(n),T} - \mathbf{g}_t^{(n)} \mathbf{y}_t^{(n),T} + \mathbf{g}_t^{(n)} \mathbf{g}_t^{(n),T} \right) \mathbf{R}^{-1} \right) \,\middle|\, \mathbf{y}_{1:T} \right] \\
&= -\frac{1}{2} \sum_{n=1}^{N} \sum_{t=1}^{T} \operatorname{tr}\left(\mathbf{y}_t^{(n)} \mathbf{y}_t^{(n),T} \mathbf{R}^{-1} - \mathbf{y}_t^{(n)} \hat{\mathbf{g}}_t^{(n),T} \mathbf{R}^{-1} - \hat{\mathbf{g}}_t^{(n)} \mathbf{y}_t^{(n),T} \mathbf{R}^{-1} + \mathbf{G}_t^{(n)} \mathbf{R}^{-1} \right) \\
&= -\frac{1}{2} \sum_{n=1}^{N} \sum_{t=1}^{T} \operatorname{tr}\left(\mathbf{y}_t^{(n)} \mathbf{y}_t^{(n),T} \mathbf{R}^{-1} \right) - \operatorname{tr}\left(\mathbf{y}_t^{(n)} \hat{\mathbf{g}}_t^{(n),T} \mathbf{R}^{-1} \right) - \operatorname{tr}\left(\hat{\mathbf{g}}_t^{(n)} \mathbf{y}_t^{(n),T} \mathbf{R}^{-1} \right) + \operatorname{tr}\left(\mathbf{G}_t^{(n)} \mathbf{R}^{-1} \right)
\end{aligned}
$$

Putting it all together, the expected complete log-likelihood across all observation sequences is given as

$$
\begin{aligned}
Q &= Q_1 + Q_2 + Q_3 + Q_4 \\
&= -\frac{NT(k+p)}{2}\ln(2\pi) - \frac{N}{2}\ln|\mathbf{V}_0| - \frac{N(T-1)}{2}\ln|\mathbf{Q}| - \frac{NT}{2}\ln|\mathbf{R}| \\
&\quad - \frac{N}{2}\operatorname{tr}\Big(\mathbf{P}_1\mathbf{V}_0^{-1}\Big) + \frac{N}{2}\operatorname{tr}\Big(\hat{\mathbf{s}}_1\mathbf{m}_0^T\mathbf{V}_0^{-1}\Big) + \frac{N}{2}\operatorname{tr}\Big(\mathbf{m}_0\hat{\mathbf{s}}_1^T\mathbf{V}_0^{-1}\Big) - \frac{N}{2}\operatorname{tr}\Big(\mathbf{m}_0\mathbf{m}_0^T\mathbf{V}_0^{-1}\Big) \\
&\quad - \frac{N}{2}\sum_{t=2}^{T}\operatorname{tr}\Big(\mathbf{P}_t\mathbf{Q}^{-1}\Big) - \operatorname{tr}\Big(\mathbf{P}_{t,t-1}\mathbf{A}^T\mathbf{Q}^{-1}\Big) - \operatorname{tr}\Big(\mathbf{A}\mathbf{P}_{t,t-1}\mathbf{Q}^{-1}\Big) - \operatorname{tr}\Big(\mathbf{A}\mathbf{P}_{t-1}\mathbf{A}^T\mathbf{Q}^{-1}\Big) \\
&\quad - \frac{1}{2}\sum_{n=1}^{N}\sum_{t=1}^{T}\operatorname{tr}\Big(\mathbf{y}_t^{(n)}\mathbf{y}_t^{(n),T}\mathbf{R}^{-1}\Big) - \operatorname{tr}\Big(\mathbf{y}_t^{(n)}\hat{\mathbf{g}}_t^{(n),T}\mathbf{R}^{-1}\Big) - \operatorname{tr}\Big(\hat{\mathbf{g}}_t^{(n)}\mathbf{y}_t^{(n),T}\mathbf{R}^{-1}\Big) + \operatorname{tr}\Big(\mathbf{G}_t^{(n)}\mathbf{R}^{-1}\Big)
\end{aligned}
$$

where Q_1, Q_2, Q_3 and Q_4 are functions of the parameters $\mathbf{A}$, $\mathbf{Q}$, $\boldsymbol{\theta}$, $\mathbf{R}$, $\mathbf{m}_0$ and $\mathbf{V}_0$:

$$
\begin{aligned}
Q_1 &= Q_1(\mathbf{V}_0, \mathbf{Q}, \mathbf{R}) \\
Q_2 &= Q_2(\mathbf{m}_0, \mathbf{V}_0) \\
Q_3 &= Q_3(\mathbf{A}, \mathbf{Q}) \\
Q_4 &= Q_4(\boldsymbol{\theta}, \mathbf{R})
\end{aligned}
$$

We can now derive the M-step equations by maximizing Q. To maximize Q w.r.t. all parameters, that is

- state dynamics matrix $\mathbf{A}$,
- state noise covariance $\mathbf{Q}$,
- measurement function parameters $\boldsymbol{\theta}$,
- measurement noise covariance $\mathbf{R}$,
- initial state mean $\mathbf{m}_0$ and
- initial state covariance $\mathbf{V}_0$,

we have to take the derivatives w.r.t. to all the above parameters and set them to zero.

- State dynamics matrix $\mathbf{A}$:

$$
\begin{aligned}
\frac{\partial Q}{\partial \mathbf{A}} &= \frac{\partial Q_1}{\partial \mathbf{A}} + \frac{\partial Q_2}{\partial \mathbf{A}} + \frac{\partial Q_3}{\partial \mathbf{A}} + \frac{\partial Q_4}{\partial \mathbf{A}} = \frac{\partial Q_3}{\partial \mathbf{A}} \\
&= -\frac{N}{2}\sum_{t=2}^{T} -\mathbf{Q}^{-1}\mathbf{P}_{t,t-1} - \mathbf{Q}^{-T}\mathbf{P}_{t,t-1}^T + \mathbf{Q}^{-T}\mathbf{A}\mathbf{P}_{t-1}^T + \mathbf{Q}^{-1}\mathbf{A}\mathbf{P}_{t-1} \\
&\overset{(1)}{=} -N\sum_{t=2}^{T} -\mathbf{Q}^{-1}\mathbf{P}_{t,t-1} + \mathbf{Q}^{-1}\mathbf{A}\mathbf{P}_{t-1} \overset{!}{=} \mathbf{0} \\
\implies \qquad \mathbf{A}^{\text{new}} &= \left(\sum_{t=2}^{T}\mathbf{P}_{t,t-1}\right)\left(\sum_{t=2}^{T}\mathbf{P}_{t-1}\right)^{-1}
\end{aligned}
\tag{A.1}
$$

[1] Covariance and correlation matrices ($\mathbf{Q}$, $\mathbf{R}$, $\mathbf{P}_t$, $\mathbf{P}_{t,t-1}$) are symmetric by definition.

- State noise covariance $\mathbf{Q}$:
  Instead of maximizing w.r.t. $\mathbf{Q}$, we can also minimize w.r.t. $\mathbf{Q}^{-1}$ which has the same effect.

$$
\begin{aligned}
\frac{\partial Q}{\partial \mathbf{Q}^{-1}} &= \frac{\partial Q_1}{\partial \mathbf{Q}^{-1}} + \frac{\partial Q_2}{\partial \mathbf{Q}^{-1}} + \frac{\partial Q_3}{\partial \mathbf{Q}^{-1}} + \frac{\partial Q_4}{\partial \mathbf{Q}^{-1}} = \frac{\partial Q_1}{\partial \mathbf{Q}^{-1}} + \frac{\partial Q_3}{\partial \mathbf{Q}^{-1}} \\
&\overset{(2)}{=} \frac{N(T-1)}{2}\mathbf{Q} - \frac{N}{2}\sum_{t=2}^{T} \mathbf{P}_t^T - \mathbf{A}\mathbf{P}_{t,t-1}^T - \mathbf{P}_{t,t-1}^T\mathbf{A}^T + \mathbf{A}\mathbf{P}_{t-1}^T\mathbf{A}^T \\
&= \frac{N(T-1)}{2}\mathbf{Q} - \frac{N}{2}\sum_{t=2}^{T} \mathbf{P}_t - \mathbf{A}\mathbf{P}_{t,t-1} - \mathbf{P}_{t,t-1}\mathbf{A}^T + \mathbf{A}\mathbf{P}_{t-1}\mathbf{A}^T \\
&= \frac{N(T-1)}{2}\mathbf{Q} - \frac{N}{2}\sum_{t=2}^{T} \mathbf{P}_t + \frac{N}{2}\sum_{t=2}^{T} \mathbf{A}\mathbf{P}_{t,t-1} + \frac{N}{2}\sum_{t=2}^{T} \mathbf{P}_{t,t-1}\mathbf{A}^T - \frac{N}{2}\sum_{t=2}^{T} \mathbf{A}\mathbf{P}_{t-1}\mathbf{A}^T \\
&= \frac{N(T-1)}{2}\mathbf{Q} - \frac{N}{2}\sum_{t=2}^{T} \mathbf{P}_t + \frac{N}{2}\mathbf{A}^{\text{new}}\sum_{t=2}^{T} \mathbf{P}_{t,t-1} + \frac{N}{2}\left(\sum_{t=2}^{T} \mathbf{P}_{t,t-1}\right)\left(\mathbf{A}^{\text{new}}\right)^T \\
&\qquad - \frac{N}{2}\mathbf{A}^{\text{new}}\left(\sum_{t=2}^{T} \mathbf{P}_{t-1}\right)\left(\mathbf{A}^{\text{new}}\right)^T \\
&= \frac{N(T-1)}{2}\mathbf{Q} - \frac{N}{2}\sum_{t=2}^{T} \mathbf{P}_t + \frac{N}{2}\mathbf{A}^{\text{new}}\sum_{t=2}^{T} \mathbf{P}_{t,t-1} + \frac{N}{2}\left(\sum_{t=2}^{T} \mathbf{P}_{t,t-1}\right)\left(\sum_{t=2}^{T} \mathbf{P}_{t-1}\right)^{-T}\left(\sum_{t=2}^{T} \mathbf{P}_{t,t-1}\right)^T \\
&\qquad - \frac{N}{2}\left(\sum_{t=2}^{T} \mathbf{P}_{t,t-1}\right)\left(\sum_{t=2}^{T} \mathbf{P}_{t-1}\right)^{-1}\left(\sum_{t=2}^{T} \mathbf{P}_{t-1}\right)\left(\sum_{t=2}^{T} \mathbf{P}_{t-1}\right)^{-T}\left(\sum_{t=2}^{T} \mathbf{P}_{t,t-1}\right)^T \\
&= \frac{N(T-1)}{2}\mathbf{Q} - \frac{N}{2}\sum_{t=2}^{T} \mathbf{P}_t + \frac{N}{2}\mathbf{A}^{\text{new}}\sum_{t=2}^{T} \mathbf{P}_{t,t-1} + \frac{N}{2}\left(\sum_{t=2}^{T} \mathbf{P}_{t,t-1}\right)\left(\sum_{t=2}^{T} \mathbf{P}_{t-1}\right)^{-1}\left(\sum_{t=2}^{T} \mathbf{P}_{t,t-1}\right) \\
&\qquad - \frac{N}{2}\left(\sum_{t=2}^{T} \mathbf{P}_{t,t-1}\right)\left(\sum_{t=2}^{T} \mathbf{P}_{t-1}\right)^{-1}\cancel{\left(\sum_{t=2}^{T} \mathbf{P}_{t-1}\right)\left(\sum_{t=2}^{T} \mathbf{P}_{t-1}\right)^{-1}}\left(\sum_{t=2}^{T} \mathbf{P}_{t,t-1}\right) \\
&= \frac{N(T-1)}{2}\mathbf{Q} - \frac{N}{2}\left(\sum_{t=2}^{T} \mathbf{P}_t - \mathbf{A}^{\text{new}}\sum_{t=2}^{T} \mathbf{P}_{t,t-1}\right) \overset{!}{=} \mathbf{0} \\
\implies \quad \mathbf{Q}^{\text{new}} &= \frac{1}{T-1}\left(\sum_{t=2}^{T} \mathbf{P}_t - \mathbf{A}^{\text{new}}\sum_{t=2}^{T} \mathbf{P}_{t,t-1}\right) \qquad \text{(A.2)}
\end{aligned}
$$

[2] Note that $\frac{\partial}{\partial \mathbf{Q}^{-1}} \ln\det \mathbf{Q} = \frac{\partial}{\partial \mathbf{Q}^{-1}} \ln\det\left(\mathbf{Q}^{-1}\right)^{-1} = \left(\frac{\partial}{\partial \det(\mathbf{Q}^{-1})^{-1}} \ln\det\left(\mathbf{Q}^{-1}\right)^{-1}\right)\left(\frac{\partial}{\partial \mathbf{Q}^{-1}} \det\left(\mathbf{Q}^{-1}\right)^{-1}\right) \overset{\text{cov. matrix}}{=} -\mathbf{Q}$

- Initial state mean $\mathbf{m}_0$:

$$\begin{aligned}
\frac{\partial Q}{\partial \mathbf{m}_0} &= \frac{\partial Q_1}{\partial \mathbf{m}_0} + \frac{\partial Q_2}{\partial \mathbf{m}_0} + \frac{\partial Q_3}{\partial \mathbf{m}_0} + \frac{\partial Q_4}{\partial \mathbf{m}_0} = \frac{\partial Q_2}{\partial \mathbf{m}_0} \\
&= \frac{N}{2}\mathbf{V}_0^{-1}\hat{\mathbf{s}}_1 + \frac{N}{2}\mathbf{V}_0^{-T}\hat{\mathbf{s}}_1 - \frac{N}{2}\left(\mathbf{V}_0^{-1}\mathbf{m}_0 + \mathbf{V}_0^{-T}\mathbf{m}_0\right) \\
&= N\mathbf{V}_0^{-1}\hat{\mathbf{s}}_1 - N\mathbf{V}_0^{-1}\mathbf{m}_0 \\
&= N\mathbf{V}_0^{-1}\left(\hat{\mathbf{s}}_1 - \mathbf{m}_0\right) \stackrel{!}{=} \mathbf{0} \\
\implies \qquad \mathbf{m}_0^{\text{new}} &= \hat{\mathbf{s}}_1 = \frac{1}{N}\sum_{n=1}^{N}\hat{\mathbf{s}}_1^{(n)}
\end{aligned} \tag{A.3}$$

- Initial state covariance $\mathbf{V}_0$:

$$\begin{aligned}
\frac{\partial Q}{\partial \mathbf{V}_0^{-1}} &= \frac{\partial Q_1}{\partial \mathbf{V}_0^{-1}} + \frac{\partial Q_2}{\partial \mathbf{V}_0^{-1}} + \frac{\partial Q_3}{\partial \mathbf{V}_0^{-1}} + \frac{\partial Q_4}{\partial \mathbf{V}_0^{-1}} = \frac{\partial Q_1}{\partial \mathbf{V}_0^{-1}} + \frac{\partial Q_2}{\partial \mathbf{V}_0^{-1}} \\
&= \frac{N}{2}\mathbf{V}_0 - \frac{N}{2}\mathbf{P}_1^T + \frac{N}{2}\mathbf{m}_0\hat{\mathbf{s}}_1^T + \frac{N}{2}\hat{\mathbf{s}}_1\mathbf{m}_0^T - \frac{N}{2}\mathbf{m}_0\mathbf{m}_0^T \\
&= \frac{N}{2}\mathbf{V}_0 - \frac{N}{2}\mathbf{P}_1^T + \frac{N}{2}\hat{\mathbf{s}}_1\hat{\mathbf{s}}_1^T + \frac{N}{2}\hat{\mathbf{s}}_1\hat{\mathbf{s}}_1^T - \frac{N}{2}\hat{\mathbf{s}}_1\hat{\mathbf{s}}_1^T \\
&= \frac{N}{2}\mathbf{V}_0 - \frac{N}{2}\mathbf{P}_1^T + \frac{N}{2}\hat{\mathbf{s}}_1\hat{\mathbf{s}}_1^T \stackrel{!}{=} \mathbf{0} \\
\implies \qquad \mathbf{V}_0^{\text{new}} &= \mathbf{P}_1 - \hat{\mathbf{s}}_1\hat{\mathbf{s}}_1^T
\end{aligned} \tag{A.4}$$

- Measurement noise covariance $\mathbf{R}$:

$$\begin{aligned}
\frac{\partial Q}{\partial \mathbf{R}^{-1}} &= \frac{\partial Q_1}{\partial \mathbf{R}^{-1}} + \frac{\partial Q_2}{\partial \mathbf{R}^{-1}} + \frac{\partial Q_3}{\partial \mathbf{R}^{-1}} + \frac{\partial Q_4}{\partial \mathbf{R}^{-1}} = \frac{\partial Q_1}{\partial \mathbf{R}^{-1}} + \frac{\partial Q_4}{\partial \mathbf{R}^{-1}} \\
&= \frac{NT}{2}\mathbf{R} - \frac{1}{2}\sum_{n=1}^{N}\sum_{t=1}^{T}\mathbf{y}_t^{(n)}\mathbf{y}_t^{(n),T} - \hat{\mathbf{g}}_t^{(n)}\mathbf{y}_t^{(n),T} - \mathbf{y}_t^{(n)}\hat{\mathbf{g}}_t^{(n),T} + \mathbf{G}_t^{(n),T} \stackrel{!}{=} \mathbf{0} \\
\implies \qquad \mathbf{R}^{\text{new}} &= \frac{1}{NT}\sum_{n=1}^{N}\sum_{t=1}^{T}\mathbf{y}_t^{(n)}\mathbf{y}_t^{(n),T} - \hat{\mathbf{g}}_t^{(n)}\mathbf{y}_t^{(n),T} - \mathbf{y}_t^{(n)}\hat{\mathbf{g}}_t^{(n),T} + \mathbf{G}_t^{(n),T}
\end{aligned} \tag{A.5}$$

This concludes the full derivation of the M-step equations. See section 5.1 for the final equations and how to apply the cubature rules to them.

E-Step We will now derive the E-step equations if we where not doing square-root filtering/smoothing. See subsection 4.3.3 and section 5.1 for the equations for that version.

To get the expectations $\hat{\mathbf{s}}_t^{(n)}$, $\mathbf{P}_t^{(n)}$ and $\mathbf{P}_{t,t-1}^{(n)}$, we need to calculate the distribution $p\left(\mathbf{s}_t^{(n)} \mid \mathbf{y}_{1:T}\right)$ for each time step t (and, consequently, for each observation sequence n). This is exactly the posterior distribution calculated by a smoother. Thus we divide the E-step into two parts [Min99]:

1. Filtering: To calculate the posterior distribution $p\left(\mathbf{s}_t^{(n)} \mid \mathbf{y}_{1:t}\right)$, we employ a Gaussian filter similar to the standard Kalman filter.

2. Smoothing: To calculate the smoothed posterior distribution $p\left(\mathbf{s}_t^{(n)} \mid \mathbf{y}_{1:T}\right)$, we employ a Gaussian RTS smoother.

Forward Pass

To derive the forward pass equations, we utilize the groundwork done in [DO11] which concludes that the filter is given as

$$\begin{aligned}\hat{\mathbf{s}}_{t|t}^{(n)} &= \hat{\mathbf{s}}_{t|t-1}^{(n)} + \mathbf{K}_{t|t-1}^{(n)}\left(\mathbf{y}_t^{(n)} - \hat{\mathbf{y}}_{t|t-1}^{(n)}\right)\\ \mathbf{V}_{t|t}^{(n)} &= \mathbf{V}_{t|t-1}^{(n)} - \mathbf{K}_{t|t-1}^{(n)}\mathbf{S}_{t|t-1}^{(n)}\mathbf{K}_{t|t-1}^{(n),T}\end{aligned}$$

with $\mathbf{K}_{t|t-1}^{(n)} = \mathbf{P}_{t|t-1}^{(n)}\left(\mathbf{S}_{t|t-1}^{(n)}\right)^{-1}$ and the expectations and covariances

$$\hat{\mathbf{s}}_{t|t-1}^{(n)} \coloneqq \mathbb{E}_{\mathbf{s}_t^{(n)}}\left[\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \tag{A.6}$$

$$\mathbf{V}_{t|t-1}^{(n)} \coloneqq \mathrm{Cov}_{\mathbf{s}_t^{(n)}}\left[\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \tag{A.7}$$

$$\hat{\mathbf{y}}_{t|t-1}^{(n)} \coloneqq \mathbb{E}_{\mathbf{y}_t^{(n)}}\left[\mathbf{y}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \tag{A.8}$$

$$\mathbf{S}_{t|t-1}^{(n)} \coloneqq \mathrm{Cov}_{\mathbf{y}_t^{(n)}}\left[\mathbf{y}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \tag{A.9}$$

$$\mathbf{P}_{t|t-1}^{(n)} \coloneqq \mathrm{Cov}_{\mathbf{s}_t^{(n)},\mathbf{y}_t^{(n)}}\left[\mathbf{s}_t^{(n)}, \mathbf{y}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \tag{A.10}$$

which are priors for the state, state covariance, measurement, measurement covariance and cross-covariance, respectively. As in our case only the measurements are nonlinear, we can easily evaluate the prior state (A.6)

and the prior covariance (A.7) by exploiting the linearity of the expectation operator:

$$
\begin{aligned}
\hat{\mathbf{s}}_{t|t-1}^{(n)} &= \mathbb{E}_{\mathbf{s}_t^{(n)}}\left[\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \\
&= \mathbb{E}_{\mathbf{s}_{t-1}^{(n)},\mathbf{w}_t^{(n)}}\left[\mathbf{A}\mathbf{s}_{t-1}^{(n)} + \mathbf{w}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \\
&= \mathbf{A}\mathbb{E}_{\mathbf{s}_{t-1}^{(n)}}\left[\mathbf{s}_{t-1}^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \\
&= \mathbf{A}\hat{\mathbf{s}}_{t-1|t-1}^{(n)} \\
\mathbf{V}_{t|t-1}^{(n)} &= \mathrm{Cov}_{\mathbf{s}_t^{(n)}}\left[\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \\
&= \mathrm{Cov}_{\mathbf{s}_{t-1}^{(n)}}\left[\mathbf{A}\mathbf{s}_{t-1}^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] + \mathrm{Cov}_{\mathbf{w}_t^{(n)}}\left[\mathbf{w}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \\
&= \mathbb{E}_{\mathbf{s}_t^{(n)}}\left[\mathbf{s}_t^{(n)}\mathbf{s}_t^{(n),T} \,\middle|\, \mathbf{y}_{1:t-1}\right] - \mathbb{E}_{\mathbf{s}_t^{(n)}}\left[\mathbf{s}_t^{(n)}\right]\mathbb{E}_{\mathbf{s}_t^{(n)}}^T\left[\mathbf{s}_t^{(n)}\right] + \mathbf{Q} \\
&= \mathbb{E}_{\mathbf{s}_{t-1}^{(n)}}\left[\mathbf{A}\mathbf{s}_{t-1}^{(n)}\mathbf{s}_{t-1}^{(n),T}\mathbf{A}^T \,\middle|\, \mathbf{y}_{1:t-1}\right] - \hat{\mathbf{s}}_{t|t-1}^{(n)}\hat{\mathbf{s}}_{t|t-1}^{(n),T} + \mathbf{Q} \\
&= \mathbf{A}\mathbb{E}_{\mathbf{s}_{t-1}^{(n)}}\left[\mathbf{s}_{t-1}^{(n)}\mathbf{s}_{t-1}^{(n),T} \,\middle|\, \mathbf{y}_{1:t-1}\right]\mathbf{A}^T - \mathbf{A}\hat{\mathbf{s}}_{t-1|t-1}^{(n)}\hat{\mathbf{s}}_{t-1|t-1}^{(n),T}\mathbf{A}^T + \mathbf{Q} \\
&= \mathbf{A}\left(\mathbb{E}_{\mathbf{s}_{t-1}^{(n)}}\left[\mathbf{s}_{t-1}^{(n)}\mathbf{s}_{t-1}^{(n),T} \,\middle|\, \mathbf{y}_{1:t-1}\right] - \hat{\mathbf{s}}_{t-1|t-1}^{(n)}\hat{\mathbf{s}}_{t-1|t-1}^{(n),T}\right)\mathbf{A}^T + \mathbf{Q} \\
&= \mathbf{A}\,\mathrm{Cov}_{\mathbf{s}_{t-1}^{(n)}}\left[\mathbf{s}_{t-1}^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] + \mathbf{Q} \\
&= \mathbf{A}\mathbf{V}_{t-1|t-1}^{(n)}\mathbf{A}^T + \mathbf{Q}
\end{aligned}
$$

But we are using a nonlinear measurement function $\mathbf{g}(\cdot)$. Hence, it is not possible to compute the integrals produced by the expectations/covariances (A.8), (A.9) and (A.10). As in the derivation of the M-step, we apply cubature methods to approximate the integrals:

$$
\begin{aligned}
\hat{\mathbf{y}}_{t|t-1}^{(n)} &= \mathbb{E}_{\mathbf{y}_t^{(n)}}\left[\mathbf{y}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \\
&= \mathbb{E}_{\mathbf{s}_t^{(n)},\mathbf{v}_t^{(n)}}\left[\mathbf{g}_t^{(n)} + \mathbf{v}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \\
&= \mathbb{E}_{\mathbf{s}_t^{(n)}}\left[\mathbf{g}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \\
&= \int \mathbf{g}_t^{(n)} p\left(\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right) \mathrm{d}\mathbf{s}_t^{(n)} \\
&= \int \mathbf{g}_t^{(n)} \mathcal{N}\left(\hat{\mathbf{s}}_{t|t-1}^{(n)}, \mathbf{V}_{t|t-1}^{(n)}\right) \mathrm{d}\mathbf{s}_t^{(n)} \\
&\approx SRC\left[\mathbf{g};\, \hat{\mathbf{s}}_{t|t-1}^{(n)}, \mathbf{V}_{t|t-1}^{(n)}\right]
\end{aligned}
\tag{A.11}
$$

$$
\begin{aligned}
\mathbf{S}_{t|t-1}^{(n)} &= \mathrm{Cov}_{\mathbf{y}_t^{(n)}}\left[\mathbf{y}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \\
&= \mathrm{Cov}_{\mathbf{s}_t^{(n)}}\left[\mathbf{g}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] + \mathrm{Cov}_{\mathbf{v}_t^{(n)}}\left[\mathbf{v}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \\
&= \mathbb{E}_{\mathbf{s}_t^{(n)}}\left[\mathbf{g}_t^{(n)}\mathbf{g}_t^{(n),T} \,\middle|\, \mathbf{y}_{1:t-1}\right] - \mathbb{E}_{\mathbf{s}_t^{(n)}}\left[\mathbf{g}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right]\mathbb{E}_{\mathbf{s}_t^{(n)}}^{T}\left[\mathbf{g}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] + \mathbf{R} \\
&= \int \mathbf{g}_t^{(n)}\mathbf{g}_t^{(n),T} p\left(\mathbf{s}_t^{(n)} \,|\, \mathbf{y}_{1:t-1}\right) \mathrm{d}\mathbf{s}_t^{(n)} - \hat{\mathbf{y}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T} + \mathbf{R} \\
&= \int \mathbf{g}_t^{(n)}\mathbf{g}_t^{(n),T} \mathcal{N}\left(\hat{\mathbf{s}}_{t|t-1}^{(n)}, \mathbf{V}_{t|t-1}^{(n)}\right) \mathrm{d}\mathbf{s}_t^{(n)} - \hat{\mathbf{y}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T} + \mathbf{R} \\
&\approx SRC\left[\mathbf{g}\mathbf{g}^T; \hat{\mathbf{s}}_{t|t-1}^{(n)}, \mathbf{V}_{t|t-1}^{(n)}\right] - \hat{\mathbf{y}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T} + \mathbf{R}
\end{aligned}
\tag{A.12}
$$

$$
\begin{aligned}
\mathbf{P}_{t|t-1}^{(n)} &= \mathrm{Cov}_{\mathbf{s}_t^{(n)},\mathbf{y}_t^{(n)}}\left[\mathbf{s}_t^{(n)}, \mathbf{y}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \\
&= \mathbb{E}_{\mathbf{s}_t^{(n)},\mathbf{y}_t^{(n)}}\left[\mathbf{s}_t^{(n)}\mathbf{y}_t^{(n),T} \,\middle|\, \mathbf{y}_{1:t-1}^{(n)}\right] - \mathbb{E}_{\mathbf{s}_t^{(n)}}\left[\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right]\mathbb{E}_{\mathbf{y}_t^{(n)}}^{T}\left[\mathbf{y}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \\
&= \mathbb{E}_{\mathbf{s}_t^{(n)},\mathbf{y}_t^{(n)}}\left[\mathbf{s}_t^{(n)}\mathbf{y}_t^{(n),T} \,\middle|\, \mathbf{y}_{1:t-1}\right] - \hat{\mathbf{s}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T} \\
&= \mathbb{E}_{\mathbf{s}_t^{(n)}}\left[\mathbf{s}_t^{(n)}\mathbf{g}_t^{(n),T} \,\middle|\, \mathbf{y}_{1:t-1}\right] - \hat{\mathbf{s}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T} \\
&= \int \mathbf{s}_t^{(n)}\mathbf{g}_t^{(n),T} p\left(\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right) \mathrm{d}\mathbf{s}_t^{(n)} - \hat{\mathbf{s}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T} \\
&= \int \mathbf{s}_t\mathbf{g}_t^{(n),T} \mathcal{N}\left(\hat{\mathbf{s}}_{t|t-1}^{(n)}, \mathbf{V}_{t|t-1}^{(n)}\right) \mathrm{d}\mathbf{s}_t - \hat{\mathbf{s}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T} \\
&\approx SRC\left[\mathbf{g}\mathbf{g}^T; \hat{\mathbf{s}}_{t|t-1}^{(n)}, \mathbf{V}_{t|t-1}^{(n)}\right] - \hat{\mathbf{s}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T}
\end{aligned}
\tag{A.13}
$$

This completes the derivation of the forward pass equations.

For completeness, we summarize all of them here:

$$
\begin{aligned}
\hat{\mathbf{s}}_{t|t-1}^{(n)} &= \mathbf{A}\hat{\mathbf{s}}_{t-1|t-1}^{(n)} \\
\mathbf{V}_{t|t-1}^{(n)} &= \mathbf{A}\mathbf{V}_{t-1|t-1}^{(n)}\mathbf{A}^T + \mathbf{Q} \\
\hat{\mathbf{y}}_{t|t-1}^{(n)} &= SRC\left[\mathbf{g}; \hat{\mathbf{s}}_{t|t-1}^{(n)}, \mathbf{V}_{t|t-1}^{(n)}\right] \\
\mathbf{S}_{t|t-1}^{(n)} &= SRC\left[\mathbf{g}\mathbf{g}^T; \hat{\mathbf{s}}_{t|t-1}^{(n)}, \mathbf{V}_{t|t-1}^{(n)}\right] - \hat{\mathbf{y}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T} + \mathbf{R} \\
\mathbf{P}_{t|t-1}^{(n)} &= SRC\left[\mathbf{g}\mathbf{g}^T; \hat{\mathbf{s}}_{t|t-1}^{(n)}, \mathbf{V}_{t|t-1}^{(n)}\right] - \hat{\mathbf{s}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T} \\
\mathbf{K}_{t|t-1}^{(n)} &= \mathbf{P}_{t|t-1}^{(n)}\left(\mathbf{S}_{t|t-1}^{(n)}\right)^{-1} \\
\hat{\mathbf{s}}_{t|t}^{(n)} &= \hat{\mathbf{s}}_{t|t-1}^{(n)} + \mathbf{K}_{t|t-1}\left(\mathbf{y}_t^{(n)} - \hat{\mathbf{y}}_{t|t-1}^{(n)}\right) \\
\mathbf{V}_{t|t}^{(n)} &= \mathbf{V}_{t|t-1}^{(n)} - \mathbf{K}_{t|t-1}^{(n)}\mathbf{S}_{t|t-1}^{(n)}\mathbf{K}_{t|t-1}^{(n),T}
\end{aligned}
$$

The forward pass is initialized with $\hat{\mathbf{s}}_{0|0}^{(n)} = \mathbf{m}_0$ and $\mathbf{V}_{0|0}^{(n)} = \mathbf{V}_0$ for all $n = 1, 2, \cdots, N$.

Backward Pass

To derive the backward pass equations, we utilize the groundwork done in [DO11] which concludes that the smoother is given as

$$\hat{\mathbf{s}}_{t-1|T}^{(n)} = \hat{\mathbf{s}}_{t-1|t-1}^{(n)} + \mathbf{J}_{t-1}^{(n)}\left(\hat{\mathbf{s}}_{t|T}^{(n)} - \hat{\mathbf{s}}_{t|t-1}^{(n)}\right)$$
$$\mathbf{V}_{t-1|T}^{(n)} = \mathbf{V}_{t-1|t-1}^{(n)} + \mathbf{J}_{t-1}^{(n)}\left(\mathbf{V}_{t|T}^{(n)} - \mathbf{V}_{t|t-1}^{(n)}\right)\mathbf{J}_{t-1}^{(n),T}$$

with:

$$\mathbf{J}_{t-1}^{(n)} \coloneqq \mathbf{V}_{t-1,t|t-1}^{(n)}\left(\mathbf{V}_{t|t-1}^{(n)}\right)^{-1}$$
$$\mathbf{V}_{t-1,t|t-1}^{(n)} \coloneqq \mathrm{Cov}_{\mathbf{s}_{t-1}^{(n)},\mathbf{s}_t^{(n)}}\left[\mathbf{s}_{t-1}^{(n)}, \mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \tag{A.14}$$

Like for the filter, we can easily evaluate the cross-covariance matrix $\mathbf{V}_{t-1,t|t-1}$ by exploiting the linearity of the expectation:

$$\begin{aligned}
\mathbf{V}_{t-1,t|t-1}^{(n)} &= \mathrm{Cov}_{\mathbf{s}_{t-1}^{(n)},\mathbf{s}_t^{(n)}}\left[\mathbf{s}_{t-1}^{(n)}, \mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \\
&= \mathbb{E}_{\mathbf{s}_{t-1}^{(n)},\mathbf{s}_t^{(n)}}\left[\mathbf{s}_{t-1}^{(n)}\mathbf{s}_t^{(n),T} \,\middle|\, \mathbf{y}_{1:t-1}\right] - \mathbb{E}_{\mathbf{s}_{t-1}^{(n)}}\left[\mathbf{s}_{t-1}^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right]\mathbb{E}_{\mathbf{s}_t^{(n)}}^{T}\left[\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right] \\
&\overset{(3)}{=} \mathbb{E}_{\mathbf{s}_{t-1}^{(n)}}\left[\mathbf{s}_{t-1}^{(n)}\mathbf{s}_{t-1}^{(n),T}\mathbf{A}^T \,\middle|\, \mathbf{y}_{1:t-1}\right] - \hat{\mathbf{s}}_{t-1|t-1}^{(n)}\hat{\mathbf{s}}_{t|t-1}^{(n),T} \\
&= \mathbb{E}_{\mathbf{s}_{t-1}^{(n)}}\left[\mathbf{s}_{t-1}^{(n)}\mathbf{s}_{t-1}^{(n),T} \,\middle|\, \mathbf{y}_{1:t-1}\right]\mathbf{A}^T - \hat{\mathbf{s}}_{t-1|t-1}^{(n)}\hat{\mathbf{s}}_{t-1|t-1}^{(n),T}\mathbf{A}^T \\
&= \left(\mathbb{E}_{\mathbf{s}_{t-1}^{(n)}}\left[\mathbf{s}_{t-1}^{(n)}\mathbf{s}_{t-1}^{(n),T} \,\middle|\, \mathbf{y}_{1:t-1}\right] - \hat{\mathbf{s}}_{t-1|t-1}^{(n)}\hat{\mathbf{s}}_{t-1|t-1}^{(n),T}\right)\mathbf{A}^T \\
&= \mathrm{Cov}_{\mathbf{s}_{t-1}^{(n)}}\left[\mathbf{s}_{t-1}^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right]\mathbf{A}^T \\
&= \mathbf{V}_{t-1|t-1}^{(n)}\mathbf{A}^T
\end{aligned}$$

This completes the derivation of the backward pass equations.

For completeness, we summarize all of them here:

$$\mathbf{V}_{t-1,t|t-1}^{(n)} = \mathbf{V}_{t-1|t-1}^{(n)}\mathbf{A}^T$$
$$\mathbf{J}_{t-1}^{(n)} = \mathbf{V}_{t-1,t|t-1}^{(n)}\left(\mathbf{V}_{t|t-1}^{(n)}\right)^{-1}$$
$$\hat{\mathbf{s}}_{t-1|T}^{(n)} = \hat{\mathbf{s}}_{t-1|t-1}^{(n)} + \mathbf{J}_{t-1}\left(\hat{\mathbf{s}}_{t|T}^{(n)} - \hat{\mathbf{s}}_{t|t-1}^{(n)}\right) \tag{A.15}$$
$$\mathbf{V}_{t-1|T}^{(n)} = \mathbf{V}_{t-1|t-1}^{(n)} + \mathbf{J}_{t-1}^{(n)}\left(\mathbf{V}_{t|T}^{(n)} - \mathbf{V}_{t|t-1}^{(n)}\right)\mathbf{J}_{t-1}^{(n),T} \tag{A.16}$$

[3] Due to the independence and zero mean of the noise, it disappears in the expectation: $\mathbb{E}_{\mathbf{s}_{t-1}^{(n)},\mathbf{s}_t^{(n)}}\left[\mathbf{s}_{t-1}^{(n)}\left(\mathbf{s}_t^{(n)}\right)^T \,\middle|\, \mathbf{y}_{1:t-1}\right] = \mathbb{E}_{\mathbf{s}_{t-1}^{(n)},\mathbf{w}_t^{(n)}}\left[\mathbf{s}_{t-1}^{(n)}\left(\left(\mathbf{s}_{t-1}^{(n)}\right)^T\mathbf{A}^T + \left(\mathbf{w}_t^{(n)}\right)^T\right) \,\middle|\, \mathbf{y}_{1:t-1}\right] = \mathbb{E}_{\mathbf{s}_{t-1}^{(n)},\mathbf{w}_t^{(n)}}\left[\mathbf{s}_{t-1}^{(n)}\left(\mathbf{s}_{t-1}^{(n)}\right)^T\mathbf{A}^T + \mathbf{s}_{t-1}^{(n)}\left(\mathbf{w}_t^{(n)}\right)^T \,\middle|\, \mathbf{y}_{1:t-1}\right] = \mathbb{E}_{\mathbf{s}_{t-1}^{(n)},\mathbf{w}_t^{(n)}}\left[\mathbf{s}_{t-1}^{(n)}\left(\mathbf{s}_{t-1}^{(n)}\right)^T\mathbf{A}^T \,\middle|\, \mathbf{y}_{1:t-1}\right]$

According to [Min99], the self- and cross-correlations are given as

$$\mathbf{P}_t^{(n)} = \mathbf{V}_{t|T}^{(n)} + \hat{\mathbf{s}}_{t|T}^{(n)} \hat{\mathbf{s}}_{t|T}^{(n),T}$$
$$\mathbf{P}_{t,t-1}^{(n)} = \mathbf{J}_{t-1}^{(n)} \mathbf{V}_{t|T}^{(n)} + \hat{\mathbf{s}}_{t|T}^{(n)} \hat{\mathbf{s}}_{t-1|T}^{(n),T}$$

This wraps up the derivation of the E-step when not using square-root filtering. See section 5.1 for that version.

A.2. Exactness for Linear Observations

If we restrict the previous generalization of the nonlinear observation function $\mathbf{g}_{\boldsymbol{\theta}} : \mathbb{R}^k \to \mathbb{R}^p$ to a linear function $\mathbf{g}(\mathrm{s}) = \mathbf{C}\mathrm{s}$ where $\boldsymbol{\theta} = \{\mathbf{C}\}$, we can show that the approximations for $\hat{\mathbf{g}}_t^{(n)}$, $\mathbf{G}_t^{(n)}$, $\hat{\mathbf{y}}_{t|t-1}^{(n)}$, $\mathbf{S}_{t|t-1}^{(n)}$ and $\mathbf{K}_{t|t-1}^{(n)}$ are exact, i.e. they match the real expectation/covariance value.

Theorem A.1. *The spherical-radial cubature approximations for,* $\hat{\mathbf{g}}_t^{(n)}$, $\mathbf{G}_t^{(n)}$, $\hat{\mathbf{y}}_{t|t-1}^{(n)}$, $\mathbf{S}_{t|t-1}$ *and* $\mathbf{K}_{t|t-1}$ *are exact (i.e. they equal the analytical result) if the observation function is linear.*

Proof. Let $\mathbf{C} \in \mathbb{R}^{p \times k}$ be a matrix such that $\mathbf{g}(\cdot)$ is linear and given as $\mathbf{g}(\mathbf{s}) = \mathbf{C}\mathbf{s}$. Firstly, we will evaluate the analytical values of the given quantities.

- Expected observation $\hat{\mathbf{g}}_t^{(n)} = \mathbb{E}\left[\mathbf{g}_t^{(n)} \,\middle|\, \mathbf{y}_{1:T}\right]$:

$$\hat{\mathbf{g}}_t^{(n)} = \int \mathbf{C}\mathbf{s}_t^{(n)} p\left(\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:T}\right) \mathrm{d}\mathbf{s}_t^{(n)} = \mathbf{C} \int \mathbf{s}_t^{(n)} p\left(\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:T}\right) \mathrm{d}\mathbf{s}_t^{(n)} = \mathbf{C}\hat{\mathbf{s}}_t^{(n)}$$

- Observation correlation $\mathbf{G}_t^{(n)} = \mathbb{E}\left[\mathbf{g}_t^{(n)} \mathbf{g}_t^{(n),T} \,\middle|\, \mathbf{y}_{1:T}\right]$:

$$\mathbf{G}_t^{(n)} = \int \mathbf{C}\mathbf{s}_t^{(n)} \mathbf{s}_t^{(n),T} \mathbf{C}^T p\left(\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:T}\right) \mathrm{d}\mathbf{s}_t^{(n)} = \mathbf{C}\left(\int \mathbf{s}_t^{(n)} \mathbf{s}_t^{(n),T} p\left(\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:T}\right) \mathrm{d}\mathbf{s}_t^{(n)}\right)\mathbf{C}^T = \mathbf{C}\mathbf{P}_t^{(n)}\mathbf{C}^T$$

- Expected prior observation $\hat{\mathbf{y}}_{t|t-1}^{(n)} = \mathbb{E}_{\mathbf{s}_t^{(n)}}\left[\mathbf{g}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right]$:

$$\hat{\mathbf{y}}_{t|t-1}^{(n)} = \int \mathbf{C}\mathbf{s}_t^{(n)} p\left(\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right) \mathrm{d}\mathbf{s}_t^{(n)} = \mathbf{C} \int \mathbf{s}_t^{(n)} p\left(\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right) \mathrm{d}\mathbf{s}_t^{(n)} = \mathbf{C}\hat{\mathbf{s}}_{t|t-1}$$

- Prior observation covariance $\mathbf{S}_{t|t-1}^{(n)} = \mathrm{Cov}_{\mathbf{y}_t^{(n)}}\left[\mathbf{y}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right]$:

$$
\begin{aligned}
\mathbf{S}_{t|t-1}^{(n)} &= \int \mathbf{C}\mathbf{s}_t^{(n)}\mathbf{s}_t^{(n),T}\mathbf{C}^T p\left(\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right) \mathrm{d}\mathbf{s}_t^{(n)} - \hat{\mathbf{y}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T} + \mathbf{R} \\
&= \mathbf{C}\left(\int \mathbf{s}_t^{(n)}\mathbf{s}_t^{(n),T} p\left(\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right) \mathrm{d}\mathbf{s}_t^{(n)}\right)\mathbf{C}^T - \mathbf{C}\hat{\mathbf{s}}_{t|t-1}^{(n)}\hat{\mathbf{s}}_{t|t-1}^{(n),T}\mathbf{C}^T + \mathbf{R} \\
&= \mathbf{C}\underbrace{\left(\int \mathbf{s}_t^{(n)}\mathbf{s}_t^{(n),T} p\left(\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right) \mathrm{d}\mathbf{s}_t^{(n)} - \hat{\mathbf{s}}_{t|t-1}^{(n)}\hat{\mathbf{s}}_{t|t-1}^{(n),T}\right)}_{=\mathrm{Cov}_{\mathbf{s}_t^{(n)}}\left[\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right]}\mathbf{C}^T + \mathbf{R} \\
&= \mathbf{C}\mathbf{V}_{t|t-1}^{(n)}\mathbf{C}^T + \mathbf{R}
\end{aligned}
$$

- Prior observation-state covariance $\mathbf{P}_{t|t-1}^{(n)} = \mathrm{Cov}_{\mathbf{s}_t^{(n)},\mathbf{y}_t^{(n)}}\left[\mathbf{s}_t^{(n)}, \mathbf{y}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right]$:

$$
\begin{aligned}
\mathbf{P}_{t|t-1}^{(n)} &= \int \mathbf{s}_t^{(n)}\mathbf{s}_t^{(n),T}\mathbf{C}^T p\left(\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right) \mathrm{d}\mathbf{s}_t^{(n)} - \hat{\mathbf{s}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T} \\
&= \left(\int \mathbf{s}_t^{(n)}\mathbf{s}_t^{(n),T} p\left(\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right) \mathrm{d}\mathbf{s}_t^{(n)}\right)\mathbf{C}^T - \hat{\mathbf{s}}_{t|t-1}^{(n)}\hat{\mathbf{s}}_{t|t-1}^{(n),T}\mathbf{C}^T \\
&= \underbrace{\left(\int \mathbf{s}_t^{(n)}\mathbf{s}_t^{(n),T} p\left(\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right) \mathrm{d}\mathbf{s}_t^{(n)} - \hat{\mathbf{s}}_{t|t-1}^{(n)}\hat{\mathbf{s}}_{t|t-1}^{(n),T}\right)}_{=\mathrm{Cov}_{\mathbf{s}_t^{(n)}}\left[\mathbf{s}_t^{(n)} \,\middle|\, \mathbf{y}_{1:t-1}\right]}\mathbf{C}^T \\
&= \mathbf{V}_{t|t-1}^{(n)}\mathbf{C}^T
\end{aligned}
$$

These match exactly the well-known Kalman filter equations.

We now plug the observation function $\mathbf{g}(\mathbf{s}) = \mathbf{C}\mathbf{s}$ into the various cubature rules:

- Expected observation $\hat{\mathbf{g}}_t^{(n)}$:

$$
\hat{\mathbf{g}}_t^{(n)} \approx \frac{1}{2k}\sum_{i=1}^{2k} \mathbf{g}\left(\sqrt{\mathbf{V}_t^{(n)}}\boldsymbol{\xi}_i + \hat{\mathbf{s}}_t^{(n)}\right) = \frac{1}{2k}\sum_{i=1}^{2k} \mathbf{C}\sqrt{\mathbf{V}_t^{(n)}}\boldsymbol{\xi}_i + \mathbf{C}\hat{\mathbf{s}}_t^{(n)} = \mathbf{C}\hat{\mathbf{s}}_t^{(n)}
$$

- Observation correlation $\mathbf{G}_t^{(n)}$:

$$
\begin{aligned}
\mathbf{G}_t^{(n)} &\approx \frac{1}{2k}\sum_{i=1}^{2k}\mathbf{g}\left(\sqrt{\mathbf{V}_t^{(n)}}\boldsymbol{\xi}_i+\hat{\mathbf{s}}_t^{(n)}\right)\mathbf{g}^T\left(\sqrt{\mathbf{V}_t^{(n)}}\boldsymbol{\xi}_i+\hat{\mathbf{s}}_t^{(n)}\right)\\
&= \frac{1}{2k}\sum_{i=1}^{2k}\mathbf{C}\left(\sqrt{\mathbf{V}_t^{(n)}}\boldsymbol{\xi}_i+\hat{\mathbf{s}}_t^{(n)}\right)\left(\sqrt{\mathbf{V}_t^{(n)}}\boldsymbol{\xi}_i+\hat{\mathbf{s}}_t^{(n)}\right)^T\mathbf{C}^T\\
&= \frac{1}{2k}\sum_{i=1}^{2k}\mathbf{C}\left(\sqrt{\mathbf{V}_t^{(n)}}\boldsymbol{\xi}_i\boldsymbol{\xi}_i^T\sqrt{\mathbf{V}_t^{(n)}}^T+\sqrt{\mathbf{V}_t^{(n)}}\boldsymbol{\xi}_i\hat{\mathbf{s}}_t^{(n),T}+\hat{\mathbf{s}}_t^{(n)}\boldsymbol{\xi}_i^T\sqrt{\mathbf{V}_t^{(n)}}^T+\hat{\mathbf{s}}_t^{(n)}\hat{\mathbf{s}}_t^{(n),T}\right)\mathbf{C}^T\\
&= \frac{1}{2k}\sum_{i=1}^{2k}\mathbf{C}\sqrt{\mathbf{V}_t^{(n)}}\boldsymbol{\xi}_i\boldsymbol{\xi}_i^T\sqrt{\mathbf{V}_t^{(n)}}^T\mathbf{C}^T+\frac{1}{2k}\sum_{i=1}^{2k}\mathbf{C}^T\hat{\mathbf{s}}_t^{(n)}\hat{\mathbf{s}}_t^{(n),T}\mathbf{C}^T\\
&\qquad+\cancel{\frac{1}{2k}\sum_{i=1}^{2k}\mathbf{C}\sqrt{\mathbf{V}_t^{(n)}}\boldsymbol{\xi}_i\hat{\mathbf{s}}_t^{(n),T}\mathbf{C}^T}+\cancel{\frac{1}{2k}\sum_{i=1}^{2k}\mathbf{C}\hat{\mathbf{s}}_t^{(n)}\boldsymbol{\xi}_i^T\sqrt{\mathbf{V}_t^{(n),T}}\mathbf{C}^T}\\
&= \mathbf{C}\sqrt{\mathbf{V}_t^{(n)}}\cancel{\left(\sum_{i=1}^{k}\mathbf{e}_i\mathbf{e}_i^T\right)}\sqrt{\mathbf{V}_t^{(n)}}^T\mathbf{C}^T+\mathbf{C}^T\hat{\mathbf{s}}_t^{(n)}\hat{\mathbf{s}}_t^{(n),T}\mathbf{C}^T\\
&= \mathbf{C}\mathbf{V}_t\mathbf{C}^T+\mathbf{C}^T\hat{\mathbf{s}}_t^{(n)}\hat{\mathbf{s}}_t^{(n),T}\mathbf{C}^T\\
&= \mathbf{C}\mathbf{P}_t^{(n)}\mathbf{C}^T
\end{aligned}
$$

- Expected prior observation $\hat{\mathbf{y}}_{t|t-1}^{(n)}$:

$$
\hat{\mathbf{y}}_{t|t-1}^{(n)} \approx \frac{1}{2k}\sum_{i=1}^{2k}\mathbf{g}\left(\sqrt{\mathbf{V}_{t|t-1}^{(n)}}\boldsymbol{\xi}_i+\hat{\mathbf{s}}_{t|t-1}^{(n)}\right) = \frac{1}{2k}\sum_{i=1}^{2k}\mathbf{C}\sqrt{\mathbf{V}_{t|t-1}^{(n)}}\boldsymbol{\xi}_i+\mathbf{C}\hat{\mathbf{s}}_{t|t-1}^{(n)} = \mathbf{C}\hat{\mathbf{s}}_{t|t-1}^{(n)}
$$

- Prior observation covariance $\mathbf{S}_{t|t-1}$:

$$
\begin{aligned}
\mathbf{S}_{t|t-1}^{(n)} &\approx \frac{1}{2k}\sum_{i=1}^{2k}\mathbf{g}\left(\sqrt{\mathbf{V}_{t|t-1}^{(n)}}\boldsymbol{\xi}_i+\hat{\mathbf{s}}_{t|t-1}^{(n)}\right)\mathbf{g}^T\left(\sqrt{\mathbf{V}_{t|t-1}^{(n)}}\boldsymbol{\xi}_i+\hat{\mathbf{s}}_{t|t-1}^{(n)}\right)-\hat{\mathbf{y}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T}+\mathbf{R}\\
&= \frac{1}{2k}\sum_{i=1}^{2k}\mathbf{C}\left(\sqrt{\mathbf{V}_{t|t-1}^{(n)}}\boldsymbol{\xi}_i+\hat{\mathbf{s}}_{t|t-1}^{(n)}\right)\left(\sqrt{\mathbf{V}_{t|t-1}^{(n)}}\boldsymbol{\xi}_i+\hat{\mathbf{s}}_{t|t-1}^{(n)}\right)^T\mathbf{C}^T-\hat{\mathbf{y}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T}+\mathbf{R}\\
&= \frac{1}{2k}\sum_{i=1}^{2k}\mathbf{C}\sqrt{\mathbf{V}_{t|t-1}^{(n)}}\boldsymbol{\xi}_i\boldsymbol{\xi}_i^T\sqrt{\mathbf{V}_{t|t-1}^{(n)}}^T\mathbf{C}^T+\cancel{\frac{1}{2k}\sum_{i=1}^{2k}\mathbf{C}\sqrt{\mathbf{V}_{t|t-1}^{(n)}}\boldsymbol{\xi}_i\hat{\mathbf{s}}_{t|t-1}^{(n),T}\mathbf{C}^T}\\
&\qquad+\cancel{\frac{1}{2k}\sum_{i=1}^{2k}\mathbf{C}\hat{\mathbf{s}}_{t|t-1}^{(n)}\boldsymbol{\xi}_i^T\sqrt{\mathbf{V}_{t|t-1}^{(n)}}^T}\mathbf{C}^T+\frac{1}{2k}\sum_{i=1}^{2k}\underbrace{\mathbf{C}\hat{\mathbf{s}}_{t|t-1}^{(n)}\hat{\mathbf{s}}_{t|t-1}^{(n),T}\mathbf{C}^T}_{=\hat{\mathbf{y}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T}}-\hat{\mathbf{y}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T}+\mathbf{R}\\
&= \frac{1}{2k}\sum_{i=1}^{2k}\mathbf{C}\sqrt{\mathbf{V}_{t|t-1}^{(n)}}\boldsymbol{\xi}_i\boldsymbol{\xi}_i^T\sqrt{\mathbf{V}_{t|t-1}^{(n)}}^T\mathbf{C}^T+\hat{\mathbf{y}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T}-\hat{\mathbf{y}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T}+\mathbf{R}\\
&= \mathbf{C}\mathbf{V}_{t|t-1}\mathbf{C}^T+\mathbf{R}
\end{aligned}
$$

- Prior observation-state covariance $\mathbf{P}_{t|t-1}^{(n)}$:

$$
\begin{aligned}
\mathbf{P}_{t|t-1}^{(n)} &\approx \frac{1}{2k}\sum_{i=1}^{2k}\left(\sqrt{\mathbf{V}_{t|t-1}^{(n)}}\boldsymbol{\xi}_i + \hat{\mathbf{s}}_{t|t-1}^{(n)}\right)\mathbf{g}^T\left(\sqrt{\mathbf{V}_{t|t-1}^{(n)}}\boldsymbol{\xi}_i + \hat{\mathbf{s}}_{t|t-1}^{(n)}\right) - \hat{\mathbf{s}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T} \\
&= \frac{1}{2k}\sum_{i=1}^{2k}\left(\sqrt{\mathbf{V}_{t|t-1}^{(n)}}\boldsymbol{\xi}_i + \hat{\mathbf{s}}_{t|t-1}^{(n)}\right)\left(\sqrt{\mathbf{V}_{t|t-1}^{(n)}}\boldsymbol{\xi}_i + \hat{\mathbf{s}}_{t|t-1}^{(n)}\right)^T\mathbf{C}^T - \hat{\mathbf{s}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T} \\
&= \frac{1}{2k}\sum_{i=1}^{2k}\sqrt{\mathbf{V}_{t|t-1}^{(n)}}\boldsymbol{\xi}_i\boldsymbol{\xi}_i^T\sqrt{\mathbf{V}_{t|t-1}^{(n)}}^T\mathbf{C}^T + \cancel{\frac{1}{2k}\sum_{i=1}^{2k}\sqrt{\mathbf{V}_{t|t-1}^{(n)}}\boldsymbol{\xi}_i\hat{\mathbf{s}}_{t|t-1}^{(n),T}\mathbf{C}^T} \\
&\qquad + \cancel{\frac{1}{2k}\sum_{i=1}^{2k}\hat{\mathbf{s}}_{t|t-1}^{(n)}\boldsymbol{\xi}_i^T\sqrt{\mathbf{V}_{t|t-1}^{(n)}}^T\mathbf{C}^T} + \frac{1}{2k}\sum_{i=1}^{2k}\underbrace{\hat{\mathbf{s}}_{t|t-1}^{(n)}\hat{\mathbf{s}}_{t|t-1}^{(n),T}\mathbf{C}^T}_{=\hat{\mathbf{s}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T}} - \hat{\mathbf{s}}_{t|t-1}^{(n)}\hat{\mathbf{y}}_{t|t-1}^{(n),T} \\
&= \frac{1}{2k}\sum_{i=1}^{2k}\sqrt{\mathbf{V}_{t|t-1}^{(n)}}\boldsymbol{\xi}_i\boldsymbol{\xi}_i^T\sqrt{\mathbf{V}_{t|t-1}^{(n)}}^T\mathbf{C}^T + \hat{\mathbf{s}}_{t|t-1}^{(n)}\left(\hat{\mathbf{y}}_{t|t-1}^{(n)}\right)^T - \hat{\mathbf{s}}_{t|t-1}^{(n)}\left(\hat{\mathbf{y}}_{t|t-1}^{(n)}\right)^T \\
&= \mathbf{V}_{t|t-1}^{(n)}\mathbf{C}^T
\end{aligned}
$$

All these equations match exactly the analytic equations we derived before. Thus, the cubature rule is exact if $\mathbf{g}(\cdot)$ is a linear function. □

B. Experiments and Discussion

In this chapter we collect the plots of the Experiments and Discussion chapters that would have cluttered the chapters to much while not adding much information.

B.1. Plots for Running on the CPU or GPU

Plot overview:

- Double pendulum experiment on CPU: Figure B.1
- Double pendulum experiment on GPU: Figure B.1

B.2. Plots for Single- and Multi-Sequence Learning

Plot overview:

- Damped pendulum experiment on a single observation sequence: Figure B.3
- Damped pendulum experiment on a multiple (two) observation sequence: Figure B.4

B.3. Remaining Plots

B.3.1. Pendulum

Plot overview:

- Pendulum, 2-dimensional latent: Figure B.5
- Pendulum, 14-dimensional latent: Figure B.6

B.3.2. Damped Pendulum

Plot overview:

- Damped pendulum, 2-dimensional latent: Figure B.7
- Damped pendulum, 30-dimensional latent: Figure B.8

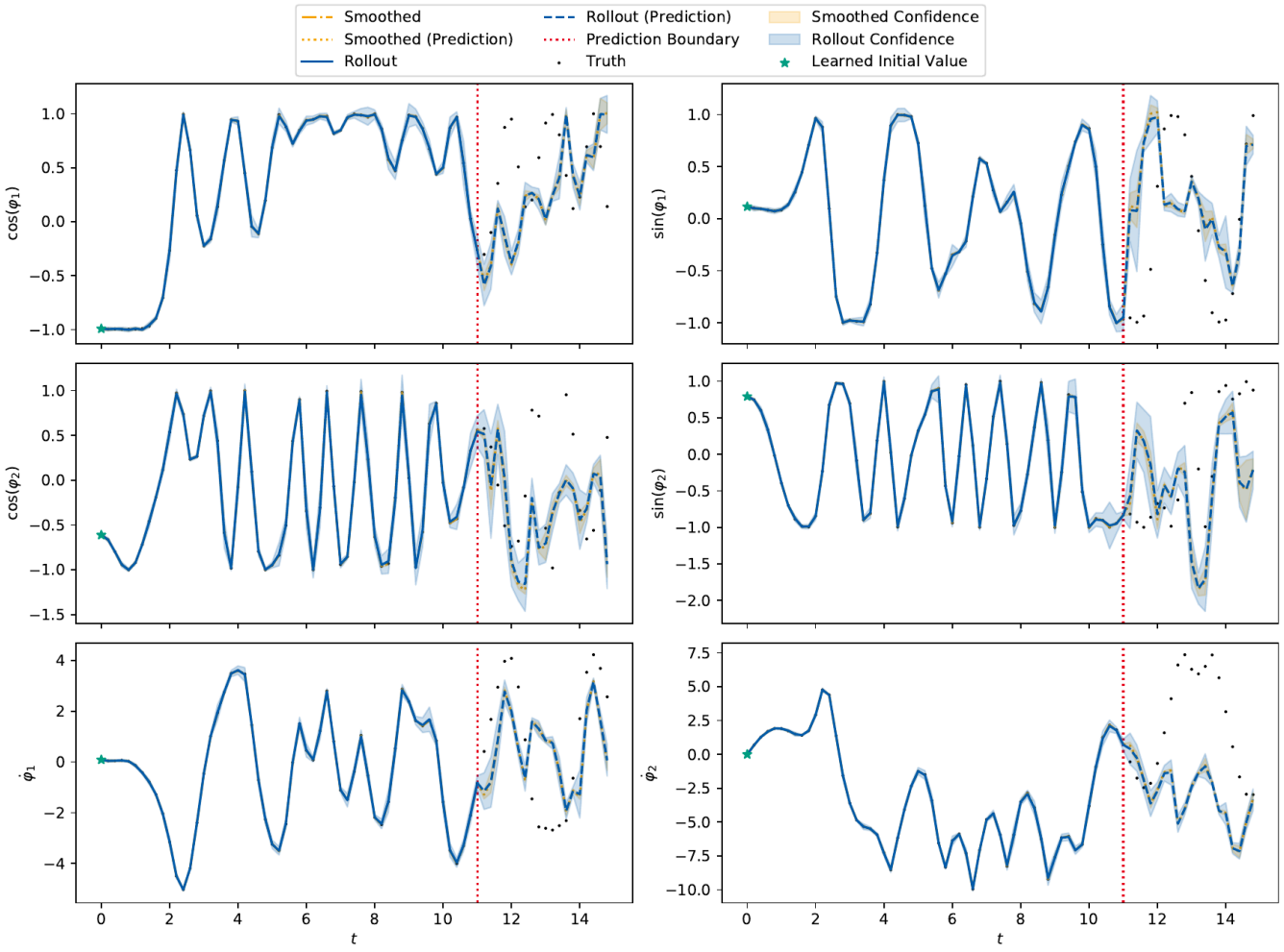


Figure B.1.: Rollout of the double pendulum experiment that ran on the CPU.

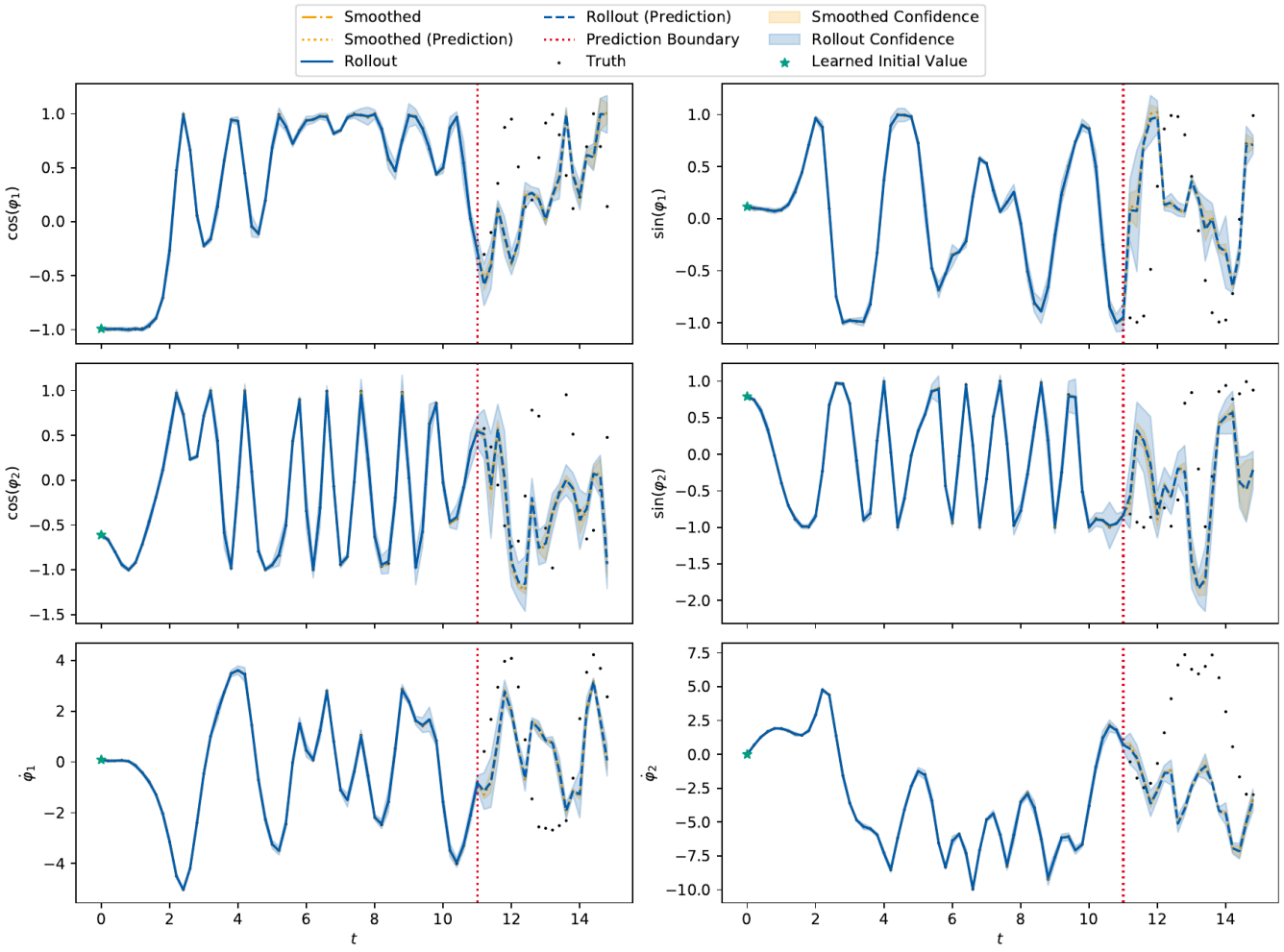


Figure B.2.: Rollout of the double pendulum experiment that ran on the GPU.

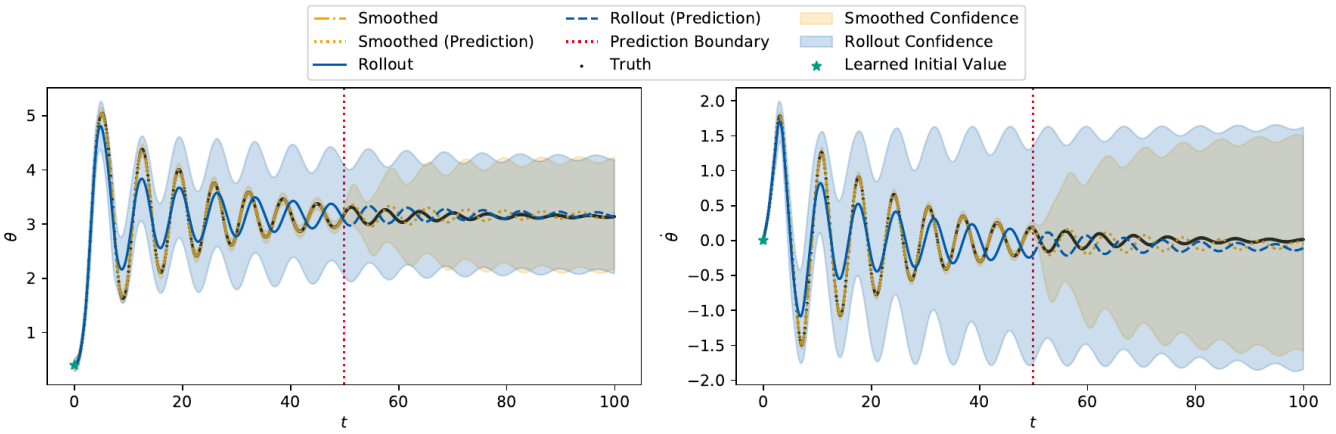


Figure B.3.: Rollout of the damped pendulum experiment learning on a single observation sequence.

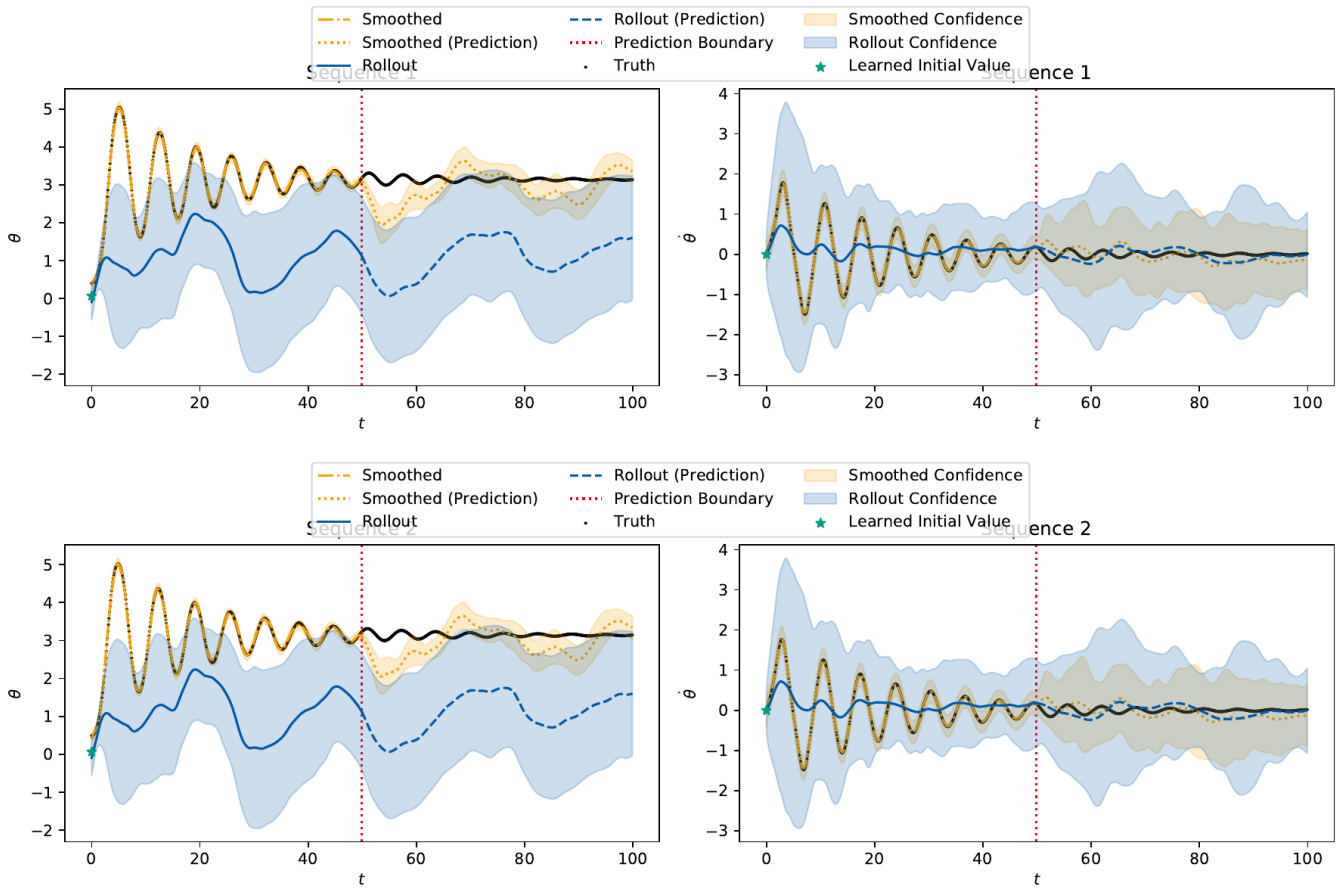


Figure B.4.: Rollout of the damped pendulum experiment learning on multiple (two) observation sequences. The top row is the first observation sequence, the bottom row is the second observation sequence.

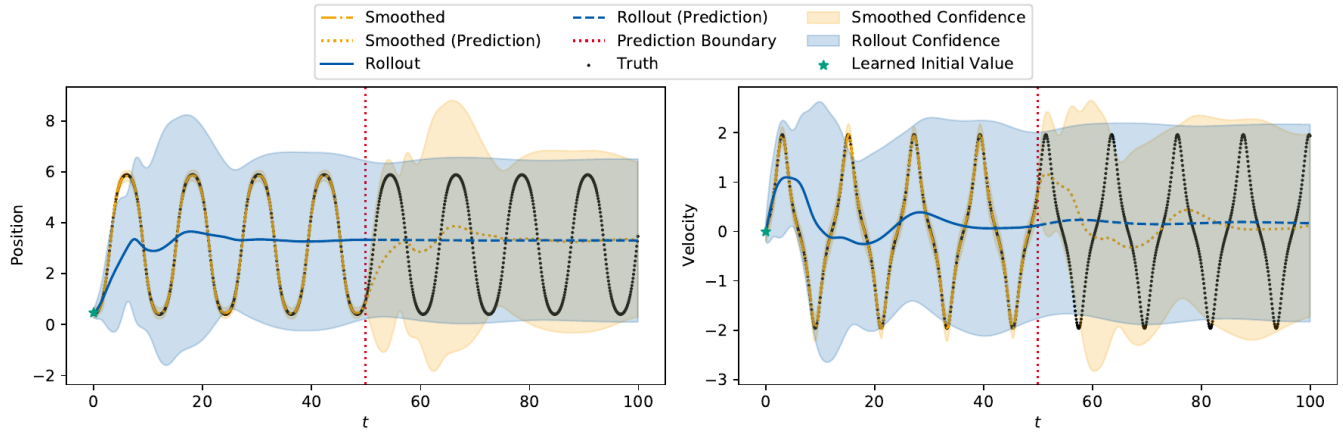


Figure B.5.: The rollout plot in the observation space of the pendulum environment for $k = 2$. The left plot shows the displacement and the right plot the angular velocity. The black dots represent the true data of which the model used everything until the red prediction boundary to train on. The blue line is the rollout, starting from the learned initial value (marked with a green star). The orange dash-dotted line is the smoothed data. The dotted orange line then is the rollout starting from the last smoothed state, forming the "smoothed prediction". The shaded regions show the confidence, i.e. two times the standard deviation.

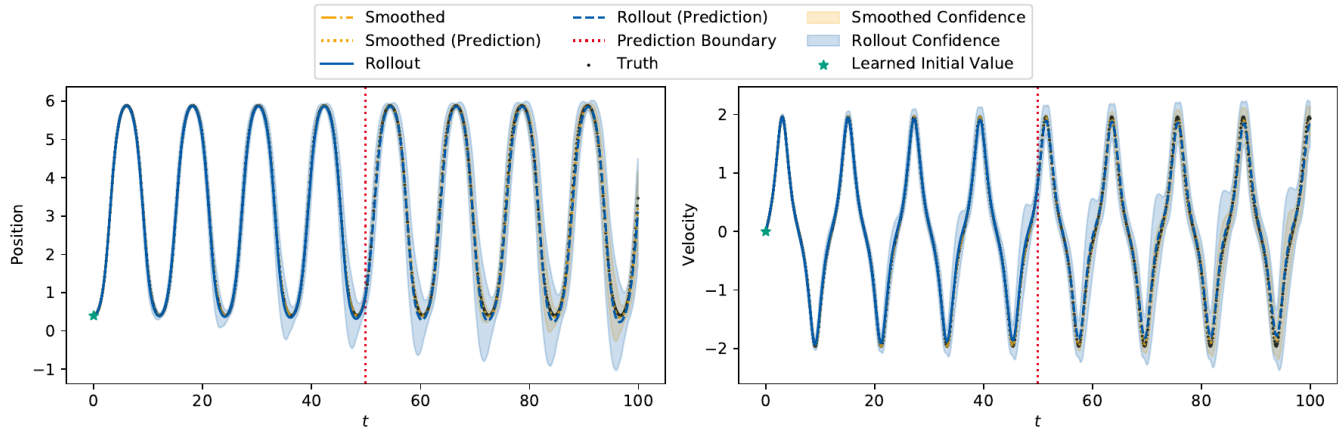


Figure B.6.: The rollout plot in the observation space of the pendulum environment for $k = 14$. The left plot shows the displacement and the right plot the angular velocity. The black dots represent the true data of which the model used everything until the red prediction boundary to train on. The blue line is the rollout, starting from the learned initial value (marked with a green star). The orange dash-dotted line is the smoothed data. The dotted orange line then is the rollout starting from the last smoothed state, forming the "smoothed prediction". The shaded regions show the confidence, i.e. two times the standard deviation.

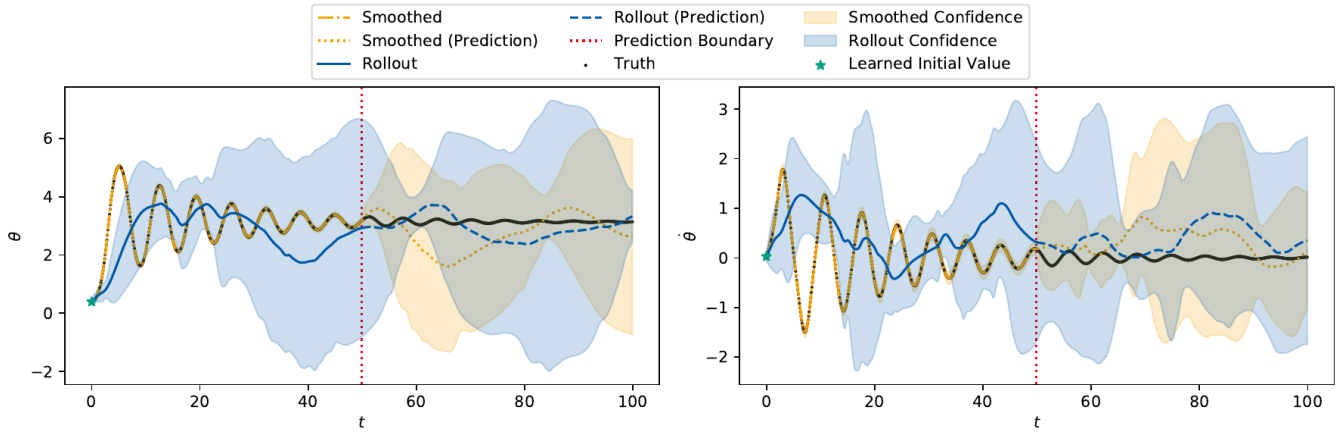


Figure B.7.: The rollout plot in the observation space of the damped pendulum environment for $k = 2$. The left plot shows the displacement and the right plot the angular velocity. The black dots represent the true data of which the model used everything until the red prediction boundary to train on. The blue line is the rollout, starting from the learned initial value (marked with a green star). The orange dash-dotted line is the smoothed data. The dotted orange line then is the rollout starting from the last smoothed state, forming the "smoothed prediction". The shaded regions show the confidence, i.e. two times the standard deviation.

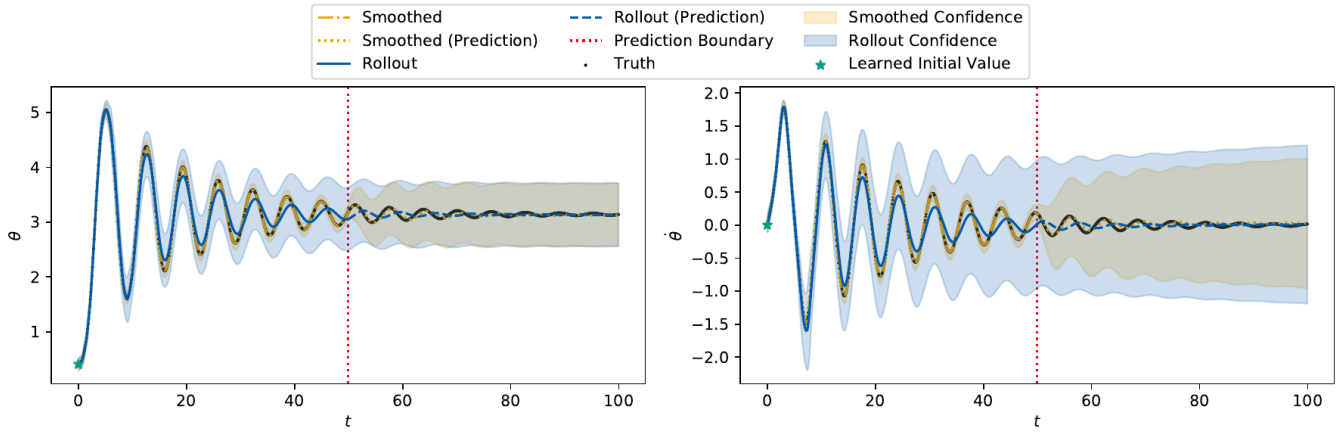


Figure B.8.: The rollout plot in the observation space of the damped pendulum environment for $k = 30$. The left plot shows the displacement and the right plot the angular velocity. The black dots represent the true data of which the model used everything until the red prediction boundary to train on. The blue line is the rollout, starting from the learned initial value (marked with a green star). The orange dash-dotted line is the smoothed data. The dotted orange line then is the rollout starting from the last smoothed state, forming the "smoothed prediction". The shaded regions show the confidence, i.e. two times the standard deviation.

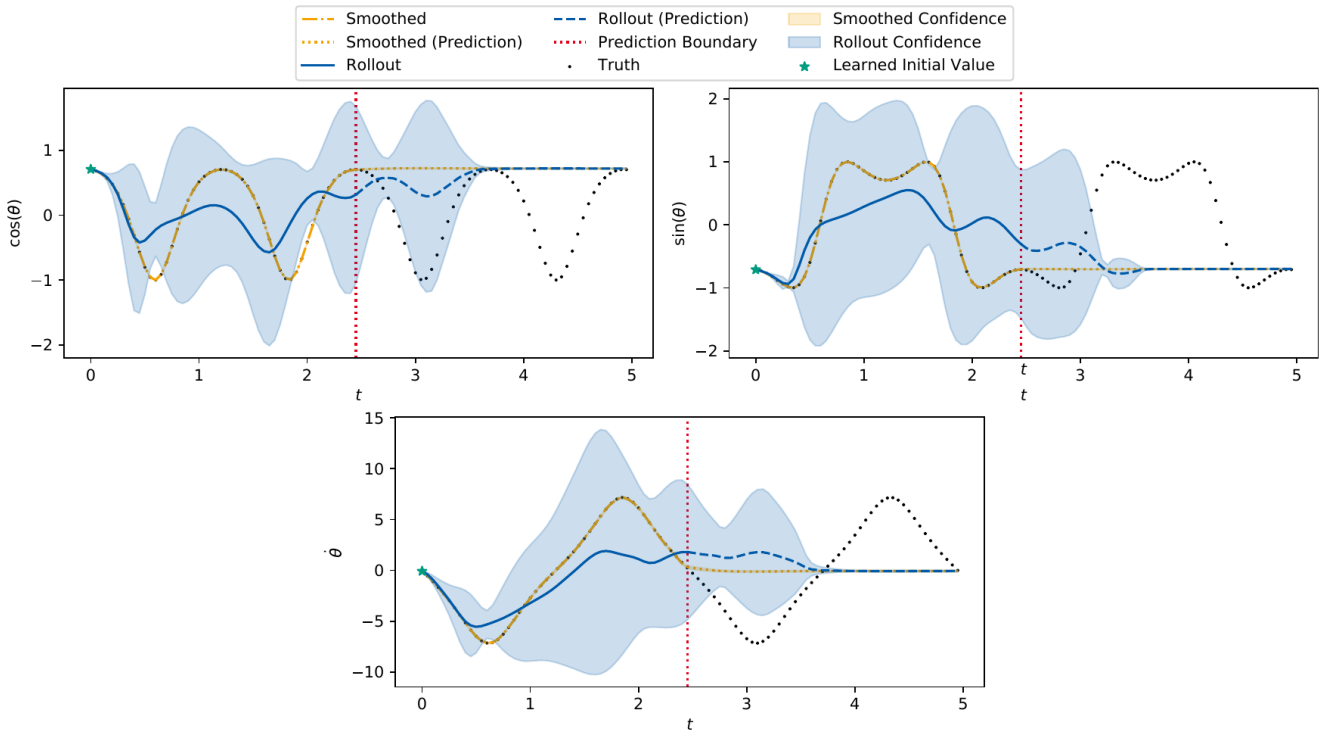


Figure B.9.: The rollout plot in the observation space of the Gym pendulum environment for $k = 2$. The left plot shows the displacement and the right plot the angular velocity. The black dots represent the true data of which the model used everything until the red prediction boundary to train on. The blue line is the rollout, starting from the learned initial value (marked with a green star). The orange dash-dotted line is the smoothed data. The dotted orange line then is the rollout starting from the last smoothed state, forming the "smoothed prediction". The shaded regions show the confidence, i.e. two times the standard deviation.

B.3.3. Gym Pendulum

Plot overview:

- Gym pendulum, 2-dimensional latent: Figure B.9
- Gym pendulum, 7-dimensional latent: Figure B.10

B.3.4. Gym Cartpole

Plot overview:

- Cartpole, 2-dimensional latent rollout without confidence: Figure B.11, with confidence: Figure B.12
- Cartpole, 10-dimensional latent, rollout without confidence: Figure B.13
- Cartpole, 14-dimensional latent rollout with confidence: Figure B.14, without confidence: Figure B.15

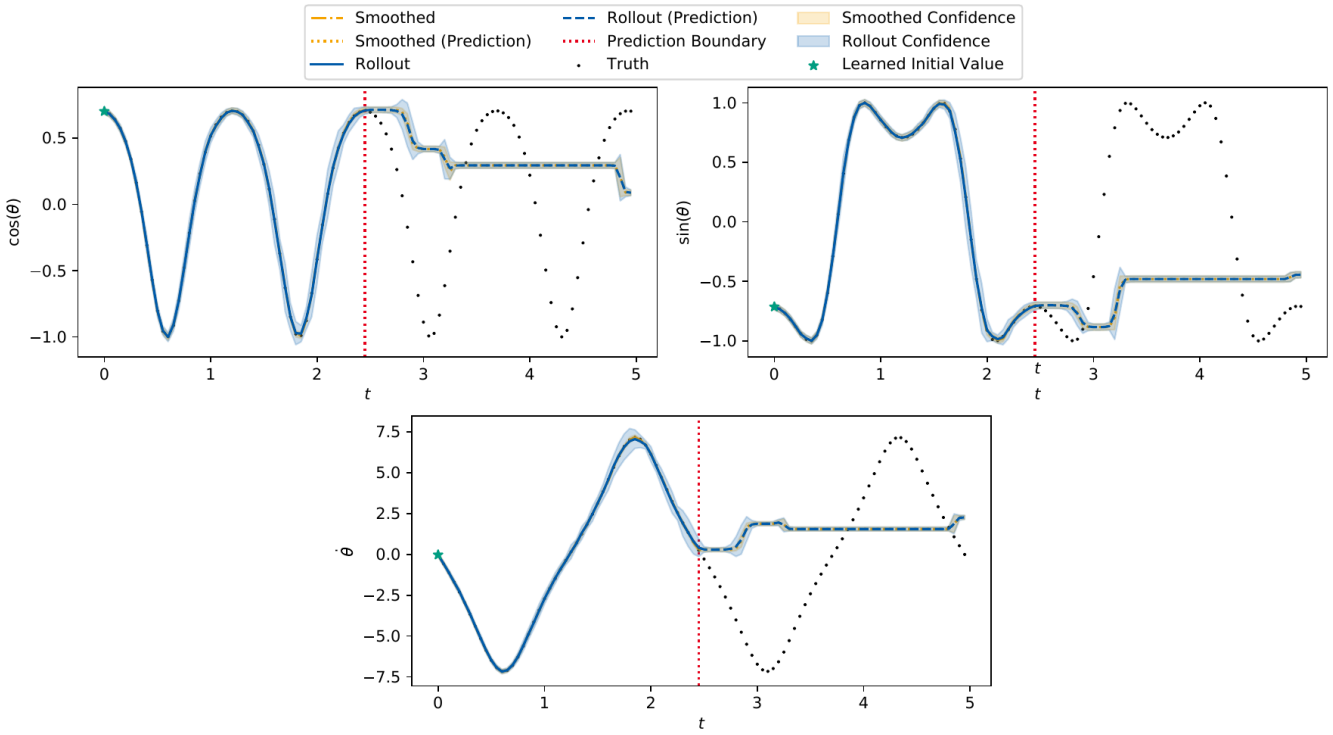


Figure B.10.: The rollout plot in the observation space of the pendulum environment for $k = 7$. The left plot shows the displacement and the right plot the angular velocity. The black dots represent the true data of which the model used everything until the red prediction boundary to train on. The blue line is the rollout, starting from the learned initial value (marked with a green star). The orange dash-dotted line is the smoothed data. The dotted orange line then is the rollout starting from the last smoothed state, forming the “smoothed prediction”. The shaded regions show the confidence, i.e. two times the standard deviation.

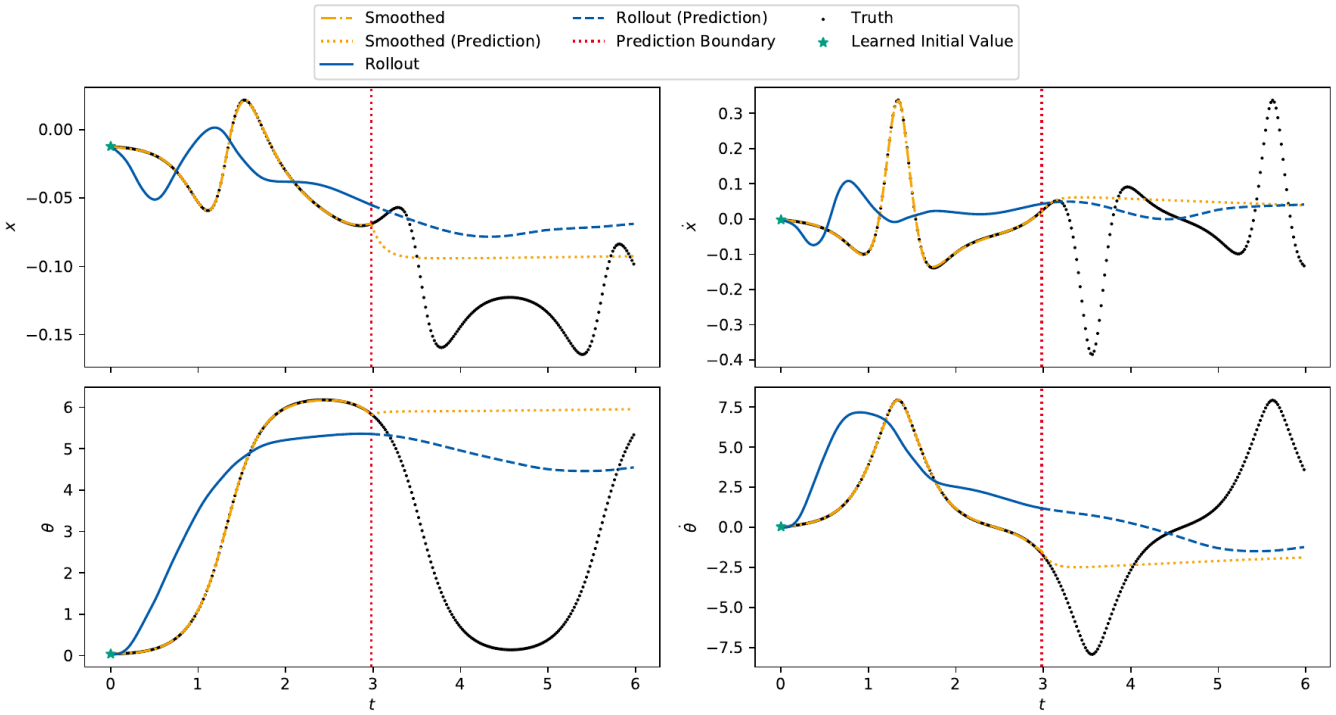


Figure B.11.: The rollout plot in the observation space of the cartpole environment for $k = 2$. The top plot is the cart position (left) and velocity (right), the row the pole displacement (left) and angular velocity (right). The black dots represent the true data of which the model used everything until the red prediction boundary to train on. The blue line is the rollout, starting from the learned initial value (marked with a green star). The orange dash-dotted line is the smoothed data. The dotted orange line then is the rollout starting from the last smoothed state, forming the "smoothed prediction". Note that we have removed the confidence on this plot as it is really low, resulting in high variances. See Figure B.12 in section B.3 for the plot with confidence.

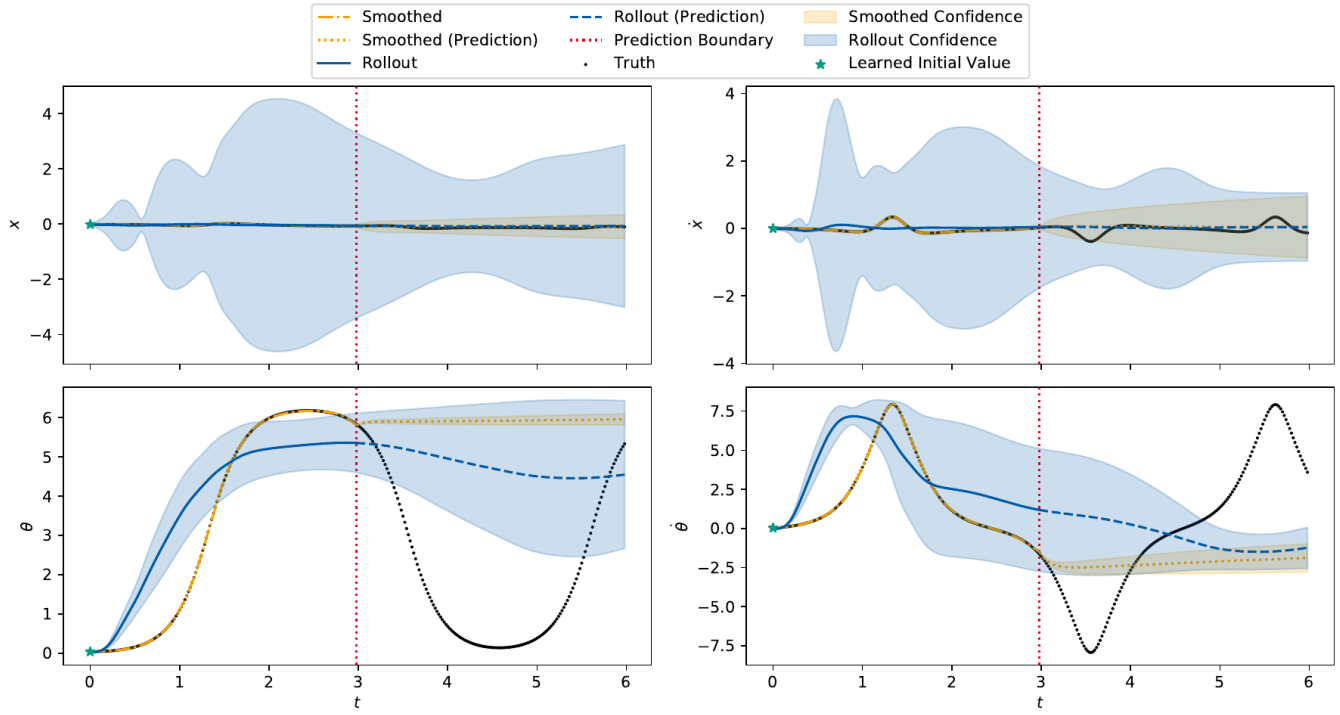


Figure B.12.: This plot shows the same data as in Figure B.11, but with the confidence shown. As the confidence is quite low, this plot is hard to interpret which is the reason why we removed the confidence plot in the first place.

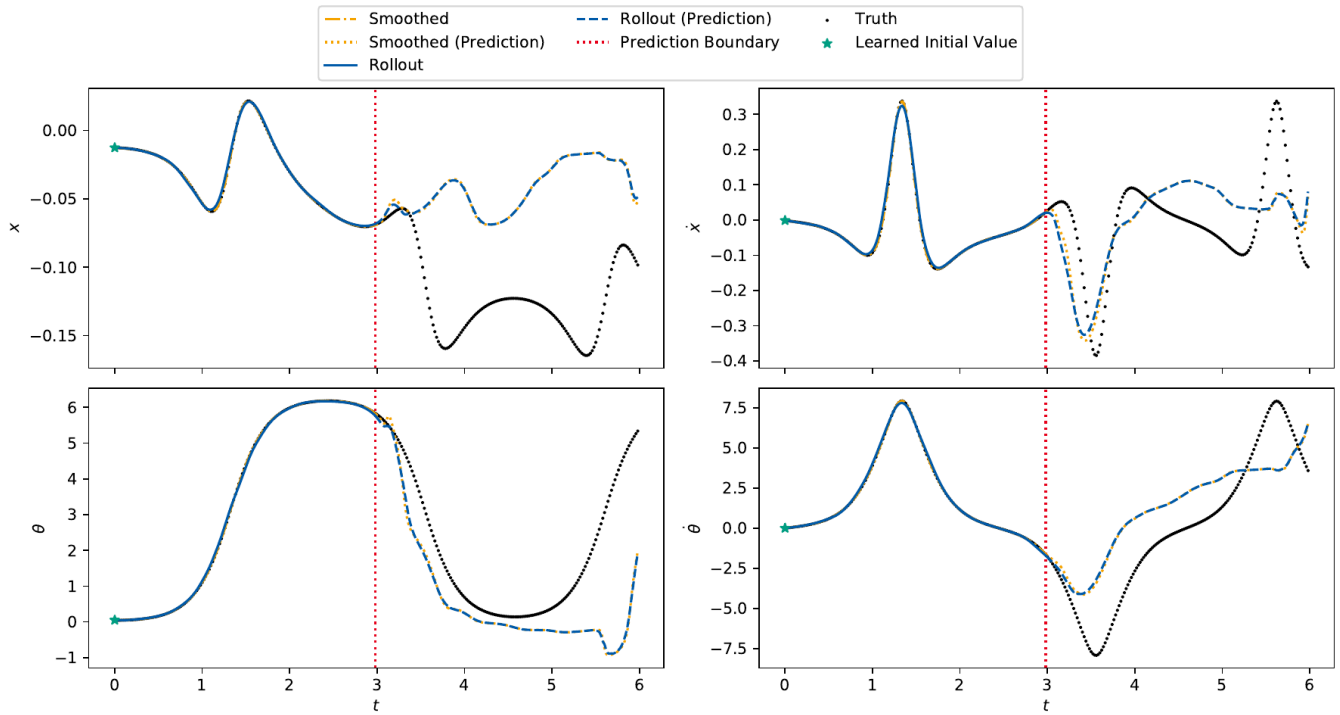


Figure B.13.: This plot shows the same data as in Figure 6.15, but without the confidence shown.

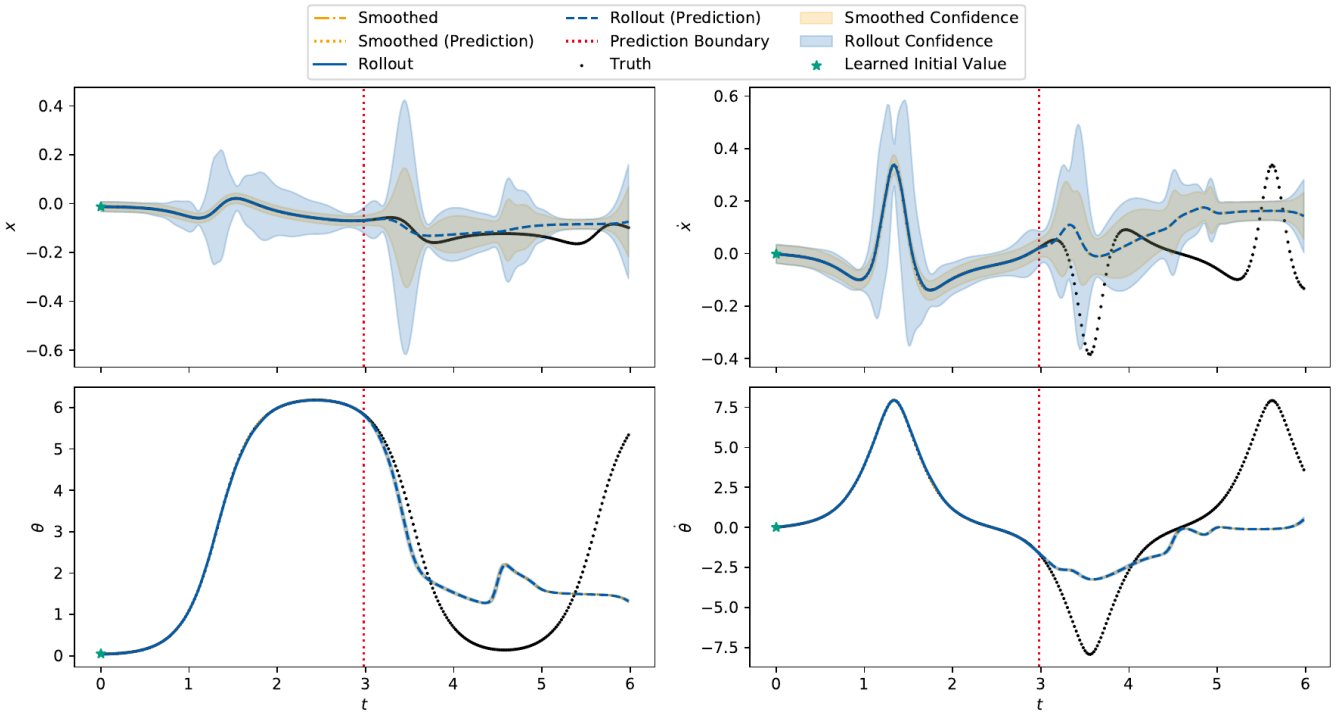


Figure B.14.: The rollout plot in the observation space of the cartpole environment for $k = 14$. The top plot is the cart position (left) and velocity (right), the row the pole displacement (left) and angular velocity (right). The black dots represent the true data of which the model used everything until the red prediction boundary to train on. The blue line is the rollout, starting from the learned initial value (marked with a green star). The orange dash-dotted line is the smoothed data. The dotted orange line then is the rollout starting from the last smoothed state, forming the "smoothed prediction". The shaded regions show the confidence, i.e. two times the standard deviation. As the variances in the top plots are still very high, see Figure B.15 in section B.3 for the plot without the confidence.

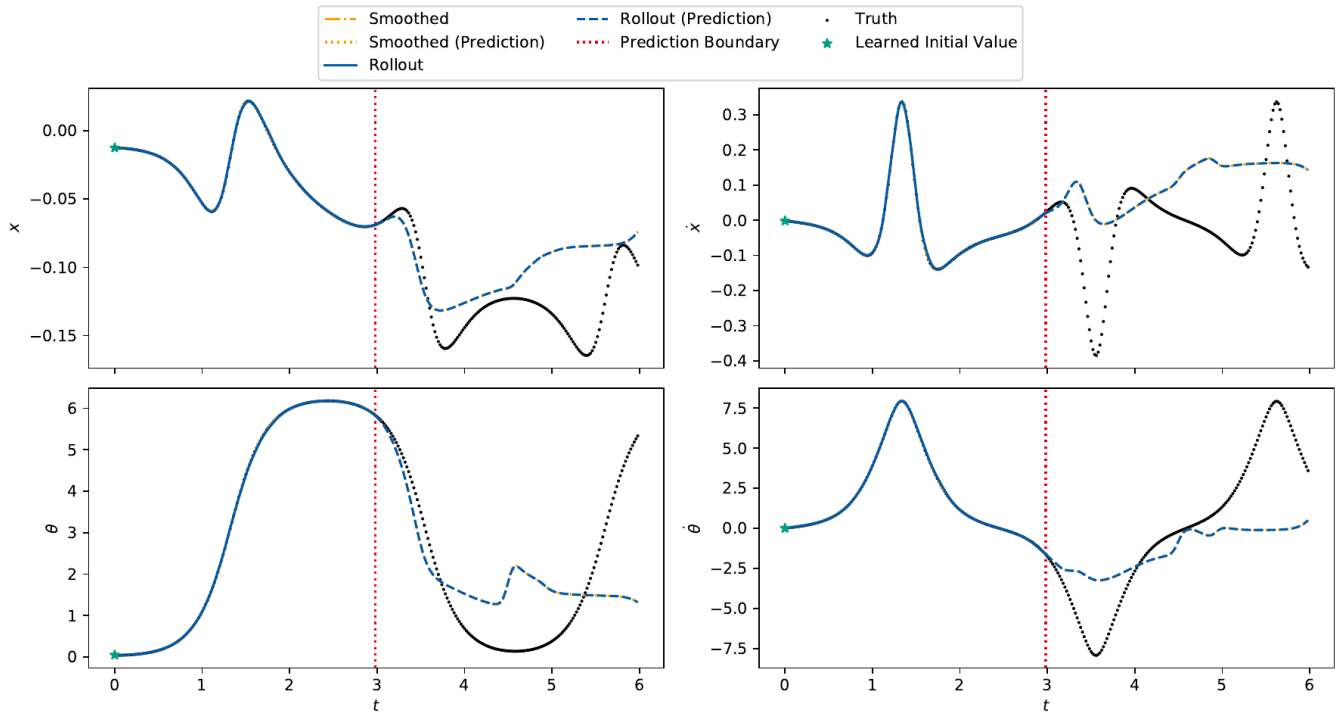


Figure B.15.: This plot shows the same data as in Figure B.14, but without the confidence shown.

B.3.5. Gym Double Pendulum

Plot overview:

- Double pendulum, 3-dimensional latent: Figure B.16
- Double pendulum, 30-dimensional latent: Figure B.17

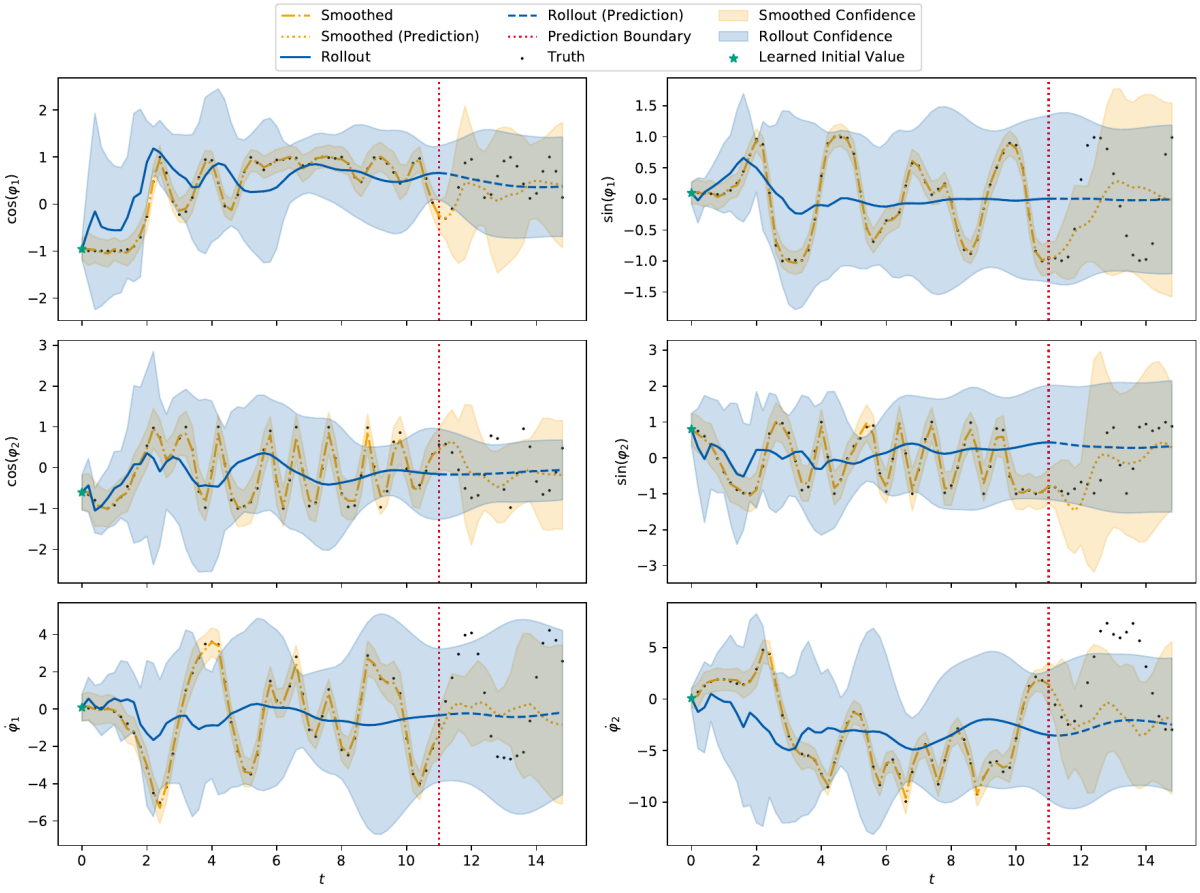


Figure B.16.: The rollout plot in the observation space of the double pendulum environment for $k = 3$. The top row shows the cosine/sine of the displacement of the inner pendulum, the middle row shows the cosine/sine of the displacement of the outer pendulum and the bottom row shows the angular velocity of the inner and outer pendulum. The black dots represent the true data of which the model used everything until the red prediction boundary to train on. The blue line is the rollout, starting from the learned initial value (marked with a green star). The orange dash-dotted line is the smoothed data. The dotted orange line then is the rollout starting from the last smoothed state, forming the "smoothed prediction". The shaded regions show the confidence, i.e. two times the standard deviation.

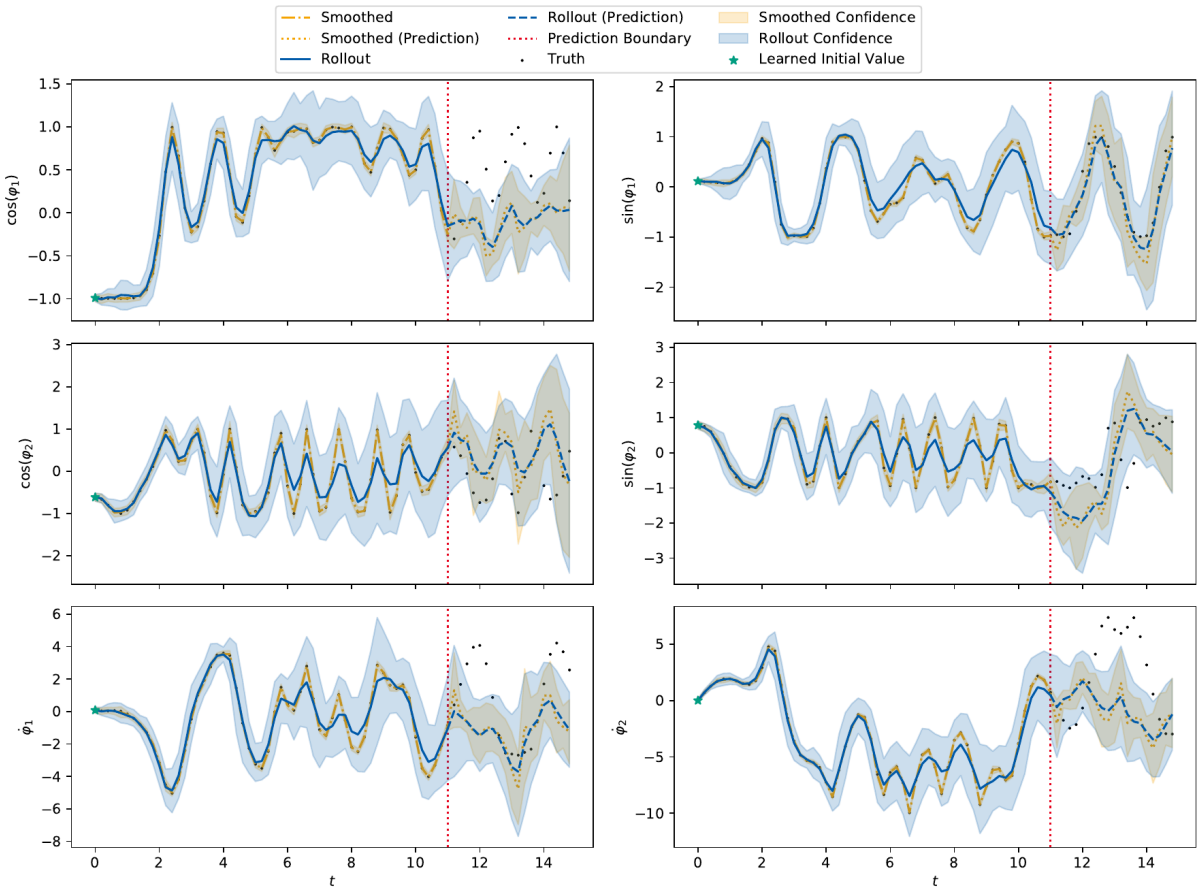


Figure B.17.: The rollout plot in the observation space of the double pendulum environment for $k = 18$. The top row shows the cosine/sine of the displacement of the inner pendulum, the middle row shows the cosine/sine of the displacement of the outer pendulum and the bottom row shows the angular velocity of the inner and outer pendulum. The black dots represent the true data of which the model used everything until the red prediction boundary to train on. The blue line is the rollout, starting from the learned initial value (marked with a green star). The orange dash-dotted line is the smoothed data. The dotted orange line then is the rollout starting from the last smoothed state, forming the "smoothed prediction". The shaded regions show the confidence, i.e. two times the standard deviation.

C. Miscellaneous

Solution of the Harmonic Oscillator

To solve the differential motion equation

$$m\ddot{x} = -kx$$

of the harmonic oscillator given in section 3.1, we use the solution approach

$$x(t) = ce^{\lambda t} \qquad \dot{x}(t) = \lambda ce^{\lambda t} \qquad \ddot{x}(t) = \lambda^2 ce^{\lambda t}$$

and insert it into the differential equation:

$$m\ddot{x} = -kx \quad \Longrightarrow \quad m\lambda^2 ce^{\lambda t} = -kce^{\lambda t} \quad \Longleftrightarrow \quad m\lambda^2 = -k \quad \Longleftrightarrow \quad \lambda = \pm\sqrt{-\frac{k}{m}}$$

As both k and m are defined to be positive, we get the complex solutions:

$$x_1(t) = e^{it\sqrt{k/m}} \qquad x_2(t) = e^{-it\sqrt{k/m}}$$

Due to the superposition principle, also $x_1 + x_2$ and $x_1 - x_2$ are solutions. Hence, we get two real solutions by using Euler's identity $e^{\varphi i} = \cos(\varphi) + i\sin(\varphi)$:

$$\begin{aligned}
x_1 + x_2 &= e^{it\sqrt{k/m}} + e^{-it\sqrt{k/m}} \\
&= \cos\left(t\sqrt{k/m}\right) + i\sin\left(t\sqrt{k/m}\right) + \cos\left(t\sqrt{k/m}\right) - i\sin\left(t\sqrt{k/m}\right) \\
&= 2\cos\left(t\sqrt{k/m}\right) \\
x_1 - x_2 &= e^{i\sqrt{k/m}t} - e^{-it\sqrt{k/m}} \\
&= \cos\left(t\sqrt{k/m}\right) i\sin\left(t\sqrt{k/m}\right) - \cos\left(t\sqrt{k/m}\right) + i\sin\left(t\sqrt{k/m}\right) \\
&= 2i\sin\left(t\sqrt{k/m}\right)
\end{aligned}$$

This yields the following general solution:

$$x(t) = c_1\cos\left(t\sqrt{k/m}\right) + c_2\sin\left(t\sqrt{k/m}\right), \quad c_1, c_2 \in \mathbb{C}$$

As both Sine and Cosine are Sinusoidal, different $c_1 \neq c_2$ only lead to a phase shift. Thus we can also write the solution as

$$x(t) = A\cos\left(t\sqrt{k/m} + \varphi\right)$$

with the amplitude A and the phase φ.

Overview over Notations for Linear Gaussian Dynamical Systems

Element / Definition	**Ghahramani** [GH96]	**Minka** [Min99]	**This Thesis**
Time	t	t	t
Last Timestep	T	T	T
State	$\mathbf{x}_t$	$\mathbf{s}_t$	$\mathbf{s}_t$
Observations	$\mathbf{y}_t$	$\mathbf{x}_t$	$\mathbf{y}_t$
State Dynamics Matrix	$\mathbf{A}$	$\mathbf{A}$	$\mathbf{A}$
State Noise Covariance	$\mathbf{Q}$	$\mathbf{\Gamma}$	$\mathbf{Q}$
Observation Matrix	$\mathbf{C}$	$\mathbf{C}$	$\mathbf{C}$
Observation Noise Covariance	$\mathbf{R}$	$\mathbf{\Sigma}$	$\mathbf{R}$
Full Sequence $\left(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T\right)$	$\{\mathbf{x}\}$	$\mathbf{x}_{1:T}$	$\mathbf{x}_{1:T}$
Subsequence $\left(\mathbf{x}_{t_0}, \mathbf{x}_{t_0+1}, \cdots, \mathbf{x}_{t_1}\right)$	$\{\mathbf{x}\}_{t_0}^{t_1}$	$\mathbf{x}_{t_0:t_1}$	$\mathbf{x}_{t_0:t_1}$
Expected Log Likelihood	Q	Q	Q
Predicted State	$\mathbf{x}_t^{t-1}$	–	$\hat{\mathbf{s}}_{t\|t-1}$
Predicted Covariance	$\mathbf{V}_t^{t-1}$	$\mathbf{P}_{t-1}$	$\hat{\mathbf{V}}_{t\|t-1}$
Filtered State	$\mathbf{x}_t^t$	$\mathbf{m}_t$	$\hat{\mathbf{s}}_{t\|t}$
Filtered Covariance	$\mathbf{V}_t^t$	$\mathbf{V}_t$	$\hat{\mathbf{V}}_{t\|t}$
Smoothed State	$\hat{\mathbf{x}}_t$	$\hat{\mathbf{m}}_t$	$\hat{\mathbf{s}}_t$, $\hat{\mathbf{s}}_{t\|T}$
Smoothed Covariance	$\mathbf{V}_t^T$	$\hat{\mathbf{V}}_t$	$\hat{\mathbf{V}}_t$, $\hat{\mathbf{V}}_{t\|T}$
Self-Correlation $\mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^T \,\middle\|\, \{\mathbf{y}\}\right]$	$\mathbf{P}_t$	–	$\mathbf{P}_t$
Cross-Correlation $\mathbb{E}\left[\mathbf{x}_t\mathbf{x}_{t-1}^T \,\middle\|\, \{\mathbf{y}\}\right]$	$\mathbf{P}_{t,t-1}$	–	$\mathbf{P}_{t,t-1}$

Table C.1.: Notations used for linear Gaussian dynamical systems in different papers.