Domain of attraction for Gaussian Processes State Space Models

BACHELOR THESIS

by
Frederik Fraaz

Submitted to the Chair of
INFORMATION-ORIENTED CONTROL
Technical University of Munich

Univ.-Prof. Dr.-Ing. Sandra Hirche

Supervisor:	M. Sc. Thomas Beckers
Start:	19.05.2017
Intermediate Report:	18.08.2017
Delivery:	06.10.2017

TECHNISCHE UNIVERSITÄT MÜNCHEN

LEHRSTUHL FÜR INFORMATIONSTECHNISCHE REGELUNG

ORDINARIA: UNIV.-PROF. DR.-ING. SANDRA HIRCHE

17.05.2017

B A C H E L O R   T H E S I S

for

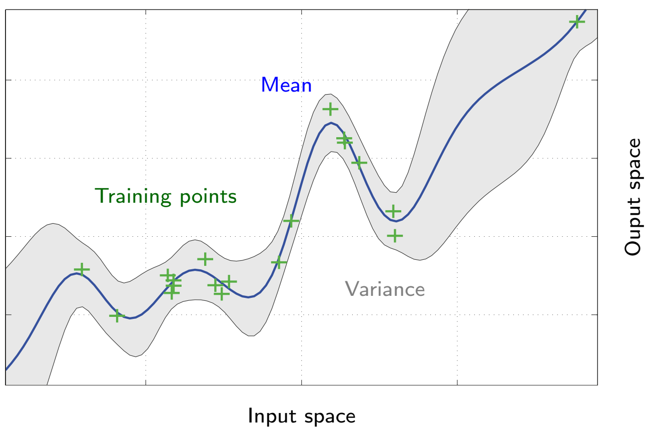
Frederik Fraaz

Mat.-Nr. 3660011, field of study EI

Domain of attraction for Gaussian Processes State Space Models

Problem description:

Currently, the Gaussian Process State Space Model (GP-SSM) is an upcoming model identification technique for nonlinear systems. The Gaussian Process generates data located throughout some domain such that any finite subset of the range follows a multivariate Gaussian distribution. The most important advantage is the prediction of the variance which contains the uncertainty of the identification. This offers a powerful tool for nonlinear function regression without any previous knowledge. Such a GP-SSM can be used in many different control applications which are based on system models, e.g model predictive control and adaptive control. Nevertheless, the fundamental system dynamics of GP-SSMs are very sparsely researched.



An important property of nonlinear systems is the stability and domains of attraction. In this thesis the student should derive a constructive and systematically applicable samplingbased approach to determine stochastic domains of attraction for GP-SSMs. For this purpose, a Lyapunov function is analyzed on a finite sampling of a bounded set and then it is extended to an infinite set by exploiting continuity properties of the GP-SSM.

Tasks:

- Stochastic extension of samplingbased approach
- Analyze the stability and the domain of attraction with examples

Bibliography:

[1] T. Beckers; S. Hirche. Stability of Gaussian Process State Space Models. Proceedings of the European Control Conference (ECC), 2016.

[2] R. Bobiti, M. Lazar. A sampling approach to finding Lyapunov functions for nonlinear discretetime systems. Proceedings of the European Control Conference (ECC), 2016.

Supervisor:	M. Sc. Thomas Beckers
Start:	19.05.2017
Intermediate Report:	18.08.2017
Delivery:	06.10.2017

(S. Hirche)
Univ.-Professor

Abstract

Gaussian Process State Space Models (GP-SSMs) are an application of Gaussian processes (GPs) to model the dynamics of nonlinear discrete-time systems. Gaussian processes are a form of machine learning model that can be applied to both regression and classification problems. One of the advantages of Gaussian processes compared to other machine learning techniques is that GPs also provide the uncertainty of each prediction. This thesis develops a sampling-based approach for verifying stability of GP-SSMs by extending an existing sampling-based approach from deterministic to stochastic systems. The approach employs multi-resolution sampling to efficiently explore the state-space. Moreover, multi-step ahead predictions are used to increase the estimated region of stability. To further increase the estimated region of stability, an expression for calculating the Lipschitz constant of the property function is found which relies on rigorous verification methods instead of naive interval analysis. The algorithm is implemented in Matlab and the results are evaluated through examples.

Zusammenfassung

Gaussian Process State Space Models (GP-SSMs) sind eine Anwendung von Gaußprozessen um die Dynamik von nichtlinearen zeitdiskreten Systemen zu modellieren. Gaußprozesse sind eine Form des Maschinellen Lernens, die sowohl auf Regressions- als auch auf Klassifikationsprobleme angewandt werden können. Einer der Vorteile von Gaußprozessen im Vergleich zu anderen Methoden des Maschinellen Lernens ist der, dass Gaußprozesse auch die Unsicherheit der Vorhersage bereitstellen. In dieser Arbeit wird ein Stichprobenverfahren für die Verifikation der Stabilität von GP-SSMs entwickelt, indem ein bereits existierendes Stichprobenverfahren von deterministischen auf stochastische Systeme erweitert wird. Das Verfahren verwendet eine variable Auflösung bei der Auswahl der Stichproben um den Zustandsraum effizient untersuchen zu können. Darüber hinaus werden Mehrschrittvorhersagen verwendet um die geschätzte Stabilitätsregion zu vergrößern. Zur weiteren Vergrößerung der geschätzten Stabilitätsregion wird ein Ausdruck für die Lipschitz-Konstante gefunden, der auf genauen Verifikationsmethoden anstelle von naiver Intervallarithmetik basiert. Der Algorithmus wird in Matlab implementiert und die Ergebnisse werden anhand von Beispielen evaluiert.

Contents

Chapter 1

Introduction

Stability is one of the most important properties in the analysis of nonlinear systems and critical to the design of control systems. The core of stability theory was developed by Maxwell [Max68], Routh [Rou77], and Lyapunov [Lya92] in the late 19th century. While Maxwell and Routh proposed methods of stability verification via linearization, Lyapunov's main contribution is the use of energy functions to directly verify stability of nonlinear systems.

About 80 years later, Lyapunov's theory was extended to stochastic systems by Kushner [Kus67], Arnold [Arn72], Friedman [Fri76] and others. Lyapunov's original work and its extension to stochastic systems focus on continuous-time systems. A systematic theory for stability analysis of stochastic discrete-time systems has been developed more recently.

Gaussian Process State Space Models (GP-SSMs) are an application of Gaussian processes (GPs) to model the dynamics of nonlinear discrete-time systems. GPs are a form of machine learning that can be applied to regression and classification problems. One of the advantages of Gaussian processes compared to other machine learning techniques is that GPs also provide the uncertainty of each prediction.

In many real world applications of GP-SSMs it is critical to have stability guarantees, i.e. it is important to know that there exists a region in which the GP-SSM does not exhibit unpredictable behavior.

The goal of this thesis is to develop a systematically applicable approach to finding such regions of stability for GP-SSMs. For this purpose, the sampling-based approach proposed by Bobiti and Lazar [BL16a] is extended from deterministic to stochastic systems.

Stability analysis of Gaussian processes has been of interest lately. Beckers and Hirche [BH16] investigated conditions for the number of equilibrium points and for stability of GP-SSMs, while Berkenkamp et al. [BMSK16] used a Lyapunov approach for sampling-based exploration of the state space to better understand the dynamics of the underlying system. Ito, Fujimoto, and Tadokoro [IFT17] used a sampling-based approach to design stable controllers for Gaussian processes and Vinogradska et al. [VBNT$^+$16] proposed an algorithm that finds a region of the state space in which the closed-loop system is provably stable. While a constant sampling resolution is used by Vinogradska et al., the algorithm in this thesis employs multi-resolution sampling to efficiently explore the state space.

Compared to classical Lyapunov methods, which rely on finding a specific Lyapunov function candidate for each system, the sampling-based approach from [BL16a] has the advantage that it is independent of the system dynamics. Furthermore, the algorithm is decentralized in the sampling points and can thus easily be parallelized to reduce computation time.

The remainder of this thesis is organized as follows: The basics of GP-SSMs, an overview of stability analysis, and basics of δ-sampling are introduced in Chapter 2. Chapter 3 contains the main contribution of this thesis, i.e. the extension of the sampling-based approach from deterministic to stochastic systems. Chapter 4 describes how the algorithm is implemented. The approach is evaluated through examples in Chapter 5, and Chapter 6 concludes the thesis.

Chapter 2

Preliminaries

This chapter starts with basic notation in Section 2.1, and gives an overview of Gaussian Process State Space Models (GP-SSMs) in Section 2.2. Furthermore, basics of deterministic stability analysis and stochastic stability analysis are introduced in Section 2.3 and Section 2.4, respectively. Section 2.5 gives the background required to understand sampling-based verification of Lyapunov's inequality.

2.1 Basic Notation

Denote by $\mathcal{K}$ the class of functions $\alpha : \mathbb{R}_+ \to \mathbb{R}_+$ for which it holds that α is continuous, strictly increasing, and $\alpha(0) = 0$. A function α belongs to class $\mathcal{K}_\infty$ if $\alpha \in \mathcal{K}$ and $\lim_{s\to\infty} \alpha(s) = \infty$.
The function $\beta : \mathbb{R}_+ \times \mathbb{R}_+ \to \mathbb{R}_+$ is said to belong to class $\mathcal{KL}$ if for each fixed $s \in \mathbb{R}_+$, $\beta(\cdot, s) \in \mathcal{K}$ and for each fixed $r \in \mathbb{R}_+$, $\beta(r, \cdot)$ is decreasing and $\lim_{s\to\infty} \beta(r, s) = 0$. Let $\mathcal{X}_r = \{\boldsymbol{x} \in \mathbb{R}^n : \|\boldsymbol{x}\| < r\}$ for $r > 0$.
Denote by $C^i(\mathcal{X}_r)$ the class of functions $V(\boldsymbol{x})$ i-times continuously differentiable with respect to $\boldsymbol{x} \in \mathcal{X}_r$. $\alpha^h := \alpha \circ \ldots \circ \alpha$ is the h-times map composition of $\alpha : \mathbb{C} \to \mathbb{C}$.
The identity function $id : \mathcal{X} \to \mathcal{X}$ is defined s.t. $id(\boldsymbol{x}) = \boldsymbol{x}$, $\forall \boldsymbol{x} \in \mathcal{X}$ and $\mathcal{X} \subseteq \mathbb{R}^n$.
Let $\mathcal{B}_\delta(\boldsymbol{x})$ denote a δ-radius ball around $\boldsymbol{x}$ in $\mathbb{R}^n$. $\overline{\mathcal{X}}$ is the closure of the set $\mathcal{X}$, and $int(\mathcal{X})$ is the interior of the set $\mathcal{X}$.
Denote by $\mathbb{E}[X]$ the expected value of a random variable X. $\mathcal{N}(\mu, \Sigma)$ is the normal distribution with mean μ and covariance Σ.

2.2 Gaussian Process State Space Models

Gaussian Process State Space Models [FCR14] can be used to learn the dynamics of nonlinear discrete-time systems. They consist of several Gaussian processes, which are a generalization of the Gaussian probability distribution from scalars or vectors to functions.

Definition 2.1 (see [RW06])**.** A Gaussian process (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution.

The mean function $m(\boldsymbol{x}) \in C^0$ together with the covariance function $k(\boldsymbol{x}, \boldsymbol{x'}) \in C^0$ completely specify a Gaussian process:

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x'})) \tag{2.1}$$
$$m(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})] \tag{2.2}$$
$$k(\boldsymbol{x}, \boldsymbol{x'}) = \mathbb{E}[(f(\boldsymbol{x}) - m(\boldsymbol{x}))(f(\boldsymbol{x'}) - m(\boldsymbol{x'}))], \tag{2.3}$$

where $\boldsymbol{x}, \boldsymbol{x'} \in \mathcal{X}$, $m(\boldsymbol{x}) : \mathcal{X} \to \mathbb{R}$, $k(\boldsymbol{x}, \boldsymbol{x'}) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, and $\mathcal{X} \subset \mathbb{R}^n$. For simplicity, we set the mean function to zero without limiting the expressive power of the model. One of the most widely used covariance functions is the squared exponential covariance function:

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \sigma_f^2 \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x'}\|^2}{2\lambda^2}\right) \tag{2.4}$$

with the hyperparameters σ_f^2 and λ. σ_f^2 is the signal variance of the GP and λ is the characteristic length-scale, which determines the mean number of level-zero upcrossings of the Gaussian process.
The following introduction to GP-SSMs is based on [BH16].
GP-SSMs are used to predict the next step ahead state $\boldsymbol{x}_{k+1}$ based on the current state $\boldsymbol{x}_k$, where $\boldsymbol{x}_k \in \mathcal{X}$ is the state of the system:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k), \quad k \in \mathbb{N} \tag{2.5}$$
$$\boldsymbol{f}(\boldsymbol{x}_k) \sim \mathcal{GP}(\boldsymbol{m}(\boldsymbol{x}_k), \boldsymbol{k}(\boldsymbol{x}_k, \boldsymbol{x}_k')) \tag{2.6}$$

Since the output of Gaussian process regression is always a scalar, we need n independent GPs to represent the dynamics of a n-dimensional system. Hence, both the mean $\boldsymbol{m}(\cdot)$ and covariance $\boldsymbol{k}(\cdot,\cdot)$ in (2.6) are $n \times 1$ vectors that aggregate the mean and covariance functions of each dimension in state space.
We train the GP-SSM using m training inputs and outputs, aggregated in the matrices $X = [\boldsymbol{x}_{j_1}, \boldsymbol{x}_{j_2}, \ldots, \boldsymbol{x}_{j_m}]$ and $Y = [\boldsymbol{x}_{j_1+1}, \boldsymbol{x}_{j_2+1}, \ldots, \boldsymbol{x}_{j_m+1}]^T$, where $j_i \in \mathbb{N}$ and $\boldsymbol{x} \in \mathcal{X}$.
The i-th component of $\boldsymbol{x}_{k+1}$ can be predicted by

$$x_{i,k+1} \sim \mathcal{N}(\mu_i(\boldsymbol{x}_{k+1}), \mathrm{var}_i(\boldsymbol{x}_{k+1})) \tag{2.7}$$
$$\mu_i(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k) = \boldsymbol{k}_{\varphi_i}(\boldsymbol{x}_k, X)^T (K_{\varphi_i}(X, X) + I\sigma_{n,i}^2)^{-1} \boldsymbol{y} \tag{2.8}$$
$$\mathrm{var}_i(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k) = k_{\varphi_i}(\boldsymbol{x}_k, \boldsymbol{x}_k) - \boldsymbol{k}_{\varphi_i}(\boldsymbol{x}_k, X)^T (K_{\varphi_i}(X, X) + I\sigma_{n,i}^2)^{-1} \boldsymbol{k}_{\varphi_i}(\boldsymbol{x}_k, X), \tag{2.9}$$

where $\boldsymbol{k}_{\varphi_l}(\boldsymbol{x}_k, X) : \mathcal{X} \times \mathcal{X}^m \to \mathbb{R}^m$ is the vector of covariances between the state $\boldsymbol{x}_k$ and the m training examples, $K_{\varphi_i}(X, X) : \mathcal{X}^m \times \mathcal{X}^m \to \mathbb{R}^{m \times m}$ is the $m \times m$ matrix of covariances between all pairs of training examples, $\boldsymbol{y}$ is the i-th column of Y, φ_i

denotes the hyperparameters corresponding to the i-th dimension of state space, and $\sigma_{n,i}^2 \in \mathbb{R}_+^*$, $\forall i \in \{1, \dots, n\}$ is the noise variance.
For notational simplicity, we combine the n normal distributed components of $\boldsymbol{x}_{k+1}$ in a multivariate distribution:

$$\boldsymbol{x}_{k+1} \sim \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}_{k+1}), \mathbf{var}(\boldsymbol{x}_{k+1})I) \tag{2.10}$$
$$\boldsymbol{\mu}(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k) = [\mu_1(\boldsymbol{x}_{k+1}), \mu_2(\boldsymbol{x}_{k+1}), \dots, \mu_n(\boldsymbol{x}_{k+1})]^T \tag{2.11}$$
$$\mathbf{var}(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k) = [\mathrm{var}_1(\boldsymbol{x}_{k+1}), \mathrm{var}_2(\boldsymbol{x}_{k+1}), \dots, \mathrm{var}_n(\boldsymbol{x}_{k+1})]^T \tag{2.12}$$

2.3 Deterministic Stability

Consider the nonlinear discrete-time system

$$\boldsymbol{x}_{k+1} = \boldsymbol{G}(\boldsymbol{x}_k), \tag{2.13}$$

where $\boldsymbol{G} : \mathbb{R}^n \to \mathbb{R}^n$ is a nonlinear-map, $\boldsymbol{x}_k \in \mathcal{X}$ is the state, and $\mathcal{X}$ is a compact set with $0 \in int(\mathcal{X})$. Assume that the origin of state space is an equilibrium point $\boldsymbol{x}^*$, i.e. $\boldsymbol{G}(\boldsymbol{x}^*) = 0$.
The most basic definition of stability is:

Definition 2.2. The equilibrium $\boldsymbol{x}^* = 0$ of (2.13) is stable if for each $\epsilon > 0$ there exists a $\delta > 0$ such that

$$\|\boldsymbol{x}_0\| < \delta \Rightarrow \|\boldsymbol{x}_k\| < \epsilon, \quad \forall k > 0. \tag{2.14}$$

Definition 2.2 can be made more strict by demanding that the initial condition $\boldsymbol{x}_0$ converges to the origin instead of just demanding that it does not leave a ball around the origin:

Definition 2.3. The equilibrium $\boldsymbol{x}^* = 0$ of (2.13) is asymptotically stable if it is stable and there exists a δ such that

$$\|\boldsymbol{x}_0\| < \delta \Rightarrow \lim_{k\to\infty} \|\boldsymbol{x}_k\| = 0. \tag{2.15}$$

The following definitions are useful in stating the theorem for proving stability of nonlinear discrete-time systems:

Definition 2.4. A continuous function $V(\boldsymbol{x})$ defined on $\mathcal{X}_r$ is said to be positive definite if $V(0) = 0$ and $V(\boldsymbol{x}) \geq \alpha(\boldsymbol{x})$ for some $\alpha \in \mathcal{K}$.

Definition 2.5. Define $\Delta V(\boldsymbol{x}_k) := V(\boldsymbol{x}_k) - V(\boldsymbol{x}_{k-1})$, where $V : \mathcal{X} \to \mathbb{R}_+$ and $\boldsymbol{x}_k \in \mathcal{X}$.

Definition 2.6. A set $\mathcal{X} \subseteq \mathbb{R}^n$ is called invariant with respect to the system (2.13) if

$$\forall \boldsymbol{x}_0 \in \mathcal{X} \Rightarrow \boldsymbol{x}_k \in \mathcal{X}, \quad \forall k > 0. \tag{2.16}$$

Lyapunov proposed a method for verifying stability of nonlinear systems without having to find the actual solution to the difference equation:

Theorem 2.7 (Lyapunov's Direct Method)**.** *Let* $\boldsymbol{x}^* = 0$ *be an equilibrium point of* (2.13) *and* $V : \mathcal{X} \to \mathbb{R}_+$ *be a continuous, positive-definite function such that*

$$\Delta V(\boldsymbol{x}_k) \leq 0, \quad \forall \boldsymbol{x}_k \in \mathcal{X}, \tag{2.17}$$

then $\boldsymbol{x}^* = 0$ *is stable. Moreover, if*

$$\Delta V(\boldsymbol{x}_k) < 0, \quad \forall \boldsymbol{x}_k \in \mathcal{X}, \tag{2.18}$$

then $\boldsymbol{x}^* = 0$ *is asymptotically stable.*

As we can see, verification of stability reduces to the task of finding a function $V(\boldsymbol{x})$ which satisfies the conditions in Theorem 2.7.

2.4 Stochastic Stability

This section introduces definitions of stochastic stability and theorems for proving different stochastic stability criteria. The definitions and theorems are adopted from [LZL13] with minor changes.
Consider the n-dimensional stochastic discrete-time system

$$\boldsymbol{x}(t+1) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{w}(t), t), \quad \boldsymbol{x}(t_0) = \boldsymbol{x}_0, \tag{2.19}$$

where $\boldsymbol{x}_0 \in \mathbb{R}^n$ is a constant vector, $\boldsymbol{f} : \mathcal{X} \to \mathbb{R}^n$ with $\mathcal{X} \subseteq \mathbb{R}^n$, and $\boldsymbol{w}(t)$ is a stochastic process defined on the complete probability space (Ω, F, P). As with deterministic systems, we assume that the origin of state space is an equilibrium point $\boldsymbol{x}^* = 0$.
The following two definitions extend the notion of stability and asymptotic stability from deterministic to stochastic systems:

Definition 2.8. The equilibrium point $\boldsymbol{x}^*$ of (2.19) is stochastically stable or stable in probability if, for every $\epsilon > 0$ and $h > 0$, there exists a $\delta = \delta(\epsilon, h, t_0) > 0$ such that

$$P\{|\boldsymbol{x}(t)| < h\} \geq 1 - \epsilon, \quad \forall t \geq t_0, \tag{2.20}$$

when $|\boldsymbol{x}_0| < \delta$. Otherwise it is stochastically unstable. If the previous δ is independent of t_0, that is, $\delta = \delta(\epsilon, h) > 0$, then the trivial solution of (2.19) is stochastically uniformly stable in probability.

Definition 2.9. The equilibrium point $\boldsymbol{x}^*$ of (2.19) is stochastically asymptotically stable in probability if it is stochastically stable, and for every $\epsilon > 0$, there exists a $\delta = \delta(\epsilon, t_0) > 0$ such that

$$P\left\{\lim_{t\to\infty} \boldsymbol{x}(t) = 0\right\} \geq 1-\epsilon, \tag{2.21}$$

when $|\boldsymbol{x}_0| < \delta$.

Theorem 2.7 can be extended to stochastic systems by taking the expected value of $\Delta V(\boldsymbol{x}(t))$ to account for the stochastic nature of (2.19):

Theorem 2.10. *If there exists a positive definite function $V(\boldsymbol{x}) \in C^2(\mathcal{X}_r)$, such that*

$$\mathbb{E}[\Delta V(\boldsymbol{x}(t))] \leq 0, \quad \forall \boldsymbol{x}(t) \in \mathcal{X}_r, \tag{2.22}$$

then the trivial solution $\boldsymbol{x}^* = 0$ of (2.19) *is stochastically stable in probability.*

Theorem 2.11. *If there exists a positive definite function $V(\boldsymbol{x}) \in C^2(\mathcal{X}_r)$ and a function $\varphi \in \mathcal{K}$, such that*

$$\mathbb{E}[\Delta V(\boldsymbol{x}(t))] \leq -\mathbb{E}[\varphi(|\boldsymbol{x}(t)|)], \quad \forall \boldsymbol{x}(t) \in \mathcal{X}_r, \tag{2.23}$$

then the trivial solution $\boldsymbol{x}^* = 0$ of (2.19) *is stochastically asymptotially stable in probability.*

2.5 δ-Sampling and Finite-Step Lyapunov Functions

The definitions and propositions in this section are adopted from [BL15] and [BL16a] with minor changes.

Sampling

The idea behind sampling-based verification of stability is that we want to verify the decrease condition of a Lyapunov function on a set $\mathcal{S}$ without having to verify it in an infinite number of points. This is achieved by choosing a finite number of samples $\mathcal{S}_s$ of $\mathcal{S}$ such that any point $\boldsymbol{x} \in \mathcal{S}$ is within a certain maximum distance to some sampling point $\boldsymbol{x}_s \in \mathcal{S}_s$. The following definition states this notion of sampling more formally:

Definition 2.12. A sampling of a compact set $\mathcal{S} \subset \mathbb{R}^n$ is a finite set $\mathcal{S}_s$ s.t. for all $\boldsymbol{x} \in \mathcal{S}$, there exists at least a pair $(\boldsymbol{x}_s, \delta_{x_s}) \in \mathcal{S}_s \times \mathbb{R}_{>0}$ s.t. $\|\boldsymbol{x} - \boldsymbol{x}_s\| \leq \delta_{x_s}$.

The following definition is useful to allow for multi-resolution sampling of $\mathcal{S}$:

Definition 2.13. Let $\boldsymbol{y} \in \mathbb{R}^n$, $\delta \in \mathbb{R}_{>0}$ arbitrary. A refinement of $\mathcal{B}_\delta(\boldsymbol{y})$ is a finite set $S_{sy} \subset \mathcal{B}_\delta(\boldsymbol{y})$ s.t. for all $\boldsymbol{x} \in \mathcal{B}_\delta(\boldsymbol{y})$, there exists at least an $\boldsymbol{x}' \in \mathcal{S}_{sy}$ s.t. $\|\boldsymbol{x} - \boldsymbol{x}'\| \leq \frac{\delta}{2}$.

In other words, a refinement increases the sampling resolution around $\boldsymbol{y}$ by making δ smaller by a factor of 2. By using multi-resolution sampling, we are able to use a coarse grid of samples in areas in which the decrease condition of the Lyapunov function is easily verified, and a finer resolution toward the boundary of the region of stability. Figure 2.1 shows the intuition behind Definitions 2.12 and 2.13.

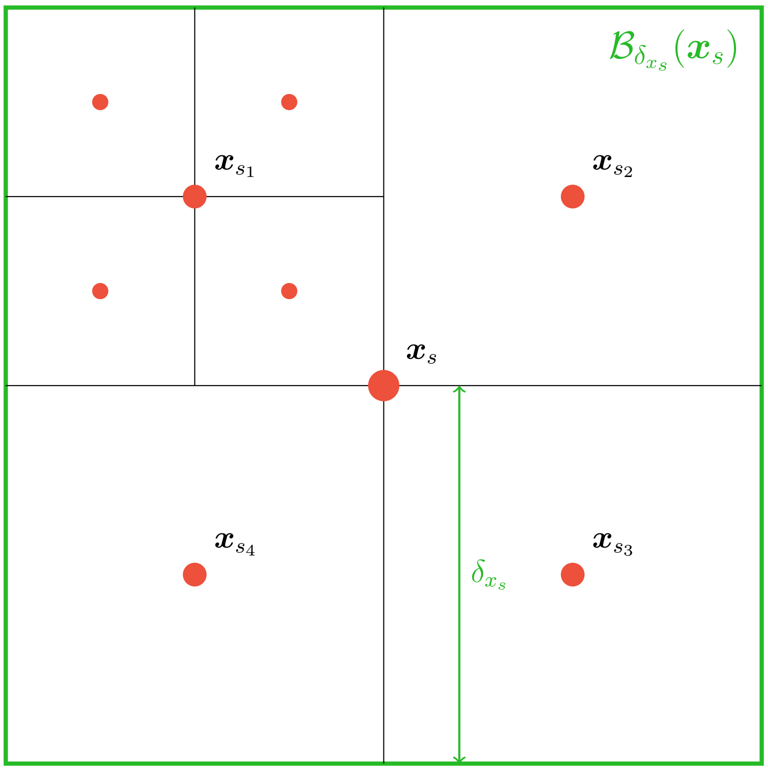


Figure 2.1: (see [BL16a]) Set sampling and refinement

Finite-Step Lyapunov Functions

Comparison functions are used widely in control theory, since they allow for succinct statements and proofs in the study of stability properties. The following two definitions use comparison functions of class $\mathcal{K}$ and $\mathcal{KL}$ to restate the definitions of stability from Section 2.3. These definitions are then used for stating some useful propositions.

Definition 2.14. The system (2.13) is called $\mathcal{KL}$-stable on $\mathcal{S}$ if there exists a $\mathcal{KL}$ function $\beta : \mathbb{R}_+ \times \mathbb{R}_+ \to \mathbb{R}_+$ such that $\|\boldsymbol{x}_{k+1}\| \leq \beta(\|\boldsymbol{x}_0\|, k)$ for all $(\boldsymbol{x}_0, k) \in \mathcal{S} \times \mathbb{Z}_+$.

Definition 2.15. A map $\boldsymbol{G} : \mathbb{R}^n \to \mathbb{R}^n$ is called $\mathcal{K}$-bounded on $\mathcal{S}$ if there exists a function $\omega \in \mathcal{K}$ such that

$$\|\boldsymbol{G}(\boldsymbol{x})\| \leq \omega(\|\boldsymbol{x}\|), \tag{2.24}$$

for all $\boldsymbol{x} \in \mathcal{S}$.

Using Definition 2.14 and Definition 2.15, we are now able to generalize Theorem 2.7 to Lyapunov functions that look more than one step ahead into the future:

Proposition 2.16. Let $\mathbb{W}$ be a compact set with $0 \in \mathit{int}(\mathbb{W})$, which is invariant with respect to the dynamics (2.13). Let $\alpha_1, \alpha_2 \in \mathcal{K}_\infty$. Suppose that the map $\boldsymbol{G}$

corresponding to the dynamics (2.13) is $\mathcal{K}$-bounded on $\mathbb{X}$ and there exists a function $V : \mathbb{R}^n \to \mathbb{R}_+$ such that

$$\alpha_1(\|\boldsymbol{x}\|) \leq V(\boldsymbol{x}) \leq \alpha_2(\|\boldsymbol{x}\|), \quad \forall \boldsymbol{x} \in \mathbb{W}, \tag{2.25}$$

and there exists an $M \in \mathbb{Z}_{\geq 1}$ and a corresponding $\rho \in \mathcal{K}$ with $\rho < id$ such that

$$V(\boldsymbol{G}^M(\boldsymbol{x})) \leq \rho(V(\boldsymbol{x})), \quad \forall \boldsymbol{x} \in \mathbb{W}. \tag{2.26}$$

Then, system (2.13) is $\mathcal{KL}$-stable in $\mathbb{W}$.

The function V which satisfies the conditions of Proposition 2.16 is called a finite-step Lyapunov function (FSLF) on $\mathbb{W}$.
If we rewrite (2.26) as

$$V(\boldsymbol{G}^M(\boldsymbol{x})) - \rho(V(\boldsymbol{x})) \leq 0, \tag{2.27}$$

then the expression looks similar to the decrease condition in Theorem 2.7. The major difference between (2.27) and (2.17) is, that (2.27) allows us to use M-step Lyapunov functions with $M > 1$.
Figure 2.2 shows the intuition behind the use of finite-step Lyapunov functions. Suppose we use $V(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{x}$ in this example, then $V(\boldsymbol{x})$ is equal to the square of the euclidean distance between a given state $\boldsymbol{x}$ and the origin. The red circle denotes the level set of V of value $L = V(\boldsymbol{x}_0)$. Since $\boldsymbol{G}(\boldsymbol{x}_0)$, $\boldsymbol{G}^2(\boldsymbol{x}_0)$, and $\boldsymbol{G}^3(\boldsymbol{x}_0)$ are outside of the red area, we know that $V(\boldsymbol{G}^i(\boldsymbol{x}_0)) - \rho(V(\boldsymbol{x}_0)) > 0$ for $i \in \{1, 2, 3\}$, i.e. the decrease condition of the FSLFs is not verified. But $\boldsymbol{G}^4(\boldsymbol{x}_0)$ is within the level set $\mathbb{V} := \{\boldsymbol{x} | V(\boldsymbol{x}) \leq L\}$ again and thus the decrease condition is verified for $M = 4$. This property of finite-step Lyapunov functions is exploited in Chapter 3 to increase the estimated region of stability.

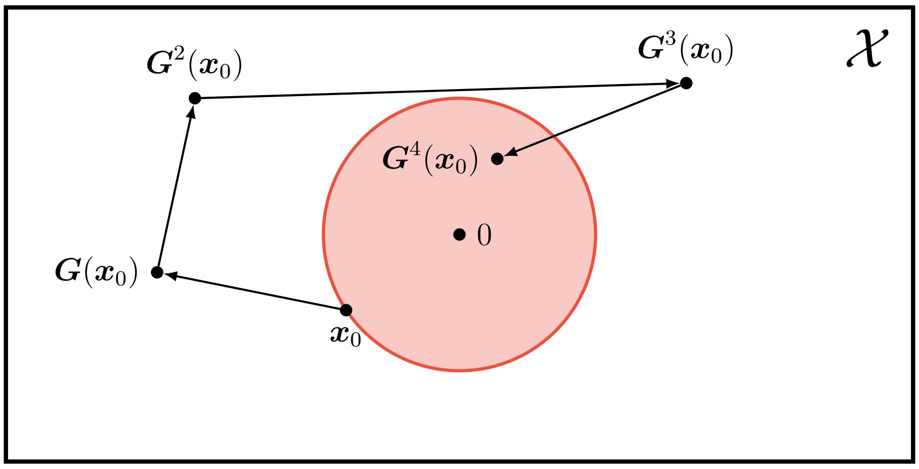

$\boldsymbol{G}^2(\boldsymbol{x}_0)$
$\boldsymbol{G}^3(\boldsymbol{x}_0)$
$\mathcal{X}$
$\boldsymbol{G}^4(\boldsymbol{x}_0)$
0
$\boldsymbol{G}(\boldsymbol{x}_0)$
$\boldsymbol{x}_0$

Figure 2.2: M-step Lyapunov functions

M finite-step Lyapunov functions are combined to obtain the final Lyapunov function of system (2.13) as follows:

Proposition 2.17. Let $\mathbb{W}$ be a compact invariant set with respect to the map $\boldsymbol{G}$, with $0 \in int(\mathbb{W})$. Let V be a FSLF on $\mathbb{W}$ for system (2.13) with $M \in \mathbb{Z}_{\geq 1}$ satisfying (2.26). Then the function $W : \mathbb{R}^n \to \mathbb{R}_+$ defined as

$$W(\boldsymbol{x}) = \sum_{j=0}^{M-1} V(\boldsymbol{G}^j(\boldsymbol{x})) \tag{2.28}$$

is a Lyapunov function for system (2.13) on $\mathbb{W}$.

Chapter 3

Sampling Approach

In this chapter we extend the sampling approach to finding Lyapunov functions proposed in [BL16a] from deterministic to stochastic systems, i.e. we propose a procedure for finding the domain of attraction (DOA) of a GP-SSM that models the dynamics of a discrete-time system. The DOA is defined as follows:

Definition 3.1 (see [BL16a])**.** The domain of attraction of the origin is the set of all initial states from which the dynamics converge to the origin.

The system considered in this chapter is a GP-SSM:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k), \quad k \in \mathbb{N} \tag{3.1}$$

$$\boldsymbol{f}(\boldsymbol{x}_k) \sim \mathcal{GP}(\boldsymbol{m}(\boldsymbol{x}_k), \boldsymbol{k}(\boldsymbol{x}_k, \boldsymbol{x}'_k)), \tag{3.2}$$

where the mean $\boldsymbol{m}(\cdot)$ and covariance $\boldsymbol{k}(\cdot,\cdot)$ in (3.2) are defined as in Section 2.2 and $\boldsymbol{x}_k \in \mathcal{X} \subset \mathbb{R}^n$.
The following steps to finding the DOA of a deterministic nonlinear discrete-time system are proposed in [BL16a]:

1. Use δ-sampling to find a set $\mathcal{A} \subseteq \overline{\mathcal{S}\backslash\mathcal{L}}$ on which the decrease condition of the Lyapunov function is verified, where $\mathcal{S} \subset \mathbb{R}^n$ is the search space and $\mathcal{L}$ is defined as in the next step.

2. Find a region of stability $\mathcal{L}$ in the neighborhood of the origin via linearization.

3. The system is stable on the set $\mathbb{W}$, where $\mathbb{W}$ is the maximum level set of the Lyapunov function inside $\mathcal{A} \cup \mathcal{L}$.

A detailed explanation of these steps and an extension to stochastic systems is found in the following sections. Figure 3.1 shows a diagram of the sets used in these steps.

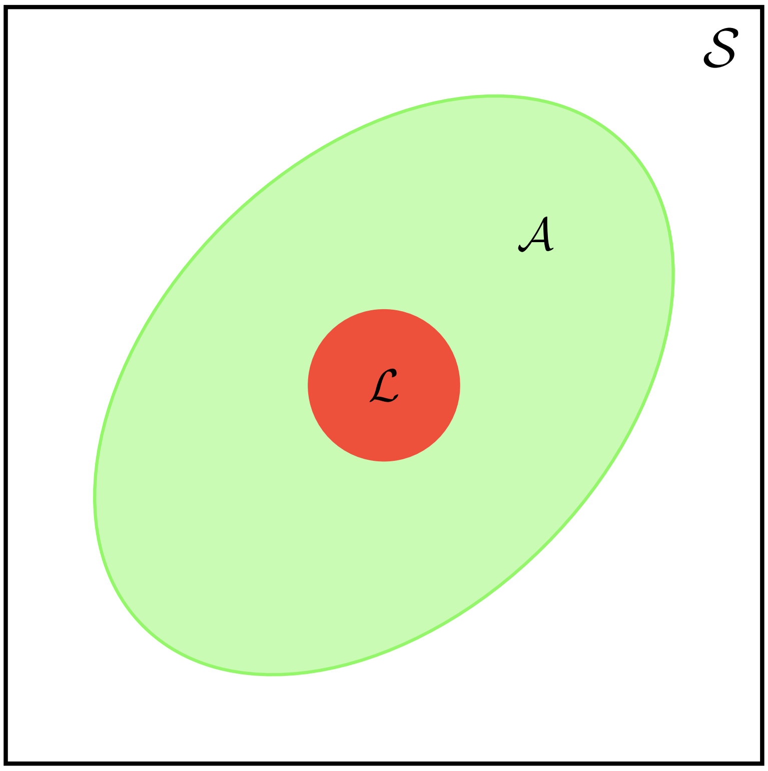


Figure 3.1: Sets $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{L}$

3.1 Sampling-Based Verification of Lyapunov's Inequality

In this section, we describe how the sampling-based verification of Lyapunov's inequality works. For this purpose, we first discuss how M-step ahead predictions of the GP can be calculated, and then define a property function for sampling-based verification of the decrease condition. Finally, we state a theorem and procedure for verifying the property function.

3.1.1 Multiple-Step Ahead Predictions

Since M-step Lyapunov functions with $M > 1$ are useful for increasing the estimated region of stability, we need a method to make multiple-step ahead predictions with our GP-SSM. In this subsection we compare three different methods that allow us to make M-step ahead predictions.
Firstly, there exists the *direct method*, which relies on training the GP-SSM to directly make M-step ahead predictions. Since the sampling approach uses Lyapunov functions with $i \in \{1, \ldots, M\}$, we need to train M independent GP-SSMs, i.e. one for making 1-step ahead predictions, a different one for making 2-step ahead predictions and so on. Hence, this method becomes computationally demanding.

Furthermore, the larger M, the more training data is needed for achieving good predictive performance [GRMS02].
Secondly, there exists the *iterative method*, which interprets the GP deterministically for the first $M-1$ steps and then calculates the variance of the last step. To obtain the M-step ahead prediction $\boldsymbol{\mu}(\boldsymbol{x}_{k+M})$ from our GP-SSM, we iteratively insert $\boldsymbol{\mu}(\boldsymbol{x}_{k+i})$ into the equation for the predictive mean and obtain $\boldsymbol{\mu}(\boldsymbol{x}_{k+i+1})$ for $i \in \{0, \dots, M-1\}$. $\mathbf{var}(\boldsymbol{x}_{k+M})$ is obtained by inserting $\boldsymbol{\mu}(\boldsymbol{x}_{k+M-1})$ into the equation for the predictive covariance. This method is simple to implement and use, but has the disadvantage that it does not propagate the uncertainty.
Thirdly, there exists the method introduced in [GRMS02], which propagates the prediction uncertainty. As with the *iterative method*, we only have to train a single GP-SSM that can then be used to make multiple-step ahead predictions. While it provides more accurate M-step predictions of the mean and variance than the iterative method, the predictive equations rely on integrals that are generally not analytically tractable. Hence, they have to be approximated using numerical methods and are consequently not suited for providing stability guarantees.
Under these considerations, the *iterative method* is used in our definition of the property function.

3.1.2 Property Function

To verify $\mathcal{KL}$-stability of (3.1) on $\overline{\mathcal{S}\backslash\mathcal{L}}$ we define the property function F as

$$\begin{aligned}
F(\boldsymbol{x}) &= \mathbb{E}[V(\boldsymbol{x}_{k+M}) - \rho(V(\boldsymbol{x}_k))] && (3.3)\\
&= \mathbb{E}[V(\boldsymbol{x}_{k+M})] - \mathbb{E}[\rho(V(\boldsymbol{x}_k))] && (3.4)\\
&= \mathbb{E}[V(\boldsymbol{x}_{k+M})] - \rho(V(\boldsymbol{x}_k)) && (3.5)\\
&= \mathbb{E}[\boldsymbol{x}_{k+M}^T P \boldsymbol{x}_{k+M}] - \rho(\boldsymbol{x}_k^T P \boldsymbol{x}_k) && (3.6)\\
&= \boldsymbol{\mu}^T(\boldsymbol{x}_{k+M}) P \boldsymbol{\mu}(\boldsymbol{x}_{k+M}) + \boldsymbol{\sigma}^T(\boldsymbol{x}_{k+M}) P \boldsymbol{\sigma}(\boldsymbol{x}_{k+M}) - \rho(\boldsymbol{x}_k^T P \boldsymbol{x}_k), && (3.7)
\end{aligned}$$

where V is a continuous, two times differentiable FSLF that satisfies (2.25), $\rho \in \mathcal{K}$ and $\rho < id$. The notion of *stochastic stability in probability* is implicit in this definition of the property function.
(3.3) is the decrease condition of the Lyapunov function from Theorem 2.10, modified such that it can be used in a sampling-based setting. (3.4) uses the linearity of the expected value and (3.5) exploits the fact that the expression $\rho(V(\boldsymbol{x}_k))$ is deterministic. The definition of our Lyapunov function $V(\boldsymbol{x}) = \boldsymbol{x}^T P \boldsymbol{x}$ is inserted to obtain (3.6). In (3.7) we use the equality $\mathbb{E}[X^2] = \mu^2 + \sigma^2$, which holds for any Gaussian distributed random variable X with mean μ and variance σ^2.
To obtain the M-step ahead predictions $\boldsymbol{\mu}(\boldsymbol{x}_{k+M})$ and $\mathbf{var}(\boldsymbol{x}_{k+M})$ from our Gaussian process, we use the *iterative method* from the previous subsection. Finally, $\boldsymbol{\sigma}(\boldsymbol{x}_{k+M})$ is the element-wise square root of $\mathbf{var}(\boldsymbol{x}_{k+M})$.

The derivation is analogous for *stochastic asymptotic stability in probability*, where Theorem 2.11 is used instead of Theorem 2.10 to obtain (3.3).

3.1.3 Verification Theorem

To verify the property function F, we need a theorem that allows us to generalize from a finite number of sampling points to the rest of the search space. Assume that the following assumption holds:

Assumption 3.2 (see [BL16a]). Assume that $F : \mathbb{R}^n \to \mathbb{R}$ is continuous and at least 2 times differentiable on $\mathcal{B}_{\delta_{x_s}}(\boldsymbol{x}_s)$ for all $\boldsymbol{x}_s \in \mathcal{S}_s$. Moreover, $\mathcal{B}_{\delta_{x_s}}(\boldsymbol{x}_s)$ is a convex set for all $\boldsymbol{x}_s \in \mathcal{S}_s$.

Now the following theorem allows us to use δ-sampling to verify the decrease condition of the Lyapunov function in the search space $\mathcal{S}$:

Theorem 3.3 (see [BL16a]). *Let $\mathcal{S}_s$ be a sampling of the compact set $\mathcal{S}$. Suppose Assumption 3.2 holds. If for all $\boldsymbol{x}_s \in \mathcal{S}_s$ it holds that*

$$F(\boldsymbol{x}_s) < -a_{x_s}\delta_{x_s}, \tag{3.8}$$

then $F(\boldsymbol{x}) < 0$ holds for all $\boldsymbol{x} \in \mathcal{S}$.

If we were able to look at an infinite number of points, we would only have to verify that $F(\boldsymbol{x}_s) < 0$. But since this is not feasible, we choose a sampling $\mathcal{S}_s$ of $\mathcal{S}$ and account for nonlinearities of the system via the term $-a_{x_s}\delta_{x_s}$ in (3.8), i.e. we require $F(\boldsymbol{x}_s)$ to be smaller than a negative number instead of smaller than 0. Here, δ_{x_s} is the sampling resolution and a_{x_s} is the Lipschitz constant of F. Intuitively, a_{x_s} is an upper bound for how different the dynamics for some $\boldsymbol{x} \in \mathcal{B}_{\delta_{x_s}}(\boldsymbol{x}_s)$ can be compared to the dynamics in the sampling point $\boldsymbol{x}_s$.
Next, we look at how a good estimate of a_{x_s} can be found.

Computation of the Lipschitz Constant a_{x_s}

One method of calculating a_{x_s} proposed in [BL16a] is

$$a_{x_s} = \left\| \nabla F(\boldsymbol{x}_s) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_s)^T \nabla^2 F(\boldsymbol{x}_s + \xi(\boldsymbol{x} - \boldsymbol{x}_s)) \right\|, \tag{3.9}$$

where $\nabla F(\boldsymbol{x})$ denotes the gradient, $\nabla^2 F(\boldsymbol{x})$ denotes the Hessian matrix, $\boldsymbol{x} \in \mathcal{B}_{\delta_{x_s}}(\boldsymbol{x}_s)$, and $\xi \in (0, 1)$. In (3.9) any operation involving intervals is calculated via interval arithmetics. This method works well for simple systems, but does not scale well with the number of operations in F. Even for modest problem sizes the results become conservative since (3.9) relies on naive interval analysis, i.e. simply replacing every operation by its corresponding interval operation, a method known to lead to inaccurate results [Rum10].

A solution to this problem is based on finding an expression for the Lipschitz constant a_{x_s} that can be calculated using optimization techniques instead of relying on naive analysis:

$$\begin{aligned} a_{x_s} &= \max_{\boldsymbol{x} \in \mathcal{B}_{\delta_{x_s}}(\boldsymbol{x}_s)} \|\nabla F(\boldsymbol{x})\|_\infty && (3.10) \\ &= \max_{\boldsymbol{x} \in \mathcal{B}_{\delta_{x_s}}(\boldsymbol{x}_s)} \left(\max\left\{|\nabla_1 F(\boldsymbol{x})|, \dots, |\nabla_n F(\boldsymbol{x})|\right\}\right) && (3.11) \\ &= \max\left\{ \max_{\boldsymbol{x} \in \mathcal{B}_{\delta_{x_s}}(\boldsymbol{x}_s)} |\nabla_1 F(\boldsymbol{x})|, \dots, \max_{\boldsymbol{x} \in \mathcal{B}_{\delta_{x_s}}(\boldsymbol{x}_s)} |\nabla_n F(\boldsymbol{x})| \right\}, && (3.12) \end{aligned}$$

where $\nabla_i F(\boldsymbol{x})$ denotes the i-th element of the gradient vector $\nabla F(\boldsymbol{x})$.
(3.10) is the definition of a Lipschitz constant. The definition of the infinity norm is inserted in (3.11). It is intuitive that we can change the order of the max operators in (3.12) without changing the result because it does not matter whether we first take the maximum element of a vector and then maximize it over a range of x-values, or if we first maximize each element of the vector and then extract the maximum element.
Now, the terms $\max_{\boldsymbol{x} \in \mathcal{B}_{\delta_{x_s}}(\boldsymbol{x}_s)} |\nabla_i F(\boldsymbol{x})|$ with $i \in \{1, \dots, n\}$ can be calculated using optimization techniques that find the global optimum of a function in a box. For the stochastic systems of interest to this thesis, this approach to calculating a_{x_s} leads to a less conservative Lipschitz constant than with (3.9).

3.1.4 Procedure

In this subsection we summarize how the property function defined in Subsection 3.1.2 and the verification theorem stated in Subsection 3.1.3 can be used to find a set $\mathcal{A} \subseteq \mathcal{S}$ for which $F(\boldsymbol{x}) < 0$.

1. Select an initial coarse sampling $\mathcal{S}_s \subset \mathcal{S}$, and an initial value for the number of steps of the FSLF M.

2. Try to verify $F(\boldsymbol{x}_s) < -a_{x_s}\delta_{x_s}$ for all $\boldsymbol{x}_s \in \mathcal{A}_s$. If the inequality cannot be verified with the initial value of δ, refine $\mathcal{B}_{\delta_{x_s}}(\boldsymbol{x}_s)$ until $F(\boldsymbol{x}_s) < -a_{x_s}\delta_{x_s}$ is verified or a maximum sampling resolution is reached.

3. Increase M for the samples that could not be verified in Step 2 until $F(\boldsymbol{x}_s) < -a_{x_s}\delta_{x_s}$ is verified or a maximum M is reached. In this step, we start directly with the samples obtained at the end of Step 2, i.e. samples refined to the highest possible sampling resolution.

4. $\mathcal{A}$ is the union of the balls $\mathcal{B}_{\delta_{x_s}}(\boldsymbol{x}_s)$ around the samples $\boldsymbol{x}_s$ for which $F(\boldsymbol{x}_s) < -a_{x_s}\delta_{x_s}$ could be verified in Step 2 or Step 3.

Details on how this method can be implemented are found in Chapter 4.

3.2 Verification of Stability in the Neighborhood of the Origin

Since $F(0) = 0$, the inequality $F(\boldsymbol{x}_s) < -a_{x_s}\delta_{x_s}$ cannot be satisfied for sampling points close to the origin. Hence, we need a different method of proving $\mathcal{KL}$-stability here, summarized in the following steps:

1. Linearize system (3.1) by calculating the Jacobian matrix, i.e. the partial derivatives of $\boldsymbol{\mu}(\boldsymbol{x}_{k+1})$ with respect to $\boldsymbol{x}_k$, and evaluating it at the origin:

$$A = \left.\begin{pmatrix} \frac{\partial \mu_1(\boldsymbol{x}_{k+1})}{\partial x_{1,k}} & \cdots & \frac{\partial \mu_1(\boldsymbol{x}_{k+1})}{\partial x_{n,k}} \\ \vdots & & \vdots \\ \frac{\partial \mu_n(\boldsymbol{x}_{k+1})}{\partial x_{1,k}} & \cdots & \frac{\partial \mu_n(\boldsymbol{x}_{k+1})}{\partial x_{n,k}} \end{pmatrix}\right|_{\boldsymbol{x}=0} \tag{3.13}$$

   Now the linearized system is given by

$$\boldsymbol{x}^+ = A\boldsymbol{x}. \tag{3.14}$$

2. Use $V_L(\boldsymbol{x}) = \boldsymbol{x}^T P_L \boldsymbol{x}$ as the Lyapunov function for the linear system, where P_L can be found by solving the discrete Lyapunov equation.

3. Show that $\mathbb{E}[V_L(\boldsymbol{x}_{k+1}) - V_L(\boldsymbol{x}_k)] < 0$ for all $\boldsymbol{x} \in \mathcal{N}(0)$, where $\mathcal{N}(0)$ is a neighborhood around the origin. This verifies that V_L is a Lyapunov function also for the nonlinear system.

4. Compute $\mathcal{L}$ as the maximum level set of V_L inside $\mathcal{N}(0)$, where the calculation of $\mathcal{L}$ is similar to the method used for the computation of $\mathbb{W}$ in Section 3.3.

3.3 Level Set Calculation

This section describes how the set $\mathbb{W}$, which is the final result of the DOA, is found. According to Proposition 2.17, the Lyapunov function of deterministic systems is obtained by summing up M FSLFs:

$$W(\boldsymbol{x}_k) = \sum_{j=0}^{M-1} V(\boldsymbol{x}_{k+j}) \tag{3.15}$$

To account for the stochastic nature of the GP-SSM, we take the expected value of $V(\boldsymbol{x}_{k+j})$ in (3.15) and obtain

$$W(\boldsymbol{x}_k) = \sum_{j=0}^{M-1} \mathbb{E}[V(\boldsymbol{x}_{k+j})]. \tag{3.16}$$

$\mathbb{E}[V(\boldsymbol{x}_{k+j})]$ is calculated as

$$\mathbb{E}[V(\boldsymbol{x}_{k+j})] = \boldsymbol{\mu}^T(\boldsymbol{x}_{k+j})P\boldsymbol{\mu}(\boldsymbol{x}_{k+j}) + \boldsymbol{\sigma}^T(\boldsymbol{x}_{k+j})P\boldsymbol{\sigma}(\boldsymbol{x}_{k+j}), \tag{3.17}$$

following the same derivation as in Section 3.1. Since the expression in (3.17) satisfies the conditions in Proposition 2.16, we have found a valid Lyapunov function of our system.
With W defined as above, the final result of the DOA is obtained by computing the maximum level set $\mathbb{W} := \{\boldsymbol{x}|W(\boldsymbol{x}) \leq L\}$ inside $\mathcal{A} \cup \mathcal{L}$. The following method of calculating the level set in a sampling-based context is proposed in [BL16b]:
First, estimates of two different level sets are found:

1. $\overline{L}_1$ is an estimate of the largest level set of W that does not intersect the balls $\mathcal{B}_{\delta_{x_s}}(\boldsymbol{x}_s)$ around any point for which the decrease condition could not be verified.

2. $\overline{L}_2$ is an estimate of the largest level set of W which is bounded by $\partial\mathcal{A} \cap \partial\mathcal{S}$.

Then, $\overline{L} = \min\{\overline{L}_1, \overline{L}_2\}$ is an estimate for the largest level set of W inside $\mathcal{A} \cup \mathcal{L}$.
$\overline{L}_1$ and $\overline{L}_2$ are calculated as

$$\overline{L}_1 = \min_{\boldsymbol{x}_s \in \mathcal{S}_{su}} \overline{L}_{x_s} \tag{3.18}$$

$$\overline{L}_2 = \min_{\boldsymbol{x}_s \in \partial\mathcal{S}} \overline{L}_{x_s}, \tag{3.19}$$

where $\mathcal{S}_{su}$ is the set of points for for which it holds that (1) Lyapunov's inequality could not be verified (2) $\mathcal{B}_{\delta_{x_s}}(\boldsymbol{x}_s) \cap \mathcal{A} \neq \emptyset$ and (3) $\boldsymbol{x}_s \notin \mathcal{L}$. $\partial\mathcal{S}$ is the set of points for which Lyapunov's inequality could be verified and which are at the border of the search space, refined with sampling resolution δ_{min}.
$\overline{L}_{x_s}$ in (3.18) and (3.19) is calculated in each sampling point $\boldsymbol{x}_s$ as

$$\overline{L}_{x_s} = W(\boldsymbol{x}_s) - a_{x_s}\delta_{x_s}, \tag{3.20}$$

where Theorem 3.3 is used to allow for the sampling-based calculation, and a_{x_s} is computed as in Section 3.1.
We refer the interested reader to [BL16b] for the derivation of (3.18), (3.19), and (3.20).

Chapter 4

Implementation

In this chapter we first give a more detailed explanation of how the main algorithm for verifying Lyapunov's inequality is implemented. Then, we consider an approximation of a_{x_s} that reduces the computational complexity of the algorithm. Since this approximative method does not provide stability guarantees, it cannot be used in safety critical applications.

4.1 Algorithm

We use Algorithm 1 (page 24) to obtain the vectors *good* and *wrong* which contain the samples for which Lyapunov's inequality is verified and not verified, respectively. Algorithm 1 starts with an initial coarse sampling of the search space and calls Algorithm 2 to verify that $F(\boldsymbol{x}_s) < -a_{x_s}\delta_{x_s}$ in each sampling point. If Algorithm 2 fails to prove the inequality with the initial values of δ and M, then it calls itself recursively with a higher sampling resolution and increasing values of M. Algorithm 2 terminates when it is either able to verify $F(\boldsymbol{x}_s) < -a_{x_s}\delta_{x_s}$, or reaches a minimum value of δ and maximum value of M. The Lipschitz constant a_{x_s} is calculated according to (3.12), where the terms $\max_{\boldsymbol{x}\in\mathcal{B}_{\delta_{x_s}}(\boldsymbol{x}_s)} |\nabla_i F(\boldsymbol{x})|$ with $i \in \{1,\dots,n\}$ can be calculated using interval analysis. We use INTLAB [Rum99] to perform efficient interval analysis in Matlab. Apart from extending basic arithmetic functions to intervals, INTLAB also provides optimization techniques that avoid the problems of naive interval analysis mentioned in Section 3.1. The function `verifyglobalmin` provided by INTLAB is used to find the maximum of $|\nabla_i F(\boldsymbol{x})|$ in the calculation of a_{x_s}.
Since the verification of Lyapunov's inequality is decentralized, Algorithm 1 can easily be parallelized by replacing the first `for`-loop with a `parfor`-loop in Matlab.

Algorithm 1 Sampling-based verification of Lyapunov's inequality

```
Input: $\mathcal{S}, \delta_0, M_0, \delta_{min}, M_{max}, G, V$
Output: $\mathcal{A}, good, wrong$

$res \leftarrow []$; $good \leftarrow []$; $wrong \leftarrow []$; $\mathcal{A} \leftarrow \emptyset$
Select coarse sampling $\mathcal{A}_s \subset \mathcal{S}$ with resolution $\delta_0$
for all $x_s \in \mathcal{A}_s$ do
    $res \leftarrow [res;$ VERIFY_SAMPLE$(x_s, \delta_0, M_0, \delta_{min}, M_{max}, G, V)]$

for all i=1:length(res) do
    if $res(i).decreasing == True$ then
        $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{B}_{\delta_{x_s}}(res(i).point)$
        $good \leftarrow [good; res(i)]$
    else
        $wrong \leftarrow [wrong; res(i)]$
```

Algorithm 2 Recursively verify $F(\boldsymbol{x}_s) < -a_{x_s}\delta_{x_s}$

```
Input: $x_s, \delta_{x_s}, M, \delta_{min}, M_{max}, G, V$
Output: $res$

function VERIFY_SAMPLE$(x_s, \delta_{x_s}, M, \delta_{min}, M_{max}, G, V)$
    $F(x) \leftarrow V(G^M(x)) - \rho(V(x))$
    $res \leftarrow []$
    if $F(x_s) < -a_{x_s}\delta_{x_s}$ then
        $res(1).decreasing \leftarrow True$
        $res(1).point \leftarrow x_s$
        $res(1).del \leftarrow \delta_{x_s}$
    else if $\delta_{x_s} \geq 2\delta_{min}$ then
        $\delta_{x_s} \leftarrow \delta_{x_s}/2$
        Generate refinement $\mathcal{B}^S_{\delta_{x_s}}$ of $x_s$
        for all $x_s^* \in \mathcal{B}^S_{\delta_{x_s}}$ do
            $res \leftarrow [res;$ VERIFY_SAMPLE$(x_s^*, \delta_{x_s}, M, \delta_{min}, M_{max}, G, V)]$
    else if $M < M_{max}$ then
        $M \leftarrow M + 1$
        $res \leftarrow$ VERIFY_SAMPLE$(x_s, \delta_{x_s}, M, \delta_{min}, M_{max}, G, V)$
    else
        $res(1).decreasing \leftarrow False$
        $res(1).point \leftarrow x_s$
        $res(1).del \leftarrow \delta_{x_s}$
    return $res$
```

4.2 Approximating a_{x_s}

Calculating the Lipschitz constant a_{x_s} is computationally demanding. To increase the usefulness of the algorithm in practice, we introduce an approximation $\tilde{a}_{x_s}$ of a_{x_s} which evaluates the gradient of the property function in a given sampling point instead of maximizing it over a range of values as in (3.10):

$$\tilde{a}_{x_s} = \|\nabla F(\boldsymbol{x}_s)\|_\infty \tag{4.1}$$

The expression in (4.1) is a first order approximation of $F(\boldsymbol{x})$ that ignores higher order terms. Since $\tilde{a}_{x_s}$ is an under-approximation of a_{x_s}, we loose stability guarantees. $F(\boldsymbol{x}_s) < -\tilde{a}_{x_s}\delta_{x_s}$ is verified in more samples and thus the estimated DOA might be larger than the actual DOA of the system.
Despite loosing stability guarantees, it often makes sense to use the approximation before running the algorithm with the more rigorous method. We can gain an intuition for our system and find a good Lyapunov function candidate at low computational cost. Furthermore, it allows us to quickly find good parameters $\mathcal{S}, \delta_0, \delta_{min}$ and M_{max} of Algorithm 1.

Chapter 5

Evaluation

In this chapter the sampling approach is evaluated on 1D and 2D systems. We first compare the region of stability for different stability criteria in Section 5.1, then compare the effect of choosing different values for M_{max} in Section 5.2, and finally evaluate the performance of the approximation of a_{x_s} in Section 5.3.

The process of training the GP-SSM is identical for all examples. First, a search space $\mathcal{S}$ is chosen. We then randomly select 7 training examples from $\mathcal{S}$ and obtain the noisy training targets

$$\boldsymbol{y} = \boldsymbol{G}(\boldsymbol{x}) + \mathcal{N}(0, \sigma_n^2 I) \tag{5.1}$$

by adding Gaussian noise with zero mean and standard deviation $\sigma_n = 0.01$ to the output of the system. Finally, this data is used to train a GP-SSM, where the squared exponential covariance function is used with the hyperparameters $\sigma_f^2 = 0.49$ and $\lambda = 0.7$.

5.1 Different Stochastic Stability Criteria

In this section we compare the results of the sampling approach for different stochastic stability criteria. For this purpose, we train the GP-SSM with the following nonlinear system:

$$x_{k+1} = G(x_k) = 0.5x_k + 0.3x_k^2 \tag{5.2}$$

in the search space $\mathcal{S} := \{-3 \leq x \leq 3\}$. Figure 5.1 shows a plot of the training examples together with the predicted mean and variance of the Gaussian process. Algorithm 1 is run with $\delta_0 = 0.1$, $\delta_{min} = 0.02$, $M_0 = 1$, $M_{max} = 2$, and the Lyapunov function $V(x) = x^2$. We are not able to verify the stability of the system in the neighborhood of the origin via linearization because we are not able to show that $V_L(\boldsymbol{x})$ is decreasing for all $\boldsymbol{x} \in \mathcal{N}(0)$ with $\mathcal{N}(0) = \{-0.2 \leq x \leq 0.2\}$. But let us assume that we know that the system is stable in $\mathcal{N}(0)$.

Stochastic stability in probability To find a region in which the system is *stochastically stable in probability* we verify $\mathbb{E}[\Delta V(x(t))] < 0$ and obtain the result $\mathbb{W} := \{x | W(x) \leq \overline{L}\}$ with $\overline{L} = 3.4$, where W is calculated as described in Section 3.3.

Stochastic asymptotic stability in probability To find a region in which the system is *stochastically asymptotically stable in probability* we verify $\mathbb{E}[\Delta V(x(t))] < \mathbb{E}[\varphi(|x(t)|)]$ with $\varphi(x) = 0.15x^2$ and obtain the result $\mathbb{W} := \{x | W(x) \leq \overline{L}\}$ with $\overline{L} = 3.0$.

The results can be seen in Figure 5.2. We can observe that the estimated region of stability is slightly larger for stochastic stability in probability compared to the case where we prove stochastic asymptotic stability in probability. As expected, the algorithm is not able to verify the decrease condition of the Lyapunov function in the neighborhood of the origin for either of the criteria.

5.2 Different M_{max}

In this section we compare the results of running the sampling algorithm with different values of M_{max}. The definition of *stochastic stability in probability* is used again.

We train the GP-SSM with the following nonlinear system:

$$\boldsymbol{x}_{k+1} = \boldsymbol{G}(\boldsymbol{x}_k) = \begin{bmatrix} 2x_{1,k}^2 \\ 0.5x_{1,k} + 0.9x_{2,k}^3 \end{bmatrix} \tag{5.3}$$

in the search space $\mathcal{S} := \{\boldsymbol{x} | |x_1| \leq 1, |x_2| \leq 1.3\}$. We run Algorithm 1 with $\delta_0 = 0.1$, $\delta_{min} = 0.02$, $M_0 = 1$, and the Lyapunov function $V(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{x}$ to obtain the results in Figure 5.3a and Figure 5.3b for $M_{max} = 1$ and $M_{max} = 2$, respectively. We can observe that there are many samples for which the inequality $F(\boldsymbol{x}_s) < -a_{x_s}\delta_{x_s}$ was verified with $M_{max} = 2$ but not with $M_{max} = 1$. As expected, the sampling-based algorithm is not able to verify the decrease condition of the Lyapunov function in the neighborhood of the origin. We are also not able to verify stability of the system in the neighborhood of the origin via linearization. But if we assume that the system is stable in $\mathcal{N}(0) = \{\boldsymbol{x} | |x_1| \leq 0.1, |x_2| \leq 0.1\}$, then we obtain the final result $\mathbb{W} := \{\boldsymbol{x} | W(\boldsymbol{x}) \leq \overline{L}\}$ with $\overline{L} = 0.16$ and $\overline{L} = 0.25$ for $M_{max} = 1$ and $M_{max} = 2$, respectively. The larger level set for $M_{max} = 2$ shows the power of using finite-step Lyapunov functions with $M > 1$.

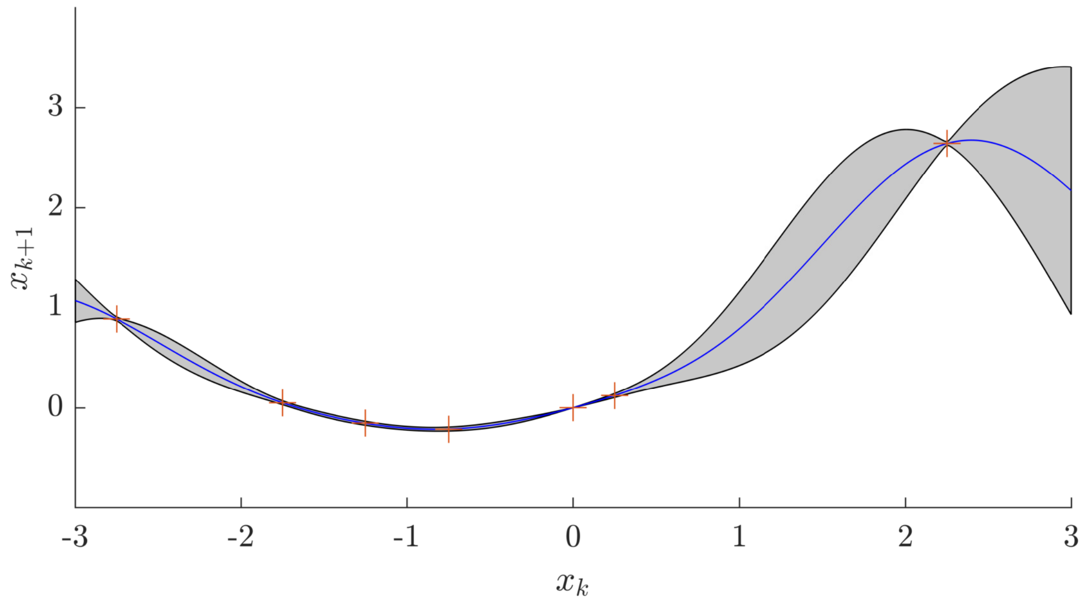
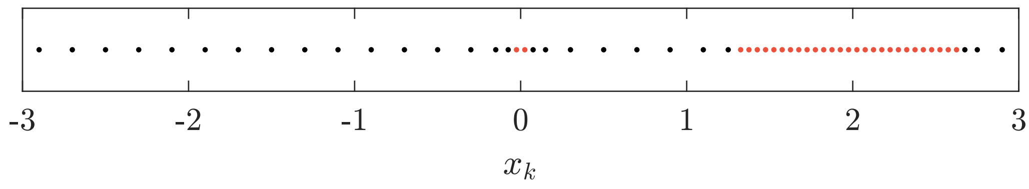
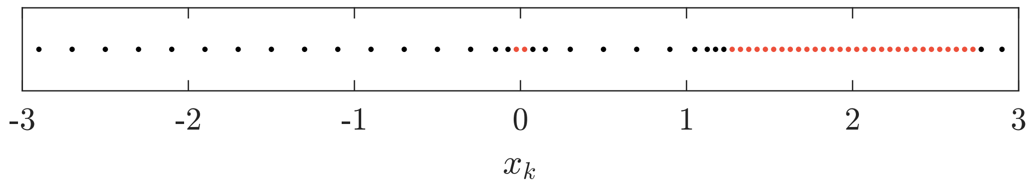


Figure 5.1: Plot of Gaussian Process trained on the 1D-system $x_{k+1} = 0.5x_k + 0.3x_k^2$ from Section 5.1. Predictive mean function in blue, training examples in orange, and twice the standard deviation as the shaded region.

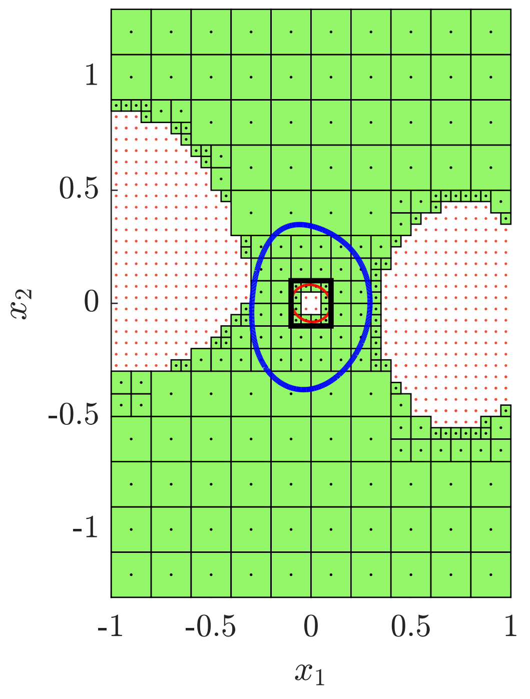


(a) stochastic stability in probability

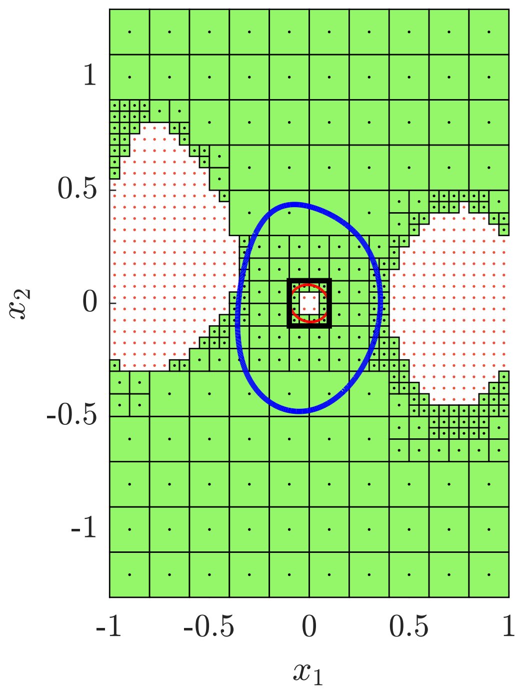


(b) stochastic asymptotic stability in probability

Figure 5.2: Results of Algrotihm 1 for the 1D-system from Section 5.1. Samples for which Lyapunov's inequality could be verified and not verified are shown in black and red, respectively. The estimated region of stability is slightly larger for stochastic stability in probability compared to the case where we prove stochastic asymptotic stability in probability. As expected, the algorithm is not able to verify the decrease condition of the Lyapunov function in the neighborhood of the origin for either of the criteria.



(a) DOA for $M_{max} = 1$

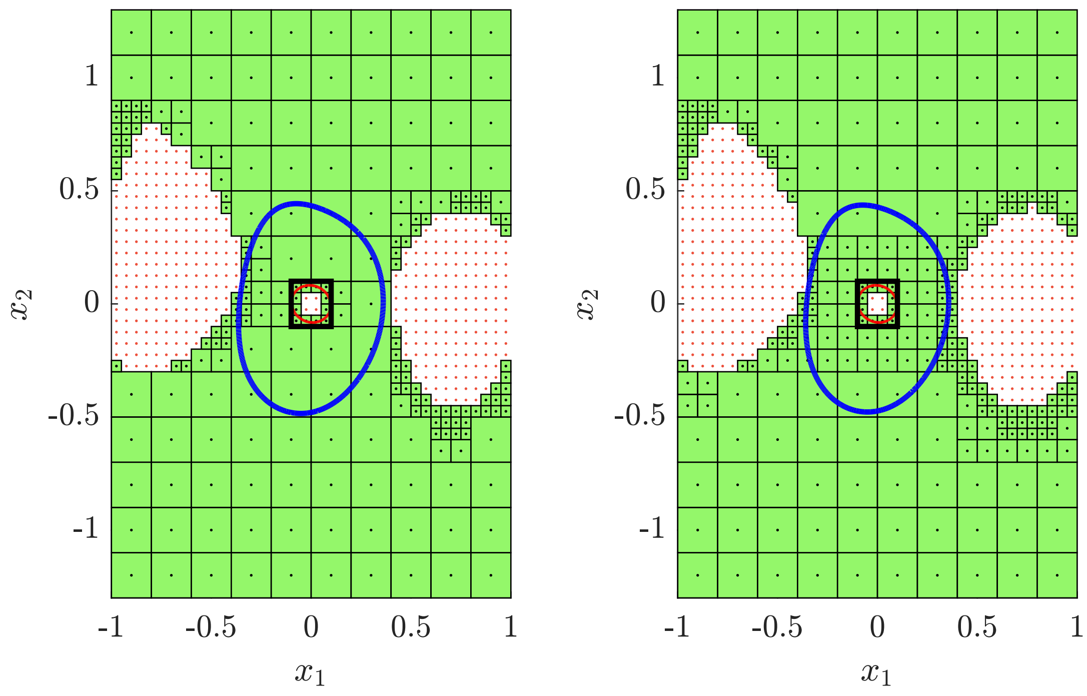


(b) DOA for $M_{max} = 2$

Figure 5.3: Results for the 2D-system from Section 5.2. Samples for which Lyapunov's inequality could be verified are depicted as black dots and their respective balls $\mathcal{B}_{\delta_{x_s}}(\boldsymbol{x}_s)$ as rectangles around them. Samples for which Lyapunov's inequality could not be verified are shown as red dots. $\mathcal{A}$ is depicted as the green area. The maximum level set of W is shown as a blue line, the neighborhood $\mathcal{N}(0)$ as a rectangle around 0, and the maximum level set of V_L inside $\mathcal{N}(0)$ as a red line. We can observe that there are many samples for which the inequality $F(\boldsymbol{x}_s) < -a_{x_s}\delta_{x_s}$ was verified with $M_{max} = 2$ but not with $M_{max} = 1$. Hence, the maximum level set of W, i.e. the estimated DOA of the origin, is larger in Panel (b). We can also observe the use of multi-resolution sampling, i.e. a fine resolution is used toward the boundary of the DOA compared to a coarse resolution in areas where Lyapunov's inequality is easily verified.

5.3 Approximated a_{x_s}

In this section we use the same system, training examples, and parameters as in the previous section. $M_{max} = 2$ is used again together with the definition of *stochastic stability in probability*. But instead of verifying Lyapunov's inequality with the rigorous method from Section 3.1, we use the approximative method from Section 4.2. The results can be seen in Figure 5.4. We can observe that while the rigorous method needs a higher sampling resolution to verify Lyapunov's inequality in some areas, the final estimate of the DOA differs only slightly. We obtain a level set of $\overline{L} = 0.25$ with the rigorous method compared to a larger level set of $\overline{L} = 0.26$ with the approximative method. This is due to the fact that $\tilde{a}_{x_s}$ underapproximates a_{x_s} and hence the inequality $F(\boldsymbol{x}_s) < -\tilde{a}_{x_s}\delta_{x_s}$ is verified in more samples. This shows that the approximation can be good to get an estimate of the real DOA at low computational cost, but should not be used in safety critical applications.



(a) DOA with approximation of a_{x_s} (b) DOA with rigorous calculation of a_{x_s}

Figure 5.4: Samples for which Lyapunov's inequality could be verified are depicted as black dots and their respective balls $\mathcal{B}_{\delta_{x_s}}(\boldsymbol{x}_s)$ as rectangles around them. Samples for which Lyapunov's inequality could not be verified are shown as red dots. $\mathcal{A}$ is depicted as the green area and the maximum level set of W is shown as a blue line. We can see that the rigorous method needs a higher sampling resolution for verifying $F(\boldsymbol{x}_s) < -a_{x_s}\delta_{x_s}$ than the approximative method toward the boundary of the region of stability. But the maximum level set of W, i.e. the estimated DOA of the origin, is only slightly larger in Panel (a).

Chapter 6

Conclusion

In this thesis, the sampling-based approach to finding Lyapunov functions proposed in [BL16a] was extended to the verification of stability of Gaussian Process State Space Models. To allow for finite-step Lyapunov functions with $M > 1$, we used a method that interprets the GP-SSM deterministically for the first $M - 1$ steps and then takes the stochastic nature of the GP-SSM into account in the last step.
Furthermore, a different method of calculating the Lipschitz constant a_{x_s} was derived to obtain less conservative estimates of the DOA. We also approximated this rigorous method to reduce the computational complexity of the algorithm. The disadvantage of the approximation is that it does not provide stability guarantees, and thus cannot be used in safety critical applications.
The algorithm was implemented in Matlab and evaluated on 1D and 2D examples. We observed the power of using multiple-step ahead predictions and compared the results for different stochastic stability criteria. Moreover, the DOA estimate of the rigorous method was compared to the DOA estimate of the approximative method.
One limitation of the approach is that verification of stability in the neighborhood of the origin currently does not work. We assume that this might be due to the fact that the variance of the GP is not small enough around the origin, but further research is needed to here.
In future work it would be interesting to investigate whether stability guarantees can be provided by less computationally expensive methods of calculating a_{x_s}.

Appendix A

Matlab Code

A.1 Main Function and Setting Parameters

```
function [result] = verify()
% VERIFY find region of stability of GP—SSM
% NOTE: main function from which all other functions are called

% initialize variables
init_vals;

% generate training data for GP—SSM
[m, X, Y] = gen_training_data(n, search_space, delta_train, G, ...
    sigma_n);

% train GP—SSM
[mu, sigma] = train_GP(n, m, x, X, Y, hyp, sigma_n);

% linearize system around the origin
[P_L, pos_definite] = linearize_system(n, x, mu, x_0);

% prove stability of the system in the neighborhood of the origin
hood_is_stable = verify_hood(hood, x, mu, sigma, P_L, pos_definite);

% construct Lyapunov function of linearized system
[V_L_fun, V_grad] = create_V_L(P_L, x);

% find level set in the neighborhood of the origin
[hood_level] = hood_level_set(n, hood, V_L_fun, V_grad, x, ...
    pos_definite);

% calculate property function and its gradient
```

```
[G] = create_G_M(M_max, x, mu);
[F, F_fun] = cal_F(sigma, G, x, M_max, rho, V);
[grad] = grad_sym(M_max, x, F);

% construct Lyapunov function
[W] = create_W(x, G, sigma, M_max, V);

% verify decrease condition of the Lyapunov function
[stable_region, instable_region] = delta_sampling(n, ...
    search_space, delta_0, delta_min, M_0, M_max, F_fun, grad);

% calculate maximum viable level set of W
level_set = find_level_set(stable_region(:,1:end-1), ...
    instable_region(:,1:end-1), W, delta_min, x, search_space, ...
    hood, V_L_fun, hood_level);

% save results
result = struct('stable_region',stable_region, ...
    'instable_region', instable_region, 'level_set',level_set, ...
    'W',W,'m', m,'X', X,'Y', Y, 'mu',mu,'sigma', sigma, ...
    'hood_is_stable',hood_is_stable,'n', n, ...
    'search_space', search_space, 'hood',hood, 'x',x, ...
    'V_L_fun',V_L_fun,'hood_level',hood_level);
end
```

```
% initialize parameters
n = 2;
x_0 = zeros(n,1);
x = sym('x', [1 n]).';

% choose the system
G_s = [1.8*x(1)^2; 1.2*x(2)^3 + 0.5*x(1)];
G = matlabFunction(G_s, 'Vars', {x});

% parameters of the sampling algorithm
search_space = [-1, 1; -1.3, 1.3];
hood = repmat([-0.1, 0.1], n, 1);
delta_train = 0.3;
delta_0 = 0.1;
delta_min = 0.02;
M_0 = 1;
M_max = 2;
rho = 0.999;
```

```

% candidate Lyapunov function
P = [1, 0; 0, 1];
V = x.'*P*x;

% hyperparameters of GP
meanfunc = [];
covfunc = @covSEiso;
likfunc = @likGauss;
hyp = struct('mean', [], 'cov', [0 0], 'lik', -1);
sigma_n = 0.01;
hyp.lik = log(sigma_n);
```

A.2 Training of GP-SSM

```
function [m, X, Y] = gen_training_data(n, search_space, ...
    delta_train, G, sigma_n)
% GEN_TRAINING_DATA generate data for training the GP-SSM

% generate samples in the search space
samples = gen_samples(n, search_space, delta_train);

% randomly remove 50% of the training samples
num_samples = size(samples,1);
k = randperm(num_samples);
samples(k(1:floor(num_samples/2)),:) = [];

% add the origin as a training sample
samples = [zeros(1,n); samples];

% initialize X and Y
m = size(samples,1);
X = samples';
Y = zeros(size(samples,1), size(samples,2));

% next step of the system as label
for i=1:size(samples,1)
    Y(i,:) = G(samples(i,:)')';
end

% add noise
```

```
Y = Y + sigma_n*randn(size(Y,1), size(Y,2));
Y(1,:) = G(samples(1,:)')';
end
```

```
function [mu, sigma] = train_GP(n, m, x, X, Y, hyp, sigma_n)
% TRAIN_GP train the GP with X, Y and the given hyperparameters

mu = sym('mu', [n 1]);
sigma = sym('sigma', [n 1]);
for dim=1:n
    y = Y(:,dim);

    % set hyperparameters
    hyp2 = hyp;
    hyp2.cov(1) = log(0.7);
    hyp2.cov(2) = log(0.7);
    ell = exp(hyp2.cov(1));
    sf = exp(hyp2.cov(2));
    P = ell^2 * eye(n);
    P_inv = inv(P);

    % calculate k and K
    K = zeros(m,m);
    k = sym('k', [m 1]);
    for i=1:m
        k(i) = SE_kernel(x, X(:,i), sf, P_inv);
        for j=1:m
            K(i,j) = SE_kernel(X(:,i), X(:,j), sf, P_inv);
        end
    end

    % add noise
    noise = sigma_n^2*eye(size(K,1),size(K,2));
    noise(1,1) = 0; % no noise at the origin
    K_n = K + noise;

    % calculate mu and sigma
    K_inv = inv(K_n);
    mu(dim) = k.'*K_inv*y;
    sigma(dim) = sqrt(SE_kernel(x,x,sf,P_inv) - k.'*K_inv*k);
end
end
```

```
function [k] = SE_kernel(x1, x2, sf, P_inv)
% SE_KERNEL squared exponential kernel of GP, parametrized as in
% gpml toolbox

k = sf^2 * exp(-(x1-x2).'*P_inv*(x1-x2)/2);
end
```

A.3 Sampling-Based Verification of Lyapunov's Inequality

```
function [G] = create_G_M(M_max, x, mu)
% CREATE_G_M construct the M-times map composition of mu

% initializations
G = cell(M_max + 1,1);
G_i = mu;
G{1} = mu;
G{M_max + 1} = x; % save G^0 at the end of the cell array

% M-times map composition
for i=2:M_max
    G_i = subs(G_i, x, mu);
    G{i} = G_i;
end
end
```

```
function [F, F_fun] = cal_F(sigma, mu, x, M_max, rho, V)
% CAL_F calculate F{M} as E[V(x_{k+M}))] - V(x_k)

% initializations
F = cell(M_max, 1);
F_fun = cell(M_max, 1);

% use M-step FSLFs
for i=1:M_max
    V_mu = subs(V, x, mu{i});
    V_sigma = subs(V, x, sigma);
    if i==1
        V_sigma = subs(V_sigma, x, mu{M_max + 1});
    else
```

```
        V_sigma = subs(V_sigma, x, mu{i - 1});
    end

    % calculate F
    F_i = V_mu + V_sigma - rho*V;
    F{i} = F_i;
    F_fun{i} = matlabFunction(F_i, 'Vars', {x});
end
end
```

```
function [grad] = grad_sym(M_max, x, F)
% GRAD_SYM calculate the gradient of the property function F

grad = cell(M_max, 1);
n = size(x,1);
for i=1:M_max
    g = gradient(F{i});
    gradi = cell(n,1);
    for j=1:n
        % add minus so that maximization becomes minimization
        g_j = -abs(g(j));
        gradi{j} = matlabFunction(g_j, 'Vars', {x});
    end
    grad{i} = gradi;
end
end
```

```
function [W] = create_W(x, mu, sigma, M_max, V)
% CREATE_W calculate the Lyapunov function W of the system as the sum
% of M_max finite-step Lyapunov functions

W = 0;
for i = 1:M_max+1
    if i~= M_max
        V_mu = subs(V, x, mu{i});
        V_sigma = subs(V, x, sigma);
        if i==1
            V_sigma = subs(V_sigma, x, mu{M_max + 1});
        elseif i==M_max+1
            V_sigma = 0; % x is deterministic here, so sigma==0
        else
            V_sigma = subs(V_sigma, x, mu{i - 1});
        end
        W = W + V_mu + V_sigma;
    end
end
end
```

```
function [stable_region, instable_region] = delta_sampling (n, ...
    search_space, delta_0, delta_min, M_0, M_max, F_fun, grad)
% DELTA_SAMPLING  use delta-sampling to verify the decrease condition
% of the Lyapunov function

global INTLAB_CONST % trickery to make INTLAB work with parfor-loop

% create grid of samples
samples = gen_samples(n, search_space, delta_0);

% verify decrease condition
region = [];
num_samples = size(samples,1);
parfor i=1:num_samples
    r = verify_sample(samples(i,:), delta_0, M_0, delta_min, ...
        M_max, F_fun, INTLAB_CONST, grad);
    region = [region; r];
end

% parse result
stable_idx = region(:,1) == 1;
instable_idx = ~stable_idx;
stable_region = region(stable_idx, 2:end);
instable_region = region(instable_idx, 2:end);
end
```

```
function [samples] = gen_samples(n, search_space, delta_0)
% GEN_SAMPLES generate grid of samples

X = cell(n,1);
for i=1:n
    X{i} = search_space(i,1)+delta_0:delta_0*2: ...
        search_space(i,2)-delta_0;
end
samples = X{1};
for i=2:n
    samples = combvec(samples, X{i});
end
```

```
samples = samples';
end
```

```
function [region] = verify_sample(point, delta, M, delta_min, ...
    M_max, F, INTLAB_CONST_TEMP, grad)
% VERIFY_SAMPLE recursively prove decrease condition of each sampling
% point

% trickery to make INTLAB work with parfor-loop
global INTLAB_CONST
INTLAB_CONST = INTLAB_CONST_TEMP;

% recursively prove decrease condition
if is_stable(point', delta, M, F, grad)
    region = [1, delta, point,M];
elseif delta >= 2*delta_min
    % apply refinement
    delta = delta / 2;
    refinement = gen_refinement(point, delta);
    region = [];
    for i=1:size(refinement, 1)
        r = verify_sample(refinement(i,:), delta, M, delta_min, ...
            M_max, F, INTLAB_CONST, grad);
        region = [region; r];
    end
elseif M < M_max
    M = M + 1;
    region = verify_sample(point, delta, M, delta_min, M_max, F, ...
        INTLAB_CONST, grad);
else
    region = [0, delta, point, M];
end
end
```

```
function [stable] = is_stable(point, delta, M, F, grad)
% IS_STABLE verify decrease condition of F in a given sampling point

a_xs = cal_a_xs(grad{M}, point, delta);
stable = F{M}(point) < -a_xs*delta;
end
```

```
function [refinement] = gen_refinement(point, delta)
% GEN_REFINEMENT generate new samples at a higher resolution
% surrounding the given sampling point

% initialization
n = size(point, 2);
num_points = 2^n;
points = repmat(point, num_points, 1);

% Trick 17
X = zeros(num_points,n);
for i = 1:num_points
    X(i,:) = de2bi(i-1, n);
end
X(X==0) = -1;
refinement = points + X*delta;
end
```

```
function [a_xs] = cal_a_xs(grad, x_s, delta)
% CAL_A_XS calculate the Lipschitz constant of the property function

% initializations
x = midrad(x_s, delta);
n = size(x_s,1);
gradient = zeros(n,1);

% find the maximum of each element of the gradient
for i=1:n
    g_fun = grad{i};
    try
        [mu,~,~] = verifyglobalmin(g_fun,x);
        gradient(i) = - inf(mu);
    catch
        % g_fun doesn't depend on x, so no optimization is needed
        gradient(i) = - g_fun(0);
    end
end

% extract the maximum element of the gradient
a_xs = max(gradient);
end
```

A.4 Verification of Stability in the Neighborhood of the Origin

```
function [P_L, pos_definite] = linearize_system(n, x, mu, x_0)
% LINEARIZE_SYSTEM linearize the system in the neighborhood
% of the origin

% calculate the jacobian
J = jacobian(mu);
J_0 = J;
J_0 = subs(J_0, x, x_0);

% find Lyapunov function for the linearized system
A = double(J_0);
Q = eye(n);
P_L = dlyap(A,Q);

% check if P_L is positive definite
[~,r] = chol(P_L);
pos_definite = r == 0 && rank(P_L) == size(P_L,1);
end
```

```
function [result] = verify_hood(hood, x, mu, sigma, P_L, ...
    pos_definite)
% VERIFY_HOOD prove that the system is stable in the neighborhood of
% the origin

if pos_definite
    % define Lyapunov function
    V_L = symfun(x.'*P_L*x, x);

    % define function to be verified in the neighborhood
    % of the origin
    fun = -(subs(V_L, x, mu) + subs(V_L, x, sigma) - V_L);
    f = matlabFunction(fun, 'Vars', {x});

    % find global minimum of f
    x_vals = midrad(zeros(size(x,1),1), hood(1,2));
    [global_min,~,~] = verifyglobalmin(f,x_vals);
    f_val = inf(global_min);

    % the system is stable in the neighborhood of the origin if V_dot
    % is smaller than 0 (i.e. the minimum of f is greater than zero)
    result = f_val > 0;
else
    result = false;
end
end
```

```
function [V_L_fun, V_grad] = create_V_L(P_L, x)
% CREATE_V_L construct the Lyapunov function of the linearized system
% and calculate its gradient

V_L = x.' * P_L * x;
V_grad = gradient(V_L);
V_L_fun = matlabFunction(V_L, 'Vars', {x});
end
```

```
function [level_set] = hood_level_set(n, hood, V_fun, V_grad, x, ...
    pos_definite)
% HOOD_LEVEL_SET find the maximum viable level set of the system
% linearized around the origin

if pos_definite
    % generate samples at the border of the hood
    delta_0 = hood(1,2)/100;
    samples = gen_samples(n, hood, delta_0);
    delta = repmat(delta_0, size(samples,1),1);
    samples = [delta, samples];
    hood_border = find_delta_S(samples, hood);

    % modify gradient of Lyapunov function so maximization becomes
    % minimization
    gradi = cell(n,1);
    for j=1:n
        g_j = -abs(V_grad(j));
        gradi{j} = matlabFunction(g_j, 'Vars', {x});
    end

    % calculate the maximum viable level set
    L = zeros(size(hood_border,1),1);
    for i=1:size(hood_border,1)
        % extract x_s and delta
        x_s = hood_border(i,2:end)';
        delta = hood_border(i,1);
        % calculate a_xs
        a_xs = cal_a_xs(gradi, x_s, delta);

        % calculate L_xs
        L(i) = V_fun(x_s) - a_xs * delta;
    end
    level_set = min(L);
else
    level_set = 0;
end
end
```

A.5 Level Set Calculation

```
function [L] = find_level_set(stable_region, instable_region, W, ...
    delta_min, x, search_space, hood, V_L, hood_level)
% FIND_LEVEL_SET  calculate the maximum viable level set of W in the
% union of A and L.

n = size(x,1);

% calculate the gradient of the Lyapunov function W and modify it
% such that global minimizers can be used
grad = gradient(W);
gradi = cell(n,1);
for j=1:n
    g_j = -abs(grad(j));
    gradi{j} = matlabFunction(g_j, 'Vars', {x});
end
W_fun = matlabFunction(W, 'Vars', {x});

% find stable sampling points that are at the border
% of the search space
delta_S = find_delta_S(stable_region, search_space);

% increase sampling resolution of samples at the border
% of the search space
S_refined = refine_S(delta_S, delta_min);

% refinement of border also creates samples that are not at the
% border, so we get rid of them
delta_S = find_delta_S(S_refined, search_space);

% find instable points that are at the border to the
% stable_region
border = find_border(instable_region, stable_region, hood, V_L, ...
    hood_level);

% calculate L_x_s for both delta_S and border
L_border = cal_L_xs(border, W_fun, gradi);
L_delta_S = cal_L_xs(delta_S, W_fun, gradi);

% calculate L_1, L_2 and L
L_1 = min(L_border);
L_2 = min(L_delta_S);

% if one of the two sets is empty, only use the level set of the
% other set
if isempty(L_1)
    L_1 = Inf;
elseif isempty(L_2)
    L_2 = Inf;
end
L = min(L_1, L_2);
end
```

```
function [L_xs] = cal_L_xs(samples, W_fun, gradi)
% CAL_L_XS calculate L_xs (can be used for both delta_S and border)

global INTLAB_CONST % trickery to make INTLAB work with parfor-loop

% calculate L_xs
L_xs = zeros(size(samples,1),1);
parfor i=1:size(samples,1)
    L_xs(i) = cal_L(W_fun, samples(i,:), INTLAB_CONST, gradi);
end
end
```

```
function [L_xs] = cal_L(W_fun, point, INTLAB_CONST_TEMP, gradi)
% CAL_L calculate the maximum viable level set that does not
% intersect the delta-radius ball around a given sample

% trickery to make INTLAB work with parfor-loop
```

```
global INTLAB_CONST
INTLAB_CONST = INTLAB_CONST_TEMP;

% extract x_s and delta
x_s = point(2:end)';
delta = point(1);

% calculate a_xs
a_xs = cal_a_xs(gradi, x_s, delta);

% calculate L_xs
L_xs = W_fun(x_s) - a_xs * delta;
end
```

```
function [S_refined] = refine_S(stable_region, delta_min)
% REFINE_S increase sampling resolution of the given samples to
% delta_min

S_refined = [];
for i=1:size(stable_region,1)
    r = refine_sample(stable_region(i,2:end), stable_region(i,1), ...
        delta_min);
    S_refined = [S_refined; r];
end
S_refined = unique(S_refined,'rows');
end
```

```
function [border] = find_border(instable_region, stable_region, ...
    hood, V_L, hood_level)
% FIND_BORDER find points for which the decrease condition is not
% verified which are also at the border to stable_region

border = [];
for i=1:size(instable_region,1)
    % select samples for which the intersection of the delta-radius
    % ball around the instable sample and stable_region is not empty
    for j=1:size(stable_region,1)
        dist = abs(instable_region(i,2:end) ...
            - stable_region(j,2:end))';
        if dist <= 1.00001*repmat(instable_region(i,1) ...
                + stable_region(j,1), size(dist,1), 1)
            % exclude points in the neighborhood of the origin
            if ~(in_the_hood(instable_region(i,:), hood) ...
                    & in_L(instable_region(i,:), V_L, hood_level))
                border = [border; instable_region(i,:)];
            end
        end
    end
end
border = unique(border,'rows');
end
```

```
function [refinement] = refine_sample(point, delta, delta_min)
% REFINE_SAMPLE recursively refine the sample-point until the maximum
% sampling resolution is reached

if delta >= 2*delta_min
    delta = delta/2;
    ref = gen_refinement(point, delta);
    refinement = [];
    for i=1:size(ref, 1)
        r = refine_sample(ref(i,:), delta, delta_min);
        refinement = [refinement; r];
    end
else
    refinement = [delta, point];
end
end
```

```
function [res] = in_L(point, V_L, hood_level)
% IN_L find out if a given sample is in L

x_s = point(2:end)';
res = V_L(x_s) <= hood_level;
end
```

```
function [valid] = in_S(point, search_space)
% IN_S find out if a given sample is in the search_space

valid = point' <= search_space(:,2) & point' >= search_space(:,1);
valid = sum(valid) == size(point,2);
end
```

```
function [in_hood] = in_the_hood(sample, hood)
% IN_THE_HOOD find out if a given sample is in the neighborhood
% of the origin
```

```
point = sample(2:end)';
in_hood = point <= hood(:,2) & point >= hood(:,1);
in_hood = sum(in_hood) == size(point,1);
end
```

```
function [delta_S] = find_delta_S(S_input, search_space)
% FIND_DELTA_S find sampling points that are at the border of the
% search space

delta_S = [];
for i=1:size(S_input, 1)
    % if any of the neighbors of a given sample is not in the
    % search_space, the sample is at the border of the search_space
    neighbors = gen_refinement(S_input(i,2:end), 1.1*S_input(i,1));
    for j=1:size(neighbors,1)
        if ~in_S(neighbors(j,:), search_space)
            delta_S = [delta_S; S_input(i,:)];
            break
        end
    end
end
end
```

List of Figures

Bibliography

[Arn72] L. Arnold. *Stochastic Differential Equations: Theory and Applications.* John Wiley Sons, 1972.

[BH16] T. Beckers and S. Hirche. Stability of Gaussian Process State Space Models. In *2016 European Control Conference (ECC)*, pages 2275–2281, June 2016. `doi:10.1109/ECC.2016.7810630`.

[BL15] R. Bobiti and M. Lazar. A delta-sampling verification theorem for discrete-time, possibly discontinuous systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, HSCC '15, pages 140–148, 2015. `doi:10.1145/2728606.2728631`.

[BL16a] R. Bobiti and M. Lazar. A sampling approach to finding Lyapunov functions for nonlinear discrete-time systems. In *2016 European Control Conference (ECC)*, pages 561–566, June 2016. `doi:10.1109/ECC.2016.7810344`.

[BL16b] R. Bobiti and M. Lazar. Sampling-based verification of Lyapunov's inequality for piecewise continuous nonlinear systems. *arXiv e-prints*, September 2016. `arXiv:1609.00302`.

[BMSK16] F. Berkenkamp, R. Moriconi, A. P. Schoellig, and A. Krause. Safe Learning of Regions of Attraction for Uncertain, Nonlinear Systems with Gaussian Processes. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 4661–4666, Dec 2016. `doi:10.1109/CDC.2016.7798979`.

[FCR14] R. Frigola, Y. Chen, and C. E. Rasmussen. Variational Gaussian Process State-Space Models. *arXiv e-prints*, June 2014. `arXiv:1406.4905`.

[Fri76] A. Friedman. *Stochastic Differential Equations and Their Applications.* Academic Press, 1976.

[GRMS02] A. Girard, C. E. Rasmussen, and R. Murray-Smith. Gaussian Process priors with Uncertain Inputs: Multiple-Step-Ahead Prediction. Technical report, University of Glasgow, October 2002.

[IFT17] Y. Ito, K. Fujimoto, and Y. Tadokoro. Second-order Bounds of Gaussian Kernel-based Functions and its Application to Nonlinear Optimal Control with Stability. *arXiv e-prints*, July 2017. `arXiv:1707.06240`.

[Kus67] H.J. Kushner. *Stochastic Stability and Control.* Academic Press, 1967.

[Lya92] A.M. Lyapunov. *The general problem of stability of motion.* PhD thesis, University of Kharkov, 1892.

[LZL13] Y. Li, W. Zhang, and X. Liu. Stability of Nonlinear Stochastic Discrete-Time Systems. *Journal of Applied Mathematics*, 2013. `doi:10.1155/2013/356746`.

[Max68] J.C. Maxwell. On Governors. *Proceedings of the Royal Society of London*, 1868.

[Rou77] E. Routh. *A Treatise on the Stability of a Given State of Motion, Particularly Steady Motion.* Macmillan, 1877.

[Rum99] S. M. Rump. INTLAB - INTerval LABoratory. In *Developments in Reliable Computing*, pages 77–104. 1999. `http://www.ti3.tu-harburg.de/rump/intlab/`.

[Rum10] S. M. Rump. Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica*, pages 287–449, 2010. `doi:10.1017/S096249291000005X`.

[RW06] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning).* The MIT Press, 2006.

[VBNT$^+$16] J. Vinogradska, B. Bischoff, D. Nguyen-Tuong, A. Romer, H. Schmidt, and J. Peters. Stability of Controllers for Gaussian Process Forward Models. In *Proceedings of The 33rd International Conference on Machine Learning*, ICML'16, pages 545–554, Jun 2016.

License