Karlsruhe Institute of Technology

Recursive Surrogate-Modeling for Stochastic Search

Bachelor's Thesis
of

Klaus Philipp Theyssen

KIT Department of Informatics
Institute for Anthropomatics and Robotics (IAR)
Autonomous Learning Robots (ALR)

Referees: Prof. Dr. Techn. Gerhard Neumann
Prof. Dr. Ing. Tamim Asfour

Advisor: M.Sc. Maximilian Hüttenrauch

Duration: November 27th, 2020 — March 26th, 2021

Erklärung

Ich versichere hiermit, dass ich die Arbeit selbstständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis beachtet habe.

Karlsruhe, den 26. März 2021

Klaus Philipp Theyssen

Zusammenfassung

Roboter werden in den nächsten Jahrzehnten zunehmend unseren Alltag durchdringen. Sie werden ein breites Spektrum an Aufgaben wie Altenpflege, Such- und Rettungsaufgaben und allgemeine Assistenz im täglichen Leben übernehmen können. Derzeit sind die meisten Roboter darauf angewiesen von einem erfahrenen menschlichen Bediener programmiert zu werden. Damit Roboter Teil des täglichen Lebens werden können, müssen sie sich in verändernden Umgegebungen zurechtfinden und sich an neue Situationen anpassen. Daher werden selbstlernende Roboter zunehmend an Bedeutung gewinnen. Die Forschung zu autonom lernende Roboter ist bereits sehr aktiv und wird weiter an Bedeutung gewinnen. Es besteht eine enge Beziehung zwischen Reinforcement Learning und Robotik, wobei sich beide Forschungsgebiete gegenseitig ergänzen. Eine wichtige Herausforderung ist es dateneffiziente Methoden für selbstlernende Roboter zu entwickeln, da Stichproben aus realen Interaktionen sehr kostspielig sind. Policy-Suchmethoden, ein Teilgebiet des Reinforcement Learning, können eingesetzt werden um verschiedene Aufgaben durch Versuch und Irrtum zu erlernen.

In dieser Arbeit versuchen wir, die Dateneffizienz eines Policy-Suchalgorithmus zu verbessern. Dafür benutzen wir rekursive Schätzmethoden wie den Kalman Filter, der aus einer Bayes'schen Perspektive eingeführt wird. Wir implementieren verschiedene Versionen von Filteralgorithmen und vergleichen sie mit bisherigen Methoden und testen unsere Algorithmen für Optimierungs-Testfunktionen und einfache planare Greifaufgaben.

Abstract

Robots will increasingly permeate our daily life over the next few decades. Potentially fulfilling a wide range of tasks like elder care, search and rescue and general assistance in daily life. Today most robots rely on being taught and programmed by a skilled human operator. For robots to become part of daily life they need to cope with changing environments and regularly adjust to new situations. For mastering these challenges autonomously learning robots pose a promising solutions. Currently robot learning is already a very active field of research. There is a close relationship between Reinforcement learning and robotics, where both fields of research complement each other. A big challenge for robot learning is being sample efficiency, because real world samples are costly to obtain. Policy search methods, a sub-field of reinforcement learning, can be used to learn different task simply through trial and error.

In this thesis we try to improve the sample efficiency of a policy search algorithm. This is done using classical recursive estimation techniques like the Kalman filter, introduced from a Bayesian perspective. We implement different versions of filtering algorithms and compare them with previous methods and benchmark them on optimization test functions and simple planar reaching tasks.

Table of Contents

Chapter 1.

Introduction

1.1. Motivation

Robots are already used extensively in industry to form production chains, where they perform the same task over and over. These robots are being programmed and fine tuned by a human engineer which requires experience and expertise.
Recently there has been a new development of robots becoming part of our daily life's, for example in the form of vacuum cleaner and lawn mowers. In the future areas like care giving and everyday assistance and household work may become successful application domains of robotics (Schaal, 2007). This poses a dramatic shift from the prior uses of robots in industry, where they mainly worked in isolated and predefined contexts. Especially countries like Japan, facing the problem of an aging population, put increasing effort to make robots viable in these new application domains.
In daily life the robots are confronted with different challenges, like adapting to different lightening conditions and objects being moved around. The robots which are currently on the market like vacuum cleaners and lawn mowers have either a "one size fits all" approach or need a special setup in software or hardware. To solve this issue machine learning and especially reinforcement learning will be key technologies to enable robots to adjust to dynamic and stochastic environments.

Robotics and reinforcement learning complement each other with robotics providing a real world testing ground, and reinforcement learning providing the framework for formulating problems and finding solution. The relationship may be similar to the one of math and physics (Kober et al., 2013).
Compared to other domains, in which reinforcement learning has been successfully employed,

robotics poses a unique set of challenges. This makes it necessary to explore methods which are adjusted to the inherent requirements of robotics, which include:

- a high dimensional state and action space,
- problem of obtaining real world samples,
- problem of goal specification,
- and dealing with under-modeling and model uncertainty.

Robots have the potential to transform our society. By bringing robots from the factories into our homes there will be many possibilities for improvement. Nonetheless to reap the benefits of this development we will have to overcome many technical and ethical challenges alike.

1.2. Contribution

Policy search algorithms have shown promise as an alternative to value function-based reinforcement learning, especially for learning motor skills in robotics (Deisenroth et al., 2013). The MORE algorithm as a policy search method based on information theoretic updates is introduced in Abdolmaleki et al. (2015). The key idea of MORE is to learn a surrogate model of the objective function to efficiently compute the updates to the policy in closed form. One of the contributions of this thesis is to explore recursive estimation for learning the surrogate model of the objective function. Withe the goal of increasing the sample efficiency of the MORE algorithm. We focus on classical methods of parameter estimation and filtering like the Recursive Least Squares algorithm and the Kalman filter, considering more advanced methods is part of future work. We benchmark our version of the MORE algorithm on optimization test functions and simple planar reaching tasks and compare them to previous results.

1.3. Structure of Thesis

The remainder of this thesis is structured as follows:

Chapter 2: In the fundamentals chapter we lay the foundation for the thesis, first introducing the basics of Reinforcement Learning and then focusing on the specifics of applying RL to robotic tasks. Then we discuss policy search as one method for solving the robot learning problem. Next we introduce the original MORE algorithm and look at Bayesian filtering. Finally we discuss some data preprocessing techniques used.

Chapter 3: Review of the related work in the field.

Chapter 4: In this chapter we first motivate the idea of using recursive estimation for learning the surrogate model. Then we present our approach of connecting the MORE algorithm with recursive estimation techniques for the surrogate model.

Chapter 5: In the evaluation chapter we conduct experiments with our algorithms on several tests function and some simple reaching tasks for a planar robot arm. We compare our algorithm with original MORE algorithm and the least squares approach for learning the surrogate model.

Chapter 6: We conclude the thesis with a summary of the achieved results and an outlook on future work.

Chapter 2.

Fundamentals

This chapter introduces basic concepts used throughout this thesis. First we discuss the basics of reinforcement learning and the problem of robot learning. Next we give an overview of policy search, a sub-field of reinforcement learning, as one method to solve the robot learning problem. The Kullback-Leibler divergence (KL) is introduced as an important information theoretic measure of similarity metric between probability distributions (Kullback and Leibler, 1951). Having discussed the underlying basics we can then introduce the MORE algorithm (Abdolmaleki et al., 2015). Next we will look at filtering from a Bayesian Estimation viewpoint and review the Kalman filter for parameter estimation. Finally we look at some data preprocessing techniques employed.

2.1. Reinforcement Learning

Reinforcement Learning (RL) is a subfield of Machine Learning concerned with agents learning to interact with their environment and solving specific tasks. This is done through exploration and trial-and-error. The agents try to discover cause and effect relationships between their actions. Compared to supervised learning and unsupervised learning it more closely resembles the way we humans learn by interacting with our environment.
As Sutton and Barto (2018) puts it, the term reinforcement learning relates to a class of problems, solution methods and the field that studies these problems and solutions. Some success stories of RL include mastering the game of Go (Silver et al., 2016) and Atari games (Mnih et al., 2013). Generally, RL is applicable to a large range of problems. Whereas in supervised learning the best action is presented to the system, the agent in a reinforcement learning setting receives a reward (or punishment) for each action. To gain information

about the rewards the agent needs to explore previously unused actions. The decision of daring to try new things or to keep performing safe well-known actions is known as the *exploration-exploitation tradeoff*. In general the agent should exploit known actions that give decent reward, but he first has to try different things to learn about these actions, and then he has to progressively focus in on them. The reward signal is typically a single scalar value, hence the amount of information the agent receives is minimal compared to supervised and unsupervised learning approaches.

The classical approach to formalizing problems in RL is through Markov Decision Processes (MDPs). MDPs are a mathematical framework for decision making in deterministic and stochastic environments. MDPs focus on only three aspects - sensation, action and goal, which are central for reinforcement learning problems. MDPs satisfy the Markov property, which states that "the future is independent of the past given the present". In our case this means the next state s' and the reward only depend on the previous state s and action a (Sutton et al., 1992). An MDP can be formally defined as a tuple (S, A, P, r):

- a set of states $s \in S$ that describe the environment,
- a set of actions $a \in A$ that can be performed by the agent in the environment
- a transition function $P(s_{t+1} | s_t, a_t)$ that gives the probability of a new state s_{t+1} after an action a_t has been taken in state s_t,
- and a reward function $r(s_t, a_t)$ that specifies the immediate reward after taking action a_t in state s_t.

The Markov property can be expressed as

$$P(s_{t+1} | s_t, a_t, s_{t-1}, a_{t-1} \dots) = P(s_{t+1} | s_t, a_t).$$

This recapitulates the notion of state - a state is a sufficient statistic for predicting the future, rendering previous observations irrelevant. In robotics, we may only find some approximate notion of state.

We model the agent and its environment as a state $s \in S$. The agent may perform action $a \in A$ which can be either discrete or continuous. For every action step, the agent receives a reward R, which is a scalar value. The overarching goal is to find a mapping from states to actions, called policy π, that picks actions in a way that the reward is maximized. We can distinguish an *episodic* setting from an on-going task. In the episodic setting the task is restarted after the end of an episode and the goal is to maximize the total reward per episode. In on-going tasks the goal is to simply achieve high average reward over the whole life-time or one can use a formulation with a discounted return (weighting the future and past differently).

Generally, the goal is to find an optimal policy π^*, a mapping from states to actions that maximizes the expected return J. Optimal behavior can be modeled in different ways, resulting in different definitions for expected return:

- for a finite-horizon model with horizon H we get

$$J = E\left\{\sum_{t=0}^{H} R_t\right\},$$

- a discounted reward can be modeled with a discount factor γ (with $0 \leq \gamma < 1$), which yields

$$J = E\left\{\sum_{t=0}^{\infty} \gamma^t R_t\right\}.$$

Reinforcement learning can also be seen as general case of optimal control as in Sutton et al. (1992). While optimal control assumes perfect knowledge, RL uses approximations and data-driven techniques.

2.2. Robot Learning

The sub-field where reinforcement learning and machine learning intersect with robotics is called *Robot Learning*. It aims to bridge the gap between programmed robots, with fine tuned controllers and fully autonomous robots. As proposed in Deisenroth et al. (2013), robot control can be modeled as a reinforcement learning problem.
The state space $\mathbf{x}$ in robotic tasks is high dimensional and made up of the internal state of the robot (e.g., joint position, body position, camera images) and external state (e.g. object locations, lighting). The true state is not observable and also not noise free. The robot chooses its next motor control u according to a policy π. This policy π may be deterministic $u = \pi(s)$ or stochastic $u \sim \pi(s,a) = P(u|s)$. The motor command u alters the state according to the probabilistic transition function $p(\mathbf{x}_{t+1}|x_t, u_t)$. This transition function is not known, in model-based policy search this function is learned from data and used to improve the policy. Collectively the states and actions of the robot form a trajectory $\tau = (x_0, u_0, x_1, u_1, ...)$ which is also called a rollout or a path. There has to be a numeric scoring system assessing the quality of the robots trajectory and returning a reward signal $R(\tau)$. For episodic learning tasks the task ends after a given number T of time steps. Then the accumulated reward $R(\tau)$ for a trajectory is given by

$$R(\tau) = r_T(x_T) + \sum_{t=0}^{T-1} r_t(x_t, u_t)$$

where r_t is an instantaneous reward function (e.g. a punishment for energy consumed) and r_T the final reward, when performing a reaching task this may take the form of a quadratic punishment term for deviation from the goal posture.

If we consider an infinite-horizon for an on-going task we get

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r(x_t, u_t)$$

where $\gamma \in [0, 1)$ is a discount factor that discounts rewards further in the future.

Many tasks in robotics can thus be formulated as choosing an optimal control policy π^* that maximizes the expected accumulated reward

$$J_\pi = \mathbb{E}[R(\tau)|\,\pi] = \int R(\tau) p_\pi(\tau) d\tau$$

where $R(\tau)$ defines the objectives of the task, and $p_\pi(\tau)$ the distribution over trajectories τ.

Formulating robotic tasks in this way allows us to apply methods from reinforcement learning to them.

2.2.1. Challenges

Reinforcement learning is in general a difficult problem. One reason for this is that the reward signal may be given only occasionally and even then it may be unclear which of the agents actions were responsible for a certain reward signal.

Robotics is different compared to other fields where Reinforcement Learning is used. First of all the states and actions of the robots in the real world are inherently continuous requiring us to deal with the resolution into a discrete representation. In addition the state space can have a high dimensionality and working with real world systems on real hardware is costly and makes manual interventions necessary. Robots require algorithms to run in real-time and working with real sensors further introduces discrepancy between sensing and execution. Generally the state represented by sensors slightly lags behind the real state due to processing and communication delays. This is in stark contrast to most other reinforcement learning algorithms, which assume actions to take effect instantaneously.

Most traditional methods from RL like TD-learning (Sutton and Barto, 2018) have been unfit for these particular requirements of robotic tasks. Stressing the importance of robotics as a special testing ground for RL that demands new developments and innovative research. We will now discuss some problems encountered when applying RL to robotics. This treatment focuses only on a certain set of challenges and is not meant to be exhaustive.

Curse of Dimensionality

The term "Curse of Dimensionality" was coined by Bellman (1957) when he explored optimal control in higher dimensions and encountered an exponential explosion of the states and actions. For example, in our evaluation we run a simulation of a simple planar reaching task with a 5 link robot arm using Dynamic Movement Primitives (Ijspeert et al., 2002) for policy representation and get a 25 dimensional state. Especially modern anthropomorphic robots tend to have many degrees of freedom, which further increases the dimensionality.

The agent in RL generally needs to collect data throughout the entire state-space to ensure global optimization, for robotics where agents operate in three dimensional space this quickly becomes infeasible. To alleviate this problem a widespread idea is to use expert demonstration to get a good initialization for the agent's policy. This eliminates the need to explore the entire search space, instead the agent can focus on locally optimizing the initial policy. This concept of imitation learning focuses on the problem of "learning from demonstrations" which plays an important role for robotics (Osa et al., 2018). Imitation learning will enable domain experts to teach motions and skills without special knowledge about robotics, which will be crucial when robots start making their way from factories into everyday life.

Curse of Real-world Samples

Robot hardware is expensive and suffers from wear and tear, making costly maintenance necessary. Hence, safe methods for real robots should avoid big jumps in policy updates, as such sudden changes may result in unpredictable movements and consequently damage the robot. This constraint is commonly referred to as *safe exploration*. Whereas in traditional reinforcement learning safe exploration does not receive much attention, it has become a key issue for robot learning (Schneider, 1997).

When working with a real physical robot different external factors like temperature may change the robot's dynamics and additionally uncertainty from the sensors makes it difficult to reproduce and compare results.

Most tasks also require a "human in the loop" who either supervises the robot or resets the setup after one episode. Even if this step can be automated the data generation is very slow compared to other applications of RL which are based only on simulation. For a single robot the training time has a natural limit and in total only relatively few executions can be completed. One method for gathering more data is "collective robot learning" described in Kehoe et al. (2015). The idea is for multiple robots to share their data on trajectories, policies and outcomes. Currently this seems only viable for large corporations with significant capital. Stressing the importance of developing sample efficient algorithms. Thus, in robot learning the constraint of using only a small number of trials is given more weight than limiting memory consumption or computational complexity.

Curse of Goal Specification

Defining a good reward function in robot reinforcement learning is difficult and often needs a lot of domain knowledge and expertise. There are certain trade-offs to keep in mind, for example performing a powerful swing for a hitting task may yield high reward but may damage or shorten the life-time of the robot. Further reinforcement learning algorithms may solve tasks in unintended ways by exploiting the reward function in an unforeseen fashion (Ng et al., 1999). An alternative to specifying the reward function manually is Inverse reinforcement learning (Russell, 1998) The goal of inverse reinforcement learning is to recover the unknown reward function from the expert's trajectories.

The discussed challenges illustrate the difficulty of the optimal control problem in robotics. Generally, autonomous learning algorithms are rarely employed on robots for real daily usage. Also most algorithms are over fitted to a particular robot architecture and do not generalize to other robots easily. Even minor flaws or errors in the employed method can completely prevent success of learning. Nonetheless, appropriately chosen algorithms and rewards functions can already achieve promising results (Kober et al., 2013), but as a large number of different methods exist there is no clear general recipe for robot learning. Underlining the fact that robot learning still has a lot of open problems, and will continue to grow as a research field.

2.2.2. Policy Search

Many traditional methods in RL try to estimate the expected long-term reward of a policy for each state $\mathbf{x}$ and time step t, which leads to formulation of the value function $V_t^{\pi}(\mathbf{x})$. With the value function we can asses the quality of executing action $\mathbf{u}$ in state $\mathbf{x}$. This assessment is used to directly compute the policy by action selection or to update the policy π. As value function methods typically require to fill the complete action-space they struggle with the high dimensionality encountered in robotics. Thus policy search methods have become more common for applying RL to robotics. Policy search methods opposed to value-based methods use parameterized policies π_θ to search directly in the parameter space Θ of the policies (with $\theta \in \Theta$). This allows using RL with high-dimensional continuous action spaces encountered in robotics by reducing the search space of possible policies. Policy search further allows the usage of predefined task-appropriate policy representations like Dynamic Movement Primitives (Schaal et al., 2005), as well as easily integrating imitation learning for policy initialization.

Generally, we can divide policy search into model-free and model-based and differentiate whether stochastic or deterministic trajectories are used. Model-free policy search uses trajectories from the robot directly for updating the policy. Model-based methods use the data from the robot to first learn a model of the robot. This model is then used to generate trajectories that are used for policy updates. Due to their simplicity and by avoiding the need

of learning a model, which further introduces the problem of under-modeling, model-free methods have been employed to a wide range of problems (Deisenroth et al., 2013).

The most important concept in policy search is computing the policy updates. Both model-free and model-based policy search methods use policy gradients (PG) which employ gradient ascent for maximizing the expected return. The REINFORCE algorithm introduced in Williams (1992) is an example for a model-free method using policy gradients. The PILCO (probabilistic inference for learning control) policy search framework (Deisenroth and Rasmussen, 2011) is an example for a model-based approach. Another way to compute the policy updates is using the Expectation-Maximization algorithm (Bishop, 2006), by formulating policy search as an inference problem. One model-free method for this category is the Policy learning by Weighting Exploration with the Returns (PoWER) algorithm (Kober and Peters, 2011). The third category of policy updates we look at are information-theoretic (Inf. Th.) approaches based on the Kullback-Leibler Divergence (section 2.3). The relative entropy policy search algorithm (REPS) (Peters et al., 2010) and the MORE algorithm (section 2.4) fit into this category. In figure 2.1 we give a graphical overview of the different policy search methods.

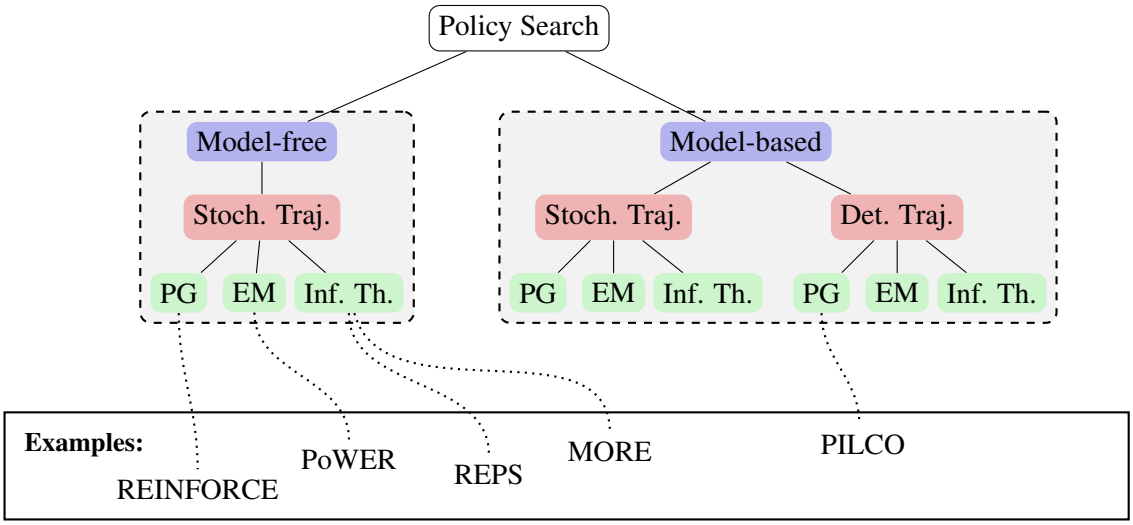


Figure 2.1.: Categorization of policy search into model-free policy search and model-based policy search. In stochastic trajectory (Stoch. Traj.) generation the trajectories are directly sampled from the robot or in the model-based case from the learned model of the robot. Some Model-based methods also use deterministic trajectory (Det. Traj.) prediction and analytically predict the trajectory distribution instead of sampling from the system. The drawing is based on Deisenroth et al. (2013).

In this thesis, we will focus on model free policy search methods where the trajectories are generated by "sampling" from the robot and information theoretic policy updates are used. More specifically we will formulate our approach in the setting of stochastic search algorithms, which are general black-box optimizers. They are used in a wide range of fields like operations

research, machine learning and also policy search. Since these algorithms do not use any knowledge about the objective function it is straightforward to apply them to policy search in the episode-based formulation.

Using stochastic search algorithms we keep an upper-level policy $\pi_\omega(\theta)$ which selects the parameters of the actual control policy $\pi_\theta(\mathbf{u}|\mathbf{x})$ of the robot. Instead of directly finding the parameters θ of the lower-level policy we want to find the parameter vector ω which defines a search distribution over θ. We can then use this search distribution to directly explore the parameter space.

2.3. Kullback-Leibler (KL) Divergence

Several algorithms that can be used for model-free policy search like NES (Wierstra et al., 2014) and REPS (Peters et al., 2010) rely on the Kullback-Leibler divergence, also known as the relative entropy, for controlling the difference between the old and updated policy. Working with real robots additionally requires to perform safe exploration. Big exploration steps may result in damaging the hardware. Specifically, it measures the Shannon entropy of one distribution relative to the other. The KL divergence from q to p is defined as

$$\mathrm{KL}(p||q) = \int p(\theta) \log\left(\frac{p(\theta)}{q(\theta)}\right) d\theta$$

where p and q are continuous probability distributions. Note, that in general the relative entropy is not symmetric under interchange of the distributions p and q.

2.4. MORE Algorithm

Model-Based Relative Entropy Stochastic Search (MORE) (Abdolmaleki et al., 2015) is a stochastic search algorithm that can be used as a policy search method for episodic reinforcement learning tasks. The key idea is using information-theoretic policy updates by bounding the relative entropy (Kullback Leibler divergence) between two subsequent policies. As MORE uses no gradient information and requires only function evaluations of the objective function it can be seen as a type of stochastic search algorithm and can be used for black box optimization problems. The essential difference of MORE compared to previous algorithms using the KL-bound like REPS (Peters et al., 2010) lies in utilizing a quadratic surrogate model of the objective function to satisfy the KL-bound in closed form without approximations. On top of that it introduces a lower bound constraint on the entropy of the new distribution to avoid premature convergence.

The MORE algorithm can be used to maximize an objective function $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$. The goal is to find one or more parameter vectors $\mathbf{x} \in \mathbb{R}^n$ with the highest possible objective

value. For this the algorithm maintains a search distribution over $\mathbf{x}$ for which the expectation over the reward is maximized. To accomplish this, the MORE algorithm iteratively draws samples from the search distribution, implemented as a multivariate Gaussian distribution, i.e. $\pi(\mathbf{x}) = \mathbf{N}(\mathbf{x}|\,\mu,\Sigma)$. This distribution corresponds to an upper-level policy and is parameterized by the mean and covariance matrix. In each iteration N samples are drawn from the search distribution. Each sample $\mathbf{x}$ is then evaluated on the objective function yielding the corresponding objective value r, this constitutes the data $\{\mathbf{x}^k, r^k\}_{k=1\ldots N}$ which is used to compute a new search distribution. This iterative process is run until the algorithm converges.

2.4.1. MORE Framework

To satisfy the bound on the relative entropy between subsequent search distributions and the bound on the entropy while optimizing the objective function we use the theory of constraint optimization (Boyd et al., 2004). In constraint optimization we want to maximize a function $f(x)$ under a set of equality constraints and inequality constraints.

$$\begin{aligned} \text{maximize} \quad & f(x) \\ \text{subject to} \quad & g_i(x) = 0, \quad i = 1,\ldots,m \\ & c_j(x) \leq 0, \quad j = 1,\ldots,p \end{aligned} \tag{2.1}$$

With this we can now formulate the constraint optimization problem to obtain a new search distribution that maximizes the expected objective value while upper-bounding the KL-divergence and lower-bounding the entropy of the search distribution at the same time

$$\begin{aligned} \max_{\pi} \quad & \int \pi(\mathbf{x}) f(\mathbf{x}) d\mathbf{x}, \\ \text{subject to} \quad & \text{KL}\left(\pi(\mathbf{x})||q(\mathbf{x})\right) \leq \varepsilon, \\ & H(\pi) \geq \beta, \\ & 1 = \int \pi(\mathbf{x}) d\mathbf{x}, \end{aligned}$$

where $H(\pi) = -\int \pi(\mathbf{x}) \log \pi(\mathbf{x}) d\mathbf{x}$ denotes the entropy of the search distribution. The parameters ε and β are hyperparameters to control the exploration-exploitation trade-off. The ε can be chosen freely to control the step size of the policy update. Generally its size will depend on the specific problem and the amount of available samples. The bound β is defined such that the relative difference between the entropy of the policy $H(\pi)$ and a minimum exploration policy $H(\pi_0)$ is decreased for a certain percentage:

$$H(\pi) - H(\pi_0) \geq \gamma(H(q) - H(\pi_0)) \rightarrow \beta = \gamma(H(q) - H(\pi_0) + H(\pi_0)$$

We obtain the solution to the constraint optimization problem using the theory of Lagrange multipliers and duality (for details see appendix A.1).

With assuming π being Gaussian solving the dual problems yields

$$\begin{aligned} \pi_{t+1} &= \mathrm{N}(\mu_{t+1}, \Sigma_{t+1}) \\ \mu_{t+1} &= (\eta\Sigma_t^{-1}\mu_t + \mathbf{r})/(\eta+\omega) \\ \Sigma_{t+1} &= (\eta\Sigma_t^{-1} + \mathbf{R})/(\eta+\omega), \end{aligned} \tag{2.2}$$

where η and ω are the lagragian multipliers obtained by minimizing the corresponding dual function. In addition the linear term $\mathbf{r} \in \mathbb{R}^n$ and the quadratic term $\mathbf{R} \in \mathbb{R}^{n\times n}$ of the current surrogate model are used to update the policy. With these equations we can iteratively update the search distribution.

2.4.2. Surrogate Model

The key idea of the MORE algorithm is to use a surrogate model of the objective function for satisfying the bound on the KL-divergence. A quadratic model is sufficient, since the exponent of a Gaussian distribution is also quadratic and thus it would not be possible to exploit the additional information of a more complex surrogate model. The surrogate model has a quadratic amount of parameters, estimating these parameters poses a bottle-neck in terms of sample efficiency for the algorithm. The original approach of MORE employs a Bayesian dimensionality reduction method and linear regression. We discuss our approach for estimating the surrogate model in section 4.1.

2.5. Bayesian Filtering

Optimal filtering is concerned with estimating the state of a time-varying system which is indirectly observed through noisy measurements. This section will focus on optimal filtering from a Bayesian perspective and is largely based on Särkkä (2013).

The term “Bayesian” refers to inference methods that represent “degrees of certainty” using probability theory. They are fundamentally based on applying Bayes’ rule to update the degree of certainty given data. More generally as Gelman et al. (2013) puts its, Bayesian inference is the process of fitting a probability model to a set of data and summarizing the result by a probability distribution on the parameters of the model and on unobserved quantities such as predictions for new observations.

Filtering methods are widely used in robotics to deal with noisy sensor measurements. This includes tasks like object tracking, robot control and robot localization (Chen, 2011). Since robots need to make decisions based on relatively small amounts of data, it is common to adopt a Bayesian perspective when using filtering methods and for reasoning about the environment in general (Thrun, 2002).

2.5.1. Bayesian Parameter Estimation

In general, when using Bayesian models for estimating *unknown parameters* θ, the following probability distributions are used:

- **Prior Distribution:** Encodes the information on parameter θ before seeing any observations. When we are uncertain about our prior information we can choose a high variance of the distribution or use a non-informative prior (which imposes the minimal amount of structure on the data).

$$p(\theta) = \text{information on parameter } \theta \text{ before seeing any observations}$$

- **Measurement Model:** Models the relationship between true parameters and the measurements.

$$p(y|\,\theta) = \text{distribution of observation } y \text{ given the parameters } \theta$$

- **Posterior Distribution:** The conditional distribution of the parameters given the observations. It represents the updated belief about the parameters after obtaining the measurements. It can be computed by using Bayes' rule.

$$p(\theta|y) = \frac{p(y|\,\theta)p(\theta)}{p(y)} \propto p(y|\,\theta)p(\theta)$$

2.5.2. Optimal Filtering as Bayesian Inference

The goal of optimal filtering can be seen as solving a statistical inversion problem where the unknown quantity is a potentially vector valued time series $\{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, ...\}$ which is observed through a set of noisy measurements $\{\mathbf{y}_1, \mathbf{y}_2, ...\}$, as depicted in Figure 2.2.

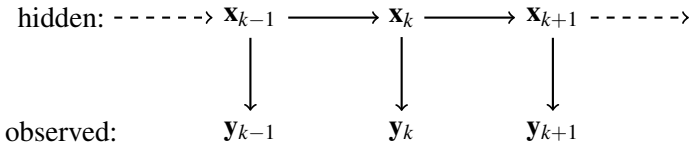


Figure 2.2.: A sequence of hidden states $\mathbf{x}_k$ is indirectly observed through noisy measurements. Drawing recreated from (Särkkä, 2013).

We want to estimate the hidden states from the observed measurements. Solving this problem in a Bayesian sense means we need to compute the joint posterior distribution of all states given all the measurements, for this we can use Bayes' rule. This yields a batch solution to

the statistical estimation problem resulting in the equation

$$p(\mathbf{x}_{0:T}|y_{1:T}) = \frac{p(y_{1:T}|\mathbf{x}_{0:T})p(\mathbf{x}_{0:T})}{p(y_{1:T})} \tag{2.3}$$

where $p(\mathbf{x}_{0:T})$ is the prior distribution defined by the dynamic model, $p(y_{1:T}|\mathbf{x}_{0:T})$ is the likelihood model for the measurements and $p(y_{1:T})$ is the normalization constant defined as

$$p(y_{1:T}) = \int p(y_{1:T}|\mathbf{x}_{0:T})p(\mathbf{x}_{0:T})d\mathbf{x}_{0:T}.$$

This formulation is unfit for dynamic estimation tasks where we receive measurements one at a time, since at each time step we would have to recompute the full posterior distribution. As the number of time steps increases, also the dimensionality of the full posterior would increase making computations intractable. If we instead only compute selected marginal distributions we can make computation feasible again. For this we need to restrict our dynamic models to probabilistic Markov sequences, with a transition probability distribution $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ that depends only on the previous state.

In *Bayesian filtering and smoothing* the following marginal distributions are of interest

- **Filtering distributions** computed by the Bayesian filter are the marginal distributions of the current state $\mathbf{x}_k$ given the current and previous measurements $y_{1:k} = \{y_1, ..., y_k\}$

$$p(\mathbf{x}_k|y_{1:k}), \quad k = 1, ..., T.$$

- **Prediction distributions** which can be computed with the prediction step of the Bayesian filter are the marginal distributions of the future state $\mathbf{x}_{k+n}$, with n steps after the current time step

$$p(\mathbf{x}_{k+n}|y_{1:k}), \quad k = 1, ..., T \quad n = 1, 2, ...$$

- **Smoothing distributions** computed by the Bayesian smoother are the marginal distributions of the state $\mathbf{x}_k$ given a certain interval $y_{1:T} = \{y_1, ..., y_T\}$ of measurements with $T > k$

$$p(\mathbf{x}_k|y_{1:T}), \quad k = 1, ..., T$$

We will focus on the filtering distribution, as we use it for our surrogate model estimation problem.

For the filtering distribution we can formulate the recursive Bayesian solution to the statistical inversion problem as follows:

1. The distribution of measurements is modeled by the likelihood function $p(y_k|\mathbf{x}_k)$ and the measurements are assumed to be conditionally independent

2. In the beginning at time step zero all information about $\mathbf{x}$ is contained in the prior distribution $p(\mathbf{x}_0)$

3. The measurements are assumed to be obtained one at a time, first y_1 then y_2 and so on. The key idea is to use the posterior distribution from the previous time step as the current prior distribution

$$\begin{aligned}
p(\mathbf{x}_1|y_1) &= \frac{1}{Z_1} p(y_1|\mathbf{x}_1) p(\mathbf{x}_0), \\
p(\mathbf{x}_2|y_{1:2}) &= \frac{1}{Z_2} p(y_2|\mathbf{x}_2) p(\mathbf{x}_1|y_1), \\
&\vdots \\
p(\mathbf{x}_k|y_{1:k}) &= \frac{1}{Z_k} p(y_k|\mathbf{x}_k) p(\mathbf{x}_k|y_{1:k-1}), \\
&\vdots \\
p(\mathbf{x}_T|y_{1:T}) &= \frac{1}{Z_T} p(y_T|\mathbf{x}_T) p(\mathbf{x}_{T-1}|y_{1:T-1}),
\end{aligned}$$

   with the normalization term $Z_k = \int p(y_k|\mathbf{x}_k) p(\mathbf{x}_k|y_{1:k-1}) d\mathbf{x}_k$.

This recursive formulation of Bayesian estimation has several useful properties. First of all it can be considered as the *online learning* solution to the Bayesian learning problem. As each step in the recursive estimation is a full Bayesian update step, batch Bayesian inference is a special case of recursive Bayesian inference. Due to the sequential nature of the estimation we can model what happens to the unknown quantity $\mathbf{x}$. This turns out to be the basis of filtering theory, where time behavior is modeled by assuming the quantity to be a time-dependent stochastic process $\mathbf{x}(t)$.

2.5.3. Least Squares

To ease the application of Bayesian filtering to our specific problem at a later stage we will now discuss a simple regression problem. First we are going to derive the least squares solution, as we use a least squares (LS) approach for comparison in our experiments.

We consider a simple linear regression problem,

$$y_k = x_1 + x_2 t_k + \varepsilon_k \tag{2.4}$$

where we assume that the measurement noise is zero mean Gaussian with a given variance $\varepsilon_k \sim \mathcal{N}(0, \sigma^2)$ and that the prior distribution of the parameters $\mathbf{x} = (x_1 \; x_2)^T$ is Gaussian with known mean and covariance $\mathbf{x} \sim \mathcal{N}(\mathbf{m}_0, \mathbf{P}_0)$. Note that in this section the parameters $\mathbf{x}$ are assumed to stay constant. We now want to estimate the parameters $\mathbf{x}$ from a set of measurement data $\mathcal{D} = \{(t_1, y_1), ..., (t_T, y_T)\}$.

In compact probabilistic notation the linear regression model can be written as

$$\begin{aligned} p(y_k|\mathbf{x}) &= \mathcal{N}(y_k|\mathbf{H}_k\mathbf{x},\sigma^2) \\ p(\mathbf{x}) &= \mathcal{N}(\mathbf{x}|\mathbf{m}_0,\mathbf{P}_0) \end{aligned} \tag{2.5}$$

Here $\mathbf{H}_k = (1\ t_k)$ is the design matrix and contains the regressors and $\mathcal{N}(\cdot)$ denotes the Gaussian probability density function (see Appendix A.2). The row vector $\mathbf{H}_k$ is denoted in matrix notation, to avoid using different notation for scalar and vector measurements. The batch solution can then be easily obtained by application of Bayes'rule

$$\begin{aligned} p(\mathbf{x}|y_{1:T}) &\propto p(\mathbf{x})\prod_{k=1}^{T} p(y_k|\mathbf{x}) \\ &= \mathcal{N}(\mathbf{x}|\mathbf{m}_0,\mathbf{P}_0)\prod_{k=1}^{T}\mathcal{N}(y_k|\mathbf{H}_k\mathbf{x},\sigma^2). \end{aligned}$$

Because the prior and likelihood are Gaussian, the posterior distribution in turn is also Gaussian and denoted by

$$p(\mathbf{x}|y_{1:T}) = \mathcal{N}(\mathbf{x}|\mathbf{m}_t,\mathbf{P}_t).$$

By completing the quadratic form in the exponent we get the equations (2.6) for the mean and covariance of the posterior distribution.

$$\begin{aligned} \mathbf{m}_T &= \left[\mathbf{P}_0^{-1} + \frac{1}{\sigma^2}\mathbf{H}^T\mathbf{H}\right]^{-1}\left[\frac{1}{\sigma^2}\mathbf{H}^T\mathbf{y} + \mathbf{P}_0^{-1}\mathbf{m}_0\right] \\ \mathbf{P}_T &= \left[\mathbf{P}_0^{-1} + \frac{1}{\sigma^2}\mathbf{H}^T\mathbf{H}\right]^{-1} \end{aligned} \tag{2.6}$$

where $\mathbf{H}_k = (1\ t_k)$ and

$$\mathbf{H} = \begin{pmatrix}\mathbf{H}_1\\ \vdots \\ \mathbf{H}_T\end{pmatrix} = \begin{pmatrix}1 & t_1\\ \vdots & \vdots \\ 1 & t_t\end{pmatrix}, \qquad y = \begin{pmatrix}y_1\\ \vdots \\ y_T\end{pmatrix}$$

2.5.4. Kalman Filter

For the least squares solution, the parameters $\mathbf{x} = (x_1\ x_2)$ of the regression model (2.4) are assumed to stay constant. Now, we assume the parameters are allowed to perform a Gaussian

random walk between measurements modeled by

$$\begin{aligned} p(y_k|\,\mathbf{x}_k) &= \mathcal{N}(y_k|\,\mathbf{H}_k\mathbf{x}_k, \sigma^2) \\ p(\mathbf{x}_k|\,\mathbf{x}_{k-1}) &= \mathcal{N}(\mathbf{x}_k|\,\mathbf{x}_{k-1}, \mathbf{Q}) \\ p(\mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_0|\,\mathbf{m}_0, \mathbf{P}_0) \end{aligned}$$

where $\mathbf{Q}$ is the covariance of the random walk.

We will now formulate the linear regression model as a time-invariant model by avoiding explicit covariates t_k. This has the advantage that the model is not dependent on the absolute time, but only on the relative positions of states and measurements in time. We denote the time difference between consecutive times as $\Delta t_{k-1} = t_k - t_{k-1}$. The idea is that if the underlying phenomenon (signal, state, parameter) x_k was exactly linear, the difference between adjacent time points could be written exactly as

$$x_k - x_{k-1} = \dot{x}\,\Delta t_{k-1}$$

where $\dot{x}$ is the derivative, which is constant in the linear case. As this may not exactly be the case we also add some noise to the model to get

$$\begin{aligned} x_{1,k} &= x_{1,k-1} + \Delta t_{k-1} x_{2,k-1} + q_{1,k-1} \\ x_{2,k} &= x_{2,k-1} + q_{2,k-1} \\ y_k &= x_{1,k} + s_k \end{aligned}$$

where the signal is the first component of the state ($x_{1,k} = x_k$) and the derivative is the second ($x_{2,k} = \dot{x}_k$). The noises are $s_k \sim \mathcal{N}(0, \sigma^2)$ and $(q_{1,k-1}, q_{2,k-1}) \sim \mathcal{N}(0, \mathbf{Q})$. Now, the linear regression model (2.5) can be written in the form

$$\begin{aligned} p(y_k|\,\mathbf{x}_k) &= \mathcal{N}(y_k|\,\mathbf{H}\,\mathbf{x}_k, \sigma^2) \\ p(\mathbf{x}_k|\,\mathbf{x}_{k-1}) &= \mathcal{N}(\mathbf{x}_k|\,\mathbf{A}_{k-1}\mathbf{x}_{k-1}, \mathbf{Q}) \end{aligned}$$

where

$$\mathbf{A}_{k-1} = \begin{pmatrix} 1 & \Delta t_{k-1} \\ 0 & 1 \end{pmatrix}, \qquad \mathbf{H} = (1 \quad 0).$$

In this formulation the model is a special case of generic linear Gaussian models of the form,

$$\begin{aligned} p(y_k|\,\mathbf{x}_k) &= \mathcal{N}(y_k|\,\mathbf{H}_k\mathbf{x}_k, \mathbf{R}_k) \\ p(\mathbf{x}_k|\,\mathbf{x}_{k-1}) &= \mathcal{N}(\mathbf{x}_k|\,\mathbf{A}_{k-1}\mathbf{x}_{k-1}, \mathbf{Q}_{k-1}) \end{aligned}$$

for which the Kalman Filter (Kalman, 1960) is the optimal recursive solution (for the full derivation see appendix A.3).

The Kalman filter equations can be expressed as prediction and update steps as follows:

- The prediction step

$$\begin{aligned} \mathbf{m}_k^- &= \mathbf{A}_{k-1}\mathbf{m}_{k-1} \\ \mathbf{P}_k^- &= \mathbf{A}_{k-1}\mathbf{P}_{k-1}\mathbf{A}_{k-1}^T + \mathbf{Q}_{k-1} \end{aligned} \tag{2.7}$$

- The update step

$$\begin{aligned} v_k &= y_k - \mathbf{H}_k\mathbf{m}_k^- \\ \mathbf{S}_k &= \mathbf{H}_k\mathbf{P}_k^T\mathbf{H}_k^T + \mathbf{R}_k \\ \mathbf{K}_k &= \mathbf{P}_k^-\mathbf{H}_k^T\mathbf{S}_k^{-1} \\ \mathbf{m}_k &= \mathbf{m}_k^- + \mathbf{K}_k v_k \\ \mathbf{P}_k &= \mathbf{P}_k^- - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^T \end{aligned} \tag{2.8}$$

2.6. Data Preprocessing Techniques

We now examine several data preprocessing techniques. Some key challenges that arise in our data are high range of objective values, and dealing with sharp bumps in reward from penalties (e.g collisions).

2.6.1. Whitening

Whitening is a common data preprocessing method in statistical analysis to transform a correlated random vector into an uncorrelated one (Kessy et al., 2018). We employ whitening to our algorithm, to reduce the general complexity of the parameters to be estimated.

Whitening is a linear transformation that converts a d-dimensional random vector $\mathbf{x} = (x_1, ..., x_d)^T$ with mean $\mathrm{E}(\mathbf{x}) = \mu = (\mu_1, ..., \mu_d)^T$ and positive definite $d \times d$ covariance matrix $\mathrm{var}(\mathbf{x}) = \Sigma$ into a new random vector

$$\mathbf{z} = (z_1, ..., z_d)^T = \mathbf{W}\mathbf{x} \tag{2.9}$$

of the same dimension d and with unit diagonal "white" covariance $\mathrm{var}(\mathbf{z}) = \mathbf{I}$. The square $d \times d$ matrix $\mathbf{W}$ is called the whitening matrix.

The whitening transformation defined in Equation Equation (2.9) requires the choice of a suitable whitening matrix W. Since $\mathrm{var}(z) = \mathbf{I}$ it follows that $\mathbf{W}\Sigma\mathbf{W}^T = \mathbf{I}$ and thus $\mathbf{W}(\Sigma\mathbf{W}^T\mathbf{W}) = \mathbf{W}$, which is fulfilled if $\mathbf{W}$ satisfies the condition

$$\mathbf{W}^T\mathbf{W} = \Sigma^{-1}$$

This constrain does not uniquely determine the whitening matrix $\mathbf{W}$, instead given Σ there are infinitely many possible matrices $\mathbf{W}$, because it allows for rotational freedom.

Chapter 3.

Related Work

The MORE algorithm can be used as a model-free policy search algorithm with an information theoretic approach for updating the policy. Information-theoretic policy algorithms use the Kullback-Leibler Divergence in a constrained optimization problem to bound the distance of the old policy to the new one. Whereas MORE uses a surrogate model to compute the new search distribution in closed form other methods like the relative entropy policy search algorithm (REPS) proposed in Peters et al. (2010) uses a sample-based approximation for the KL-divergence. While in Peters et al. (2010) a step-based policy search algorithm is proposed the episode-based version of REPS is presented in Kupcsik et al. (2013) which is equivalent to stochastic search. Using Taylor approximations of the KL-divergence leads to the natural evolutionary strategies (NES) presented in Wierstra et al. (2014). NES uses the concept of the natural gradient, where the update is performed according to the standard gradient while simultaneously satisfying the KL-bound between subsequent search distributions.

The Covariance Matrix Adaptation-Evolutionary Strategy (CMA-ES) (Hansen, 2006) is a widely used stochastic search algorithm. It is based on well-defined heuristics to update the search distribution. In our experiments we try to beat the benchmark for sample efficiency set by this algorithm.

A contextual version of MORE has been explored in Tangkaratt et al. (2017), where variables that do not change for a given task but vary from one task to another give the notion of context. Enabling learning from high-dimensional context variables like camera images.

Model-based methods for approximating the objective function with a local surrogate have been used in derivative free optimization Nocedal and Wright (2006). Also, local surrogate models have been used in trust region methods like Powell (2009). These methods maintain a

point estimate and a trust region around this point instead of a search distribution. The point estimate is updated by optimizing the surrogate and staying in the trust region.

Recursive estimation with methods like the Kalman Filter is extensively used in robotics (Chen, 2011), mainly to estimate the state and environment of the robot from noisy sensor measurements.

Chapter 4.

Recursive Surrogate-Modeling for MORE

In this chapter we will first motivate the idea of recursively estimating the surrogate model for the MORE algorithm. We formulate the surrogate model estimation task as a regression problem and present our solution to it. Finally we connect our approach with the MORE algorithm.

4.1. Surrogate Model Estimation

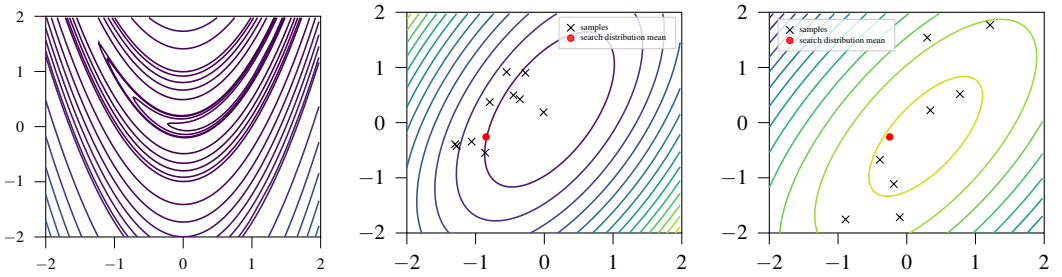
4.1.1. Motivation

Previously the surrogate model has been estimated from scratch in each iteration using samples and corresponding values of the objective function. However, subsequent models are correlated due to the locality of the data, see figure 4.1 for an example. This arises from the fact that the KL-divergence is used to bound the distance between subsequent search distributions. Therefore recursive estimation techniques may be able to utilize the information contained in the previous surrogate model and thus reduce the total amount of samples needed. Additionally this may also reduce the runtime of the algorithm.

The surrogate model has the following form

$$f(\mathbf{x}) \approx \hat{f}(\mathbf{x}) = \mathbf{x}^T \mathbf{R}\mathbf{x} + \mathbf{x}^T \mathbf{r} + r.$$

where $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ is the original objective function. The surrogate model $\hat{f}(\mathbf{x})$ has a quadratic term $\mathbf{R} \in \mathbb{R}^{n \times n}$, a linear term $\mathbf{r} \in \mathbb{R}^n$ and a scalar r. Using a quadratic model is



(a) objective function (b) surrogate model (iteration i) (c) surrogate model (iteration $i+1$)

Figure 4.1.: In (a) the original objective function, a 2 dimensional Rosenbrock function (section 5.2.1) is shown. In (b) and (c) we see the search distribution mean, the surrogate model and the samples used to estimate the model from two subsequent MORE iterations.

sufficient as the exponential of the Gaussian is also quadratic in the parameters. A more complex model could not be exploited by a Gaussian distribution (Abdolmaleki et al., 2015). Nevertheless estimating the surrogate model is a challenging task as we have to estimate $\mathcal{O}(n^2)$ many parameters while using a minimal amount of samples in each MORE Iteration.

4.1.2. Regression Problem

We can formulate the task of estimating the surrogate model as a regression problem (4.1). For this we use a feature function $\phi(\mathbf{x})$ which returns a bias term, all linear and all quadratic terms. The dimensionality of $\phi(\mathbf{x})$ is $D = 1 + d + d(d+1)/2$, where d is the dimensionality of the parameter space.

$$y = \phi(\mathbf{x})\beta + \varepsilon \tag{4.1}$$

Our data consists of the samples and corresponding objective values $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$. To solve the regression problem we set up the design matrix $\mathbf{X}$ as depicted in equation (4.2). One row of $\mathbf{X}$ contains a one as the first entry for the bias term. Then the next d entries are made up of the corresponding sample $\mathbf{x}_k$ for the linear term. The final $d(d+1)/2$ entries are the lower triangular matrix of the product $\mathbf{x}_k\mathbf{x}_k^T$, which we denote by $\text{tril}(\mathbf{x}_k\mathbf{x}_k^T)$.

For the least squares (LS) approach with $n \geq D$ samples we get the following system of linear equations

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & \mathbf{x}_1 & \text{tril}(\mathbf{x}_1\mathbf{x}_1^T) \\ 1 & \mathbf{x}_2 & \text{tril}(\mathbf{x}_2\mathbf{x}_2^T) \\ \vdots & \vdots & \vdots \\ 1 & \mathbf{x}_n & \text{tril}(\mathbf{x}_n\mathbf{x}_n^T) \end{bmatrix}}_{\mathbf{X}} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_D \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}. \tag{4.2}$$

We assume that the measurement noise ε_k is zero mean Gaussian $\varepsilon_k \sim N(0, \sigma^2)$ distributed. Equation (4.2) can be solved with ordinary least squares $\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}\mathbf{y}$. For our experiments we use a form of ridge regression (Hoerl et al., 1975) for comparison to our recursive estimation approach.

The solution vector β contains the surrogate model parameters $(r, \mathbf{r}, \mathbf{R})$.

Using recursive estimation we can process each pair of samples and rewards $(\mathbf{x}_k, y_k)$ one at a time.

$$y_k = \begin{pmatrix} 1 & \mathbf{x}_k & \text{tril}(\mathbf{x}_k\mathbf{x}_k^T) \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_D \end{pmatrix} + \varepsilon_k$$

4.2. Recursive Least Squares

As we did not find a good state transition model we only use the update step of the Kalman filter equation (2.8). We tried some momentum based approaches during our evaluation but did not receive any promising results. Therefore our approach is a special case of the Kalman filter equations with an Identity matrix as the state transition matrix $\mathbf{A}$. We call our approach recursive least squares with drift model for the parameters (Algorithm 1).

Let us now explain how we use the recursive least squares (RLS) algorithm to compute the solution vector β to the regression problem. As the model parameters do not stay constant we assume they perform a Gaussian random walk between measurements and for an arbitrary step k we get

$$\begin{aligned} p(y_k|\,\beta_k) &= \mathcal{N}(y_k|\,\mathbf{H}_k\beta_k, \sigma^2) \\ p(\beta_k|\,\beta_{k-1}) &= \mathcal{N}(\beta_k|\,\beta_{k-1}, \mathbf{Q}) \\ p(\beta_0) &= \mathcal{N}(\beta_0|\,\mathbf{m}_0, \mathbf{P}_0) \end{aligned}$$

where $\mathbf{Q}$ is the covariance of the random walk. The parameter noise is added to the covariance matrix of the parameters before each update step. Our prior $\mathbf{m}_0$ is chosen such that for the first surrogate model parameters we set the quadratic term to an identity matrix $\mathbf{R} = \mathbf{I}$ and the other terms to zero $\mathbf{r} = \mathbf{0}$ and $r = 0$. The covariance matrix is initialized as $\mathbf{P}_0 = \delta\mathbf{I}$ with the parameter δ controlling how confident we are in our prior. We want to compute the filtering distribution

$$\begin{aligned} p(\beta_k|\,y_{1:k}, \mathbf{x}_{1:k}) &\propto p(y_k, \mathbf{x}_k|\,\beta_k)\, p(\beta_k|\,y_{1:k-1}, \mathbf{x}_{1:k-1}) \\ &\propto \mathcal{N}(\beta_k|\,\mathbf{m}_k, \mathbf{P}_k), \end{aligned}$$

for this we can use the update equation of the Kalman filter (2.8). Increasing the covariance matrix of the parameters by the model noise combined with the Kalman filter update step gives us algorithm 1, which computes the model parameters in a recursive fashion.

Algorithm 1 : Recursive Least Squares with Drift Model

Input : stream of samples as rows of design matrix $\mathbf{X}_n$ and rewards y_n with $n = 1,\dots,N$, $\mathbf{Q}$ model noise, σ^2 measurement noise
Initialization: $\mathbf{m}_0 = (0,\mathbf{0},\mathbf{I})$, $\mathbf{P}_0 = \delta\mathbf{I}$
for $n = 1,...,N$ **do**
  begin Adding the model noise
    $\mathbf{P}_n^- = \mathbf{P}_{n-1} + \mathbf{Q}$
  end

  begin Update step
    $S_n = \mathbf{X}_n \mathbf{P}_n^- \mathbf{X}_n^T + \sigma^2$
    $\mathbf{K}_n = \mathbf{P}_n^- \mathbf{X}_n^T S_n^{-1}$
    $\mathbf{m}_n = \mathbf{m}_{n-1} + \mathbf{K}_n[y_n - \mathbf{X}_n\mathbf{m}_{n-1}]$
    $\mathbf{P}_n = \mathbf{P}_n^- - \mathbf{K}_n S_n \mathbf{K}_n^T$
  end
end

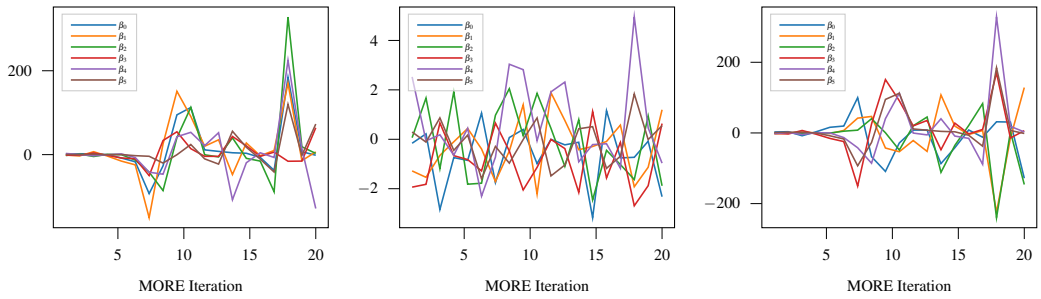
4.2.1. Sample pool

The original MORE algorithm uses a sample pool, which contains samples from previous iterations, for estimating the surrogate model. In theory when using a recursive estimation approach the information in a sample pool is redundant.

In our experiments we did not receive good results when using only new samples without a sample pool, since the predictions for the surrogate models are to inaccurate. The first adjustment is to use a warm start, meaning we get a accurate prediction in the first iteration by processing a large batch of samples. This prediction can then be subsequently updated.

On optimization test functions like the Rosenbrock function **??** we also did not receive promising results using a warm start. Thus we also tried using a sample pool for our recursive estimation approach. From a theoretical standpoint this is rather imprecise, this improved our results regardless. When using a sample pool we generally process the newest samples last. Additionally we explored giving older samples a greater model noise and by this encoding our uncertainty about them. For this we simply introduce a counter for each sample, indicating how many times the sample has been used. For older samples (higher counter) we increased the model noise controlled by a weight factor γ. When using this sample weighting we increase the model noise in the RLS algorithm by computing

$$\mathbf{Q} + \gamma \cdot \text{counter} \cdot \mathbf{I}$$



(a) without whitening (b) in whitened space (c) unwhitened parameters

Figure 4.2.: Example of using MORE with a least squares approach for surrogate-modeling with and without whitening on the 2-dimensional Rosenbrock function (section 5.2.1). The plots show the predicted surrogate model parameters β. In (a) the parameters are estimated without whitening the data. In (b) a whitening transformation is applied to the data before parameter estimation and (c) shows the parameters from (b) after reversing the whitening transformation. We can see that the parameters in whitened space stay in a smaller range making the estimation task easier.

before adding the model noise to the covariance matrix of the parameters. This improved our results on the Rosenbrock test function compared to the RLS version with constant model noise for all samples. Still the use of a sample pool is theoretically unsound and part of future work is to explore ways to avoid it.

4.2.2. Whitening

We used *Cholesky whitening* which is based on Cholesky factorization of the precision matrix $\mathbf{L}\mathbf{L}^T = \Sigma^{-1}$, which leads to the whitening matrix $\mathbf{W}^{\text{Chol}} = \mathbf{L}^T$. If the Cholesky whitening is unsuccessful due to numerical problems, we instead standardize the random vector meaning $\text{var}(\mathbf{z}) = 1$ but the correlations are not removed. In figure 4.2 we provide an example illustrating the effect of whitening for our parameter estimation task.

4.3. MORE with Recursive Surrogate-Modeling

We now present the MORE algorithm with recursive surrogate-modeling. First we initialize the search distribution $\pi(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\,\mu, \Sigma)$ and the RLS algorithm. In the robot learning setting the search distribution may be intialized through imitation learning. In each iteration the search distribution is then used to create samples which are evaluated on the objective function. The samples are used to create the design matrix $\mathbf{X}$ (as in 4.2). Data preprocessing techniques like whitening (section 2.6.1) and normalization may be apllied to the design matrix and rewards before they are given to the RLS algorithm to learn the surrogate model, which in turn

is used to compute the update to the search distribution in closed form. MORE with Recursive Surrogate-Modeling is summarized in Algorithm 2.

Algorithm 2 : MORE with Recursive Surrogate-Modeling

Input : Parameters ε and β,
K number of iterations, N samples per iteration
begin Initialization:
    Initialize search distribution π
    Initialize RLS with $\mathbf{m}_0, \mathbf{P}_0$
end
for $k = 1, ..., K$ **do**
    for $n = 1, ..., N$ **do**
        Draw sample $\mathbf{x}_n \sim \pi$
        Evaluate $\mathbf{x}_n$ on objective function $f(\mathbf{x}_n) = y_n$
    end
    begin Estimate quadratic model $\hat{f}$
        for $n = 1, ..., N$ **do**
            Whitening: $\mathbf{X}_n = \mathbf{W}\mathbf{X}_n$
            Optionally: Increase model noise for older samples
            Compute surrogate model parameters using Algorithm 1
        end
    end
    Solve $\text{argmin}_{\eta>0,\omega>0}\, g(\eta, \omega)$ using Equation (A.1)
    Update search distribution π using Equation (2.2)
end

Chapter 5.

Evaluation

In this chapter we will describe the setup for our experiments and discuss the results we obtained.

5.1. Setup

All algorithms were implemented in python, the optimization algorithm for MORE is Low-storage BFGS from nlopt[1]. The clusterwork2 (cw2) framework, developed at the ALR group, was used for running experiments in an convenient way. The experiments were conducted on a machine with two 8-core AMD Ryzen 2700X processores clocked at 2.6 GHz and 31GB of RAM.

5.1.1. Hyperparameter Search

We did a mixture of manual hyperparameter tuning and grid search with the cw2 framework. (name some number of different parameter sets tested) The parameters used for optimization are listed.

[1] https://nlopt.readthedocs.io/en/latest/

MORE parameters

- KL-Bound ε
- entropy-loss bound β

RLS parameters

- model noise $\mathbf{Q}$
- measurement noise σ^2

5.2. Experiments

5.2.1. Test Functions for Optimization

Test functions provide artificial landscapes to evaluate the performance of optimization algorithms. They can be used to assess robustness, convergence speed and general performance of the algorithms (Molga and Smutnicki, 2005).

The Rosenbrock function (figure 5.1) is unimodal and convex. It has one global optimum which is inside a parabolic shaped flat valley. It is usually easy to find the valley, however finding the global optimum is particularly difficult, which makes this function popular with testing algorithms' performance. It is defined as

$$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2].$$

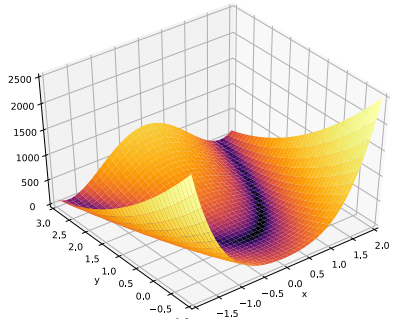


Figure 5.1.: 2D Rosenbrock function

We test our algorithm on a 8 dim and 15 dimensional version of the rosenbrock function

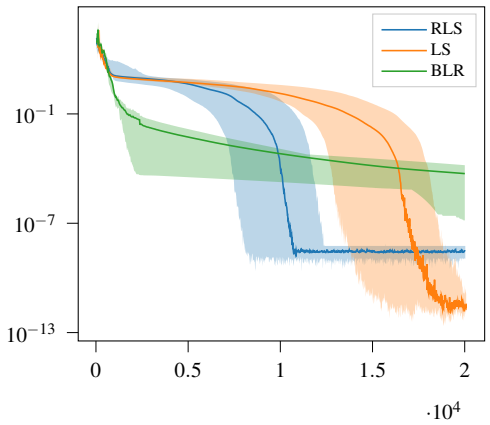


Figure 5.2.: This figure shows the median of (10 runs) for RLS, LS and BLR (original MORE) on 15 dimensional Rosenbrock

5.2.2. Planar Reaching Tasks

Dynamic Movement Primitives

- dynamic system-based motor primitives based on work [Ijspeert 2002, Schaal 2003,2007] improved imitation and reinforcementlearning

- represent both discrete point-to-point movements as well as rhythmic motion with motor primitives

- first system canonical system (phase variable)

- we can build complex motor skills from basic primitives, its based on dynamical systems

- discrete movement primitives using only a single first order system

- key advantage the second system is linear in the shape parameters and can be efficiently learned

- initialized with imitation learning

- DMPs as policy representation

5.2.3. Via Point Reaching Task

- TODO: create figure to explain task

We used a 5 link robot, similar to MORE setup. Therefore 25 dimensions of the problem, which should be at viapoint (1,1) at time step 100 and at viapoint (5,0) at timestep 200. In Figure 5.3 we see the task solved by the original MORE algorithm.

Problem of 351 parameters for recursive least squares

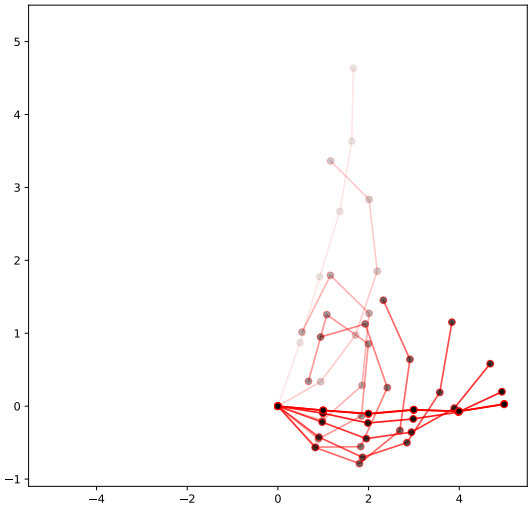


Figure 5.3.: This figure shows the movement resulting from the policy learned by BLR for the viapoint reaching task. The postures of the resulting motion are shown as overlay, where darker postures indicate a posture which is close in time to the hole.

5.2.4. Tests on Performance

Here I want to include some tests on different versions of final algorithm and what we learned form experiments. Like using weighted model noise on older samples, and normalization but also different KL-bounds etc.

Tests on data preprocessing methods, different normalization techniques. Whitening very important, making it possible to track the model parameters more easily.

- compare normalization techniques (moving average, batch normalization)

- best performance comparison (more iterations)

- best performance comparison of different methods (total samples)

- test influence of KL bound

- test influence of whitening, normalization?

- test influence of noise of drift model

5.3. Evaluation

- some theoretical problems: sample pool, model noise

- improve sample efficency of rosenbrock and reaching task, better than original MORE and beating CMA-ES benchmark

Chapter 6.

Conclusion and Future Work

6.1. Conclusion

- managed to improve sample efficiency on simple test functions

- epsilon for kl-bound should have effect on effectiveness of RLS since the surrogate models become more or less locally correlated

We achieved some good results on the rosenbrock test function considerably improving the sample efficiency. Nonetheless for the planar reaching tasks we did not manage to achieve a significant improvement. Also our proposed algorithm did not work with multi modal and noisy objective functions.

- One main problem seems to be tuning the recursive estimation for one specific task.

The data seems to be difficult to Still

- inexperience of author with kalman filter (setting up matrices, choosing good state transition model)

- use some form of dimensionality reduction as we deal with high dimensional –> need to estimate $1 + 25 + (25 \times 26/2) = 351$

6.2. Future Work

- remove sample pool

- usage of state transition model (kalman filter prediction step)

- make learning more stable for reaching tasks (safe exploration)

- learning the model noise: run original MORE approach and do backpropagation with pytorch over the weights

Bibliography

A. Abdolmaleki, R. Lioutikov, J. R. Peters, N. Lau, L. Pualo Reis, and G. Neumann. Model-based relative entropy stochastic search. *Advances in Neural Information Processing Systems*, 28:3537–3545, 2015.

R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

S. Chen. Kalman filter for robot vision: a survey. *IEEE Transactions on Industrial Electronics*, 59(11):4409–4420, 2011.

M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. Citeseer, 2011.

M. P. Deisenroth, G. Neumann, and J. Peters. *A survey on policy search for robotics*. now publishers, 2013.

A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian data analysis*. CRC press, 2013.

N. Hansen. The cma evolution strategy: a comparing review. *Towards a new evolutionary computation*, pages 75–102, 2006.

A. E. Hoerl, R. W. Kannard, and K. F. Baldwin. Ridge regression: some simulations. *Communications in Statistics-Theory and Methods*, 4(2):105–123, 1975.

A. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. *Advances in neural information processing systems*, 15:1547–1554, 2002.

R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, Mar. 1960.

B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg. A survey of research on cloud robotics and automation. *IEEE Transactions on automation science and engineering*, 12(2):398–409, 2015.

A. Kessy, A. Lewin, and K. Strimmer. Optimal whitening and decorrelation. *The American Statistician*, 72(4):309–314, 2018.

J. Kober and J. Peters. Policy search for motor primitives in robotics. *Machine learning*, 84 (1-2):171–203, 2011.

J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

A. G. Kupcsik, M. P. Deisenroth, J. Peters, G. Neumann, et al. Data-efficient generalization of robot skills with contextual policy search. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence, AAAI 2013*, pages 1401–1407, 2013.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

M. Molga and C. Smutnicki. Test functions for optimization needs. *Test functions for optimization needs*, 101:48, 2005.

A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287, 1999.

J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

T. Osa, J. Pajarinen, G. Neumann, J. Bagnell, P. Abbeel, and J. Peters. An algorithmic perspective on imitation learning. *Foundations and Trends in Robotics*, 7(1-2):1–179, Mar. 2018.

J. Peters, K. Mülling, and Y. Altun. Relative entropy policy search. In *AAAI*, volume 10, pages 1607–1612. Atlanta, 2010.

M. J. Powell. The bobyqa algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, pages 26–46, 2009.

S. Russell. Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 101–103, 1998.

S. Särkkä. *Bayesian filtering and smoothing*, volume 3. Cambridge University Press, 2013.

S. Schaal. The new robotics—towards human-centered machines. *HFSP journal*, 1(2): 115–126, 2007.

S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Learning movement primitives. In *Robotics research. the eleventh international symposium*, pages 561–572. Springer, 2005.

J. G. Schneider. Exploiting model uncertainty estimates for safe dynamic control learning. *Advances in neural information processing systems*, pages 1047–1053, 1997.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

R. S. Sutton, A. G. Barto, and R. J. Williams. Reinforcement learning is direct adaptive optimal control. *IEEE Control Systems Magazine*, 12(2):19–22, 1992.

V. Tangkaratt, H. van Hoof, S. Parisi, G. Neumann, J. Peters, and M. Sugiyama. Policy search with high-dimensional context variables. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

S. Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber. Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980, 2014.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Appendix A.

Appendix

A.1. MORE: Equations

- add derivation of lagragian function for MORE with clear steps

Forming the Lagrangian for our optimization problem we get

$$\mathfrak{L}(\pi,\eta,\omega) = \int \pi(\theta)\mathcal{R}_\theta d\theta \; + \; \eta\left(\varepsilon - \int \pi(\theta)\,\log\frac{\pi(\theta)}{q(\theta)}d\theta\right) - \; \omega\left(\beta + \int \pi(\theta)\,\log(\pi(\theta))d\theta\right)$$

Optimizing the Lagrangian by computing the derivative with respect to the mean and covariance matrix yields

$$\pi(\theta) \propto q(\theta)^{\eta/(\eta+\omega)}\exp\left(\frac{\mathcal{R}_\theta}{\eta+\omega}\right)$$

the solution depends on the quadratic and linear term of the surrogate model and the Lagrangian multipliers η and ω. These in turn can be obtained by minimizing the dual function:

$$g(\eta,\omega) = \eta\varepsilon - \omega\beta + (\eta-\omega)\log\left(\int q(\theta)^{\frac{\eta}{\eta+\omega}}\exp\left(\frac{\mathcal{R}_\theta}{\eta+\omega}\right)d\theta\right) \tag{A.1}$$

Assuming we are given a quadratic surrogate model

$$\mathcal{R}_\theta \approx \theta^T \mathbf{R}\theta + \theta^T \mathbf{r} + r_0$$

we can solve the dual function in closed form.

$$g(\eta,\omega) = \eta\varepsilon - \beta\omega + \frac{1}{2}\left(\mathbf{f}^T\mathbf{F}\mathbf{f} - \eta\mathbf{b}^T\mathbf{Q}^{-1}\mathbf{b} - \eta\log|2\pi\mathbf{Q}|p + (\eta+\omega)\log|2\pi(\eta+\omega)\mathbf{F}|\right)$$

with $\mathbf{F} = (\eta\Sigma^{-1} - 2\mathbf{R})^{-1}$ and $\mathbf{f} = \eta\Sigma^{-1}\mu + \mathbf{r}$

A.2. Gaussian probability function

A random variable $x \in \mathbb{R}^n$ has a Gaussian distribution with mean $m \in \mathbb{R}^n$ and covariance $P \in \mathbb{R}^{n\times n}$ if its probability density has the form

$$\mathcal{N}(x|m,P) = \frac{1}{(2\pi)^{n/2}|P|^{1/2}}\exp\left(-\frac{1}{2}(x-m)^T P^{-1}(x-m)\right)$$

where $|P|$ is the determinant of the matrix P.

A.3. Kalman Filter Derivation