UNIVERSITY OF CALIFORNIA,
IRVINE

Efficient Reinforcement Learning with Bayesian Optimization

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Mechanical and Aerospace Engineering

by

Danyan Ganjali

Dissertation Committee:
Professor Athanasios Sideris, Chair
Professor Kenneth D. Mease
Professor Jeffrey L. Krichmar

2016

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my mother Zahra and my father John, who were my first teachers, and also the other members of my family. I would also like to thank Ottavia Golfetto for her support and encouragement especially during the last few months of my studies; I am incredibily grateful for having her. I will be forever thankful to my adviser, Prof. Sideris, who has guided me through the years of my studies at UCI, and has dedicated his time into helping me learn the material included in this dissertation.

CURRICULUM VITAE

Danyan Ganjali

EDUCATION

Doctor of Philosophy in Mechanical Engineering **2016**
University of California, Irvine *Irvine, California*

Master of Science in Electrical and Computer Engineering **2009**
Clemson University *Clemson, SC*

Master of Science in Electrical Engineering **2005**
New Jersey Institute of Technology *Newark, NJ*

Bachelor of Science in Electrical Engineering **2003**
New Jersey Institute of Technology *Newark, NJ*

RESEARCH EXPERIENCE

Graduate Research Assistant **2011–2016**
University of California, Irvine *Irvine, California*

Graduate Research Assistant **2007–2009**
Clemson University *Clemson, SC*

TEACHING EXPERIENCE

Teaching Assistant **2012–2015**
University of California, Irvine *Irvine, CA*

ABSTRACT OF THE DISSERTATION

Efficient Reinforcement Learning with Bayesian Optimization

By

Danyan Ganjali

Doctor of Philosophy in Mechanical and Aerospace Engineering

University of California, Irvine, 2016

Professor Athanasios Sideris, Chair

A probabilistic reinforcement learning algorithm is presented for finding control policies in continuous state and action spaces without a prior knowledge of the dynamics. The objective of this algorithm is to learn from minimal amount of interaction with the environment in order to maximize a notion of *reward*, i.e. a numerical measure of the quality of the resulting state trajectories. Experience from the interactions are used to construct a set of probabilistic *Gaussian process* (GP) models that predict the resulting state trajectories and the reward from executing a policy on the system. These predictions are used with a technique known as *Bayesian optimization* to search for policies that promise higher rewards. As more experience is gathered, predictions are made with more confidence and the search for better policies relies less on new interactions with the environment.

The computational demand of a GP makes it eventually impractical to use as the number of observations from interacting with the environment increase. Moreover, using a single GP to model different regions that may exhibit disparate behaviors can produce unsatisfactory representations and predictions. One way of mitigating these issues is by partitioning the observation points into different regions each represented by a local GP. With the sequential arrival of the observation points from new experiences, it is necessary to have an adaptive clustering method that can partition the data into an appropriate number of regions. This led to the development of EM^+ algorithm presented in the second part of this work, which is an extension to the *Expectation Maximization* (EM) for the Gaussian mixture models, that assumes no prior knowledge of the number of components.

Lastly, an application of the EM^+ algorithm to filtering problems is presented. We propose a filtering algorithm that combines the advantages of the well-known particle filter and the mixture of Gaussian filter, while avoiding their issues.

Chapter 1

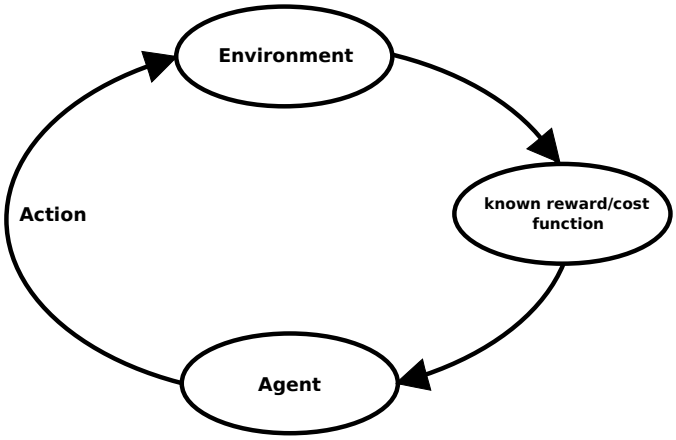
Introduction

The task of iteratively making decisions over an extended period of time to achieve a certain goal appears in a wide variety of fields and applications. Dynamic Programming (DP) and Reinforcement Learning (RL) are families of algorithms that are used to solve such problems. DP requires a pre-defined *model* (a mathematical representation that can predict the behavior of the environment) of the system, whereas RL does not. Even though RL does not require this existing knowledge of the model, it can still construct an approximation of it based on the experience that it has gained from interacting with the environment. The branch of RL where this approximated model is not required is called Direct RL, whereas Model-based RL constructs the explicit model of the environment that can be used in planning [1, 2].

Whether a biological or an artificial agent, the task of learning is achieved by interaction with the environment. RL is learning from these interactions and planning in order to achieve a certain goal. The common approach is to define a notion of *reward* (or cost), a numerical performance index assigned to each state trajectory of the system. The problem then becomes the optimization of this reward function over an extended period of time. The goal of all the algorithms in DP and RL is essentially the same, to find a *policy*, a mapping from the states to actions, that maximizes the future rewards [1, 2]. Figure 1.1 depicts the interaction between the agent and the environment and the rewards received, which is then used in future decision making.

In RL, two definitions of efficiency are considered, *data-efficiency* and *computational-efficiency*. The first is concerned with how much data from the environment is used for learning; in other words, data-efficiency requires minimal interaction with the environment. On the other hand, computational-efficiency deals with how much computation is required to achieve learning. Both Direct and Model-based RL have advantages

Figure 1.1: Learning and decision making by interacting with the environment.



and disadvantages but in general Direct RL is more computationally-efficient, while Model-based approaches are more data-efficient, find better trajectories and policies, and handle changing goals more effectively [3]. Figure 1.2 illustrates how these different approaches can be integrated together to get the best of both. The interaction with the environment from executing a policy provides experience for the learning agent in the form of the trajectories visited and the rewards obtained. This experience can be used for Direct RL, or for learning/updating the model of the environment, which in turn can be used to generate simulated experiences that aid the Direct RL. Since the learned model is based on limited interactions with the system, it performs better in the areas of the environment with more experience and worse in others; in other words there is uncertainty in the model, and the learning algorithm should be capable of making decisions based on this uncertain knowledge about the environment[1].

1.1 Defining The Problem

Given a problem with no prior knowledge of the dynamics, the objective is to find a policy that achieves a pre-defined goal. The discrete-time control problem considered has continuous observable states and corresponding actions of the form

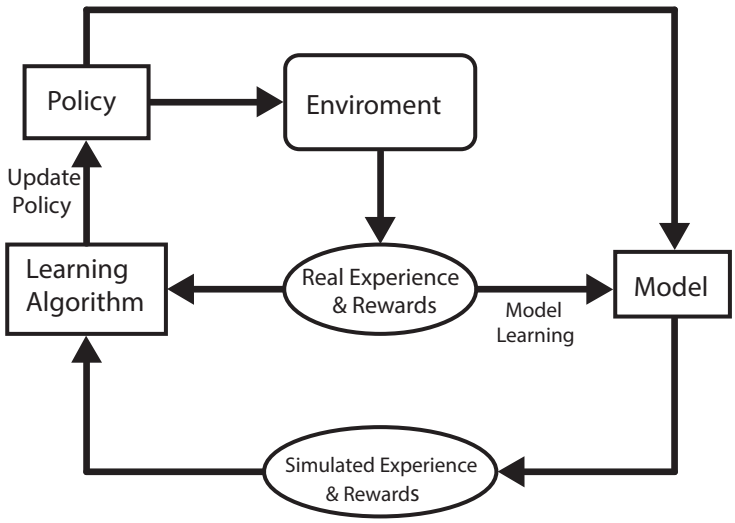
$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{1.1}$$

where the state is $\boldsymbol{x} \in \mathbb{R}^E$ and $\boldsymbol{u} \in \mathbb{R}^U$ is the action, and the unknown deterministic[2] function $f(.)$ maps

[1] this is similiar to the *Dyna* structure introduced in [1, 4], with the difference that the experiences from simulation will not be treated as if they had come from the real-world

[2] taking an action $\boldsymbol{u}_t$ in a state $\boldsymbol{x}_t$ always results in the same state $\boldsymbol{x}_{t+1}$

Figure 1.2: Combining Direct and Model-based RL. This is similar to the Dyna structure in [1, 4].



the state and the action to the state at the next time step. The policy, which is the mapping of the states to actions is represented by a non-linear function as,

$$\boldsymbol{u}_t = h(\boldsymbol{x}_t, \boldsymbol{\theta}) \tag{1.2}$$

where $\boldsymbol{\theta} \in \mathbb{R}^m$ is a vector of parameters specifying the policy.

The total reward for a finite-horizon time of T steps is defined as

$$\boldsymbol{V}_h(\boldsymbol{x}_0) = \sum_{t=0}^{T} r(\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{1.3}$$

where $r(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is a known immediate reward function. Since the initial state $\boldsymbol{x}_0$ is fixed and the dynamics are deterministic, the total reward for a finite horizon T is a function of the policy parameters $\boldsymbol{\theta}$. The unknown policy function can be represented by a function approximator with $\boldsymbol{\theta}$ as its parameters, and the objective becomes finding a set of parameters such that the total reward is maximized,

$$\begin{aligned} \boldsymbol{\theta}_{\text{optimal}} &= \underset{\boldsymbol{\theta}}{\text{argmax}}\, \boldsymbol{V}(\boldsymbol{\theta}) \\ &= \underset{\boldsymbol{\theta}}{\text{argmax}} \sum_{t=0}^{T} r(\boldsymbol{x}_t, h(\boldsymbol{x}_t, \boldsymbol{\theta})) \end{aligned} \tag{1.4}$$

Finding the policy function that maps the states to actions corresponds to feedback control. A special case is when the policy function in equation(1.2) does not depend on the current state, and is only a function of

time, where the objective is to find a set of actions/controls $\boldsymbol{\theta} = \{u_1, \ldots, u_T\}$ that are applied consecutively at each time step.

1.2 Learning the Dynamical Model

Experience is defined as interacting with the system and gathering data, more specifically the current state and the resulting next state. Trajectories are the chain of states visited, and interactions with the system results in different trajectories and more experience. A function approximator can be used to fit this data to represent the experience via a dynamical model that generalizes the behavior of the system throughout the entire state space. Generally, function approximators divide into two types, parametric and non-parametric. Parametric models are defined by a predetermined family of functions that are based on a finite number of parameters. They can be readily trained but are prone to under-fitting or over-fitting unless special care is taken. Nonparametric approximators however, are less restrictive and don't make any assumption about the nature or numbers of the parameters (not upper bounded) [5]. The drawback of nonparametric approximators is that they can be more computationally expensive to train (depending on the size of the training data); this will be discussed in detail in the coming sections.

The modeling problem is defined as having a set of N training input data as $\boldsymbol{Z} = \{\boldsymbol{z}_1 \ldots \boldsymbol{z}_N\}$, where the input is $\boldsymbol{z}_i \in \mathbb{R}^D$ and the scalar noisy observed output is $\boldsymbol{y} = \{y_1 \ldots y_N\}$ with $y_i \in \mathbb{R}$. The mapping from the input to output is given by a latent function $f(\boldsymbol{z})$ and an independent additive Gaussian noise as

$$y = f(\boldsymbol{z}) + \epsilon \tag{1.5}$$

where $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$. We are interested in making predictions or approximating the value of the function $f(.)$ at a test input $\boldsymbol{z}^*$ based on the observed data.

The dynamical model is expected to perform well only around the trajectories that have already been visited. The prediction of the dynamical model should therefore have higher uncertainty in unknown regions, and lower in the areas where experience has already been gathered, which can be achieved by a probabilistic model that puts a belief on the current model. This belief on the model is updated as new training data becomes available. The Gaussian process, a type of Bayesian non-parametric function approximator, is suited for this purpose. It puts a prior on the possible functions that can represent the current data and which is then updated with the arrival of new data. The content of the next section is from [6].

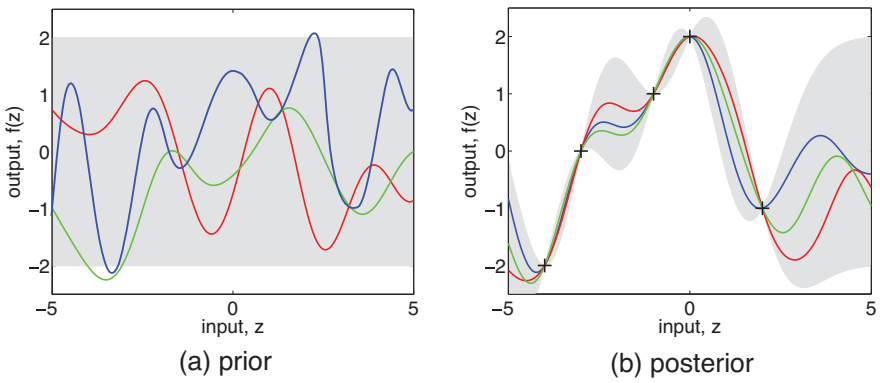


Figure 1.3: Gaussian process [6]. Functions are drawn from the prior in (a) and in (b) the updated posterior of the functions is shown when measurements become available. The gray region refers to uncertainty. Measurements do not have any noise in this example.

1.3 Gaussian Process (GP)

A GP is formally defined as "a collection of random variables, any finite number of which have a joint Gaussian distribution" [6]. A GP is completely defined by its mean and covariance function as

$$f(\boldsymbol{z}), f(\boldsymbol{z}') \sim \mathcal{GP}([m(\boldsymbol{z}), m(\boldsymbol{z}')]^\top, \boldsymbol{k}(\boldsymbol{z}, \boldsymbol{z}')) \tag{1.6}$$

where the mean function $m(.)$, is often chosen to be zero for simplicity. According to the GP assumption, the joint distribution between the outputs of the latent function in equation (1.5) is Gaussian and is given by

$$f(\boldsymbol{z}_1) \dots f(\boldsymbol{z}_N) \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{K}(\boldsymbol{Z}, \boldsymbol{Z})) \tag{1.7}$$

where the symmetric and positive-semi definite matrix $\boldsymbol{K} \in \mathbb{R}^{N \times N}$ is the covariance matrix of the outputs as a function of the N input vectors in $\boldsymbol{Z}$. The (i, j) element of this matrix is given by $K(\boldsymbol{z}_i, \boldsymbol{z}_j)$[3], a covariance function picked beforehand. The covariance function conveys information about the similarities between the data points and the overall behavior of the function.

The Matérn class of covariance functions is defined as

[3]the covariance between the outputs is a function of the inputs

$$K_{\text{Matern}}(z) = \frac{2^{1-\nu}}{\Gamma(\nu)}(\sqrt{2\nu}||z||)^{\nu} K_{\nu}(\sqrt{2\nu}||z||) \tag{1.8}$$

where $\nu>0$ is a parameter that controls the smoothness of the prior, and K_ν is a modified Bessel function [6,7]. For $\nu \to \infty$ and $\nu = \frac{5}{2}$, equation (1.8) results in two common choices of covariance functions known as squared exponential (SE) and Matérn-52 (M52)[4] given as

$$\begin{aligned} K_{SE}(\boldsymbol{z}_p, \boldsymbol{z}_q) &= \alpha^2 \exp\left[-\frac{1}{2} r^2(\boldsymbol{z}_p, \boldsymbol{z}_q)\right] \\ K_{M52}(\boldsymbol{z}_p, \boldsymbol{z}_q) &= \alpha^2 (1 + \sqrt{5r^2(\boldsymbol{z}_p, \boldsymbol{z}_q)} + \frac{5}{3} r^2(\boldsymbol{z}_p, \boldsymbol{z}_q)) \exp\left[-\sqrt{5r^2(\boldsymbol{z}_p, \boldsymbol{z}_q)}\right] \\ r^2(\boldsymbol{z}_p, \boldsymbol{z}_q) &= \sum_{d=1}^{D} (\boldsymbol{z}_p(d) - \boldsymbol{z}_q(d))^2 / l_d^2 \end{aligned} \tag{1.9}$$

where $\boldsymbol{l} = \{l_1 \dots l_D\}$ is a vector of the length-scales for each of the input dimensions, and α^2, the coefficient term added is the signal variance of the latent function f. While the SE and Matérn covariance functions are the most commonly used, there are numerous other functions in the machine learning literature that include non-stationary covariance functions [8–10]. Refer to [6, 11] for an overview on the choice and properties of the various covariance functions in the literature.

As mentioned earlier the measurement noise is independent and additive so the covariance between the noisy observations is given by

$$\text{cov}(\boldsymbol{y}) = \boldsymbol{K}(\boldsymbol{Z}, \boldsymbol{Z}) + \sigma_n^2 \boldsymbol{I} \tag{1.10}$$

Similar to equation (1.7), the joint distribution of the observations and a new prediction is Gaussian and is given by

$$\begin{bmatrix} \boldsymbol{y} \\ f^* \end{bmatrix} \sim \mathcal{N}(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{K}(\boldsymbol{Z}, \boldsymbol{Z}) + \sigma_n^2 \boldsymbol{I} & \boldsymbol{K}(\boldsymbol{Z}, \boldsymbol{z}^*) \\ \boldsymbol{K}(\boldsymbol{z}^*, \boldsymbol{Z}) & K(\boldsymbol{z}^*, \boldsymbol{z}^*) \end{bmatrix}) \tag{1.11}$$

where f^* is the prediction of the latent function at a test input $\boldsymbol{z}^*$. Using the conditional distribution property in section A.1, we can arrive at the predictive equation of

$$f^* | \boldsymbol{Z}, \boldsymbol{y}, \boldsymbol{z}^* \sim \mathcal{N}(\mu(\boldsymbol{z}^*), \sigma^2(\boldsymbol{z}^*)) \tag{1.12}$$

[4] Refer to chapter 4 of [6] for more details

where

$$\begin{aligned}\mu(\boldsymbol{z}^*) &= \boldsymbol{K}(\boldsymbol{z}^*, \boldsymbol{Z})(\boldsymbol{K}(\boldsymbol{Z}, \boldsymbol{Z}) + \sigma_n^2 \boldsymbol{I})^{-1}\boldsymbol{y} \\ \sigma^2(\boldsymbol{z}^*) &= K(\boldsymbol{z}^*, \boldsymbol{z}^*) - \boldsymbol{K}(\boldsymbol{z}^*, \boldsymbol{Z})(\boldsymbol{K}(\boldsymbol{Z}, \boldsymbol{Z}) + \sigma_n^2 \boldsymbol{I})^{-1}\boldsymbol{K}(\boldsymbol{Z}, \boldsymbol{z}^*)\end{aligned} \tag{1.13}$$

The equation above is the predictive distribution of the output given the training data. It can be derived in a similar manner for a set of inputs $\boldsymbol{Z}^* = \{\boldsymbol{z}_1^* \dots \boldsymbol{z}_m^*\}$, giving the predictive equations for a set of m test inputs. In short GP behaves like a function approximator with the difference that instead of returning a deterministic output, it gives a distribution over the output.

1.3.1 Hyperparameters of the GP

The vector of the length-scales $\boldsymbol{l} = \{l_1 \dots l_D\}$, the signal variance α^2, and the noise variance σ_n^2 are called the *hyperparameters* of the GP. Changing these values can alter how the GP behaves. Figure 1.4 shows a modeling example where the input and the output are both 1-dimensional. The GP approximates the latent target function (blue) based on the available measurements (black dots). In Figure 1.5, it can be seen that the length-scale (on the input) can change the expected range of the output. Decreasing the length-scale means that the output is expected to change within a smaller scale of the input, hence resulting in larger uncertainties away from the measurements. Figure 1.6 shows the results of changing α^2; although it affects the overall uncertainty of the GP, it can be seen from equation (5.3) that α^2 is the variance far away from the measurements. Increasing the measurement noise, σ_n^2, results in less certainty about the measurements and throughout the entire space (Figure 1.7).

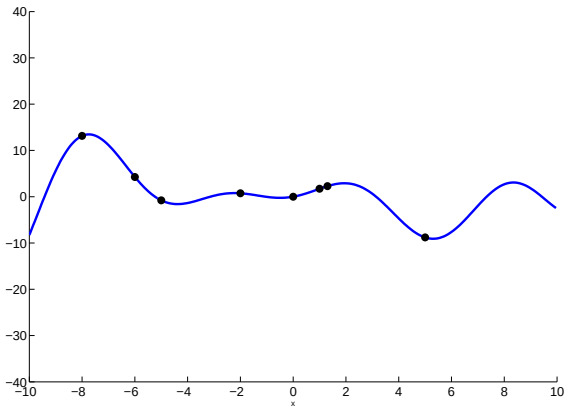


Figure 1.4: measurements (black dots) taken from a latent function (blue line).

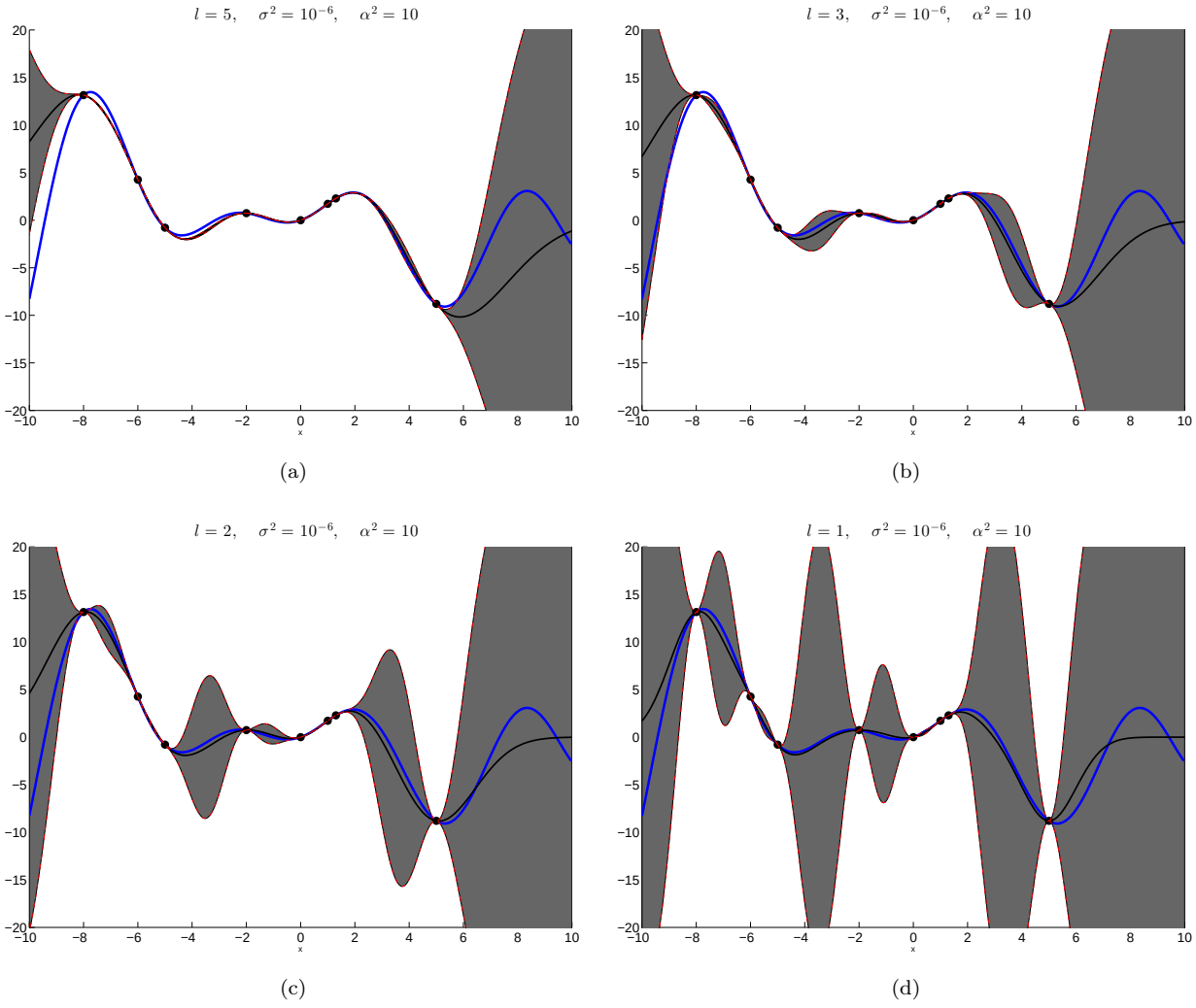


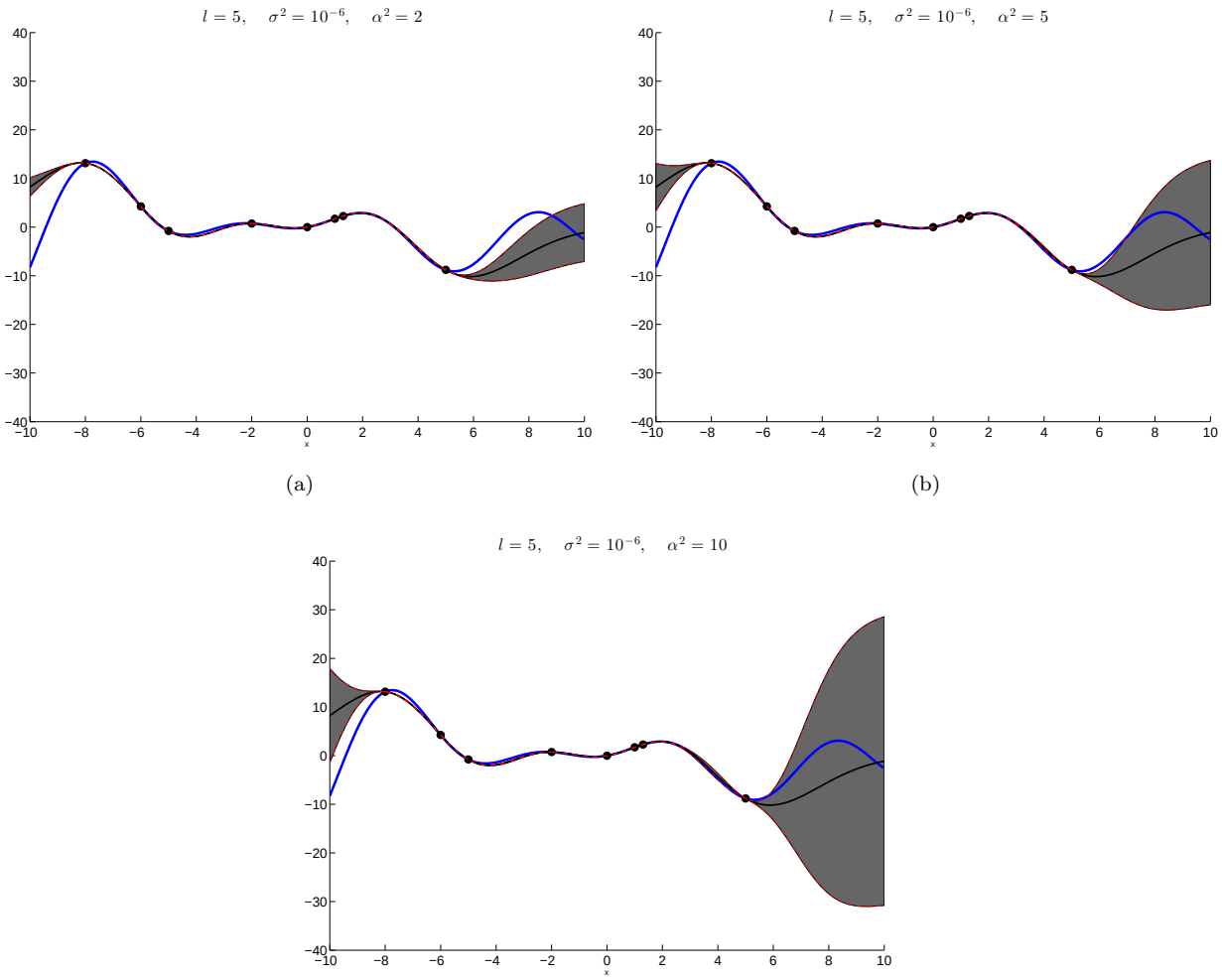
Figure 1.5: Effect of changing the length-scales in a GP. The blue line is the latent function. The black line is the predicted mean, and the grey region is the 99.7% confidence region.

Generally there are two common approaches in dealing with the hyperparameters of GPs. The first is the purely Bayesian approach where priors are placed on the hyperparameters and are then integrated out, and the second is the empirical Bayes approach where the hyperparameters are found by maximizing the marginal likelihood [12]. Here the latter approach is discussed.

The marginal likelihood or evidence of the GP is given by

$$p(\boldsymbol{y}|\boldsymbol{Z}) = \int p(\boldsymbol{y}|\boldsymbol{f}, \boldsymbol{Z}) p(\boldsymbol{f}|\boldsymbol{Z}) d\boldsymbol{f} \tag{1.14}$$

where $p(\boldsymbol{y}|\boldsymbol{f}, \boldsymbol{Z}) = \mathcal{N}(\boldsymbol{f}, \sigma_n^2 \boldsymbol{I})$ is the likelihood and $p(\boldsymbol{f}|\boldsymbol{Z}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{K})$ is the prior. These are both Gaussians,



(a) (b) (c)

Figure 1.6: Effect of changing the hyperparameter α in a GP. The blue line is the latent function. The black line is the predicted mean, and the grey region is the 99.7% confidence region.

and because the product of two Gaussians is also a Gaussian, the integral can be easily computed which gives the following marginal likelihood[5],

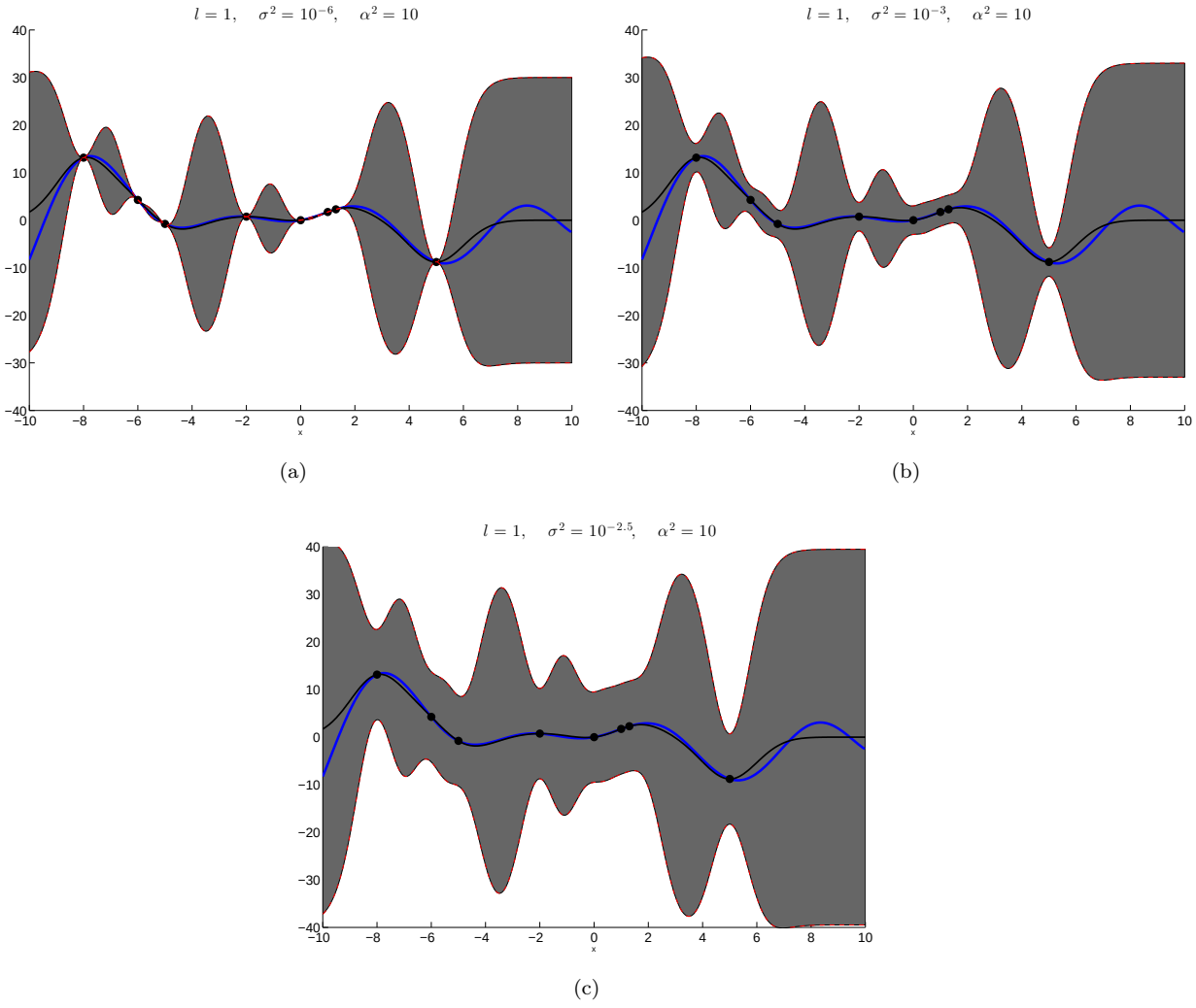
$$p(\boldsymbol{y}|\boldsymbol{Z}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{K} + \sigma_n^2 \boldsymbol{I}) \tag{1.15}$$

The log likelihood is therefore:

$$\log p(\boldsymbol{y}|\boldsymbol{Z}) = -\frac{1}{2}\boldsymbol{y}^\top (\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})^{-1} \boldsymbol{y} - \frac{1}{2}\log|\boldsymbol{K} + \sigma_n^2 \boldsymbol{I}| - \frac{n}{2}\log 2\pi \tag{1.16}$$

The function above is maximized with respect to the hyperparameters, typically by a conjugate gradient-

[5]this can also be observed directly by writing the distribution of $\boldsymbol{y}$ from equation (1.5)



(a) (b)

(c)

Figure 1.7: Effect of changing the hyperparameter σ_n in a GP. The blue line is the latent function. The black line is the predicted mean, and the grey region is the 99.7% confidence region.

based optimization technique that results in finding a set of suitable hyperparameters for the model; this process is called *training* the GP [6, 12]. Figure 1.8 shows the earlier example of a GP with optimized hyperparameters.

1.3.2 Issues with GPs

Training the GP as explained above is dominated by the inversion of the matrix $(\boldsymbol{K}+\sigma_n^2\boldsymbol{I})$, which is required to be performed at every iteration of the optimization algorithm. This means $\mathcal{O}(N^3)$ operations per iteration is required for training, and $\mathcal{O}(DN^2)$ for prediction in equation (5.3) (N is the number of training pairs and D is the dimension of the training input). This makes the GP impractical as the size of the data becomes larger. The simplest way to deal with this is to take a subset of the data or to forget the old trajectories,

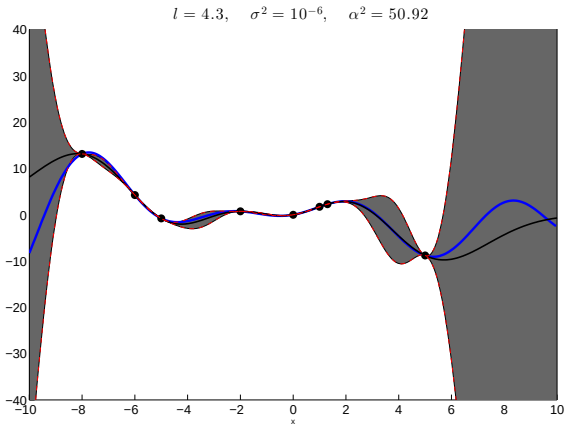


Figure 1.8: Results of optimizing the hyperparameters of the GP used in the example given. The blue line is the latent function. The black line is the predicted mean, and the grey region is the 99.7% confidence region.

but by doing this, one might be discarding interesting regions of the state space. There is an extensive part of the GP literature that deals with this problem and is presented under a unifying view of "sparse" GPs in [13]. The main idea of these approaches is sparse approximation, which is to approximate the $\boldsymbol{K}$ matrix by some lower rank $M \times M$ matrix where $M << N$. The optimization step then typically requires $\mathcal{O}(NM^2)$ operations instead of $\mathcal{O}(N^3)$. This can be achieved by using M points that can be found simply by taking a subset of the data or as suggested in [14], by using "pseudo-inputs", a collection of M virtual induced points that can be found along with the hyperparameters by optimizing a modified likelihood function. These methods consider a global representation of the model, where all the training data is used for prediction.

Although the global sparse representations seem promising for modeling dynamical systems, the issue still remains that the data is growing as more experience is gained. The number and locations of the pseudo-inputs may also need to be changed to better represent the new incoming data. An alternative approach would be to consider local representations. [15] suggests using a set of local GPs assigned to each regions of the data partitioned by a clustering algorithm. The number of the members of the clusters are kept constant, and the mean of a prediction at a test input is found by averaging the output of the nearby GPs. This local representation can be less computationally expensive since the local GPs are smaller in size and are only re-trained when new members are added. The prediction step also requires less computations since only the neighboring GPs are used to predict the output of a test input.

In the application considered here, new training points belonging to different regions of the space are discovered incrementally, which means that the clustering algorithm that partitions the training data should

be capable of adding new clusters as new regions are discovered. This adaptive process is analogous to a common issue in clustering problems when the number of clusters are unknown, which by itself is a topic of extensive research in the literature. The existing approaches and algorithms that attempt to solve this problem have their shortcomings. In Chapter 4 we present EM$^+$, a novel algorithm that can be viewed as an extension to the well-known Expectation Maximization(EM) approach for clustering, that automatically infers the number of clusters.

1.4 Bayesian Optimization

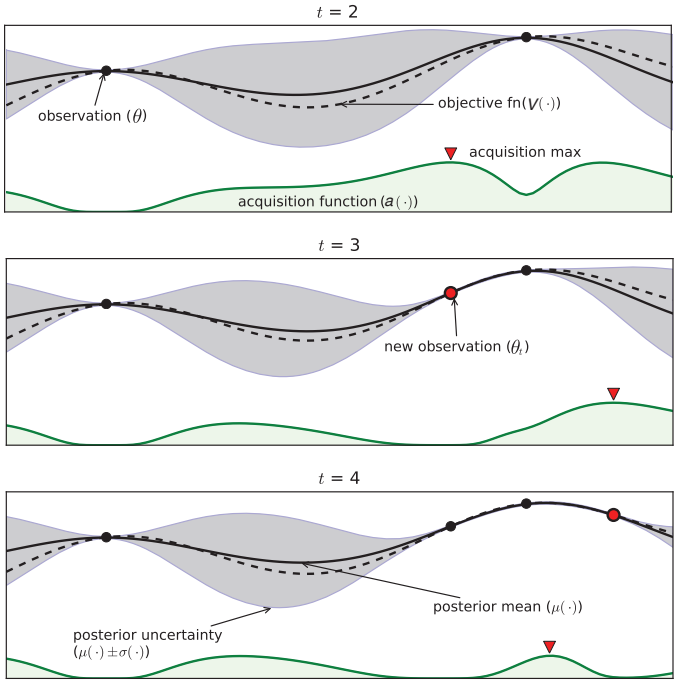


Figure 1.9: Bayesian optimization [16]. Three iterations of a 1-D Bayesian optimization is shown here to find the maximum of the objective function $V(\theta)$. The maximum of the acqusition function that is generated based on the mean and uncertainty of the model of the objective function, tell the algorithm where to sample next.

Consider the problem defined in equation (1.4), where one would like to optimize an objective function. The closed-form expression of this function is not available, and only observations can be obtained that are expensive to evaluate.[6] Given such a problem, a class of global, gradient-free approaches called Bayesian

[6]expense in the control problem defined earlier refers to the amount of interaction with the real-world which is desired to be

optimization can be utilized [17–21]. Figure 1.9 illustrates three iterations of a "toy" Bayesian optimization problem; a probabilistic model(i.e. a GP) is put on the observed data and the expected value and the uncertainty of the objective function at unobserved locations are approximated. Then an *acquisition* function based on these values is used to decide where to sample next. A number of acquisition functions that can be employed will be discussed in the coming sections, but in general they have higher values in promising regions where improvement over the current highest objective value is more likely. The value of the acquisition function at a location is evaluated using the mean and uncertainty outputted by the probabilistic model rather than the objective function and is thus cheaper to evaluate. This conversion to a cheaper function to optimize, is essentially what the Bayesian optimization aims for. At every iteration the optimization of an expensive objective function is converted to a cheap optimization of an acquisition function that can be solved using a global optimization algorithm. The Bayesian optimization approach works well in low-dimensions, its performance however starts to degrade when the number of parameters to be optimized is above 15-20 [22].

The objective function to be optimized in this work is the total reward, V described in equation (1.3), which is a function of the policy parameters, $\boldsymbol{\theta}$. A GP is used to construct a probabilistic model between the set of already-executed policy parameters $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1 \dots \boldsymbol{\theta}_n\}$ and their corresponding obtained total rewards $\boldsymbol{V} = \{V_1 \dots V_n\}$ as

$$V(\boldsymbol{\theta}) \sim \mathcal{GP}(\mathbf{0}, \boldsymbol{K}(\boldsymbol{\Theta}, \boldsymbol{\Theta})) \tag{1.17}$$

with the predictive expectation and the variance of a new policy parameter given by equation (5.3). This GP that maps the policy parameters to rewards will be referred to as the *rewards-GP*. The Bayesian optimization uses this GP along with an acquisition function that guides the algorithm to the next promising policy parameters to execute on the system.

1.4.1 Acquisition Function

The goal of the acquisition function is to guide the optimization process. Its value is high in the regions where there is potentially a better value, and low where it is unlikely to improve the objective function [16]. The location that maximizes the acquisition function is used in the next iteration of the Bayesian optimization algorithm.

The *Probability of Improvement* (PI) [23] over V_{max}, the current maximum of the objective function at $\boldsymbol{\theta}_{max}$,

kept to a minimum.

is defined as

$$\begin{aligned} \text{PI}(\boldsymbol{\theta}^*) &= P(V(\boldsymbol{\theta}^*) \geq V_{max}) \\ &= \Phi(\frac{\mu(\boldsymbol{\theta}^*) - V_{max}}{\sigma(\boldsymbol{\theta}^*)}) \end{aligned} \tag{1.18}$$

where $\mu(\boldsymbol{\theta}^*)$ and $\sigma(\boldsymbol{\theta}^*)$ are the predicted mean and standard deviation of $V(\boldsymbol{\theta}^*)$ outputted by the GP in equation (5.3), and Φ is the standard normal cumulative distribution function. This formulation is simply the area under the curve of the predicted Gaussian distribution of $V(\boldsymbol{\theta}^*)$ from V_{max} to ∞. Figure 1.10 shows the PI acquisition function for the case where $\boldsymbol{\theta}$ is 1-dimensional.

The consequence of using equation (1.18) is that the points near $\boldsymbol{\theta}_{max}$ that have infinitesimally higher expected values with low uncertainty[7] are preferred to locations with high expected values and high uncertainty. This results in a greedy acquisition function that prefers *exploitation* over *exploration*. To prevent this from happening in the early stages of the algorithm where exploration is preferred, a trade-off parameter $\xi \geq 0$ is added to the current V_{max} that results in exploration[8] [16, 23]. The formulation of the PI becomes

$$\begin{aligned} \text{PI}(\boldsymbol{\theta}^*) &= P(V(\boldsymbol{\theta}^*) \geq V_{max} + \xi) \\ &= \Phi(\frac{\mu(\boldsymbol{\theta}^*) - V_{max} - \xi}{\sigma(\boldsymbol{\theta}^*)}) \end{aligned} \tag{1.19}$$

The value of ξ is decreased in the later stages of the optimization algorithm to switch from exploration to exploitation [16, 23].

The Upper Confidence Bound (UCB) acquisition function is defined as

$$\text{UCB}(\boldsymbol{\theta}^*) = \mu(\boldsymbol{\theta}^*) + \kappa\sigma(\boldsymbol{\theta}^*) \tag{1.20}$$

where $\kappa \geq 0$, which much like ξ in PI, is a parameter set by the user that controls the amount of exploration and exploitation.

Another common acquisition function is derived from maximizing the expected value of the *improvement* on

[7] most of the area of the distribution of these locations are concentrated just above V_{max} since these Gaussians are very thin

[8] by doing this the most of the area under these thin Gaussians are not considered, whereas the area under the wide Gaussians where there is high uncertainty won't change much

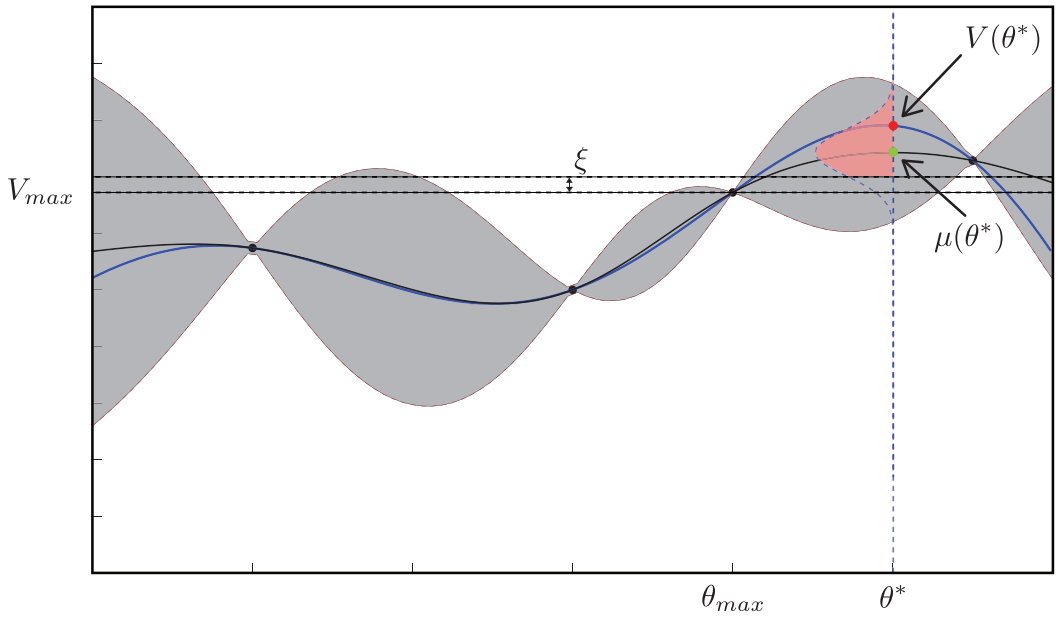


Figure 1.10: The Probability of Improvement (PI) Acquisition Function. V_{max} corresponding to θ_{max} is the current maximum observation. The PI for a a candidate point θ^* corresponds to the pink area under the Gaussian curve from $V_{max} + \xi$ to ∞. $V(\theta^*)$ is unknown.

the current max, where the improvement function as defined in [20, 24] is

$$\mathrm{I}(\boldsymbol{\theta}^*) = \max\{0, V_{t+1}(\boldsymbol{\theta}^*) - V_{max}\}. \tag{1.21}$$

where $V_{t+1}(\boldsymbol{\theta}^*) \sim \mathcal{N}(\mu(\boldsymbol{\theta}^*), \sigma^2(\boldsymbol{\theta}^*))$. The new candidate is a point that maximizes the expected value of improvement, which is the integral given by

$$\mathbb{E}(\mathrm{I}) = \int_{\mathrm{I}=0}^{\mathrm{I}=\infty} \mathrm{I} \frac{1}{\sqrt{2\pi\sigma^2(\boldsymbol{\theta}^*)}} \exp(-\frac{(\mu(\boldsymbol{\theta}^*) - V_{max} - \mathrm{I})^2}{2\sigma^2(\boldsymbol{\theta}^*)}) d\mathrm{I} \tag{1.22}$$

which as demonstrated in [20, 25] can be calculated analytically as

$$\mathrm{EI}(\boldsymbol{\theta}^*) = \begin{cases} (\mu(\boldsymbol{\theta}^*) - V_{max})\Phi(Z) + \sigma(\boldsymbol{\theta}^*)\phi(Z) & \sigma(\boldsymbol{\theta}^*) > 0 \\ 0 & \sigma(\boldsymbol{\theta}^*) = 0 \end{cases} \tag{1.23}$$

where

$$Z = \frac{\mu(\boldsymbol{\theta}^*) - V_{max}}{\sigma(\boldsymbol{\theta}^*)} \tag{1.24}$$

and $\phi(.)$ is the standard normal probability density function(PDF).

1.4.2 Convergence of Bayesian optimization

The EI acquisition function has been shown to converge when the priors on the functions (GP hyperparameters) are fixed [26, 27]. In practice however, the observations arrive sequentially and therefore an existing set of hyperparameters might not represent the new data well. [25] suggests updating the hyperparameters based on the observations as they become available. The optimization is carried out in two steps. First the value of α^2 that maximizes the log-likelihood in equation (1.16) (given the other hyperparameters) is found by solving

$$\begin{aligned} \frac{\partial p(\boldsymbol{y} \mid Z)}{\partial \alpha^2} &= 0 \\ &= \frac{1}{2}\boldsymbol{y}^\top \boldsymbol{K}_y^{-1} \frac{\partial \boldsymbol{K}_y}{\partial \alpha^2} \boldsymbol{K}_y^{-1} \boldsymbol{y} - \frac{1}{2}\mathrm{tr}(\boldsymbol{K}_y^{-1} \frac{\partial \boldsymbol{K}_y}{\partial \alpha^2}) \end{aligned} \tag{1.25}$$

where $\boldsymbol{K}_y = K + \sigma_n^2 \boldsymbol{I}$. Second, with the obtained value of α^2, the other hyperparameters are found by maximizing equation (1.16) as discussed in section 1.3.1.

For $\alpha^2 >> \sigma_n^2$ (or any value of α^2 when $\sigma_n^2 = 0$), the closed form expression for α^2 can be obtained from equation (1.25) which gives

$$\alpha^2 = \frac{\boldsymbol{y}^\top \boldsymbol{R}^{-1} \boldsymbol{y}}{n} \tag{1.26}$$

where R is the part of the covariance function in equation (1.9) that doesn't involve α^2 ($K = \alpha^2 R$). The sequential change of the hyperparameters with methods such as the one described above may have an adverse effect on the convergence and can cause the EI to not converge at all. An example of this for a special case was shown in [28], which was then extended by [27] for a more general setting. [27] addresses this issue by showing that convergence can still be achieved with the choice of $\alpha^2 = \boldsymbol{y}^\top \boldsymbol{R}^{-1} \boldsymbol{y}$ instead of equation (1.26).

The convergence of EI shown in [27] assumes that the measurements are noise-free, which usually is not the case when dealing with real systems. In practice a small "artificial" noise is added to the diagonal terms of the covariance matrix even when the measurements are noise-free; this is done to ensure that the covariance matrix is positive-definite and to avoid badly scaled matrices. The complications of this practice on the convergence presented in [27] is not clear.

A common issue with the Bayesian optimization approach as illustrated in Figure 1.5 is that the increase of

the length-scales increases the confidence of the model which results in increasingly more exploitation than exploration that can cause the search to get stuck in a local optimum. [29] attempts to alleviate this issue through an algorithm that puts a bound on the length-scales. The upper bound is decreased when the model becomes too confident, which in turn forces the search to explore more. This will cause the upper bound to decrease more and more, and approach the lower bound, which is fixed and is initialized by the user at the beginning of the search. [29] shows the convergence of the EI acquisition function with their algorithm even when the measurements are noisy. The issue with this approach however is that prolonging the search of the Bayesian optimization defeats the purpose of using it in the first place and goes against its promise of minimizing function evaluations. Unfortunately to the best of our knowledge, alternative methods on the convergence of the EI acquisition function with noisy measurements have not been offered in the literature, and the question of how best to avoid prolonging the search (possibly infinitely) in order to not get stuck in local optima, and at the same time assume convergence remains open.

1.5 Related Work

Most of the early work in Direct RL such as TD(λ) [30], Sarsa [31] and Q-learning [32, 33], dealt with discretized state space and with learning of the *Value function*, the discounted future rewards of being in a state and following the current policy, and the *Q-function*, the discounted future reward from executing an action in a state, and following the current policy thereafter. This was then extended to continuous state space by using function approximators that generalize the Value function (or Q-function) for the entire space. These include the use of sparse-coarse-coded function approximator (CMAC [34]) in [35, 36], the neural nets in [37, 38], the radial basis functions (RBF) in [39] and the Gaussian process in [40]. As mentioned before these Direct methods of learning require a lot of interaction with the real-world and are not data-efficient, therefore the focus was shifted to the Model-based methods. In [41], a proposed Model-based learning algorithm used the nonlinear dynamical factor analysis (NDFA) introduced in [42] to represent the dynamical model. A radial basis function (RBF) was used in [43] to approximate the Value function, with an additional RBF network used for learning the dynamical model. In [44], a GP was first used to model the dynamical training data that was obtained by sampling the states-actions and their subsequent states. A second GP was then used to represent the Value function, and in a very similar fashion to the dynamic programming, the updating of the Value function, and the search for better policies was done off-line with the use of the probabilistic model of the dynamics.

One Model-based approach is to use the learned dynamical model to carry out the optimization steps of

the total rewards in equation (1.4) (with respect to the policy parameters) in simulation. The general steps of such a simulation-based algorithm is summarized in Table 1.1. PILCO [45, 46] uses this approach that amongst the Model-based algorithms studied for this work has produced the most efficiency in terms of the real-world interactions. This is done by learning a probabilistic model of the dynamics of the system by a GP, and finding a locally optimal set of policy parameters by utilizing a gradient-based optimizer that evaluates the total expected reward based on simulated runs for T time steps. The new policy parameters are then tried on the real system and if the total reward is close to that of the simulation run, then the duration of the simulation, T, is increased (since it would suggest that the model is accurate), and the optimization step is repeated. Such a method does not use the previous policies already executed on the real system to aid the search for a new policy. Moreover, the uncertainty of the expected total reward of a policy obtained in the simulation step is not explicitly considered in the decision making of the algorithm.

A Model-based Optimization Algorithm [46]	
1:	Run initial random policies, gather training data for the dynamical model
2:	Train dynamics-GP
3:	Find policy by optimizing the rewards function using simulation
4:	Execute policy on real-world and observe total reward
5:	Optional: increase final horizon time if observed total reward closely matches the one from simulation
6:	Return to 2

Table 1.1: The steps of a Model-based optimizer similar to the work of [46]. The partial derivatives of the total reward with respect to the policy parameters can be calculated as suggested in [46]

The use of Bayesian optimization to find a control policy was previously presented in [47], where a double-pendulum was kept in a balanced position by using a control policy consisting of 8 parameters[9]. Success was defined as keeping the poles balanced between $\pm 36^{\circ}$ and was achieved after 195 evaluations, or episodes of real-time interactions. Much like other data-inefficient Direct learning algorithms, the dynamical model is not learned and the real-world trajectories are essentially discarded. The steps of such a simulation-free algorithm are summarized in Table 1.2.

Our method differs from the above mentioned approaches in that learning is done both with Direct RL from

[9]The pendulums are already in the up-right position and the goal is to keep them balanced. This is not a difficult problem to solve and can be achieved by utilizing a linear controller.

Direct Bayesian Optimization Algorithm [47]	
1:	Run initial random policies, gather training data for the rewards model
2:	Train rewards-GP
3:	Find policy by Bayesian optimization
4:	Execute policy on real-world and observe rewards
6:	Return to 2

Table 1.2: The steps of a Direct Bayesian optimization algorithm as presented in [47].

the model of the rewards function, and with simulations from the probabilistic dynamical model. This is similar to the Dyna structure mentioned earlier. At the initial stages of the algorithm, learning is primarily direct, and is then shifted gradually to model-based as the dynamical model is learned better. Throughout the algorithm, the already-tried policies on the real-system (and simulation), and the corresponding rewards are stored and used at every policy search in the optimization step, giving a more global view and search for an optimal solution. Since most of the planning is done from simulations, the real-world interaction is kept to a minimum.

Chapter 2

Proposed Method

As shown in equation (1.4), solving the problem described requires the optimization of the total reward function, $V(\boldsymbol{\theta})$. To obtain the total reward for a finite-horizon time step, a policy is executed on the system and the immediate rewards at each state visited are summed together; this will be referred to as a running a policy on the real-world. The real-world interaction is desired to be kept to a minimum and the goal is to have an efficient algorithm that can find candidates that maximize $V(\boldsymbol{\theta})$ based on as few real-world tries as possible. This is where the Bayesian optimization can be utilized as a direct learning algorithm to find the policy parameters that give the most rewards. As mentioned earlier, [47] implements this approach to find a policy to keep a double pendulum balanced without learning the dynamics. This approach however essentially discards the real-world trajectories much like the other direct learning algorithms.

In the proposed method, two probabilistic models are learned based on the interactions with the real-world. The first model is the rewards-GP (RGP), which as mentioned in section 1.4 maps the policies to the total rewards. The second model, the dynamics-GP (DGP), is created based on the state trajectories visited during the real-world interactions, and is used to simulate an estimated state trajectory for a given policy. Based on this simulated trajectory, an estimate of the total reward is computed. The Bayesian optimization which is based on the RGP, is used to guide the search for a suitable policy. The suggested policy found by the Bayesian optimization is executed on the DGP, which will predict the expected total reward and its uncertainty. This information is used to update the RGP, which as will be shown, results in the reduction of uncertainty at and around the simulated policy that provide better guidance for the Bayesian optimization search in that region in the future. The Bayesian optimization step and the update using simulations continues until a candidate is found that according to both the RGP and the DGP is highly likely to improve

the current best policy, and only then that candidate policy is applied on the real-world. The trajectory of states resulting from executing this candidate on the real-world and the obtained total returned reward, are used to update and re-train the DGP and the RGP. This will decrease the uncertainty of the model of the dynamics around that trajectory, and also of the model of the rewards around the executed policy, which result in better decision making in those regions in the future. As more trajectories are visited and more experience is gained, planning relies increasingly more on simulations using the DGP, which can considerably reduce the amount of real-world interaction. Before the proposed algorithm is formally presented, its key components are covered in detail in the following sections.

2.1 Simulation Using the Dynamical Model (DGP)

Simulations are performed by utilizing the learned probabilistic dynamical model based on the training data. The training data is gathered as input $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{u})$ where $\boldsymbol{x} \in \mathbb{R}^E$ and $\boldsymbol{u} \in \mathbb{R}^U$ are the current state and current action, and output $\boldsymbol{x'} \in \mathbb{R}^E$, which is the next state. As suggested in [44], a GP is used for every dimension of the state making a total number of E GPs as shown in Figure 2.1. Each of the GPs is trained based on a training set of N points in the form of $\mathcal{D}_i = \{(\boldsymbol{X}, \boldsymbol{U}), \boldsymbol{X}'(i)\}$, where $i = 1 : E$, $\boldsymbol{X} \in \{\boldsymbol{x}_1 \dots \boldsymbol{x}_N\}$, $\boldsymbol{U} = \{\boldsymbol{u}_1 \dots \boldsymbol{u}_N\}$ and $\boldsymbol{X}'(i)$ is the i^{th} dimension of the next state.

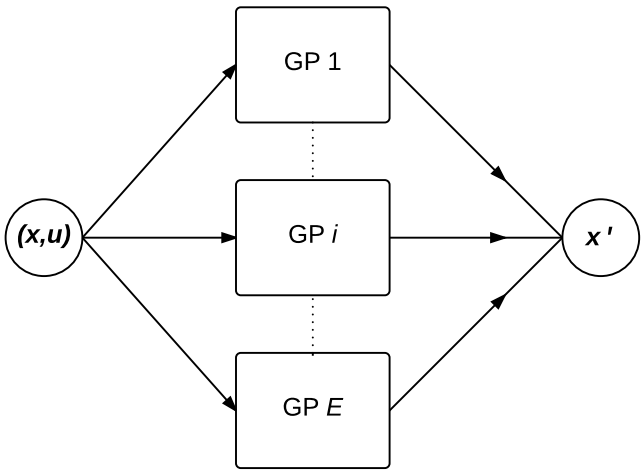


Figure 2.1: Dynamical model. A GP is used for every dimension of the state and are then combined to get the next state.

The DGP is the collection of these GPs and its predictive output is obtained by putting the outputs of the E GPs together as

$$\boldsymbol{x}' \sim \mathcal{N}(\begin{bmatrix} \mu_1 \\ \vdots \\ \mu_E \end{bmatrix}, \text{diag}(\sigma_1^2 \dots \sigma_E^2)) \tag{2.1}$$

The GPs are trained after new training data becomes available. The above diagonal predictive covariance of the dynamical model assumes no correlation between the outputs of each of the GPs, i.e. the state at the next time step. Such a model that treats outputs as independent does not exploit their possible similarities. Unfortunately joint prediction of dependent GPs is difficult since defining cross-covariance functions that result in positive-definite covariance matrices is problematic, and therefore is not considered here. This issue is discussed in detail in [48] and different ways are suggested for addressing it.

Given a deterministic dynamical model, running the simulation would be simply executing a policy for T steps and computing the immediate reward at each state visited. But since the dynamical model is probabilistic, there is uncertainty about every visited state during simulation. Two possible ways of performing simulation with a probabilistic model in order to get the expected rewards and its variance are the following: the first is to construct/approximate the probability distribution (PDF) of the state at each iteration and compute the immediate expected reward at each step, and the second is to use a Monte Carlo approximation method.

Given the initial state and the policy parameters, the distribution of the next state can be computed as the output of the dynamical model which is Gaussian. Using the work of [49] on the Gaussian process with uncertain inputs, the Gaussian pdf of this state can be inputted into the GP along with the action (also a random variable) to give the distribution of the next state. This non-Gaussian output[1] is approximated by a Gaussian distribution using moment matching[2] and is used again as the input of the next step

$$\boldsymbol{x}_0 \rightarrow \boldsymbol{x}_1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \dots\dots \boldsymbol{x}_T \sim \mathcal{N}(\boldsymbol{\mu}_T, \boldsymbol{\Sigma}_T) \tag{2.2}$$

To avoid the approximation of the distributions as Gaussians at every step, a Monte Carlo method can be employed. One iteration of the Monte Carlo approximation is to start from the initial state and action, then draw a sample state from the output of the DGP which is exactly Gaussian. This sampled state, and the resulting deterministic action, is then passed on to the DGP as an input and the process continues for T steps. This is done for a number of runs that are used to approximate the expectation and the variance of

[1] mapping a Gaussian through a non-linear function (in this case a GP) results in a non-Gaussian distribution
[2] for $E > 1$ the off-diagonal elements of the covariance matrix can be computed as explained in [46]

the total reward.

2.1.1 Dealing with Growth of Training Data

As mentioned in Chapter 1, the training data for the DGP continues to grow during the interaction with the real system. For every real-world interaction, there are an additional T training inputs and output that are gathered, and if not managed properly, the use of GPs for dynamical modeling becomes infeasible. The goal here is to have a model that deals with the growing size of the training data by managing the new incoming data well.

As mentioned in the last chapter, one approach to deal with the growth of the data would be to use local GPs with a fixed number of members to represent the dynamical model in different regions of the state-space. An alternative approach to the local GPs taken here is to place an upper limit on the number of data points of the dynamical model, $N_{\max}$. Once this limit is reached, the observations from the state trajectories that are close to each other are put into different bins by using a clustering algorithm, and the centers of these bins are used as the training data of a global dynamical model. The clustering algorithm used here is *DP-means* from the work in [50], which is explained in detail in section 4.1.3.1. This algorithm is similar to the K-means, except that a new cluster is created when a point is farther than a distance λ away from the center of every existing cluster, which means that the maximum radius of every bin is λ. The choice of λ is problem-dependent and is selected based on the desired resolution on the training points. The topic of clustering and the limitations of various existing adaptive approaches, such as the computational cost and the need of prior selection of parameters (e.g. λ in DP-means) is covered in detail in Chapter 4. This inspired the need of a true adaptive clustering algorithm that is easy and fast to implement, which as mentioned earlier, led to the development of the EM$^+$ algorithm presented in section 4.2.

2.1.1.1 A Note on Clustering

Figure 2.2 considers a simple clustering problem where the state is x, the horizontal axis, and the output is $y = f(x)$. Clustering only the x values can be problematic in the region where the slope of the function is high since the many states closely related on the x-axis pertain to a large range of y values. The same is true when clustering only the y value in the flat regions of the function. A solution would be to cluster the $x - y$ values together as shown in the Figure 2.2. When clustering the dynamical data, the data to be

clustered has the form $(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')$, the current state and action, along with the next state.

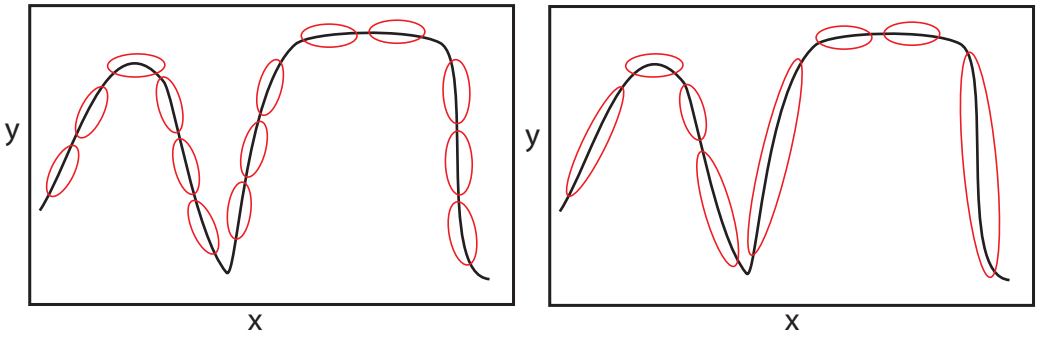


Figure 2.2: A simple clustering problem. The right graph shows clustering the x-values only, while the left graph shows clustering x-y values together.

2.2 Rewards-GP (RGP)

In the proposed algorithm, the search for new policies is carried out by the Bayesian optimization, which uses an acquisition function that is based on the predictions of the RGP. The mapping of policies to rewards modeled by the RGP is therefore an essential part of the search for better policies. According to [16] the choice of the covariance function (used by the RGP) is critical in the Bayesian optimization. [51] advises against using the SE covariance function as it assumes unrealistic smoothness that is not suitable for Bayesian Optimization. Instead it suggests using the M52 covariance function[3] from the Matérn family of functions explained in section 1.3, which enables the GP to capture sudden changes on the output. To allow more flexibility in modeling, we use a combination of SE and M52 covariance functions. The hyperparameters of the RGP are optimized at every step by maximizing the evidence as explained in section 1.3.1.

The choice of the PI and the UCB acquisition functions in the Bayesian optimization both require a selection of a parameter that promotes exploration at the beginning and exploitation towards the end of the search (see section 1.4.1). A *cooling* schedule on this parameter as suggested by [23] can be used, that initially starts with a high value that promotes global searches, and gradually gets reduced which shifts the search locally as the algorithm nears its end. Setting this cooling schedule however would require some experimentation and is problem-dependent. The PI and the UCB acquisition functions are therefore not employed in this work. Instead the EI acquisition function has been selected, as it automatically combines exploration and

[3]M52 function looks similar to SE except that instead of having a bell shaped peak, it is thinner on top, which results in less smoothness

exploitation, and is the preferred choice in the literature [16, 51].

The returned rewards from the real-world provide noiseless observations that are used as training points of the RGP. The next section explains how simulated runs from section 2.1 are used as uncertain measurements that are included in the training data set of the RGP.

2.2.1 Combining Predictions of RGP and Uncertain Measurements from Simulations Using DGP

Given the RGP model with N existing input observations $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1 \dots \boldsymbol{\theta}_N\}$, and output observations $\boldsymbol{V} = \{V_1, \dots V_N\}$ with variance of $\boldsymbol{\sigma}^2 = \{\sigma_1^2, \dots \sigma_N^2\}$, a prediction for the mean and the variance of a candidate policy $\boldsymbol{C}$ can be written as

$$\begin{aligned} V_{C,R} &= \boldsymbol{K}(\boldsymbol{C}, \boldsymbol{\Theta})(\boldsymbol{K}(\boldsymbol{\Theta}, \boldsymbol{\Theta}) + \boldsymbol{\Sigma})^{-1}\boldsymbol{V} \\ \sigma_{C,R}^2 &= K(\boldsymbol{C}, \boldsymbol{C}) - \boldsymbol{K}(\boldsymbol{C}, \boldsymbol{\Theta})(\boldsymbol{K}(\boldsymbol{\Theta}, \boldsymbol{\Theta}) + \boldsymbol{\Sigma})^{-1}\boldsymbol{K}(\boldsymbol{\Theta}, \boldsymbol{C}) \end{aligned} \tag{2.3}$$

where $\boldsymbol{\Sigma} = \text{diag}(\boldsymbol{\sigma}^2)$ (this is the same as equation (5.3) with $\sigma_1^2 = \dots = \sigma_N^2 = \sigma_n^2$). In the above equations the subscript R means that this prediction has come from the RGP. It was shown in the previous sections that the simulations done by the DGP can also be used to obtain a prediction of the total reward for the candidate policy $\boldsymbol{C}$. This prediction is written as $V_{C,D}$ with variance of $\sigma_{C,D}^2$, where the subscript D denotes that the DGP was used for this prediction. The RGP is updated by adding this uncertain measurement to its input and output training set, where the input training set become $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1 \dots \boldsymbol{\theta}_N, \boldsymbol{C}\}$, with the output of $\boldsymbol{V} = \{V_1, \dots V_N, V_{C,D}\}$, and variance of $\boldsymbol{\sigma}^2 = \{\sigma_1^2, \dots \sigma_N^2, \sigma_{C,D}^2\}$. The predictions of this updated RGP will be now different at the policy $\boldsymbol{C}$, and also at policies near $\boldsymbol{C}$. To see exactly how this affects the predictions, the prediction of the updated RGP at $\boldsymbol{C}$ is written

$$\begin{aligned} V_{C,R_{\text{updated}}} &= \begin{bmatrix} \boldsymbol{K}(\boldsymbol{C}, \boldsymbol{\Theta}) & K(\boldsymbol{C}, \boldsymbol{C}) \end{bmatrix} \boldsymbol{M}^{-1} \begin{bmatrix} \boldsymbol{V} \\ V_{C,D} \end{bmatrix} \\ \sigma_{C,R_{\text{updated}}}^2 &= K(\boldsymbol{C}, \boldsymbol{C}) - \begin{bmatrix} \boldsymbol{K}(\boldsymbol{C}, \boldsymbol{\Theta}) & K(\boldsymbol{C}, \boldsymbol{C}) \end{bmatrix} \boldsymbol{M}^{-1} \begin{bmatrix} \boldsymbol{K}(\boldsymbol{C}, \boldsymbol{\Theta}) & K(\boldsymbol{C}, \boldsymbol{C}) \end{bmatrix}^\top \end{aligned} \tag{2.4}$$

where

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{K}(\boldsymbol{\Theta},\boldsymbol{\Theta}) + \boldsymbol{\Sigma} & \boldsymbol{K}(\boldsymbol{C},\boldsymbol{\Theta})^\top \\ \\ \boldsymbol{K}(\boldsymbol{C},\boldsymbol{\Theta}) & K(\boldsymbol{C},\boldsymbol{C}) + \sigma^2_{C,D} \end{bmatrix} \tag{2.5}$$

By using the matrix block inversion property in equation (A.9) of the section A.1 and some simplification, the predictive equations above can be rearranged that result the following

$$\begin{aligned} V_{C,R_{\text{updated}}} &= \frac{\frac{V_{C,D}}{\sigma^2_{C,D}} + \frac{\boldsymbol{K}(\boldsymbol{C},\boldsymbol{\Theta})(\boldsymbol{K}+\boldsymbol{\Sigma})^{-1}\boldsymbol{V}}{K(\boldsymbol{C},\boldsymbol{C})-\boldsymbol{K}(\boldsymbol{C},\boldsymbol{\Theta})(\boldsymbol{K}+\boldsymbol{\Sigma})^{-1}\boldsymbol{K}(\boldsymbol{\Theta},\boldsymbol{C})}}{\frac{1}{\sigma^2_{C,D}} + \frac{1}{K(\boldsymbol{C},\boldsymbol{C})-\boldsymbol{K}(\boldsymbol{C},\boldsymbol{\Theta})(\boldsymbol{K}+\boldsymbol{\Sigma})^{-1}\boldsymbol{K}(\boldsymbol{\Theta},\boldsymbol{C})}} \\ &= \frac{\frac{V_{C,D}}{\sigma^2_{CD}} + \frac{V_{C,R}}{\sigma^2_{C,R}}}{\frac{1}{\sigma^2_{C,D}} + \frac{1}{\sigma^2_{C,R}}} \end{aligned} \tag{2.6}$$

$$\begin{aligned} \sigma^2_{C,R_{\text{updated}}} &= \frac{1}{\frac{1}{\sigma^2_{C,D}} + \frac{1}{K(\boldsymbol{C},\boldsymbol{C})-\boldsymbol{K}(\boldsymbol{C},\boldsymbol{\Theta})(\boldsymbol{K}+\boldsymbol{\Sigma})^{-1}\boldsymbol{K}(\boldsymbol{\Theta},\boldsymbol{C})}} \\ &= \frac{1}{\frac{1}{\sigma^2_{C,D}} + \frac{1}{\sigma^2_{C,R}}} \end{aligned} \tag{2.7}$$

The equations above show the important benefit of updating the RGP with simulations. Including an uncertain measurement from simulation for a policy $\boldsymbol{C}$, has the effect that at $\boldsymbol{C}$, the prediction of the updated RGP combines the prediction of the original RGP and the simulated measurement as an inverse-variance weighted sum. The variance of this prediction is less than both $\sigma^2_{C,D}$, the variance of the simulated run, and $\sigma^2_{C,R}$, the variance of the original RGP. Since the predictions of the RGP are continuous, the decrease of uncertainty at $\boldsymbol{C}$ will also mean that the uncertainty of policies near $\boldsymbol{C}$ are reduced. It can also be seen that the combined prediction of the reward at $\boldsymbol{C}$ puts more weight on the less uncertain prediction, therefore highly certain measurements in simulated runs act as *corrections* to the regions of the RGP around that candidate, effectively reducing the uncertainty. Adversely, no significant change will be made to the RGP if the measurements from the simulations are highly uncertain.

2.3 Bayesian Optimization in the Proposed Method

The Bayesian optimization search for policies with higher rewards is performed by the global maximization of the EI acquisition function. In the Bayesian optimization literature, the $DIRECT$ [52] algorithm, which is a derivative-free global algorithm that relies on Lipschitzian-based partitioning techniques is a commonly used. To our experience, this method does not have a clear advantage over other global optimization methods[4], and can often fail where other algorithms succeed. [53] provides a good review and comparison of different global optimization algorithms[5] including the $DIRECT$ method, and the others that are used in this paper.

Unfortunately global optimizers in practice do not guarantee that the solution found is the global optimum since from a practical point each search can only be done for a number of iterations and a limited time. Failure in finding a global solution results in the Bayesian optimization suggesting candidates that are less likely to improve over the best solution, and this may prolong the interaction with the real-world. To reduce the likelihood of this, two global algorithms are used in series, the first is the $DIRECT$ algorithm, and the second is the Multi-Level Single-Linkage (MLSL) algorithm [54], which is a multi-start stochastic search technique that uses a local algorithm to carry out parallel local searches. A common technique in global optimization is to "polish" the results of the global search by a local search. For this, two local algorithms are used, the derivative-free $COBYLA$ [55] algorithm, and a derivative based algorithm such a LBFGS[6]. It should be noted that global here is referring to a solution within the search bounds of the Bayesian optimization, which are defined by $[L_{\text{bound}}, U_{\text{bound}}]$, where L_{bound} is the lower bound, and U_{bound} is the upper one. As will be seen shortly, during the algorithm, the search regions for a set of parameters can be adjusted to concentrate the search locally.

Upon the completion of the Bayesian optimization search, a candidate policy is obtained with a corresponding reward and uncertainty from the RGP. Using the DGP the estimate of the reward is calculated, which can then be used to update the RGP and obtain the updated estimates of the reward and the variance shown in equations (2.6) and (2.7)[7]. The decision to execute this candidate policy on the real-world will be based on *expected improvement percentage* (EIP), which for candidate $\boldsymbol{C}$ is defined as

[4]Donarl R. Jones has authored both the $DIRECT$ [52] algorithm and one of the most commonly cited papers in Bayesian optimization [25]

[5]some that are not free

[6]the algorithms mentioned here are all included in the NLopt package found at http://ab-initio.mit.edu/wiki/index.php/NLopt

[7]predictive equations in (2.4) give the same values as the equations (2.6) and (2.7), although the latter equations are computationally cheaper and are therefore used in the algortihm

$$EIP_C = \frac{EI(\boldsymbol{C}, V_{max})}{V_{max}} \tag{2.8}$$

where the mean and variance of the updated RGP are used in calculating the $EI(\boldsymbol{C}, V_{max})$, the expected improvement (EI) in equation (1.23). The EIP is a relative expected improvement over V_{max}, the current best total reward amongst the policies that have been tried on the real-world (not including the simulated runs). Much like the EI acquisition function values, the EIP_C considers both the probability of improvement and the magnitude of this improvement, where the uncertain regions of the DGP and the RGP with high expected rewards are encouraged to be explored. If the EIP_C value of a candidate is high, it means that running that candidate on the real-world is likely to improve the current best policy. Regions of the policy space in which high EIP values are obtained will be referred to as *promising* regions and the aim is for the Bayesian optimization to discover these regions. A promising region is declared when the EIP_C value of a policy is higher than a threshold, tol_{EIP} (this threshold can be adaptive). A conservative approach would be to immediately switch to the real-world once such a policy is found. However, running simulations as shown earlier can enhance the representation of the RGP, and therefore exploiting the dynamical model by further search and update could produce better candidates. For this reason the search is continued as long as the new candidates proposed by the Bayesian optimization search that are simulated produce EIP values in promising regions. Since the new candidates are always expected to do better than the previous ones, if the search is to be continued in simulation, the definition of a promising region is redefined by increasing the current value of tol_{EIP} to the current EIP_C. Once a future candidate exists this promising region, the search is stopped and the last candidate, i.e. the candidate with the highest EIP value, is executed on the real-world.

The decision making becomes particularly more challenging when candidates with high EIP values are not found. This can be the result of selecting a tol_{EIP} value that is too high, which means that it is necessary to have a cooling schedule on tol_{EIP} if candidates offered by the Bayesian optimization step consistently have EIP_C values that are lower than tol_{EIP}. The other scenario that this happens is when predictions using the current representation of the RGP have little magnitudes of improvement, that results in the search moving away from the observations into highly uncertain regions where the uncertainty and the predicted mean flatten out due to the properties of GPs. It is possible for the uncertainty of these regions to be reduced by better predictions from simulated runs as described above, but if the dynamical model is also non-informative, the search will become stranded in uncertain regions with low magnitude of improvement[8].

[8]DGP also flattens away from observations

It should be stressed however, that this does not mean that better results do not exist around current observations. Instead, it reflects that the current representation of the RGP is possibly unable to predict them[9]. This especially happens when the number of parameters of the policy are high, which can makes it difficult to construct a good representation of the mapping of policy parameters to the rewards with the RGP. To prevent the search to continuously being carried out in highly uncertain regions, the scope of the search is reduced to concentrate the search around policies that are known to produce high rewards. In this case if the RGP fails to give good predictions, the DGP can come to the rescue and reduce the uncertainties. In scenarios where simulated runs also have high uncertainty but show a high value in the magnitude of the likely improvement, the use of the EIP encourages exploration by visiting the corresponding regions.

Based on the above if low EIP values are obtained repeatedly for i_{max} number of iterations (e.g. $i_{max} = 5$), the definition of a promising region is changed by reducing the value of tol_{EIP} (e.g. $tol_{EIP} = \frac{1}{2}tol_{EIP}$), and the bounds on the search in the global optimization of the acquisition function are reduced to focus the search around $\boldsymbol{\theta}_{\text{best}}$, our current best policy with the highest reward that has been executed on the real-world. For this we define a variable *scope*, where $scope = 1$ corresponds to the maximum search scope on the allowable search region of $[L_{\text{bounds}}, U_{\text{bounds}}]$, where U_{bounds} is the upper bound, and L_{bounds} is the lower bound. The scope of the search is changed by reducing the value of *scope* (e.g. $scope = \frac{1}{2}scope$), and the new search region will be set to the region of $[L_{\text{search}}, U_{\text{search}}]$ obtained by

$$\begin{aligned} U_{\text{search}} &= \min(U_{\text{bounds}}, \boldsymbol{\theta}_{\text{best}} + scope \times |U_{\text{bounds}}|) \\ L_{\text{search}} &= \max(L_{\text{bounds}}, \boldsymbol{\theta}_{\text{best}} - scope \times |L_{\text{bounds}}|) \end{aligned} \tag{2.9}$$

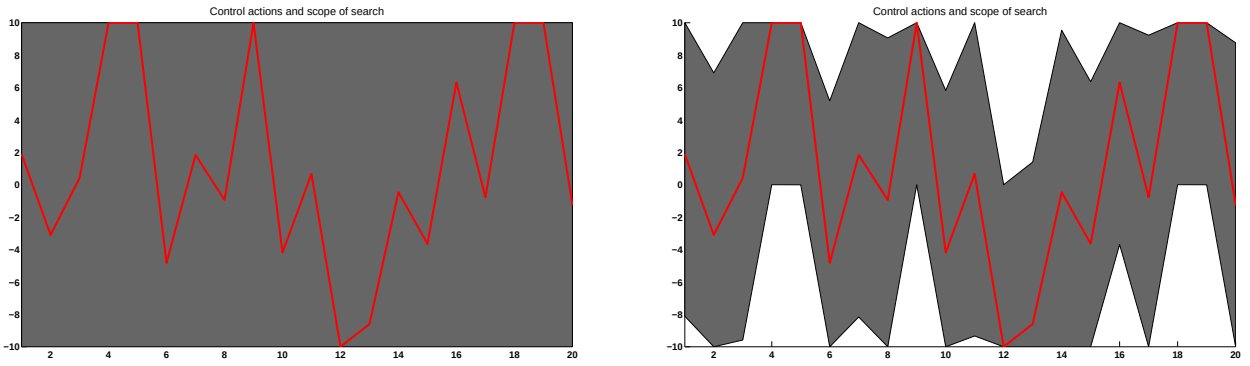
Figure 2.3 shows an example of reducing the scope of the search.

As a consequence of the above, convergence[10] in the algorithm is reached under two conditions. The first is when the candidates from the Bayesian optimization are highly unlikely to improve the best results so far, i.e. they have very low EIP values, and the second is when the scope of the search has been reduced to a point that the relative change between the suggested policies approach zero. This relative change is defined as

[9] this has to do with the values of the hyperparameters.

[10] By convergence here we mean that the algorithm stops improving, this could happen before finding a successful solution that sets $success = 1$.

$$\Delta\boldsymbol{\theta}_{\text{relative}} = \frac{U_{\text{bounds}} - L_{\text{bounds}}}{\boldsymbol{\theta}_{\text{best}}} \tag{2.10}$$



(a) full search scope

(b) reduced search scope

Figure 2.3: Example of search scope for candidate consisting of 20 actions/parameters. The bounds on the parameters are $[-10, 10]$. The red line shows the current best policy that was implemented on the real-world, and the grey area shows the scope of the search where for (a) is set to the original limits at [-10,10], and for (b) it is recuded by half around the the current best candidate (red line).

2.4 Steps of The Proposed Algorithm

We now present the steps of the proposed algorithm. These steps are summarized in Table 2.1. Success is a problem-dependent notion that is defined when the objective of the control algorithm has been met, e.g. the final state of the trajectory is within a tolerance of the pre-defined goal state; this condition is represented by a boolean value named *success* which is originally set to zero. *convergence* is another boolean variable that is originally set to zero, and is changed to 1 when convergence as described earlier is reached. To begin the algorithm, a number of random policies are generated and executed on the real-world with the purpose of gathering the initial data for the DGP and the RGP, which are then trained by maximizing the evidence as described in section 1.3.1. Following the guidelines explained in the previous section that are outlined in the next section and in Table 2.2, the algorithm then looks for a new candidate to be tried on the real-world using the Bayesian optimization with the help of simulations. Once a suitable candidate is found, it is executed on the real-world. This will result in a set of new experiences that include new state trajectories, and a new noiseless total reward, which are used as observations of the DGP and the RGP respectively. By updating and re-training the DGP and the RGP, future decisions are made with more confidence. Previous policies in the RGP that were updated by the DGP can now be simulated again since a better representation

of the DGP is available. The estimate of the expected reward and its variance of a candidate obtained by old simulation are combined with estimates from the new simulation run using the inverse-variance weighted sum equations in equations (A.14) and (A.15). The combined simulated estimates for a given policy will have less uncertainty than both the old and new simulated runs and result in better decision making. A search for a candidate is then carried out again, and the process is continued until either the success or the convergence is met. Figure 2.4 demonstrates the way different components of the algorithm are connected.

Proposed Algorithm	
1:	initialize random policies and execute them on real world to gather initial data for DGP and RGP , set *convergence* and *success* to 0
2:	while ($convergence == 0$) && ($success == 0$)
3:	find a new candidate (see Table 2.2 and section 2.4.1)
4:	execute policy on real-world, gather data for DGP and RGP
5:	train DGP and RGP
6:	re-run and update previous simulations (if any) from step 3 using the updated DGP
7:	end

Table 2.1: The steps of the proposed algorithm.

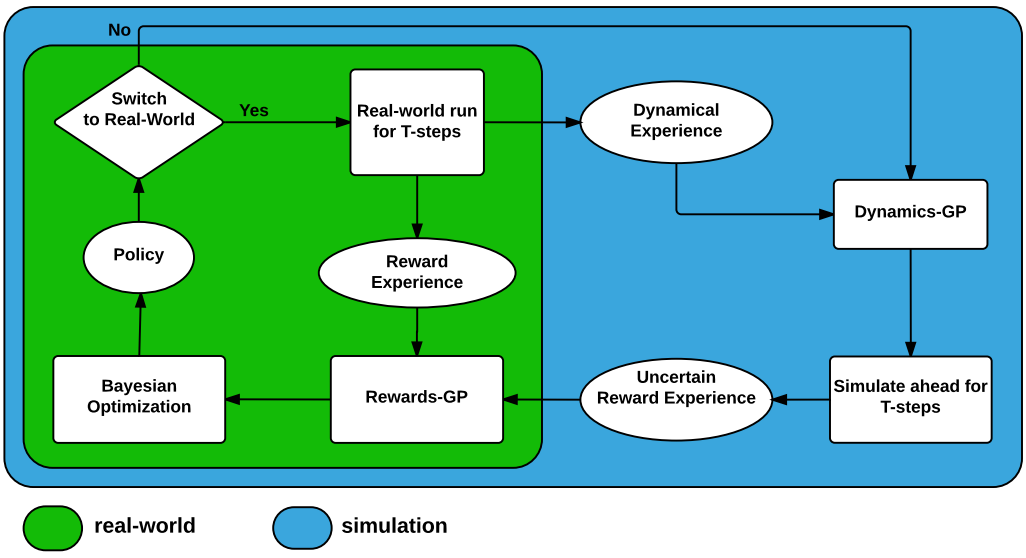


Figure 2.4: The proposed algorithm. Learning can be performed by direct interaction with the real-world and also by learning the dynamical model and planning with the help of simulations.

2.4.1 Finding a New Candidate

This section outlines the steps of finding a new candidate to be tried on the real-world by using the Bayesian optimization search with the help of simulation as discussed in section 2.3. Finding a new candidate consists of a loop that performs a Bayesian optimization search, runs this candidate in simulation, then decides if the candidate should be tried on the real-world. Here simulation is used as the means of predicting the outcome of running the candidate on the real-world, and therefore facilitating the decision making step by updating the measure of how likely it is that this candidate will improve the rewards of $\boldsymbol{\theta}_{\text{best}}$, our current best policy.

In Table 2.2, part **A**, **B** and **C** list the steps of finding a candidate. In part **A**, a number of parameters discussed earlier are initialized that include tol_{EIP} (e.g. $tol_{EIP} = 0.1$) , $promising = 0$, i_{max} (e.g. $i_{max} = 5$), and $scope = 1$. tol_{EIP} is a parameter that defines the promising regions, i.e. if EIP value of a candidate in equation (2.8) is greater than tol_{EIP}, the value of $promising$ is set to 1. i_{max} is the maximum number of failed attempts of finding a candidate in a promising region, which signals the need for re-defining the search scope by reducing $scope$. In part **B**, first a Bayesian optimization search is performed that offers a candidate policy. After running this candidate policy in simulation and updating the RGP, the corresponding EIP value is calculated. In part **C**, the EIP value is used to make decisions. As mentioned earlier, If this value is greater than tol_{EIP} a promising region has been found, $promising$ is set to 1, the value of tol_{EIP} is increased to the current EIP value, and the search for a candidate is continued in simulation. This is continued until the candidate found exits the promising region, i.e. its EIP drops below the current tol_{EIP} value. The last candidate that has the highest EIP value is selected and is executed on the real-world. If the EIP values of candidates offered by the Bayesian optimization are less than tol_{EIP} for i_{max} number of times, the scope of the search is changed by reducing the value of $scope$ (e.g. $scope = \frac{1}{2}scope$), and the new search region is reduced according to equation (2.9). This is followed by changing the definition of a promising region, which is done by reducing the value of tol_{EIP} (e.g. $tol_{EIP} = \frac{1}{2}tol_{EIP}$). This is continued until either a promising region is found, or convergence is met.

2.5 Reward Function

In classical RL it is common to assign a negative or no reward to the states everywhere except at the states at the target. In such a setting where the target location is unknown to the agent, the search is random until the target is visited for the first time. It is only then that the algorithm can improve upon the future rewards. Although this works for easier problems, it is highly data-inefficient. The method used here is to

Finding a Candidate

A. Initialization:

1: initialize tol_{EIP}, the min value of EIP that correspond to promising region (e.g. 0.1 which is 10% improvement over the current max)

2: set $promising = 0$ (not in promising region initially)

3: set i_{max} (e.g. $i_{max} = 5$), max # of failed searches that don't result in promising regions

4: set $scope = 1$, search bounds are initially at the limit, i.e. $[L_{\text{bounds}}, U_{\text{bounds}}]$

B. Bayesian optimization Search and Update (section 2.3)

1: perform Bayesian optimization, find candidate policy $\boldsymbol{C}$ with corresponding $\{V_{C,R}, \sigma^2_{C,R}\}$

2: perform simulation (using DGP) with $\boldsymbol{C}$, get $\{V_{C,D}, \sigma^2_{C,D}\}$

3: update RGP simulation, get $\{V_C, \sigma^2_C\}$ (equations in (2.4), or equations (2.6) and (2.7))

4: get EIP_C with $\{V_C, \sigma^2_C\}$ (equation (2.8))

5: re-train RGP

C. Decision Making

1: if $EIP_C > tol_{EIP}$

2: promising region is found, set $promising = 1$ if not set already

3: increase tol_{EIP} to EIP_C

4: keep searching by going back to **B**

5: if $EIP_C < tol_{EIP}$ && $promising == 1$

6: search leaving promising region, Exit, apply last candidate on real-world (line 4 in Table 2.1)

7: if $EIP_C < tol_{EIP}$ && $promising == 0$

8: still in search of promising region, see how many times this has happened

9: if $i < i_{max}$

10: $i++$

11: elseif $i >= i_{max}$

12: if $tol_{EIP} \to 0$ && $\Delta\boldsymbol{\theta}_{\text{relative}} \to 0$ (within a threshold, e.g. 1e-6)

13: set $convergence = 1$, Exit the algorithm.

14: reset $i = 0$

15: decrease scope of the search by decreasing $scope$ (e.g. $\frac{1}{2}scope$)

16: decrease tol_{EIP} (e.g. $tol_{EIP} = \frac{1}{2}tol_{EIP}$)

17: continue looking for promising region by going back to **B**

Table 2.2: The steps of a search for a candidate.

construct the reward function as a mixture of M unnormalized Gaussians centered at a known target state as

$$r(\boldsymbol{x}) = \sum_{i=1}^{M} a_i \mathcal{N}_{un}(\boldsymbol{x} \mid \boldsymbol{x}_{target}, \boldsymbol{\Sigma}_i) \tag{2.11}$$

where the diagonal matrix $\boldsymbol{\Sigma}$ is chosen by the user. The diagonal terms of $\boldsymbol{\Sigma}$ are the variances around the target for each state and are chosen to be small numbers such that states far away from the targets receive almost no rewards. This essentially only gives high rewards to points that are close to the target.

2.6 Policy Approximator

As given in equation (1.2), a non-linear policy function maps the states to actions given some policy parameters. A GP named *policy-GP* was chosen to represent this function. Similar to [46], n_p artificial *support points* are chosen as the training input and output data, which along with the hyperparameters of the policy-GP (chosen by user), govern the behavior of the output. The support input states are $\boldsymbol{X}_p = \{\boldsymbol{x}_1 \dots \boldsymbol{x}_{n_p}\}$, where each element in $\boldsymbol{x}_p$ is in $[-1, 1]$, and the corresponding support output actions are $\boldsymbol{U}_p = \{u_1 \dots u_{np}\}$, where $-1 < u_p < 1$. The parameters of the policy-GP are then

$$\boldsymbol{\theta} = \{\boldsymbol{X}_p, \boldsymbol{U}_p\} \tag{2.12}$$

and a set of hyperparameters of the policy-GP chosen by the user. For a given state, the deterministic feedback policy is

$$u(\boldsymbol{x}_{\text{normalized}}) = \boldsymbol{K}(\boldsymbol{x}_{\text{normalized}}, \boldsymbol{X}_p)(\boldsymbol{K}(\boldsymbol{X}_p, \boldsymbol{X}_p) + \sigma^2 \boldsymbol{I}_{n_p})^{-1} \boldsymbol{U}_p \tag{2.13}$$

where $\boldsymbol{K}$ is the SE covariance function, and $\boldsymbol{x}_{\text{normalized}}$ is the result of the normalization of the input state $\boldsymbol{x}$ by its bounds. For problems that define the permissible region of action between the value of $[u_{\min}, u_{\max}]$, the output of the policy-GP is passed though a sigmoid function as

$$u_{\text{out}} = \frac{u_{\max} - u_{\min}}{1 + \exp\left(-b\ u_{\text{in}}\right)} + u_{\min} \tag{2.14}$$

where u_{in} is the output of the policy-GP, and $b > 0$ is a parameter to be chosen. With the support output points always being in the region of $[-1, 1]$, the value of b is chosen to be 0.4, which in a case of $u_{\min} = -10$ and $u_{\max} = 10$ produces u_{out} in Figure 2.5. During the Bayesian optimization step, a set of policy parameters are found by moving the support input and output points around in a way that the actions, or the outputs

of the policy-GP, maximize the total reward over a finite horizon time.

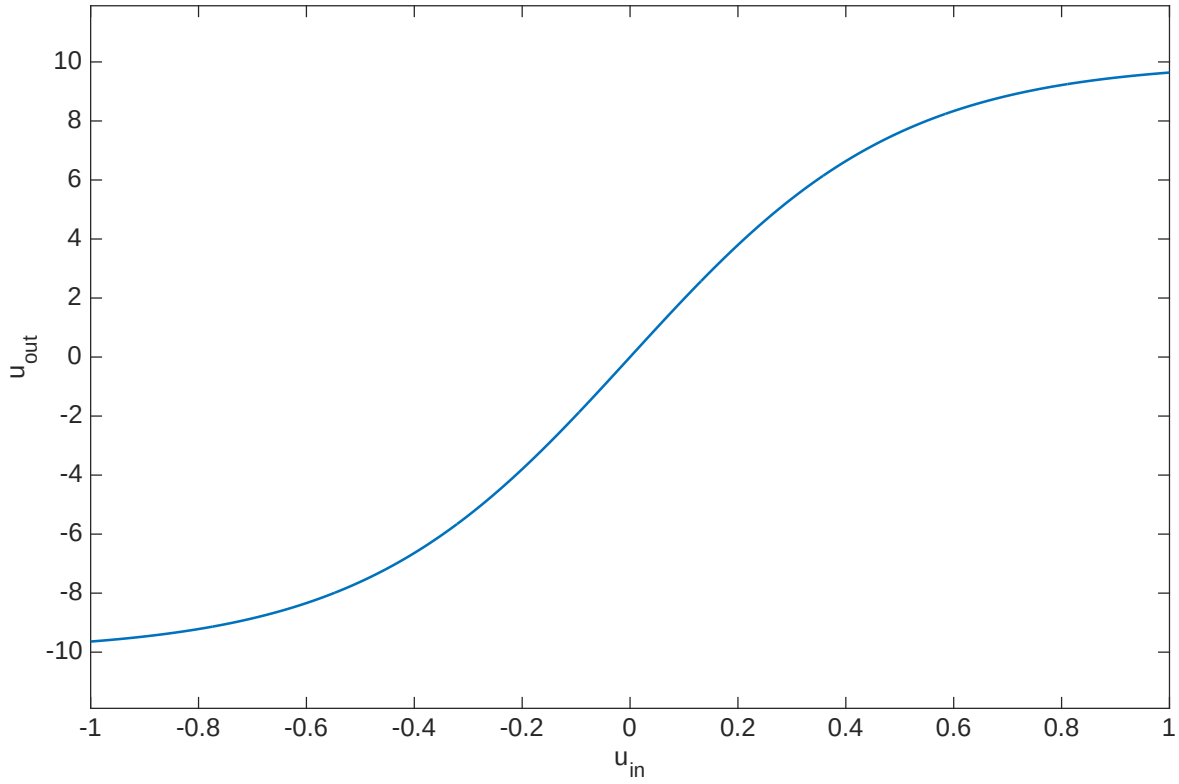


Figure 2.5: Sigmoid function

Chapter 3

Numerical Results

To evaluate the performance of the proposed algorithm two widely known problems, namely the mountain car problem and the inverted pendulum problem, are considered. The real-world is replaced by a numerical ordinary differential equation (ODE) integration[1] that use the equations of motions listed in section A.3; although this is often called simulation, we continue to use the term "real-world" to refer to using the ODEs, and use the term "simulation" only for predicting ahead with the use of the DGP as described in section 2.1.

The following two sections briefly describe each of the two problems and their variations found in the literature. The proposed algorithm is then used in two different approaches to solve these problems. The first approach is to find a set of actions, $\boldsymbol{\theta} = \{u_1 \dots u_T\}$, that are applied one after the next that would bring the final state to the target. In the second approach, we find the parameters of a feedback control policy $\boldsymbol{\theta} = \{\boldsymbol{X}p, \boldsymbol{U}p\}$ (the hyperparameters are selected in advance and are listed shortly) that are used with the policy approximator in section 2.6 to find a set of actions applied at each time step to reach the target state. In both approaches, *zero-order-hold* control is used, where each discrete-time action is applied continuously to the system for a known time interval of t_d seconds. The upper bound on the training size of the DGP for both cases in set to 500 points, and the λ parameter used in the DP-means algorithm for clustering the dynamical data (if larger than 500) into different bins is set to 0.1.

The one-step reward function used in solving the two problems is chosen as a mixture of 20 unnormalized multivariate Gaussians centered at the target in the form of

[1] The Runge-Kutta method is used. This corresponds to ODE45 function in Matlab

$$r(\boldsymbol{x}_t) = \sum_{i=1}^{20} a_i \mathcal{N}_{\text{un}}(\boldsymbol{x}_t \mid \boldsymbol{x}_{target}, \boldsymbol{\Sigma}_i) \tag{3.1}$$

where the i^{th} weight is $a_i = \sum_{j=1}^{i} j$, and the covariance matrix $\boldsymbol{\Sigma}_i = \sigma_i^2 \boldsymbol{I}$ is diagonal, where $\sigma_1^2 \dots \sigma_{20}^2$ are placed logarithmically between 0.6 and 0.02. Unlike a single Gaussian function (or a similar looking function such as the t-distribution), the produced function has both a wide tail and a sharp peak at the center, which gives little reward to states that are far away from the target, and progressively increase the reward as the states near the target. Using this function results in increasingly higher values of EIP for policies that produce trajectories that are near the target, hence encouraging the Bayesian optimization to choose these policies. The one-step reward function in a single dimension with the target at zero is shown in Figure 3.1.

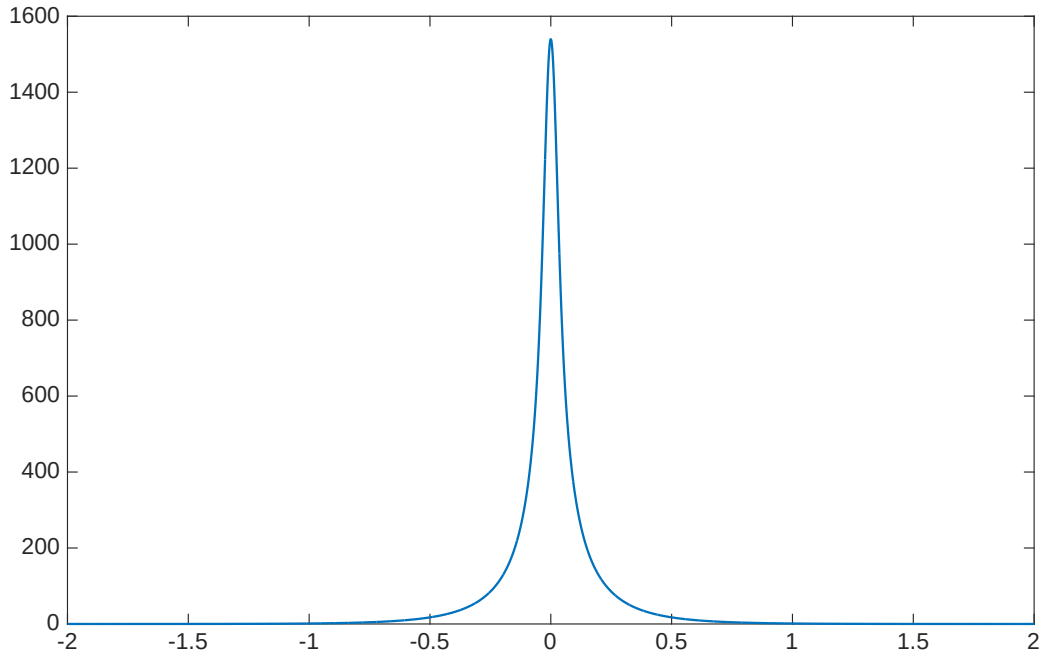


Figure 3.1: The one-step reward function in equation (3.1) in a single dimension.

3.1 Mountain Car Problem

The mountain car is a well-known problem in machine learning and its earliest form was used in [56]. As demonstrated in Figure 3.2 a car is positioned between a valley and the goal is to bring the car to the top of

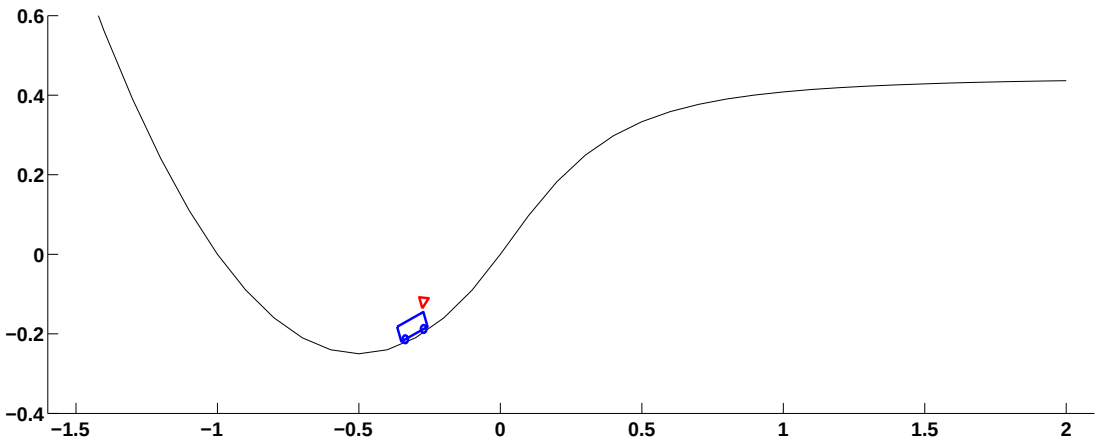


Figure 3.2: The mountain car problem

the hill. The car does not have enough power by itself to simply drive up the hill (constrained action space) and it has to swing back and forth to create enough momentum to come up the valley. There are different variations of this problem in the literature, some with bang-bang control, and others with discrete action and state [1]. Here we consider continuous action and state space. In the literature success is often defined as simply getting the car over the hill at any velocity, although here we adopt a more difficult version of the problem similar to the work in [44], where the goal is to park the car on the hill. Since the car does not have a hand-brake, keeping the car parked at the target requires applying actions continuously. The two states are $\boldsymbol{x} = (x, \dot{x})$, the action is bounded by $u = [-4, 4]$, and the velocity is bounded by $\dot{x} = [-2\frac{m}{s}, 2\frac{m}{s}]$. The equations of the motion of the mountain car problem are given in section A.3. The car is initially at the bottom of the hill at $(x, \dot{x}) = (-0.5, 0)$, and the target state is chosen to be at $(x, \dot{x}) = (1, 0)$. We define the target region as $.95 < x < 1.05$ and $|\dot{x}| < 0.05$.

The early tabular RL algorithms that were applied to this problem were mostly interested in bringing the car over the hill, with discretized states and actions. These model-free approaches were extremely data inefficient. [38] reports the performance of the tabular Q-learning [32] that took 300,000 iteration, each iteration taking a maximum number of 300 steps, with each step of 0.05 seconds. The direct RL algorithms with the use of function approximators were generally more data-efficient on this problem; [38] employed an algorithm named *Neural Fitted Q Iteration* that used a Neural network to approximate the Q-function, i.e. the mapping of state-actions to future rewards, and was able to find a policy that brings the car over the top of the hill (no parking) after 139 seconds of interactions. This and other similar direct RL methods often assume that it is possible to start a real-world execution at any random initial state, making it more likely

that states near the target are visited. In [44], a GP was used to represent a value function, i.e. mapping of states to future rewards, with the objective of parking the car between $0.5 \leq x \leq 0.7$, with $-0.1 \leq \dot{x} \leq 0.1$. It was assumed that the state-action space and the subsequent states could be sampled uniformly before the algorithm began, and a total of 50 states-actions samples, and their subsequent states after applying each action for 0.3 seconds were used to represent the dynamical model by a GP. The learning could then be done in a very similar fashion to the dynamic programming that would provide the algorithm with a prior probabilistic knowledge of the dynamics. Part of the challenge in learning however, is discovering interesting regions of the dynamical state-space near the target when policies that take the states there are not available. Here we assume no prior knowledge of the dynamics, and that sampling the dynamical state-space is not possible and experience is only gathered by running available policies. In the next two sections, we present two different ways of solving the mountain car problem starting from the initial state of $\boldsymbol{x} = (-0.5, 0)$.

3.1.1 Finding A Set of Controls for The Mountain Car Problem

Here we find a set of action/controls $\boldsymbol{\theta} = \{u_1 \dots u_T\}$ that bring the car in the target region in T steps. The number of steps here are selected based on the suggestion of [22], which states that the performance of the Bayesian optimization technique degrades when the number of parameters is above 15-20. We therefore pick $T = 20$ steps. In order to to bring the car to the top of the mountain in only 20 steps, and still have the ability to end in the target region, the time-steps t_d at which the zero-order-hold controls are applied are chosen to be longer at the beginning of the finite horizon time, and to decrease logarithmically towards the end. This gives the ability of applying more control as the trajectory nears the target towards the end of a finite horizon time. For the mountain car problem, the time steps are chosen to decrease logarithmically from $t_d = 0.2$ to $t_d = 0.1$ in 20 steps[2] that correspond to a total of 2.8876 seconds of real-world execution time. The change in the time-steps as explained above means that for a given state-action pair, different time steps produce different subsequent states. The time-steps corresponding to state-action pairs are therefore included as observations of the DGP, which means that one training sets for the DGP (see section 2.1) is in the form of $\{(\boldsymbol{x}, u, t_d), \boldsymbol{x}'\}$.

It is also desired that the last state of the produced trajectory is the closest to the target. This can be incorporated in the reward function by progressively giving more weights to the rewards of the states towards the end of the trajectory in the form of

[2] This is equal to logspace(-.699, -1, 20) in Matlab

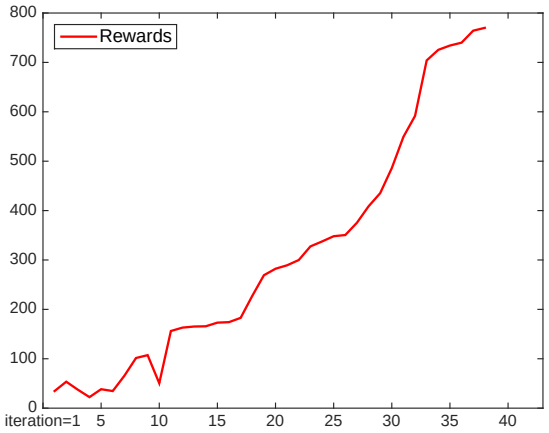
$$r_{total}(\boldsymbol{x}_{1:T}) = \frac{1}{\sum_{t=1}^{T} \exp(0.5t)} \sum_{t=1}^{T} \exp(0.5t) r(\boldsymbol{x}_t) \tag{3.2}$$

where r is a one-step reward function given in equation (3.1).

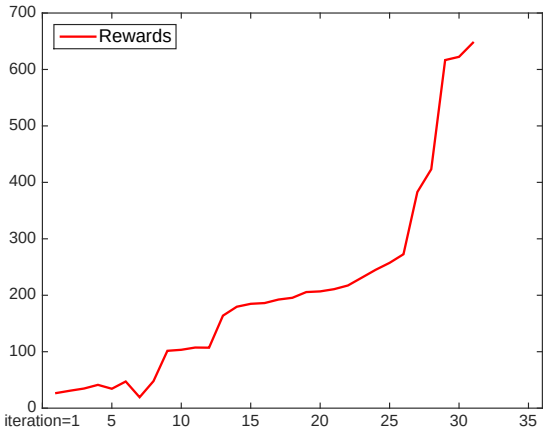
To start the algorithm, five random real-world executions are performed, where the actions are drawn randomly from $u = [-4, 4]$. In Figure 3.3 to Figure 3.5, the results of 6 sample runs of the algorithm are given that show the progression of the total rewards at each real-world execution, along with the best set of actions found and the resulting state trajectory from the initial state to the target region. Each run is terminated when the final state reaches the target region. To get the momentum to go up the hill, the car first swings to the right in some of the runs, and in others to the left first. The end state of each run is given in Table 3.1.

Table 3.1: The final states for 6 different runs of the algorithm on the mountain car problem in Figure 3.3 to Figure 3.5

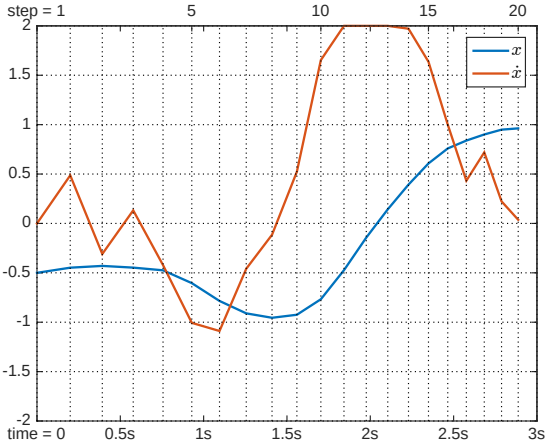
Run	**Position**	Velocity
Run 1	0.9623	0.0371
Run 2	0.9695	0.0439
Run 3	0.9714	0.0461
Run 4	1.0281	-0.0363
Run 5	0.9563	0.0057
Run 6	0.9591	0.0400



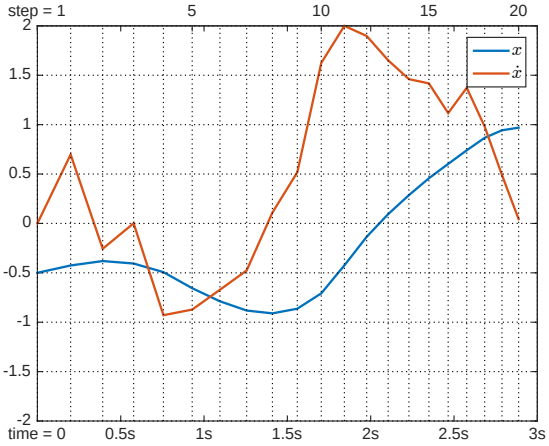
(a) Run 1 rewards at each iteration



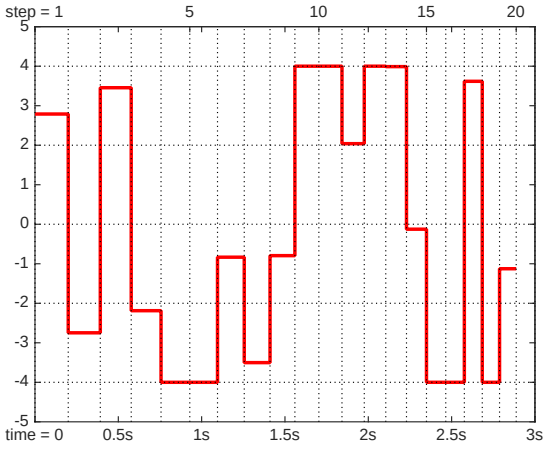
(b) Run 2 rewards at each iteration



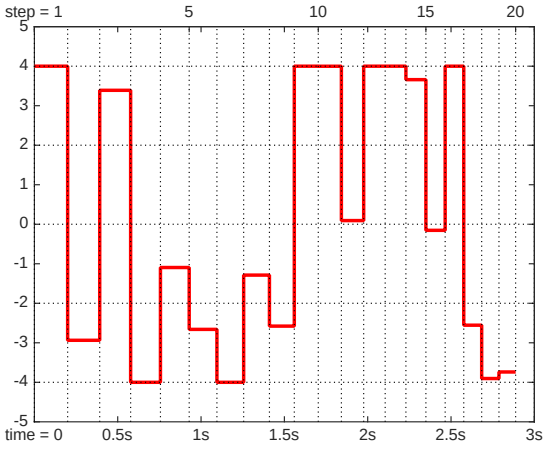
(c) Run 1 state trajectories



(d) Run 2 state trajectories

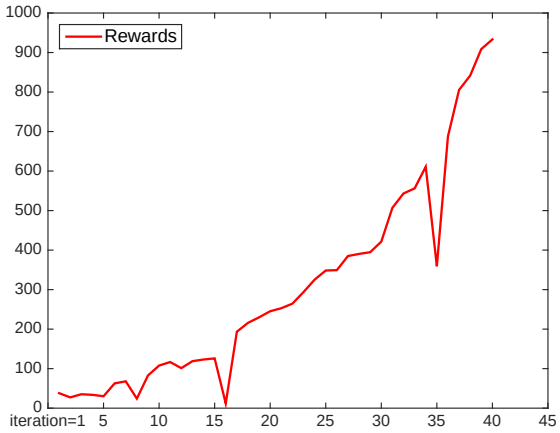


(e) Run 1 actions at each step

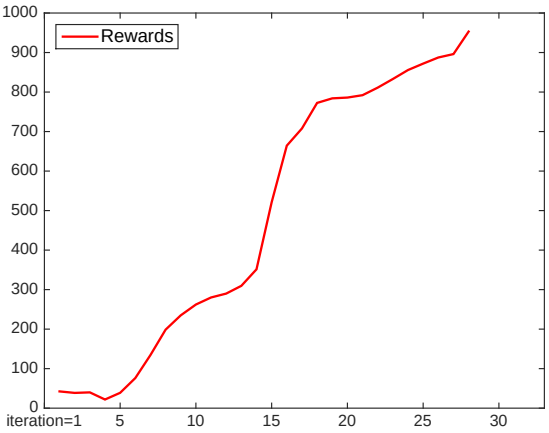


(f) Run 2 actions at each step

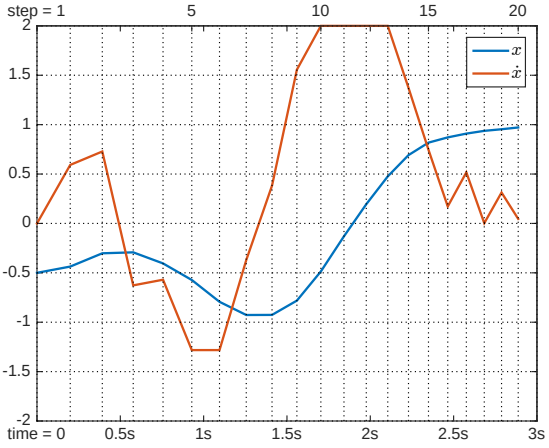
Figure 3.3: Total observed rewards of the execution of candidates on the real-world (top), the best set of actions found (bottom) and the produces trajectories from the best set of actions (middle) in run 1 and 2.



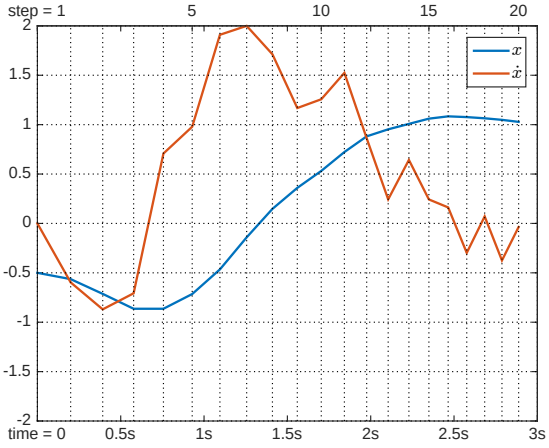
(a) Run 3 rewards at each iteration



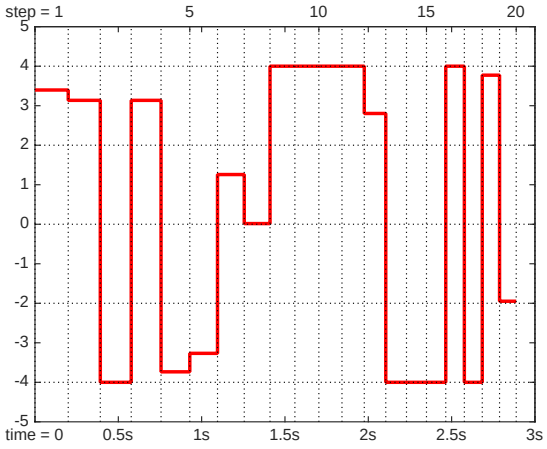
(b) Run 4 rewards at each iteration



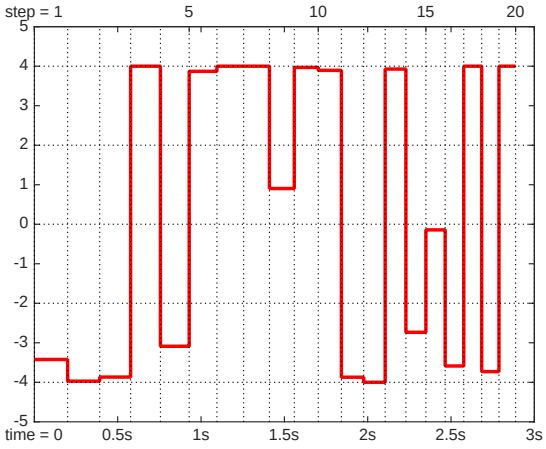
(c) Run 3 state trajectories



(d) Run 4 state trajectories

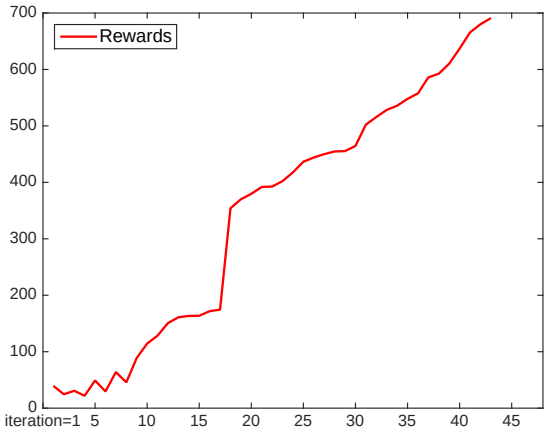


(e) Run 3 actions at each step

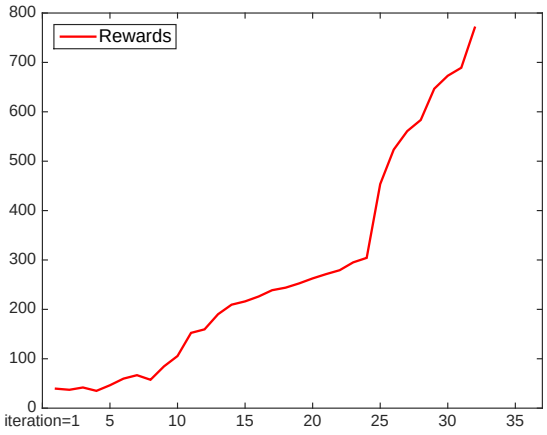


(f) Run 4 actions at each step

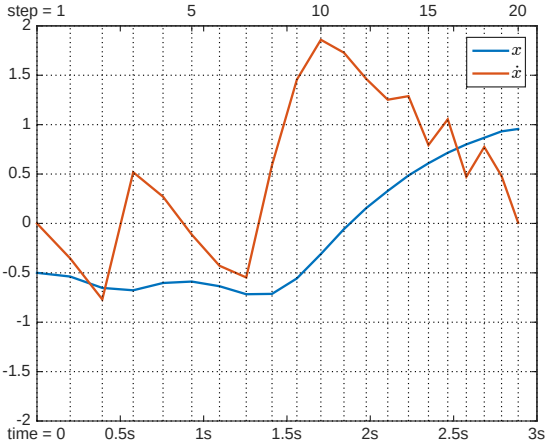
Figure 3.4: Total observed rewards of the execution of candidates on the real-world (top), the best set of actions found (bottom) and the produces trajectories from the best set of actions (middle) in run 3 and 4.



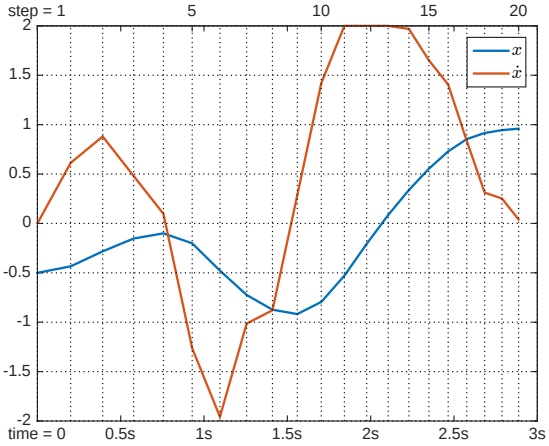
(a) Run 5 rewards at each iteration



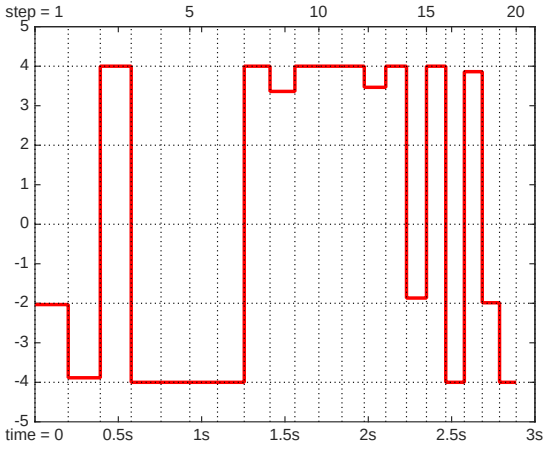
(b) Run 6 rewards at each iteration



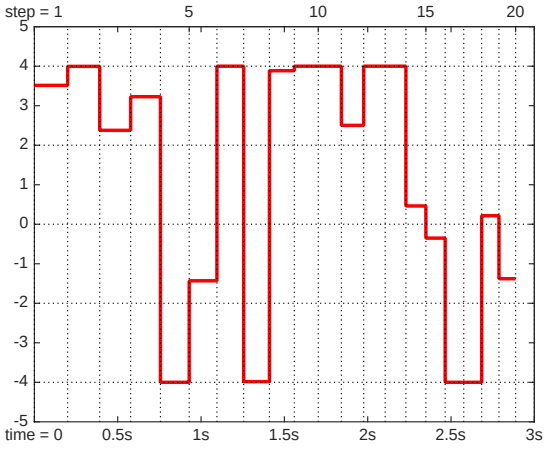
(c) Run 5 state trajectories



(d) Run 6 state trajectories



(e) Run 5 actions at each step



(f) Run 6 actions at each step

Figure 3.5: Total observed rewards of the execution of candidates on the real-world (top), the best set of actions found (bottom) and the produces trajectories from the best set of actions (middle) in run 5 and 6.

The proposed algorithm consistently finds a set of controls that bring the car into the target region. Figure 3.6 show the number of iterations and the real-world interaction time in seconds that is required for 10 different runs/trials (the first 6 correspond to Figure 3.3 to Figure 3.5) to reach the target after the initial 5 random iterations. It takes an average of 35 iterations for the proposed algorithm to find a solution. As mentioned before, each real-world execution amounts to 2.8876 seconds, which means that on average about 101 seconds is needed to find a successful set of actions to park the car in the target region near the $x = 1$ position.

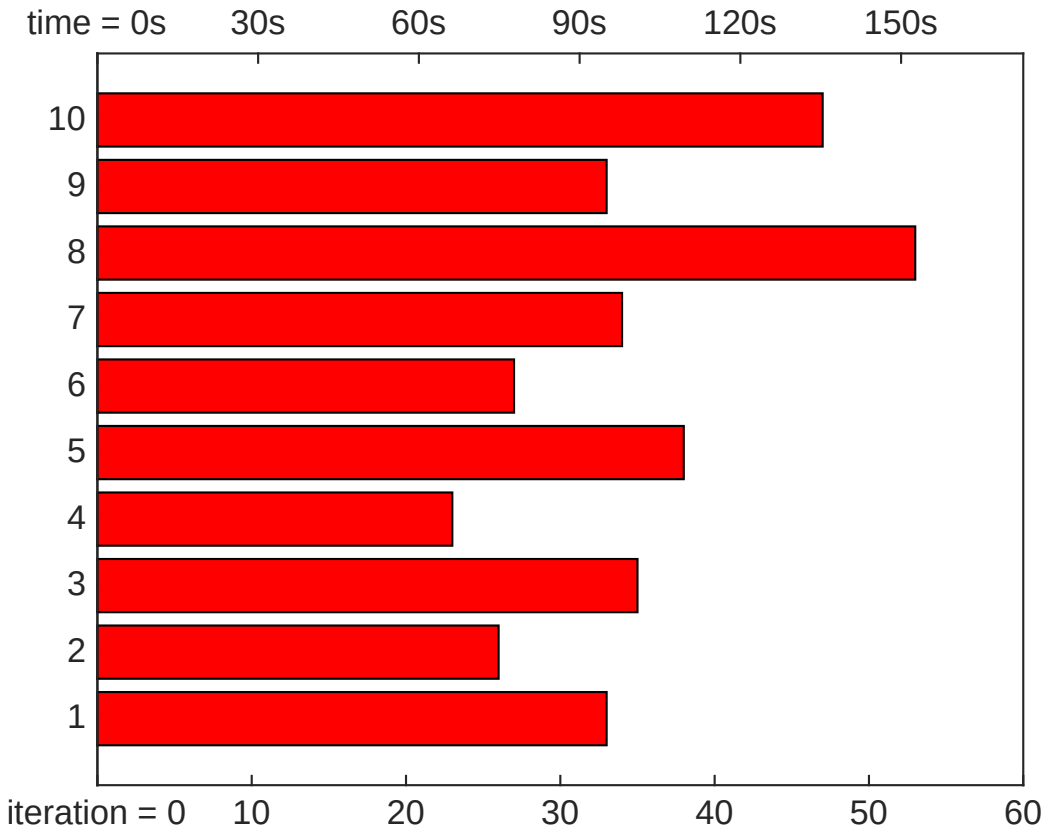


Figure 3.6: The number of iterations and time in seconds that it takes the proposed algorithm to find a set of controls for the mountain car problem in 10 different runs.

In the proposed method, the expected improvements of the candidates that are offered by the Bayesian optimization step are updated by simulation runs, that result in the execution of only the candidates that are highly likely to improve on the current maximum reward. This greatly reduces the number of candidate executions on the real-world. Figure 3.7 demonstrates this reduction by showing the reward of all the 280 candidates offered by the Bayesian optimization (blue lines) in run 1 of Figure 3.3. Amongst these candidates, only the 33 red dots (not counting the first random 5) were tried on the real-world until the target was reached (the red dots correspond to Figure 3.3 (a)). Based on all the 10 runs, the proposed method

reduces the amount of real-world interaction on average by a factor of about 9 times.

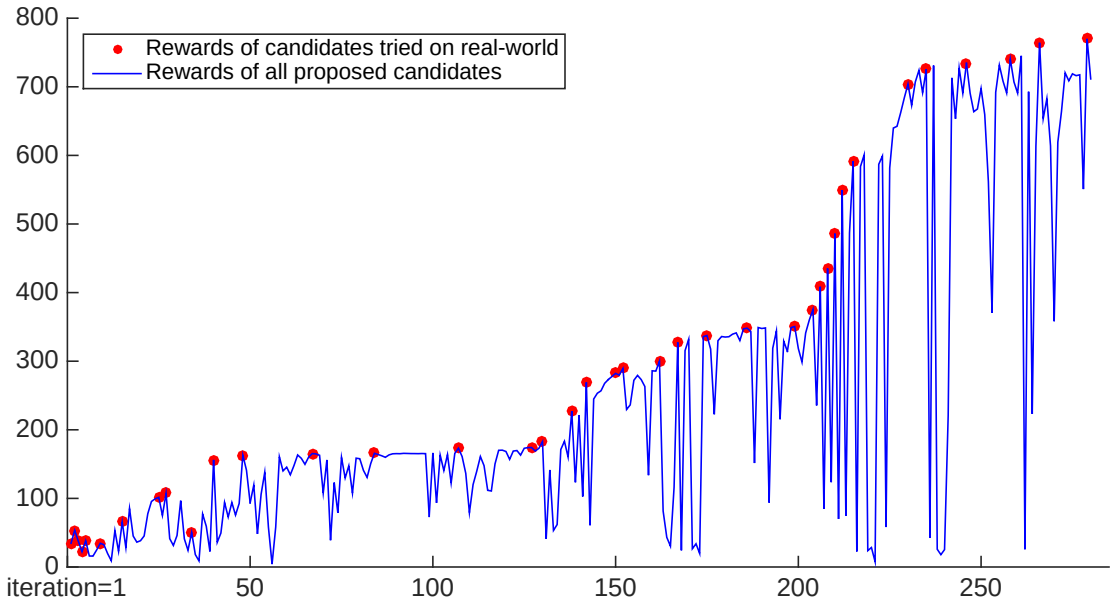


Figure 3.7: The observed rewards of the 33 candidates tried on the real-world (red dots), and the real-world rewards that would have been observed if all the 280 candidates that were tried in simulation were executed on the real-world (blue lines) for the mountain car problem. This plot corresponds to run 1 in Figure 3.3.

3.1.2 Finding a Feedback Policy for The Mountain Car Problem

In the second approach, a feedback policy that maps the states to the actions is found. A feedback policy generally requires more parameters than the limit of the Bayesian optimization as suggested by [22] (20 parameters), and is used here to test the performance of the proposed algorithm when the dimension of the parameters is increased. Using the policy-GP in section 2.6, the objective is to find a set of parameters $\boldsymbol{\theta} = \{\boldsymbol{X}_p, \boldsymbol{U}_p\}$, where $\boldsymbol{X}_p = \{\boldsymbol{x}_{p1}, \ldots, \boldsymbol{x}_{pn_p}\}$ and $\boldsymbol{U} = \{u_{p1}, \ldots, u_{pn_p}\}$ are n_p pairs of support inputs and outputs, that bring the car in the target region. The hyperparameters of the policy-GP are picked in advance and consist of the length-scales, which are all taken to be 1, small number for noise $\sigma^2 = 1^{-6}$ to ensure the covariance matrix of the policy-GP is positive-definite, and the signal variance[3] $\alpha^2 = 1$. To solve the mountain car problem, we pick 25 support points, i.e. $n_p = 25$, which means that $\boldsymbol{\theta}$ comprises of 75 parameters. As stated in section 2.6, the support input points in $\boldsymbol{X}_p$, and the support output points in $\boldsymbol{U}_p$ are between $[-1, 1]$. The input states to the policy-GP are therefore normalized to fall within this range, this is done by dividing the velocities by the velocity by 2 since $\dot{x} \in [-2, 2]$. Although no bounds are placed

[3]notice that the choice of the signal variance α^2 doesn't effect the mean of policy-GP

on the position, it is reasonable to assume that control is desired only in the region of $x \in [-1.5, 1.5]$ since the target is at $x = 1$, so the positions of the input states to the policy-GP are divided by 1.5.

Using the feedback policy, the goal is to bring the car into the target region and to keep it there for an extended finite-horizon time. A successful policy is therefore defined as one that during the learning phase brings the car into the target region in T_1 steps, and is then able to maintain it there for an additional number of steps T_2. Keeping the car in the target region for as long as possible suggests giving equal weights for the reward at each step. The total reward function is defined as

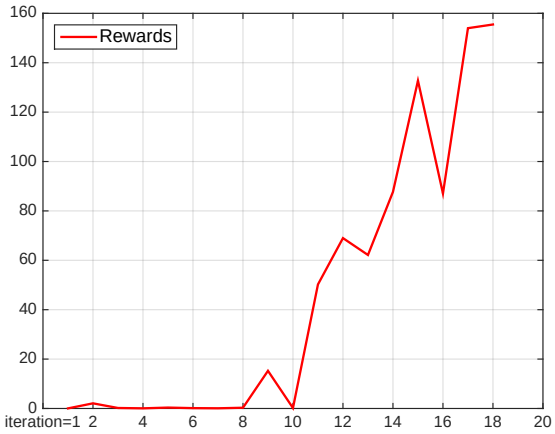
$$r_{total}(\boldsymbol{x}_{1:T_1}) = \frac{1}{T_1} \sum_{t=1}^{T_1} r(\boldsymbol{x}_t) \tag{3.3}$$

where $\boldsymbol{x} = (x, \dot{x})$. Furthermore the time step t_d is fixed to 0.05 seconds, to allow the same policy found for T_1 steps to be applied for an additional T_2 steps. For the mountain car problem, T_1 is set to 100 steps, which is equal to 5 seconds of interactions for each real-world execution. T_2 is set to 60, which means that successful policies should hold the car in the target region for an additional 3 seconds.

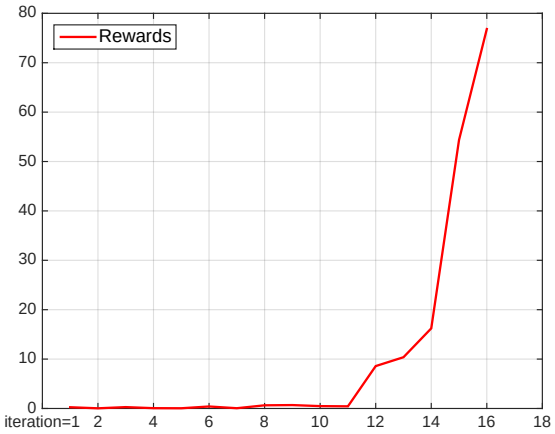
Figure 3.8 to Figure 3.10 show the results of 6 sample runs of finding a feedback control policy, where the first 5 policy are generated randomly. The action and trajectory plots are shown for $T_1 + T_2 = 8$ seconds that demonstrate the success of each policy. Similar to the previous approach, the car gets the momentum it needs for going up the hill by swinging either to the left or right first. Since the car does not have a hand-brake, and position $x = 1$ is at a slope, a constant action of about 0.6 to 0.7 (depending on exactly where the car is held) needs to be applied to maintain the car in the target region.

Despite the increase of the number of parameters from 20 to 75 in this approach, the proposed algorithm can consistently find a successful policy to bring and hold the car in the target region. Figure 3.11 shows the number of iterations and the time that it took for the algorithm to find successful policies for 10 different runs (the first 6 are of the plots above). On average it takes about 17 iterations or real-world tries (not counting the first random 5) to find a successful policy, which amounts to 85 seconds of interactions.

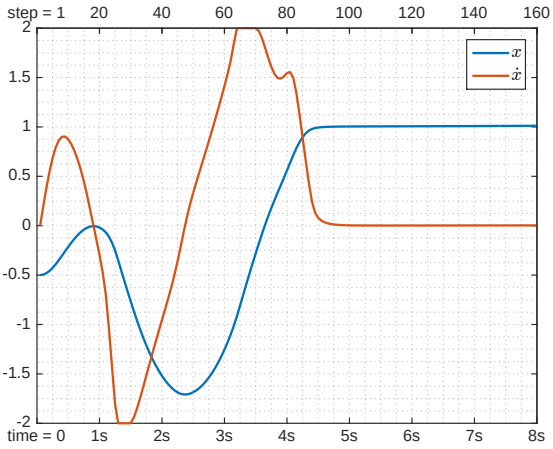
Figure 3.12 showcases the benefit of using simulations, where the blue line is the total rewards for all the 431 candidates offered by the Bayesian optimization step in run 1 of Figure 3.8, which the majority of are carried out in simulation. Excluding the first random 5 policies, only 13 of the candidates shown by red dots were executed on the real-world (the red dots correspond to Figure 3.8 (a)). Based on all the runs,



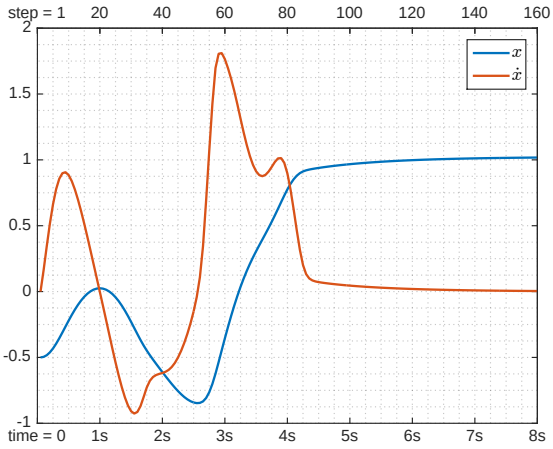
(a) Run 1 rewards at each iteration



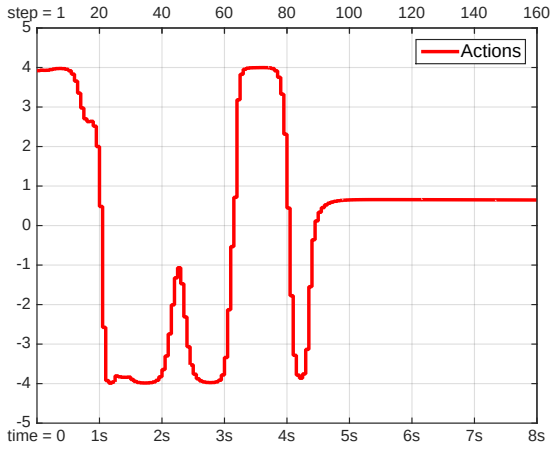
(b) Run 2 rewards at each iteration



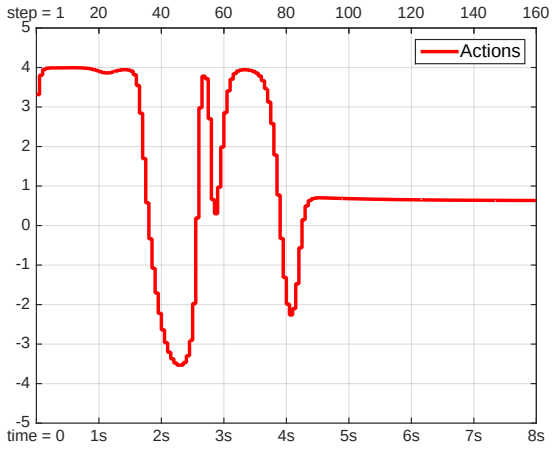
(c) Run 1 state trajectories



(d) Run 2 state trajectories

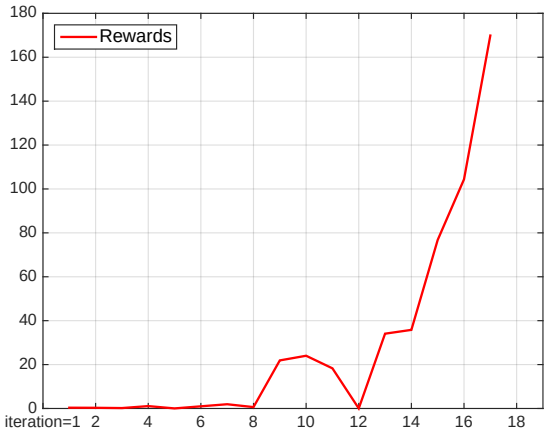


(e) Run 1 actions at each step

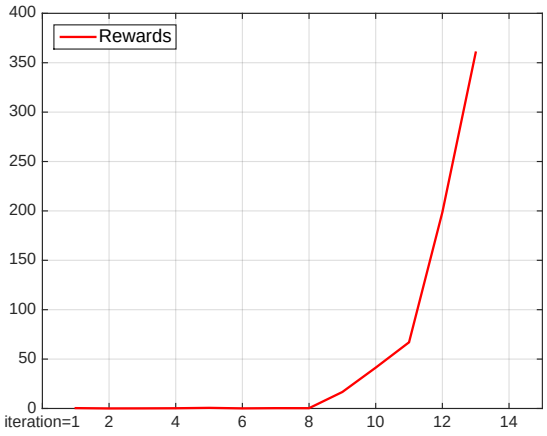


(f) Run 2 actions at each step

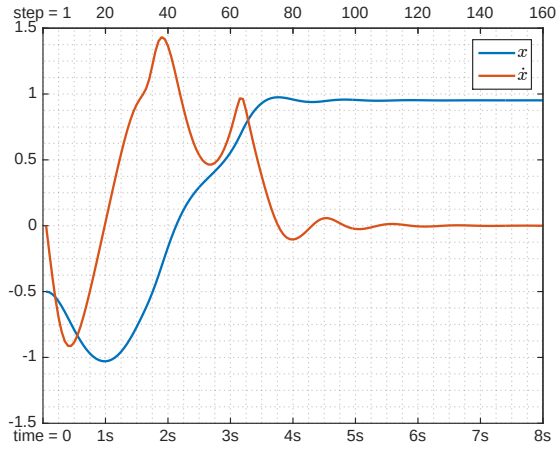
Figure 3.8: Total observed rewards of the execution of candidates on the real-world (top), the actions produced by the best policy for 8 seconds (bottom) and the resulting trajectories for the best policy (middle) in run 1 and 2 of finding a feedback policy for the mountain car problem.



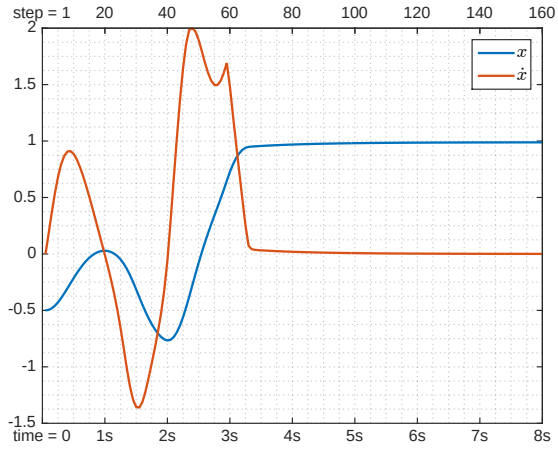
(a) Run 3 rewards at each iteration



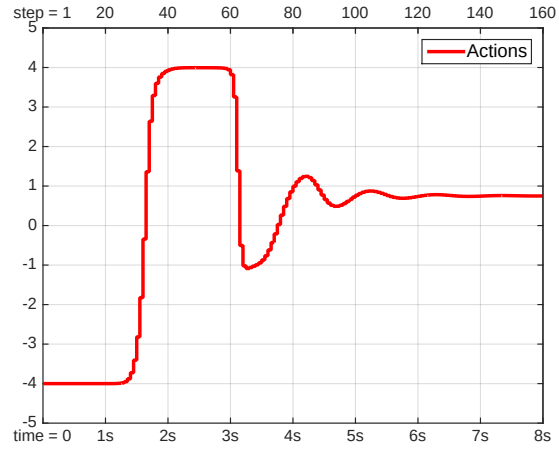
(b) Run 4 rewards at each iteration



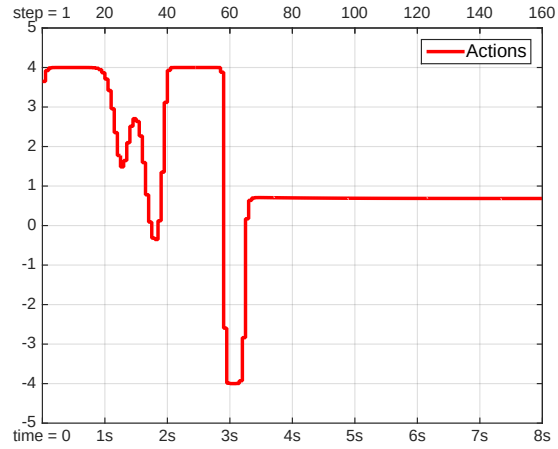
(c) Run 3 state trajectories



(d) Run 4 state trajectories

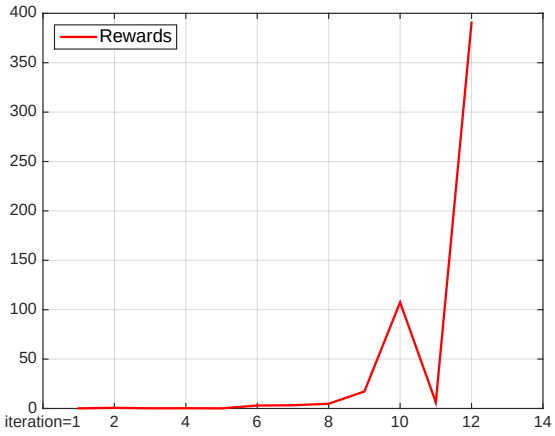


(e) Run 3 actions at each step

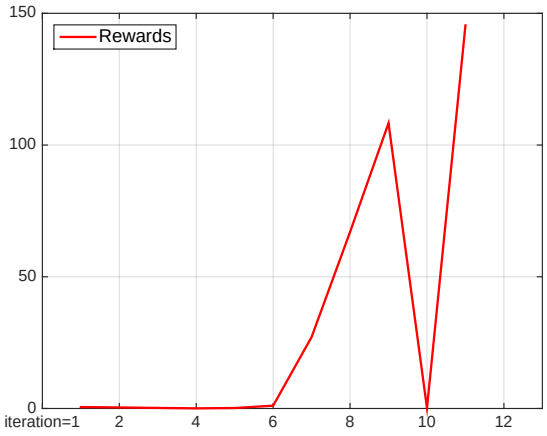


(f) Run 4 actions at each step

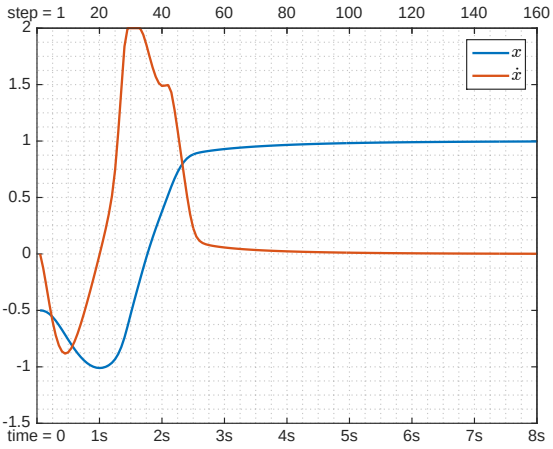
Figure 3.9: Total observed rewards of the execution of candidates on the real-world (top), the actions produced by the best policy for 8 seconds (bottom) and the resulting trajectories for the best policy (middle) in run 3 and 4 of finding a feedback policy for the mountain car problem.



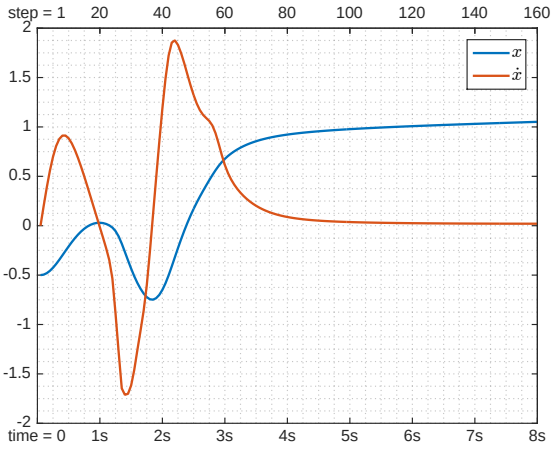
(a) Run 5 rewards at each iteration



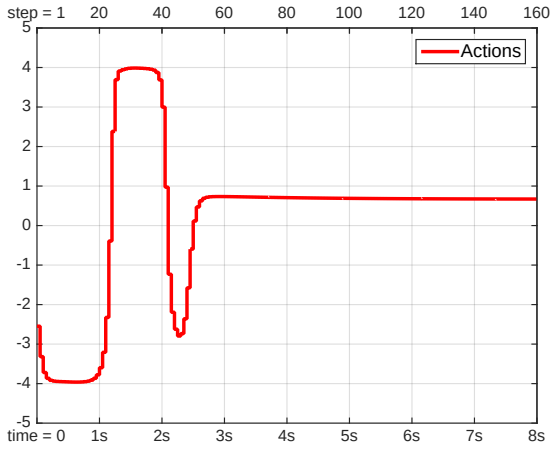
(b) Run 6 rewards at each iteration



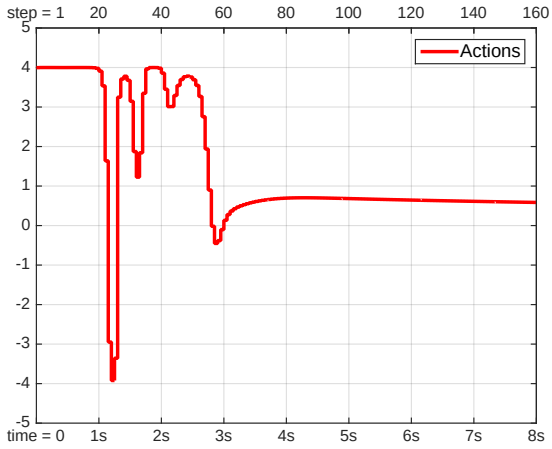
(c) Run 5 state trajectories



(d) Run 6 state trajectories



(e) Run 5 actions at each step



(f) Run 6 actions at each step

Figure 3.10: Total observed rewards of the execution of candidates on the real-world (top), the actions produced by the best policy for 8 seconds (bottom) and the resulting trajectories for the best policy (middle) in run 5 and 6 of finding a feedback policy for the mountain car problem.

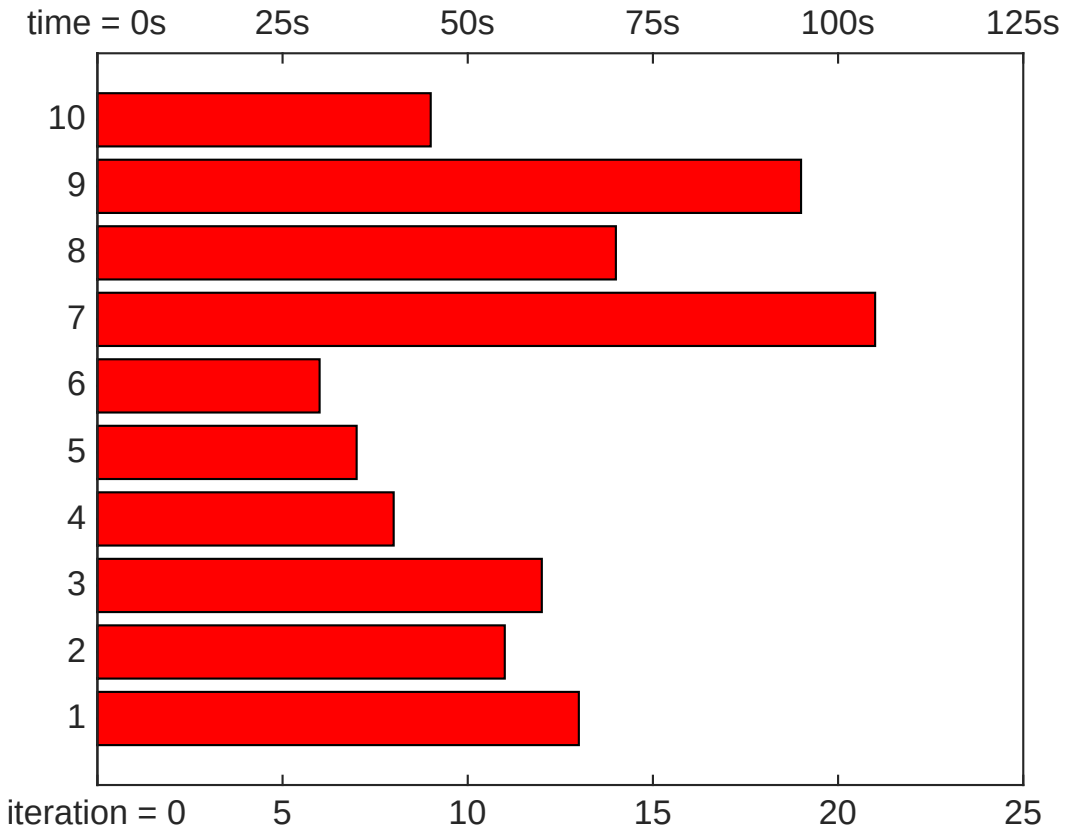


Figure 3.11: The number of iterations and time in seconds that it takes the proposed algorithm to find a feedback policy for the mountain car problem.

the average ratio of candidates tried on simulation to candidates executed on the real-world was about 18 to 1, which means the interaction with the real-world is reduced by a factor of about 18 when a model of the dynamics is learnt. This factor is twice as much as in the last approach, where a set of actions were found. This is because here each real-time execution gathers 100 training points for the DGP as opposed to 20, and furthermore they all have the same time step of $t_d = 0.05$ seconds. In other words, the training points of the dynamical model is not only larger, but each point can be used to predict the next state at any step of the simulation. This gives the ability to carry out more simulations when compared to before, and subsequently it takes less time to learn the task.

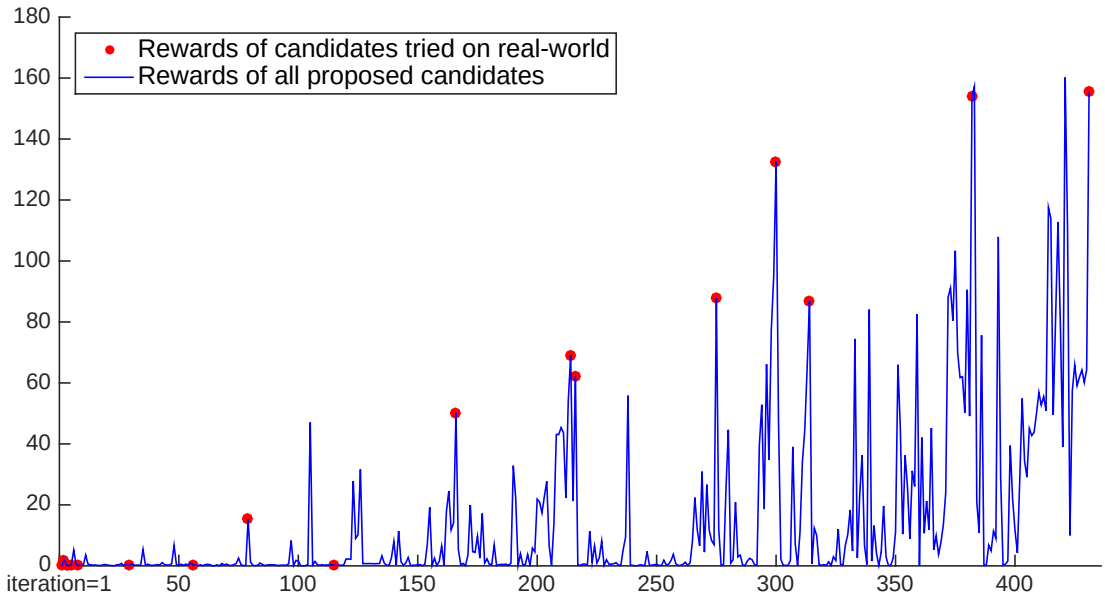


Figure 3.12: The observed rewards of the 13 candidates tried on the real-world (red dots), and the real-world rewards that would have been observed if all the 431 candidates that were tried in simulation were executed on the real-world (blue lines) for the mountain car problem. This plot corresponds to run 1 in Figure 3.8.

3.2 Inverted Pendulum

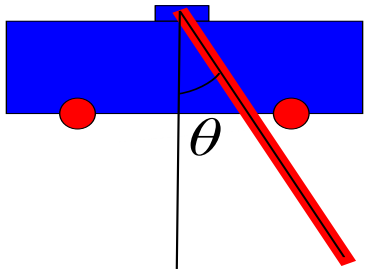


Figure 3.13: The inverted pendulum setup.

The inverted pendulum is another benchmark problem that is often used in reinforcement learning. Figure 3.13 shows the setup of this problem, where a pendulum is attached to a cart and the goal is to swing up and balance the pendulum. There is no actuator attached to the pendulum, therefore the cart has to be moved back and forth in order to swing up and keep the pendulum balanced. Some variations of the problem in the literature start with the pendulum already in the upright position, and others try to swing up the pendulum and keep it balanced, with the latter being a much harder problem. The task considered here is to swing up the pendulum from rest and to keep it balanced, with the velocity of the cart and the angular velocity of the pole approaching zero. The control is in the region of $u = [-10, 10]$, and the states are $\boldsymbol{x} = \left(x, \dot{x}, \dot{\theta}, \sin(\theta), \cos(\theta)\right)$, where the last two states define the angle of the pendulum and the quadrant in which it is in. This avoids issues that arise from only having θ as a state. If θ is used and no bounds are placed on it, different training states that are essentially the same (e.g. $\theta = \frac{\pi}{4}$ and $\theta = 2\pi + \frac{\pi}{4}$) can create different training points for the dynamical model that are seen to be far apart from each other. This results in the DGP failing to detect two neighboring states if their pendulum angles are separated by $2\pi \pm \epsilon$, where ϵ is a small angle. On the other hand, defining θ to be in the region of $[-\pi, \pi]$ (or $[0, 2\pi]$) creates discontinuity which still suffers from the same issue (e.g. $\theta = -\pi - \epsilon$ and $\theta = \pi + \epsilon$ are two similar orientations of the pendulum, but are far apart from each other numerically). By defining the angle of the pendulum by two states of $\sin(\theta)$ and $\cos(\theta)$, these issues are eliminated. The equations of motion of the inverted pendulum problem is given in section A.3.

The inverted pendulum has appeared numerous times in the RL literature. [57] used a combination of Q-learning and local linear controllers to swing-up and balance the pendulum, with the swing-up defined as the pendulum being within 0.133π of the upright angle, and $|\dot{\theta}| < 2\frac{rad}{s}$, with success defined as keeping

the pendulum in that region. This task was completed without learning the dynamical model in $144,000$ seconds. [43] used a radial basis function (RBF) to approximate the value function, with an additional RBF network used for learning the dynamical model and was able to learn the task of bringing and maintaining the pendulum within $\frac{\pi}{4}$ of the upright position in $16,000$ seconds. In [38, 58] the pendulum started at an upright position, and the Neural Fitted Q-iteration algorithm was used to approximate the Q-function which only learnt the task of balancing the pendulum in the region of the pendulum being within $\frac{\pi}{2}$ of the upright position, which took 120 seconds to complete. [41] used a model-based learning algorithm where the dynamical system was modeled by the nonlinear dynamical factor analysis (NDFA) based on the work in [42], and learned the task of swing-up and balance as defined by [57] in 125 seconds. PILCO [46], a model-based algorithm described earlier in section 1.5, offers the best efficiency in terms of real-world interactions, and learns the task of swing-up and balance of the pole based on 17.5 seconds of interaction time.

We adopt the swing-up task definition from [57], which is achieved when the pendulum is within 0.133π of the upright position, and $|\dot{\theta}| < 2\frac{rad}{s}$, regardless of the velocity of the cart. We however place stricter criteria on success by defining a target region of $|\dot{x}| < 0.05\frac{m}{s}$, $|\dot{\theta}| < 0.05\frac{rad}{s}$, and $-1 < \cos(\theta) < -.999$. This means that the pendulum should be within 0.0142π (or 2.5 degrees) of the upright angle. In the case of finding a set of controls for a fixed finite horizon time described next, success is defined as the last state being in the target region, whereas a successful feedback policy is one that is able to keep the states in this target region if the finite-horizon time is extended. The bounds on the velocities are $\dot{x} \in [-5, 5]$ and $\dot{\theta} \in [-4\pi, 4\pi]$. As in the mountain car problem the length-scales of the policy-GP are all set to 1, with $\sigma^2 = 10^{-6}$ and $\alpha^2 = 1$. The upper limit of the number of point of the DGP is set to 500, with $\lambda = 0.1$ in the DP-means algorithm.

3.2.1 Finding A Set of Controls for The Inverted Pendulum Problem

Similar to the mountain car problem, this approach finds a set of 20 actions that bring the inverted pendulum system from the start state, to the target region in 20 steps. The reward function used is the same as in equation (3.2), and the 20 time steps are placed logarithmically between $t_d = 0.1$ and $t_d = 0.05$[4]. The total time for each real-world execution is 1.4472 seconds.

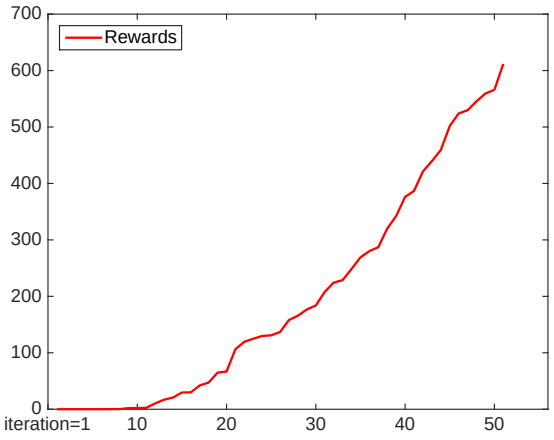
Each run of the algorithm is started with five real-world executions, where the actions are selected randomly from the allowable region of the control $u = [-10, 10]$. The search is then carried out until a successful set of controls are found. Similar to before, Figure 3.14 to Figure 3.16 show the results of 6 runs of the algorithm that include the progression of the total reward at each iteration, the best set of controls found,

[4] this corresponds to logspace(-1,-1.3,20) in Matlab

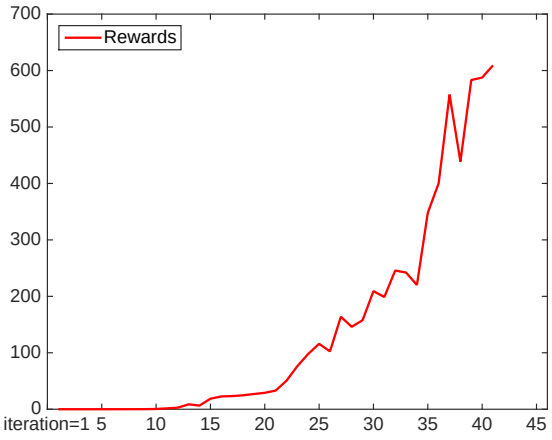
and the trajectory that is resulted from applying the best set of controls on the system. It can be seen in the produced trajectories that to swing up the pendulum, the cart can either move to the right or to the left first. The final state of each of the 6 runs are given in Table 3.2.

Table 3.2: The final states for 6 different runs of the algorithm on the inverted pendulum problem.

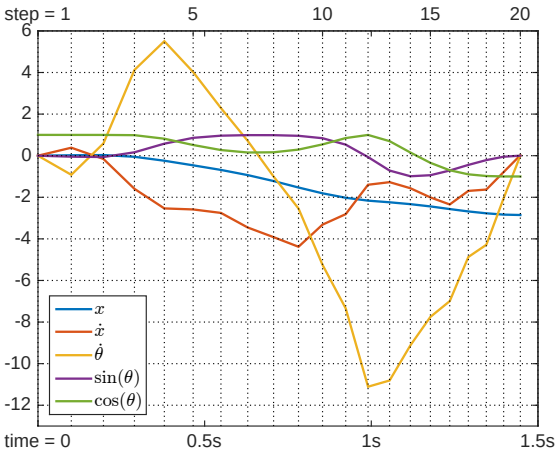
Run	**Position**	Velocity	Ang. Velocity	Sine	Cosine
Run 1	-2.8525	0.0217	-0.0014	0.0008	-1.0000
Run 2	3.6603	-0.0464	0.0492	0.0259	-0.9997
Run 3	-3.3259	0.0169	-0.0113	-0.0134	-0.9999
Run 4	3.3024	0.0005	0.0060	0.0060	-1.0000
Run 5	2.7843	0.0406	0.0469	0.0057	-1.0000
Run 6	2.4960	0.0055	0.0302	0.0128	-0.9999



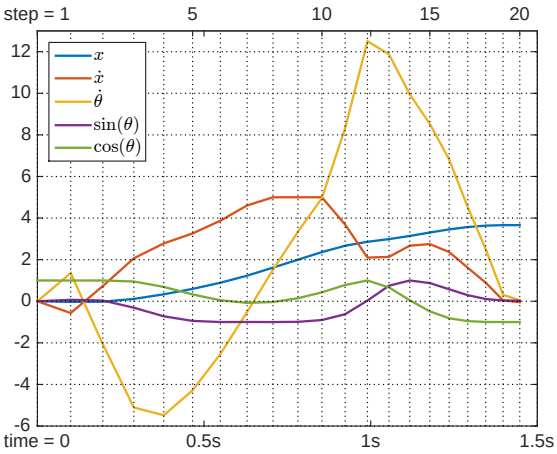
(a) Run 1 rewards at each iteration



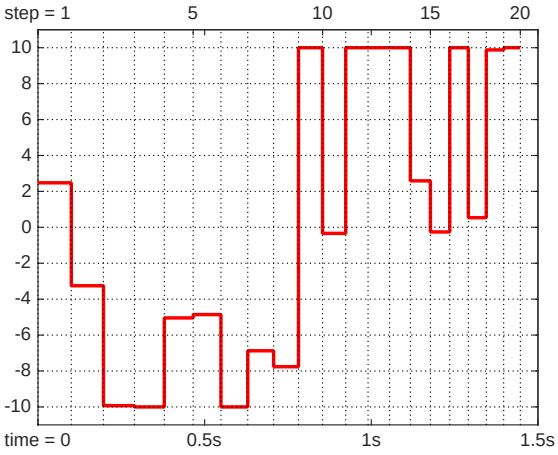
(b) Run 2 rewards at each iteration



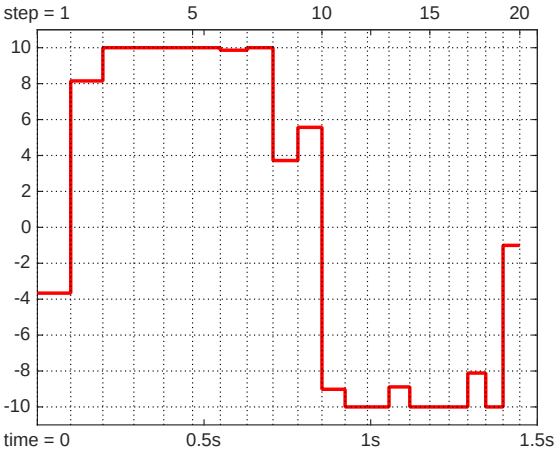
(c) Run 1 state trajectories



(d) Run 2 state trajectories

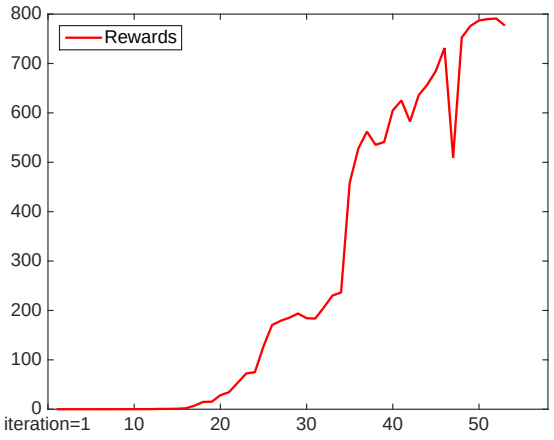


(e) Run 1 actions at each step

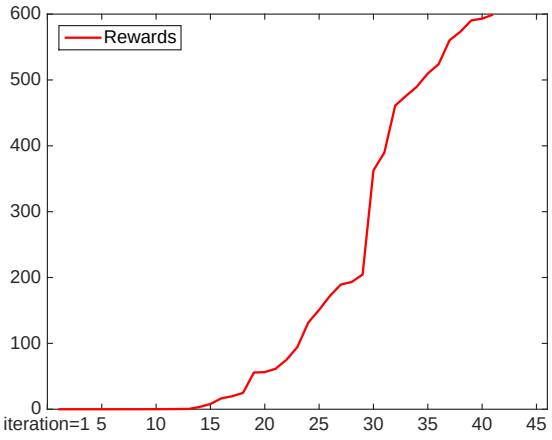


(f) Run 2 actions at each step

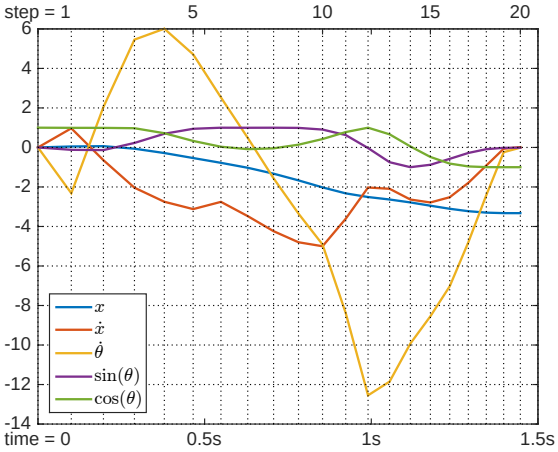
Figure 3.14: Total observed rewards of the execution of candidates on the real-world (top), the best set of actions found (bottom) and the resulting trajectories from the best set of action (middle) in run 1 and 2 of the inverted pendulum problem.



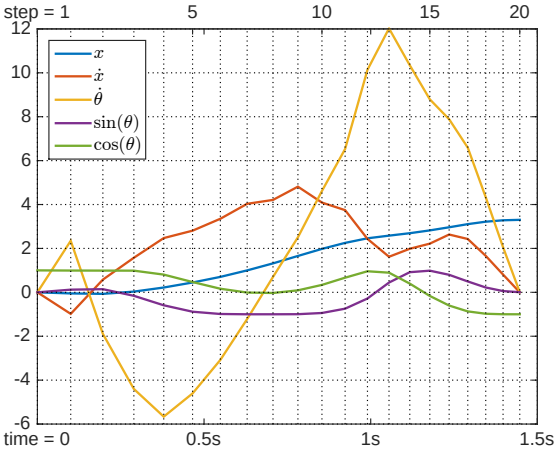
(a) Run 3 rewards at each iteration



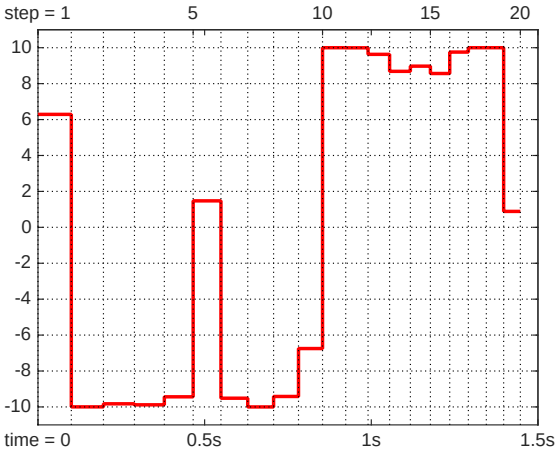
(b) Run 4 rewards at each iteration



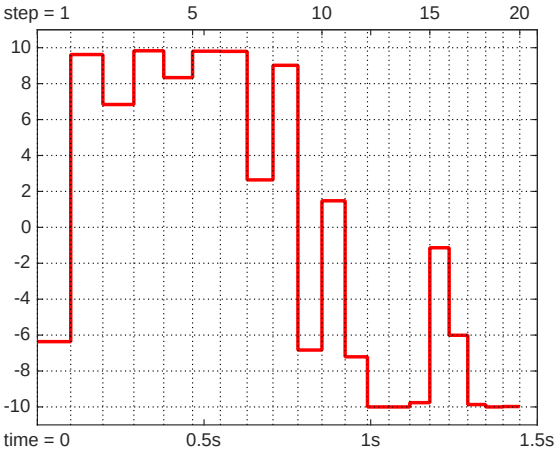
(c) Run 3 state trajectories



(d) Run 4 state trajectories

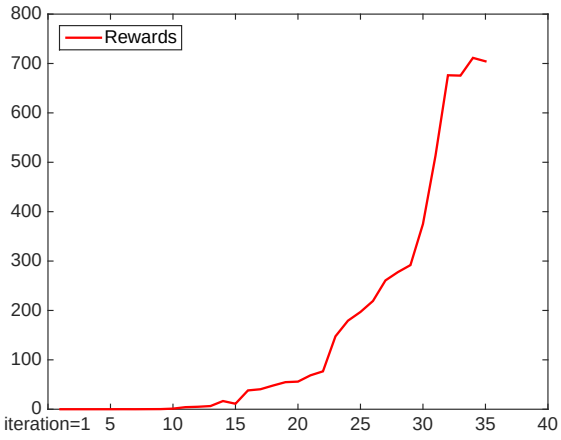


(e) Run 3 actions at each step

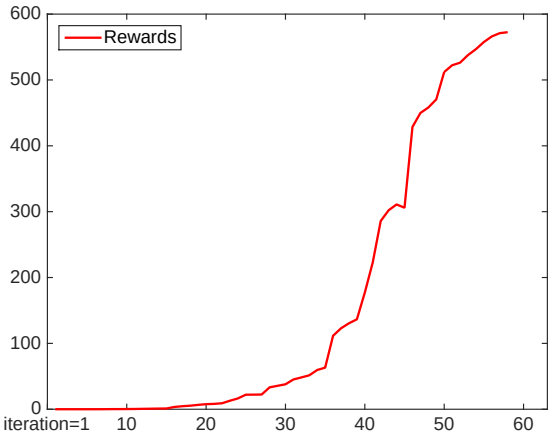


(f) Run 4 actions at each step

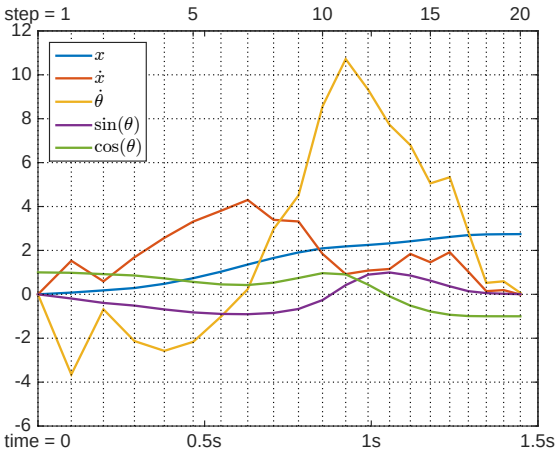
Figure 3.15: Total observed rewards of the execution of candidates on the real-world (top), the best set of actions found (bottom) and the resulting trajectories from the best set of action (middle) in run 3 and 4 of the inverted pendulum problem.



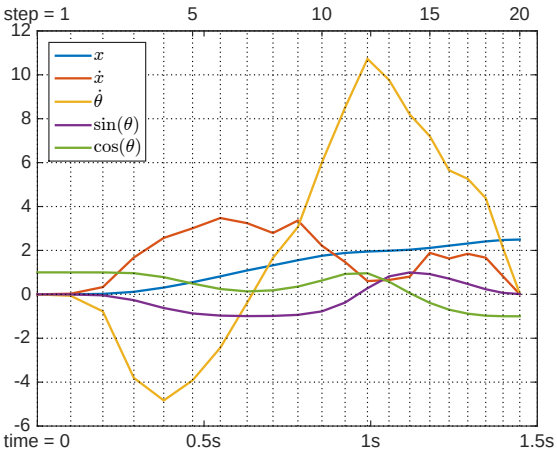
(a) Run 5 rewards at each iteration



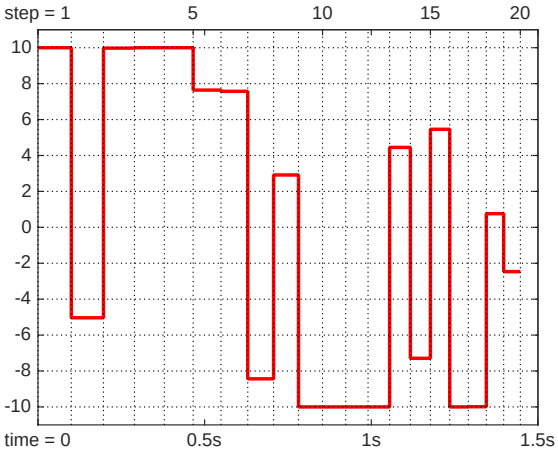
(b) Run 6 rewards at each iteration



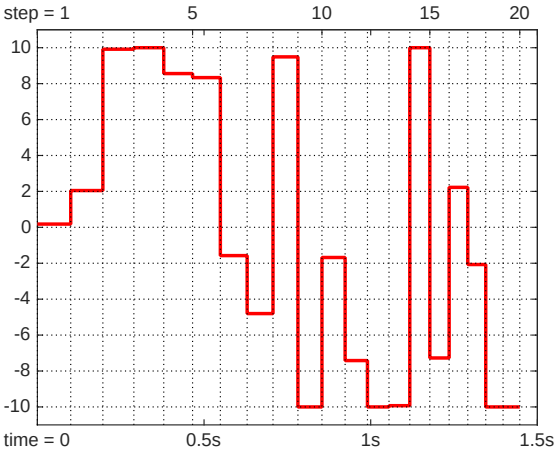
(c) Run 5 state trajectories



(d) Run 6 state trajectories



(e) Run 5 actions at each step



(f) Run 6 actions at each step

Figure 3.16: Total observed rewards of the execution of candidates on the real-world (top), the best set of actions found (bottom) and the resulting trajectories from the best set of action (middle) in run 5 and 6 of the inverted pendulum problem.

Figure 3.17 shows the number of iterations and time in seconds that are required to find a successful set of controls (red bars) for 10 different runs of the algorithm. It takes an average of 45 iterations, or 65 seconds to learn the task of bringing the last state of the trajectory in the target region. The blue bars in the figure show the number of iterations and time needed for the swing-up task defined earlier. On average it takes about 13 iterations, or 19.1 seconds of real-world interaction to learn this task.

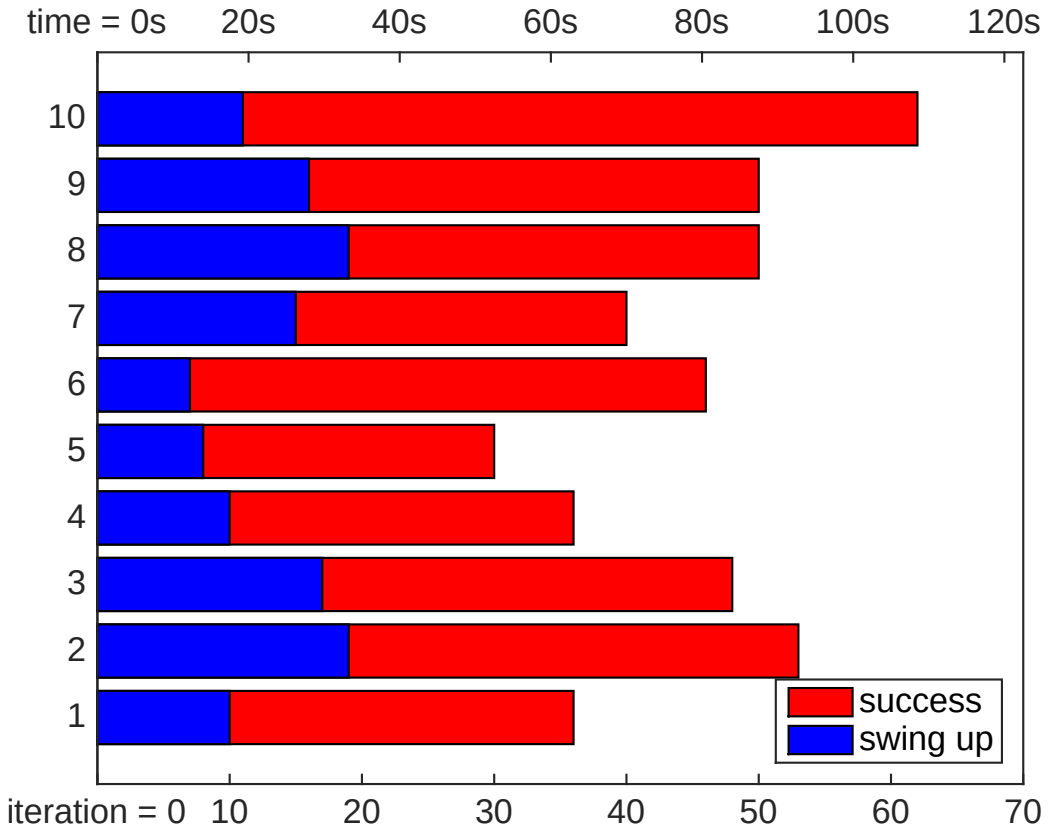


Figure 3.17: The number of iterations and time in seconds that it takes the proposed algorithm to find a set of controls for the swing-up task (blue), and for bringing the last state in the target region (red) for the inverted pendulum problem.

Figure 3.18 shows the progression of the total rewards in run 1 of Figure 3.14 for all the 536 suggested candidates of the Bayesian optimization step (blue lines), and the 46 candidates (red dots) that were executed on the real-world. Based on all the runs, real-world execution were decreased on average by a factor of 11.

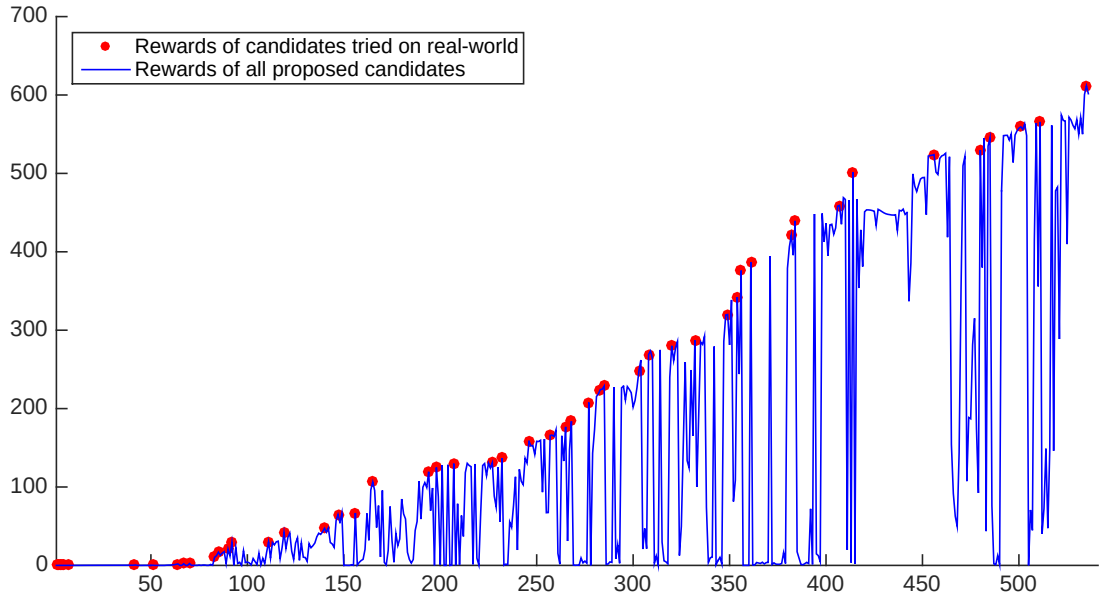


Figure 3.18: The observed rewards of the 46 candidates tried on the real-world (red dots), and the real-world rewards that would have been observed if all the 536 candidates that were tried in simulation were executed on the real-world (blue lines) for the inverted pendulum problem. This plot corresponds to run 1 in Figure 3.14

3.2.2 Finding A Feedback Policy for The Inverted Pendulum Problem

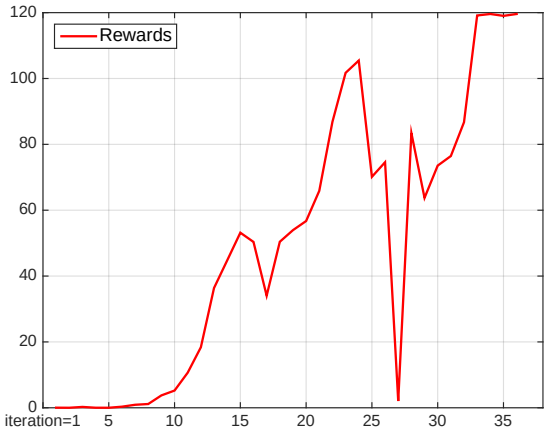
In completing the search for a feedback policy of the mountain car problem using the proposed method, we were able increase the suggested limit of the number of parameters of the Bayesian optimization from 20 to 75. This meant using 25 input and output support points for the policy-GP, which gave enough coverage on the two-dimensional state-space that produced a successful control policy. The increase of the number of states from two in the mountain car problem to five (or 4 if θ is used as an input state of the policy function instead of $\sin(\theta)$ and $\cos(\theta)$) in the inverted pendulum problem, and also the difficulty of the problem means that more support points are needed to produce a successful control policy. In a similar set up, [46] used 100 support points which is equal to 500 parameters (400 if θ is used), to complete the swing up and balance of the inverted pendulum problem, that required the final position of the cart to be also near zero. In the set up considered here, the final position of the cart is not important when the pendulum is balanced[5]. We also convert back $\sin(\theta)$ and $\cos(\theta)$ to θ before inputting the states into the policy-GP, which means that a support point in the policy approximator can have only the three states of $(\dot{x}, \dot{\theta}, \theta)$. This suggests that for a successful policy, the number of parameters can be decreased from 500. The exact number of the policy pa-

[5] by observing the equations of motion, one can see that the position does not appear in the equations of the other states, and therefore does not effect their progress over time. As an example, a policy that can swing-up the pendulum from position of x_1 at rest, can be successfully applied for swing-up when the cart is at x_2.

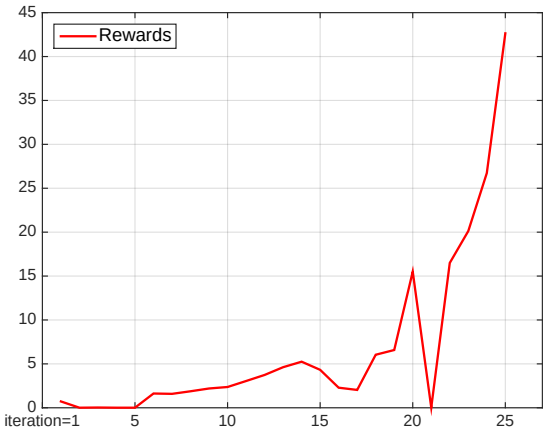
rameters needed for the feedback control of the inverted pendulum in such a setting is unknown (perhaps in the range of 200-300 parameters), but to our experience the search for a policy using the proposed algorithm in its current format did not produce any promising results when the number of parameters were increased substantially over the 75 used in the mountain car problem.

Instead here we present the results of using only 25 input and output support points, or 100 parameters for which the performance of the proposed algorithm did not degrade significantly. The time step is set to $t_d = 0.05$, and finite-horizon time in the learning phase is set to 1.75 seconds, or $T_1 = 35$ steps. Similar to the mountain car problem, successful policy is defined as one that can keep the system in the target region for an additional T_2 steps, where here $T_2 = 10$, or 0.5 seconds. The reward function is the same as the mountain car problem in equation (3.3). As with all the problems presented, the first five policies are generated randomly.

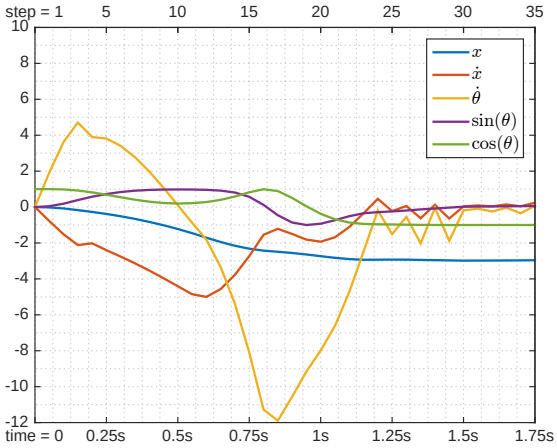
The proposed algorithm is able to complete the much easier task of the swing-up, it however fails to reach the more strict success criteria when it converges after failing to find policies that improve on the current best ($tol_{EIP} \to 0$ and $\Delta\boldsymbol{\theta}_{\text{relative}} \to 0$ in part C of Table 2.2). Figure 3.19 to Figure 3.21 show the results of 6 sample runs of the algorithm, all of which were able to swing up the pendulum in T_1 steps, but failed to reach the target region except for run 6 in Figure 3.21. Figure 3.22 shows the actions and trajectories that are resulted when the policy found in run 6 of Figure 3.21 is applied on the real-world for $T_1 + T_2$ steps. It can be seen that the velocity of the cart has exited the target region, and therefore the policy is unsuccessful. Although this result is promising in getting close to the target region, it is not one that has been consistently observed from all the different runs. The results suggest that more support points are needed in order to achieve successful control in bringing the system to the target region and maintaining it there, a task that the proposed method with the use of the Bayesian optimization is unable to achieve in its current form. In the next section, we provide possible ways the performance of the Bayesian optimization can be improved when the number of parameters are increased.



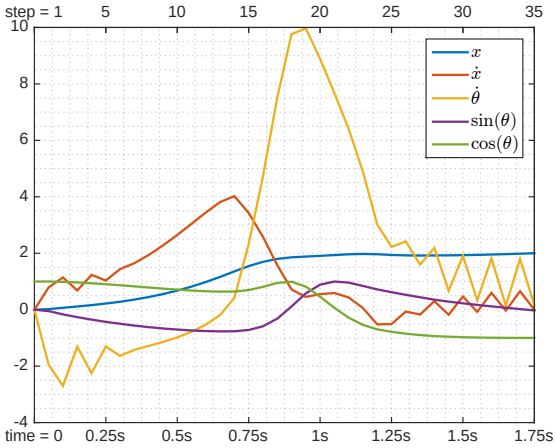
(a) Run 1 rewards at each iteration



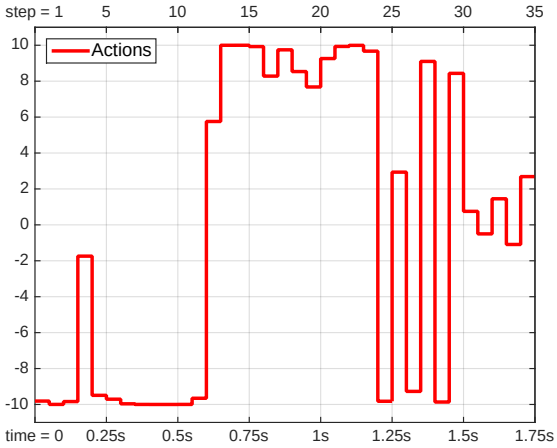
(b) Run 2 rewards at each iteration



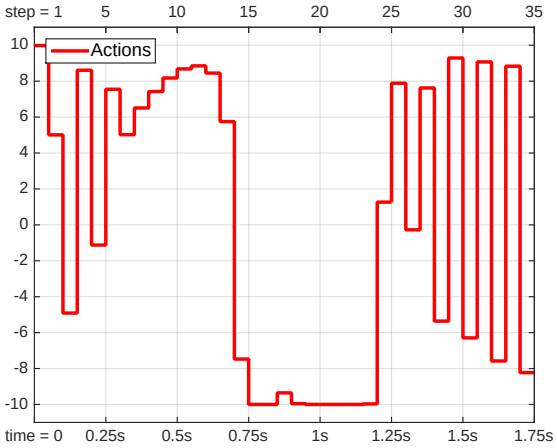
(c) Run 1 state trajectories



(d) Run 2 state trajectories

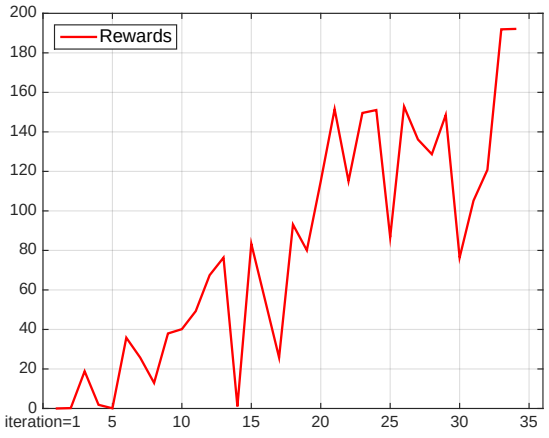


(e) Run 1 actions at each step

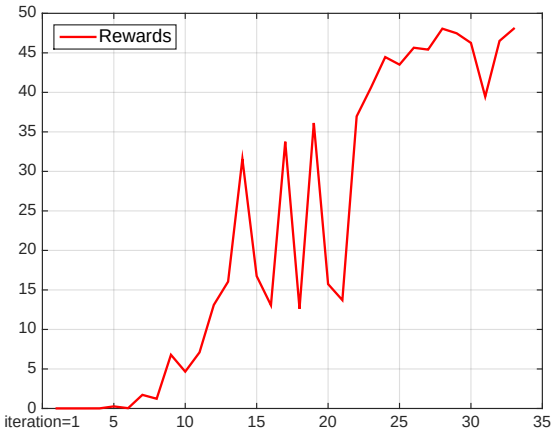


(f) Run 2 actions at each step

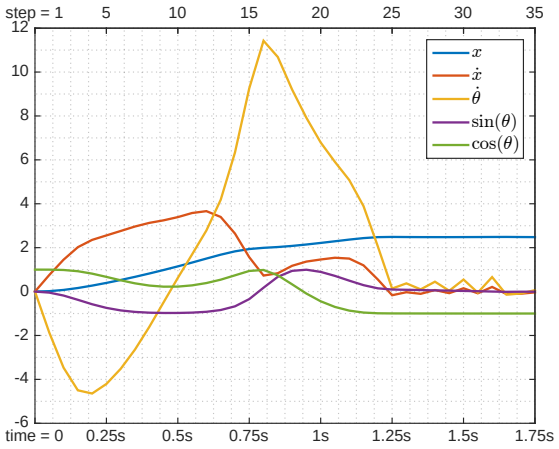
Figure 3.19: Total observed rewards of the execution of candidates on the real-world (top), the actions produced by the best policy (bottom) and the resulting trajectories for the best policy (middle) in run 1 and 2 of finding a feedback policy for the inverted pendulum problem.



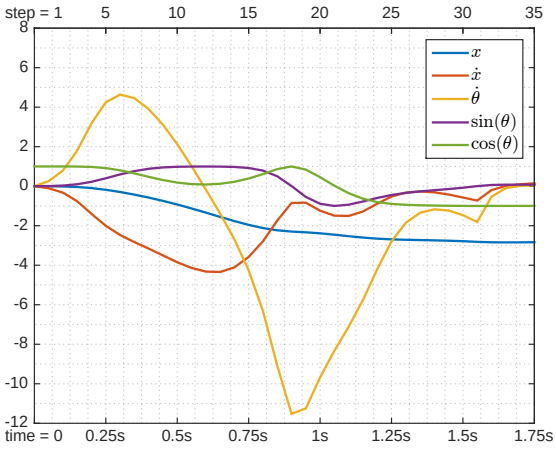
(a) Run 3 rewards at each iteration



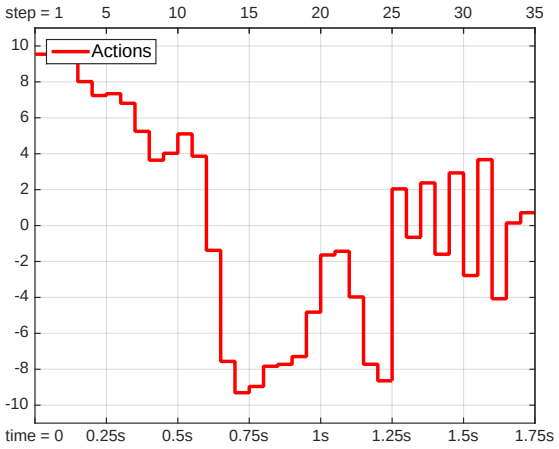
(b) Run 4 rewards at each iteration



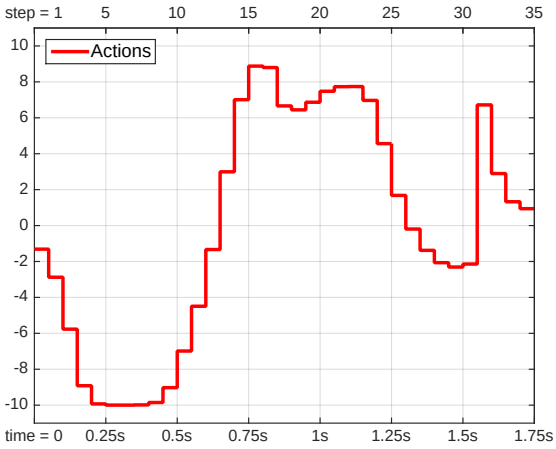
(c) Run 3 state trajectories



(d) Run 4 state trajectories

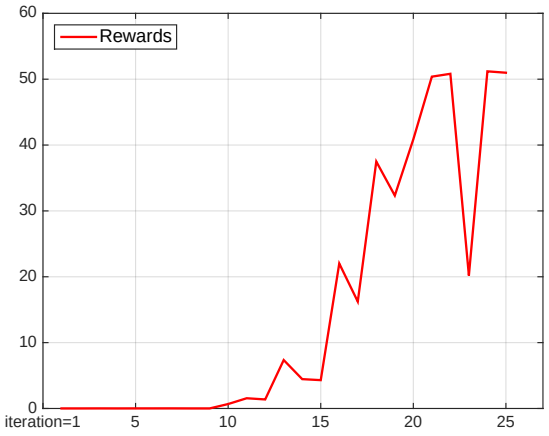


(e) Run 3 actions at each step

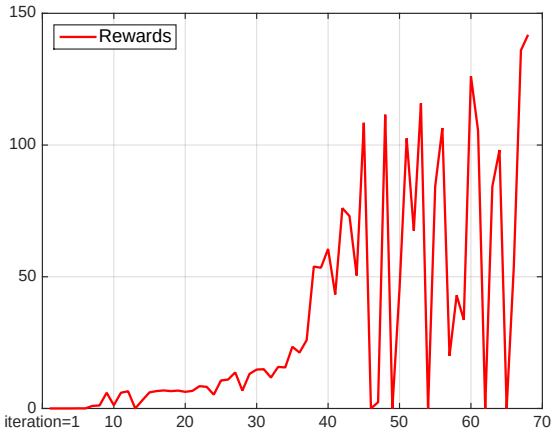


(f) Run 4 actions at each step

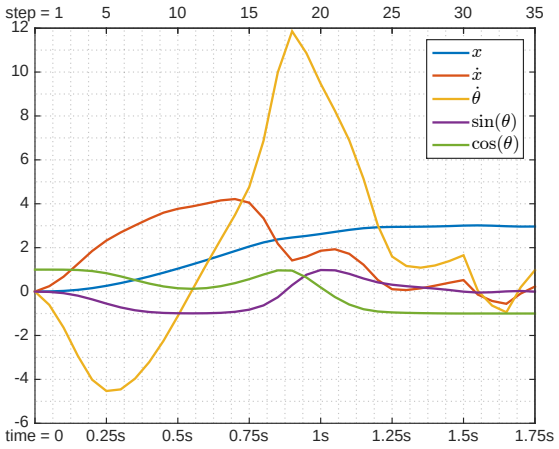
Figure 3.20: Total observed rewards of the execution of candidates on the real-world (top), the actions produced by the best policy (bottom) and the resulting trajectories for the best policy (middle) in run 3 and 4 of finding a feedback policy for the inverted pendulum problem.



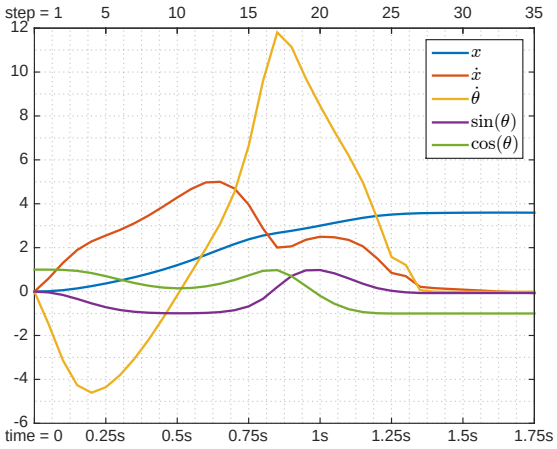
(a) Run 5 Rewards at each iteration



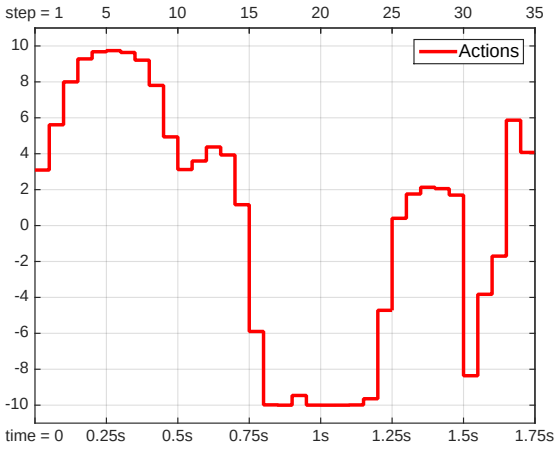
(b) Run 6 Rewards at each iteration



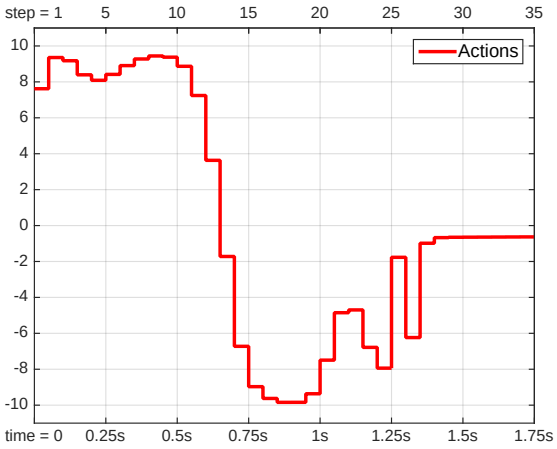
(c) Run 5 state trajectories



(d) Run 6 state trajectories

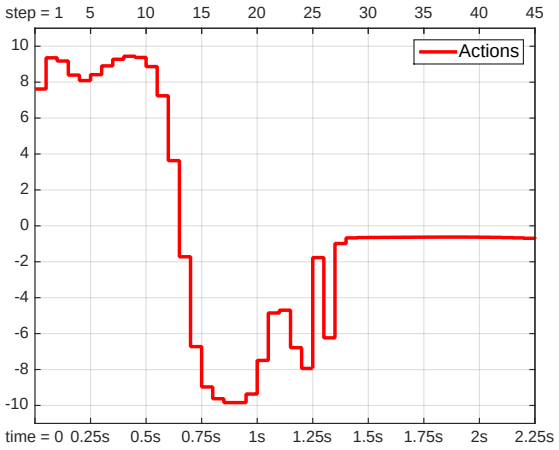


(e) Run 5 actions at each step

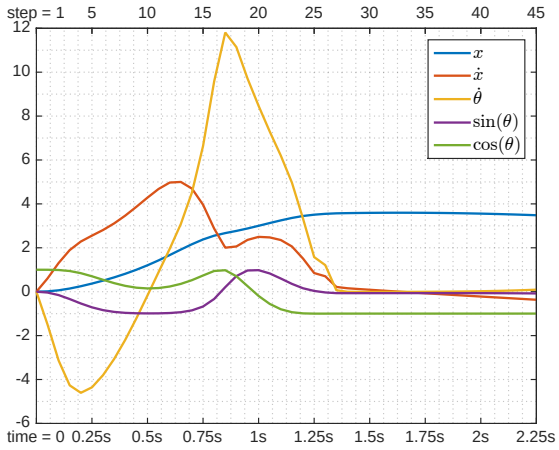


(f) Run 6 actions at each step

Figure 3.21: Total observed rewards of the execution of candidates on the real-world (top), the actions produced by the best policy (bottom) and the resulting trajectories for the best policy (middle) in run 5 and 6 of finding a feedback policy for the inverted pendulum problem.



(a) The actions for $T_1 + T_2$ steps



(b) The trajectories for $T_1 + T_2$ steps

Figure 3.22: The actions and trajectories from run 6 in Figure 3.21 when the finite-horizon time is extended to $T_1 + T_2$ steps

Figure 3.23 shows the number of iterations and total time in seconds that it took for 10 different runs of the algorithm to complete the swing-up task, the first 6 correspond to the runs in the figures above. As stated before, none of the runs were successful at reaching and maintaining the states in the target region. It takes an average of 22.95 seconds of real-world interaction, or about 13 iterations to learn the swing-up task.

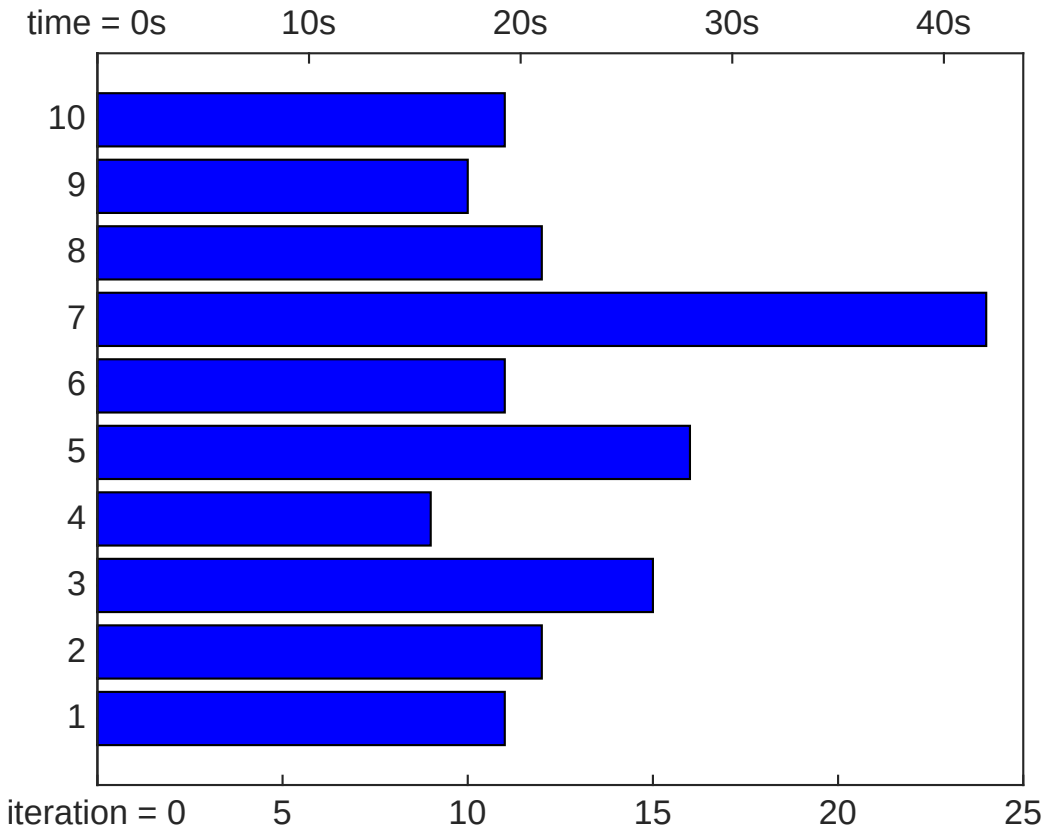


Figure 3.23: The number of iterations and time in seconds that it takes the proposed algorithm to find a feedback policy for the swing-up task in the inverted pendulum problem.

3.3 Discussion

A data-efficient algorithm that attempts to solve a control problem in continuous state and action space without the knowledge of the dynamics was presented. The main objective of this algorithm was to find a solution while reducing the amount of real-world interactions. This was done by employing a probabilistic model of the dynamics that help the Bayesian optimization in reducing uncertainties around policy parameters that were tried in simulations. The Bayesian optimization approach without the use of a dynamical model was previously used in [47], where the parameters for a neural network (8 parameters on average) were found that kept an already-upright double-pendulum balanced, a task that took 194 real-world executions each lasting for 10 seconds, or 1940 seconds of total real-world interactions. We used the algorithm presented here to solve the mountain car and the inverted pendulum problems. To solve these two problems, two different set-ups were used, the first found a set of actions/controls that were applied at every step for a fixed finite-horizon time, and the second found a feedback policy that could be applied for an extended finite-horizon time. For the mountain car problem the set of controls were found on average after 101 seconds of interactions with the real-world, and the feedback policy was learned in 85 seconds (section 3.1.2 discusses why the feedback policy is learnt faster). The set of controls that could swing-up the pendulum were learnt after 19.1 seconds of interactions, and the task of brining the pendulum within 2.5 degrees of the upright position with the velocities less than 0.05 took 65 seconds of real-world interaction. Although the feedback policy for the inverted pendulum at times came close to reaching the strict target region defined, it could only consistently learn the swing-up task, which was learnt on average in 22.9 seconds. In the problems that were successfully completed, learning a probabilistic model of the dynamics greatly reduced the real-world interactions; on average only every one out of 12 candidates that were suggested by the Bayesian optimization were executed out on the real-world. In addition to improving the efficiency of the Bayesian optimization in terms of the real-world interactions, to our knowledge we are the first that use the Bayesian optimization approach to find a high number of policy parameters for the purpose of control, which were 75 parameters in the mountain car problem, and 100 in the swing-up of the inverted pendulum as opposed to 8 parameters (on average) in [47].

Next, we discuss the limitations and shortcomings of the proposed algorithm, and offer possible ways they can be addressed. Table 3.3 lists the characteristics of the Bayesian optimization algorithm as suggested by [47], the PILCO algorithm which is the most data efficient model-based algorithm in the literature [46], and the proposed algorithm described in this work. The common property in these algorithms is that the dynamical equations that describe the behavior of the environment are unknown. Since decisions are made

without having this knowledge and are based on limited experience, the solutions found might not be as close in reaching the target as algorithms that have the dynamical equations. The notion of success in the RL literature is therefore often defined in a more "generous" way such as bringing/keeping the states within a large and often undefined region of the target. Nevertheless in this work, more strict criteria were placed when defining success, and the results from the previous chapter show that the proposed algorithm can successfully meet them in most cases.

Table 3.3: A comparison of the different algorithms considered.

	Direct Bayesian Optimization	PILCO	Proposed Algorithm
Type	Direct	Model-based	Direct and Model-based
Dynamics-GP	✗	✓	✓
Rewards-GP	✓	✗	✓
Uncertainty of rewards [6]	considered	not considered	considered
Data-efficient	✗	✓	✓
Comp-efficient	✗	✗	✗
Search Guidance	probabilistic, global Bayes opt.	deterministic, conj-grad. method	probabilistic, global Bayes opt.
Global vs Local	global	local[7]	global
Explore vs. Exploit	based on rewards-GP	based on dynamics-GP	based on both
Exploration parameter	automatic [8]	not automatic [9]	automatic
# of policy parameters	low	high	medium

Similar to optimal control methods, RL methods provide local solutions in the sense that success is described as finding *any* solution that meets the goal criteria, and finding the global optimum is not ensured. Although Bayesian optimization offers a global view for the search of a solution, finding a global optimal solution in practice would require an infinite amount of interaction with the expensive objective function. This is due to the fact that the search for a solution depends heavily on the hyperparameters of the GP; the search can get stuck in local optima due to the over-confidence of the model, and promoting exploration by changing the hyperparameters (lower values on the length-scales, or higher values on signal variance) and prolonging the search would need to be repeatedly performed if all possible regions are to be visited. This issue of exploration vs. exploitation is not unique to the Bayesian optimization approach, and all global techniques can suffer from missing better solutions in undiscovered regions. Although it has been shown in

[6] whether or not the uncertainity of rewards is considered in the step of searching for a policy

[7] policy search at every iteration is done using a local conjugate gradient method. The overall search might not be local depending on the amound of exploration.

[8] if EI acquisition function is used

[9] there should be more exploration at the begining and less towards the end. Exploration can be encouraged by modifying the rewards function to include the variance of the rewards

the literature that with the right choice of the hyperparameters convergence can be achieved, in practice finding a global solution with the Bayesian optimization, while keeping the amount of interaction with the environment bounded, cannot be guaranteed.

As mentioned in section 1.4.2, [27] has shown the convergence of the Expected Improvement in theory when the observations are not noisy. This convergence relies on putting bounds on the length-scales, finding the signal variance α^2 based on the current length-scales, and maximizing the likelihood based on this α^2 to find the updated length-scales. In practice however, choosing the bounds of the length-scales proved to be highly problem dependent and a challenge all by itself. This is because when the length-scales increase, confidence in the model is increased, but high values of the length-scales result in bigger α^2 values, which would increase uncertainty away from the observations, and therefore promote exploration. Adaptively changing the bounds on the length-scales similar to the approach presented in [29][10] would perhaps balance this issue, but our attempts to incorporate this to ensure theoretical convergence, while at the same time keeping real-world interaction to a minimum has not been successful. Instead the un-bounded hyperparameters were optimized together by maximizing the evidence as mentioned in equation (1.16).

A key feature in the PILCO algorithm that enhances the data-efficiency is its ability to extended the finite-horizon time during the learning phase of the algorithm if the dynamical model is performing well as compared to the observed trajectories from the real-world. This is beneficial when the feedback policy is expected to perform well with the extension of time. In the proposed method, each policy parameter is assigned a returned reward for a fixed finite-horizon time, which are then used to construct the observations of the RGP. If the time during the learning phase is to be extended, previous policies would all have to be executed on the real-world again to observe the returned rewards for the extended period of time, which would require a lot of real-world interactions. An alternative to this would be to use the dynamical model to simulate ahead only the extended portions of the finite-horizon time for the policies already implemented on the real-world. This however would make all the training data of the RGP up to that point as uncertain measurements, which could potentially have an adverse effect on the modeling of the RGP. The extension of time should therefore be performed only when uncertainty is low in the trajectories produced by the DGP, and the mean of these trajectories are close to the ones observed from the real-world.

One of the issues with the proposed algorithm in its current format is that the number of the parameters to be optimized (the policy parameters) cannot be large. This stems from an inherent problem of the Bayesian

[10] which was also used to show convergence of EI in the presence of noisy observations

optimization approach that is known to work well in low-dimensions, but degrades in performance when the number of parameters grow large. Previously, [22] reported this degradation in performance to happen with problems that require above 15-20 parameters. We were able to used the proposed algorithm to find feedback policies with 75 parameters for the mountain car, and 100 parameters for the swing-up of the inverted pendulum, but a search for a policy consisting of a few hundred parameters was not successful. The failure of the Bayesian optimization in such a setting is caused by the inability of its surrogate GP, in our case the RGP, to model well in higher dimensional space, which then fails to guide the search in the right direction. In the proposed method this issue was somewhat ameliorated by adaptively changing the scope of the search around known policies that produce high rewards, and also by the selection of a combination of covariance functions that can represent both the flat low-reward regions, and regions of the parameter-space that include sudden increases in the returned rewards. To further improve the performance of the Bayesian optimization, the mapping of the policy parameters to rewards in the RGP needs to be enhanced as explained below.

Much like the conventional Bayesian optimization technique, we currently use a single GP to globally map the policy parameters to the returned rewards. With this global GP, one set of hyperparameters is used to model the entire parameter-space. At the beginning of a run of the algorithm when high-reward regions are not yet discovered, the length-scales are set to high values that convey that the expected changes in the returned rewards from policies that are close to each other are small. At the later stages of the algorithm however, high-reward regions are discovered. In these regions, policies that are close to each other can produce very different returned rewards, which calls for the need of small length-scale values to allow the model to capture these sudden changes on the output from the small changes on the input. The reduction of the length-scales however, would create large uncertainties in the flat regions that would promote exploration, and therefore prolonging the search, and as a result the increase in the real-world interactions. To eliminate this important limitation in the modeling of the RGP, local GPs can be used that would use a different set of lenght-scales for each region of the parameter-space, with low values for the length-scales in the high-reward regions, and high values for the length-scale in the flat regions where the expected rewards are low. An alternative approach to the use of local GPs would be to create a non-stationary covariance function that can vary the hyperparameters based on the location of the input. Similar to the local GPs, low length-scales values are assigned to high reward regions, and high length-scales values are assigned to low reward regions.

The use of local GPs would require an algorithm that can adaptively cluster the training data. This is the topic of the next chapter where we propose EM$^+$, an algorithm based on the expectation maximization

approach that can automatically infer the number of components needed to model the observations well.

Chapter 4

EM^+ , An Algorithm for Mixture of Models with Unknown Number of Components

4.1 Introduction

The Mixture model problems arise in a wide range of fields such as clustering, probability density function estimation, and local regression models. When the number of components of the mixture model is known, the Expectation Maximization (EM) algorithm, and its special case, the K-means algorithm, are the most popular methods used. The problem however becomes a much more difficult one when the number of components is unknown. This chapter begins with a formal introduction to a special case of the mixture model problem, namely the mixture of Gaussians. This is followed by the EM algorithm and its variations that all assume a prior knowledge of the number of components in the mixture model. Methods that do not assume this prior knowledge, such as the Dirichlet process (DP), and other methods that seek to find the right number of components by utilizing some pre-defined measure are then reviewed. We then presents EM^+, an extension to the EM algorithm that takes inspiration from the DP method, and combines its flexibility with the efficiency of the EM approach that can be used to solve mixture model problems with an unknown number of mixture components.

4.1.1 Mixture of Gaussians

We consider mixtures of normal PDFs expressed as[1]

$$\boldsymbol{X} \;\sim\; p(\boldsymbol{x} \mid \pi_{1:K}, \boldsymbol{\mu}_{1:K}, \boldsymbol{\Sigma}_{1:K}) \equiv \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{4.1}$$

where the random variable (RV) $\boldsymbol{X}$ takes values in $\mathbb{R}^D$ and $\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ is the multivariate Gaussian distribution

$$\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-\frac{D}{2}} \mid \boldsymbol{\Sigma} \mid^{-\frac{1}{2}} \exp(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})) \tag{4.2}$$

with mean $\boldsymbol{\mu} \in \mathbb{R}^D$ and covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{D\times D}$. The $\pi_{1:K}$ satisfying

$$\pi_k \geq 0, \quad k = 1, \ldots, K, \qquad \sum_{k=1}^{K} \pi_k = 1 \tag{4.3}$$

are the mixing proportions for the K components of the mixture. Given N i.i.d. observations $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots \boldsymbol{x}_j \ldots \boldsymbol{x}_N\} \sim p(\boldsymbol{x} \mid \boldsymbol{\pi}_{1:K}, \boldsymbol{\mu}_{1:K}, \boldsymbol{\Sigma}_{1:K})$, with the latent model identifier vector $\boldsymbol{Z} = \{z_1, \ldots, z_i, \ldots, z_N\}$ having values from $1 \ldots K$ for each of the observations[2], the objective is to estimate the mixture parameters $\boldsymbol{\Theta} \equiv \{\boldsymbol{\mu}_{1:K}, \boldsymbol{\Sigma}_{1:K}, \pi_{1:K}\}$ if K is known, and if K is unknown to estimate the number of mixture components K and the mixture parameters $\boldsymbol{\Theta} \equiv \{\boldsymbol{\mu}_{1:K}, \boldsymbol{\Sigma}_{1:K}, \pi_{1:K}\}$. The *Maximum Likelihood* estimate of the parameters $\boldsymbol{\Theta}$ is obtained by maximizing the log-likelihood

$$\boldsymbol{\Theta}_{ML} = \underset{\boldsymbol{\Theta}}{\operatorname{argmax}} \log\{l(\boldsymbol{\Theta})\} \tag{4.4}$$

where the likelihood is

$$l(\boldsymbol{\Theta}) = \prod_{i=1}^{N} p(\boldsymbol{x}_i \mid \pi_{1:K}, \boldsymbol{\mu}_{1:K}, \boldsymbol{\Sigma}_{1:K}). \tag{4.5}$$

Alternatively, in a Bayesian setting, one considers the parameters as RV's, assigns a prior distribution $p(\boldsymbol{\Theta} \mid \boldsymbol{\Psi})$, and seeks to compute their posterior distribution $p(\boldsymbol{\Theta} \mid \boldsymbol{x}_{1:N}, \boldsymbol{\Psi})$. Here, $\boldsymbol{\Psi}$ denotes the so-called *hyperparameters* defining the prior distribution on the mixture parameters. A commonly used point estimate is the mode of the posterior distribution, that is the *Maximum A Posteriori* estimate of $\boldsymbol{\Theta}$, obtained as

$$\boldsymbol{\Theta}_{\text{MAP}} = \underset{\boldsymbol{\Theta}}{\operatorname{argmax}} \log\left\{l(\boldsymbol{\Theta}) \cdot p(\boldsymbol{\Theta} \mid \boldsymbol{\Psi})\right\}. \tag{4.6}$$

MAP estimates avoid certain disadvantages of the ML estimates, such as singularities during the optimization, however, they can be criticized for the need of considering an arbitrary prior distribution. The latter

[1] In equation (4.1) and the remainder of the paper, the notation $v_{1:K}$ signifies collectively the variables $\{v_1, \ldots, v_K\}$.
[2] z_i signifies which component $\boldsymbol{x}_i$ was drawn from

issue can be mitigated by employing either a *Hierarchical Bayes* approach in which higher level priors are brought in on the hyperparameters or an *Empirical Bayes* approach in which the hyperparameters are optimally selected based on the data.

The optimization problems (4.4) and (4.6) require a numerical approach in general. However, an analytical solution is possible if the information about which component of the mixture produced each sample is assumed to be available. Thus these problems are amenable to *Expectation/Maximization* (EM) algorithms ([59, 60]). A standard EM algorithm for fitting normal mixtures is briefly reviewed next since the proposed algorithm builds on it.

4.1.2 EM Algorithm

Although previous works using special cases of the EM approach existed, its general format was first formally presented in [59], where it was shown that as the algorithm progresses, the likelihood of the observations given the parameters increases monotonically. With the absence of the model identifier vector $\boldsymbol{Z} = \{z_1, \dots z_N\}$ in practice[3], we only have access to an *incomplete* data. The *complete* log likelihood for N i.i.d samples, $\boldsymbol{X} = \{\boldsymbol{x}_1, \dots \boldsymbol{x}_N\}$, is

$$\begin{aligned} L(\boldsymbol{\Theta}) &= \log p(\boldsymbol{X}, \boldsymbol{Z} \mid \boldsymbol{\Theta}) \\ &= \log \prod_{i=1}^{N} p(\boldsymbol{x}_i, z_i \mid \boldsymbol{\Theta}) \\ &= \log \left\{ \prod_{i=1}^{N} \prod_{k=1}^{K} \left(\mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_k) \cdot \pi_k\right)^{\mathbb{I}(z_i = k)} \right\} \end{aligned} \tag{4.7}$$

which can be maximized analytically, but the resulting estimates of the parameters become also functions of $\boldsymbol{Z}$, which is generally unknown. Instead, the EM algorithm alternates between the *Expectation* and *Maximization* steps as follows. At iteration t of the EM algorithm, let $\boldsymbol{\Theta}(t)$ be the current estimate of the parameters and consider an auxiliary function $Q(\boldsymbol{\Theta}, \boldsymbol{\Theta}(t))$ as the expectation of the full log-likelihood with respect to $p(\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{\Theta}(t))$

[3] knowledge of the $\boldsymbol{Z}$ vector would reduce the problem to a far easier one.

$$
\begin{aligned}
Q(\boldsymbol{\Theta}, \boldsymbol{\Theta}(t)) &= E\left[\log \prod_{i=1}^{N} p(\boldsymbol{x}_i, z_i \mid \boldsymbol{\Theta})\right] \\
&= \sum_{i=1}^{N} E\left[\log p(\boldsymbol{x}_i, z_i \mid \boldsymbol{\Theta})\right] \\
&= \sum_{i=1}^{N} E\left[\log \left(p(\boldsymbol{x}_i \mid z_i, \boldsymbol{\Theta}) p(z_i \mid \boldsymbol{\Theta})\right)\right] \\
&= \sum_{i=1}^{N} E\left[\log \left(\prod_{k=1}^{K} \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_k)^{\mathbb{I}(z_i=k)} \prod_{k=1}^{K} \pi_k^{\mathbb{I}(z_i=k)}\right)\right] \\
&= \sum_{i=1}^{N} E\left[\sum_{k=1}^{K} \mathbb{I}(z_i = k) \log\left(\pi_k \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_k)\right)\right] \\
&= \sum_{i=1}^{N} \sum_{k=1}^{K} p(z_i = k \mid \boldsymbol{x}_i, \boldsymbol{\Theta}(t)) \log\left(\pi_k \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_k)\right)
\end{aligned}
\tag{4.8}
$$

In the *Expectation* step (E-step), $p(\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{\Theta}(t))$ is computed based on the current parameters. Then in the *Maximization* step (M-step), a new set parameters are found[4] such that

$$
\boldsymbol{\Theta}(t+1) = \underset{\boldsymbol{\Theta}}{\operatorname{argmax}}\, Q(\boldsymbol{\Theta}, \boldsymbol{\Theta}(t)) \tag{4.9}
$$

It can be shown that the likelihood (4.5) is increasing, i.e. $l(\boldsymbol{\Theta}(t+1)) \geq l(\boldsymbol{\Theta}(t))$, but only convergence to a local maximum can be guaranteed in general.

The EM algorithm as described above aims at maximizing the likelihood (4.5) and thus at producing the maximum likelihood estimates of the parameters $\boldsymbol{\Theta}_{\mathrm{ML}}$; this will be referred to as the maximum likelihood approach (EM-ML). When applied to normal mixtures, it may suffer from singularities (i.e. estimated variances converge to zero) and from slow convergence as the dimensionality of the problem increases [61,62]. An alternative approach has been proposed that instead aims at maximizing the posterior distribution of the parameters given the samples [63] and thus at producing the maximum a posteriori estimate of the parameters $\boldsymbol{\Theta}_{\mathrm{MAP}}$; this will be referred to as the maximum a posteriori approach (EM-MAP). In the EM-MAP algorithm, the full log-likelihood in (4.7) is replaced by

$$
L(\boldsymbol{\Theta}) + \sum_{k=1}^{K} \log\{p(\boldsymbol{\Theta}_k \mid \boldsymbol{\Psi})\} = \log\left\{\prod_{i=1}^{N} \prod_{k=1}^{K} \left(\mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_k) \cdot \pi_k\right)^{\mathbb{I}(z_i=k)} \prod_{k=1}^{K} p(\boldsymbol{\Theta}_k \mid \boldsymbol{\Psi})\right\}. \tag{4.10}
$$

Since the additional term in (4.10) does not depend on $\boldsymbol{Z}$, the expectation step (4.8) remains the same as in

[4] for derivation of the estimates of the parameters in M-step, refer to CH 9 of [61]

(EM-ML); however, the maximization step is replaced by

$$\boldsymbol{\Theta}(t+1) = \underset{\boldsymbol{\Theta}}{\operatorname{argmax}}\{Q(\boldsymbol{\Theta}, \boldsymbol{\Theta}(t)) + \log p(\boldsymbol{\Theta} \mid \boldsymbol{\Psi})\}. \tag{4.11}$$

In the following, we summarize the formulas implementing the (EM-MAP) algorithm from [61]. We assume that conjugate Dirichlet and independent Normal-Inverse-Wishhart priors have been selected for the $\pi_{1:K}$ and $\{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$, $k = 1, \ldots, K$ parameters, respectively. Namely,

$$p(\boldsymbol{\Theta} \mid \boldsymbol{\Psi}) = \mathrm{Dir}(\pi_{1:K} \mid \alpha_{1:K}) \cdot \prod_{k=1}^{K} \mathrm{NIW}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k \mid \boldsymbol{m}_0, \kappa_0, \nu_0, \boldsymbol{S}_0) \tag{4.12}$$

where

$$\mathrm{Dir}(\pi_{1:K} \mid \alpha_{1:K}) \equiv \frac{\Gamma(\sum_{k=1}^{K} \alpha_k)}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \prod_{k=1}^{K} \pi_k^{\alpha_k - 1}, \qquad \alpha_k > 0 \tag{4.13}$$

and

$$\mathrm{NIW}(\boldsymbol{\mu}, \boldsymbol{\Sigma} \mid \boldsymbol{m}_0, \kappa_0, \nu_0, \boldsymbol{S}_0) \equiv \mathcal{N}(\boldsymbol{\mu} \mid \boldsymbol{m}_0, \frac{1}{\kappa_0}\boldsymbol{\Sigma}) \cdot \mathrm{IW}(\boldsymbol{\Sigma} \mid \boldsymbol{S}_0, \nu_0), \tag{4.14}$$

with the Inverse Wishhart prior expressed as

$$\mathrm{IW}(\boldsymbol{\Sigma} \mid \boldsymbol{S}_0, \nu_0) \equiv \frac{2^{-\frac{\nu_0 D}{2}} \det(\boldsymbol{S}_0)^{\frac{\nu_0}{2}}}{\Gamma_D(\nu_0/2)} \cdot \mid \boldsymbol{\Sigma} \mid^{-\frac{\nu_0 + D + 1}{2}} \cdot \exp\left\{-\frac{1}{2}\mathrm{trace}(\boldsymbol{S}_0 \boldsymbol{\Sigma}^{-1})\right\}. \tag{4.15}$$

where Γ_D is the multivariate gamma function. Thus, the hyperparameters considered are $\boldsymbol{\Psi} \equiv \{\alpha_{1:K}, \boldsymbol{m}_0, \kappa_0, \nu_0, \boldsymbol{S}_0\}$. In the E-step, one computes the responsibility coefficients, which is $p(\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{\Theta}(t))$, the posterior probabilities for the indicator variables $\boldsymbol{Z}$ given the data $\boldsymbol{X}$ and the current estimate of the parameters $\boldsymbol{\Theta}$. More specifically,

$$R_{ik} \equiv p(z_i = k \mid x_i, \boldsymbol{\Theta}(t)) = \frac{\bar{R}_{ik}}{\sum_{k=1}^{K} \bar{R}_{ik}} \tag{4.16}$$

where

$$\bar{R}_{ik} \equiv \; p(x_i \mid z_i = k, \boldsymbol{\Theta}(t)) \cdot p(z_i = k \mid \boldsymbol{\Theta}(t)) = \mathcal{N}(x_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \cdot \pi_k. \tag{4.17}$$

Next, using (4.7) and formulas (4.14) and (4.47) for the priors in (4.8), the following expression for the auxiliary function $Q(\boldsymbol{\Theta}, \boldsymbol{\Theta}(t))$ can be readily computed:

$$Q(\boldsymbol{\Theta}, \boldsymbol{\Theta}(t)) = \sum_{i=1}^{N}\sum_{k=1}^{K} R_{ik} \log\left\{\mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\cdot \pi_k\right\} + \sum_{k=1}^{K} \log\left\{\mathrm{NIW}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k \mid \boldsymbol{m}_0, \kappa_0, \nu_0, \boldsymbol{S}_0)\right\} + \log\left\{\mathrm{Dir}(\pi_{1:K} \mid \alpha_{1:K})\right\}. \tag{4.18}$$

Finally, the maximization step (4.11) can be analytically carried out by solving the first order optimality conditions yielding the next parameter iterates from

$$N_k = \sum_{i=1}^{N} R_{i,k} \tag{4.19}$$

$$\pi_k = \frac{N_k + \alpha_k - 1}{N + \sum_{k=1}^{K}\alpha_k - K} \tag{4.20}$$

$$\bar{\boldsymbol{x}}_k = \frac{\sum_{i=1}^{N} R_{ik}\boldsymbol{x}_i}{N_k} \tag{4.21}$$

$$\boldsymbol{\mu}_k = \frac{N_k \bar{x}_k + \kappa_o \boldsymbol{m}_o}{N_k + \kappa_0} \tag{4.22}$$

$$\boldsymbol{S}_k = \sum_{i=1}^{N} R_{ik}(\boldsymbol{x}_i - \bar{\boldsymbol{x}}_k)(\boldsymbol{x}_i - \bar{\boldsymbol{x}}_k)^T \tag{4.23}$$

$$\boldsymbol{\Sigma}_k = \frac{\boldsymbol{S}_0 + \boldsymbol{S}_k + \frac{\kappa_0 N_k}{\kappa_0 + N_k}(\bar{\boldsymbol{x}}_k - \boldsymbol{m}_0)(\bar{x}_k - \boldsymbol{m}_0)^T}{\nu_o + N_k + D + 2}. \tag{4.24}$$

We remark that the responsibilities R_{ik} can be interpreted as the degree to which a sample $\boldsymbol{x}_i$ belongs to cluster k based on current information. Then $R_{ik}\cdot\boldsymbol{x}_i$ can be viewed as the fraction of sample $\boldsymbol{x}_i$ that belongs to cluster k and N_k in (4.19) as the *effective* number of samples in cluster k. On the other hand, $\alpha_k - 1$ signifies the number of samples belonging to cluster k out of $\sum_{k=1}^{K}\alpha_k - K$ samples based on our prior belief. Therefore, the posterior cluster probabilities in (4.20) are naturally given as the fraction of total points from the prior and those estimated from the data as belonging to each component. Furthermore, the effective samples $R_{ik}\boldsymbol{x}_i$, $i = 1, \ldots, N$ in cluster k are combined to yield cluster sample means $\bar{\boldsymbol{x}}_k$ and sample covariances $\boldsymbol{S}_k$ in (4.21) and (4.24), respectively. Then, the estimates in (4.22) and (4.24) naturally represent the mode of the posterior distributions of the parameters $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$, each being a NIW distribution based on normal samples with statistics given by (4.21) and (4.24) and our selection of the conjugate NIW prior in (4.14).

We also remark that (4.20) is valid when $N_k + \alpha_k - 1 > 0$ for all $k = 1, \ldots, K$. In the more general case, (4.20) should be replaced by

$$\pi_k = \begin{cases} (N_k + \alpha_k - 1) \Big/ \left(\sum_{k \in K^*} N_k + \alpha_k - 1 \right) \\ 0, \quad k \notin K^* \end{cases} \tag{4.25}$$

with $K^* \equiv \{k = 1, \ldots, K \mid N_k + \alpha_k > 1\}$. This more general case is usually not considered since, if any $N_k + \alpha_k - 1 < 0$, the maximization of (4.18) with respect to the π_k's is not well defined. This situation can be avoided by enforcing $\alpha_k \geq 1$.

4.1.2.1 K-means Algorithm

The K-means algorithm is a popular variation of the EM algorithm for the Gaussian mixture models, where the covariance of each of the mixtures is assumed to be spherical and fixed to $\boldsymbol{\Sigma}_k = \sigma^2 \boldsymbol{I}_D$, and the mixtures weights are fixed to $\pi_k = \frac{1}{K}$. The only parameters to be estimated are then the centers of the mixtures, $\boldsymbol{\mu}_k$ [62]. Therefore we have

$$p(\boldsymbol{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{(2\pi\sigma^2)^{\frac{D}{2}}} \exp(-\frac{1}{2}||\boldsymbol{x}_i - \boldsymbol{\mu}_k||^2) \tag{4.26}$$

In the E-step, the responsibility of the observation $\boldsymbol{x}_i$ belonging to mixture j is

$$R_{ij} = \frac{\exp(-\frac{1}{2\sigma^2}||\boldsymbol{x}_i - \boldsymbol{\mu}_j||^2)}{\sum_{k=1}^{K} \exp(-\frac{1}{2\sigma^2}||\boldsymbol{x}_i - \boldsymbol{\mu}_k||^2)} \tag{4.27}$$

As $\sigma^2 \to 0$, the denominator term above belonging to component k for which $||\boldsymbol{x}_i - \boldsymbol{\mu}_k||^2$ is smallest goes to zero more slowly than the others, and therefore responsibilities $R_{i1} \ldots R_{iK}$ will all go to zero except for the k component that will make R_{ik} unity. As a result during the E-step of the K-means algorithm, each point is assigned to its nearest component as

$$z_i = \underset{k}{\operatorname{argmin}} \, ||\boldsymbol{x}_i - \boldsymbol{\mu}_k|| \tag{4.28}$$

which is why the K-means algorithm is often referred to as the "hard EM". In the M-step, the centers of the mixture are updated by the mean of the points belonging to that mixture by

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i:z_i=k} \boldsymbol{x}_i \tag{4.29}$$

where $N_k \in \mathbb{Z}$ is the number of observations assigned to mixture K.

4.1.3 Dirichlet Process Mixture Models

So far it has been assumed that K, the number of components of the mixtures model is known. This however is often not the case which requires approaches that do not require the knowledge of K. The Bayesian nonparametric mixture modeling gives such flexibility, where it is possible to find a suitable number of mixtures by assuming a countable infinite mixture model [64–67]. In such a setting, a Dirichlet process prior distribution (infinite dimensional Dirichlet distribution) is placed on the parameters. When used in practice however, only a few of the components dominate, which results in a finite number of mixtures [67].

As introduced by Ferguson in [68], the DP is a probability measure on a space of probability measures. Assume that, similar to before, the observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, which are exchangeable, (i.e. drawn independently from some unknown distribution) are drawn from some distributions. The distribution which $\boldsymbol{x}_i$ is drawn from is represented by a component of the form $f(\boldsymbol{\phi}_i)$, where $\boldsymbol{\phi}_i$ is the set of parameters of the corresponding mixture, e.g. $\boldsymbol{\phi}_i = (\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ if f is Gaussian. The prior which the parameters are drawn from is drawn from a DP, defined by a base distribution $\boldsymbol{G}_0$, and a positive scalar α, known as the concentration parameter. The model described is

$$\begin{aligned} \boldsymbol{x}_i \mid \boldsymbol{\phi}_i &\sim f(\boldsymbol{\phi}_i) \\ \boldsymbol{\phi}_i \mid \boldsymbol{G} &\sim \boldsymbol{G} \\ \boldsymbol{G} &\sim \mathrm{DP}(\boldsymbol{G}_0, \alpha) \end{aligned} \tag{4.30}$$

As demonstrated by Sethuraman [69], a constructive representation of $\boldsymbol{G}$ can be written as

$$\boldsymbol{G} = \sum_{j=1}^{\infty} \pi_j \delta_{\boldsymbol{\phi}_j} \tag{4.31}$$

where

$$\begin{aligned} \boldsymbol{\phi}_j &\sim \boldsymbol{G}_0 \\ \pi_j &\sim \beta_k \Pi_{m=1}^{k-1} (1 - \beta_m) \\ \beta_m &\sim \mathcal{B}(1, \alpha) \end{aligned} \tag{4.32}$$

where $\mathcal{B}$ is the Beta distribution, and $\delta_{\boldsymbol{\phi}_j}$ refers to the Dirac delta measure located at $\boldsymbol{\phi}_j$ [67]. This shows an important property of $\boldsymbol{G}$, that it is discrete. As a result there is a positive probability that two $\boldsymbol{\phi}_j$'s will have the same value, and therefore $\boldsymbol{G}$ has a clustering property, effectively putting the data into a finite number of mixtures. Small values of α makes $\boldsymbol{G}$, the samples drawn from the DP, to be composed of small number of different samples with large weights. On the other hand, for larger values of α, the samples are likely to be distinct, making $\boldsymbol{G}$ look like the base distribution $\boldsymbol{G}_0$ [65, 70].

An equivalent representation of a DP mixture model can be constructed by taking the limit of the number of parametric mixture models K approaching infinity [66,67]. Assuming that the mixtures are Gaussian, the finite Gaussian mixture model of N samples, $\boldsymbol{x} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i, \ldots \boldsymbol{x}_N\}$, is the same as equation (4.1)

$$p(\boldsymbol{x}_i \mid \boldsymbol{\mu}_{1:K}, \boldsymbol{S}_{1:K}, \pi_{1:K}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_k, {\boldsymbol{S}_k}^{-1}) \tag{4.33}$$

where $\boldsymbol{S}_k$ is the inverse of the covariance referred to as the *precision*. Since the model identifier vector $\boldsymbol{Z}$ is unknown, for each observation we define a variable c_i, that represents the mixture that the observation currently belongs to. Furthermore a joint prior distribution $\boldsymbol{G}_0$ is placed over the mixture parameters $(\boldsymbol{\mu}, \boldsymbol{S})$. The finite model is therefore

$$\begin{aligned} \boldsymbol{x}_i \mid c_i, \Phi &\sim \mathcal{N}(\boldsymbol{\mu}_{c_i}, {\boldsymbol{S}_{c_i}}^{-1}) \\ c_i \mid \boldsymbol{\pi} &\sim \text{Discrete}(\pi_1, \ldots, \pi_K) \\ (\boldsymbol{\mu}_k, \boldsymbol{S}_k) &\sim \boldsymbol{G}_0 \\ \boldsymbol{\pi} \mid \alpha &\sim \text{Dir}(\alpha/K, \ldots, \alpha/K) \end{aligned} \tag{4.34}$$

where Dir() is the dirichlet distribution. In the DP mixture models, priors are placed on the parameters that are to be found, which can then be inferred by performing Gibbs sampling. In Gibbs sampling, each parameter to be inferred is sampled from its conditional distribution given all the other current parameters, which converge to the true conditional distributions on the parameters as the number of samples goes to infinity [71]. In the case of the indicator variables, Gibbs sampling is used to update every c_i from its conditional distribution given all the other indicator variables [66]. The joint prior on the indicator variables $\boldsymbol{c} = \{c_1 \ldots c_N\}$ given $\boldsymbol{\pi}$ is

$$p(\boldsymbol{c} \mid \boldsymbol{\pi}) = \prod_{k=1}^{K} \pi_k^{N_k} \tag{4.35}$$

where $N_k \in \mathbb{Z}$ is the number of observations in mixture component k. Integrating out the mixture weights gives the distribution on $\boldsymbol{c} = \{c_1 \dots c_N\}$ as

$$\begin{aligned} p(\boldsymbol{c} \mid \alpha) &= \int p(\boldsymbol{c} \mid \boldsymbol{\pi}, \alpha) p(\boldsymbol{\pi} \mid \alpha) d\boldsymbol{\pi} \\ &= \frac{\Gamma(\alpha)}{\Gamma(N+\alpha)} \prod_{k=1}^{K} \frac{\Gamma(N_k + \alpha/K)}{\Gamma(\alpha/K)} \end{aligned} \tag{4.36}$$

The conditional distribution for each individual indicator can be derived by fixing all the indicators but a single indicator variable (refer to page 3 of [67] for details), which gives

$$p(c_i = k \mid \boldsymbol{c}_{-i}, \alpha) = \frac{N_{-i,k} + \alpha/K}{N - 1 + \alpha} \tag{4.37}$$

where subscript $-i$ indicates all indices except for i. This means that if mixture component k has N_k members, and $\boldsymbol{x}_i$ currently belongs to mixture component k, then $N_{-i,k} = N_k - 1$, otherwise $N_{-i,k} = N_k$. The infinite DP mixture model is constructed by taking the limit as $K \to \infty$, which gives the conditional prior of the indicator variable c_i as

$$p(c_i = k \mid \boldsymbol{c}_{-i}, \alpha) = \frac{N_{-i,k}}{N - 1 + \alpha} \tag{4.38}$$

for the components with $N_{-i,k} > 0$, i.e. mixtures that have observation points associated with them. Since observation $\boldsymbol{x}_i$ only belongs to one components at any given time, the summation of the probabilities of $\boldsymbol{x}_i$ belonging to every mixture component that has observations associated with them is $\frac{N-1}{N-1+\alpha}$. Therefore, the probability of $\boldsymbol{x}_i$ belonging to all the other infinitely possible components that do not have any observations associated with them combine to $\frac{\alpha}{N-1+\alpha}$. In summary, the conditional prior probabilities of an observation point belonging to each component are

$$p(c_i = k \mid \boldsymbol{c}_{-i}, \alpha) = \begin{cases} \frac{N_{-i,k}}{N-1+\alpha} & k \text{ represented} \\ \\ \frac{\alpha}{N-1+\alpha} & k \text{ unrepresented} \end{cases} \tag{4.39}$$

To obtain the conditional posterior distribution of the indicators, the prior above is combined with the likelihood. The likelihood of the unrepresented components is obtained by integrating over the prior of them which gives

$$p(c_i = k \mid \boldsymbol{c}_{-i}, \boldsymbol{x}_i, \alpha, \boldsymbol{\phi}_{1:K}) \propto \begin{cases} p(c_i = k \mid \boldsymbol{c}_{-i}, \alpha)\ p(\boldsymbol{x}_i \mid \boldsymbol{\phi}_i) & k \text{ represented} \\ p(c_i = k \mid \boldsymbol{c}_{-i}, \alpha) \int p(\boldsymbol{x}_i \mid \boldsymbol{\phi}_*) d\boldsymbol{G}_0(\boldsymbol{\phi}_*) & k \text{ unrepresented} \end{cases} \tag{4.40}$$

$$= \begin{cases} \frac{N_{-i,k}}{N-1+\alpha}\ p(\boldsymbol{x}_i \mid \boldsymbol{\phi}_i) & k \text{ represented} \\ \frac{\alpha}{N-1+\alpha} \int p(\boldsymbol{x}_i \mid \boldsymbol{\phi}_*) d\boldsymbol{G}_0(\boldsymbol{\phi}_*) & k \text{ unrepresented} \end{cases}$$

With the above conditional probabilities, conventional Gibbs sampling is only feasible if one can compute the integral $\int p(\boldsymbol{y}_i \mid \boldsymbol{\phi}_*) d\boldsymbol{G}_0(\boldsymbol{\phi}_*)$. This is generally the case when $\boldsymbol{G}_0$ is the conjugate prior to the likelihood [67] (conjugacy of the joint prior distribution of parameters, i.e. the mean and the precision etc., to the likelihood [65]). In [67] Neal discusses various ways Gibbs sampling can still be performed when using both conjugate and non-conjugate priors. In section A.4 we list the details of the sampling process of a DP for all the parameters as suggested in [66, 67].

Although using conjugate priors in the DP mixture models make the integral in equation (4.40) tractable and Gibbs sampling straightforward to implement, the priors of the mean and the covariance of the mixtures become dependent on each other which can result in some unappealing properties, and is usually done only for mathematical and practical convenience [72]. On the other hand choosing non-conjugate priors can increase the computational cost. [65] discusses this at length and compares the computational efficiency and the modeling performance when conjugate and non-conjugate priors are used, and suggests that using conditionally conjugate priors can improve the modeling with a minor increase in the computation cost.

Despite its mathematical elegance, the drawback of the DP approach is that at every iteration, the Gibbs sampling is performed on the indicators of each observation. This becomes impractical with the increase of the number of observations. Furthermore, sampling the indicators means that the observations are constantly moved around into different mixtures, and although this fluctuation decreases considerably after a number of iterations, two subsequent iterations can have very different representations, which makes it difficult to decide which representation should be taken as the final result.

4.1.3.1 DP-means

A special case of the DP called DP-means outlined in Table 4.1 is derived in [64], where the resulting algorithm behaves very similar to the K-means algorithm, except that new mixture components can be added when needed. This is inspired by the derivation of the K-means algorithm from the EM algorithm as shown in section 4.1.2.1, where the covariance is assumed to be $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I}_D$, and as $\sigma^2 \to 0$, the responsibilities of an observation belonging to all mixture components go to zero except for the component which is closest to the observation. In the same manner, for the derivation of DP-means it is assumed that the covariance matrices in the DP mixture are spherical in the form of $\sigma^2 \boldsymbol{I}_D$. This essentially means that $\boldsymbol{G}_0$ is only the prior on the mean, which is picked to be $\boldsymbol{\mu} \sim \mathcal{N}(\mathbf{0}, \rho \boldsymbol{I}_D)$ for some value of ρ. Furthermore it is assumed that α is a function of σ^2 and ρ in the form of $\alpha = (1 + \frac{\rho}{\sigma^2})^{D/2} \exp(-\frac{\lambda}{2\sigma^2})$ for some λ. It is then shown that the conditional probabilities of an observation point $\boldsymbol{x}_i$ belonging to K existing components and a new component in equation (4.40) is dominated by the smallest values between $\{||\boldsymbol{x}_i - \boldsymbol{\mu}_1||^2, \ldots, ||\boldsymbol{x}_i - \boldsymbol{\mu}_K||^2, \lambda\}$. This means that much like the K-means algorithm, observation $\boldsymbol{x}_i$ is assigned to the closest component, unless that closest component is more than a distance λ away. In that case a new component is created with $\boldsymbol{x}_i$ as its new member. By taking the limit of σ^2 to zero, it is also shown that similar to the K-means algorithm, the mean of every component becomes the mean of its members. The algorithm converges when no observation is a distance λ away from all existing mixtures, and also much like the K-means, the center of the mixtures stop changing to within some pre-defined tolerance.

4.1.4 Other Mixture Model Methods

Part of the literature related to mixture models is focused on finding the right number of components by utilizing a relative quality measures to compare a finite set of models having different components. Two of the most commons measures are the Akaike information criterion (AIC) [73] and the Bayesian information criterion (BIC) [74]. These methods do not test the quality of the model in an absolute sense, and one would not know if the best candidate model produces poor representation if all the other models in the finite set are also poor. [75] provides a review of these comparative methods and evaluates the performance of each measure.

A related topic presented here arises in filtering with a mixture of Gaussians first shown in [76], and the ever-growing number of components (see section 5.1.3 for details), that requires a method of combining the components. [77] presented this in a similar setting of finding a Bayesian solution to tracking a target in a

DP-means Algorithm

1: initialize λ, set $K = 1$, $\boldsymbol{\mu}_1$ to the mean of the data.

2: set cluster indicator $c_i = 1$ of point $\boldsymbol{x}_i$ for $i = 1, \ldots, N$

3: repeat until convergence

3: for each point $\boldsymbol{x}_i$ in $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots \boldsymbol{x}_N\}$

4: compute $d_{ik} = ||\boldsymbol{x}_i - \boldsymbol{\mu}_k||^2$ for $k = 1, \ldots, K$

5: if $\min_k d_{ik} > \lambda$, set $K = K + 1$, $c_i = K$ and $\boldsymbol{\mu}_K = \boldsymbol{x}_i$

6: otherwise, set $c_i = \text{argmin}_k \, d_{ik}$

7: compute N_k, number of members of cluster k based on $c_1, \ldots, c_N$ and compute mean $\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{j:c_j=k} \boldsymbol{x}_j$, for $k = 1, \ldots, K$

8: end

Table 4.1: The steps of the DP-means algorithm [64].

random clutter, when the origin of the sensor measurements is uncertain, and proposed how the mixtures should be combined. This has led to a plethora of mixture reduction algorithms that address this problem by progressively combining similar mixtures that result in a reduced number of components. [78] used the Integral Squared Error (ISE) as the similarity measure to combine mixtures in a greedy fashion. In [79], an upper bound on the Kullback-Leiber (KL) divergence between two un-merged mixtures is instead used as the pair-selection criteria. [80] uses the result of the method in [79] as the initialization step of a K-means algorithm between the mixture components in order to group different components together, and represents the grouped mixtures by the center of that group. A good review of such reduction methods is given in [81], the problem with these approaches however is that the lower limit on the number of components is assumed to be known.

The vanilla version of the EM and the K-means algorithms are known to suffer from initialization and can get stuck in local optima. A common time-consuming approach is to use a multiple random starts and take the representation with the highest likelihood. [82] uses a greedy EM that iteratively adds components up to a known K, in such a way that the log-likelihood is increased with the addition of every component. This is shown to outperform the EM with pre-determined number of components. A similar method presented in [83] uses the K-means algorithm to iteratively add components up to K components, and suggests ways for speeding up such execution. [84] proposes an EM-based algorithm using split and merge operations to

escape from local optima of the likelihood. In [85] a modified EM algorithm is presented that has a property of annihilating weak components. This approach starts with a relatively high number of components, where the strong components attract more members and the weaker ones are killed off, until convergence to a remaining number of components.

Approximate inference approaches rely on the assumption that the distribution between the latent model indicator and the parameters can be written in factored form as

$$p(\boldsymbol{\Theta}, \boldsymbol{Z}) \approx q(\boldsymbol{\Theta}) \prod_{i=1}^{N} q_i(z_i) \tag{4.41}$$

this is known as the variational Bayes (VB) EM method based on the work in [86–88], where much like the EM algorithm, one alternates between updating $q_i(z_i)$ (variational E-step) and $q(\boldsymbol{\Theta})$ (variational M-step) [62]. In the absence of the knowledge of the number of components K, a similar approach to [85] mentioned above can be used, where starting with a large number of components, the weaker mixtures get pruned and only the stronger ones remain at the end.

In the next section we propose EM$^+$, an algorithm that takes inspiration from both the DP and the EM approach. EM$^+$ has the flexibility of the DP without its high computational cost, and at the same time enjoys the simplicity of the EM without the limitation of the prior knowledge of the number of components. Furthermore, by progressively adding and removing components, EM$^+$ avoids getting stuck in local optima.

4.2 The Proposed EM$^+$ Algorithm

As mentioned before, the standard EM algorithm requires a prior knowledge of the number of components in the mixture. Our approach can be viewed as an extension of the EM algorithm capable of automatically figuring out the number of components in the mixture. Several EM algorithms have been proposed previously with this capability, for example [82, 84]. In all of these approaches, it is assumed that the model has K components on which the standard EM algorithm can be applied and in addition mechanisms to increase or decrease K are provided. The main differentiating feature in our approach is that during the course of the algorithm, we rather focus on estimating the parameters $\boldsymbol{\Theta}_{1:K}$ of K components of the model without restricting at any time the actual number of components K_a. In this perspective, the parameters of the remaining components are considered as *nuisance* parameters and are integrated-out from the likelihood. More specifically, the so-called *Integrated Likelihood* [89] for a single sample is obtained as follows:

$$p(\boldsymbol{x}_j \mid \boldsymbol{\pi}_{1:K}, \boldsymbol{\mu}_{1:K}, \boldsymbol{\Sigma}_{1:K}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_j \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) + \sum_{k=K+1}^{K_a} \pi_k \int \mathcal{N}(\boldsymbol{x}_j \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) p(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) d\boldsymbol{\mu}_k d\boldsymbol{\Sigma}_k \quad (4.42)$$

Letting the parameter prior $p(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ be the Normal-Inverse-Wishart conjugate distribution (4.14) allows the explicit computation of the integral in (4.42), namely

$$\int \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) \mathrm{NIW}(\boldsymbol{\mu}, \boldsymbol{\Sigma} \mid \boldsymbol{m}_0, \kappa_0, \nu_0, \boldsymbol{S}_0) d\boldsymbol{\mu} d\boldsymbol{\Sigma} = \mathcal{T}(\boldsymbol{x} \mid \boldsymbol{m}_0, \frac{\kappa_0 + 1}{\kappa_0(\nu_0 - D + 1)} \boldsymbol{S}_0, \nu_0 - D + 1) \quad (4.43)$$

where

$$\mathcal{T}(\boldsymbol{x} \mid \boldsymbol{\mu}_{\mathcal{T}}, \boldsymbol{\Sigma}_{\mathcal{T}}, \nu) = \frac{\Gamma(\frac{\nu+D}{2})}{\Gamma(\frac{\nu}{2})} \mid \pi\nu\boldsymbol{\Sigma}_{\mathcal{T}} \mid^{-\frac{1}{2}} \left[1 + (\boldsymbol{x} - \boldsymbol{\mu}_{\mathcal{T}})^T (\nu\boldsymbol{\Sigma}_{\mathcal{T}})^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_{\mathcal{T}})\right]^{-\frac{\nu+D}{2}} \quad (4.44)$$

is the multivariate $\mathcal{T}$-distribution [62]. Therefore, inserting (4.43) in (4.42) and defining

$$\pi_+ = \sum_{k=K+1}^{K_a} \pi_k = 1 - \sum_{k=1}^{K} \pi_k \quad (4.45)$$

yields

$$p(\boldsymbol{x}_j \mid \boldsymbol{\pi}_{1:K}, \boldsymbol{\mu}_{1:K}, \boldsymbol{\Sigma}_{1:K}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_j \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) + \pi_+ \mathcal{T}(\boldsymbol{x}_j \mid \boldsymbol{m}_0, \frac{\kappa_0 + 1}{\kappa_0(\nu_0 - D + 1)} \boldsymbol{S}_0, \nu_0 - D + 1). \quad (4.46)$$

4.2.1 Parameter Estimation in the EM$^+$ Algorithm

The parameters $\boldsymbol{\Theta}_{1:K} = \{\pi_{1:K}, \boldsymbol{\mu}_{1:K}, \boldsymbol{\Sigma}_{1:K}\}$ in (4.46) with $\pi_{1:K}$ satisfying (4.45) rather than (4.3) can be estimated by using the standard (EM-MAP) algorithm. However, the indicated mixture consists of the normal components $z = 1, \ldots, K$ with parameters to be estimated *and* the $z = K^+$ component, which is the fixed $\mathcal{T}$-distribution component representing undeveloped clusters. Having to consider the K^+ cluster requires updating the prior for the cluster probabilities $\pi_{1:K}$ as follows:

$$p(\pi_{1:K} \mid \boldsymbol{\Psi}) = \mathrm{Dir}(\pi_{1:K}, \pi_+ \mid \alpha_{1:K}, \alpha_+), \quad (4.47)$$

that is, α_+ must be included in the hyperparameters $\boldsymbol{\Psi}$. Then in the expectation step, by considering $\bar{R}_{i,k}$ defined by (4.17) for $k = 1, \ldots, K$ as before, and

$$\bar{R}_{iK^+} \equiv\ p(\boldsymbol{x}_i \mid z_i = K^+, \boldsymbol{\Theta}(t)) \cdot p(z_i = K^+ \mid \boldsymbol{\Theta}(t)) = \mathcal{T}(\boldsymbol{x}_j \mid \boldsymbol{m}_0, \frac{\kappa_0 + 1}{\kappa_0(\nu_0 - D + 1)}\boldsymbol{S}_0, \nu_0 - D + 1) \cdot \pi_+, \quad (4.48)$$

the responsibilities $R_{i,k}$ are obtained for $k = 1, \ldots, K, K^+$ from

$$R_{ik} \equiv p(z_i = k \mid \boldsymbol{X}, \boldsymbol{\Theta}(t)) = \frac{\bar{R}_{ik}}{\sum_{k=1}^{K} \bar{R}_{ik} + \bar{R}_{iK^+}} = \frac{\bar{R}_{ik}}{p(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K})}. \quad (4.49)$$

Furthermore, the maximization step results in the following equations for the cluster probabilities

$$\pi_k = \frac{N_k + \alpha_k - 1}{N + \sum_{k=1}^{K} \alpha_k + \alpha_+ - K - 1}, \quad k = 1, \ldots, K \quad (4.50)$$

$$\pi_+ = \frac{N_{K^+} + \alpha_+ - 1}{N + \sum_{k=1}^{K} \alpha_k + \alpha_+ - K - 1} \quad (4.51)$$

where N_k is defined again by (4.19) and

$$N_{K^+} = \sum_{i=1}^{N} R_{iK^+} \cdot \boldsymbol{x}_i. \quad (4.52)$$

In the course of the EM$^+$ algorithm, the α_k's are re-estimated under the constraints $\alpha_k \geq 1$ so that $N_k + \alpha_k - 1 \geq 0$ and (4.50) and (4.51) are applicable. Therefore, a component probability π_k can become zero, indicating that this component should be removed from the mixture, only asymptotically. Finally, (4.21) to (4.24) for updating the parameters $\boldsymbol{\mu}_{1:K}, \boldsymbol{\Sigma}_{1:K}$ remain unchanged.

4.2.2 Addition of a new component

The standard EM algorithm increases the likelihood at each iteration. The proposed algorithm is initialized with $K = 0$ and at each iteration a new component is considered as long as $\pi_+ > 0$ and is added to the mixture only if the objective function maximized in (4.6) increases. The addition of new components is done at the expense of the K^+ component and each time a new component is added, a smaller π_+ results. Actually, π_+ can decrease by redistributing the probability it represents to existing clusters during the EM step and without adding new components. The final number of mixture components is implicitly determined as the K for which the estimated parameters give $\pi_+ \approx 0$.

Let us assume that the current mixture consists of K components (besides the K^+ component) and that $\pi_+ > 0$. We consider adding a component $K+1$ and thus obtain a new mixture

$$\begin{aligned} p(\boldsymbol{x} \mid \boldsymbol{\Theta}_{1:K+1}) &= \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) + \pi_{K+1} \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1}) + (\pi_+ - \pi_{K+1}) \mathcal{T}(\boldsymbol{x} \mid \boldsymbol{\lambda}) \\ &= \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) + \pi_+ \mathcal{T}(\boldsymbol{x} \mid \boldsymbol{\lambda}) + \pi_{K+1} \left(\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1}) - \mathcal{T}(\boldsymbol{x} \mid \boldsymbol{\lambda}) \right) \\ &= p(\boldsymbol{x} \mid \boldsymbol{\Theta}_{1:K}) + \pi_{K+1} \left(\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1}) - \mathcal{T}(\boldsymbol{x} \mid \boldsymbol{\lambda}) \right) \end{aligned} \tag{4.53}$$

where

$$\boldsymbol{\lambda} \equiv \left\{ \boldsymbol{m}_0, \frac{\kappa_0 + 1}{\kappa_0(\nu_0 - D + 1)} \boldsymbol{S}_0, \nu_0 - D + 1 \right\} \tag{4.54}$$

summarizes the hyperparameters of the $\mathcal{T}$ distribution in (4.46). In (4.53), $\boldsymbol{\Theta}_{1:K} \equiv \{\pi_{1:K},\ \boldsymbol{\mu}_{1:K},\ \boldsymbol{\Sigma}_{1:K}\}$ are fixed at their current values and the objective is to find $\pi_{K+1} \in (0,\ \pi_+]$, $\boldsymbol{\mu}_{K+1}$, and $\boldsymbol{\Sigma}_{K+1}$ so as to maximize the MAP log-likelihood

$$\begin{aligned} F_{K+1}(\boldsymbol{\Theta}_{1:K+1}; \boldsymbol{\Psi}) &= \sum_{i=1}^{N} \log p(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K+1}) + \sum_{k=1}^{K+1} \log \frac{\mathrm{NIW}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k \mid \boldsymbol{\lambda}_0)}{\mathrm{NIW}(\boldsymbol{\mu}_k^*, \boldsymbol{\Sigma}_k^* \mid \boldsymbol{\lambda}_0)} + \\ &\qquad \log \mathrm{Dir}(\pi_{1:K}, \pi_{K+1}, \pi_+ - \pi_{K+1} \mid \alpha_{1:K}, \alpha_{K+1}, \alpha_+) + \eta_{K+1} \end{aligned} \tag{4.55}$$

where $\boldsymbol{\lambda}_0 \equiv \{\boldsymbol{m}_0, \kappa_0, \nu_0, \boldsymbol{S}_0\}$, $\boldsymbol{\mu}_k^* = \boldsymbol{m}_0$ and $\boldsymbol{\Sigma}_0^* = \boldsymbol{S}_0/(\nu_0 + D + 2)$ are the mode of the NIW distribution with parameters $\boldsymbol{\lambda}_0$, and η_{K+1} is a constant, the role of which will be discussed in the following. Of course, an additional hyperparameter α_{K+1} needs to included in $\boldsymbol{\Psi}$.

4.2.2.1 Continuity of the Objective Function

When a new component is added (or as discussed later deleted), structural changes occur in the objective function since the number of variables and parameters changes. Therefore, it is important to establish conditions under which such transitions result in an overall continuous objective function. To this end, we observe that the $K+1$-model is essentially equivalent to the K-model if $\pi_{K+1} = 0$. Therefore, we require that

$$\max_{\boldsymbol{\mu}_{K+1},\boldsymbol{\Sigma}_{K+1}} F_{K+1}(\boldsymbol{\Theta}_{1:K}, \pi_{K+1}=0, \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1}) = F_K(\boldsymbol{\Theta}_{1:K}). \tag{4.56}$$

From (4.13), we can write

$$\begin{aligned}\log \mathrm{D}ir(\pi_{1:K}, \pi_{K+1}, \pi_+ - \pi_{K+1} \mid \alpha_{1:K}, \alpha_{K+1}, \alpha_+) &= \log \mathrm{D}ir(\pi_{1:K}, \pi_+ \mid \alpha_{1:K}, \alpha_+) + \\ \log \frac{\Gamma(\sum_{k=1}^{K+1}\alpha_k + \alpha_+)}{\Gamma(\sum_{k=1}^{K}\alpha_k + \alpha_+)\cdot\Gamma(\alpha_{K+1})} &+ (\alpha_{K+1}-1)\log\pi_{K+1} + (\alpha_+ - 1)\log\frac{\pi_+ - \pi_{K+1}}{\pi_+}.\end{aligned} \tag{4.57}$$

Furthermore, using (4.53) and (4.57) in (4.55) yields

$$\begin{aligned}F_{K+1}(\boldsymbol{\Theta}_{1:K+1}) &= F_K(\boldsymbol{\Theta}_{1:K}) + \sum_{i=1}^{N}\log\left(1 + \pi_{K+1}\frac{\mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1}) - \mathcal{T}(\boldsymbol{x}_i \mid \boldsymbol{\lambda})}{p(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K})}\right) + \\ &\log\frac{\mathrm{NIW}(\boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1} \mid \boldsymbol{\lambda}_0)}{\mathrm{NIW}(\boldsymbol{\mu}^*_{K+1}, \boldsymbol{\Sigma}^*_{K+1} \mid \boldsymbol{\lambda}_0)} + \log\frac{\Gamma(\sum_{k=1}^{K+1}\alpha_k + \alpha_+)}{\Gamma(\sum_{k=1}^{K}\alpha_k + \alpha_+)\cdot\Gamma(\alpha_{K+1})} + \\ &(\alpha_{K+1}-1)\log\pi_{K+1} + (\alpha_+ - 1)\log\frac{\pi_+ - \pi_{K+1}}{\pi_+} + (\eta_{K+1} - \eta_K).\end{aligned} \tag{4.58}$$

From (4.58), it is clear that upon the introduction of a new component $K+1$, we must take $\alpha_{K+1} = 1$, otherwise F_{K+1} is not well defined for $\pi_{K+1} = 0$. However, the α_k's of existing components are not restricted to remain equal to 1, since as it is discussed later, they are re-estimated during the course of the algorithm. The hyperparameter α_+ is initialized at a value $\alpha_{0+} \geq 1$ and is optimized along with the $\alpha_{1:K}$ during the course of the algorithm. With these choices, we obtain from (4.58)

$$\max_{\boldsymbol{\mu}_{K+1},\boldsymbol{\Sigma}_{K+1}} F_{K+1}(\boldsymbol{\Theta}_{1:K}, \pi_{K+1}=0, \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1}) = F_K(\boldsymbol{\Theta}_{1:K}) + \log\frac{\Gamma(\sum_{k=1}^{K}\alpha_k + \alpha_+ + 1)}{\Gamma(\sum_{k=1}^{K}\alpha_k + \alpha_+)} + \eta_{K+1} - \eta_K \tag{4.59}$$

and for (4.56) to hold, we take

$$\eta_{K+1} = \eta_K - \log\left(\sum_{k=1}^{K}\alpha_k + \alpha_+\right). \tag{4.60}$$

Recursion (4.60) is initialized with $\eta_0 = 0$.

4.2.2.2 Optimization of the Objective Function

We can optimize F_{K+1} over π_{K+1}, $\boldsymbol{\mu}_{K+1}$, and $\boldsymbol{\Sigma}_{K+1}$ using standard numerical optimization algorithms, Alternatively, we can view F_{K+1} as the MAP objective function of a mixture with $K+2$ components (components from 1 to $K+1$ and the K^+ component), of which only component $K+1$ is variable and the

component probabilities $\pi_{1:K}$ of the first K components are also fixed. We can then employ the (standard) EM algorithm as follows. We first write,

$$
\begin{aligned}
&F_{K+1}(\boldsymbol{\Theta}_{1:K+1}) = \\
&\quad = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) + \pi_{K+1} \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1}) + (\pi_+ - \pi_{K+1}) \mathcal{T}(\boldsymbol{x}_i \mid \boldsymbol{\lambda}) \right) + \\
&\qquad \sum_{k=1}^{K+1} \log \mathrm{NIW}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k \mid \boldsymbol{\lambda}_0) + \sum_{k=1}^{K+1} (\alpha_k - 1) \log \pi_k + (\alpha_+ - 1) \log(\pi_+ - \pi_{K+1}) + \ const.
\end{aligned} \tag{4.61}
$$

In (4.61), the log-likelihood of the mixture appears on the first line while the second line contains the terms due to the parameter priors. Considering the full log-likelihood and performing the Expectation step yields the auxiliary function

$$
\begin{aligned}
&Q(\boldsymbol{\Theta}_{1:K+1}, \boldsymbol{\Theta}_{1:K+1}(t)) = \\
&\quad = \sum_{i=1}^{N} \sum_{k=1}^{K+1} R_{ik} \log \{ \pi_k \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \} + R_{i+} \log \{ (\pi_+ - \pi_{K+1}) \mathcal{T}(\boldsymbol{x}_i \mid \boldsymbol{\lambda}) \} \\
&\qquad \sum_{k=1}^{K+1} \log \mathrm{NIW}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k \mid \boldsymbol{\lambda}_0) + \sum_{k=1}^{K+1} (\alpha_k - 1) \log \pi_k + (\alpha_+ - 1) \log(\pi_+ - \pi_{K+1})
\end{aligned} \tag{4.62}
$$

with the responsibilities computed as previously from

$$
R_{ik} = \frac{\bar{R}_{ik}}{\sum_{k=1}^{K+1} \bar{R}_{ik} + \bar{R}_{iK^+}} \tag{4.63}
$$

where $\bar{R}_{i,k}$ are defined by (4.17) for $k = 1, \ldots, K+1$, while for $k = K^+$ from

$$
\bar{R}_{iK^+} = (\pi_+ - \pi_{K+1}) \mathcal{T}(\boldsymbol{x}_i \mid \boldsymbol{\lambda}). \tag{4.64}
$$

We remark that $R_{i,K+1}$ and $R_{i,+}$ depend on the current estimate for $\pi_{K+1} \equiv \pi_{K+1}(t)$ and considered constant for the maximization step. Next, to maximize (4.62) with respect to π_{K+1}, we set

$$
\frac{\partial Q}{\partial \pi_{K+1}} = \sum_{i=1}^{N} \left(\frac{R_{i,K+1}}{\pi_{K+1}} - \frac{R_{i,+}}{\pi_+ - \pi_{K+1}} \right) + \frac{\alpha_{K+1} - 1}{\pi_{K+1}} - \frac{\alpha_+ - 1}{\pi_+ - \pi_{K+1}} = 0 \tag{4.65}
$$

and obtain the next iterate

$$\pi_{K+1} = \frac{(N_{K+1} + \alpha_{K+1} - 1) \cdot \pi_+}{N_{K+1} + N_+ + (\alpha_{K+1} - 1) + (\alpha_+ - 1)} \tag{4.66}$$

where $N_k = \sum_{i=1}^{N} R_{i,k}$ and $N_+ = \sum_{i=1}^{N} R_{i,+}$ as before. Given the previous choices $\alpha_{K+1} = 1$ and $\alpha_+ = 1$, (4.66) simplifies to

$$\pi_{K+1} = \frac{N_{K+1} \cdot \pi_+}{N_{K+1} + N_+}. \tag{4.67}$$

We observe that $\pi_{K+1} < \pi+$ as $N_+ > 0$ for $\pi_+ > 0$ and thus π_+ can only asymptotically converge to zero. The optimization of (4.62) with respect to $\boldsymbol{\mu}_{K+1}$ and $\boldsymbol{\Sigma}_{K+1}$ remains the same as in the general case, therefore, formulas (4.21) to (4.24) for $k = K + 1$ give the next iterates for $\boldsymbol{\mu}_{K+1}$ and $\boldsymbol{\Sigma}_{K+1}$.

Next, we discuss the initialization of π_{K+1}, $\boldsymbol{\mu}_{K+1}$, and $\boldsymbol{\Sigma}_{K+1}$ for the application of the above EM procedure. Let us assume that $\boldsymbol{\mu}_{K+1}$ and $\boldsymbol{\Sigma}_{K+1}$ have been selected and focus on picking π_{K+1} so that the F_{K+1} is larger than for $\pi_{K+1} = 0$. The first and second derivatives of (4.61) with respect to π_{K+1} can be easily obtained as follows

$$\frac{\partial F_{K+1}}{\partial \pi_{K+1}} = \sum_{i=1}^{N} \frac{\delta(\boldsymbol{x}_i; \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1})}{p(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K}) + \pi_{K+1}\delta(\boldsymbol{x}_i; \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1})} + \frac{\alpha_{K+1} - 1}{\pi_{K+1}} - \frac{\alpha_+ - 1}{\pi_+ - \pi_{K+1}} \tag{4.68}$$

$$\frac{\partial^2 F_{K+1}}{\partial \pi_{K+1}^2} = \sum_{i=1}^{N} \frac{-\delta(\boldsymbol{x}_i; \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1})^2}{[p(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K}) + \pi_{K+1}\delta(\boldsymbol{x}_i; \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1})]^2} - \frac{(\alpha_{K+1} - 1)}{\pi_{K+1}^2} - \frac{\alpha_+ - 1}{[\pi_+ - \pi_{K+1}]^2} \tag{4.69}$$

where we defined

$$\delta(\boldsymbol{x}_i; \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1}) \equiv \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1}) - \mathcal{T}(\boldsymbol{x}_i \mid \boldsymbol{\lambda}). \tag{4.70}$$

Equation (4.69) implies that F_{K+1} is a concave function of $\pi_{K+1} \in (0,\ \pi_+)$ for $\alpha_{K+1} \geq 1$ and $\alpha_+ \geq 1$ and in particular with our previous choice $\alpha_{K+1} = 1$, the π_{K+1} maximizing F_{K+1} is greater than zero if and only if

$$\begin{aligned} \frac{\partial F_{K+1}}{\partial \pi_{K+1}} \mid_{\pi_{K+1}=0} &= \sum_{i=1}^{N} \frac{\delta(\boldsymbol{x}_i; \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1})}{p(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K})} - \frac{\alpha_+ - 1}{\pi_+} \\ &= \sum_{i=1}^{N} \frac{\mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1})}{p(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K})} - \sum_{i=1}^{N} \frac{\mathcal{T}(\boldsymbol{x}_i \mid \boldsymbol{\lambda})}{p(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K})} - \frac{\alpha_+ - 1}{\pi_+} > 0 \end{aligned} \tag{4.71}$$

The last two terms in (4.71) do not depend on $\boldsymbol{\mu}_{K+1}$ and $\boldsymbol{\Sigma}_{K+1}$, so we seek to maximize the first term to achieve $\partial F_{K+1}/\partial \pi_{K+1} \mid_{\pi_{K+1}=0} > 0$. The $K + 1$ mixture component is introduced to accommodate samples

that are not represented well by the existing K components. Therefore, $\mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1})$ is typically concentrated about such a sample $\boldsymbol{x}_i^*$. Then, we can assume that $\mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1})$ is small for $\boldsymbol{x}_i \neq \boldsymbol{x}_i^*$ and we can approximate:

$$\frac{\partial F_{K+1}}{\partial \pi_{K+1}} \mid_{\pi_{K+1}=0} \approx \frac{constant}{p(\boldsymbol{x}_i^* \mid \boldsymbol{\Theta}_{1:K})} - \sum_{i=1}^{N} \frac{\mathcal{T}(\boldsymbol{x}_i \mid \boldsymbol{\lambda})}{p(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K})} - \frac{\alpha_+ - 1}{\pi_+} \tag{4.72}$$

Therefore to maximize $\partial F_{K+1}/\partial \pi_{K+1} \mid_{\pi_{K+1}=0}$, (4.72) suggests selecting x_i^* such that $p(\boldsymbol{x}_i^* \mid \boldsymbol{\Theta}_{1:K})$ is small. We found that minimizing $p(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K})$ does not always work well since, small $p(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K})$ may also indicate a sample from a low probability area and not just a sample not represented well by the existing K components. Instead, we select x_i^* by sampling the available samples with weights $1/p(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K})$. In this manner, samples from high probability areas, which however, are not well represented in the current mixture tend to be selected as desired. Nevertheless, should this process result in a component that does not increase F_{K+1}, this component is rejected and another attempt to add a new component is made in the next iteration. Since $\boldsymbol{x}_i^*$ is viewed as a sample from a new component $\mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1})$, we obtain our starting values for $\boldsymbol{\mu}_{K+1}$ and $\boldsymbol{\Sigma}_{K+1}$ as the mode of the posterior NIW distribution based on a the single sample $\boldsymbol{x}_i^*$ and the conjugate NIW$(\boldsymbol{\mu}, \boldsymbol{\Sigma} \mid \boldsymbol{\lambda}_0)$ prior given by (4.14). Namely,

$$\boldsymbol{\mu}_{K+1} = \frac{\boldsymbol{x}_i^* + \kappa_o \boldsymbol{m}_o}{\kappa_0 + 1} \tag{4.73}$$

$$\boldsymbol{\Sigma}_{K+1} = \frac{\boldsymbol{S}_0 + \frac{\kappa_0}{\kappa_0+1}(\boldsymbol{x}_i^* - \boldsymbol{m}_0)(\boldsymbol{x}_i^* - \boldsymbol{m}_0)^T}{\nu_o + D + 3}. \tag{4.74}$$

Finally, once $\boldsymbol{\mu}_{K+1}$ and $\boldsymbol{\Sigma}_{K+1}$ are given, the π_{K+1} maximizing the concave function F_{K+1} can be obtained by a one-dimensional search. However, we found that good results are obtained with using the starting value

$$\pi_{K+1} = \frac{\pi_+ \cdot \sum_{i=1}^{N} \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1})}{\sum_{i=1}^{N} \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_{K+1}, \boldsymbol{\Sigma}_{K+1}) + \sum_{i=1}^{N} \mathcal{T}(\boldsymbol{x}_i \mid \boldsymbol{\lambda})}, \tag{4.75}$$

which can be thought as the result of one iteration of (4.67) starting with the value $\pi_{K+1} = \pi_+/2$.

4.2.3 Deletion of weak components

During the course of the proposed algorithm, the probabilities of certain mixture components may become small and indeed go to zero, indicating that these components are being absorbed by stronger ones. The removal of a weak component should not affect the MAP log-likelihood in the limiting case of its probability tending to zero while at the same time resulting in a more parsimonious mixture.

The process of $\pi_k \to 0$ is promoted by the re-estimation of the hyperparameters $\alpha_{1:K}$ and α_+ as discussed in a following section. Indeed, should π_k become relatively small, the term $(\alpha_k - 1)\log \pi_k$ in F_K results in a small maximizing α_k and in case that $\alpha_k < 1$, the next maximization of F_{K+1} in (4.58) over π_k will result in $\pi_k = 0$. However, to avoid an unbounded optimal value for the objective function, we impose the constraints $\alpha_{1:K},\ \alpha_+ \geq 1$ during the estimation of these hyperparameters and as a result component probabilities can only asymptotically tend to zero.

Next, let us assume without loss of generality that $\pi_{K+1} \to 0$ and the $K+1$ component is to be deleted. From the maximization step of the EM algorithm and in particular (4.50), $\pi_{K+1} \to 0$ and the imposed constraint $\alpha_{K+1} \geq 0$ imply that $N_{K+1} \to 0$ and $\alpha_{K+1} \to 0$. Therefore, we are justified in arranging that $\alpha_{K+1} = 1$ when deleting cluster $K+1$. Also from (4.58), we conclude that as $\pi_{K+1} \to 0$ the corresponding component parameters $\boldsymbol{\mu}_{K+1} \to \boldsymbol{\mu}^*_{K+1}$ and $\boldsymbol{\Sigma}_{K+1} \to \boldsymbol{\Sigma}^*_{K+1}$ the mode of the NIW distribution. This is to maximize the third term in (4.58), which is the only term depending on $\boldsymbol{\mu}_{K+1}$ and $\boldsymbol{\Sigma}_{K+1}$ once $\pi_{K+1} = 0$.

Therefore, we can express (4.58) at the deletion of cluster $K+1$ as

$$F_K(\boldsymbol{\Theta}_{1:K}) = F_{K+1}(\boldsymbol{\Theta}_{1:K}, \pi_{K+1} = 0, \boldsymbol{\mu}^*_{K+1}, \boldsymbol{\Sigma}^*_{K+1}) - \log \frac{\Gamma(\sum_{k=1}^K \alpha_k + \alpha_+ + 1)}{\Gamma(\sum_{k=1}^K \alpha_k + \alpha_+)} - \eta_{K+1} + \eta_K. \tag{4.76}$$

The last three terms in the right-hand-side of (4.76) cancel by the choice of the recursion (4.60) and thus, the continuity of the objective function during the deletion of the $K+1$ component with $\pi_{K+1} = 0$ is maintained.

In practice, we remove the component $K+1$ when the conditions $\alpha_{K+1} = 1$ and $\pi_{K+1} \leq \pi_0$. Here, π_0 is chosen to be small relative to the highest probabilities among mixture components and possibly based on prior knowledge on the maximum number of components. Since when the $K+1$ component is deleted $\pi_{K+1} > 0$, albeit small, rather than $\pi_{K+1} = 0$, its probability must be distributed to the other components of the mixture, including the K^+ component. A simple approach is to normalize the probabilities of the remaining components by taking

$$\begin{aligned} \pi_{k,new} &= \frac{\pi_k}{\sum_{1:K} \pi_k + \pi_+}, \quad k = 1, \ldots, K \\ \pi_{k,new} &= \frac{\pi_+}{\sum_{1:K} \pi_k + \pi_+}. \end{aligned} \tag{4.77}$$

Alternatively, we may consider maximizing the MAP log-likelihood F_{K+1} in (4.55) with respect to the $\pi_{1:K}$ and under the constraint $\sum_{k=1}^{K} \pi_k = 1$, i.e. $\pi_{K+1} = 0$. This is possible since $\alpha_{K+1} = 1$ is assumed. However, this optimization requires an iterative approach such as the (standard) EM algorithm where the $\pi_{1:K}$ need to be initialized, for example, by (4.77). Since, during the subsequent iterations of the proposed algorithm, such steps are performed nevertheless and since the probability of deleted component is relatively small, we choose to use (4.77) for updating $\pi_{1:K}$. The deletion process is completed by discarding the parameters $\boldsymbol{\mu}_{K+1}$ and $\boldsymbol{\Sigma}_{K+1}$ of the deleted component. We remark that due to the approximation involved during a practical deletion, a (small) increase in the MAP log-likelihood may occur.

4.2.4 Deletion of the K^+ component

The K^+ component acts as a placeholder for yet undeveloped mixture components. Its probability π_+ is reduced each time a new component is added but π_+ may increase during the EM iteration and the deletion of a mixture component. The K^+ component represents all samples and typically it concedes "ownership" of sample points against newly introduced components that better represent the clusters of the mixture. We have observed that generally $\pi_+ \to 0$ and we conjecture that this is true as the number of sample points $N \to \infty$. From a practical perspective, the $K+$ component is eliminated when the condition $\pi_+ \leq \pi_0$ is met. Notice that $\alpha_+ \geq 1$ is always enforced and arguing as in the case of deleting mixture components, when the K^+ component is deleted we can assume that $\alpha_+ = 1$. The deletion of the K^+ component proceeds in the same manner as the deletion of any other mixture component. Namely, we normalize the probabilities of the mixture using the first equation in (4.77) with $\pi_+ = 0$, and eliminate the $\mathcal{T}$-distribution component from the mixture. The objective function F_K^- without the K^+ component relates to the objective function F_K with the K^+ component present as follows

$$F_K(\boldsymbol{\Theta}_{1:K}) = F_K^-(\boldsymbol{\Theta}_{1:K}) + \sum_{i=1}^{N} \log\left(1 + \pi_+ \frac{\mathcal{T}(\boldsymbol{x}_i \mid \boldsymbol{\lambda})}{p^-(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K})}\right) + \log \frac{\Gamma(\sum_{k=1}^{K} \alpha_k + 1)}{\Gamma(\sum_{k=1}^{K} \alpha_k)} + (\eta_K - \eta_K^-), \tag{4.78}$$

where $p^-(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K})$ is the mixture in (4.46) without the $\mathcal{T}$ distribution component. From (4.78), to maintain continuity in the objective function after the deletion of the K^+ component, we must update η_K^- in F_K^- by

$$\eta_K^- = \eta_K + \log \frac{\Gamma(\sum_{k=1}^{K} \alpha_k + 1)}{\Gamma(\sum_{k=1}^{K} \alpha_k)} = \eta_K + \log\left(\sum_{k=1}^{K} \alpha_k\right). \tag{4.79}$$

After the K^+ component is deleted, it is no more possible to add new components. However, the algorithm continues to run optimizing the existing components through standard EM steps, re-estimating the

hyperparameters κ_0 and $\alpha_{1:K}$, and deleting components that become weak till convergence is achieved.

4.2.5 Selection of the Hyperparameters

A drawback of the (EM-MAP) algorithm is the introduction of a number of hyperparameters and the need to set appropriate values for them. Such values are suggested in [60, 63], however, we found that the behavior of the algorithm can be sensitive to the values used for the hyperparameters. In a hierarchical Bayesian approach, one brings in priors for the hyperparameters and attempts to integrate them out with the (well-founded) expectation that the result will be less sensitive to the higher level parameters. On the other hand Empirical Bayes, considers optimizing over the hyperparameters, preferably after all other parameters have been integrated out. The latter approach is also called Maximum Likelihood II or Evidence Maximization [90]. Although, its principles have been questioned in the literature, it offers a practical and empirically proven successful approach to select values for the hyperparameters in the sense that the values determined are tailored to the samples at hand [91]. The hyperparameters in the proposed EM$^+$ algorithm are $\boldsymbol{\Psi} = \{\alpha_{1:K}, \alpha_+, \boldsymbol{\lambda}_0\}$, where $\alpha_{1:K}$ and α_+ define the Dirichlet prior (4.13) and $\boldsymbol{\lambda}_0 = \{\boldsymbol{m}_0, \kappa_0, \nu_0, \boldsymbol{S}_0\}$ define the NIW prior (4.14). In the following, we first discuss the estimation of $\boldsymbol{m}_0$, ν_0, and the combined parameter $\overline{\boldsymbol{S}\kappa}_0 \equiv (1 + \kappa_0^{-1})\boldsymbol{S}_0$. Then, we discuss the estimation of κ_0 and $\boldsymbol{S}_0$, and finally the estimation of $\alpha_{1:K}$ and α_+.

4.2.5.1 Estimation of the hyperparameters $\boldsymbol{m}_0$, ν_0, and $\overline{\boldsymbol{S}\kappa}_0 \equiv (1 + \kappa_0^{-1})\boldsymbol{S}_0$

In the EM$^+$ approach, we integrate out all but K mixture components as shown in (4.42). Using a conjugate prior for the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, the integration can be carried out analytically and obtain the equivalent expression (4.46). The algorithm starts with $K = 0$ and with all parameters having been integrated out resulting in the (log) evidence

$$\begin{aligned} L(\boldsymbol{X} \mid \boldsymbol{\lambda}_0) &= \sum_{i=1}^{N} \log \mathcal{T}(\boldsymbol{x}_i \mid \boldsymbol{m}_0, \frac{\kappa_0 + 1}{\kappa_0(\nu_0 - D + 1)}\boldsymbol{S}_0, \nu_0 - D + 1) \\ &= \sum_{i=1}^{N} \log \left(\frac{\Gamma(\frac{\nu_0+1}{2})}{\Gamma(\frac{\nu_0-D+1}{2})} \mid \pi\overline{\boldsymbol{S}\kappa}_0 \mid^{-\frac{1}{2}} \left[1 + (\boldsymbol{x} - \boldsymbol{m}_0)^T \left(\overline{\boldsymbol{S}\kappa}_0\right)^{-1} (\boldsymbol{x} - \boldsymbol{m}_0)\right]^{-\frac{\nu_0+1}{2}} \right) \end{aligned} \quad (4.80)$$

and where we used (4.44). We can then obtain the desired estimates as

$$\hat{\boldsymbol{\lambda}}_0 = \underset{\boldsymbol{\lambda}_0}{\operatorname{argmax}}\, L(\boldsymbol{X} \mid \boldsymbol{\lambda}_0). \quad (4.81)$$

Note that κ_0 and $\boldsymbol{S}_0$ appear in the objective function (4.80) as a unit $\overline{\boldsymbol{S}\kappa}_0 = (1+\kappa_0^{-1})\boldsymbol{S}_0$ and, therefore, cannot be estimated separately. This optimization is implemented only once in the beginning of the algorithm. The values $\boldsymbol{m}_0 = \bar{\boldsymbol{x}} \equiv (\sum_{i=1}\boldsymbol{x}_i)/N$, $\kappa_0 = 0.01$, $\nu_0 = D+2$, and $\boldsymbol{S}_0 = \text{diag}(\sum_{i=1}^{N}(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^2)/N^{(D+1)/D}$ suggested in [60] are provided as initial values to the software. The gradient of the objective function (4.80) with respect to the estimated parameters can be readily computed as follows and is also made available to the optimization routine.

$$\frac{\partial L(\boldsymbol{X} \mid \boldsymbol{\lambda}_0)}{\partial \boldsymbol{m}_0} = \sum_{i=1}^{N} \frac{(\nu_0+1)(\overline{\boldsymbol{S}\kappa}_0)^{-1}(\boldsymbol{x}_i - \boldsymbol{m}_0)}{1+(\boldsymbol{x}_i-\boldsymbol{m}_0)^T(\overline{\boldsymbol{S}\kappa}_0)^{-1}(\boldsymbol{x}_i-\boldsymbol{m}_0)} \tag{4.82}$$

$$\frac{\partial L(\boldsymbol{X} \mid \boldsymbol{\lambda}_0)}{\partial \overline{\boldsymbol{S}\kappa}_0} = -\frac{N}{2}(\overline{\boldsymbol{S}\kappa}_0)^{-1} + \left(\frac{\nu_0+1}{2}\right)\sum_{i=1}^{N} \frac{(\overline{\boldsymbol{S}\kappa}_0)^{-1}(\boldsymbol{x}_i-\boldsymbol{m}_0)(\boldsymbol{x}_i-\boldsymbol{m}_0)^T(\overline{\boldsymbol{S}\kappa}_0)^{-1}}{1+(\boldsymbol{x}_i-\boldsymbol{m}_0)^T(\overline{\boldsymbol{S}\kappa}_0)^{-1}(\boldsymbol{x}_i-\boldsymbol{m}_0)} \tag{4.83}$$

$$\frac{\partial L(\boldsymbol{X} \mid \boldsymbol{\lambda}_0)}{\partial \nu_0} = -\frac{N}{2}\Xi\left(\frac{\nu_0+1}{2}\right) - \frac{N}{2}\Xi\left(\frac{\nu_0-D+1}{2}\right) - \\ -\frac{1}{2}\sum_{i=1}^{N}\log\left[1+(\boldsymbol{x}_i-\boldsymbol{m}_0)^T(\overline{\boldsymbol{S}\kappa}_0)^{-1}(\boldsymbol{x}_i-\boldsymbol{m}_0)\right] \tag{4.84}$$

where $\Xi(x) \equiv \frac{d}{dx}\log\Gamma(x)$ is the *digamma* function. The result in (4.83) is a $D \times D$ matrix, the ijth element of which gives the derivative of $L(\boldsymbol{X} \mid \boldsymbol{\lambda}_0)$ with respect to the ijth element of $\overline{\boldsymbol{S}\kappa}_0$. Although not necessary, we maintain for simplicity a diagonal $\boldsymbol{S}_0$, consistent with its starting value and, therefore, only the diagonal elements of (4.83) are used.

4.2.5.2 Estimation of the hyperparameters κ_0, and $\boldsymbol{S}_0$

During the addition of a new component, the hyperparameters $\boldsymbol{\lambda}_0 = \{\boldsymbol{m}_0, \nu_0, \kappa_0, \boldsymbol{S}_0\}$ of the NIW distribution (4.14) affect the initialization of the parameters $\boldsymbol{\mu}_{K+1}$ and $\boldsymbol{\Sigma}_{K+1}$ in (4.73) and (4.74), respectively. But more importantly, they bias the values of $\boldsymbol{\mu}_{1:K}$ and $\boldsymbol{\Sigma}_{1:K}$ during the maximization of the MAP log-likelihood objective function (4.55) towards the mode of the NIW distribution, namely $\boldsymbol{m}_0$ and $\boldsymbol{S}_0/(\nu_0+D+2)$. In the previous section, the parameters $\boldsymbol{m}_0$, ν_0, and the combined parameter $\overline{\boldsymbol{S}\kappa}_0$ have been estimated by maximizing the evidence (4.80) after all mixture parameters have been integrated out. The covariance matrix of the predictive posterior in (4.43) is given by $\overline{\boldsymbol{S}\kappa}_0/(\nu_0-D-1) \cong \boldsymbol{S}_0/\kappa_0$, where the latter approximation holds for initial values $\kappa_0 \ll 1$ and $\nu_0 = D+2$. Therefore, $\boldsymbol{S}_0$ should be selected to match the typical covariance matrix of the components of the mixture, while κ_0 should be selected small enough so that $\boldsymbol{S}_0/\kappa_0$ produces a predictive posterior covering the range of the data $\boldsymbol{X}$. Then, at the start of the algorithm without any mixture components having been established already, only $\overline{\boldsymbol{S}\kappa}_0 \cong \boldsymbol{S}_0/\kappa_0$ can be estimated and not $\boldsymbol{S}_0$ and κ_0 separately. The estimation of κ_0, and implicitly of $\boldsymbol{S}_0 = \kappa_0\overline{\boldsymbol{S}\kappa}_0/(\kappa_0+1)$, needs to based on the

already developed components of the mixture and is effected after such components have been optimized along with a newly added component and by maximizing the MAP log-likelihood

$$\begin{aligned} F_K(\boldsymbol{\Theta}_{1:K}; \alpha_{1:K}, \alpha_+, \boldsymbol{\lambda}_0) &= \sum_{i=1}^{N} \log p(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K}) + \sum_{k=1}^{K} \log \frac{\mathrm{NIW}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k \mid \boldsymbol{\lambda}_0)}{\mathrm{NIW}(\boldsymbol{\mu}_k^*, \boldsymbol{\Sigma}_k^* \mid \boldsymbol{\lambda}_0)} + \\ &\quad \log \mathrm{Dir}(\pi_{1:K}, \pi_+ \mid \alpha_{1:K}, \alpha_+) + \eta_K \end{aligned} \tag{4.85}$$

with respect to κ_0 in $\boldsymbol{\lambda}_0$. This is accomplished by explicitly solving for κ_0 the first order optimality condition

$$\frac{\partial F_K(\boldsymbol{\Theta}_{1:K}, \alpha_{1:K}, \alpha_+, \boldsymbol{\lambda}_0)}{\partial \kappa_0} = \sum_{i=1}^{N} \frac{\partial}{\partial \kappa_0} \log p(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K}, \boldsymbol{\lambda}_0) + \sum_{k=1}^{K} \frac{\partial}{\partial \kappa_0} \log \left(\frac{\mathrm{NIW}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k \mid \boldsymbol{\lambda}_0)}{\mathrm{NIW}(\boldsymbol{\mu}_k^*, \boldsymbol{\Sigma}_k^* \mid \boldsymbol{\lambda}_0)} \right) = 0 \tag{4.86}$$

as explained next. From (4.46), we compute

$$\frac{\partial}{\partial \kappa_0} \log p(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K}, \boldsymbol{\lambda}_0) = \frac{\partial p(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K}, \boldsymbol{\lambda}_0)/\partial \kappa_0}{p(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K}, \boldsymbol{\lambda}_0)} = \frac{\pi_+ \cdot \partial \mathcal{T}(\boldsymbol{x}_i \mid \boldsymbol{\lambda}_0)/\partial \kappa_0}{p(\boldsymbol{x}_i \mid \boldsymbol{\Theta}_{1:K}, \boldsymbol{\lambda}_0)} = 0, \tag{4.87}$$

since κ_0 enters in $\mathcal{T}(\boldsymbol{x}_i \mid \boldsymbol{\lambda}_0)$ through $\overline{\boldsymbol{S}\kappa}_0$ which is estimated at the beginning of the algorithm and is kept fixed thereafter.

Furthermore, from the expression for the NIW distribution in (4.14) and (4.15) and plugging in the expressions for its mode $\boldsymbol{\mu}^* = \boldsymbol{m}_0$, $\boldsymbol{\Sigma}^* = \boldsymbol{S}_0/(\nu_0 + D + 2)$, we obtain

$$\begin{aligned} &\log \left(\frac{\mathrm{NIW}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k \mid \boldsymbol{\lambda}_0)}{\mathrm{NIW}(\boldsymbol{\mu}_k^*, \boldsymbol{\Sigma}_k^* \mid \boldsymbol{\lambda}_0)} \right) = \\ &= -\frac{\nu_0 + D + 2}{2} \log \mid \boldsymbol{\Sigma}_k \mid - \frac{\kappa_0}{2} (\boldsymbol{\mu}_k - \boldsymbol{m}_0)^T \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{\mu}_k - \boldsymbol{m}_0) - \frac{1}{2} \mathrm{trace}(\boldsymbol{S}_0 \boldsymbol{\Sigma}_k^{-1}) - \\ &\quad - \frac{D(\nu_0 + D + 2)}{2} \log \left(\frac{\nu_0 + D + 2}{2} \right) + \frac{\nu_0 + D + 2}{2} \log \mid \boldsymbol{S}_0 \mid + \frac{D(\nu_0 + D + 2)}{2}. \end{aligned} \tag{4.88}$$

Substituting $\boldsymbol{S}_0 = \kappa_0 \overline{\boldsymbol{S}\kappa}_0/(\kappa_0 + 1)$ in (4.88), the required derivative is easily computed as

$$\begin{aligned} &\frac{\partial}{\partial \kappa_0} \log \left(\frac{\mathrm{NIW}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k \mid \boldsymbol{\lambda}_0)}{\mathrm{NIW}(\boldsymbol{\mu}_k^*, \boldsymbol{\Sigma}_k^* \mid \boldsymbol{\lambda}_0)} \right) = \\ &= -\frac{1}{2} (\boldsymbol{\mu}_k - \boldsymbol{m}_0)^T \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{\mu}_k - \boldsymbol{m}_0) - \frac{1}{2(\kappa_0 + 1)^2} \mathrm{trace}(\overline{\boldsymbol{S}\kappa}_0 \boldsymbol{\Sigma}_k^{-1}) + \frac{D(\nu_0 + D + 2)}{2\kappa_0(\kappa_0 + 1)}. \end{aligned} \tag{4.89}$$

Then, using (4.87) and (4.89) in (4.86) yields the following equation for the optimal κ_0:

$$\frac{A}{\kappa_0(\kappa_0+1)} - \frac{B}{(\kappa_0+1)^2} - C = 0 \tag{4.90}$$

where, we defined

$$\begin{aligned} A &= KD(\nu_0 + D + 2) \\ B &= \sum_{k=1}^{K} \mathrm{trace}(\overline{\boldsymbol{S}}\kappa_0 \boldsymbol{\Sigma}_k^{-1}) \\ C &= \sum_{k=1}^{K} (\boldsymbol{\mu}_k - \boldsymbol{m}_0)^T \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{\mu}_k - \boldsymbol{m}_0) \end{aligned} \tag{4.91}$$

Equation (4.90) is a cubic equation for κ_0 that can be shown to have a unique positive solution corresponding to a maximum of the MAP log-likelihood $F_K(\boldsymbol{\Theta}_{1:K}, \alpha_{1:K}, \alpha_+, \boldsymbol{\lambda}_0)$ as a function of κ_0.

4.2.5.3 Estimation of the hyperparameters $\alpha_{1:K}, \alpha_+$

The hyperparameters $\alpha_{1:K}, \alpha_+$ defining the Dirichlet prior (4.13) are also estimated based on the already developed and optimized components of the mixture by maximizing the MAP log-likelihood $F_K(\boldsymbol{\Theta}_{1:K}, \boldsymbol{\Psi})$ over $\alpha_{1:K}, \alpha_+$ subject to the constraints $\alpha_{1:K}, \alpha_+ \geq 1$. In addition, we impose the constraint

$$\sum_{k=1}^{K} \alpha_k + \alpha_+ = K + \alpha_{0+}. \tag{4.92}$$

The role of (4.92) is two-fold. On the one hand, it serves the technical purpose of making the variables η_K in (4.60) independent of the α_k, and therefore, these variables can be ignored during their optimization. On the other hand, it promotes competition amongst the mixture components, with the most fit ones (i.e. the ones with higher probability π_k) achieving through the optimization higher α_k at the expense of the less fit components. This in turn, results in even smaller probabilities for the weak components during the EM algorithm, thus accelerating their removal. The opportunity for such competition is strengthened by taking a larger value for the constant α_{0+}. Indeed, if $\alpha_{0+} = 1$, the above constraints imply that $\alpha_{1:K} = 1$ and $\alpha_+ = 1$ is the only feasible solution and effectively the prior on the π_k is rendered uniform. Also, (4.72) implies that a large value for α_{0+} hinders the introduction of new components, especially at the beginning of the algorithm when $\alpha_+ \approx \alpha_{0+}$. This is desirable to the extent that only strong components are allowed to develop but it is possible that an overly large value for α_{0+} can prevent any components from forming at all.

The maximization of $F_K(\boldsymbol{\Theta}_{1:K}, \alpha_{1:K}, \alpha_+, \boldsymbol{\lambda}_0)$ in (4.85) with respect to the $\alpha_{1:K}, \alpha_+$ is equivalent with the

following program

$$\min_{\alpha_{1:K},\alpha_+} \quad \sum_{k=1}^{K}\Big(\log\Gamma(\alpha_k)-(\alpha_k-1)\log\pi_k\Big)+\Big(\log\Gamma(\alpha_+)-(\alpha_+-1)\log\pi_+\Big)$$

$$\text{subject to}: \quad \sum_{k=1}^{K}\alpha_k+\alpha_+ = K+\alpha_{0+} \qquad (4.93)$$
$$\alpha_k \geq 1,\ k=1,\ldots,K, \quad \alpha_+ \geq 1$$

We approach solving (4.93) via a partial duality approach in order to take advantage of its inherent structure. Specifically, by considering the Lagrange multiplier $\boldsymbol{\mu}$ corresponding to the equality constraint in (4.93), we maximize the dual function

$$\max_{\boldsymbol{\mu}} \quad \phi(\boldsymbol{\mu}) \equiv \left(\sum_{k=1}^{K}\min_{\alpha_k\geq 1} f_k(\alpha_k,\boldsymbol{\mu}) \ + \ \min_{\alpha_+\geq 1} f_+(\alpha_+,\boldsymbol{\mu})\right) \qquad (4.94)$$

where we defined

$$f_k(\alpha_k,\boldsymbol{\mu}) \equiv \log\Gamma(\alpha_k)-(\alpha_k-1)\log\pi_k+\boldsymbol{\mu}\cdot\alpha_k \qquad (4.95)$$

$$f_+(\alpha_+,\boldsymbol{\mu}) \equiv \log\Gamma(\alpha_+)-(\alpha_+-1)\log\pi_++\boldsymbol{\mu}\cdot\alpha_+ \qquad (4.96)$$

Thus, the evaluation of the dual function requires solving $K+1$ decoupled scalar minimization subproblems that can be readily shown to be convex. Furthermore, the first and second derivatives of $\phi(\boldsymbol{\mu})$ can be easily obtained and program (4.94) is shown to be concave. We employ a safeguarded Newton's method to efficiently solve all of these subproblems.

4.3 Experimental Results of The EM$^+$ Algorithm

The steps of the proposed EM$^+$ algorithm are summarized in Table 4.2. Here we present the results of using the EM$^+$ algorithm on different sets of synthetically generated data. For a one-dimensional example, we first construct the underlying "actual" distribution shown as the blue lines in Figure 4.1 (for three different cases of (a), (b) and (c)), as a mixture of 10 Gaussians that have normalized random weights. We then take 1000 samples from this distribution which provides the observations of the EM$^+$ algorithm shown by their histogram (normalized by area) in Figure 4.1. The means of the 10 Gaussians that the samples are

The EM$^+$ Algorithm

1: Initialize hyperparameters $\boldsymbol{\lambda}_0 = \{\boldsymbol{m}_0, \nu_0, \kappa_0, \boldsymbol{S}_0\}$, see section 4.2.5

2: while N_k^+ is not empty || convergence not reached

3: new component $K+1$

initialize parameter of new component, π_{K+1} in (4.75), $\boldsymbol{\mu}_{K+1}$ in (4.73), $\boldsymbol{\Sigma}_{K+1}$ in (4.74)

find parameters such that objective function is maximized

add component if objective function in (4.61) is increased and set $K = K+1$, see section 4.2.2

4: E-step: update responsibilities

compute R_{iK^+} and R_{iK} for all other components in (4.49)

5: M-step: update parameters

compute $\boldsymbol{\mu}_{1:K}$ in (4.22)

compute $\boldsymbol{\Sigma}_{1:K}$ in (4.24)

compute $\pi_{1:K}$ and π_+ in (4.50) and (4.51)

6: update $\alpha_{1:K}$ and α^+, see section 4.2.5.3

7: delete week components, see section 4.2.3

8: update κ_0 and $\boldsymbol{S}_0$, see section 4.2.5.1

9: end

Table 4.2: The steps of the EM$^+$ algorithm.

generated from are selected randomly between $[-20, 20]$. The variance of the components will determine how much overlap there will be between each component, and for this three different cases are considered shown as (a), (b) and (c) in Figure 4.1. In the first case Figure 4.1 (a), the variance of each component of the "actual" distribution is randomly selected between $[0.1, 0.5]$, which creates components that are apart. In the other two cases, Figure 4.1 (a) and (b), the variance of each component is selected between $[0.5, 2]$ and $[2, 5]$, which will increases the overlaps of the resulting mixtures in each case. The results of the EM$^+$ are shown with the red lines, that can be seen to closely follow the latent "actual" distribution in the blue lines. The EM$^+$ generally takes less iterations to converge for the case where the components are apart from each other (Figure 4.1 (a)) since each region can be represented by a single Gaussian, and the estimated parameters by the algorithm quickly converge to Gaussians that represent each region well.

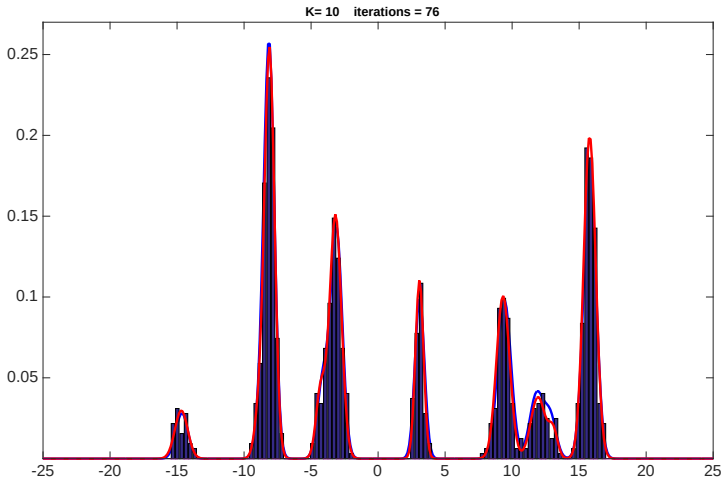
In a similar manner, the EM$^+$ algorithm is implemented on three sets of two-dimensional observations generated from three different mixtures each consisting of 10 Gaussians components. The components of the "actual" mixtures are represented by the blue ellipses in Figure 4.2 (a), (b) and (c). Each dimension of the means (centers) of the components are randomly drawn from $[-20, 20]$ similar to before, and three different cases are considered for the covariances that are in the form of $\text{diag}(\sigma_1^2, \sigma_2^2)$. Figure 4.2 (a) corresponds to drawing the σ^2 values randomly in the range of $[0.1, 0.5]$, and Figure 4.2 (a) and (b) to $[0.5, 2]$ and $[2, 5]$ respectively; the amount of overlapping between the components increases from (a) to (c). The observations that are drawn from the "actual" distribution are represented by black dots in each case, and the resulting components from the EM$^+$ algorithm are shown in red. Similar to before, case (a) converges faster. It can be seen in case (b) and (c) that the overlapping components can often be represented by a single component.

Next, we implement the EM$^+$ algorithm on the spiral data set presented in [84]. The three-dimensional noisy observations are generated from the spiral function as the following

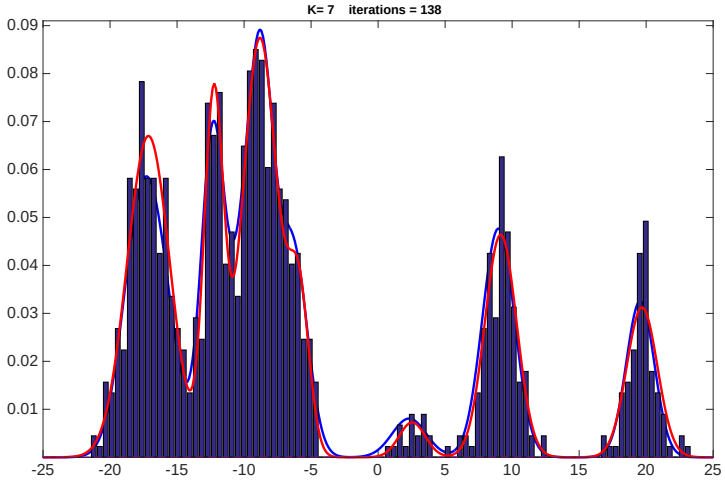
$$\boldsymbol{x} = (x_1, x_2, x_3) = (13 - 0.5t)cos(t), -(13 - 0.5t)sin(t), t) + \boldsymbol{v} \tag{4.97}$$

where $t \in [0, 4\pi]$ and $\boldsymbol{v} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$, with $\sigma^2 = 0.5^2$.

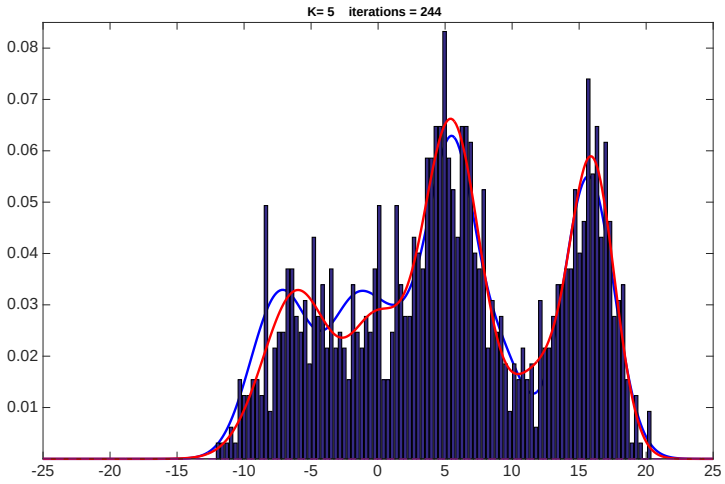
The vanilla EM algorithm is sensitive to the initialization of the parameters that can often get stuck in local optima and produce undesirable results as shown in Figure 4.3. As mentioned before, [84] addresses this issue by splitting and merging components throughout the algorithm, which results in escaping from the local optima.



(a)

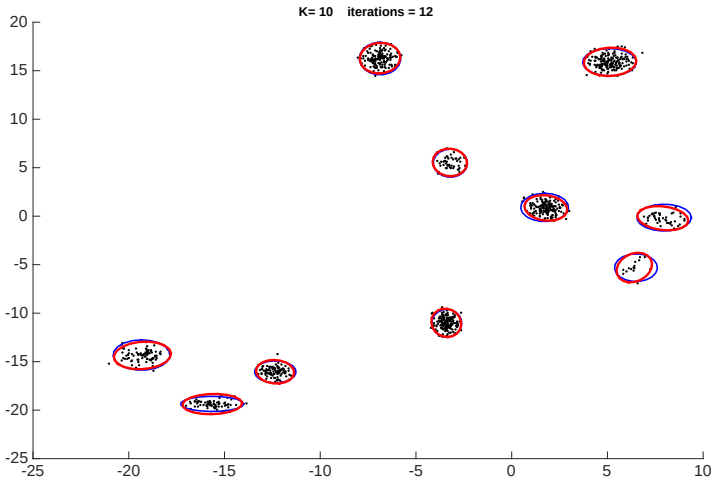


(b)

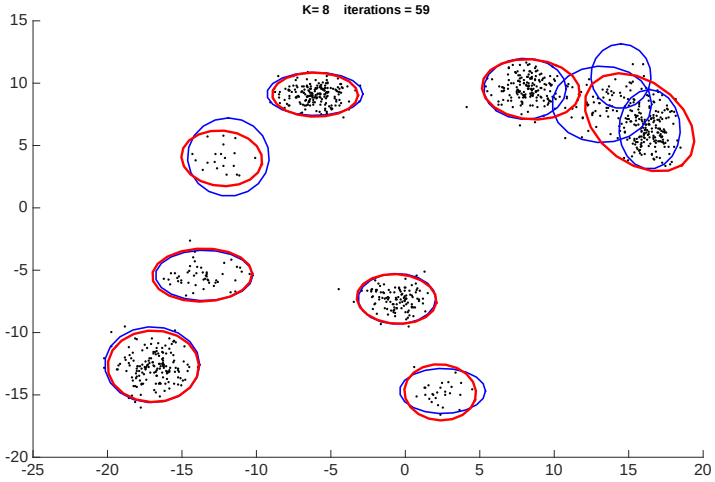


(c)

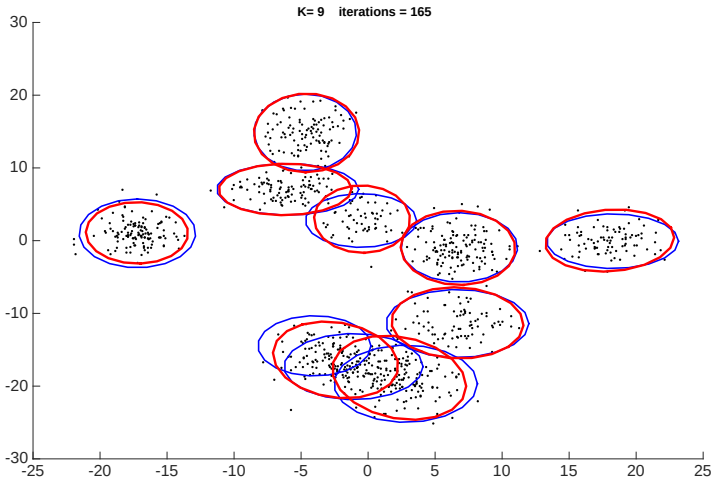
Figure 4.1: 1D example of the EM$^+$ algorithm. The bars are the histogram of the observations (normalized by area), which are generated from the "actual" distribution in blue. The red is the result of the EM$^+$ algorithm. The number of components found and iterations it took is shown on top of the plots.



(a)



(b)



(c)

Figure 4.2: Two dimensional example of the EM$^+$ algorithm. In each plot, the dots represent the observations, which are generated from the "actual" components in blue ellipses. The red ellipses are the result of the EM$^+$ algorithm. The number of components found and iterations it took is shown on top of the plots.

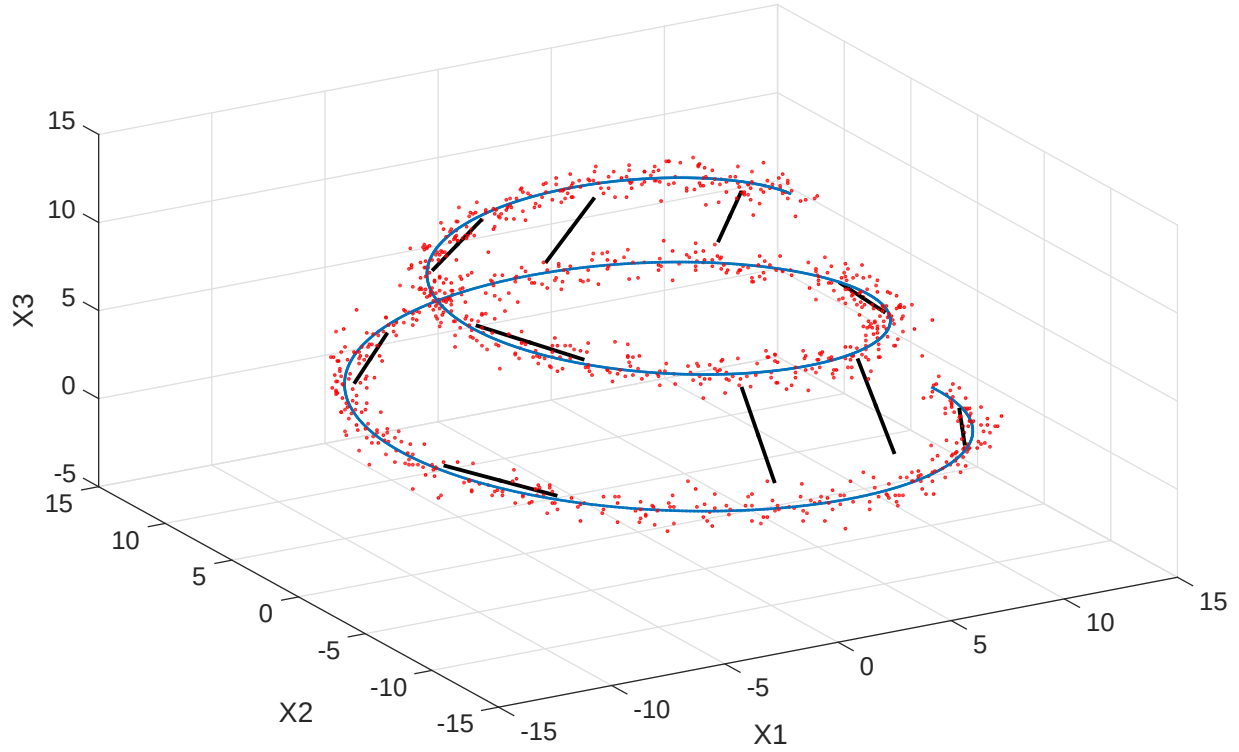
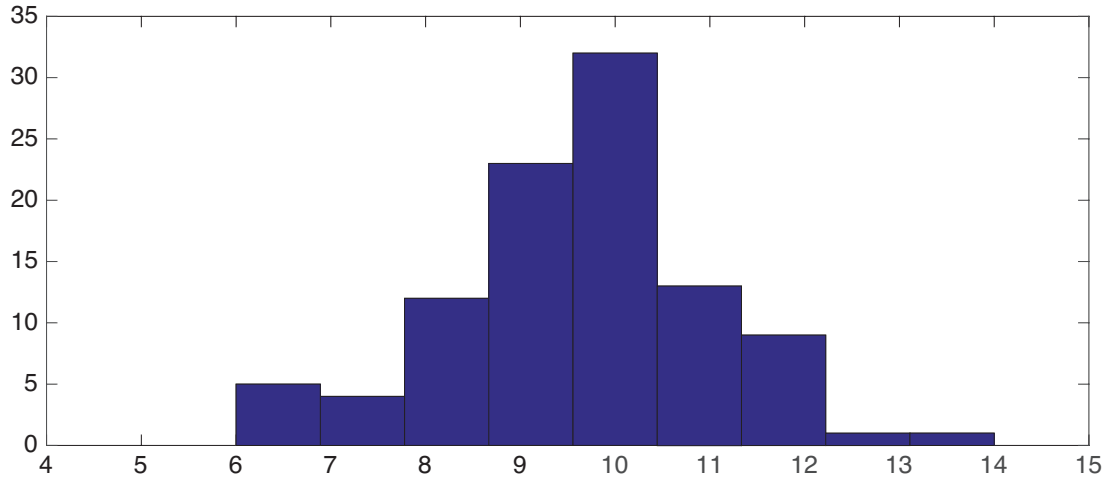
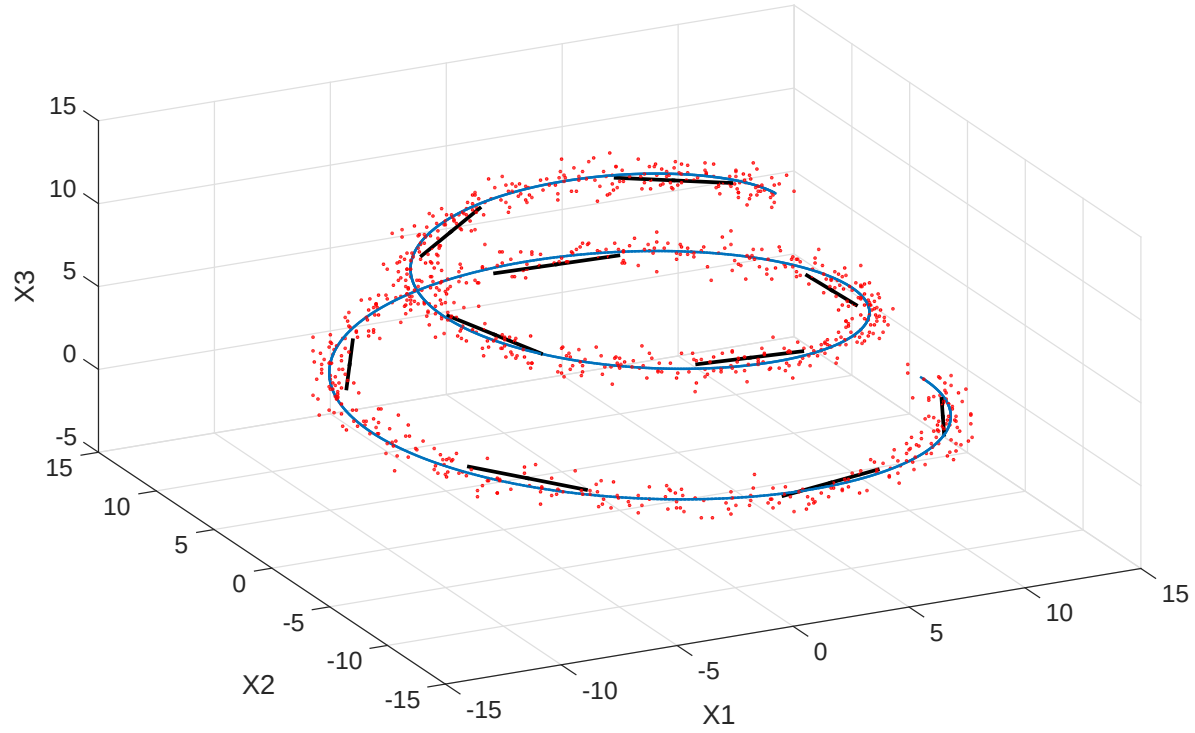


Figure 4.3: The vanilla EM is prone to getting stuck in local optima and producing undersireable results. The blue line is the spiral funciton, and the red dots are the generated observations. The black lines represent the detected components.

By progressively adding and deleting components, the proposed EM$^+$ is also capable of escaping local optima and unlike the vanilla EM, is not sensitive to the initialization of the parameters. Moreover, the number of clusters are detected automatically. Figure 4.4 shows the results of the EM$^+$ algorithm on the spiral data set, where (a) shows the histogram of the number of components detected for 100 different runs of the algorithm, and (b) shows the results of the most frequent number of components found, $K = 10$. In the next chapter, we present a filtering algorithm that utilizes the EM$^+$ for generating the predictive distribution as a mixture of Gaussians from samples.



(a) histogram of number of components detected for 100 runs



(b) result of the EM$^+$ when 10 components are detected

Figure 4.4: Results of EM$^+$ on the spiral data set. (a) shows the histogram of the number of components detected for 100 different runs of the algorithm, with the most common result, when $K = 10$ shown in (b). The blue line is the spiral funciton, and the red dots are the generated observations. The black lines represent the detected components.

Chapter 5

Filtering with EM$^+$

5.1 Introduction

The problem of filtering arises in a variety of different areas of science, where the objective is to estimate the state of the system based on noisy observations of the states from measurements. Consider a stochastic non-linear dynamic system defined by a Markovian process of

$$\boldsymbol{x}_t = f_t(\boldsymbol{x}_{t-1}) + \boldsymbol{v}_{t-1} \tag{5.1}$$

where $\boldsymbol{x} \in \mathbb{R}^{D_x}$, $f()$ is the non-linear dynamics function, and $\boldsymbol{v}_{t-1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{v_{t-1}})$ is the process noise at time $t-1$.

The measurement of the state at each time is defined by

$$\boldsymbol{z}_t = h_t(\boldsymbol{x}_t) + \boldsymbol{q}_t \tag{5.2}$$

where $\boldsymbol{z} \in \mathbb{R}^{D_z}$, $h()$ is a non-linear measurement function, and $\boldsymbol{q}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{q_t})$ is the measurement noise at time t.

In a Bayesian perspective the filtering problem is updating some belief in the state conditioned on the measurements. Given an initial prior $p(\boldsymbol{x}_0)$ and a dynamic system described above, the goal of filtering is to recursively infer the distribution on the hidden state $\boldsymbol{x}_t$ by incorporating the sequential measurements.

The posterior probability distribution function (pdf) $p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t})$ can be updated recursively using Bayes's theorem in two stages, the prediction step and the update step. The prediction step is obtained by using the Chapman-Kolmogorov equation as

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t-1}) = \int p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}) p(\boldsymbol{x}_{t-1} \mid \boldsymbol{z}_{1:t-1}) d\boldsymbol{x}_{t-1} \tag{5.3}$$

and once a measurement $\boldsymbol{z}_t$ becomes available, it is used to update the prior via Bayes' rule to obtain the posterior of the current state

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t}) = \frac{1}{c} p(\boldsymbol{z}_t \mid \boldsymbol{x}_t) p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t-1}) \tag{5.4}$$

where

$$c = p(\boldsymbol{z}_t \mid \boldsymbol{z}_{1:t-1}) = \int p(\boldsymbol{z}_t \mid \boldsymbol{x}_t) p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t-1}) d\boldsymbol{x}_t \tag{5.5}$$

Obtaining the density in equation (5.4) gives the most complete description of the state at each iteration, which means that the filtering problem can be regarded as solved [76, 92].

5.1.1 Gaussian Filtering

If the state transition in equation (5.1) is linear in the form of

$$\boldsymbol{x}_t = \boldsymbol{F}_t \boldsymbol{x}_{t-1} + \boldsymbol{v}_{t-1} \tag{5.6}$$

and the the measurement in equation (5.2) is also linear as

$$\boldsymbol{z}_t = \boldsymbol{H}_t \boldsymbol{x}_t + \boldsymbol{q}_t \tag{5.7}$$

and $\boldsymbol{v}_{t-1}$ and $\boldsymbol{q}_t$ are drawn from Gaussian distributions (which is the case here), then the posterior on the state at every iteration is Gaussian and the Kalman filter gives the optimal filtering set of equations for updating the belief in the state [76, 92]. The steps of the Kalman filter are summarized in Table 5.1.

Unfortunately these highly restrictive assumptions are not true in most cases and the Kalman filter cannot be used, or needs to be modified in order to be applied to non-linear systems. The most popular variation of the Kalman filter is the Extended Kalman filter (EKF) that estimates the state by using local linearization of the state transition and measurement equations around the mean of the current state prediction. To obtain

The Kalman Filter

1: initialize prior distribution on the state $p(\boldsymbol{x}_{t-1} \mid \boldsymbol{z}_{1:t-1}) = \mathcal{N}(\boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1})$

2: get prediction

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t-1}) = \mathcal{N}(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1})$$
$$\boldsymbol{\mu}_{t|t-1} = \boldsymbol{F}_t \boldsymbol{\mu}_{t-1|t-1}$$
$$\boldsymbol{\Sigma}_{t|t-1} = \boldsymbol{\Sigma}_{v_{t-1}} + \boldsymbol{F}_t \boldsymbol{\Sigma}_{t-1|t-1} \boldsymbol{F}_t^\top$$

3: measurement $\boldsymbol{z}_t$ comes in

4: compute the Kalman gain matrix

$$\boldsymbol{K}_t = \boldsymbol{\Sigma}_{t|t-1} \boldsymbol{H}_t^\top (\boldsymbol{H}_t \boldsymbol{\Sigma}_{t|t-1} \boldsymbol{H}_t^\top + \boldsymbol{\Sigma}_{q_t})^{-1}$$

5: update (get posterior)

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t}) = \mathcal{N}(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t})$$
$$\boldsymbol{\mu}_{t|t} = \boldsymbol{\mu}_{t|t-1} + \boldsymbol{K}_t (\boldsymbol{z}_t - \boldsymbol{H}_t \boldsymbol{\mu}_{t|t-1})$$
$$\boldsymbol{\Sigma}_{t|t} = \boldsymbol{\Sigma}_{t|t-1} - \boldsymbol{K}_t \boldsymbol{H}_t \boldsymbol{\Sigma}_{t|t-1}$$

6: estimate state based on $p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t})$

7: set $t = t + 1$, set prior equal to the posterior, and go back to 2.

Table 5.1: The steps of the Kalman filter.

a closed-form solution of the prediction step of the filter, $f_t(\boldsymbol{x}_{t-1})$ is linearized around $\boldsymbol{\mu}_{t-1|t-1}$. Ommiting the higher terms of the Taylor series expansion, the linearized state transition equation around $\boldsymbol{\mu}_{t-1|t-1}$ is

$$f_t(\boldsymbol{x}_t) \approx f_t(\boldsymbol{\mu}_{t-1|t-1}) + \hat{\boldsymbol{F}}_t[\boldsymbol{x}_{t-1} - \boldsymbol{\mu}_{t-1|t-1}] \tag{5.8}$$

where

$$\hat{\boldsymbol{F}}_t = \frac{\partial f_t}{\partial \boldsymbol{x}_{t-1}}(\boldsymbol{\mu}_{t-1|t-1}) \tag{5.9}$$

Similarly to obtain a closed-form solution of the posterior of the filter, $h_t(\boldsymbol{x}_t)$ is linearized around $\boldsymbol{\mu}_{t|t-1}$, as

$$H_t(\boldsymbol{x}_t) \approx H_t(\boldsymbol{\mu}_{t|t-1}) + \hat{\boldsymbol{H}}_t[\boldsymbol{x}_t - \boldsymbol{\mu}_{t|t-1}] \tag{5.10}$$

where

$$\hat{\boldsymbol{H}}_t = \frac{\partial h_t}{\partial \boldsymbol{x}_t}(\boldsymbol{\mu}_{t|t-1}) \tag{5.11}$$

The steps of the EKF are given in Table 5.2. The local linearization used in the EKF and other variations of the Kalman filter generally do not capture the non-linearity away from centers of the linearization. Moreover, since the posterior of these filters at every iteration is Gaussian, they can never represent the true density in a multimodal system. Instead, they act more as a maximum likelihood estimator, and follow one of the peaks of the density function [76].

5.1.2 Particle Filtering

The particle filter is used when the system is non-linear, and the probability distribution on the states are non-Gaussian at each time step. The earliest form of the particle filtering was introduced in [93], where the central idea was to approximate the posterior density function by a set of weighted particles as

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t}) \approx \sum_{i=1}^{N_s} w_t^i \delta(\boldsymbol{x}_t - \boldsymbol{x}_t^i) \tag{5.12}$$

where $\boldsymbol{x}_t^{1:N_s}$ are the particles at time step t, with associated weights of $w_t^{1:N_s}$ that sum to 1. The particles are selected based on a technique known as *importance sampling*. The importance sampling is used when it is difficult to draw samples from a distribution $p(x) \propto r(x)$, but one can evaluate $r(x)$. Then samples are drawn from a user-defined proposal distribution $s()$, and $p(x)$ is approximated as

The Extended Kalman Filter

1: initialize prior distribution on the state $p(\boldsymbol{x}_{t-1} \mid \boldsymbol{z}_{1:t-1}) = \mathcal{N}(\boldsymbol{\mu}_{t-1|t-1}, \boldsymbol{\Sigma}_{t-1|t-1})$

2: linearize f_t around $\boldsymbol{\mu}_{t-1|t-1}$, compute $\hat{\boldsymbol{F}}_t$, get prediction

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t-1}) = \mathcal{N}(\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1})$$
$$\boldsymbol{\mu}_{t|t-1} = f_t(\boldsymbol{\mu}_{t-1|t-1})$$
$$\boldsymbol{\Sigma}_{t|t-1} = \boldsymbol{\Sigma}_{v_{t-1}} + \hat{\boldsymbol{F}}_t \boldsymbol{\Sigma}_{t-1|t-1} \hat{\boldsymbol{F}}_t^\top$$

3: measurement $\boldsymbol{z}_t$ comes in

4: linearize h_t around $\boldsymbol{\mu}_{t|t-1}$, compute $\hat{\boldsymbol{H}}_t$, compute the Extended Kalman gain matrix

$$\boldsymbol{K}_t = \boldsymbol{\Sigma}_{t|t-1} \hat{\boldsymbol{H}}_t^\top (\hat{\boldsymbol{H}}_t \boldsymbol{\Sigma}_{t|t-1} \hat{\boldsymbol{H}}_t^\top + \boldsymbol{\Sigma}_{q_t})^{-1}$$

5: update (get posterior)

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t}) = \mathcal{N}(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t})$$
$$\boldsymbol{\mu}_{t|t} = \boldsymbol{\mu}_{t|t-1} + \boldsymbol{K}_t(\boldsymbol{z}_t - h_t(\boldsymbol{\mu}_{t|t-1}))$$
$$\boldsymbol{\Sigma}_{t|t} = \boldsymbol{\Sigma}_{t|t-1} - \boldsymbol{K}_t \hat{\boldsymbol{H}}_t \boldsymbol{\Sigma}_{t|t-1}$$

6: estimate state based on $p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t})$

7: set $t = t + 1$, set prior equal to the posterior, and go back to 2.

Table 5.2: The steps of the Extended Kalman filter.

$$p(x) \approx \sum_{i=1}^{N_s} w^i \delta(x - x^i) \tag{5.13}$$

where the weights are

$$w^i \propto \frac{r(x^i)}{s(x^i)} \tag{5.14}$$

and are normalized to sum to 1. In the particle filter one can derive the update formulas for the weights of the particles from one step to the next by using the posterior update in equation (5.4) (for a detailed derivation refer to [92]), which gives

$$w_t^i \propto w_{t-1}^i \frac{p(\boldsymbol{z}_t \mid \boldsymbol{x}_t^i) p(\boldsymbol{x}_t^i \mid \boldsymbol{x}_{t-1}^i)}{s(\boldsymbol{x}_t^i \mid \boldsymbol{x}_{t-1}^i, \boldsymbol{z}_t)} \tag{5.15}$$

which are again normalized to sum to 1. The state transition probability $p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}^i)$ is an intuitive and common choice for the proposal distribution, which means that the weights of the particles are updated according to

$$w_t^i \propto w_{t-1}^i p(\boldsymbol{z}_t \mid \boldsymbol{x}_t^i) \tag{5.16}$$

The posterior of the filter in equation (5.4), which gives the complete description of the state at the next iteration, is then approximated by the updated weights that result from equation (5.16). In the next iteration of the filter, the updated particles become the currents ones, and the same process is repeated.

A common issue with the particle filter is that after a number of iterations, all but a few particles will have negligible weights; this is known as the *degeneracy* problem. This means that most of the computation is spent on updating the small weights that don't contribute to approximating the posterior distribution. The effect of the degeneracy can be reduced by a better choice of the proposal distribution, and also by a process known as *resampling*. When degeneracy is detected, the discrete distribution in equation(5.12) is sampled N_s times (with replacement) which will result in the elimination of the particles with small weights. This however can lead to the issue of "particle collapse", i.e. the loss of diversity amongst particles, that results in poor representation of the posterior distribution [92].

The choice of the proposal distribution, and the resampling step are crucial in the performance of the particle filter. The variations of the particle filter in the literature are concerned with the selection of the proposal distribution, and when and how the resampling process should be performed in order to reduce the effect

of degeneracy and the loss of diversity amongst particles. The sampling importance sampling (SIR) filter in [93] uses $p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}^i)$ as the proposal distribution and performs resampling from the discrete posterior distribution at every iteration. As mentioned above, this can quickly suffer from the collapse of particles (especially if the process noise is small). To address this issue the regularized particle filter (RPF) [94] performs resampling on a continuous approximation (as opposed to discrete) of the posterior distribution created by $p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t}) \approx \sum_{i=1}^{N_s} w_t^i K(\boldsymbol{x}_t - \boldsymbol{x}_t^i)$, where $K()$ is a Kernel density of choice. Another common particle filter is the auxiliary particle filter (APF) [95], which is a variation of the SIR filter that uses the knowledge about the next observation to infer which particles are likely to be compatible with the future measurement, and therefore should have a higher chance of surviving in the resampling step. An excellent review on the various approaches of the particle filter is given in [92], and a benchmark problem in the filtering literature is used to compare their performance. In section 5.3, we use the same problem to compare the results of the proposed filtering method to the particle filters.

5.1.3 Filtering with Mixtures of Gaussians

Non-linear state estimation with the use of mixtures of Gaussians dates back to the work of [76]. Suppose that in the prediction step, the distribution of the state is represented by a finite summation of Gaussians as

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t-1}) = \sum_{i=1}^{K} \pi^{(i)} \mathcal{N}(\boldsymbol{x}_t \mid \boldsymbol{\mu}_{t|t-1}^{(i)}, \boldsymbol{\Sigma}_{t|t-1}^{(i)}) \tag{5.17}$$

where $\pi^{(i)}$ is the weight of each Gaussian and K is the total number of components of the mixture. Using Bayes' rule in equation (5.4) and the likelihood given by

$$p(\boldsymbol{z}_t \mid \boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{z}_t \mid h_t(\boldsymbol{x}_t), \Sigma_{q_t}) \tag{5.18}$$

the posterior $p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t})$ can be written as

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t}) \propto \sum_{i=1}^{K} \pi^{(i)} \mathcal{N}(\boldsymbol{x}_t \mid \boldsymbol{\mu}_{t|t-1}^{(i)}, \boldsymbol{\Sigma}_{t|t-1}^{(i)}) \mathcal{N}(\boldsymbol{z}_t \mid h_t(\boldsymbol{x}_t), \Sigma_{q_t}) \tag{5.19}$$

Since the measurement function $h_t(\boldsymbol{x}_t)$ in $\mathcal{N}(\boldsymbol{z}_t \mid h_t(\boldsymbol{x}_t), \Sigma_{q_t})$ is non-linear, the expression above cannot be written in closed-form. To obtain a closed-form solution of the posterior of the filter, $h_t(\boldsymbol{x}_t)$ is linearized around $\boldsymbol{\mu}_{t|t-1}^{(i)}$, the center of each mixture in $p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t-1})$. Omitting higher terms of the Taylor series expansion, the linearized measurement function around $\boldsymbol{\mu}_{t|t-1}^{(i)}$ is

$$h_t(\boldsymbol{x}_t) \approx h_t(\boldsymbol{\mu}_{t|t-1}^{(i)}) + \hat{\boldsymbol{H}}_t^{(i)} (\boldsymbol{x}_t - \boldsymbol{\mu}_{t|t-1}^{(i)}) \tag{5.20}$$

where

$$\hat{\boldsymbol{H}}_t^{(i)} = \frac{\partial h_t}{\partial \boldsymbol{x}_t}(\boldsymbol{\mu}_{t|t-1}^{(i)}) \tag{5.21}$$

after linearization of the measurement function, the two Gaussians in equation (5.19) for each $\boldsymbol{\mu}_{t|t-1}^{(i)}$ combine into one, and the posterior as shown in [76] can be written as

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t}) = \sum_{i=1}^{K} \pi'^{(i)} \mathcal{N}(\boldsymbol{x}_t \mid \boldsymbol{\mu}_{t|t}^{(i)}, \boldsymbol{\Sigma}_{t|t}^{(i)}) \tag{5.22}$$

where

$$\begin{aligned}
\boldsymbol{K}_t^{(i)} &= \boldsymbol{\Sigma}_{t|t-1}^{(i)} {\boldsymbol{H}_t^{(i)}}^\top (\boldsymbol{H}_t^{(i)} \boldsymbol{\Sigma}_{t|t-1}^{(i)} {\boldsymbol{H}_t^{(i)}}^\top + \boldsymbol{\Sigma}_{q_t})^{-1} \\
\boldsymbol{\mu}_{t|t}^{(i)} &= \boldsymbol{\mu}_{t|t-1}^{(i)} + \boldsymbol{K}_t^{(i)} (\boldsymbol{z}_t - h_t(\boldsymbol{\mu}_{t|t-1}^{(i)})) \\
\boldsymbol{\Sigma}_{t|t}^{(i)} &= \boldsymbol{\Sigma}_{t|t-1}^{(i)} - \boldsymbol{K}_t^{(i)} \boldsymbol{H}_t^{(i)} \boldsymbol{\Sigma}_{t|t-1}^{(i)} \\
\pi'^{(i)}_{\text{un}} &\propto \pi^{(i)} \mathcal{N}(\boldsymbol{z}_t \mid h_t(\boldsymbol{\mu}_{t|t-1}^{(i)}), \boldsymbol{H}_t^{(i)} \boldsymbol{\Sigma}_{t|t-1}^{(i)} {\boldsymbol{H}_t^{(i)}}^\top + \boldsymbol{\Sigma}_{q_t}) \\
\pi'^{(i)} &= \frac{\pi'^{(i)}_{\text{un}}}{\sum_{j=1}^{K} \pi'^{(j)}_{\text{un}}}
\end{aligned} \tag{5.23}$$

One can see that the parameters obtained above for each of the components of the posterior mixture, are the parameters that would be obtained when using an extended Kalman filter. Therefore representing the predictive distribution as a summation of K Gaussians, and the linearization of the measurement function around the means of each of the components, is equivalent to finding the posterior distribution by running K extended Kalman filters in parallel [76].

To get the predictive distribution of the next state, [76] suggests linearizing the dynamics around $\boldsymbol{\mu}_{t|t}^{(i)}$, the means of the posterior mixture above that gives

$$f_{t+1}(\boldsymbol{x}_t) \approx f_{t+1}(\boldsymbol{\mu}_{t|t}^{(i)}) + \hat{\boldsymbol{F}}_{t+1}^{(i)} (\boldsymbol{x}_t - \boldsymbol{\mu}_{t|t}^{(i)}) \tag{5.24}$$

where

$$\hat{\boldsymbol{F}}_{t+1}^{(i)} = \frac{\partial f_{t+1}}{\partial \boldsymbol{x}_t}(\boldsymbol{\mu}_{t|t}^{(i)}) \tag{5.25}$$

Using the Chapman-Chapman-Kolmogorov integral in equation (5.3), the predictive distribution will result in

$$p(\boldsymbol{x}_{t+1} \mid \boldsymbol{z}_{1:t}) = \sum_{i=1}^{K} \pi'^{(i)} \mathcal{N}(\boldsymbol{x}_{t+1} \mid f_{t+1}(\boldsymbol{\mu}_{t|t}^{(i)}), \hat{\boldsymbol{F}}_{t+1}^{(i)} \boldsymbol{\Sigma}_{t|t}^{(i)} \hat{\boldsymbol{F}}_{t+1}^{(i)\top} + \boldsymbol{\Sigma}_{v_t}) \tag{5.26}$$

In the predictive mixture above, if the plant covariance $\boldsymbol{\Sigma}_{v_t}$ is large compared to $\Sigma_{t|t}^{(i)}$, the variance of each of the Gaussians will be increased by the large plant noise that results in the mixtures having large overlaps with each other; this means that the linearization is no longer valid. Furthermore, the next measurement will cause the means of the Gaussians to be nearly the same, which will reduce the mixture filter into only one extended Kalman filter. To prevent this from happening, the large Gaussian plant noise is written as a summation of K_2 "thinner" Gaussians as

$$p(\boldsymbol{v}_t) = \sum_{j=1}^{K_2} \gamma^{(j)} \mathcal{N}(\boldsymbol{v}_t \mid \boldsymbol{\mu}_{v_t}^{(j)}, \boldsymbol{\Sigma}_{v_t}^{(j)}) \tag{5.27}$$

the linearization of dynamic function around $\boldsymbol{\mu}_{t|t}^{(i)}$ results in the state transition distribution as a summation of Gaussians

$$p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t) = \sum_{j=1}^{K_2} \gamma^{(j)} \mathcal{N}(\boldsymbol{x}_{t+1} \mid f_{t+1}(\boldsymbol{\mu}_{t|t}^{(i)}) + \hat{\boldsymbol{F}}_{t+1}^{(i)}(\boldsymbol{x}_t - \boldsymbol{\mu}_{t|t}^{(i)}) + \boldsymbol{\mu}_{v_t}^{(j)}, \boldsymbol{\Sigma}_{v_t}^{(j)}) \tag{5.28}$$

and the result of the predictive integral will now be a summation of $K_1 K_2$ Gaussians as

$$p(\boldsymbol{x}_{t+1} \mid \boldsymbol{z}_{1:t}) = \sum_{i=1}^{K_1} \sum_{j=1}^{K_2} \pi'^{(i)} \gamma^{(j)} \mathcal{N}(\boldsymbol{x}_{t+1} \mid \boldsymbol{\mu}_{t+1|t}^{(ij)}, \boldsymbol{\Sigma}_{t+1|t}^{(ij)}) \tag{5.29}$$

where

$$\begin{aligned} \boldsymbol{\mu}_{t+1|t}^{(ij)} &= f_{t+1}(\boldsymbol{\mu}_{t|t}^{(i)}) + \boldsymbol{\mu}_{v_t}^{(j)} \\ \boldsymbol{\Sigma}_{t+1|t}^{(ij)} &= \boldsymbol{\Sigma}_{v_t}^{(j)} + \hat{\boldsymbol{F}}_{t+1}^{(i)} \boldsymbol{\Sigma}_{t|t}^{(i)} \hat{\boldsymbol{F}}_{t+1}^{(i)\top} \end{aligned} \tag{5.30}$$

The above however will result in the number of the components of the next prediction step to grow by a factor of K_2, and thereafter resulting in an exponential growth of the number of components. It is suggested in [76] to reduce the number of mixtures by combining the similar components. The ever-increasing number of mixtures was also demonstrated in a similar setting in [77], which highlighted the need of merging similar components. As stated before this led to many publications on the topic of mixture reduction algorithms,

some of which were discussed earlier. In the next section, we propose a filter that avoids encountering this issue by utilizing the EM$^+$ algorithm presented earlier.

The Mixture of Gaussians Filter

1: initialize prior distribution on the state $p(\boldsymbol{x}_{t-1} \mid \boldsymbol{z}_{1:t-1}) = \sum_{i=1}^{K_1} \pi'^{(i)} \mathcal{N}(\boldsymbol{\mu}_{t-1|t-1}^{(i)}, \boldsymbol{\Sigma}_{t-1|t-1}^{(i)})$

2: write distribution on process noise as a mixture of K_2 components

$$p(\boldsymbol{v}_{t-1}) = \sum_{j=1}^{K_2} \gamma^{(j)} \mathcal{N}(\boldsymbol{v}_{t-1} \mid \boldsymbol{\mu}_{v_{t-1}}^{(j)}, \boldsymbol{\Sigma}_{v_{t-1}}^{(j)})$$

3: linearize f_t around $\boldsymbol{\mu}_{t-1|t-1}^{(i)}$, compute $\hat{\boldsymbol{F}}_t^{(i)}$, get prediction

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t-1}) = \sum_{i=1}^{K_1} \sum_{j=1}^{K_2} \pi'^{(i)} \gamma^{(j)} \mathcal{N}(\boldsymbol{x}_t \mid \boldsymbol{\mu}_{t|t-1}^{(ij)}, \boldsymbol{\Sigma}_{t|t-1}^{(ij)})$$
$$\boldsymbol{\mu}_{t|t-1}^{(ij)} = f_t(\boldsymbol{\mu}_{t-1|t-1}^{(i)}) + \boldsymbol{\mu}_{v_{t-1}}^{(j)}$$
$$\boldsymbol{\Sigma}_{t|t-1}^{(ij)} = \boldsymbol{\Sigma}_{v_{t-1}}^{(j)} + \hat{\boldsymbol{F}}_t^{(i)} \boldsymbol{\Sigma}_{t-1|t-1}^{(i)} \hat{\boldsymbol{F}}_t^{(i)\top}$$

4: measurement $\boldsymbol{z}_t$ comes in

5: reduce mixture in step 3 to K components, find new $\pi^{(i)}$, $\boldsymbol{\mu}_{t|t-1}^{(i)}$ and $\boldsymbol{\Sigma}_{t|t-1}^{(i)}$

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t-1}) = \sum_{i=1}^{K} \pi^{(i)} \mathcal{N}(\boldsymbol{x}_t \mid \boldsymbol{\mu}_{t|t-1}^{(i)}, \boldsymbol{\Sigma}_{t|t-1}^{(i)})$$

6: linearize h_t around $\boldsymbol{\mu}_{t|t-1}^{(i)}$, compute $\hat{\boldsymbol{H}}_t^{(i)}$, compute the EKF gain matrix for each component

$$\boldsymbol{K}_t^{(i)} = \boldsymbol{\Sigma}_{t|t-1}^{(i)} \boldsymbol{H}_t^{(i)\top} (\boldsymbol{H}_t^{(i)} \boldsymbol{\Sigma}_{t|t-1}^{(i)} \boldsymbol{H}_t^{(i)\top} + \boldsymbol{\Sigma}_{q_t})^{-1}$$

7: update by getting posterior

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t}) = \sum_{i=1}^{K} \pi'^{(i)} \mathcal{N}(\boldsymbol{\mu}_{t|t}^{(i)}, \boldsymbol{\Sigma}_{t|t}^{(i)})$$
$$\boldsymbol{\mu}_{t|t}^{(i)} = \boldsymbol{\mu}_{t|t-1}^{(i)} + \boldsymbol{K}_t^{(i)} (\boldsymbol{z}_t - h_t(\boldsymbol{\mu}_{t|t-1}^{(i)}))$$
$$\boldsymbol{\Sigma}_{t|t}^{(i)} = \boldsymbol{\Sigma}_{t|t-1}^{(i)} - \boldsymbol{K}_t^{(i)} \boldsymbol{H}_t^{(i)} \boldsymbol{\Sigma}_{t|t-1}^{(i)}$$
$$\pi_{\text{un}}'^{(i)} \propto \pi^{(i)} \mathcal{N}(\boldsymbol{z}_t \mid h_t(\boldsymbol{\mu}_{t|t-1}^{(i)}), \boldsymbol{H}_t^{(i)} \boldsymbol{\Sigma}_{t|t-1}^{(i)} \boldsymbol{H}_t^{(i)\top} + \boldsymbol{\Sigma}_{q_t})$$
$$\pi'^{(i)} = \frac{\pi_{\text{un}}'^{(i)}}{\sum_{j=1}^{K} \pi_{\text{un}}'^{(j)}}$$

8: estimate state based on $p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t})$

9: set $t = t + 1$, $K_1 = K$, set prior equal to the posterior, and go back to 2.

Table 5.3: The steps of the mixture of Gaussian filter

5.2 Proposed Filtering Method with EM$^+$

In the proposed method, the particle filter and the mixture of Gaussians filter are combined in a way that their advantages are exploited, while eliminating the disadvantages that both approaches suffer from. Assuming that the prior distribution, $p(\boldsymbol{x}_{t-1} \mid \boldsymbol{z}_{1:t-1})$ is known, N_s samples $\{\boldsymbol{x}_{t-1}^1 \dots \boldsymbol{x}_{t-1}^{N_s}\}$ are taken, which are then passed through the non-linear state transition function in equation (5.1) which results in

$$\boldsymbol{x}_t^i = f_t(\boldsymbol{x}_{t-1}^i) + \boldsymbol{v}_{t-1}^i \tag{5.31}$$

for $i = 1, \dots, N_s$, where $\boldsymbol{v}_{t-1}^i$ is a sample from $\mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_{v_{t-1}})$. The produced samples are essentially samples of the predictive distribution $p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t-1})$, which can then be used with EM$^+$ to construct the underlying distribution as

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t-1}) = \sum_{i=1}^{K} \pi^{(i)} \mathcal{N}(\boldsymbol{x}_t \mid \boldsymbol{\mu}_{t|t-1}^{(i)}, \boldsymbol{\Sigma}_{t|t-1}^{(i)}) \tag{5.32}$$

The flexibility of EM$^+$ allows for estimating the means and covariances, and for automatically inferring the number of mixture components needed above to construct a distribution that represents the samples well. Once the predictive distribution is represented as a mixture of Gaussians, the posterior is obtained by linearizing the measurement equation around $\boldsymbol{\mu}_{t|t-1}^{(i)}$, the mean of each component similar to the mixture of Gaussian filter discussed earlier. This will produce the same posterior distribution in equation (5.22), which is

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t}) = \sum_{i=1}^{K} \pi'^{(i)} \mathcal{N}(\boldsymbol{\mu}_{t|t}^{(i)}, \boldsymbol{\Sigma}_{t|t}^{(i)}) \tag{5.33}$$

where

$$\begin{aligned}
\boldsymbol{K}_t^{(i)} &= \boldsymbol{\Sigma}_{t|t-1}^{(i)} \boldsymbol{H}_t^{(i)\top} (\boldsymbol{H}_t^{(i)} \boldsymbol{\Sigma}_{t|t-1}^{(i)} \boldsymbol{H}_t^{(i)\top} + \boldsymbol{\Sigma}_{q_t})^{-1} \\
\boldsymbol{\mu}_{t|t}^{(i)} &= \boldsymbol{\mu}_{t|t-1}^{(i)} + \boldsymbol{K}_t^{(i)} (\boldsymbol{z}_t - h_t(\boldsymbol{\mu}_{t|t-1}^{(i)})) \\
\boldsymbol{\Sigma}_{t|t}^{(i)} &= \boldsymbol{\Sigma}_{t|t-1}^{(i)} - \boldsymbol{K}_t^{(i)} \boldsymbol{H}_t^{(i)} \boldsymbol{\Sigma}_{t|t-1}^{(i)} \\
\pi'^{(i)}_{\text{un}} &\propto \pi^{(i)} \mathcal{N}(\boldsymbol{z}_t \mid h_t(\boldsymbol{\mu}_{t|t-1}^{(i)}), \boldsymbol{H}_t^{(i)} \boldsymbol{\Sigma}_{t|t-1}^{(i)} \boldsymbol{H}_t^{(i)\top} + \boldsymbol{\Sigma}_{q_t}) \\
\pi'^{(i)} &= \frac{\pi'^{(i)}_{\text{un}}}{\sum_{j=1}^{K} \pi'^{(j)}_{\text{un}}}
\end{aligned} \tag{5.34}$$

In the next iteration of the filter, the posterior above becomes the prior mixture distribution, which is then sampled again N_s times. These samples are passed through the non-linear state transition function as mentioned above and the results are used to construct the predictive distribution with the EM$^+$ algorithm. By using samples to represent the predictive distribution, the state transition function is not linearized and the problem of the ever-increasing number of Gaussians is averted. Furthermore, the need for a proposal distribution and the problems of degeneracy and collapse of the particles in the particle filter are avoided. The steps of the proposed filter are given the Table 5.4. In the next section, we compare the results of the EKF, the particle filter and the proposed filter on a simple example commonly used in the filtering literature.

The Proposed Filter

1: initialize prior distribution on the state $p(\boldsymbol{x}_{t-1} \mid \boldsymbol{z}_{1:t-1}) = \sum_{i=1}^{K_1} \pi'^{(i)} \mathcal{N}(\boldsymbol{\mu}_{t-1|t-1}^{(i)}, \boldsymbol{\Sigma}_{t-1|t-1}^{(i)})$

2: take N_s samples from the prior $\{\boldsymbol{x}_{t-1}^1, \ldots, \boldsymbol{x}_{t-1}^i, \ldots, \boldsymbol{x}_{t-1}^{N_s}\}$

3: get samples from predictive distribution by passing each samples through state transition function as $\boldsymbol{x}_t^i = f_t(\boldsymbol{x}_{t-1}^i) + \boldsymbol{v}_{t-1}^i$, where $\boldsymbol{v}_{t-1}^i$ is a sample from $\mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_{v_{t-1}})$

4: measurement $\boldsymbol{z}_t$ comes in

5: represent the samples in step 3 with a mixture of Gaussians distribution using EM$^+$ algorithm

$$p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t-1}) = \sum_{i=1}^{K} \pi^{(i)} \mathcal{N}(\boldsymbol{x}_t \mid \boldsymbol{\mu}_{t|t-1}^{(i)}, \boldsymbol{\Sigma}_{t|t-1}^{(i)})$$

6: linearize h_t around $\boldsymbol{\mu}_{t|t-1}^{(i)}$, compute $\hat{\boldsymbol{H}}_t^{(i)}$, compute the gain matrix for each component

$$\boldsymbol{K}_t^{(i)} = \boldsymbol{\Sigma}_{t|t-1}^{(i)} \boldsymbol{H}_t^{(i)\top} (\boldsymbol{H}_t^{(i)} \boldsymbol{\Sigma}_{t|t-1}^{(i)} \boldsymbol{H}_t^{(i)\top} + \boldsymbol{\Sigma}_{q_t})^{-1}$$

7: update by getting posterior

$$\begin{aligned}
p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t}) &= \sum_{i=1}^{K} \pi'^{(i)} \mathcal{N}(\boldsymbol{\mu}_{t|t}^{(i)}, \boldsymbol{\Sigma}_{t|t}^{(i)}) \\
\boldsymbol{\mu}_{t|t}^{(i)} &= \boldsymbol{\mu}_{t|t-1}^{(i)} + \boldsymbol{K}_t^{(i)} (\boldsymbol{z}_t - h_t(\boldsymbol{\mu}_{t|t-1}^{(i)})) \\
\boldsymbol{\Sigma}_{t|t}^{(i)} &= \boldsymbol{\Sigma}_{t|t-1}^{(i)} - \boldsymbol{K}_t^{(i)} \boldsymbol{H}_t^{(i)} \boldsymbol{\Sigma}_{t|t-1}^{(i)} \\
\pi_{\text{un}}'^{(i)} &\propto \pi^{(i)} \mathcal{N}(\boldsymbol{z}_t \mid h_t(\boldsymbol{\mu}_{t|t-1}^{(i)}), \boldsymbol{H}_t^{(i)} \boldsymbol{\Sigma}_{t|t-1}^{(i)} \boldsymbol{H}_t^{(i)\top} + \boldsymbol{\Sigma}_{q_t}) \\
\pi'^{(i)} &= \frac{\pi_{\text{un}}'^{(i)}}{\sum_{j=1}^{K} \pi_{\text{un}}'^{(j)}}
\end{aligned}$$

8: estimate state based on $p(\boldsymbol{x}_t \mid \boldsymbol{z}_{1:t})$

9: set prior equal to the posterior, and go back to 2.

Table 5.4: The steps of the proposed filter

5.3 Results

The benchmark non-linear time series model that is used here has been analyzed before in many publications [92]. The state space and measurement equations of the model are

$$\begin{aligned} x_t &= f_t(x_{t-1}, t) + v_{t-1} \\ z_t &= \frac{(x_t)^2}{20} + w_t \end{aligned} \tag{5.35}$$

where

$$\begin{aligned} f_t(x_{t-1}, t) &= \frac{x_{t-1}}{2} + \frac{25x_{t-1}}{1 + x_{t-1}^2} + 8\cos(1.2t) \\ v_{t-1} &\sim \mathcal{N}(0, 10) \\ w_t &\sim \mathcal{N}(0, 1) \end{aligned} \tag{5.36}$$

[92] reports the results of the EKF and three variations of the particle filter mentioned earlier, the SIR, the APF, and the RPF. In addition to this, a filter named the "likelihood" particle filter which uses the likelihood in the proposal distribution, is designed to work well for this specific example. Table 5.5 shows the root mean squared error (RMSE) values averaged over 100 MC runs, where each run consists of 100 time steps. The particle filters each use 50 particles, and are resampled at every iteration. It can be seen that a single EKF gives poor results. The results of the SIR, and the Regularized particle filters are close to each other. The APF performs better than both of these filter, and the "likelihood" particle filter has the best performance. For this problem, since the likelihood is far tighter than the prior, the posterior will look more similar to the likelihood, therefore using an importance density based on the likelihood that resembles the posterior will produce better results. It is important to note however that this is not in general true and the good results produced by this filter on this example is only used to show the importance of the proposal distribution in the performance of the particle filter [92].

The proposed filter with the use of the EM^+ algorithm uses 50 samples in step 3 of Table 5.4. The result of the proposed filter shown in Table 5.5 significantly improves the RMSE value over a single Extended Kalman filter. Moreover there is an improvement over the general-purpose SIR, the APF and the RPF. The results of the proposed filter is slightly better than the "likelihood" particle filter that was designed to work well for this problem.

Algorithm	Root Mean Squared Error
Extended Kalman filter (EKF)*	23.19
Sampling importance sampling particle filter (SIR) [93]*	5.54
Auxiliary particle filter (APF) [95]*	5.35
Regularized particle filter (RPF) [94]*	5.55
"Likelihood" particle filter [92]*	5.30
Proposed filter with EM^{+}	5.29

Table 5.5: RMSE values of different filtering algorithms averaged over 100 MC runs. The results of * are taken from [92].

5.3.1 Summary

A filter that combines the advantages of the particle filter and the mixture of Gaussians filter, while avoiding the disadvantages of the two was presented. The use of samples in the prediction step of the filter avoids the problem of the ever-increasing number of components in the mixture of Gaussians filter. Furthermore, the main issues of the particle filter, which are the dependency of its performance on the choice of the proposal distribution, the degeneracy in the particles over time, and the collapse of particles are averted by having an explicit distribution of the prior as a mixture of Gaussians for sampling in the prediction step of the filter. The results in Table 5.5 show that the performance of the proposed filter is better than both the general purpose, and the customized particle filter that were used on a non-linear example shown in the previous section. The proposed filter can be used for non-linear problems and is able to represent the multimodality of the distributions at the prediction and update steps of the filter by inferring the number of Gaussians needed using the EM^{+} algorithm. This makes it applicable to a wide range of filtering problems without the need of modifications.

Appendix A

Appendix

A.1 Mathematical Background

A.1.1 The Gaussian Distribution

A multivariate Gaussian distribution of $\boldsymbol{x} \in \mathbb{R}^D$ with mean $\boldsymbol{\mu} \in \mathbb{R}^D$ and covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{D \times D}$ is given by

$$p(\boldsymbol{x}) = (2\pi)^{-\frac{D}{2}} |\boldsymbol{\Sigma}|^{-\frac{1}{2}} \exp(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})) \tag{A.1}$$

and is written as $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$

The standard normal probability density function is defined as

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}x^2) \tag{A.2}$$

The standard normal cumulative distribution function is defined as

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp(-\frac{t^2}{2})dt \tag{A.3}$$

which can be written in terms of the error function as

$$\Phi(x) = \frac{1}{2}\left[1 + \text{erf}(\frac{x}{\sqrt{2}})\right] \tag{A.4}$$

if $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are jointly Gaussian,

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}) \tag{A.5}$$

then the conditional distributions of $\boldsymbol{x}_1$ given $\boldsymbol{x}_2$ is

$$p(\boldsymbol{x}_1|\boldsymbol{x}_2) = \mathcal{N}(\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}_2 - \boldsymbol{\mu}_2), \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}) \tag{A.6}$$

A.1.2 Multivariate T-distribution

In general the multivariate t-distribution is written as

$$\mathcal{T}(\boldsymbol{x} \mid \boldsymbol{\mu}_{\mathcal{T}}, \boldsymbol{\Sigma}_{\mathcal{T}}, \nu) = \frac{\Gamma(\frac{\nu+D}{2})}{\Gamma(\frac{\nu}{2})} |\pi\nu\boldsymbol{\Sigma}_{\mathcal{T}}|^{-\frac{1}{2}} \left[1 + (\boldsymbol{x} - \boldsymbol{\mu}_{\mathcal{T}})^T (\nu\boldsymbol{\Sigma}_{\mathcal{T}})^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_{\mathcal{T}})\right]^{-\frac{\nu+D}{2}} \tag{A.7}$$

For the set of $\boldsymbol{\lambda} = \{m_0, \boldsymbol{S}_0, \kappa_0, \nu_0\}$ used in this work the t-distribution can be written as

$$\begin{aligned} \mathcal{T}(\boldsymbol{x} \mid \boldsymbol{\lambda}) &= \mathcal{T}(\boldsymbol{x} \mid \boldsymbol{m}_0, \frac{\kappa_0 + 1}{\kappa_0 \underbrace{\nu_0 - D + 1}_{\nu}} \boldsymbol{S}_0, \underbrace{\nu_0 - D + 1}_{\nu}) \\ &= \frac{\Gamma(\frac{\nu+D}{2})}{\Gamma(\frac{\nu}{2})} (\pi \frac{\kappa_0 + 1}{\kappa_0})^{-\frac{D}{2}} |\boldsymbol{S}_0|^{-\frac{1}{2}} \left[1 + \frac{\kappa_0}{\kappa_0 + 1} (\boldsymbol{x} - \boldsymbol{m}_0)^T (\boldsymbol{S}_0)^{-1} (\boldsymbol{x} - \boldsymbol{m}_0)\right]^{-\frac{\nu+D}{2}} \end{aligned} \tag{A.8}$$

A.1.3 Matrix Algebra

For non-singular square matrix $\boldsymbol{A}$, and $\boldsymbol{D} - \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B}$, the following block-wise inversion property holds

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{A}^{-1} + \boldsymbol{A}^{-1}\boldsymbol{B}(\boldsymbol{D} - \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B})^{-1}\boldsymbol{C}\boldsymbol{A}^{-1} & -\boldsymbol{A}^{-1}\boldsymbol{B}(\boldsymbol{D} - \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B})^{-1} \\ -(\boldsymbol{D} - \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B})^{-1}\boldsymbol{C}\boldsymbol{A}^{-1} & (\boldsymbol{D} - \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B})^{-1} \end{bmatrix} \tag{A.9}$$

A.2 Combining Two Uncertain Estimates

Two independent estimates of an unknown from different sources with different uncertainties (both Gaussian) can be combined to create a better estimate. If $\{V_1, \sigma_1^2\}$ and $\{V_2, \sigma_2^2\}$ are two uncertain estimates of V, it is desired to obtain an estimate of the mean of the combined measurement. The joint likelihood, or the conditional probability of obtaining the measurement given V is

$$\begin{aligned} \mathcal{L}(V) &= p(V_1, V_2 \mid V) \\ &= p(V1 \mid V)p(V2 \mid V) \end{aligned} \tag{A.10}$$

a good estimate of V is then a value that maximizes the likelihood above

$$\hat{V} = \underset{V}{\operatorname{argmax}}\, \mathcal{L}(V) \tag{A.11}$$

this is solved by taking the derivative of equation (A.10) w.r.t V and setting it to zero. This operation would be easier on the log-likelihood which is proportional to

$$\log(\mathcal{L}(V)) \propto -\frac{(V_1 - V)^2}{2\sigma_1^2} - \frac{(V_2 - V)^2}{2\sigma_2^2} \tag{A.12}$$

and

$$\frac{\partial \mathcal{L}(V)}{\partial V} = \frac{(V_1 - V)}{\sigma_1^2} - \frac{(V_2 - V)}{\sigma_2^2} = 0 \tag{A.13}$$

which gives

$$\hat{V} = \frac{\frac{V_1}{\sigma_1^2} + \frac{V_2}{\sigma_2^2}}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}} \tag{A.14}$$

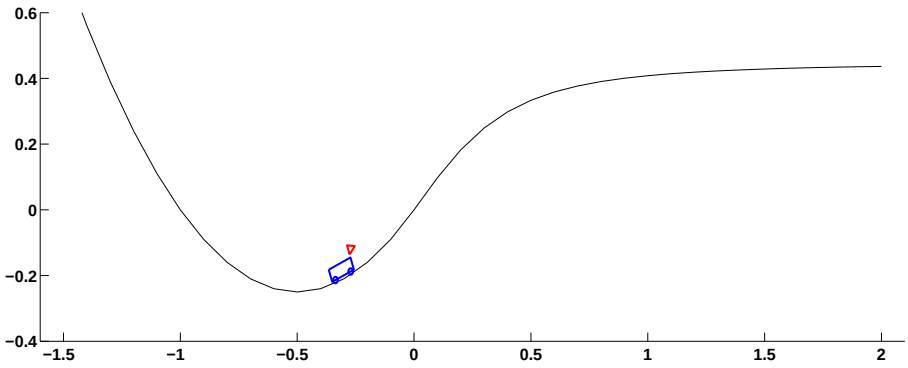
the variance of the estimate above is

$$\mathrm{Var}[\hat{V}] = \frac{1}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}} \tag{A.15}$$

A.3 Equations of Motion

A.3.1 Mountain Car

Figure A.1: The Mountain Car



The mountain is described by the following equation

$$H(x) = \begin{cases} x^2 + x & x < 0 \\ \frac{x}{\sqrt{1+5x^2}} & x \geq 0 \end{cases}$$

and the dynamics of the car (unknown to the algorithm) is given by:

$$\dot{x} = \frac{a}{M\sqrt{1 + H'(x)^2}} - \frac{gH'(x)}{1 + H'(x)^2} \tag{A.16}$$

where $M = 1$ kg and $g = 9.8\ \frac{m}{s^2}$.

A.3.2 Inverted Pendulum

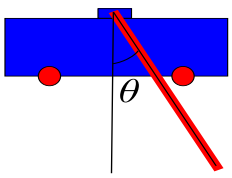


Figure A.2: The inverted pendulum set up

The equations of motion of the inverted pendulum system are the following

$$
\begin{aligned}
(m_c + m_p)\ddot{x} + \frac{1}{2} m_p l \ddot{\theta} \cos\theta - \frac{1}{2} m_p l \dot{\theta}^2 \sin\theta &= u - b\dot{x} \\
2l\ddot{\theta} + 3\ddot{x}\cos\theta + 3g\sin\theta &= 0
\end{aligned}
\tag{A.17}
$$

where m_c and m_p are the masses of the cart and the pendulum respectively, u is the force applied to the cart, and b is the kinetic friction coefficient. Please refer to [46] for the derivation of the equations of motion.

A.4 Gibbs Sampling for Dirichlet Process Mixture Models

Here we summarize the non-conjugate model given in [66], along with the Gibbs sampling technique used to update each of the parameters (Algorithm 8 in [67]). The process given here is for a 1-D case but can be easily extended for higher dimensions.

The prior on the mean is Gaussian taken to be

$$p(\mu_k \mid \lambda, r) = \mathcal{N}(\lambda, r^{-1}) \tag{A.18}$$

where λ and r are hyperparameters that are updated later. Their priors are broad (vague) Gaussians and gamma taken to be

$$p(\lambda) = \mathcal{N}(\mu_x, \sigma_x^2) \qquad p(r) = \mathcal{G}(1, \sigma_x^{-2}) \tag{A.19}$$

where μ_y and σ_y^2 are the mean and variance of the observations.

The prion on the precision, s_k, is given by

$$p(s_k \mid \beta, \omega) = \mathcal{G}(\beta, \omega^{-1}) \tag{A.20}$$

with hyperparameters having priors

$$p(\beta^{-1}) = \mathcal{G}(1,1) \Rightarrow p(\beta) \propto \beta^{-\frac{3}{2}} \exp(-\frac{1}{2\beta}) \qquad p(\omega) = \mathcal{G}(1, \sigma_y^2) \tag{A.21}$$

The prior on α, the concentration parameter of the DP is

$$p(\alpha) = \mathcal{G}(1,1) \tag{A.22}$$

This hyperparameter will also be updated below.

A.4.1 Monte Carlo Sampling

In a DP, the upper bound on the number of clusters is infinity, but in practice only a number of clusters will be representer, i.e. have observations associated with them. Here the number of represented clusters at any given time will be referred to as k_{ref}. One iteration of the Gibbs sampling based on the conditional posteriors is:

1. For $i = 1 : \text{n}$

   sample indicator variable c_i according to Algorithm 8 in [67]

   End

2. Update k_{rep}, the number of represented mixtures

3. For $k = 1 : k_{rep}$

   Update N_k, the number of elements belonging to cluster k

   Update mixing weights: $\pi_k = \frac{N_k}{N+\alpha}$

   End

   Update overall weight of unrepresented mixtures, $\pi = \frac{\alpha}{N+\alpha}$

4. For $j = 1 : k_{rep}$

   Sample μ_k from its conditional distribution:

$$p(\mu_k \mid \boldsymbol{c}, \boldsymbol{X}, s_k, \lambda, r) = \mathcal{N}\left(\frac{\bar{x}_k N_k \sigma^2 + \lambda r}{N_k \sigma^2 + r}, \frac{1}{N_k \sigma^2 + r}\right)$$

$$p(s_k \mid \boldsymbol{c}, \boldsymbol{X}, \mu_k, \beta, \omega) = \mathcal{G}\left(\beta + N_k, \left[\frac{1}{\beta + N_k}(\omega\beta + \sum_{i:c_i=k}(x_i - \mu_k)^2)\right]^{-1}\right)$$

   where $\bar{x}_k = \frac{1}{N_k} \sum_{i:c_k=j} x_i$, the members in cluster k.

   End

5. Update hyperparameters by sampling from their conditional posteriors:

$$p(\lambda \mid \boldsymbol{\mu}, r) = \mathcal{N}\left(\frac{\mu_x\sigma_x^{-2} + r\sum_{k=1}^{k_{rep}}\mu_k}{\sigma_x^{-2} + k_{rep}r}, \frac{1}{\sigma_x^{-2} + k_{rep}r}\right)$$

$$p(r \mid \boldsymbol{\mu}, \lambda) = \mathcal{G}\left(k_{rep}+1, \left[\frac{1}{k_{rep}+1}\left(\sigma_x^2 + \sum_{k=1}^{k_{rep}}(\mu_k - \lambda)^2\right)\right]^{-1}\right)$$

$$p(\omega \mid \boldsymbol{s}, \beta) = \mathcal{G}\left(k_{rep}\beta+1, \left[\frac{1}{k_{rep}\beta+1}(\sigma_x^{-2} + \beta\sum_{k=1}^{k_{rep}}s_k)\right]^{-1}\right)$$

$$p(\beta \mid \boldsymbol{s}, \omega) \propto \Gamma(\frac{\beta}{2})^{-k_{rep}} \exp\left(\frac{-1}{2\beta}\right)\left(\frac{\beta}{2}\right)^{\frac{k_{rep}\beta-3}{2}} \prod_{k=1}^{k_{rep}}(s_k\omega)^{\frac{\beta}{2}} \exp\left(\frac{-\beta s_k\omega}{2}\right)$$

$$p(\alpha \mid k_{rep}, N) \propto \frac{\alpha^{k_{rep}-\frac{3}{2}}\exp(-\frac{1}{2\alpha})\Gamma(\alpha)}{\Gamma(N+\alpha)}$$

$p(\alpha \mid k_{rep}, N)$ is log-concave and can be sampled using Gibbs sampling methods such as the adaptive rejection sampling (ARS) presented in [96].

In step 1, the indicator variables are sampled from the posterior given by

$$p(c_i = k \mid \boldsymbol{c}_{-i}, \mu_k, s_k, \alpha) \propto \begin{cases} \frac{N_{-i,k}}{N-1+\alpha} s_k^{\frac{1}{2}} \exp(-\frac{s_k}{2}(x_i - \mu_k)^2) & k \text{ rep} \\ \\ \frac{\alpha}{N-1+\alpha}\int p(x_i \mid \mu_k, s_k)p(\mu_k, s_k \mid \lambda, r, \beta, \omega)d\mu_k ds_k & k \text{ unrep} \end{cases}$$

Gibbs sampling is performed only for those θ_k's that are currently have observations associated with them [67].

Bibliography

[1] R. S. Sutton and A. G. Barto, "Reinforcement learning: Introduction," 1998.

[2] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement learning and dynamic programming using function approximators*. CRC press, 2010.

[3] C. G. Atkeson and J. C. Santamaria, "A comparison of direct and model-based reinforcement learning," in *In International Conference on Robotics and Automation*. IEEE Press, 1997, pp. 3557–3564.

[4] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Proceedings of the seventh international conference on machine learning*, 1990, pp. 216–224.

[5] A. Agostini and E. Celaya, "Reinforcement learning with a gaussian mixture model." in *IJCNN*. IEEE, 2010, pp. 1–8.

[6] C. K. I. W. Carl Edward Rasmussen, "Gaussian processes for machine learning." MIT Press, 2006.

[7] B. Matérn, *Spatial variation*. Springer Science & Business Media, 2013, vol. 36.

[8] M. N. Gibbs, "Bayesian gaussian processes for regression and classification," Ph.D. dissertation, University of Cambridge, 1998.

[9] C. J. Paciorek, "Nonstationary gaussian processes for regression and spatial modelling," Ph.D. dissertation, Carnegie Mellon University, 2003.

[10] C. Paciorek and M. Schervish, "Nonstationary covariance functions for gaussian process regression," *Advances in neural information processing systems*, vol. 16, pp. 273–280, 2004.

[11] M. G. Genton, "Classes of kernels for machine learning: a statistics perspective," *The Journal of Machine Learning Research*, vol. 2, pp. 299–312, 2002.

[12] D. J. C. MacKay, "Hyperparameters: optimize, or integrate out?" in *In Maximum Entropy and Bayesian Methods, Santa Barbara*. Kluwer, 1996, pp. 43–60.

[13] J. Quiñonero-candela, C. E. Rasmussen, and R. Herbrich, "A unifying view of sparse approximate gaussian process regression," *Journal of Machine Learning Research*, vol. 6, p. 2005, 2005.

[14] E. Snelson and Z. Ghahramani, "Sparse gaussian processes using pseudo-inputs," in *Advances in Neural Information Processing Systems 18*. MIT press, 2006, pp. 1257–1264.

[15] D. Nguyen-Tuong, M. Seeger, and J. Peters, "Model learning with local gaussian process regression," no. 15, pp. 2015–2034, 2009.

[16] E. Brochu, V. M. Cora, and O. D. Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," Tech. Rep., 2009.

[17] D. R. Jones, "A taxonomy of global optimization methods based on response surfaces," *Journal of global optimization*, vol. 21, no. 4, pp. 345–383, 2001.

[18] A. Žilinskas, "On the use of statistical models of multimodal functions for the construction of the optimization algorithms," in *Optimization Techniques*. Springer, 1980, pp. 138–147.

[19] A. O'Hagan and J. Kingman, "Curve fitting and optimal design for prediction," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 1–42, 1978.

[20] J. Mockus, V. Tiesis, and A. Zilinskas, "The application of bayesian methods for seeking the extremum," *Towards global optimization*, vol. 2, no. 117-129, p. 2, 1978.

[21] D. Büche, N. N. Schraudolph, and P. Koumoutsakos, "Accelerating evolutionary algorithms with gaussian process fitness function models," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 35, no. 2, pp. 183–194, 2005.

[22] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and N. de Freitas, "Bayesian optimization in a billion dimensions via random embeddings," *arXiv preprint arXiv:1301.1942*, 2013.

[23] H. J. Kushner, "A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise," *Journal of Basic Engineering*, vol. 86, no. 1, pp. 97–106, 1964.

[24] J. Mockus, "Application of bayesian approach to numerical methods of global and stochastic optimization," *Journal of Global Optimization*, vol. 4, no. 4, pp. 347–365, 1994.

[25] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[26] E. Vazquez and J. Bect, "Convergence properties of the expected improvement algorithm with fixed mean and covariance functions," *Journal of Statistical Planning and inference*, vol. 140, no. 11, pp. 3088–3095, 2010.

[27] A. D. Bull, "Convergence rates of efficient global optimization algorithms," *The Journal of Machine Learning Research*, vol. 12, pp. 2879–2904, 2011.

[28] M. Locatelli, "Bayesian algorithms for one-dimensional global optimization," *Journal of Global Optimization*, vol. 10, no. 1, pp. 57–76, 1997.

[29] Z. Wang and N. de Freitas, "Theoretical analysis of bayesian optimisation with unknown gaussian process hyper-parameters," *arXiv preprint arXiv:1406.7758*, 2014.

[30] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[31] G. A. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," 1994.

[32] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge England, 1989.

[33] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[34] J. S. Albus, "A new approach to manipulator control: The cerebellar model articulation controller (cmac)," *Journal of Dynamic Systems, Measurement, and Control*, vol. 97, no. 3, pp. 220–227, 1975.

[35] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," *Advances in neural information processing systems*, pp. 1038–1044, 1996.

[36] S. P. Singh and R. S. Sutton, "Reinforcement learning with replacing eligibility traces," *Machine learning*, vol. 22, no. 1-3, pp. 123–158, 1996.

[37] J. Boyan and A. W. Moore, "Generalization in reinforcement learning: Safely approximating the value function," *Advances in neural information processing systems*, pp. 369–376, 1995.

[38] M. Riedmiller, "Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method," in *Machine Learning: ECML 2005*. Springer, 2005, pp. 317–328.

[39] R. M. Kretchmar and C. W. Anderson, "Comparison of cmacs and radial basis functions for local function approximators in reinforcement learning," in *Neural Networks, 1997., International Conference on*, vol. 2. IEEE, 1997, pp. 834–837.

[40] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with gaussian processes," in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 201–208.

[41] T. Raiko and M. Tornio, "Variational bayesian learning of nonlinear hidden state-space models for model predictive control," *Neurocomputing*, vol. 72, no. 16, pp. 3704–3712, 2009.

[42] H. Valpola and J. Karhunen, "An unsupervised ensemble learning method for nonlinear dynamic state-space models," *Neural computation*, vol. 14, no. 11, pp. 2647–2692, 2002.

[43] K. Doya, "Reinforcement learning in continuous time and space," *Neural computation*, vol. 12, no. 1, pp. 219–245, 2000.

[44] C. E. Rasmussen and M. Kuss, "Gaussian processes in reinforcement learning," in *Advances in Neural Information Processing Systems 16*. MIT Press, 2004, pp. 751–759.

[45] M. P. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *In Proceedings of the International Conference on Machine Learning*, 2011.

[46] M. Deisenroth, *Efficient Reinforcement Learning Using Gaussian Processes*. kit scientific publishing, 2010.

[47] M. Frean and P. Boyle, "Using gaussian processes to optimize expensive functions," *Advances in Artificial Intelligence*, vol. 5360, 2008.

[48] P. Boyle, "Gaussian processes for regression and optimisation," Ph.D. dissertation, Victoria University of Wellington, 2007.

[49] A. Girard, J. Q. Candela, R. Murray-smith, and C. E. Rasmussen, "Gaussian process priors with uncertain inputs - application to multiple-step ahead time series forecasting," 2003.

[50] B. Kulis and M. I. Jordan, "Revisiting k-means: New algorithms via bayesian nonparametrics," *arXiv preprint arXiv:1111.0352*, 2011.

[51] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *CoRR*, vol. abs/1206.2944, 2012.

[52] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, "Lipschitzian optimization without the lipschitz constant," *Journal of Optimization Theory and Applications*, vol. 79, no. 1, pp. 157–181, 1993.

[53] L. M. Rios and N. V. Sahinidis, "Derivative-free optimization: a review of algorithms and comparison of software implementations," *Journal of Global Optimization*, vol. 56, no. 3, pp. 1247–1293, 2013.

[54] A. R. Kan and G. Timmer, "Stochastic global optimization methods part i: Clustering methods," *Mathematical programming*, vol. 39, no. 1, pp. 27–56, 1987.

[55] M. J. Powell, "A direct search optimization method that models the objective and constraint functions by linear interpolation," in *Advances in optimization and numerical analysis*. Springer, 1994, pp. 51–67.

[56] A. W. Moore, "Variable resolution dynamic programming: Efficiently learning action maps in multivariate realvalued state-spaces," in *Proceedings of the Eighth International Conference on Machine Learning*, 1991, pp. 333–337.

[57] H. Kimura and S. Kobayashi, "Efficient non-linear control by combining q-learning with local linear controllers," in *Proceedings of the Sixteenth International Conference on Machine Learning*, ser. ICML '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 210–219.

[58] M. Riedmiller, "Neural reinforcement learning to swing-up and balance a real pole," in *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, vol. 4. IEEE, 2005, pp. 3191–3196.

[59] 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)*, pp. 1–38, 1977.

[60] C. Fraley and A. E. Raftery, "Bayesian regularization for normal mixture estimation and model-based clustering," *Journal of Classification*, vol. 24, no. 2, pp. 155–181, 2007.

[61] C. M. Bishop, *Pattern recognition and machine learning*. springer New York, 2006, vol. 4, no. 4.

[62] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[63] C. Fraley and A. E. Raftery, "Bayesian regularization for normal mixture estimation and model-based clustering," University of Washington, Technical Report 486, August 2005.

[64] B. Kulis and M. I. Jordan, "Revisiting k-means: New algorithms via bayesian nonparametrics," *CoRR*, vol. abs/1111.0352, 2011.

[65] D. Görür and C. Edward Rasmussen, "Dirichlet process gaussian mixture models: Choice of the base distribution," *Journal of Computer Science and Technology*, vol. 25, no. 4, pp. 653–664, 2010.

[66] C. E. Rasmussen, "The infinite gaussian mixture model," in *In Advances in Neural Information Processing Systems 12*. MIT Press, 2000, pp. 554–560.

[67] R. M. Neal, "Markov chain sampling methods for dirichlet process mixture models," *Journal of Computational and Graphical Statistics*, vol. 9, no. 2, pp. 249–265, 2000.

[68] T. S. Ferguson, "A bayesian analysis of some nonparametric problems," *The annals of statistics*, pp. 209–230, 1973.

[69] J. Sethuraman, "A constructive definiition of dirichlet priors," *Statistica sinica*, vol. 4, pp. 639–650, 1994.

[70] F. Caron, M. Davy, A. Doucet, E. Duflos, and P. Vanheeghe, "Bayesian inference for linear dynamic models with dirichlet process mixtures," *Signal Processing, IEEE Transactions on*, vol. 56, no. 1, pp. 71–84, 2008.

[71] A. E. Gelfand, "Gibbs sampling," *Journal of the American Statistical Association*, vol. 95, no. 452, pp. 1300–1304, December 2000.

[72] M. West, P. Müller, and M. D. Escobar, "Hierarchical priors and mixture models, with application in regression and density estimation," *Aspects of Uncertainty: A tribute to D.V. Lindley*, pp. 363–386, 1994.

[73] H. Akaike, "A new look at the statistical model identification," *Automatic Control, IEEE Transactions on*, vol. 19, no. 6, pp. 716–723, 1974.

[74] G. Schwarz *et al.*, "Estimating the dimension of a model," *The annals of statistics*, vol. 6, no. 2, pp. 461–464, 1978.

[75] K. L. Nylund, T. Asparouhov, and B. O. Muthén, "Deciding on the number of classes in latent class analysis and growth mixture modeling: A monte carlo simulation study," *Structural equation modeling*, vol. 14, no. 4, pp. 535–569, 2007.

[76] D. L. Alspach, "Nonlinear bayesian estimation using gaussian sum approximations," *IEEE Transactions on Automatic Control*, vol. Ac-17, no. 4, pp. 439–448, August 1972.

[77] D. J. Salmond, "Mixture reduction algorithms for target tracking in clutter," in *OE/LASE'90, 14-19 Jan., Los Angeles, CA*. International Society for Optics and Photonics, 1990, pp. 434–445.

[78] J. L. Williams and P. S. Maybeck, "Cost-function-based hypothesis control techniques for multiple hypothesis tracking," *Mathematical and Computer Modelling*, vol. 43, no. 9, pp. 976–989, 2006.

[79] A. R. Runnalls, "Kullback-leibler approach to gaussian mixture reduction," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 43, no. 3, pp. 989–999, 2007.

[80] D. Schieferdecker and M. F. Huber, "Gaussian mixture reduction via clustering," in *Information Fusion, 2009. FUSION'09. 12th International Conference on.* IEEE, 2009, pp. 1536–1543.

[81] D. Crouse, P. Willett, K. Pattipati, and L. Svensson, "A look at gaussian mixture reduction algorithms," in *Information Fusion (FUSION), 2011 Proceedings of the 14th International Conference on.* IEEE, 2011, pp. 1–8.

[82] N. Vlassis and A. Likas, "A greedy em algorithm for gaussian mixture learning," *Neural processing letters*, vol. 15, no. 1, pp. 77–87, 2002.

[83] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern recognition*, vol. 36, no. 2, pp. 451–461, 2003.

[84] N. Ueda, R. Nakano, Z. Ghahramani, and G. E. Hinton, "Smem algorithm for mixture models," *Neural computation*, vol. 12, no. 9, pp. 2109–2128, 2000.

[85] M. A. Figueiredo and A. K. Jain, "Unsupervised learning of finite mixture models," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 3, pp. 381–396, 2002.

[86] H. Attias, "A variational bayesian framework for graphical models," *Advances in neural information processing systems*, vol. 12, no. 1-2, pp. 209–215, 2000.

[87] M. J. Beal, Z. Ghahramani *et al.*, "Variational bayesian learning of directed graphical models with hidden variables," *Bayesian Analysis*, vol. 1, no. 4, pp. 793–831, 2006.

[88] V. Šmídl and A. Quinn, *The variational Bayes method in signal processing.* Springer Science & Business Media, 2006.

[89] R. E. Kass and A. E. Raftery, "Bayes factors," *Journal of the american statistical association*, vol. 90, no. 430, pp. 773–795, 1995.

[90] D. H. Wolpert and C. E. M. Strauss, *Maximum Entropy and Bayesian Methods: Santa Barbara, California, U.S.A., 1993.* Dordrecht: Springer Netherlands, 1996, ch. What Bayes has to Say about the Evidence Procedure, pp. 61–78.

[91] D. J. MacKay, "Comparison of approximate methods for handling hyperparameters," *Neural computation*, vol. 11, no. 5, pp. 1035–1068, 1999.

[92] M. S. Arulampalam, S. Maskell, and N. Gordon, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, February 2002.

[93] N. J. Gordon, D. J. Salmond, and A. F. Smith, "Novel approach to nonlinear/non-gaussian bayesian state estimation," in *Radar and Signal Processing, IEE Proceedings F*, vol. 140, no. 2. IET, 1993, pp. 107–113.

[94] C. Musso, N. Oudjane, and F. LeGland, "Improving regularised particle filters," *Sequential Monte Carlo Methods in Practice*, vol. A. Doucet, J. F.G. de Freitas, and N. J. Gordon, Eds. New York: Springer-Verlag, 2001.

[95] M. K. Pitt and N. Shephard, "Filtering via simulation: Auxiliary particle filters," *Journal of the American statistical association*, vol. 94, no. 446, pp. 590–599, 1999.

[96] W. R. Gilks and P. Wild, "Adaptive rejection sampling for gibbs sampling," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 41, no. 2, pp. 337–348, 1992.